Tomáš Oberhuber

Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University in Prague

Video na Youtube

Porovnávání sekvencí

- pokud biologové objeví nový gen, většinou se nezná jeho funkce
- zkouší se zjistit, kterým jiným genům je nejpodobnější
- bereme tedy postupně geny z databáze a porovnáváme je se zkoumaným genem
- snažíme se pak nějakým způsobem určit, jak moc jsou si dva geny podobné
- otázka je, jak ohodnotit podobnost dvou řetězců

Editační vzdálenost

- mějme dva l-mery v a w

Definition 1

Hammingova vzdálenost dvou řetězců v a w je definována jako počet pozic, na kterých se v a w liší.

Example 2

v = "ATTGTC"
w = "ACTCTC"

$$d_H(v, w) = 2$$

Editační vzdálenost

- Hammingova vzdálenost se ale **nehodí** k naší úloze porovnávání dvou řetězců

Example 3

v =	"ATATATAT"	v =	"ATATATAT-"
w =	"TATATATA"	w =	"-TATATATA"
	$d_H(v, w) = 8$		$d_H(v, w) = 2$

Editační vzdálenost

- vraťme se k naší úloze porovnávání dvou řetězců
- 1966, Vladimir Levenshtein – zadefinoval *editační* nebo také *levenshteinovu vzdálenost*
- ta je definována pomocí *editačních operací*
  - smazání znaku
  - vložení znaku
  - přepis znaku

Editační vzdálenost

Definition 4

Levenshteinova nebo editační vzdálenost mezi dvěma řetězci obecně různé délky je minimální počet editačních operací, které je potřeba provést k přepsání jednoho řetězce na druhý.

- Levenshtein sám nenašel algoritmus pro výpočet této vzdálenosti
- algoritmus byl objeven později nezávisle v několika různých oborech

Zarovnání řetězců

- k výpočtu editační vzdálenosti nejprve musíme provést tzv. zarovnání řetězců – *string alignment*

Definition 5

Mějme dva řetězce *v* a *w*. Jejich zarovnání je dvouřádková matice, která v prvním řádku obsahuje řetězec *v* a ve druhém řádku řetězec *w* oba případně doplněné o mezery tak, že v žádném sloupci nejsou dvě mezery.

Example 6

- **v =** `ATGTTAT`
- **w =** `ATCCTAC`

v = `AT-GTTAT-`

w = `ATCCT-A-C`

Zarovnání řetězců

- pokud jsou v jednom sloupci oba znaky stejné, jde o shodu (*match*)
- pokud jsou v jednom sloupci oba znaky různé, jde o neshodu/přepis (*mismatch*)
- je-li v jednom sloupci mezera, jde o tzv. indel
  - mezera v prvním řádků je vložení znaku – *insertion*
  - mezera v druhém řádku je mazání znaku – *deletion*
- jde nám o to, jak najít zarovnání s co nejmenším počtem editačních operací
- to získáme pomocí dynamického programováni
- nejprve si ale ukážeme jednduší úlohu, kde nebudeme uvažovat přepisování znaků, tj. jenom vkládání a mazání

Nejdelší společná podposloupnost

Definition 7

Podposloupnost (*subsequence*) řetězce je posloupnost znaků jdoucích po sobě, ale ne nutně bezprostředně po sobě. Tj. pro $\mathbf{v}=v_1 v_2 \ldots v_n$ a libovolnou ostře rostoucí posloupnost indexů $1 \leq i_1 < i_2 \ldots i_k \leq n$ je řetězec $v_{i_1} v_{i_2} \ldots v_{i_k}$ podposloupnost **v.**

Example 8

- **v =** `ATTGCTA`
- `AGCA` **a** `ATTA` **jsou podposloupnosti v**
- `TGTT` **není podposloupnost v – po** `GT` **již nenásleduje T**

Nejdelší společná podposloupnost

Definition 9

Společná podposloupnost (*common subsequence*) řetězců $\mathbf{v}=v_1 \ldots v_n$ a $\mathbf{w}=w_1 \ldots w_m$ a je posloupnost pozic

$$1 \leq i_1 < i_2, \ldots i_k \leq n,$$

ve $\mathbf{v}$ a posloupnost pozic

$$1 \leq j_1 < j_2, \ldots, j_k \leq m,$$

ve $\mathbf{w}$ takových, že

$$v_{i_t} = w_{j_t},$$

pro všechna $t = 1, \ldots, k$.

