Text Algorithms (6EAP)

Time Warping and sound

Jaak Vilo

2012 fall

Soundex distance metric

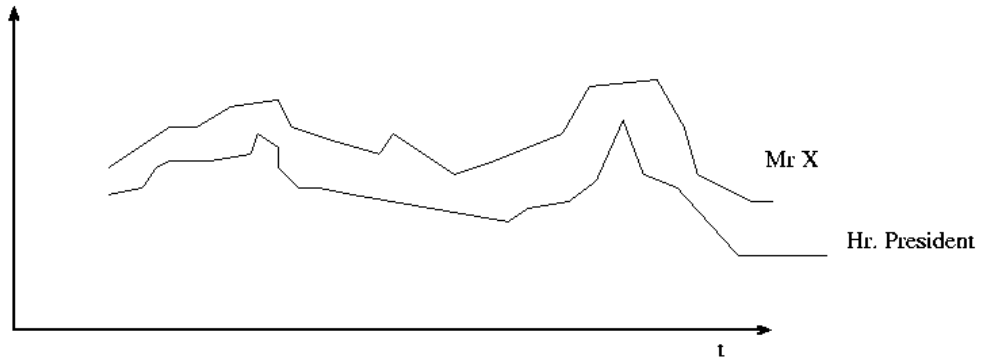
- Soundex is a coarse phonetic indexing scheme, widely used in genealogy. This approach focuses upon individuals names and as such has not been provably applied to a more general context.
- Soundex allows phonetic mispellings to be easily evaluated, for instance the names *John*, *Johne* and *Jon* are often geneologically the same person. This is a term based evaluation where each term is given a Soundex code, each soundex code consists of a letter and three numbers between 0 and 6, e.g. **"Chapman" is "C155".** The letter is always the first letter of the surname. The numbers hash together the rest of the name. This approach is very promising for disabiguation of translitterated/misspelt names, i.e non english names represented as ASCII are frequently misrepresented. The basic algorithm for soundex is now detailed.

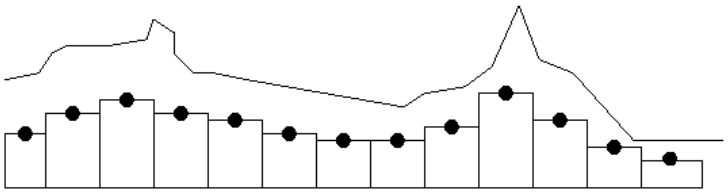
Soundex cnt.

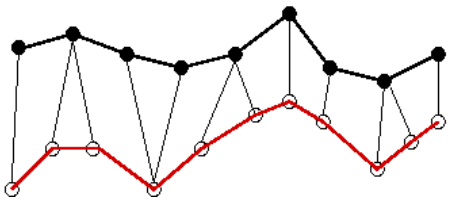
- A soundex code consists of the first letter of the surname, followed by three digits. The digits are based on the consonants as in the following table, (this can differ in implementations):
- 1) B,P,F,V
  2) C,S,K,G,J,Q,X,Z
  3) D,T
  4) L
  5) M,N
  6) R
- The vowels are not used. If two or more adjacent (not separated by a vowel) letters have the same numeric value, only one is used. This also applies if the first (which is used for the letter in the code), and the second letter in name have the same value; the second letter would not be used to generate a digit. If there are not three digits after the consonants are convert, the code is filled out with zeros. The name Sue has no consonants after the S, so the soundex code would be S000.
- This Metric is included in the **SimMetric open source library**.

Time warping

- Previous examples all worked on strings or character sequences
- But if we had a sequence of numercial values, e.g. a time course?
- Are two numerical time courses similar?
- Usage:
- Speech recognition
- Comparison of dynamic processes, e.g. the gene expression changes


Mr X
Hr. President
t





Dynamic Time Warping and Minimum Distance Paths for Speech Recognition

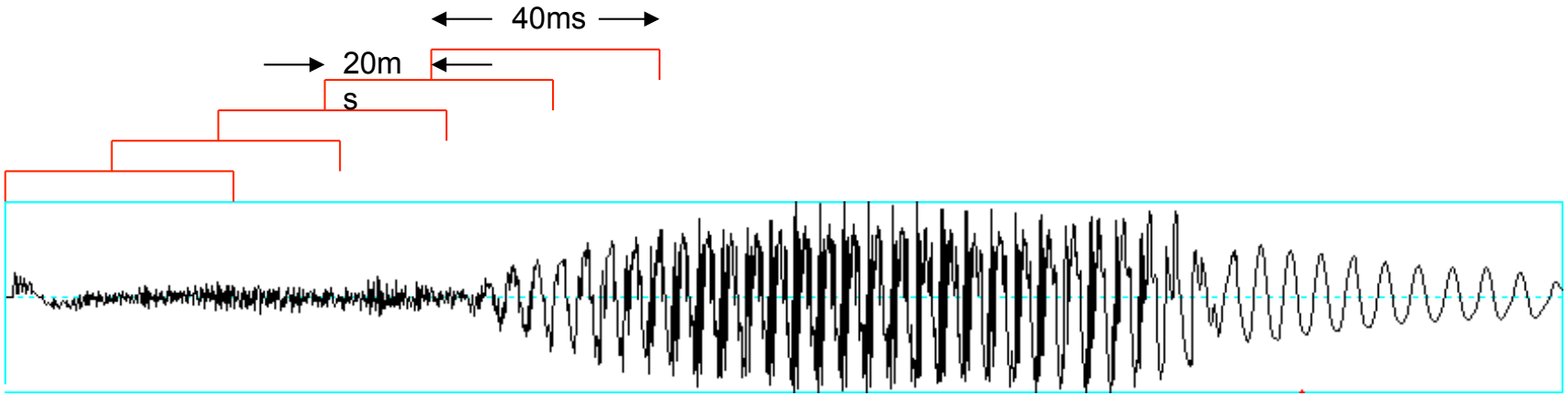
Isolated word recognition:

