

Zarovnání sekvencí

Bioinformatika

Tomáš Martínek

martinto@fit.vutbr.cz

Osnova

- Úvod
 - Dot Plot
 - Jednoduché zarovnání
 - Skórovací matice
- Dynamické programování
 - Needleman-Wunch
 - Smith-Waterman
- Heuristické algoritmy
 - BLAST
 - FASTA
- Shrnutí

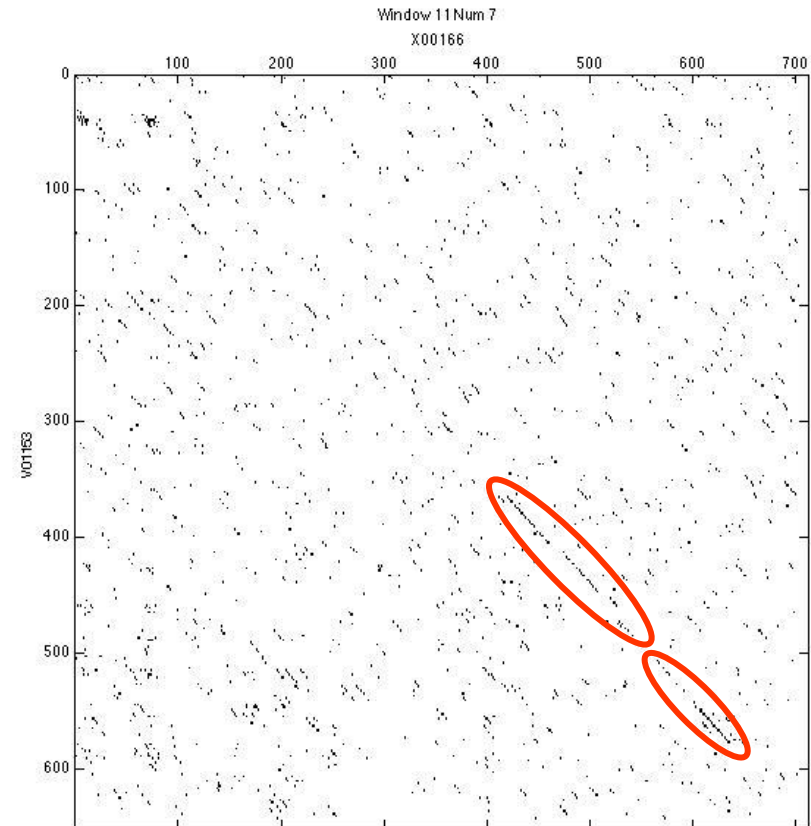
Motivace

- Hledání zarovnání řetězců je jednou ze **základních úloh bioinformatiky** - dává informaci např. o tom, jak se organizmy od sebe liší a napomáhají určit i funkci jednotlivých elementů
- **Příklady základních otázek a úloh:**
 - Jak se liší genom člověka od genomu šimpanze? [porovnání]
 - Jak se liší protein Myosin u různých organizmů? [porovnání]
 - Vyskytuje se daný element (gen, protein, promotor, reg. faktor, ...) i u jiných organizmů? [vyhledávání]
- **Nejčastější změny v rámci evolučního procesu**
 - **mutace = záměna znaku** - poměrně časté
 - **vložení/odstranění znaku** - menší pravděpodobnost

Dot Plot

- Jedna z nejjednodušších metod
- Základem je matice, kde:
 - sloupce reprezentují jeden řetězec
 - řádky reprezentují druhý řetězec
 - na příslušné pozici průsečíku řádku a sloupce se vloží bod, pouze pokud jsou znaky shodné
- Nevýhody:
 - pouze orientační grafická reprezentace, nevíme, jak je výsledná sekvence zarovnána
 - vede na vznik šumu

- Příklad:

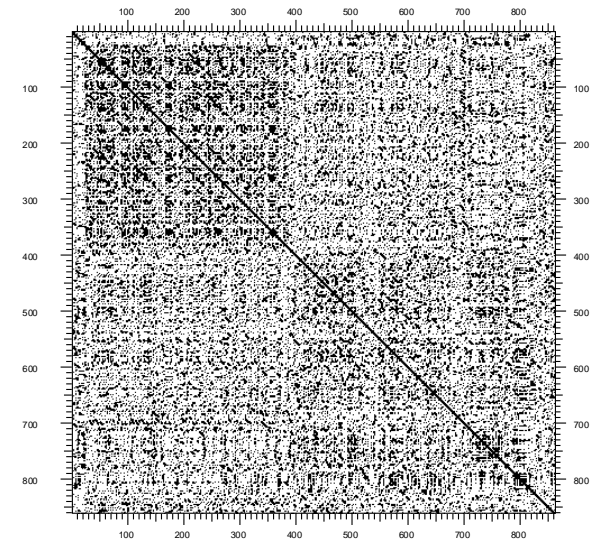


Dot Plot

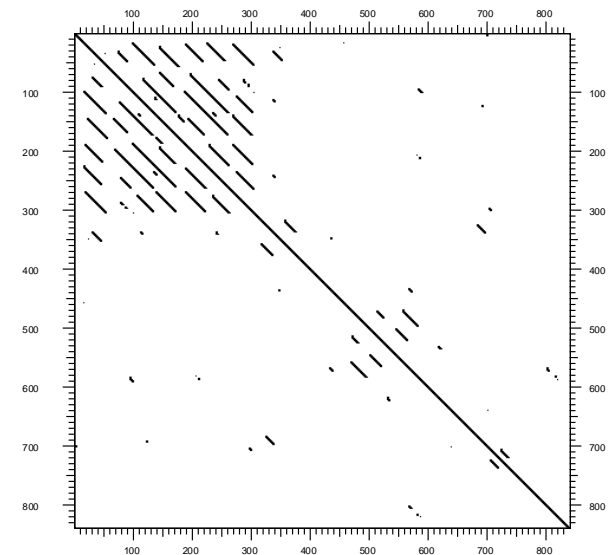
- **Vylepšení: posouvající se okénko**
 - okénko o velikosti např. 10 znaků se postupně jedním nukleotidu posouvá jak ve směru sloupců, tak ve směru řádků
 - v okénku se porovnává celých 10 znaků a pokud je alespoň 8 z nich shodných, potom se vloží na pozici začátku okénka bod
 - je potřeba vhodně zvolit velikost okénka a míru shody
- **Výhody:**
 - odstraní šum
- **Nevýhody:**
 - pouze grafická reprezentace
 - vysoká časová složitost algoritmu - $O(n^2)$

• Příklad:

• Window Size = 1



• Window Size = 7



Jednoduchá metoda zarovnání

- Uvažujeme pouze mutace, nikoliv vložení/odstranění znaku
- Předpokládáme, že kratší řetězec vznikne z delšího vložním mezer, u delšího řetězce mezery neuvažujeme
- Postup:
 - začátek kratšího řetězce se postupně posunuje vzhledem k delšímu řetězci a pro každý posun se vyhodnocuje počet shodujících/neshodujících se znaků
- Vzorec pro výpočet skóre:

$$\sum_{i=1}^n \begin{cases} 1: & \text{if } (s_1[i] = s_2[i]) \\ 0: & \text{if } (s_1[i] \neq s_2[i]) \end{cases}$$

- Příklad:
 - zarovnání sekvencí AATCTATA a AAGATA

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A | A | T | C | T | A | T | A |
| A | A | G | A | T | A | | |
| 1 | 1 | 0 | 0 | 1 | 1 | | |

Skóre

4

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A | A | T | C | T | A | T | A |
| | A | A | G | A | T | A | |
| | 1 | 0 | 0 | 0 | 0 | 0 | |

