Curs 6.

Tehnici algoritmice utilizate în bioinformatică (continuare). Algoritmi pentru alinierea globală și locală a secvențelor.

Biblio: cap. 6 din “An introduction to Bioinformatics algorithms”, N.Jones, P. Pevzner

Tehnici algoritmice utilizate in bioinformatica

- Tehnica căutarii exhaustive (metoda forței brute)
  - Sistematizarea căutarii: backtracking
  - Limitarea căutarii: branch and bound
- Tehnica căutarii local optimale (greedy)

(Exemplu: identificarea șabloanelor comune în seturi de secvențe)

- Programare dinamică
  (Exemplu: alinierea secvențelor)

Tipuri de probleme rezolvate prin programare dinamica

- Probleme clasice din algoritmică:
  - Determinarea celui mai lung subșir comun a două șiruri
  - Determinarea distanței de editare
  - Problema turistului

- Aplicabilitate în bioinformatică:
  - Analiza similarității între secvențe
  - Alinierea secvențelor

Similaritatea dintre secvente

Context:

- Identificarea similarității dintre o secvență ADN cu rol necunoscut și o genă cu funcționalitate cunoscută permite efectuarea de inferențe asupra rolului primeia. Determinarea unui scor de similaritate între două gene exprimă cât de apropiate ar putea fi funcțiile celor două.

- In 1984 au fost descoperite similarități între o genă cauzatoare de cancer și o genă ce codifică o proteină care intervine în procesul normal de creștere.
- In 1989 biologii au descoperit similarități intre gena fibrozei cistice (boala caracterizată prin secreție anormală de mucus în sistemul respirator) și cea a unor proteine corelate cu adenin-trifosfatul (ATP binding proteins).

- Obs: Calculul scorului de similaritate pe baza matricii profil (curs 5) presupune că mutațiile conduc la înlocuirea unui element cu un alt element (se ignoră eliminările/inserțiile de elemente). Calculul scorului pe baza alinierii secvențelor ține cont și de aceste aspecte.

Similaritatea dintre secvente

Cum se poate măsura similaritatea / disimilaritatea între două secvențe care nu au neapărat aceeași lungime ?

- "numărând" elementele comune
  - Cel mai lung subșir comun – utilitate limitată (nu penalizează diferit nepotrivirile și eliminarile/inserțiile de elemente)
- "numărând" elementele diferite
  - Distanța Hamming - utilitate limitată (secvențele trebuie să aibă aceeași lungime și nu se ține cont de eventualele mutații)
  - Distanța de editare – utilitate mare în alinierea secvențelor

Similaritatea dintre secvente

Determinarea celui mai lung subșir comun a două șiruri:

Enunțul problemei:

Fiind date două secvențe $a_1,\ldots, a_n$ și $b_1,\ldots,b_m$ să se găsească un subșir $c_1,\ldots c_k$ de elemente nu neapărat consecutive din șirurile inițiale care satisface:

- există $i_1,\ldots,i_k$ si $j_1,\ldots,j_k$ astfel încât

  $$c_1=a_{i1}=b_{j1},\ c_2=a_{i2}=b_{j2},\ \ldots\ ,\ c_k=a_{ik}=b_{jk}$$

- k este maxim

Remarcă: două secvențe ADN pot fi considerate cu atât mai similare cu cât conțin un subșir comun mai lung

Cel mai lung subsir comun

Exemplu:

a: 2 1 4 3 2
b: 1 3 4 2

Subșiruri comune:

1, 3
1, 2
4, 2
1, 3, 2
1, 4, 2

Exemplu:

sir1: ATCTGATG
sir2: TGCATAC

Subșiruri comune:

ATA
GAT
TCTA
TGAT ….

Programare dinamica - specific

Etapele principale în aplicarea programării dinamice:

- Analiza structurii unei soluții: verificarea proprietății de substructură optimă
- Identificarea relației de recurență
- Dezvoltarea relației de recurență
- Construirea soluției pe baza informațiilor completate la etapa anterioară

Programare dinamica - specific

- Se aplică problemelor de optimizare care pot fi descompuse în subprobleme și care au proprietatea de substructură optimă:

  “o soluție optimă a problemei conține soluții optime ale subproblemelor”

- Se caracterizează prin faptul că fiecare subproblemă este rezolvată o singură dată iar soluția ei este stocată și utilizată ori de câte ori este cazul

