

Tomáš Oberhuber

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

Video na Youtube

Lokální porovnávání řetězců

- v některých situacích nás více zajímá co nejlepší shoda určitých podřetězců obou sekvencí raději, než porovnání celých sekvencí

```

- - T - - CC - C - AGT - - TATGT - CAGGGGACACG - - A - GCATGCAGA - GAC
    |  | | |  | | | | |  | | | | | | | | | |
AATTGCCGCC - GTCGT - T - TTCAG - - - CA - GTTATG - - T - CAGAT - - C

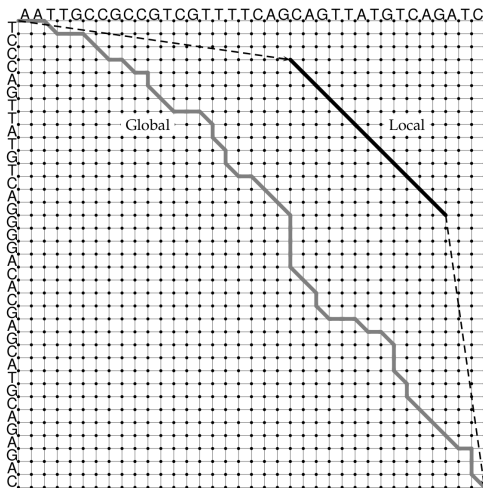
                                tccCAGTTATGTCTCAGgggacacgagcatgcagagac
                                | | | | | | | | | |
aattgccgccgctcgtttttcagCAGTTATGTCTCAGatc

```

Zdroj: Jones, Pevzner, An introduction to bioinformatics algorithms

- v grafu dynamického programování takové zarovnání vypadá takto...

Lokální porovnávání řetězců

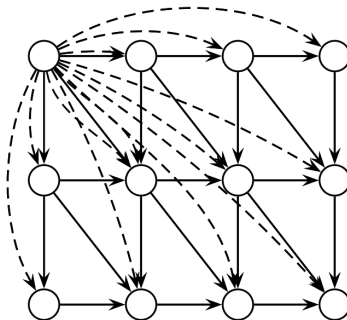


Zdroj: Jones, Pevzner, An introduction to bioinformatics algorithms

Lokální porovnávání řetězců

- co brání nalezení takové cesty?
- jsou to penalizace nabrané před a po požadovaném zarovnání
- rádi bychom našli způsob, jak je najít
- penalizací před zarovnáním se lze zbavit tak, umožníme přeskočit tu část zarovnání, kde získáváme záporné body
- v grafu to znamená, že ke každému vrcholu přidáme hranu vedoucí do levého horního rohu s nulovou vahou

Lokální porovnávání řetězců



Zdroj: Jones, Pevzner, An introduction to bioinformatics algorithms

Lokální porovnávání řetězců

- vztahy pro úlohu dynamického programování pak mají tvar

$$s_{ij} = \max \begin{cases} 0 \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$

- jak se zbavit penalizací za lokálním zarovnáním?
- po ohodnocení všech vrcholů grafu algoritmem dynamického programování najdeme vrchol s nejvyšším skóre
- ten znamená konec nejlepšího lokálního zarovnání
- od něj pak běžným způsobem rekonstruujeme nejdelší cestu, dokud nenarazíme na vrchol s nulovou hodnotou
- jde o Smithův-Watermanův algoritmus

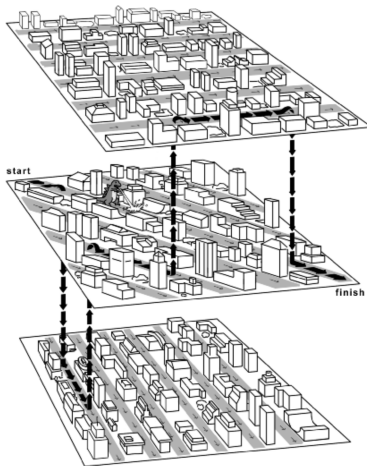
Porovnávání řetězců s dlouhými sekvencemi

