Počítačové zpracování řeči

Přednáška 4

Algoritmus DTW

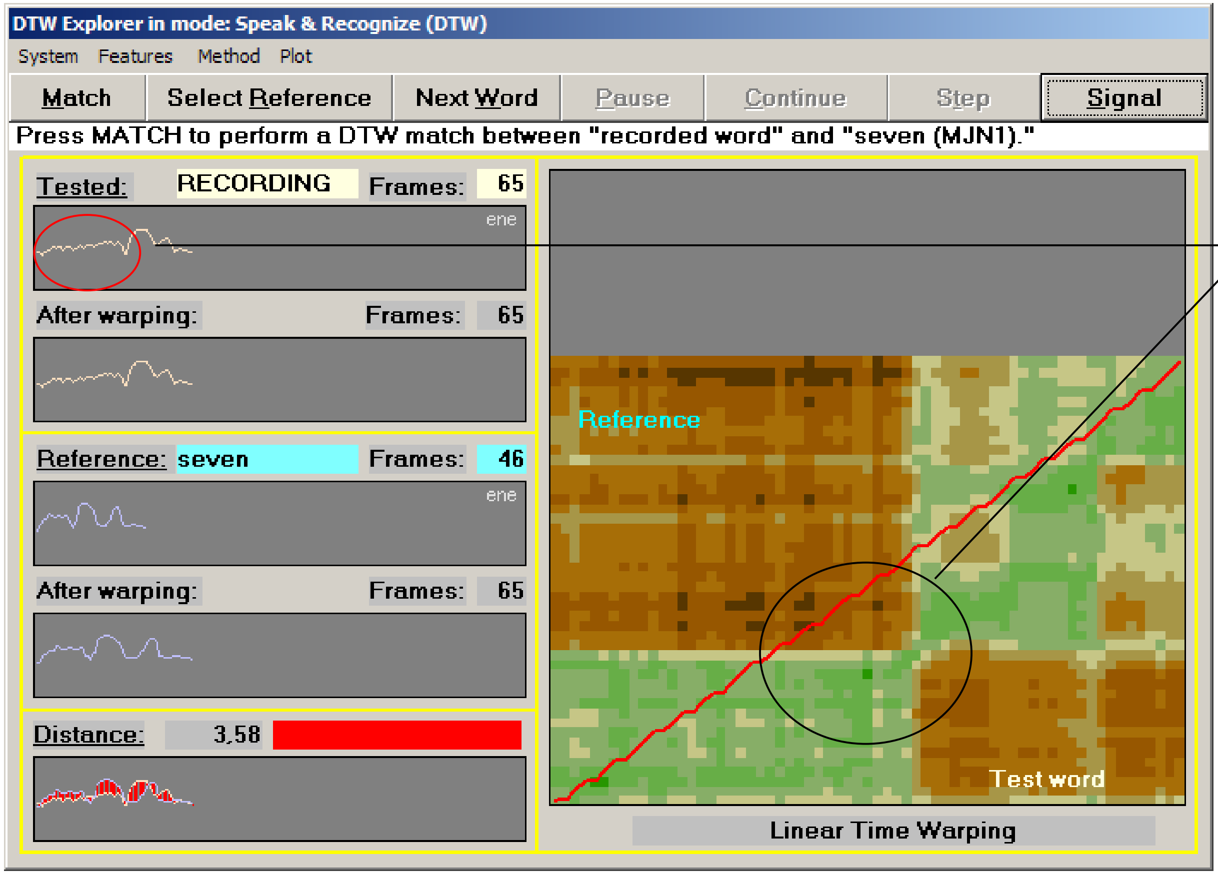
Z předchozí přednášky a úlohy

1. **Metoda LTW pomohla při vyřešení základního problému měření podobnosti mezi různě dlouhými příznakovými sekvencemi.**
2. **Umožňuje tím aplikovat klasickou metodu hledání nejbližšího souseda (reference) metodou nejmenší vzdálenosti.**
3. **Za ideálních podmínek (malý slovník, výrazně se lišící slova, testovaná slova i reference od stejné osoby a nahraná za stejných podmínek, …) metoda dává slušné výsledky i s jednoduchými příznaky.**
4. **Za běžných podmínek budou výsledky ale výrazně horší.**
5. **Metoda LTW <u>řeší pouze problém různých délek</u> dvou nahrávek bez ohledu na to, co je v nich obsaženo.**

Proč občas LTW selhává (1)