- Task :
  - Want to build an isolated ‘word’ recogniser e.g. voice dialling on mobile phones
- Method:
  1. Record, parameterise and store vocabulary of reference words
  2. Record test word to be recognised and parameterise
  3. Measure distance between test word and each reference word
  4. Choose reference word ‘closest’ to test word

Words are parameterised on a frame-by-frame basis

Choose frame length, over which speech remains reasonably stationary

Overlap frames e.g. 40ms frames, 10ms frame shift

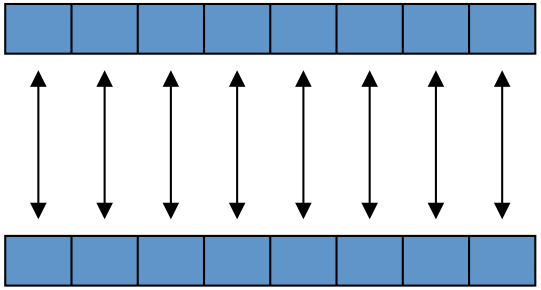


We want to compare frames of test and reference words
i.e. calculate distances between them

Calculating Distances

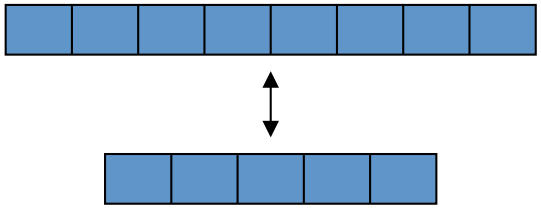
- Easy:

  Sum differences between corresponding frames



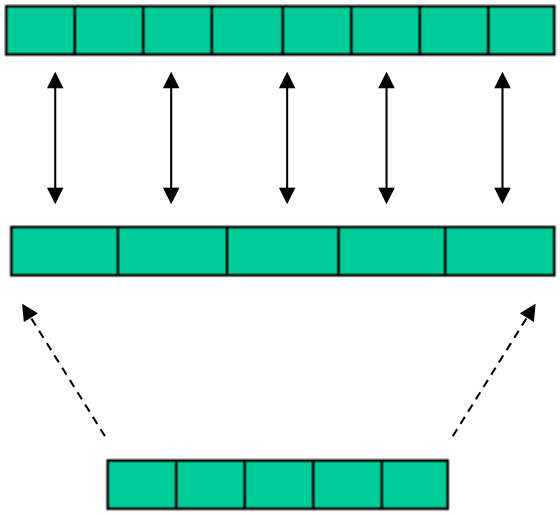
- Problem:

  Number of frames won't always correspond



- Solution 1: Linear Time Warping

  Stretch shorter sound

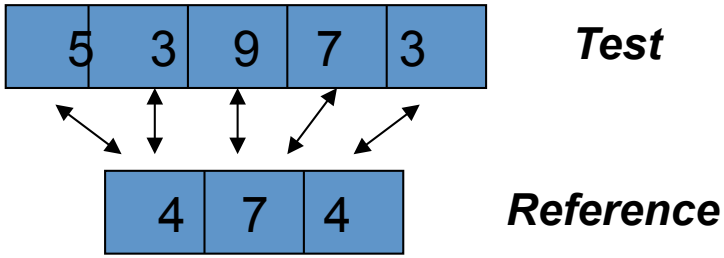


- Problem?

  Some sounds stretch more than others

• Solution 2:

Dynamic Time Warping (DTW)