- v přírodě jsou častější podobnosti, kdy se záměny provádí po delších sekvencích a ne po jednotlivých nukleotidech
- k nalezení takových zarovnání je potřeba zavést vyšší penalizaci pro vložení první mezery
 - první mezeru v sekvenci mezer budeme navíc penalizovat hodnotou $-\rho$
 - další mezery hodnotou $-\sigma$
- problém je v tom, jak v daném bodě zjistit, na jak dlouhou sekvenci mezer napojujeme další mezeru
- mohli bychom zkusit opět přidat další hrany do grafu
- bylo by ale nutné přidat všechny možné svislé a vodorovné hrany, což by zpomalilo algoritmus

Porovnávání řetězců s dlouhými sekvencemi

- zavedeme si dva pomocné grafy $s^{\rightarrow}$ a $s^{\downarrow}$
- nyní budeme hledat nejdelší cestu po třech grafech
- v jednom můžeme jít jen doprava, ve druhém dolů a ve třetím šikmo dolů

Porovnávání řetězců s dlouhými sekvencemi



Zdroj: Jones, Pevzner, An introduction to bioinformatics algorithms

Porovnávání řetězců s dlouhými sekvencemi

- rekurentní vztahy pak mají tvar

$$s_{ij}^{\downarrow} = \max \begin{cases} s_{i-1,j}^{\downarrow} - \sigma \\ s_{i-1,j} - (\rho + \sigma) \end{cases}$$

$$s_{ij}^{\rightarrow} = \max \begin{cases} s_{i,j-1}^{\rightarrow} - \sigma \\ s_{i,j-1} - (\rho + \sigma) \end{cases}$$

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

Porovnávání více sekvencí

- nyní se pokusíme o porovnávání více sekvencí najednou
- pro tři řetězce v , w , u to může vypadat například takto

```

- - T - - CC - C - AGT - - TATGT - CAGGGGACACG - - A - GCATGCAGA - GAC
  |  | | | |  | | | | |  | | | | |  | | | | |  |
AATTGCCGCC - GTCGT - T - TTCAG - - - - CA - GTTATG - - T - CAGAT - - C
  | | | | | |  X | | | | |  | | XXX | | | | | |  |
- ATTGC - G - - ATTCGTAT - - - - - GGGACA - TGGATGCATGCAG - TGAC
  
```

Zdroj: Jones, Pevzner, An introduction to bioinformatics algorithms

Porovnávání více sekvencí

- nyní při sestavování každého dalšího sloupce matice můžeme provést:
 - vložení znaku z jednoho řetězce a dvě mezery
 - vložení různých znaků ze dvou řetězců a jedné mezery
 - vložení dvou stejných znaků a jedné mezery
 - vložení tří různých znaků
 - vložení tří stejných znaků
- to vše musíme opět vhodně ohodnotit
- použijeme opět "matici" δ
- úloha dynamického programování se bude řešit ve třech dimenzích

Porovnávání více sekvencí

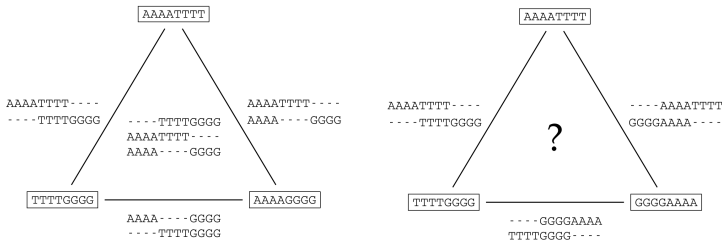
- rekurence pro úlohu dynamického programování má tvar:

$$s_{ijk} = \max \left\{ \begin{array}{l} s_{i-1,j,k} + \delta(v_i, -, -) \\ s_{i,j-1,k} + \delta(-, w_j, -) \\ s_{i,j,k-1} + \delta(-, -, u_k) \\ s_{i-1,j-1,k} + \delta(v_i, w_j, -) \\ s_{i-1,j,k-1} + \delta(v_i, -, u_k) \\ s_{i,j-1,k-1} + \delta(-, w_j, u_k) \\ s_{i-1,j-1,k-1} + \delta(v_i, w_j, u_k) \end{array} \right.$$