1. Stejná slova jsou často vyslovována s různým “rytmem”.
– např. 1. slabika slova je kratší než u reference, 2. slabika naopak delší

Ilustrace: testované slovo “seven” má 1. slabiku delší než reference téhož slova

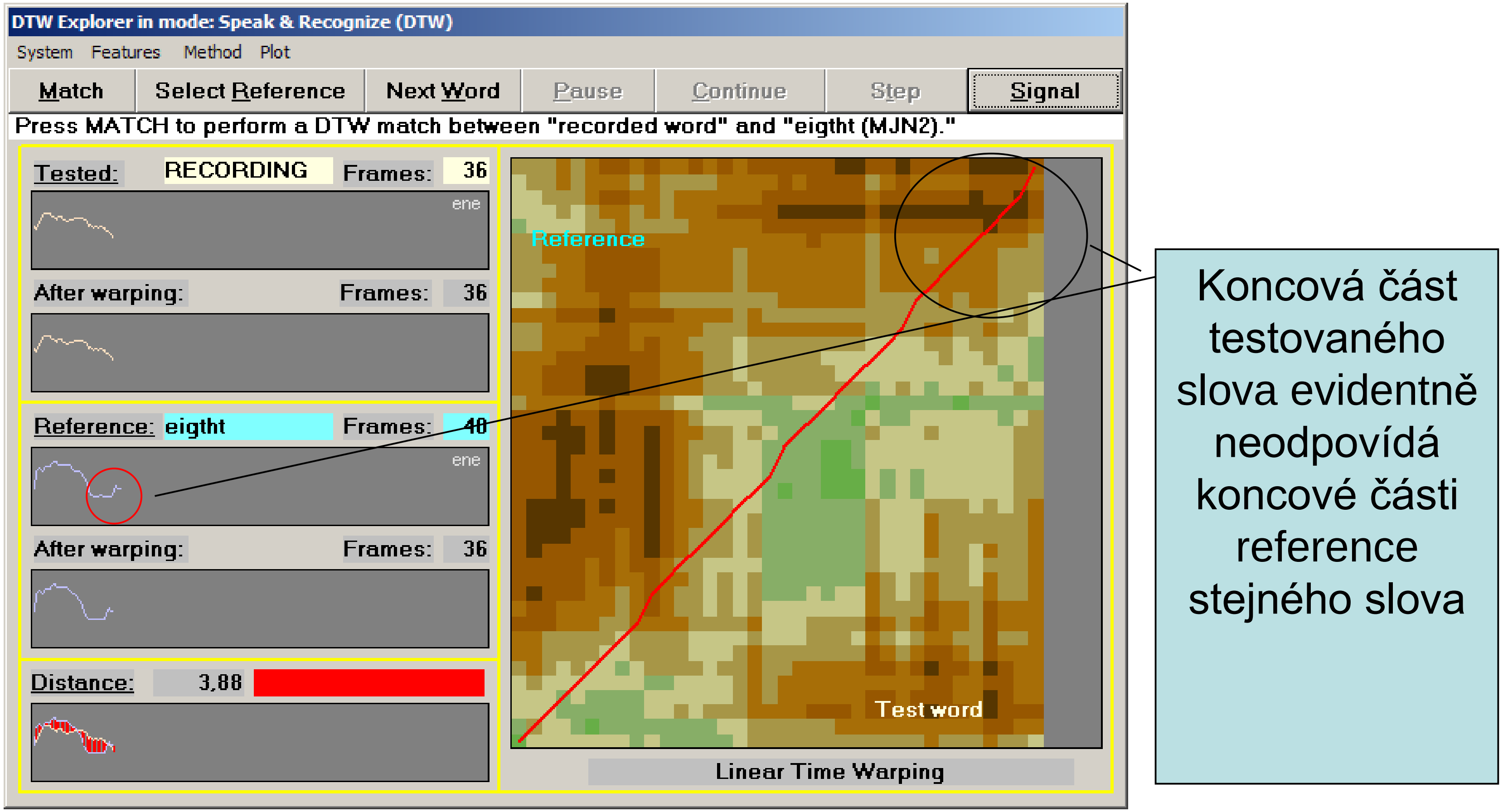


Linearní transformace zde evidentně není optimální – přiřazuje k sobě nepatřičné framy (cesta vede „přes hory“, i když jde o stejná slova)

Proč občas LTW selhává (2)