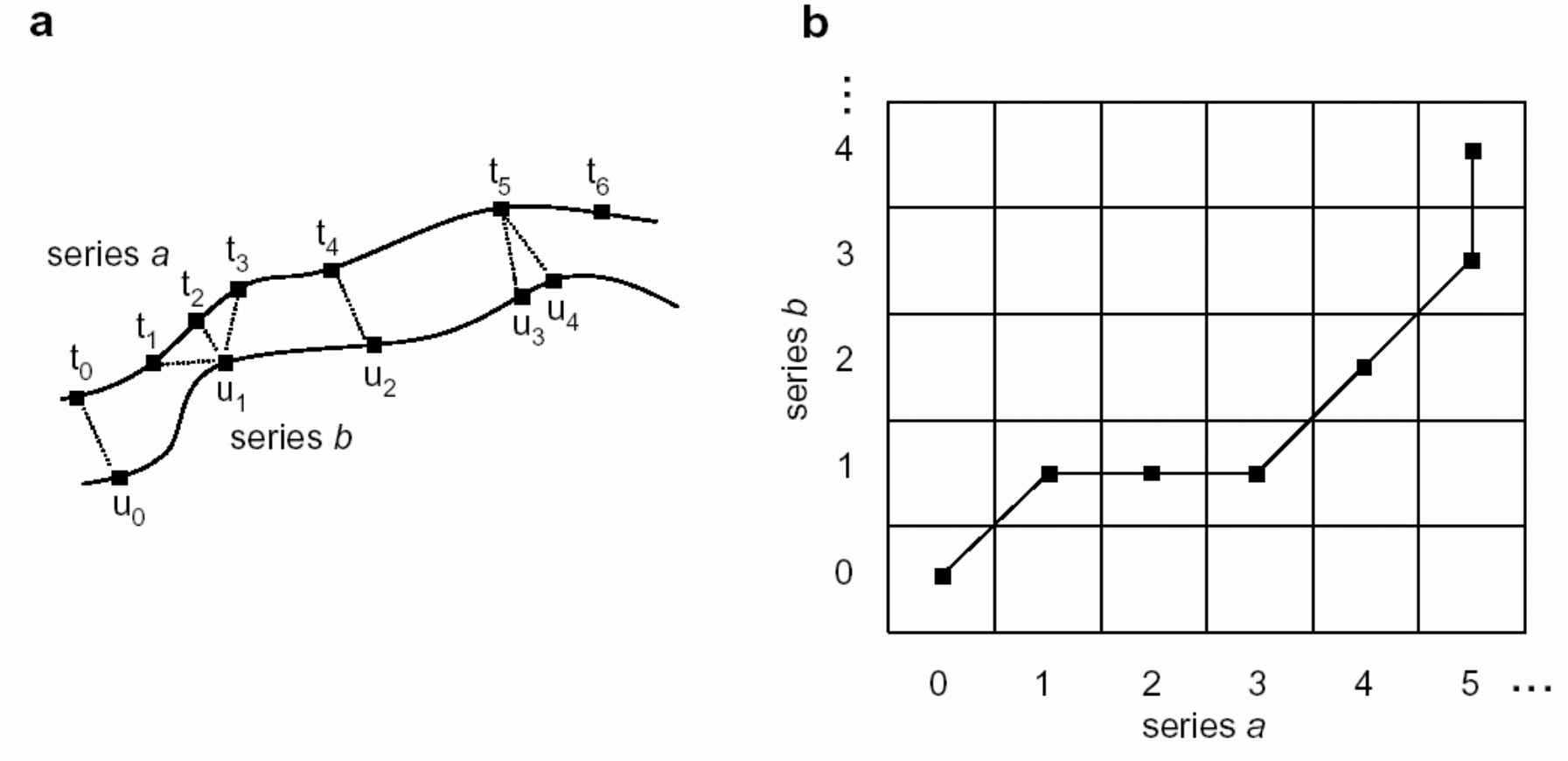
Using a dynamic alignment, make most similar frames correspond

Find distances between two utterences using these corresponding frames

- Discretise the time (e.g. A/D mapping)
- Search for best match between sequences
- Can use the formulas from integral calculus
- substitution cost is replaced by mathematical formulas and interpolations
- Usually just narrowing and widening, no deletion or insertion
- Let be the discrete time unit T

- $T_{ij} = \min \begin{cases} T_{i-1,\,j} + 1/2 \cdot T \cdot w(ai,bj) \\ T_{i-1,\,j-1} + T \cdot w(ai,bj) \\ T_{i,\,j-1} + 1/2 \cdot T \cdot w(ai,bj) \end{cases}$

- **w(ai,bj) could be, for example, |ai - bj|**
- One can define several mappings and costs, these lead to slightly modified recurrence formulas


a
series a
series b
t0
t1
t2
t3
t4
t5
t6
u0
u1
u2
u3
u4
b
series b
series a
0
1
2
3
4
5
...

Demo

```
vilo@muhu:~/TextAlgorithms$ head -15 time_warp.pl
@S1 = qw( 0 5 3 9 7 3 4 3 8 ) ;
@S2 = qw( 0 4 7 4 7 8 9 ) ;
print join "\t" , @S1 , "\n" ; print join "\t" , @S2 , "\n" ;

for ( $j = 0 ; $j < @S1 ; $j++ ) {  $D[0][$j] =  $D[0][$j-1] + abs( $S1[$j] - $S2[0] )  }
for ( $i = 0 ; $i < @S2 ; $i++ ) {  $D[$i][0] =  $D[$i-1][0] + abs( $S2[$i] - $S1[0] )  }

for ( $i = 1 ; $i < @S2 ; $i++ ) {
  for ( $j = 1 ; $j < @S1 ; $j++ ) {
    $D[$i][$j] = minval(
                $D[$i-1][$j-1] + abs( $S2[$i] - $S1[$j] ) ,
                $D[$i-1][$j] +   abs( $S2[$i] - $S1[$j] ) / 2 ,
                $D[$i][$j-1] +   abs( $S2[$i] - $S1[$j] ) / 2 ,
      ) ;
 }
vilo@muhu:~/TextAlgorithms$
```