1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A | A | T | C | T | A | T | A |
| | | A | A | G | A | T | A |
| | | 0 | 0 | 0 | 1 | 1 | |

2

Vložení mezer

- Pro analýzu reálných sekvencí je nezbytné uvažovat vložení a odstranění znaků
- Vede na vznik mezer a hledání takového zarovnání výrazně komplikuje výpočet
- Zatímco u příkladu jednoduchého zarovnání byly pouze 3 možnosti, s mezerami je to celkem 28 možností
- Vzorec pro výpočet skóre:

$$\sum_{i=1}^n \begin{cases} 1 & \text{if } (s_1[i] = s_2[i]) \\ 0 & \text{if } (s_1[i] \neq s_2[i]) \\ -1 & \text{if } (s_1[i] = '-' \parallel s_2[i] = '-') \end{cases}$$

- Poznámka:** Je běžné, že dvě nebo více zarovnání obsahují stejné skóre

Příklad:

- zarovnání sekvencí AATCTATA a AAGATA - pouze 3 z 28 kombinací

Skóre

| | | | | | | | |
|---|---|---|----|---|---|----|---|
| A | A | T | C | T | A | T | A |
| A | A | G | - | A | T | - | A |
| 1 | 1 | 0 | -1 | 0 | 0 | -1 | 1 |

1

| | | | | | | | |
|---|---|----|---|----|---|---|---|
| A | A | T | C | T | A | T | A |
| A | A | - | G | - | A | T | A |
| 1 | 1 | -1 | 0 | -1 | 1 | 1 | 1 |

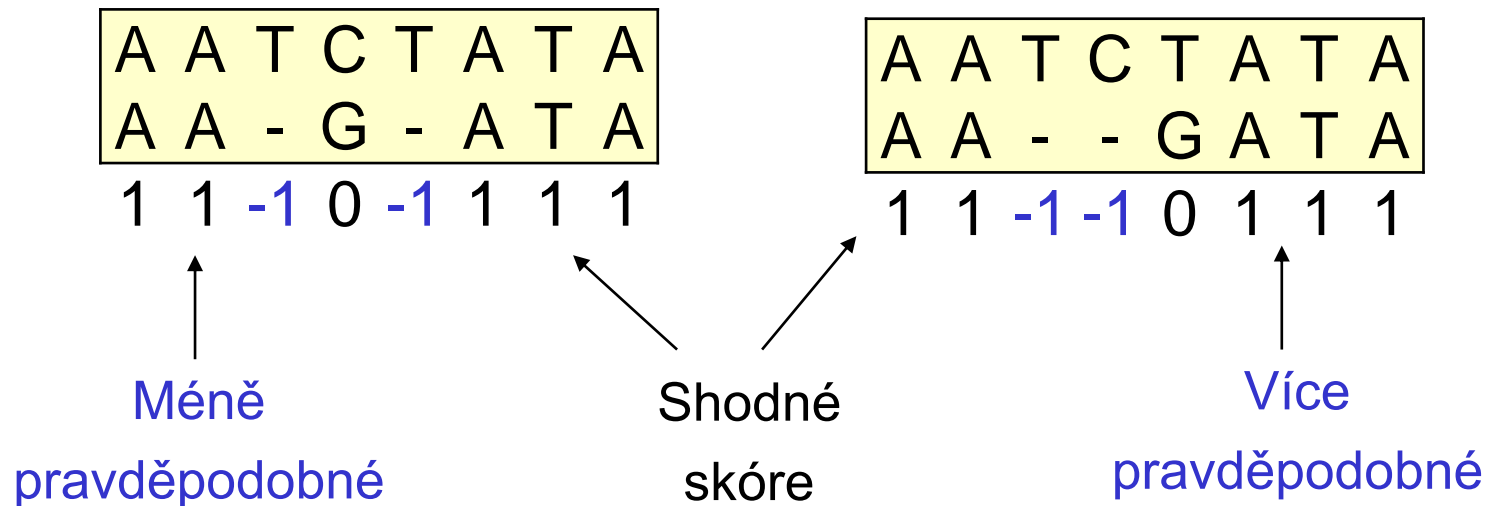
3

| | | | | | | | |
|---|---|----|----|---|---|---|---|
| A | A | T | C | T | A | T | A |
| A | A | - | - | G | A | T | A |
| 1 | 1 | -1 | -1 | 0 | 1 | 1 | 1 |

3

Vložení mezer

- Z pohledu evoluce je však potřeba rozlišovat mezi různými typy mezer
- Je daleko pravděpodobnější vznik menšího počtu delších mezer, než velkého počtu krátkých mezer



Vložení mezer

- Zavedení dvojí penalizace za vložení mezery:

- ρ - za započetí mezery
- σ - za rozšíření mezery

- Výpočet skóre pro mezeru délky x je ve tvaru:

$$-(\rho + \sigma x)$$

- Příklad modifikace vzorce:

$$\sum_{i=1}^n \begin{cases} 1 & \text{shoda} \\ 0 & \text{neshoda} \\ -1 & \text{pokracujic } i \text{ mezera} \\ -2 & \text{pocatecni mezera} \end{cases}$$

- Poznámka:** Všimněte si, že kvalita výsledku závisí na vybrané skórovací funkci

- Příklad:

- zarovnání sekvencí AATCTATA a AAGATA - pouze 3 z 28 kombinací

Skóre

| | | | | | | | |
|---|---|---|----|---|---|----|---|
| A | A | T | C | T | A | T | A |
| A | A | G | - | A | T | - | A |
| 1 | 1 | 0 | -3 | 0 | 0 | -3 | 1 |

-3

| | | | | | | | |
|---|---|----|---|----|---|---|---|
| A | A | T | C | T | A | T | A |
| A | A | - | G | - | A | T | A |
| 1 | 1 | -3 | 0 | -3 | 1 | 1 | 1 |

-1

| | | | | | | | |
|---|---|----|----|---|---|---|---|
| A | A | T | C | T | A | T | A |
| A | A | - | - | G | A | T | A |
| 1 | 1 | -3 | -1 | 0 | 1 | 1 | 1 |

1

Skórovací matice

- Prozatím jsme rozlišovali různé typy mezer
- Ve skutečnosti je ale potřeba rozlišovat i záměny mezi různými znaky
- Příklady:
 - záměna mezi A/G (puríny) a C/T (pyrimidíny) jsou daleko pravděpodobnější, než změny mezi purínem a pyrimidýnem
 - na úrovni kodonů (trojice nukleotidů tvořící aminokyselinu): jsou pravděpodobnější jednobodové mutace než dvoubodové nebo třibodové
 - podobně u aminokyselin: změny mezi hydrofobními aminokyselinami mají daleko menší důsledky na změnu funkce než změny z hydrofobní na aminokyselinu s nábojem

Skórovací matice

- Vznik skórovacích matic pro ohodnocení záměny mezi všemi kombinacemi znaků
- Příklady matic pro nukleotidy:

| | A | T | C | G |
|---|---|---|---|---|
| A | 1 | 0 | 0 | 0 |
| T | 0 | 1 | 0 | 0 |
| C | 0 | 0 | 1 | 0 |
| G | 0 | 0 | 0 | 1 |

Matice
identity

| | A | T | C | G |
|---|----|----|----|----|
| A | 5 | -4 | -4 | -4 |
| T | -4 | 5 | -4 | -4 |
| C | -4 | -4 | 5 | -4 |
| G | -4 | -4 | -4 | 5 |