Nejdelší společná podposloupnost

- pokud provedeme zarovnání dvou řetězců bez přepisů, stejné znaky pod sebou tvoří společnou podposlounpost

Example 10

- **v =** ATGTTAT
- **w =** ATCCTAC

v = AT-G-TTAT-
w = ATC-CT-A-C

Nejdelší společná podposloupnost

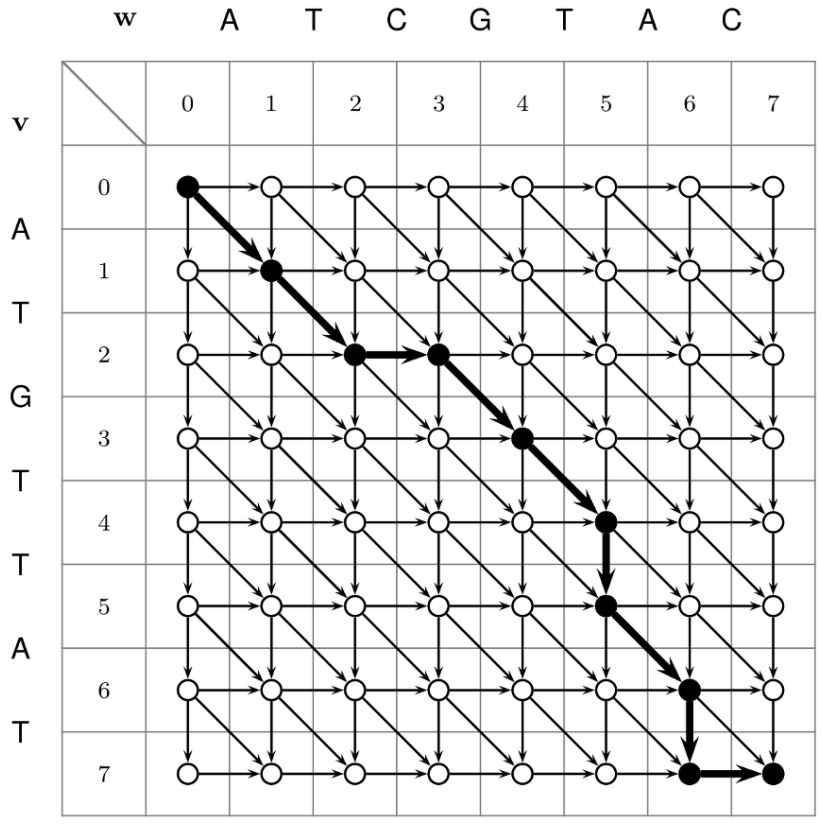
LCS = longest common subsequence

- naším úkolem bude najít nejdelší společnou podposloupnost dvou řetězců **v** a **w**
- k výpočtu zarovnání použijeme algoritmus dynamického programování
- řetězce napíšeme podél horní a levé strany grafu, tak že znaky řetězců jsou zarovnané na hrany grafu
- tam, kde se řetězce shodují, uděláme šikmou hranu doprava dolu
- dolníme všechny svislé a vodorovné hrany