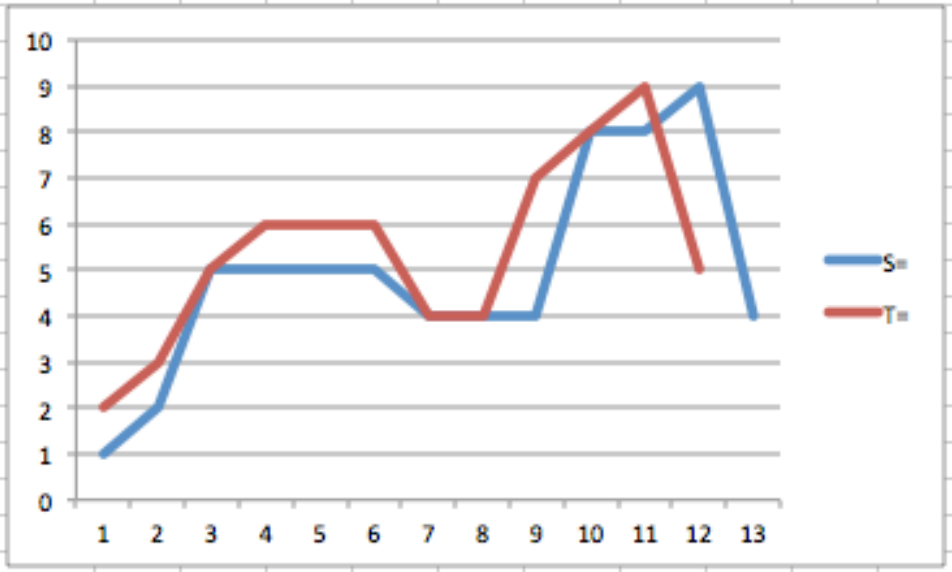
```
vilo@muhu:~/TextAlgorithms$ perl time_warp.pl
0	5	3	9	7	3	4	3	8
0	4	7	4	7	8	9
D( 0,5,3,9,7,3,4,3,8 -:- 0,4,7,4,7,8,9 ) = 6.5
	0	5	3	9	7	3	4	3	8
0	0	5	8	17	24	27	31	34	42
4	4	1	1.5	4	5.5	6	6	6.5	8.5
7	11	2	3.5	3.5	3.5	5.5	7	8.5	7.5
4	15	2.5	3	5.5	5	4.5	4.5	5	7
7	22	3.5	5	5	5	6.5	6	7	6
8	30	5	7.5	5.5	5.5	8	8	9.5	6
9	39	7	10	5.5	6.5	9.5	10.5	12.5	6.5
vilo@muhu:~/TextAlgorithms$
```

	0	5	3	9	7	3	4	3	8
0	0	5	8	17	24	27	31	34	42
4	4	1	1.5	4	5.5	6	6	6.5	8.5
7	11	2	3.5	3.5	3.5	5.5	7	8.5	7.5
4	15	2.5	3	5.5	5	4.5	4.5	5	7
7	22	3.5	5	5	5	6.5	6	7	6
8	30	5	7.5	5.5	5.5	8	8	9.5	6
9	39	7	10	5.5	6.5	9.5	10.5	12.5	6.5

MIN

=MIN(N10+ABS(O$5-$A11), O10+ABS(O$5-$A11)*0.5, N11+ABS(O$5-$A11)*0.5)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	S=	1	2	5	5	5	5	4	4	4	8	8	9	4							
2	T=	2	3	5	6	6	6	4	4	7	8	9	5								
3																					
4																					
5			1	2	5	5	5	5	4	4	4	8	8	9	4						
6		0	0.5	1.5	4	6.5	9	11.5	13.5	15.5	17.5	21.5	25.5	30	32						
7	2	1	1	0.5	2	3.5	5	6.5	7.5	8.5	9.5	12.5	15.5	19	20						
8	3	2.5	2	1	2	3	4	5	5.5	6	6.5	9	11.5	14.5	15						
9	5	5	4	2.5	1	1	1	1	1.5	2	2.5	4	5.5	7.5	8						
10	6	8	6.5	4.5	1.5	1.5	1.5	1.5	2.5	3	3.5	4.5	5.5	7	8						
11	6	11	9	6.5	2	2	2	2	3	4	4.5	5.5	6.5	8	=MIN(N10+ABS(O$5-$A11), O10+ABS(O$5-$A11)*0.5, N						
12	6	14	11.5	8.5	2.5	2.5	2.5	2.5	3.5	4.5	5.5	6.5	7.5	9	10						
13	4	16	13	9.5	3	3	3	3	2.5	2.5	2.5	4.5	6.5	9	9						
14	4	18	14.5	10.5	3.5	3.5	3.5	3.5	2.5	2.5	2.5	4.5	6.5	9	9						
15	7	21.5	17.5	13	4.5	4.5	4.5	4.5	4	4	4	3.5	4	5	6.5						
16	8	25.5	21	16	6	6	6	6	6	6	6	3.5	3.5	4	6						
17	9	30	25	19.5	8	8	8	8	8.5	8.5	8.5	4	4	3.5	6						
18	5	32.5	27	21	8	8	8	8	8.5	9	9	5.5	5.5	5.5	4.5						