2. Detektor začátku a konce řeči nenajde přesné hranice slova

Ilustrace: v testovaném slovu “eight” detektor opomněl koncové „t“



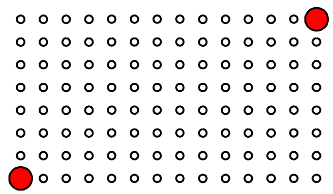
Nelineární transformační funkce

Idea: Místo lineární funkce zkusme nelineární časovou transformaci - „dynamické borcení času“ (angl. Dynamic Time Warping - DTW)

Nelineárních funkcí existuje mnoho. Musíme vybrat takové, které odpovídají naší úloze, tj. měření podobnosti dvou slov. Měly by splňovat následující **požadavky**:

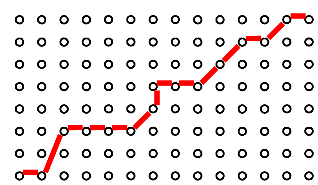
1. **Okrajové podmínky**

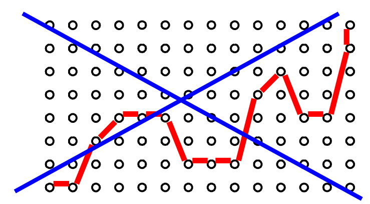
   $w(1) = 1,\ w(I) = J$



2. **Monotonicita (neklesající průběh)**

   tj. nesmíme obrátit tok času





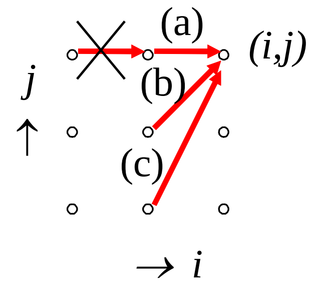
3. **Podmínky spojitosti**

   Itakurovy podmínky:

$$w(i) = j \Leftrightarrow w(i-1) = j \wedge w(i-2) \neq j \quad \text{or} \quad \text{(a)}$$

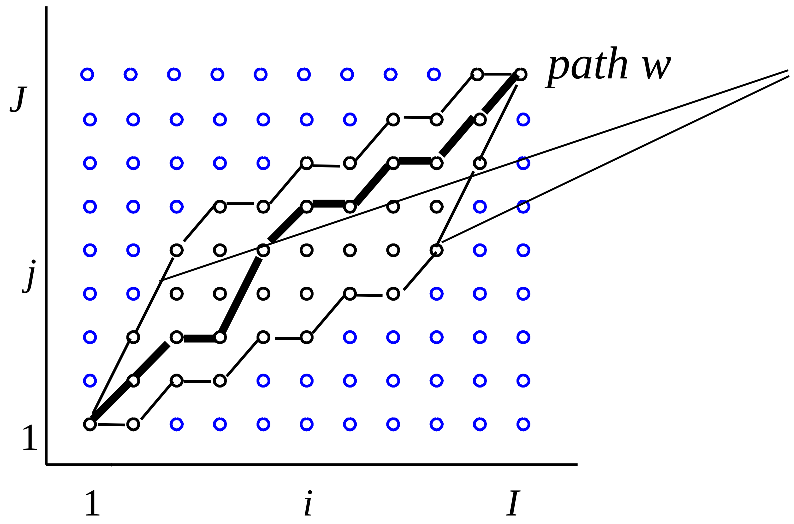
$$w(i-1) = j-1 \quad \text{or} \quad \text{(b)}$$

$$w(i-1) = j-2 \quad \text{(c)}$$



Která z nelineárních funkcí je ta nejlepší?

Předchozí požadavky splňuje velké množství nelineárních funkcí – kterou vybrat?



Tyto dvě funkce („cesty“) jsou ty nejkrajnější.
Žádná cesta odpovídající Itakurovým podmínkám nemůže procházet body mimo prostor, který vymezují.
Vyznačují tzv. **globální omezení.**

Nejlepší funkce (cesta) je ta, která vede na **minimální globální vzdálenost**.

$$DTW: \qquad D(\mathbf{X},\mathbf{R}) = \underset{w}{Min} \sum_{i=1}^{I} d(x_i, r_{w(i)})$$

Všimněme si zásadního rozdílu mezi LTW and DTW:
U LTW cesta závisí ***pouze na hodnotách I a J***,
u DTW cesta závisí ***též na příznakových vektorech a na jejich vzdálenostech***.