Nejdelší společná podposloupnost

		0	1	2	2	3	4	5	6	7	7
v	=		A	T	-	G	T	T	A	T	-
			|	|		|	|		|		
w	=		A	T	C	G	T	-	A	-	C
		0	1	2	3	4	5	5	6	6	7

↘	↘	→	↘	↘	↓	↘	↓	→
A	T	-	G	T	T	A	T	-
A	T	C	G	T	-	A	-	C

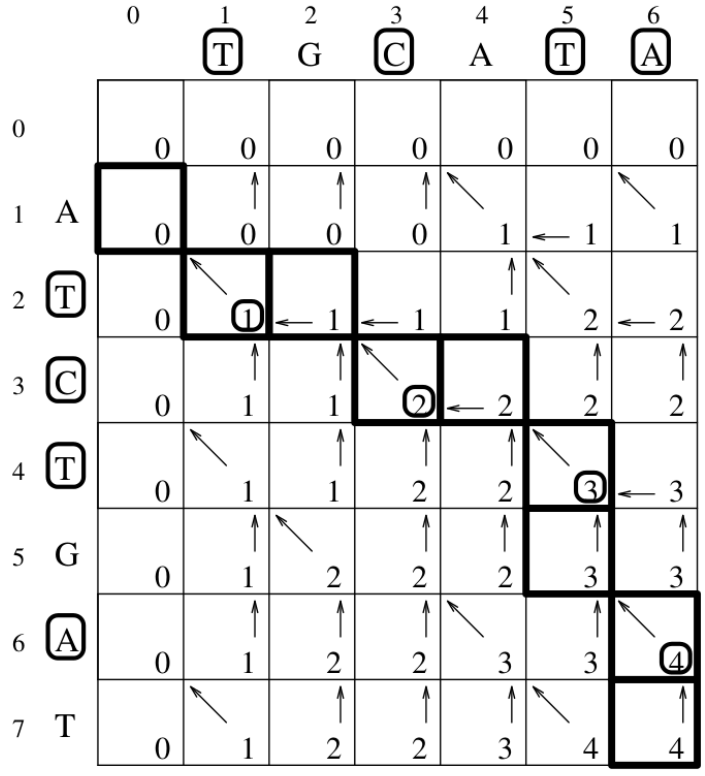


Zdroj: Jones, Pevzner, An introduction to bioinformatics algorithms

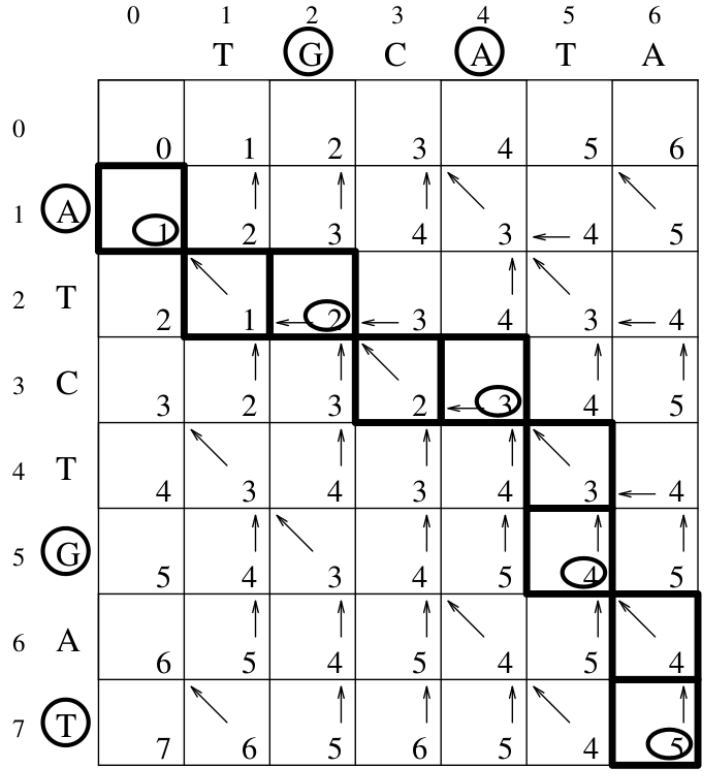
Nejdelší společná podposloupnost

- jakákoliv cesta z levého horního do pravého dolního rohu pak odpovídá nějakému zarovnání
  - krok po šikmé hraně odpovídá shodě
  - krok po svislé nebo vodorovné hraně odpovídá mezeře v jednom z řetězců
- pokud šikmé hrany ohodnotím jedničkou a ostatní nulou, stačí už jen najít nejdelší cestu v grafu
- její délka pak odpovídá délce nejdelší společné podposloupnosti

Nejdelší společná podposloupnost



Computing similarity s(V,W)=4
V and W have a subsequence TCTA in common



Computing distance d(V,W)=5
V can be transformed into W by deleting A,G,T and inserting G,A

Alignment:

A T - C - T G A T
- T G C A T - A -

Nejdelší společná podposloupnost

- řešíme tedy úlohu dynamického programování s předpisem

$$s_{ij} = \max \begin{cases} s_{i-1,j} + 0 & \\ s_{i,j-1} + 0 & \\ s_{i-1,j-1} + 1 & \text{pokud } v_i = w_j \end{cases}$$

- pokud chci následně získat LCS, jde pouze o rekonstrukci nejdelší cesty

Globální porovnávání řetězců

- nyní se vracíme k naší původní úloze porovnávání řetězců
- kromě vkládání a mazání znaků umožníme provádět i přepisy právě podle definice editační vzdálenosti
- do našeho grafu dynamického programování tak vložíme šikmé hrany i tam, kde se oba řetězce nerovnají