- Elementul specific este deducerea unei relații de recurență care exprimă legătura dintre soluția optimă a problemei și soluțiile subproblemelor

Cel mai lung subsir comun

1. Analiza structurii unei soluții optime

Fie P(i,j) problema generică a determinării celui mai lung subșir comun al șirurilor parțiale a(1:i) si b(1:j)

Obs: P(n,m) reprezintă problema inițială

2. Determinarea relației de recurență. Fie L(i,j) lungimea unei soluții optime pentru P(i,j)

$$L(i,j)=\begin{cases} 0 & \text{dacă } i=0 \text{ sau } j=0 \\ 1+L(i-1,j-1) & \text{dacă } a(i)=b(j) \\ \max\{L(i-1,j),L(i,j-1)\} & \text{altfel} \end{cases}$$

Cel mai lung subsir comun

Exemplu:

a: C A T G C

b: A G T C

$$L(i,j)=\begin{cases} 0 & \text{dacă } i=0 \text{ sau } j=0 \\ 1+L(i-1,j-1) & \text{dacă } a(i)=b(j) \\ \max\{L(i-1,j),L(i,j-1)\} & \text{altfel} \end{cases}$$

			A	G	T	C
		0	1	2	3	4
	0	0	0	0	0	0
C	1	0	0	0	0	1
A	2	0	1	1	1	1
T	3	0	1	1	2	2
G	4	0	1	2	2	2
C	5	0	1	2	2	3

Cel mai lung subsir comun

Exemplu:

a: C A T G C

b: A G T C

$$L(i,j)=\begin{cases} 0 & \text{dacă } i=0 \text{ sau } j=0 \\ 1+L(i-1,j-1) & \text{dacă } a[i]=b[j] \\ \max\{L(i-1,j),L(i,j-1)\} & \text{altfel} \end{cases}$$

Dezvoltarea relației de recurență

```
compute(a[1:n],b[1:m])
for i=0:n
      L[i,0]=0
end
for j=1:m
     L[0,j]=0
end
for i=1:n DO
  for j=1:m DO
     if a[i]==b[j] then L[i,j]=L[i-1,j-1]+1
                else L[i,j]=max(L[i-1,j],L[i,j-1])
     end
  end
end
```

Cel mai lung subsir comun

Exemplu (varianta recursivă):

```
Construct(i,j)
if i>=1 and j>=1 then
    if a[i]==b[j]
    then
        construct(i-1,j-1)
        k:=k+1
        c[k]=a[i]
   else
        if L[i-1,j]>L[i,j-1]
            then  construct (i-1,j)
            else  construct (i,j-1)
        end
   end
end
```

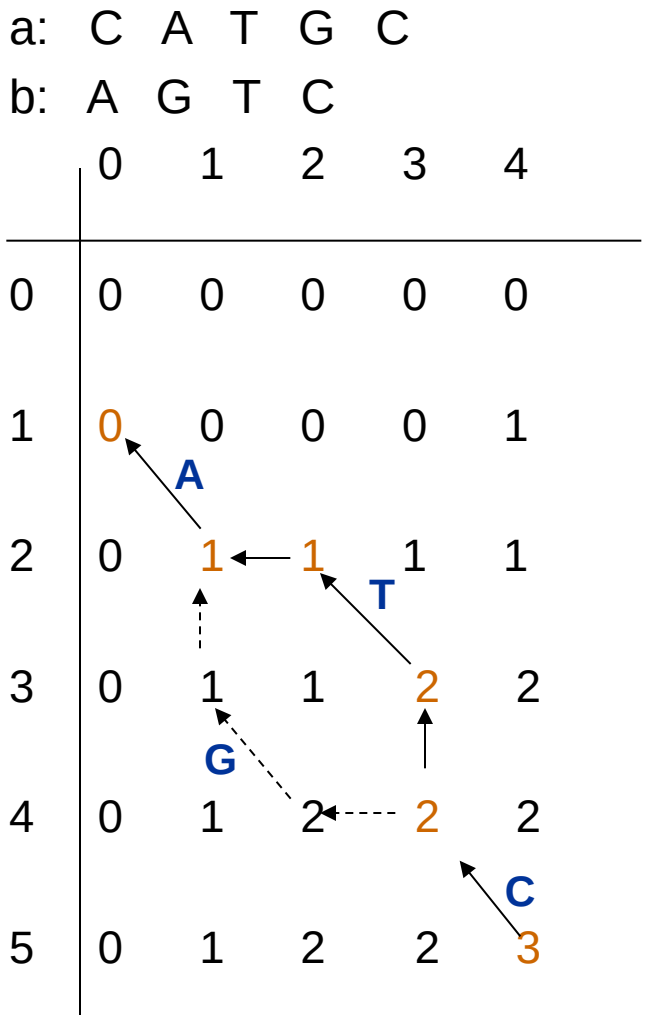
Obs:

c și k sunt variabile globale

k e inițializat cu 0

Apel:

Construct(n,m)



Distanta Hamming

- Distanța Hamming = numărul de poziții în care două șiruri având aceeași lungime diferă

Exemplu:

S1: ATATATA

S2: TATATAT

H(S1,S2)=7 (valoarea maximă)

Obs: o simplă deplasare a șirului 2 cu o poziție la dreapta conduce la:

ATATATA-

- TATATAT adica H(S1,S2)=2

- Aceste șiruri diferă în doar două poziții. In bioinformatică secvențele ADN nu sunt de regulă aliniate
- Distanța Hamming nu e adecvată pentru a măsura disimilaritatea dintre secvențe ADN

Distanta de editare

- Distanța de editare (distanța Levenshtein) = numărul minim de operații de editare simple (înlocuire caracter, inserare caracter, ștergere caracter) necesare pentru a transforma un șir in alt șir

Exemplu:

S1: ATATATA

S2: TATATAT

L(S1,S2)=?

Distanta de editare

- Distanța de editare (distanța Levenshtein) = numărul minim de operații de editare simple (înlocuire caracter, inserare caracter, ștergere caracter) necesare pentru a transforma un șir in alt șir

Exemplu:

S1: ATATATA

S2: TATATAT

L(S1,S2)=2

- o ștergere: se șterge primul caracter
- o inserare: se inserează T pe ultima poziție

ATATATA ->TATATA->TATATAT

Distanta de editare

Problema: Pentru două șiruri a[1..n] și b[1..m] se caută numărul minim de operații de editare prin care șirul a[1..n] poate fi transformat în șirul b[1..m]

1. Analiza structurii unei soluții optime

Fie P(i,j) problema generică a determinării distanței de editare dintre a(1:i) si b(1:j); Solutia lui P(i,j) depinde de soluțiile lui P(i-1,j-1), P(i-1,j), P(i,j-1)

2. Fie L[i,j] soluția optima a problemei P(i,j)

$$L(i,j)=\begin{cases} i & \text{dacă } j=0 \\ j & \text{dacă } i=0 \\ L(i-1,j-1) & \text{dacă } a(i)=b(j) \\ \min\{L(i-1,j-1)+1, L(i,j-1)+1, L(i-1,j)+1\} & \text{dacă } a(i) \neq b(j) \end{cases}$$

Inlocuire	Inserare	Eliminare
a[i] cu b[j]	b[j]	a[i]

Distanta de editare

- Exemplu:

		0	1	2	3	4
			T	G	C	A
0		**0**	1	2	3	4
1	A	**1**	1	2	3	3
2	T	2	**1**	2	3	4
3	C	3	2	**2**	2	3
4	T	4	3	3	**3**	3
5	G	5	4	3	**4**	3
6	A	6	5	4	4	**4**

- Secvența transformări:

ATCTGA

eliminare G

ATCTA

înlocuire T cu C

ATCCA

înlocuire C cu G

ATGCA

eliminare A

TGCA

↑ eliminare ← inserare ↖ Inlocuire sau nemodificat

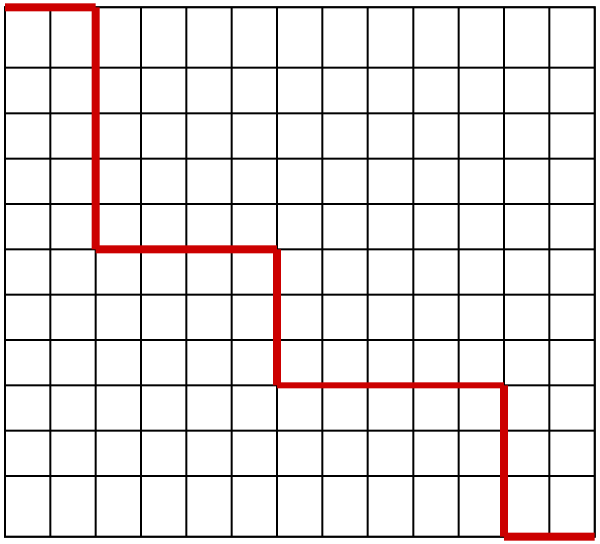
Distanta de editare

- Identificare transformări (varianta iterativă)

```
i=n; j=m;
while i>0 and j>0 and L[i,j]>0
  if a[i]==b[j] then i=i-1; j=j-1
  else
    if L[i,j]==L[i-1,j-1]+1 then “inlocuire a[i] cu b[j]”; i=i-1;j=j-1;  end
    else  if L[i,j]==L[i-1,j]+1 then “stergere a[i]”; i=i-1;
                                  else “inserare b[j]”; j=j-1;
          end
    end
end
if i==0 then while L[i,j]>0 DO “inserare b[j]”; j=j-1;   end
if j==0 then while L[i,j]>0 DO “stergere a[i]”; i=i-1;   end
```

Problema turistului

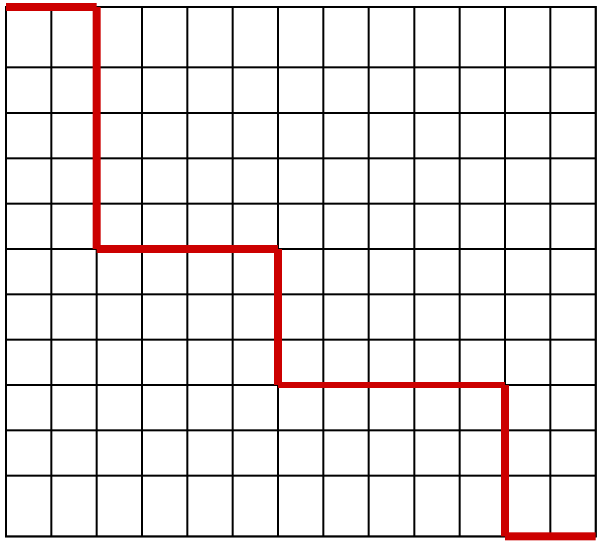
- Se consideră o grilă mxn (harta străzilor din Manhattan). Fiecare muchie din grilă are asociat un scor. Se dorește traversarea grilei pornind de la colțul din stânga sus către colțul din dreapta jos astfel încât să se realizeze deplasare doar în jos sau către dreapta iar scorul traversării să fie maxim



- Scoruri: fiecărui nod al grilei i se asociază un scor corespunzator deplasării la dreapta și în jos
- Scorurile se pot stoca în două matrici D (pt deplasare la dreapta) și J (pt deplasare în jos)
- D(i,j) = scor deplasare de la (i,j-1) la (i,j)
- J(i,j) = scor deplasare de la (i-1,j) la (i,j)

Problema turistului

- Se consideră o grilă mxn (harta străzilor din Manhattan). Fiecare muchie din grilă are asociat un scor. Se dorește traversarea grilei pornind de la colțul din stânga sus către colțul din dreapta jos astfel încât să se realizeze deplasare doar în jos sau către dreapta iar scorul traversării să fie maxim

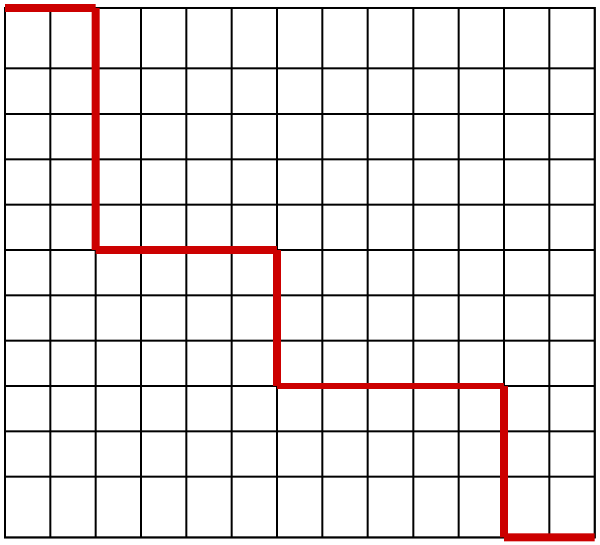


$$S(i,j)=\begin{cases} 0 & \text{dacă } i=1,\ j=1 \\ S(i-1,j)+J(i,j), & \text{dacă } j=1 \\ S(i,j-1)+D(i,j), & \text{dacă } i=1 \\ \max\{S(i-1,j)+J(i,j), & \\ \quad S(i,j-1)+D(i,j)\} & \text{altfel} \end{cases}$$