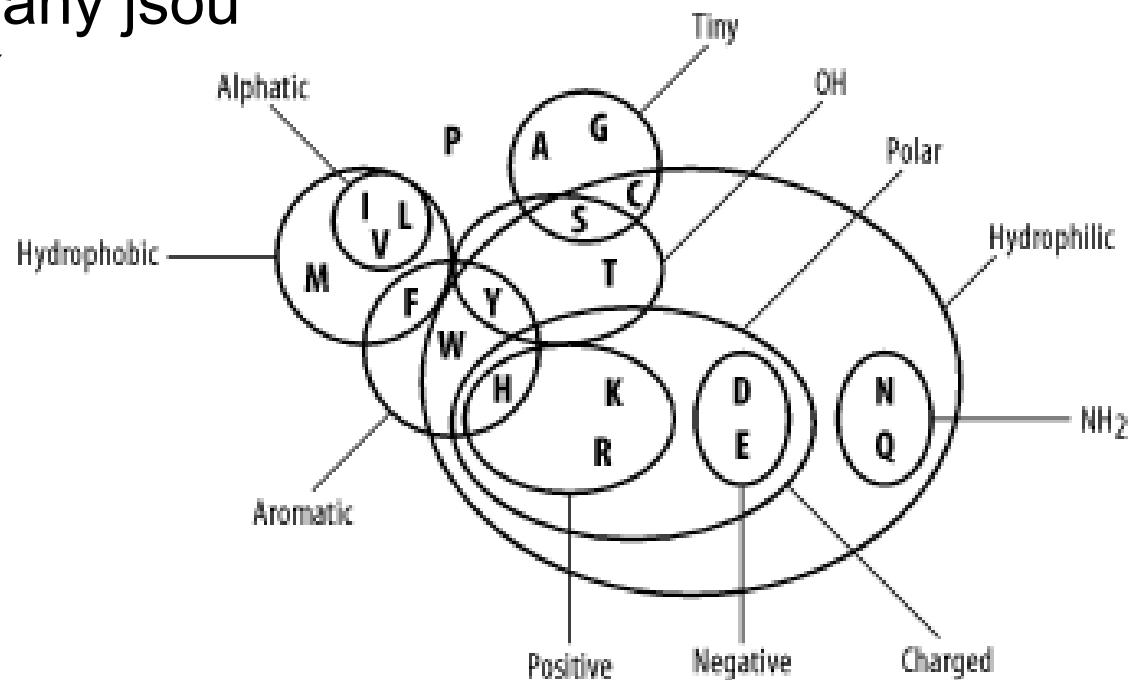
Matice
BLAST

| | A | T | C | G |
|---|----|----|----|----|
| A | 1 | -5 | -5 | -1 |
| T | -5 | 1 | -1 | -5 |
| C | -5 | -1 | 1 | -5 |
| G | -1 | -5 | -5 | 1 |

Matice s
transverzí

Skórovací matice

- Při sestavování skórovací matice pro aminokyseliny je nezbytné uvažovat podobnost z pohledu chemické struktury:
hydrofonicitu, náboj, elektronegativitu a velikost
- Diagram chemické podobnosti aminokyselin [Margaret Dayhoff, 70-tá léta]: zakroužkovány jsou aminokyseliny jež mají podobné vlastnosti
- Změny mezi aminokyselinami s podobnou chemickou strukturou jsou více pravděpodobné



Skórovací matice

- Obecné odvození je velmi komplikované. Proto se velmi často používá přístup, kdy se hodnoty v maticích určí experimentálně na základě rychlosti substituce mezi sekvencemi, u kterých je znám směr vývoje (fylogenetický strom)
- Příkladem jsou tzv. **PAM (Point Accepted Mutation) matice**
 - jsou sestaveny a normalizovány tak, že jejich vzájemným vynásobením získáme matice pro různě odlišné sekvence
 - **PAM-1** pro velmi blízké sekvence
 - **PAM-1000** pro sekvence, které značně liší
 - **PAM-250** obvyklý kompromis
- Podobně jsou konstruovány i matice **BLOSUM-XX**, kde XX označuje procentuelní míru podobnosti sekvencí (např. **BLOSUM-62** pro porovnání sekvencí s 62% podobností)

Skórovací matice

- Příklad
BLOSUM62

| | C | S | T | P | A | G | N | D | E | Q | H | R | K | M | I | L | V | F | Y | W |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| C | 9 | -1 | -1 | -3 | 0 | -3 | -3 | -3 | -4 | -3 | -3 | -3 | -3 | -1 | -1 | -1 | -1 | -2 | -2 | -2 |
| S | -1 | 4 | 1 | -1 | 1 | 0 | 1 | 0 | 0 | 0 | -1 | -1 | 0 | -1 | -2 | -2 | -2 | -2 | -2 | -3 |
| T | -1 | 1 | 4 | 1 | -1 | 1 | 0 | 1 | 0 | 0 | 0 | -1 | 0 | -1 | -2 | -2 | -2 | -2 | -2 | -3 |
| P | -3 | -1 | 1 | 7 | -1 | -2 | -1 | -1 | -1 | -1 | -2 | -2 | -1 | -2 | -3 | -3 | -2 | -4 | -3 | -4 |
| A | 0 | 1 | -1 | -1 | 4 | 0 | -1 | -2 | -1 | -1 | -2 | -1 | -1 | -1 | -1 | -1 | -2 | -2 | -2 | -3 |
| G | -3 | 0 | 1 | -2 | 0 | 6 | -2 | -1 | -2 | -2 | -2 | -2 | -2 | -3 | -4 | -4 | 0 | -3 | -3 | -2 |
| N | -3 | 1 | 0 | -2 | -2 | 0 | 6 | 1 | 0 | 0 | -1 | 0 | 0 | -2 | -3 | -3 | -3 | -3 | -2 | -4 |
| D | -3 | 0 | 1 | -1 | -2 | -1 | 1 | 6 | 2 | 0 | -1 | -2 | -1 | -3 | -3 | -4 | -3 | -3 | -3 | -4 |
| E | -4 | 0 | 0 | -1 | -1 | -2 | 0 | 2 | 5 | 2 | 0 | 0 | 1 | -2 | -3 | -3 | -3 | -3 | -2 | -3 |
| Q | -3 | 0 | 0 | -1 | -1 | -2 | 0 | 0 | 2 | 5 | 0 | 1 | 1 | 0 | -3 | -2 | -2 | -3 | -1 | -2 |
| H | -3 | -1 | 0 | -2 | -2 | -2 | 1 | 1 | 0 | 0 | 8 | 0 | -1 | -2 | -3 | -3 | -2 | -1 | 2 | -2 |
| R | -3 | -1 | -1 | -2 | -1 | -2 | 0 | -2 | 0 | 1 | 0 | 5 | 2 | -1 | -3 | -2 | -3 | -3 | -2 | -3 |
| K | -3 | 0 | 0 | -1 | -1 | -2 | 0 | -1 | 1 | 1 | -1 | 2 | 5 | -1 | -3 | -2 | -3 | -3 | -2 | -3 |
| M | -1 | -1 | -1 | -2 | -1 | -3 | -2 | -3 | -2 | 0 | -2 | -1 | -1 | 5 | 1 | 2 | -2 | 0 | -1 | -1 |
| I | -1 | -2 | -2 | -3 | -1 | -4 | -3 | -3 | -3 | -3 | -3 | -3 | -3 | 1 | 4 | 2 | 1 | 0 | -1 | -3 |
| L | -1 | -2 | -2 | -3 | -1 | -4 | -3 | -4 | -3 | -2 | -3 | -2 | -2 | 2 | 2 | 4 | 3 | 0 | -1 | -2 |
| V | -1 | -2 | -2 | -2 | 0 | -3 | -3 | -3 | -2 | -2 | -3 | -3 | -2 | 1 | 3 | 1 | 4 | -1 | -1 | -3 |
| F | -2 | -2 | -2 | -4 | -2 | -3 | -3 | -3 | -3 | -3 | -1 | -3 | -3 | 0 | 0 | 0 | -1 | 6 | 3 | 1 |
| Y | -2 | -2 | -2 | -3 | -2 | -3 | -2 | -3 | -2 | -1 | 2 | -2 | -2 | -1 | -1 | -1 | -1 | 3 | 7 | 2 |
| W | -2 | -3 | -3 | -4 | -3 | -2 | -4 | -4 | -3 | -2 | -2 | -3 | -3 | -1 | -3 | -2 | -3 | 1 | 2 | 11 |