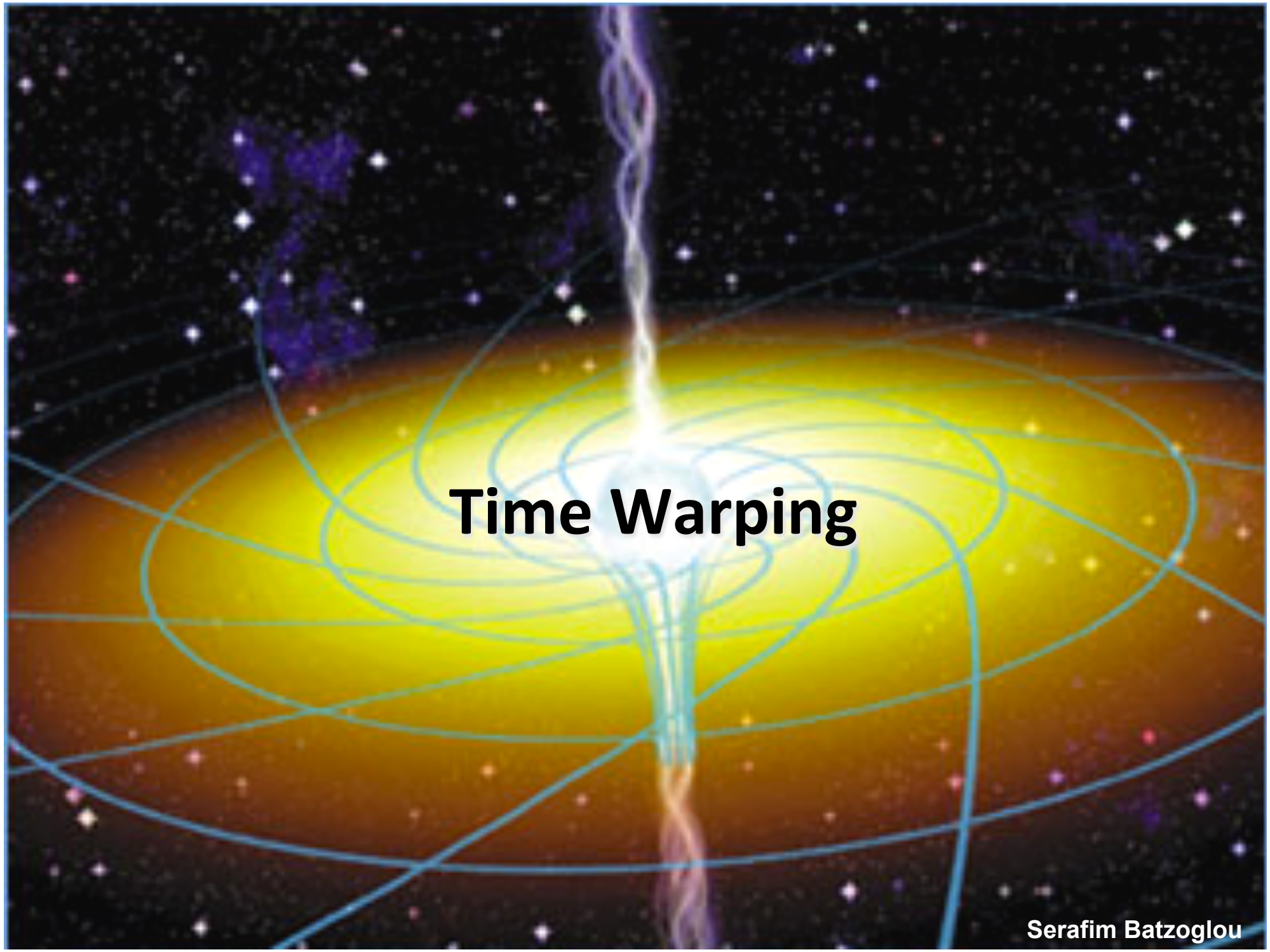
- Aach J, Church GM. Aligning gene expression time series with time warping algorithms. Bioinformatics. 2001 Jun;17(6):495-508. PubMed:11395426
- **Time Warps, String Edits, and Macromolecules**: The Theory and Practice of Sequence Comparison (The David Hume Series) David Sankoff (Editor), Joseph Kruskal (Editor). Paperback - 407 pages (28 January, 2000) Cambridge University Press; ISBN: 1575862174Reviews: Book Description
- Time Warps, String Edits and Macromolecules is a young classic in computational science, scientific analysis from a computational perspective. The computational perspective is that of sequence processing, in particular the problem of recognizing related sequences. The book is the first, and still best compilation of papers explaining how to measure distance between sequences, and how to compute that measure effectively. This is called string distance, Levenshtein distance, or edit distance. The book contains lucid explanations of the basic techniques; well-annotated examples of applications; mathematical analysis of its computational (algorithmic) complexity; and extensive discussion of the variants needed for weighted measures, timed sequences (songs), applications to continuous data, comparison of multiple sequences and extensions to tree-structures. In molecular biology the sequences compared are the macromolecules DNA and RNA. Sequence distance allows the recognition of homologies (correspondences) between related molecules. One may interpret the distance between molecular sequences in terms of the mutations necessary for one molecule to evolve into another. A further application explores methods of predicting the secondary structure (chemical bonding) of RNA sequences. In speech recognition speech input must be compared to stored patterns to find the most likely interpretation (e.g., syllable). Because speech varies in tempo, part of the comparison allows for temporal variation, and is known as "time-warping". In dialectology Levenshtein distance allows analysis of the learned variation in pronunication, its cultural component. Levenshtein distance introduces a metric which allows more sophisticated analysis than traditional dialectology's focus on classes of alternative pronunciations. A similar application is the study of bird song, where degrees of distance in song are seen to correspond to the divergence of bird populations. A final application area is software, where Levenshtein distance is employed to located differing parts of different versions of computer files, and to perform error correction.