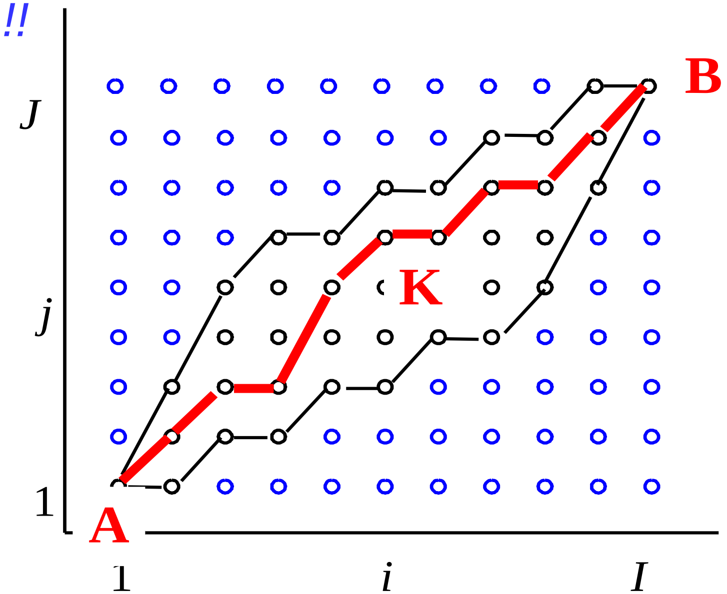
Dynamické Programování – efektivní výpočet DTW

Vzdálenost určenou pomocí DTW metody můžeme vypočítat velice efektivně s použitím **Bellmanova principu optimality**, který je základem optimalizačních strategií nazývaných **Dynamické Programování.**

Z pohledu naší úlohy můžeme Princip optimality vyjádřit takto:

Jestliže optimální cesta jdoucí z bodu A do B prochází bodem K, pak také cesta z A do K je optimální.

Pozor: Obráceně to neplatí. Je-li cesta z A do K optimální, neznamená to, že bod K leží na optimální cestě z A do B !!



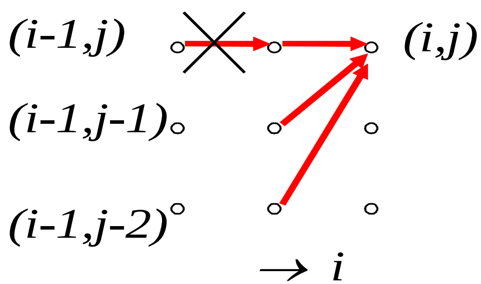
Tato strategie nám umožňuje hledat optimální cestu rekurzivně / po krocích.

DTW algoritmus (1)

Definujme: Akumulovanou vzdálenost v bodě (i, j):

$$A(i, j) = d(x_i, r_j) + Min[A(i-1, j)*q, A(i-1, j-1), A(i-1, j-2)]$$

Je složena ze součtu **lokální vzdálenosti v bodě (i, j)** a **nejmenší z akumulovaných vzdáleností** ve 3 možných předchozích bodech. Pomocný faktor q zohledňuje Itakurovu podmínku (3a) a nabývá pouze hodnot 1 nebo ∞.



Poznámka: V dalším výkladu budeme rozlišovat 3 typy vzdáleností:

- **Lokální vzdálenost *d (x, r)* –** vzdálenost mezi dvěma framovými vektory
- **Akumulovaná vzdálenost *A(i, j)* -** vzdálenost akumulovaná do bodu (i,j)
- **Globální vzdálenost *D (X, R)*** – vzdálenost mezi dvěma slovy

DTW algoritmus (2)

DTW algorithm (for Itakura's constraints)	
Step 1:	*Initialisation*
	$A(1,1) = d(x_1, r_1)$ $A(1,j) = \infty$ pro j=2,…J $B(1,1) = 0$
Step 2:	*Recursion*
	For $i = 2, \ldots. I$
	For $j = 1, \ldots. J$
	$O(k) = A(i-1,k)$ for $k = j, j-1, j-2$ *(temporary variable)*
	if $B(i-1,j) = j$ then $O(j) = \infty$
	$A(i,j) = d(x_i, r_j) + \underset{k}{Min}[O(k)]$ $B(i,j) = \underset{k}{ArgMin}[O(k)]$
Step 3:	*Termination*
	$D(\mathbf{X},\mathbf{Y}) = A(I,J)$
Step 4:	*Backtracking*
	$w(I) = J$ for $\mathrm{i} = I-1, \ldots .1$ $w(i) = B(i+1, w(i+1))$

Komentář:

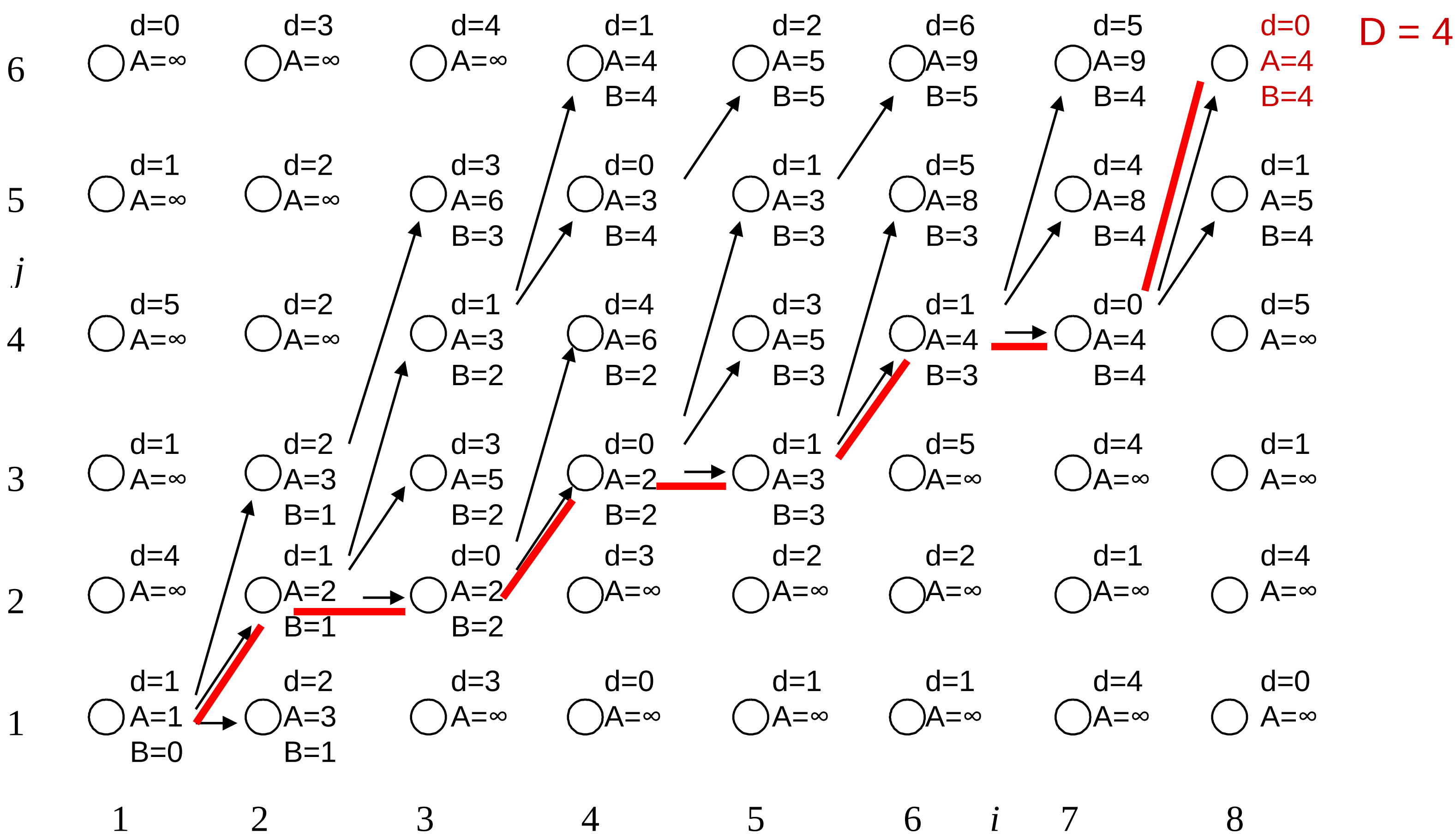
- *A(i, j)* je pole *akumulovaných vzdáleností*
- *B(i, j)* je pole zpětných ukazatelů, pro každý bod*(i,j)* ukládá *j*-tý index nejlepšího předchůdce
- *O(k)* je dočasné pole použité zde pouze z *důvodů snazšího vysvětlení* (při implementaci není nezbytné)
- Hledaná vzdálenost slov *D (X, R)* je akumulovaná vzdálenost v bodě *(I,J)*
- *Optimální cesta* může být identifikována (pokud ji potřebujeme znát) **teprve tehdy** až dosáhneme bod *(I,J)* – cesta se pak identifikuje od konce na začátek procesem zvaným *backtracking*.