- tyto nové hrany ale musíme ohodnotit menší vahou než 1, aby algoritmus upřednostňoval aplikaci shody řetězců

Globální porovnávání řetězců

Obecně můžeme použít následující penalizace:

- $-\mu$ pro přepis
- $-\sigma$ pro vložení nebo mazání znaku
- 1 pro shodu znaků

Tím získáme následující pravidlo:

$$s_{ij} = \max \begin{cases} s_{i-1,j} - \sigma & \\ s_{i,j-1} - \sigma & \\ s_{i-1,j-1} - \mu & \text{pro } v_i \neq w_j \\ s_{i-1,j-1} + 1 & \text{pro } v_i = w_j \end{cases}$$

Example 11

http://demonstrations.wolfram.com/GlobalAndLocalSequenceAlignmentAlgorithms/

Globální porovnávání řetězců

- to lze zobecnit zavedením matice δ, která všechny možné dvojice znaků, které se mohou vyskytnout pod sebou v matici zarovnání.
- v našem případě by vypadala takto

$$\delta \equiv \left(\begin{array}{c|ccccc} & A & T & C & G & - \\ \hline A & 1 & -\mu & -\mu & -\mu & -\sigma \\ T & -\mu & 1 & -\mu & -\mu & -\sigma \\ C & -\mu & -\mu & 1 & -\mu & -\sigma \\ G & -\mu & -\mu & -\mu & 1 & -\sigma \\ - & -\sigma & -\sigma & -\sigma & -\sigma & -\infty \end{array} \right)$$

- matici δ nazýváme maticí skóre (*scoring matrix*)

Globální porovnávání řetězců

- algoritmus dynamického programování by se pak řídil pravidlem

$$s_{ij} = \max \begin{cases} s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$

- výhodou je, že nyní můžeme každý přepis ohodnotit jinak

Program diff

- je založen na algoritmu pro hledání LCS ale po řádcích, ne po znacích
- E.W.Myers, *An O(ND) Difference Algorithm and Its Variations*, Algorithmica, (1), 251–266, 1986.
  - N je součet délek řetězců v a w
  - D jejich editační vzdálenost
  - lze dosáhnout složitosti $O(N + D^2)$
  - paměťová náročnost je $O(N)$