Porovnávání více sekvencí

- snadno tak lze odvodit algoritmus pro porovnávání obecně k sekvencí
- problém je ale v tom, že složitost je $O(n^k)$, pokud budou mít všechny sekvence délku přibližně n
- pro $n = 1000$ a $k = 3$ jsou paměťové nároky $\text{sizeof}(\text{int}) \cdot 1000^3 = 4 \text{ GB}$
- často tedy musíme použít heuristiky
- mohli bychom zkusit napočítat optimální zarovnání všech $\binom{k}{2}$ dvojic a ty pak zkombinovat dohromady
- to se ale nemusí vždy podařit

Porovnávání více sekvencí



Zdroj: Jones, Pevzner, An introduction to bioinformatics algorithms

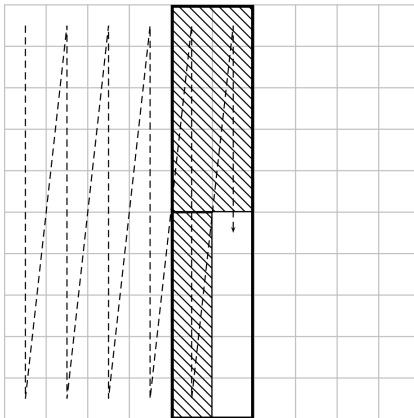
Porovnávání více sekvencí

- efektivním algoritmem je progresivní zarovnávání sekvencí
- vybere se dvojice sekvencí s nejlepším zarovnáním
- ty se pak sloučí do jednoho řetězce
- k němu se pak přidává další
- tento algoritmus používá známý program Clustal

Paměťově efektivní porovnání sekvencí

- problém algoritmu pro porovnávání sekvencí není ani tak v časové složitosti jako spíš ve velkém nároku na paměť
- 1975 - Daniel Hirschberg, navrhl algoritmus typu "rozděl a panuj", který paměťové nároky snižuje
- využívá faktu, že k napočítání j -tého sloupce matice/grafu s nám stačí znát pouze $j - 1$ -ní sloupec
- předchozí sloupce využívá pouze při rekonstrukci nejdelší cesty
- pokud by nám stačilo napočítat pouze délku nejdelší cesty, nemusíme si tyto sloupce už pamatovat

Paměťově efektivní porovnání sekvencí



Zdroj: Jones, Pevzner, An introduction to bioinformatics algorithms

Paměťově efektivní porovnání sekvencí

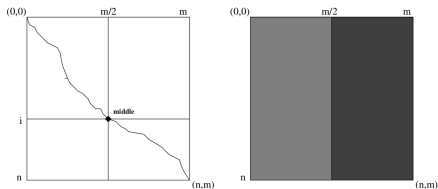
- buď j^* index prostředního sloupce
- nechť nejdelší cesta prochází jeho i^* -tým řádkem
- otázka je, jak tento index i^* najít?
- délky všech n cest z levého horního rohu do pravého dolního lze vyjádřit pro $i = 1, \dots, n$ jako součet délek
 - z levého horního rohu do j^* -tého sloupce – $prefix_i$
 - z pravého dolního rohu do j^* -tého sloupce – $suffix_i$
 - řešíme úlohu dynamického programování se startem v pravém dolním rohu
- snadno pak vidíme, že

$$i^* = \arg \max_{i=1, \dots, n} (prefix_i + suffix_i)$$

Paměťově efektivní porovnání sekvencí

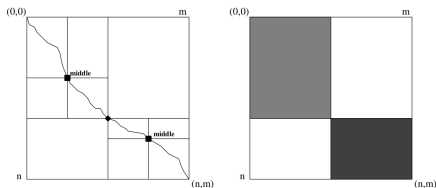
- k výpočtu *prefix_i* řešíme úlohu dynamického programování se zahazováním nepotřebných sloupců s počátkem v **levém horním** rohu
- k výpočtu *suffix_i* řešíme úlohu dynamického programování se zahazováním nepotřebných sloupců s počátkem v **pravém dolním** rohu
- celkem tedy potřebujeme ukládat jen $3n$ prvků
- pokud známe souřadnice (i^*, j^*) rozdělíme problém na dva
 - nalezení nejdelší cesty z levého horního bodu do (i^*, j^*)
 - nalezení nejdelší cesty z (i^*, j^*) do pravého dolního rohu