Dynamické programování

- Pokud máme k dispozici vhodnou skórovací matici, můžeme jednoduše aplikovat výpočet skóre na všechny možné kombinace zarovnání a vybrat z nich to nejlepší (s nejvyšším skóre)
- Bohužel počet všech kombinací bývá zpravidla velmi vysoký
- **Příklad:** porovnání sekvencí o délce 100 a 95 znaků vede cca na 55 miliónů možných zarovnání => pro delší sekvence se tento přístup stává nepoužitelný
- Řešení spočívá v aplikaci tzv. **dynamického programování** - tj. rozdělení problému na menší podproblémy (poprvé aplikovali na zarovnání sekvencí autoři **Needleman-Wunch 1970**)
- Tento přístup se stal základním kamenem algoritmů v bioinformatice

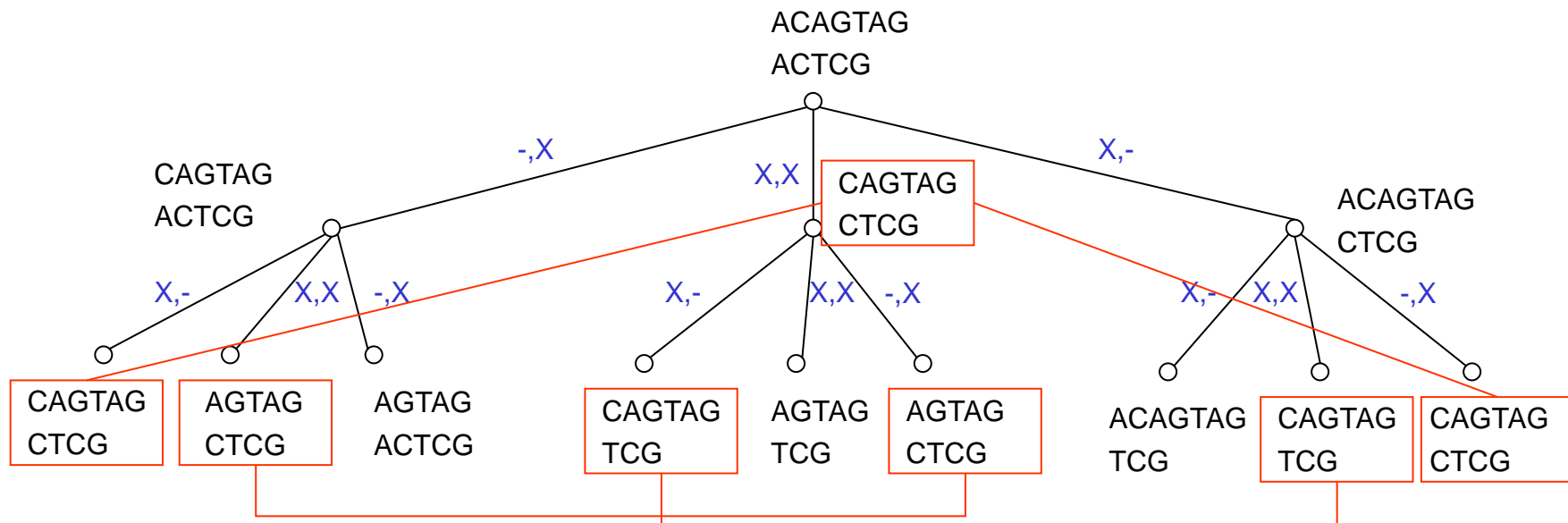
Dynamické programování

- **Příklad:** předpokládejme dvě sekvence **ACAGTAG** a **ACTCG**,
- Jako skórovací funkci zvolme pro jednoduchost:
 - 1 shoda,
 - 0 neshoda,
 - -1 vložení mezery
- **V první kroku máme tři možnosti:**
 1. nevložíme mezeru - zarovnáme první dva znaky
 2. vložíme mezeru do první sekvence
 3. vložíme mezeru do druhé sekvence
- Po prvním kroku máme tři rozpracované stavy. Celkové skóre ale závisí na tom, jak dopadne zarovnání zbývajících částí!
- Při dalším zpracování se opět každá z možností rozpadne na tři varianty atd.

| <i>První pozice</i> | <i>Skóre</i> | <i>Zbytek</i> |
|--------------------------------|---------------------|----------------------|
| A | +1 | CAGTAG |
| A | | CTCG |
| - | -1 | ACAGTAG |
| A | | CTCG |
| A | -1 | CAGTAG |
| - | | ACTCG |

Dynamické programování

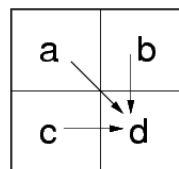
- Příklad stromu všech možností po dvou krocích:



- Určité kombinace se ve stromu opakují a není potřeba je dále počítat opakovaně ve všech větvích
- Lze dokázat, že všechny možné kombinace zarovnání lze reprezentovat pomocí 2D matice

Dynamické programování

- **Reprezentace algoritmu pomocí 2D tabulky, kde:**
 - řádky reprezentují jeden řetězec
 - sloupce reprezentují druhý řetězec
- **Vnitřní buňka se vypočte jako maximum ze tří možností:**
 - převzetí hodnoty v nalevo s přičtením penalizace za vložení mezery
 - převzetí hodnoty ze shora s přičtením penalizace za vložení mezery
 - převzetí hodnoty z levého horního rohu s přičtením skóre za shodu, nebo s penalizací za záměnu znaku
- **Hodnota v pravém spodním rohu reprezentuje skóre optimálního zarovnání**



$$d = \max \begin{cases} a+1 \text{ shoda} \\ a+0 \text{ neshoda} \\ c-1 \text{ odstranění} \\ b-1 \text{ vložení} \end{cases}$$

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

- **Poznámky:**
 - První řádek a sloupec se nastaví na postupně se zvyšující penalizační skóre za vložení mezery
 - Maximum vybere pouze nejperspektivnější cestu ze tří možných variant








Dynamické programování

- Průchodem tabulky z pravého spodního rohu směrem k levému hornímu rohu můžeme zpětně získat tvar optimálního zarovnání
- Při zpětném průchodu se zvažují tři možnosti:
 1. posun zpět ve směru diagonály
 2. posun nahoru (vlození mezery do horizontálního řetězce)
 3. posun doleva (vlození mezery do vertikálního řetězce)
- Vybere se ten, ze kterého byla vypočtena hodnota skóre dané buňky
- Ve skutečnosti může být více ekvivalentních možností

- Příklad:

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

- Výsledné zarovnání

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

Globální vs. lokální zarovnání

- Předchozí případ popisoval tzv. **globální zarovnání**, t.j. porovnání dvou sekvencí jako celek, kdy jakýkoliv výskyt mezery je penalizován
- V reálných případech ale potřebuje např. vyhledat výskyt krátké sekvence uvnitř dlouhého řetězce nebo dokonce celého genomu
- Potencionálně dlouhé mezery na začátku a konci kratšího řetězce nechceme penalizovat
- Tento způsob se nazývá **semiglobální zarovnání**