Following is borrowed from

- **Serafim Batzoglou,**
  Associate Professor

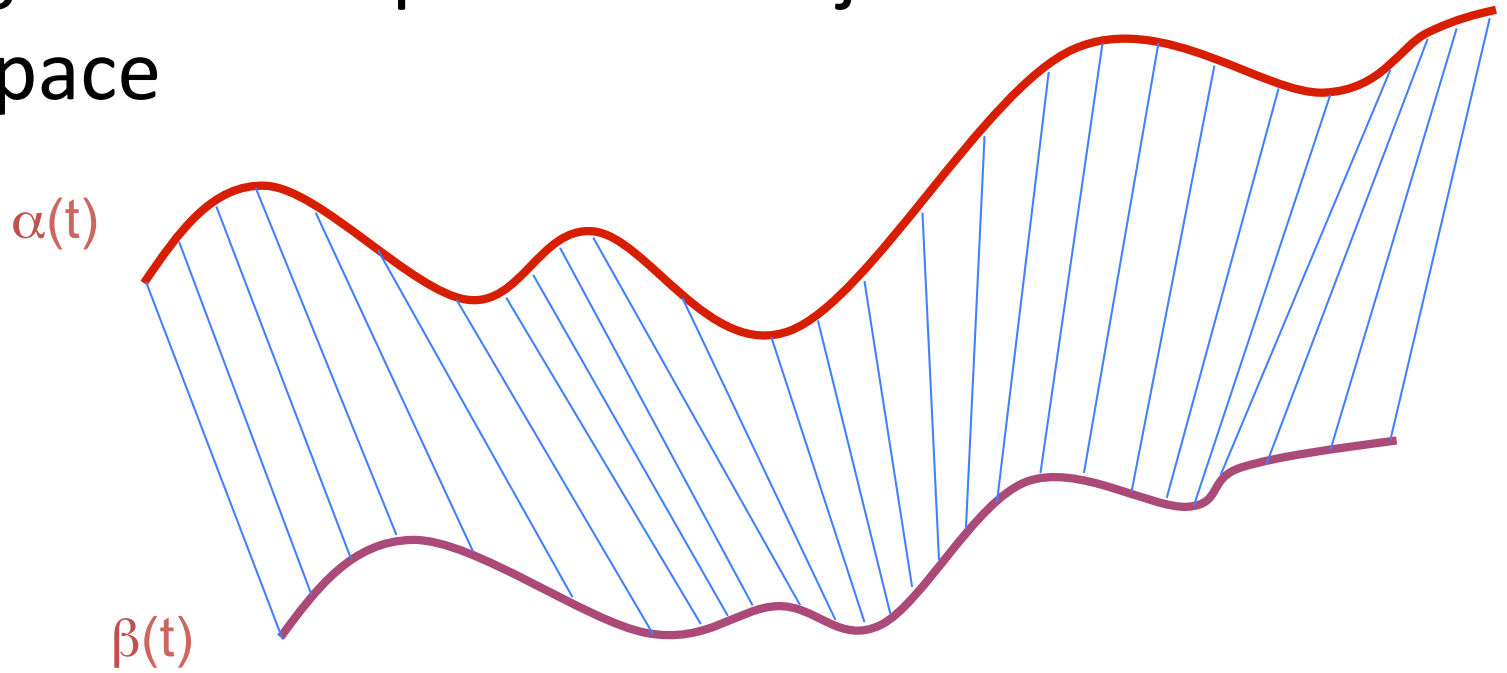
- http://ai.stanford.edu/~serafim/

- *http://ai.stanford.edu/~serafim/cs262/Spring2003/Slides/Lecture4.ppt*


Time Warping
Serafim Batzoglou

Time Warping

Align and compare two trajectories in multi-D space

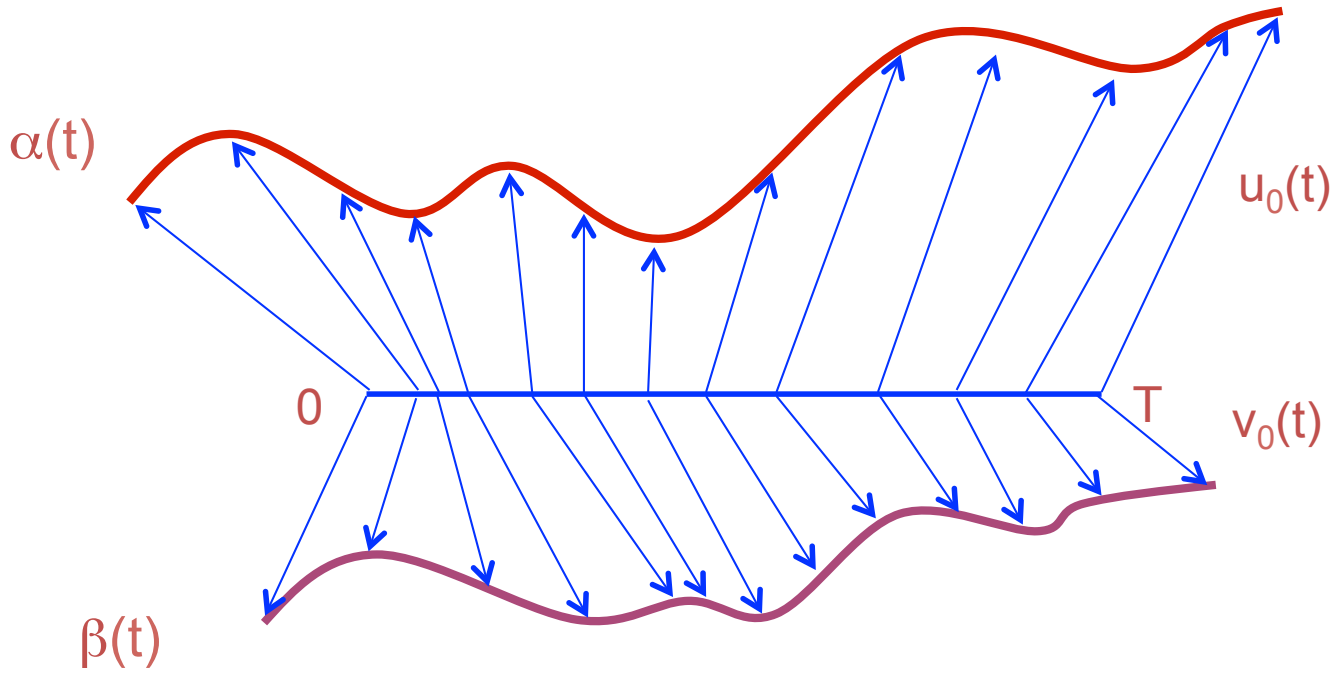


- Additive random error
- Variations in speed from one segment to another

Time Warping

Definition: $\alpha(u)$, $\beta(u)$ are connected by an approximate continuous time warping (u_0, v_0), if:

u_0, v_0 are strictly increasing functions on [0, T], and
$\alpha(u_0(t)) \cong \beta(v_0(t))$ for $0 \le t \le T$



Time Warping

How do we measure how "good" a time warping is?