DTW algoritmus (3)

Příklad rekurzivního výpočtu (animovaná prezentace)

Uvažujme P = 1 a sekvence **x = (1, 4, 5, 2, 3, 7, 6, 1)** I = 8

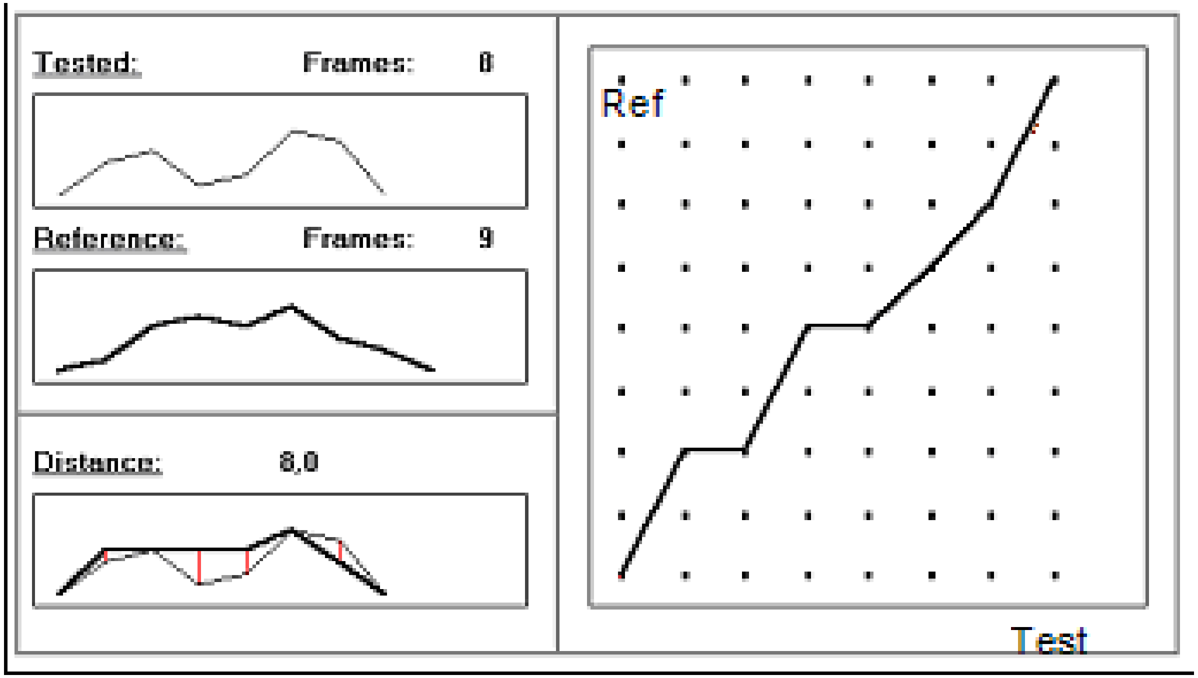
r = (2, 5, 2, 6, 2, 1) J = 6



DTW algoritmus (4)

Jiný příklad (tento i předchozí příklad mohou sloužit jako kontrola správné implementace)

x = (1, 4, 5, 2, 3, 7, 6, 1) I = 8
r = (1, 2, 5, 6, 5, 7, 4, 3, 1) J = 9



Poznámka: Oba uvedené příklady byly též řešeny v přednášce o LTW. Můžete si tedy porovnat výsledky obou algoritmů.

DTW algoritmus (5)

Co se změní, když budeme pracovat s více příznaky? Pouze výpočet lokál. vzdálenosti

Např. P=3 **x = ([1, 3, 2], [4, 0, 2], [5, 2,-1] ….)** I = 8

r = ([1, 4, 1], [2, 3, 1],[5, 4, 2], ….) J = 9

Euklid. lokál. vzdálenost v bodě (1,1)

$$d(x_1, r_1) = \sqrt{\sum_{p=1}^{P} (x_{1p} - r_{1p})^2} = \sqrt{(1-1)^2 + (3-4)^2 + (2-1)^2}$$

Euklid. lokál. vzdálenost v bodě (3,2)