Semiglobální zarovnání

- Úpravy původního algoritmu globálního zarovnání
 - první řádek a sloupec jsou inicializovány na nuly (tímto ignoruje mezery na začátku řetězce)
 - v posledním řádku a sloupci se nepřičítá penalizace za vložení mezery
- Příklad:
 - hledání podřetězce **ACGT** v řetězci **AACACGTGTCT**

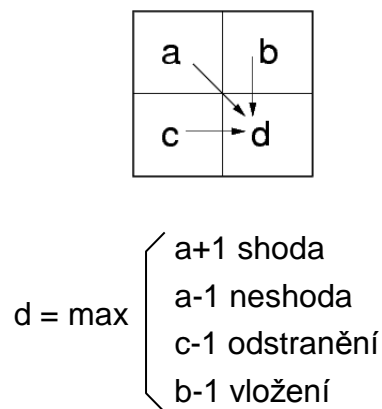
| | | A | A | C | A | C | G | T | G | T | C | T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 1 | 1 | 0 | 2 | 1 | 0 | 0 | 0 | 1 | 0 |
| G | 0 | 0 | 0 | 1 | 1 | 0 | 3 | 2 | 1 | 0 | 0 | 1 |
| T | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 4 | 4 | 4 | 4 | 4 |

Lokální zarovnání

- Velmi často ale potřebujeme hledat všechny výskyty daného podřetězce v celém genomu – všechna tzv. **lokální zarovnání**
- V tomto případě nebudou předchozí přístupy pracovat správně
- **Úpravy původního algoritmu globálního zarovnání:**
 - **Výpočet matice:**
 - první řádek a sloupec jsou inicializovány na nuly
 - i neshoda je penalizována (např. -1), aby při neshodě skóre klesalo
 - pokud je skóre v lib. pozici menší než nula, potom se nastaví na nulu
 - **Zpětný průchod:**
 - v celé tabulce se hledají maximální výskyty skóre a od těchto výskytu se spouští zpětný průchod (nikoliv pouze od pravého-spodního rohu)
- Tento algoritmus tvoří jeden ze základních kamenů bioinformatiky a byl poprvé publikován v roce 1981 autory **Smith-em** a **Waterman-em**

Lokální zarovnání

- **Příklad:**
 - hledání lokálního zarovnání řetězce **GCGATATA** v řetězci **AACCTATAGCT**



| | | A | A | C | C | T | A | T | A | G | C | T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| G | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 1 |
| G | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| A | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| T | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 1 | 0 | 0 | 1 |
| A | 0 | 1 | 1 | 0 | 0 | 0 | 2 | 0 | 3 | 2 | 1 | 0 |
| T | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 3 | 2 | 2 | 1 | 2 |
| A | 0 | 1 | 1 | 0 | 0 | 0 | 2 | 2 | 4 | 3 | 2 | 1 |

- **Poznámka:** Všimněte si, že při výpočtu se ignorují nejen mezery na začátku a konci řetězce, ale i neshody.

Statistická významnost zarovnání

- **Základní otázka:**
 - Uvažujme lokální zarovnání vstupní sekvence délky m se skórem S , jaká je pravděpodobnost, že nalezneme lokální výskyt náhodně vygenerované sekvence délky m se skórem $\geq S$?
- **Odpověď dává Karín-Altschulova rovnice**

$$E = kmNe^{-\lambda S}$$

- E – počet výskytů zarovnání náhodné sekvence délky m v sekvenci délky N se skórem $\geq S$
- k, λ - konstanty (obvykle závisí na skórovací matici)
- **Statisticky významný výsledek:**
 - **nukleotidy:** $E < 10^{-6}$ s alespoň **70%** identitou a výše
 - **proteiny:** $E < 10^{-3}$ s alespoň **25%** identitou a výše

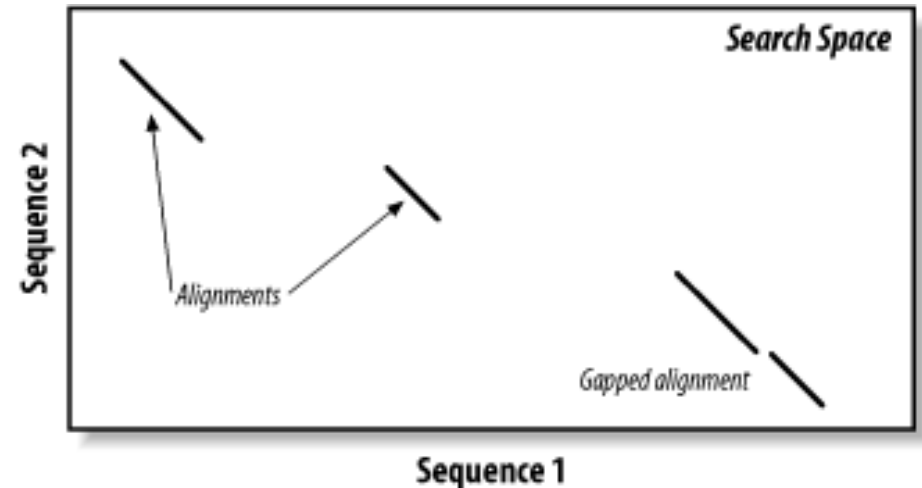
Prohledávání rozsáhlých databází

- V současných databázích je ohromné množství biologických dat
- Obvykle, pokud biologové potřebují např. ověřit zda našli nový gen, potom se snaží tuto sekvenci najít i v jiných genomech => prohledávání velkého množství dat
- Nevýhody metod NW a SW pro výpočet zarovnání:
 - kvadratická časová složitost
 - potřeba vymyslet rychlejší přístup založený na heuristice
 - NW a SW jsou použity pouze pro detailní analýzu vybraných kandidátních řešení
- Příklad:
 - Xeon 3GHz je schopen vypočítat 50M položek tabulky za sekundu

| N | Xeon 3 GHz |
|---------------|------------|
| 100 | 0,2 ms |
| 1.000 | 0,02 s |
| 10.000 | 2 s |
| 100.000 | 3 minuty |
| 1.000.000 | 5 hodin |
| 10.000.000 | 23 dnů |
| 100.000.000 | 6,5 roků |
| 1.000.000.000 | 650 roků |

BLAST

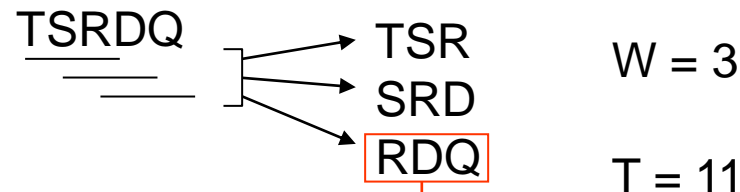
- S. Altschul (1990)
- **Cílem algoritmu je:**
 - nalézt všechna lokální zarovnání vstupní sekvence v rámci prohledávané sekvence
 - aniž by prohledával prostor všech možných zarovnání
- **Algoritmus je rozdělen do tří hlavních fází:**
 1. Osévání (Seeding)
 2. Rozšiřování (Extension)
 3. Ohodnocení (Evaluation)



- Prostor všech zarovnání lze reprezentovat jako matici algoritmu Smith-Waterman, kde zarovnané úseky sekvencí se vyskytují na diagonálách