Let's try:

$$\int_0^T w(\alpha(u_0(t)), \beta(v_0(t)))\, dt$$

However, an equivalent time warping ($u_1(s)$, $v_1(s)$), is given by:

$s = f(t);$ $\qquad\qquad$ $f: [0, T] \rightarrow [0, S]$

has score

$$\int_0^S w(\alpha(u_1(s)), \beta(v_1(s)))\, ds = \int_0^T w(\alpha(u_0(t)), \beta(v_0(t)))\, f'(t)\, dt$$

This is arbitrarily different

Time Warping

This one works:

$d(u_0, v_0) = \int_0^T w(\alpha(u_0(t)), \beta(v_0(t))) [(u'_0(t) + v'_0(t))/2] dt$

Now, if $s = f(t)$; $t = g(s)$, and $g = f^{-1}$,

$\int_0^S w(\alpha(u_1(s)), \beta(v_1(s))) (u'_1(s) + v'_1(s))/2 \, ds =$

$f(t) = f(g(s)) = s;$
$f'(t) = f'(g(s)) \, g'(s) = 1$, therefore $g'(s) = 1/f'(t)$
$u_0(t) = u_0(g(s))$, therefore $u'_0(t) = u'_0(g(s)) \, g'(s)$

$\int_0^T w(\alpha(u_0(t)), \beta(v_0(t))) (u'_0(t)+v'_0(t))/2 \; g'(s) \, f'(t) \, dt =$
$\int_0^T w(\alpha(u_0(t)), \beta(v_0(t))) [(u'_0(t) + v'_0(t))/2] dt$

Time Warping

From continuous to discrete:

Let's discretize the signals:

$\alpha(t)$: $a = a_0 \ldots\ldots a_M$

$\beta(t)$: $b = b_0 \ldots\ldots b_N$

Definition:

a, **b** are connected by an approximate discrete time warping (u, v), if u and v are weakly increasing integer functions on $1 \le h \le H$, such that

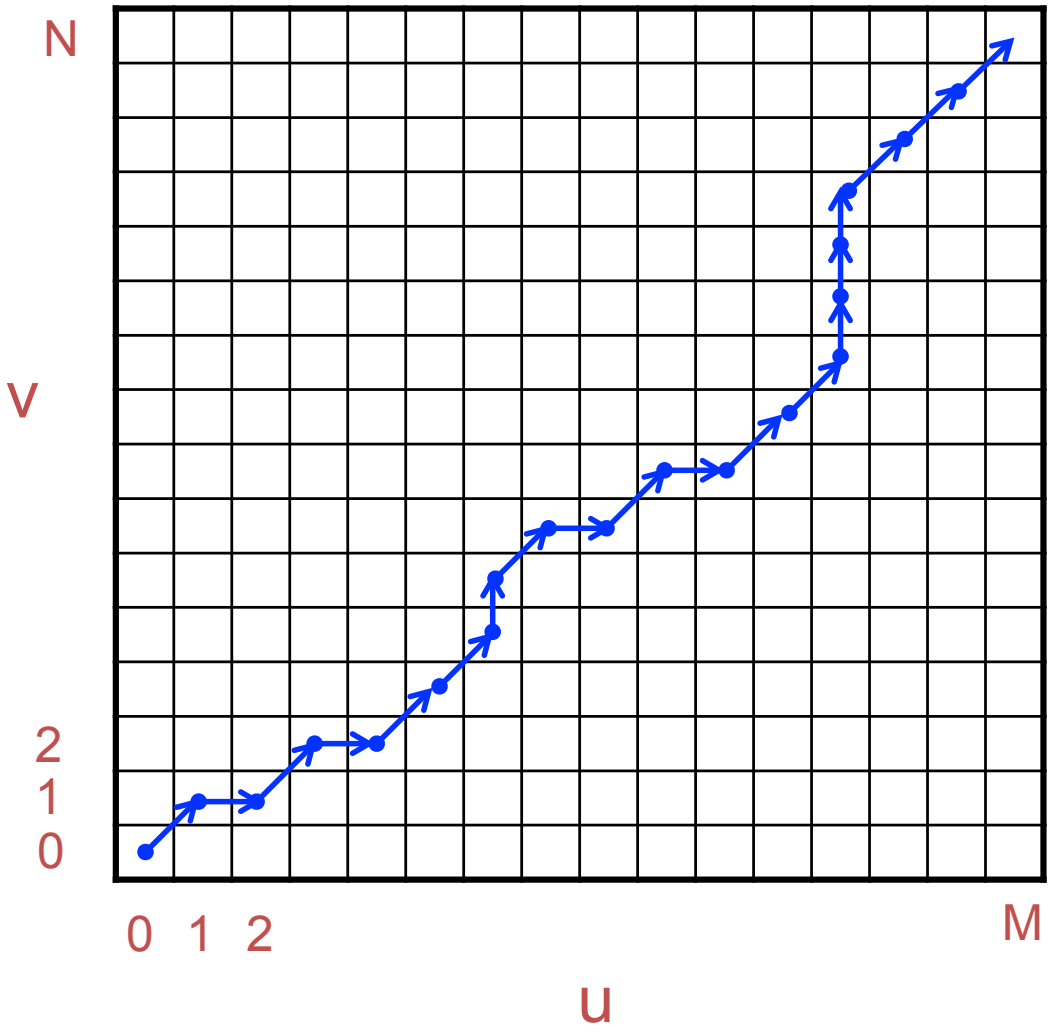
$$a_{u[h]} \cong b_{v[h]} \text{ for all } h = 1 \ldots\ldots H$$