Paměťově efektivní porovnání sekvencí



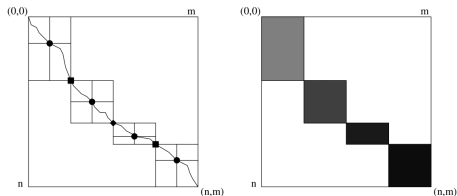
Zdroj: Jones, Pevzner, An introduction to bioinformatics algorithms

Paměťově efektivní porovnání sekvencí



Zdroj: Jones, Pevzner, An introduction to bioinformatics algorithms

Paměťově efektivní porovnání sekvencí



Zdroj: Jones, Pevzner, An introduction to bioinformatics algorithms

Paměťově efektivní porovnání sekvencí

- jaká je časová složitost?
- po každém dělení procházíme jen polovinu uzlů grafu s
znovu, tj.

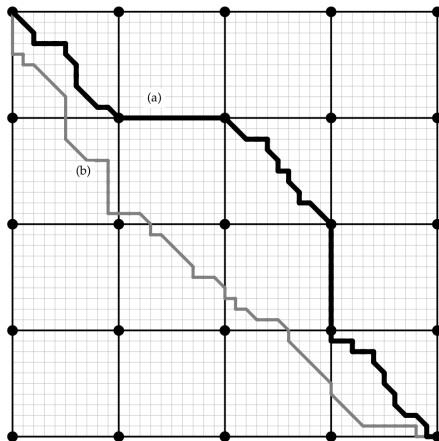
$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots \leq 2$$

- algoritmus tedy poběží maximálně dvakrát tak dlouho
jako normální algoritmus
- protože ale máme větší šanci, že výpočet poběží v
cache CPU, může být výpočet ještě výrazně rychlejší

Blokové porovnávání sekvencí

- zatím není známý žádný spodní odhad složitosti pro porovnávání sekvencí
- algoritmus dynamického programování vede na kvadratickou složitost
- ukážeme si, jak algoritmus urychlit za cenu horšího výsledku
- použijeme tzv. **blokové porovnávání**
- předpokládejme pro jednoduchost velikost úlohy $n \times n$, tj. $|v| = |w| = n$
- graf s rozdělíme na bloky o velikosti $t \times t$, a cesty omezíme tak, že musí procházet vždy pouze skrz rohy bloků

Blokové porovnávání sekvencí



Zdroj: Jones, Pevzner, An introduction to bioinformatics algorithms

Blokové porovnávání sekvencí

- nyní víme, že do každého bloku může každá cesta vstupovat jen levým horním rohem a vystupovat
 - pravým horním rohem $\rightarrow -t\sigma$
 - levým dolním rohem $\rightarrow -t\sigma$
 - pravým dolním rohem $\rightarrow \beta_{ij}$ - musíme spočítat
- můžeme si předpočítat cesty skrz jednotlivé bloky
- získané skóre v každém bloku označíme jako β_{ij} tj. délka nejdelší cesty blokem
- následně řešíme úlohu dynamického programování pro bloky
- sestavíme tedy graf S tvořený jednotlivými bloky a řešíme úlohu

$$S_{ij} = \max \begin{cases} S_{i-1,j} - t\sigma \\ S_{i,j-1} - t\sigma \\ S_{i-1,j-1} + \beta_{ij} \end{cases},$$