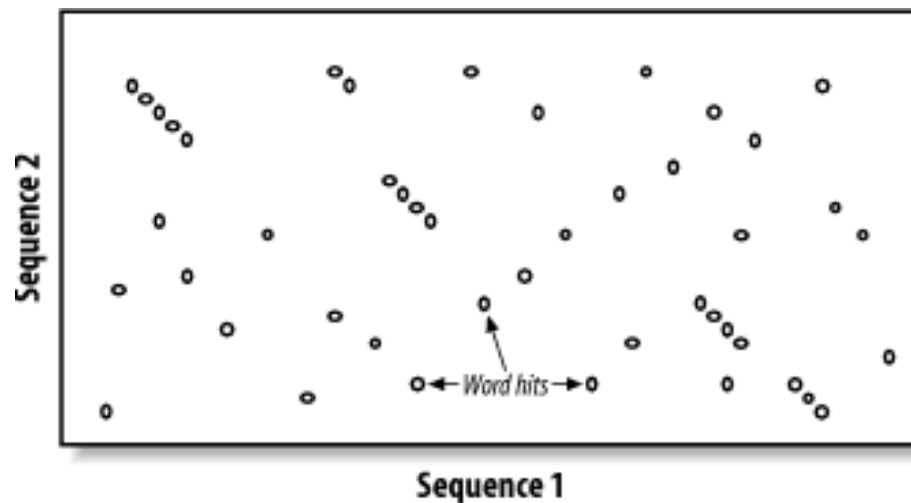
BLAST - Osívání

1. Hledaná sekvence je rozdělena do slov o velikosti W (např. $W=3$)
2. Rozdělení probíhá skrze okénko, které se posouvá po jednom znaku zleva doprava
3. Ke každému slovu se hledá množina alternativních slov (skrze substituce jednotlivých znaků slova). Ke každému alternativnímu slovu je vypočteno skóre podobnosti vzhledem k původnímu slovu (např. pomocí matice BLOSUM62) a pouze slova se skóre vyšším než je **zadaný práh** (threshold, např. $T=11$), jsou ponechány v tzv. tabulce sousednosti
4. Jednotlivé slova z tabulky sousednosti se hledají v sekvenci (v databázi)



Tabulka sousednosti

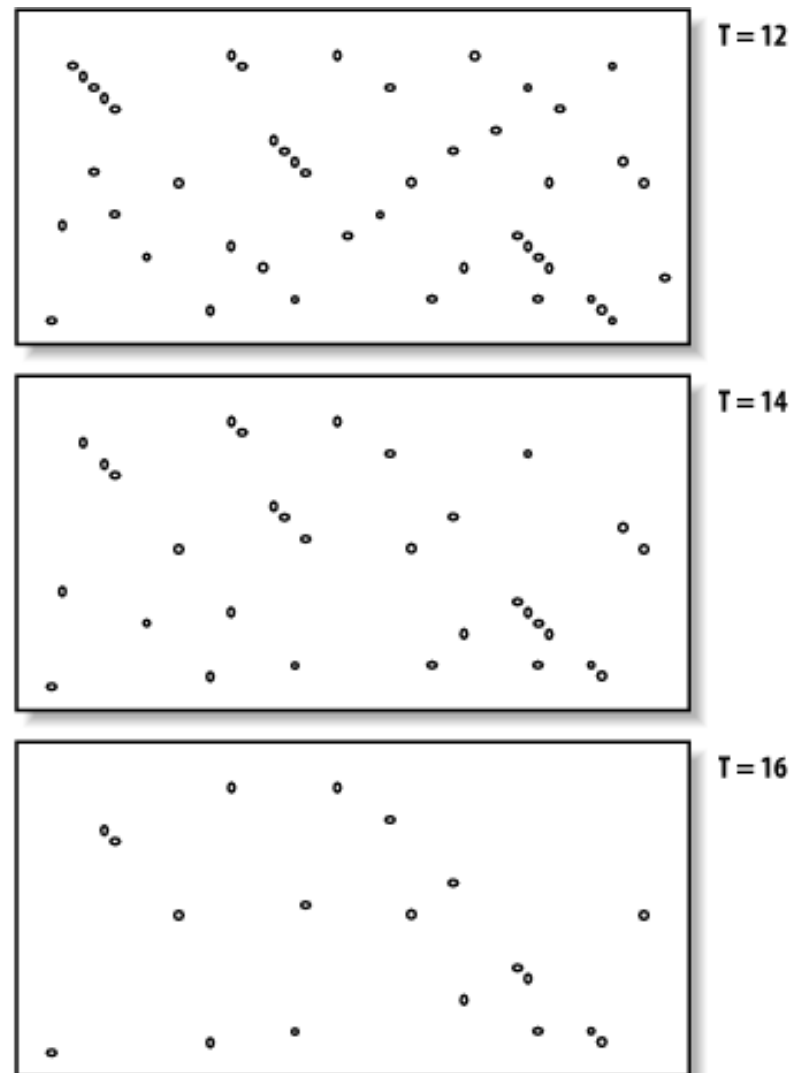
| | | | | |
|--------|--------|--------|--------|--------|
| RDQ 16 | QDQ 12 | EDQ 11 | RDN 11 | RDB 11 |
| RBQ 14 | REQ 12 | HDQ 11 | RDD 11 | ADQ 10 |
| RDA 14 | RDR 12 | ZDQ 11 | RDH 11 | MDQ 10 |
| KDQ 13 | RDK 12 | RNQ 11 | RDM 11 | SDQ 10 |
| RDE 13 | NDQ 11 | RZQ 11 | RDS 11 | TDQ 10 |



BLAST - Osívání

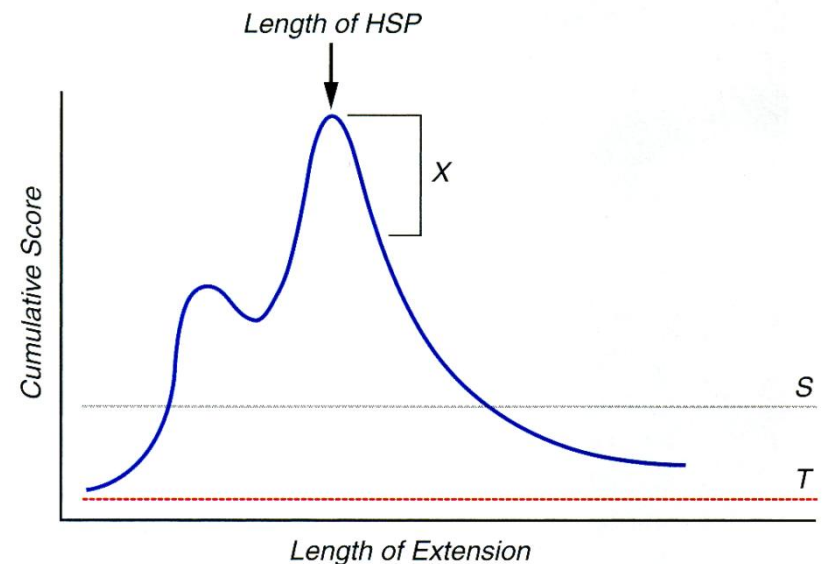
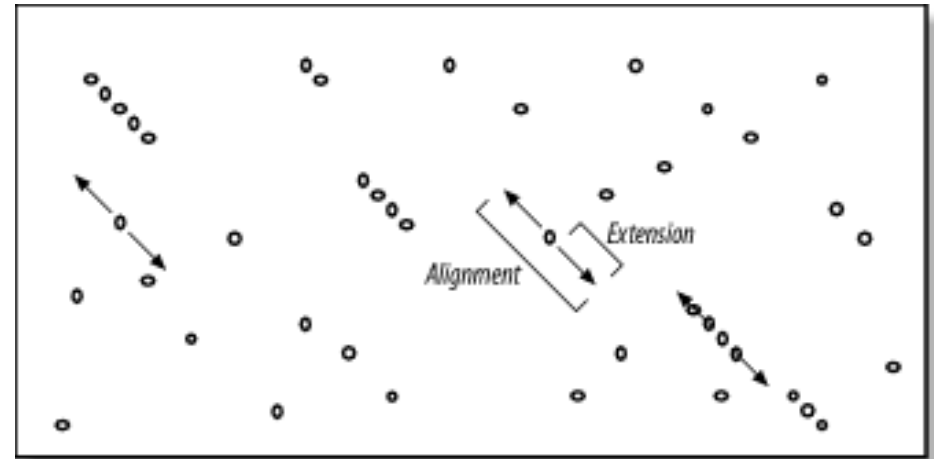
- **Volba parametru T a W**
ovlivňuje citlivost algoritmu vs.
rychlost výpočtu
- **Parametr T**
 - čím vyšší hodnota, tím hledá
algoritmus přesnější shody,
klesá počet výskytů, výpočet se
zrychluje
- **Parametr W**
 - menší hodnota vede na vyšší
počet výskytů, zvyšuje citlivost
výpočtu a zvyšuje dobu běhu

| | T | W |
|------------|-----------|----------|
| Nukleotidy | - | ≥ 7 |
| Proteiny | ≥ 10 | 2, 3 |