Moreover, we require $u[0] = v[0] = 0$;

$u[H] = M$;

$v[h] = N$

Time Warping



Define possible steps:

$(\Delta u, \Delta v)$ is the possible difference of u and v

between steps h-1 and h

$$(\Delta u, \Delta v) = \begin{cases} (1, 0) \\ (1, 1) \\ (0, 1) \end{cases}$$

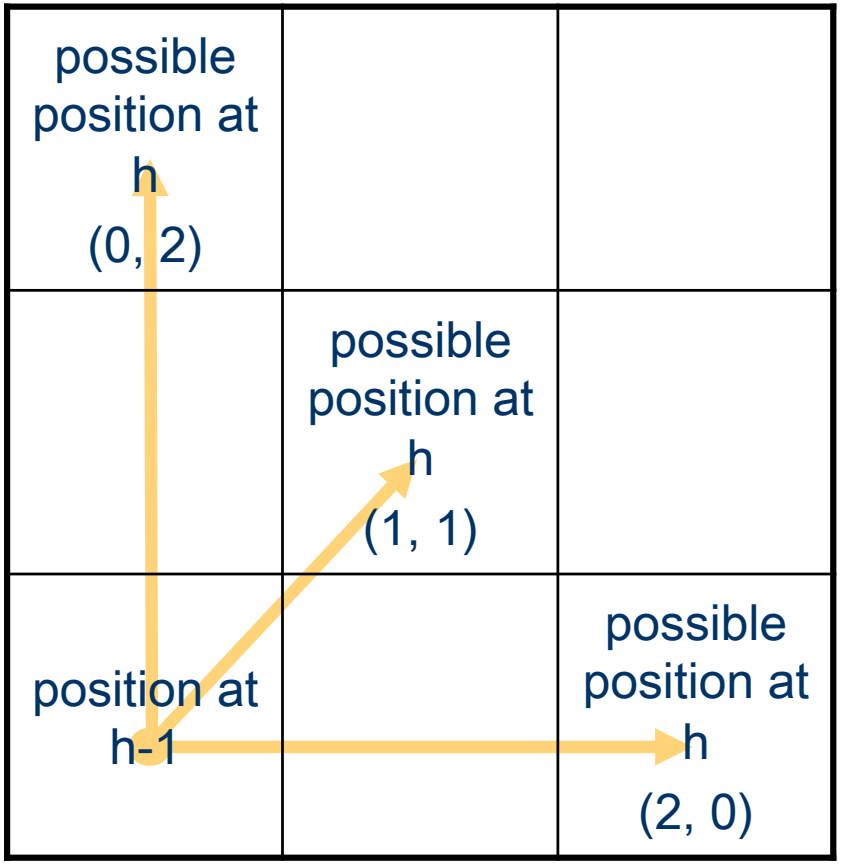
Time Warping

Alternatively:

$$(\Delta u, \Delta v) = \begin{cases} (2, 0) \\ (1, 1) \\ (0, 2) \end{cases}$$

Advantage:

Every time warp has the same number of steps



Time Warping

Discrete objective function:

For $0 \leq i = u[h] \leq M$; $0 \leq j = v[h] \leq N$,
Define $w(i, j) = w(a_{u[h]}, b_{v[h]})$

Then,

$D(u, v) = \Sigma_h \, w(u[h], v[h]) \, (\Delta u + \Delta v)/2$

In the case where we allow (2, 0), (1, 1), and (0, 2) steps,

$D(u, v) = \Sigma_h \, w(u[h], v[h])$

Time Warping

Algorithm for optimal discrete time warping:

Initialization:

$D(i, 0) = ½ \Sigma_{i'<i} w(i, 0)$

$D(0, j) = ½ \Sigma_{j'<j} w(0, j)$

$D(1, j) = D(i, 1) = w(i, j) + w(i-1, j-1)$

Iteration:

For i = 2……M

For j = 2……N

$$D(i, j) = \min \begin{cases} D(i-2, j) + w(i, j) \\ D(i-1, j-1) + w(i, j) \\ D(i-2, j) + w(i, j) \end{cases}$$

```
int DTWDistance(char s[1..n], char t[1..m]) {
    declare int DTW[0..n, 0..m]
    declare int i, j, cost

    for i := 1 to m
        DTW[0, i] := infinity
    for i := 1 to n
        DTW[i, 0] := infinity
    DTW[0, 0] := 0

    for i := 1 to n
        for j := 1 to m
            cost:= d(s[i], t[j])
            DTW[i, j] := cost + minimum(DTW[i-1, j  ],    // insertion
                                        DTW[i  , j-1],    // deletion
                                        DTW[i-1, j-1])    // match

    return DTW[n, m]
}
```

```
int DTWDistance(char s[1..n], char t[1..m], int w) {
    declare int DTW[0..n, 0..m]
    declare int i, j, cost

    w := max(w, abs(n-m)) // adapt window size (*)

    for i := 0 to n
        for j:= 0 to m
            DTW[i, j] := infinity
    DTW[0, 0] := 0

    for i := 1 to n
        for j := max(1, i-w) to min(m, i+w)
            cost := d(s[i], t[j])
            DTW[i, j] := cost + minimum(DTW[i-1, j  ],    // insertion
                                        DTW[i, j-1],    // deletion
                                        DTW[i-1, j-1])    // match

    return DTW[n, m]
}
```