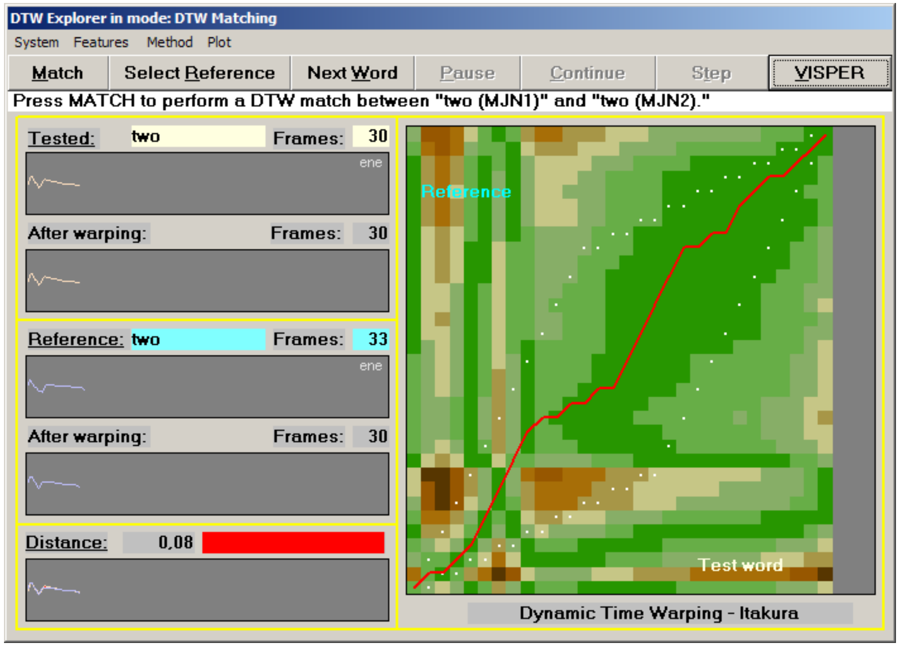
$$d(x_3, r_2) = \sqrt{\sum_{p=1}^{P} (x_{3p} - r_{2p})^2} = \sqrt{(5-2)^2 + (2-3)^2 + (-1-1)^2}$$

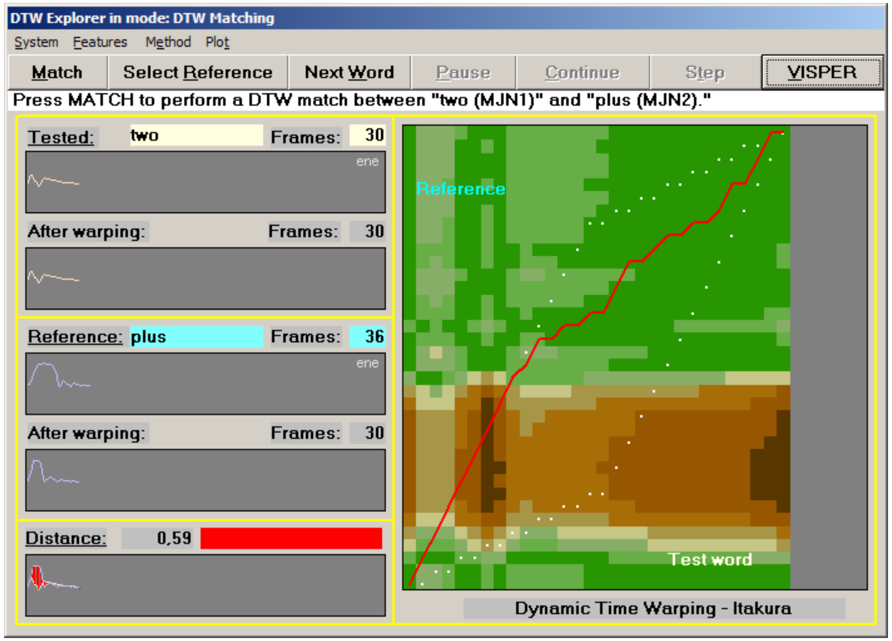
DTW v prostředí VISPER

Příklady skutečných slov – ilustrace pomocí “barevné mapy”

Slovo “two” (representované jediným příznakem - energií) porovnáváno s

referencí “two” referencí “plus”