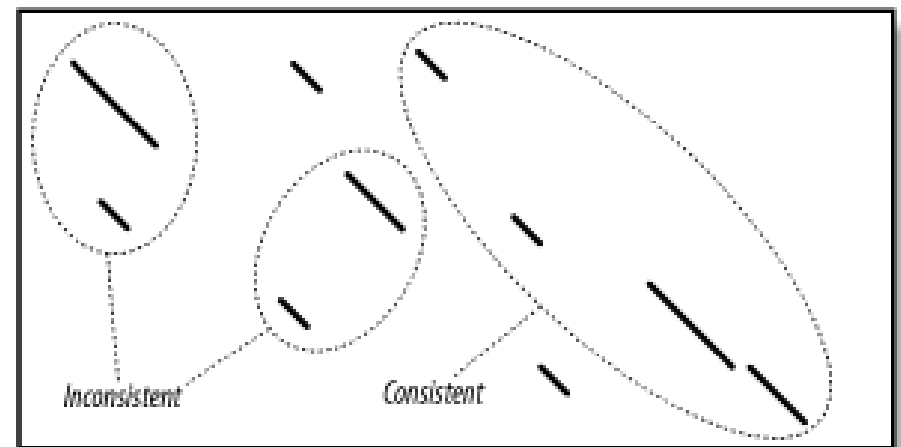
BLAST - Rozšiřování

- Pokud je některé slovo z tabulky nalezeno, snaží se algoritmus rozšiřovat nalezený úsek na obě strany
- Jakmile skóre vzroste nad minimální požadovanou hodnotu S bude sekvence reportována na výstupu algoritmu
- Pokud dále skóre klesne pod zadaný práh X , rozšiřování se ukončí a uloží se pozice maxima, než došlo ke klesání
- Výsledkem je seznam úseků s nejvyšším skóre (**High Scoring Segments**)



BLAST - Ohodnocení

- Ze seznamu všech nalezených segmentů se odstraní ty, které mají nízkou statistickou významnost (parametr E)
- Hledá se **skupina konzistentních úseků**, které na sebe navazují a spojují se ve větší celky
- Přístupy pro ohodnocení se liší např. při hledání genů složených z exonů a intronů (zatímco v genomu jsou exony i introny, v exprimovaném genu introny chybí)



BLAST - Varianty

| Program | Databáze | Dotaz | Použití |
|---------|----------------------------------|----------------------------------|---|
| BLASTN | Nukleotidy | Nukleotidy | Mapování oligonucleotidů, cDNAs a PCR produktů na genom; hledání opakujících se elementů; hledání společných sekvencí mezi organizmy; |
| BLASTP | Proteiny | Proteiny | Identifikace společných úseků mezi proteiny; analýza vzdáleností proteinů pro účely fylogenetiky |
| BLASTX | Proteiny | Nukleotidy přeložené do proteinů | Hledání genů kódující proteiny; určování, zda cDNA odpovídá známému proteinu |
| TBLASTN | Nukleotidy přeložené do proteinů | Proteiny | Identifikace transkriptů, pocházejících s více organismů, podobných danému proteinu; mapování proteinu na genomickou DNA |
| TBLASTX | Nukleotidy přeložené do proteinů | Nukleotidy přeložené do proteinů | Predikce genů v genomu nebo na úrovni transkripce; hledání netradičních genů nebo genů, které ještě nejsou v databázi |

FASTA

- David J. Lipman a William R. Pearson, 1988
- **Algoritmus rozdělen do čtyřech kroků**
 1. Nalezení identických segmentů
 2. Přepočet skóre skrze matici PAM nebo BLOSUM
 3. Spojování segmentů do delších úseků
 4. Výpočet optimálního skóre skrze SW algoritmus
- Existují varianty jak pro nukleotidy **FAST-N** tak i proteiny **FAST-P**

FASTA – Krok 1

- **Nalezení identických segmentů** mezi dotazem a databází s **využitím vyhledávacích tabulek**
 - Vstupní řetězec je rozsekán na slova (o délce **4-6** znaků pro nukleotidy, **1-2** znaků pro proteiny)
 - Z těchto slov je sestavena vyhledávací tabulka (**hash**) obsahující všechny pozice těchto slov v původní sekvenci
- **Příklad:**
 - Dotazovací sekvence **FAMLGFIKYLPGCM**

| A | C | D | E | F | G | H | I | K | L | M | N | P | Q | R | S | T | V | W | Y |
|---|----|---|---|---|----|---|---|---|----|----|---|----|---|---|---|---|---|---|---|
| 2 | 13 | | | 1 | 5 | | 7 | 8 | 4 | 3 | | 11 | | | | | | | 9 |
| | | | | 6 | 12 | | | | 10 | 14 | | | | | | | | | |

Tabulka1: Vyhledávací tabulka pro dotazovací sekvenci

FASTA - Krok 1

- Je sestavena druhá tabulka ze slov sekvence v databázi, kde:
 - každé slovo **x** z databáze se vyhledá v **tabulce 1**
 - pro všechny offsety slova **x** v **tabulce 1** se vypočítají nové offsety v tabulce 2 podle vztahu:
 - $\text{Offset} = \text{Seq1 Location} - \text{Seq2 Location}$
- Položky tabulky 2, které často vedle sebe obsahují stejný offset, ukazují na výskyt hledané sekvence nebo její části
- **Příklad:**
 - Sekvence v databázi **TGFIKYLPGACT** vzhledem k dotazovací sekvenci z předchozího snímku

| | | | | | | | | | | | |
|---|----|----|---|---|---|----|---|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| T | G | F | I | K | Y | L | P | G | A | C | T |
| | 3 | -2 | 3 | 3 | 3 | -3 | 3 | -4 | -8 | 2 | |
| | 10 | 3 | | | | 3 | | 3 | | | |