Blokové porovnávání sekvencí

- složitost blokové úlohy je $O(n^2/t^2)$
- vyřešení jednoho bloku je $O(t^2)$
- bloků je n^2/t^2 , což dává opět $O(n^2)$
- abychom získali rychlejší algoritmus, je potřeba si skóre pro jednotlivé bloky β_{ij} předpočítat
- každý blok je jednoznačně určen částí řetězce w nad jeho horní hranou a částí řetězce v podél jeho levé hrany
- tj. znaky $v_{it}, v_{it+1}, \dots, v_{it+t-1}$ a $w_{jt}, w_{jt+1}, \dots, w_{jt+t-1}$
- v abecedě se čtyřmi znaky je to celkem $4^t \times 4^t$ možností
- pokud volíme $t = \log_4 n/2$ je

$$4^t \cdot 4^t = 4^{\log_4 n/2} \cdot 4^{\log_4 n/2} = n^{\frac{1}{2}} n^{\frac{1}{2}} = n$$

- to lze předpočítat do tabulky se složitostí $O(n \log^2 n)$

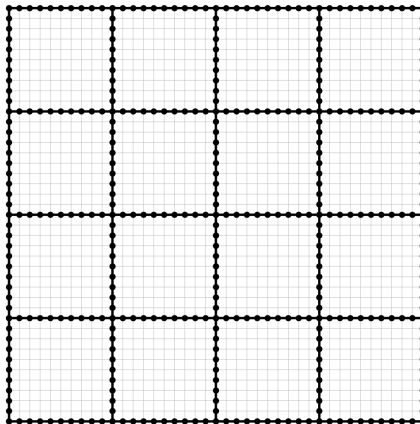
Blokové porovnávání sekvencí

- složitost blokového dynamického programování je $O(n^2 / \log^2 n)$
- celkem tedy $O(n^2 / \log^2 n)$
- trik je tedy v tom, že pokud si velikost bloků zvolíme dostatečně malou, musí se některé bloky opakovat
- tím, že je ale předpočítám do tabulky, napočítám každý blok jen jednou

Blokové porovnávání pro LCS

- pro úlohu nalezení nejdelší společné podposloupnosti lze pomocí bloků získat dokonce optimální řešení
- umožníme, aby libovolná cesta mohla křížovat hranici bloku kdekoliv, ne jen v protějších rozích

Blokové porovnávání pro LCS



Zdroj: Jones, Pevzner, An introduction to bioinformatics algorithms

Blokové porovnávání pro LCS

- řešíme tedy úlohu dynamického programování na všech uzlech na hranách bloků
- každý blok je nyní popsán
 - podřetězci $v_{it}, v_{it+1}, \dots, v_{it+t-1}$ a $w_{jt}, w_{jt+1}, \dots, w_{jt+t-1}$
 - hodnotami skóre s_{ij} podél své horní a levé hrany
- ze znalosti těchto vstupních dat dokážeme napočítat skóre s_{ij} v libovolném uzlu podél pravé a dolní hrany bloku
- pro úlohu globálního porovnávání řetězců je to příliš mnoho kombinací na to, aby se daly předpočítat do tabulky
- pro úlohu LCS to ale lze

Blokové porovnávání pro LCS

- úloha LCS je dána předpisem

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

- skóre s_{ij} se tedy může podél horní a levé strany bloku měnit pouze o 0 nebo 1 mezi jednotlivými uzly
 - hodnota s_{ij} udává maximální možný počet šikmých hran na cestě do uzlu (i, j)
 - jelikož se nemohu vracet zpátky, tak počet šikmých hran, kterými jsem došel do sousedního uzlu se může lišit maximálně o 1
- to lze zakódovat do posloupnosti nul a jedniček o délce t

Blokové porovnávání pro LCS

- celkem tak máme

$$4^t \dots 4^t \cdot 2^t \cdot 2^t = 2^{6t}$$

kombinací

- pro $t = \log_2 n/4 \rightarrow n^{1.5}$ kombinací
- zmíněná technika se nazývá Four-Russians Speed-up po autorech
- V.L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradzev. On economical construction of the transitive closure of an oriented graph. Soviet Math. Dokl., 11:1209– 1210, 1970.