Úloha pro cvičení

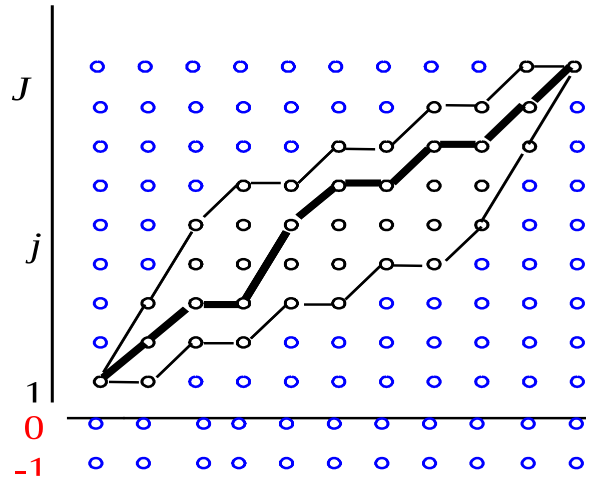
Rozpoznávač číslic založený na DTW

1) Využijte vše, co jste vytvořili pro rozpoznávač založený na LTW
2) Naprogramujte modul DTW a ověřte jeho činnost na jednoduchých datech uvedených v přednášce (a na elearningu)
3) Ověřte si, že modul správně funguje i pro více příznaků
4) Fungující modul se snažte alespoň trochu optimalizovat (zejména s ohledem na rychlost), neboť DTW je významně časově náročnější.
5) Hotový modul DTW vložte do rozpoznávače na místo LTW
6) Ověřte jej na vašich i dodaných datech
7) Proveďte rozpoznávací testy se všemi sadami příznaků, které jste použili minule (ene, ene+zcr, spe16)
8) Porovnejte skóre dosažené metodami LTW a DTW
9) Implementaci a tabulku výsledků pro LTW a DTW pošlete opět do pondělí (stačí vždy prům. za všechny osoby a typ příznaků)

Tipy pro implementaci (1)

Implementace DTW algoritmu

- **Reprezentace neznámého slova** musí být vždy **na vodorovné ose.**
- V Matlabu můžete pro „nekonečno" použít „inf", nebo např. 1E38.
- V prvních krocích algoritmu, tj. pro malé hodnoty i a j, vzniká problém s neexistujícími předchůdci, např. pro bod A(2,1) nám schází 2 předchůdci, pro bod A(2,2) jeden. Než to složitě řešit nějakými podmínkami typu IF, je implementačně vhodnější zavést ještě 2 řádky s j-tými indexy 0 a -1, tedy např. body A(1, 0) a A(1,-1), A(2, 0) a A(2,-1), ….. a do nich při inicializace vložit „nekonečno". Bohužel v Matlabu indexy 0 a -1 nejsou povoleny, takže si vytvořte pomocný index jj = j + 2, s nímž vhodně pracujte.



Tipy pro implementaci (2)

Implementace DTW algoritmu

- Operaci výběru minimálního předchůdce naprogramujte efektivně, nejlépe tak, že nejprve najdete menší z předchůdců A(i-1, j-2) a A(i-1, j-1), a toho pak porovnejte s A(i-1, j), přičemž vezměte v úvahu předchozí cestu do A(i-1, j). **V případě shody preferujte cestu jinou než horizontální** .

- Asi sami přijdete na to, že v **určitých případech bude DTW algoritmus nepoužitelný**. To se stane, když bude příliš velký rozdíl mezi délkami slova a reference. Odvoďte si přesně, kdy toto nastane – najděte si vztah pro nejstrmější možnou cestu i pro tu nejpozvolnější a z toho vám vyjdou 2 podmínky. Co v takovém případě? Z logiky věci vychází, že rozpoznávané slovo zde máme porovnat s délkově výrazně odlišnou referencí, a protože je malá pravděpodobnost, že správná reference by se délkou tolik lišila, můžeme rovnou vrátit vzdálenost rovnou „nekonečnu" a ušetřit si tak 1 výpočet.

- Je dobré implementovat DTW algoritmus jako **funkci** s parametry, např.

ComputeDTW (X, I, R, J, P)

kde .. P je počet příznaků
X je sekvence příznakových vektorů slova s I framy
R je sekvence příznakových vektorů reference s J framy

Tipy pro implementaci (3)

Implementace DTW algoritmu a rozpoznávače

- Využijte možnost kontroly implementace na dodaných datech.
- Na elearningu máte 2 příklady skutečných dvojic (testované slovo + reference). Všechna slova jsou reprezentována 2 příznaky. Na datech si můžete vyzkoušet, zda vaše implementace dává stejné výsledky jako moje. Do souboru MAT jsem zabalil jak vstupní data, tak výsledky, a dokonce i celé matice A a B.
- Při implementaci celého testovacího řetězce využijte modul pro načtení nahrávek ze seznamu, který jste použili již minule. Stačí potom poslat pouze váš program a požadované výsledky (např. v excelu).
- Řešení prosím opět nejdéle do pondělí 12.00.