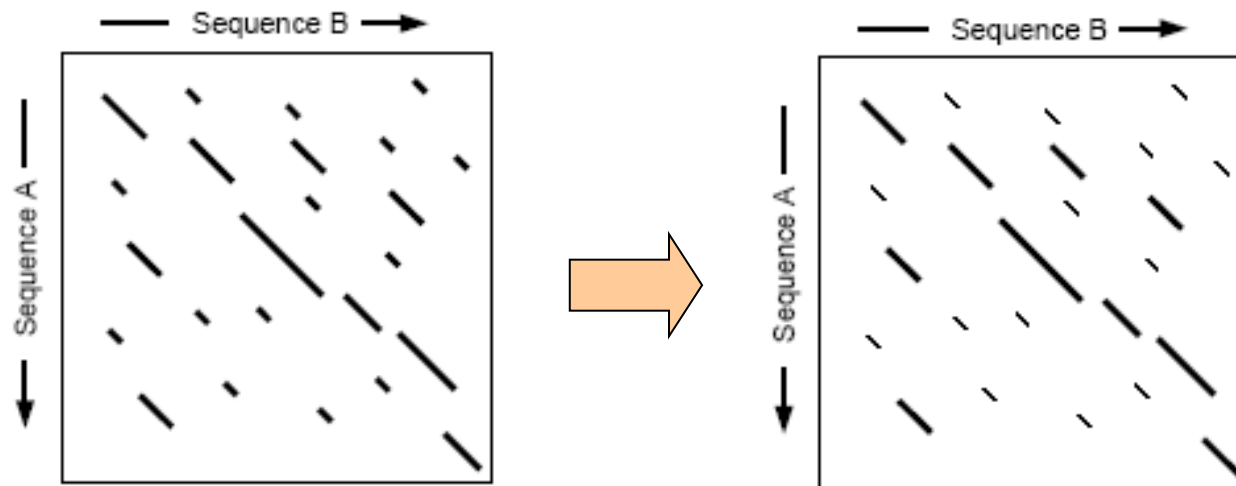
FAMLGFIKYLPGCM
| | | | | | |
TGFIKYLPGACT

Tabulka2: Přepočet offsetu v rámci sekvence v databázi

Nalezen výsky na offsetu 3

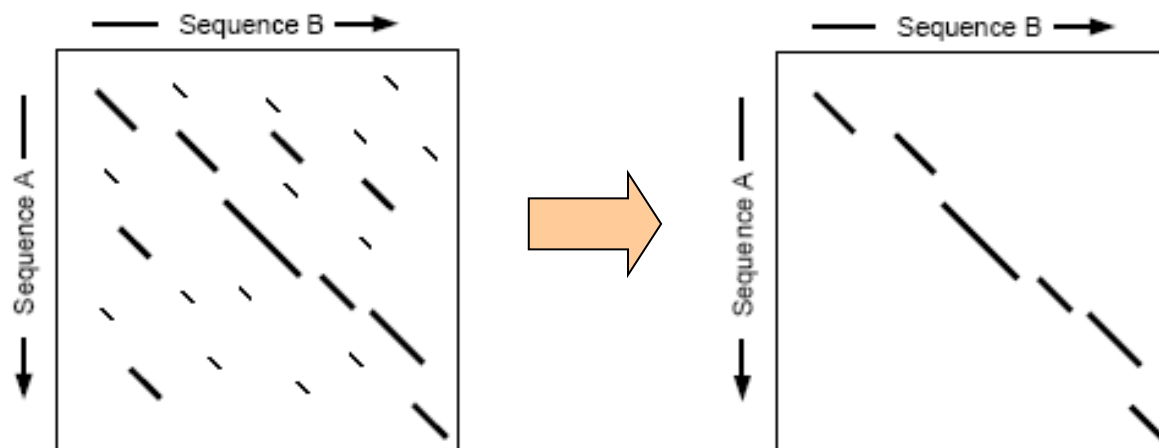
FASTA - Krok 2

- Přepočítání skóre nejdelších segmentů s použitím PAM nebo BLOSUM matice
 - Výběr těch, které mají nejvyšší skóre
 - Vypočtené skóre se označuje jako `init1`



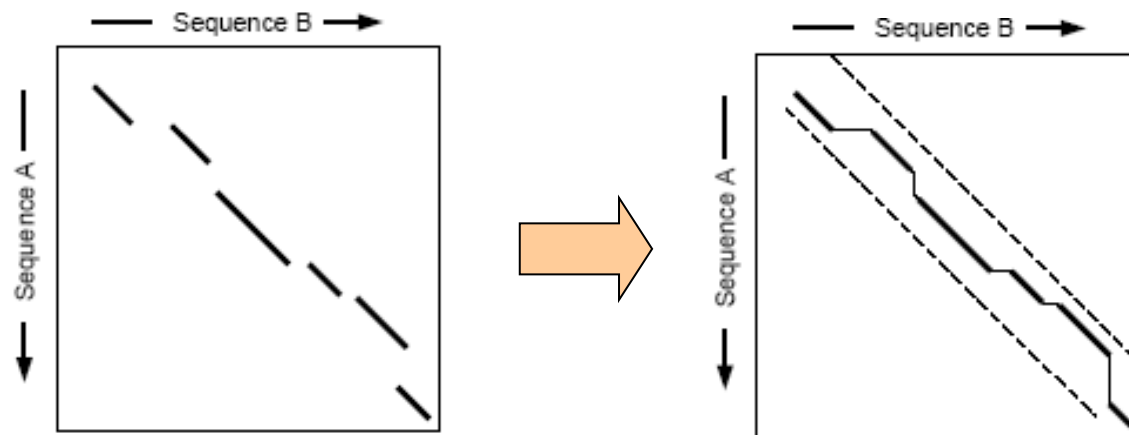
FASTA - Krok 3

- **Spojování několika segmentů dohromady**
 - Překrývající-se segmenty jsou eliminovány na základě skóre
 - Skóre spojení několika segmentů je vypočteno jako suma skóre jednotlivých segmentů mínus penalizace za spojování (vypočtené skóre se označuje jako `initn`)
 - Spojené segmenty jsou ohodnoceny a pouze ty s nejvyšším skóre jsou brány v úvahu pro další krok



FASTA - Krok 4

- Vybrané segmenty z kroku 3 jsou přepočítány pomocí algoritmu Smith-Waterman
 - vypočtené skóre se oznamuje jako `opt (optimized)`
 - aplikace SW pouze na perspektivní úseky je mnohem rychlejší, než pro celou matici vstupní sekvence a databázi



Porovnání BLAST a FASTA

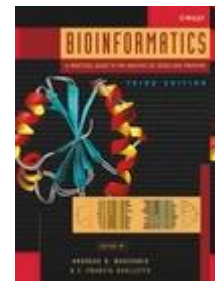
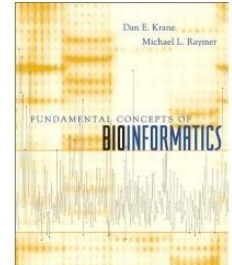
- Zatímco FASTA zpočátku toleruje pouze identické segmenty, BLAST uvažuje i segmenty obsahující záměny znaků
- Zatímco BLAST se snaží rozšiřovat nalezené výskyty slov, FASTA se snaží propojovat identické úseky
- Každá z metod má své výhody/nevýhody
- FASTA:
 - má větší citlivost prohledávání, výsledné zarovnání je přesnější
 - vyžaduje více výpočetního času než BLAST
 - volba krátkého slova vede na vyšší paměťovou složitost při konstrukci vyhledávacích tabulek

Shrnutí

- Při porovnávání biologických sekvencí je nezbytné brát v úvahu chyby v podobě vložení, odstranění nebo záměny znaku
- Kvalita výsledného zarovnání závisí z velké části na vybrané skórovací funkci
- Optimální výpočet zarovnání nabízí algoritmy:
 - Needleman-Wunch – globální zarovnání
 - Smith-Waterman – lokální zarovnání
- Optimální algoritmy mají kvadratickou časovou složitost – nepoužitelné pro prohledávání rozsáhlých databází
- Nutnost použití heuristik
 - BLAST – rozšiřování nalezených výsledků
 - FASTA – využití tabulky indexů

Literatura

- Dan K. Krane, Michael L. Raymer: ***Fundamental Concepts of Bioinformatics***, ISBN: 0-8053-4633-3, Benjamin Cummings 2003.
- Andreas D. Baxevanis, B. F. Francis Ouellette: Bioinformatics: ***A Practical Guide to the Analysis of Genes and Proteins***, ISBN: 0-471-47878-4, Wiley-Interscience, 2005.



Konec

Děkuji za pozornost