P(i,j) conține direcția ultimului pas făcut pt. a ajunge în (i,j) (jos sau dreapta)

Problema turistului

```
traseu( i, j)
   if i >1 or  j >1then
       if P(i,j)== "jos" then traseu(i-1,j); Write ("jos“)
                         else traseu(i, j-1); Write ("dreapta“)
      endif
endif
```



$$S(i,j)=\begin{cases} 0 \quad \text{dacă } i=1,\ j=1 \\ S(i-1,j)+J(i,j), \quad \text{dacă } j=1 \\ S(i,j-1)+D(i,1), \text{ dacă } i=1 \\ \max\{S(i-1,j)+J(i,j), \\ \qquad S(i,j-1)+D(i,j)\} \quad \text{altfel} \end{cases}$$

P(i,j) conține direcția ultimului pas făcut pt. a ajunge în (i,j) (jos sau dreapta)

Problema turistului

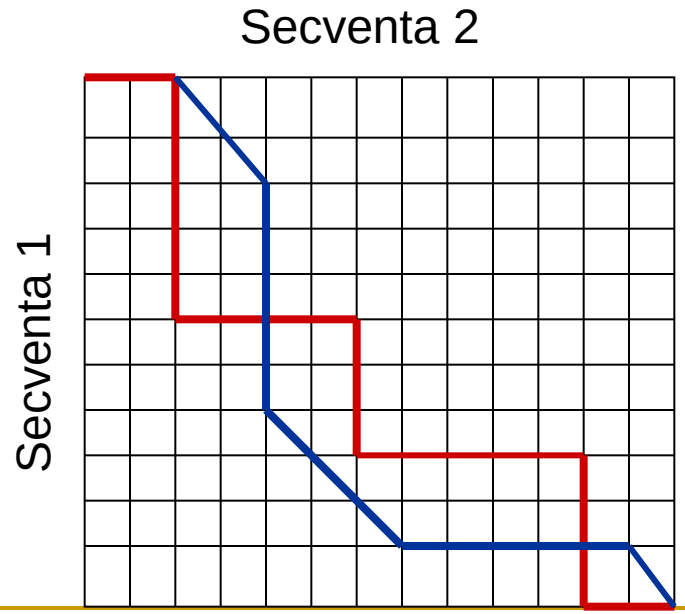
Obs: problema poate fi extinsă pentru cazul deplasărilor pe diagonală (dreapta jos) – costurile se reţin în matricea O (O(i,j) conţine costul deplasării din (i-1,j-1) in (i,j))

Interpretare deplasări în contextul problemei alinierii:

jos: inserare spaţii în secvenţa 2

dreapta: inserare spaţii în secvenţa 1

diagonală: potrivire sau nepotrivire



$$S(i,j)=\begin{cases} 0 \quad \text{dacă } i=1,\ j=1 \\ S(i-1,j)+J(i,j), \quad \text{dacă } j=1 \\ S(i,j-1)+D(i,j), \quad \text{dacă } i=1 \\ \max\{S(i-1,j)+J(i,j), \\ \qquad S(i,j-1)+D(i,j) \\ \qquad S(i-1,j-1)+O(i,j)\} \end{cases}$$

Alinierea globala a secventelor

- Alinierea globală a două secvențe = inserarea de spații (gap-uri) în cadrul secvențelor astfel încât acestea să aibă aceeași lungime și "corespondența" dintre ele să fie cât mai mare

Exemple de aliniere a secvenţelor ATCTGA și TGCA:

ATCTGA
– TGC–A

ATCTGA
–T– GCA

ATCTG – A
– – –TGC A

- Corespondența poate fi măsurata prin numărul de operații necesare pentru a transforma o secvența în cealaltă (ca la distanța de editare) sau printr-un scor asociat alinierii
- Scorul se calculează pe baza unei matrici de scor ce poate asocia costuri distincte diferitelor operații; scorul depinde de valorile elementelor și nu de poziția lor în cadrul secvențelor.

Alinierea globala a secventelor

- Asociere de scoruri diferențiate:
  - Potrivire (a[i]=b[j]): c0
  - Nepotrivire (a[i]!=b[j]): -c1
  - Inserare sau ștergere: -c2
- Matrice de scor sau de similaritate (M)

	A	C	G	T	-
A	c0	-c1	-c1	-c1	-c2
C	-c1	c0	-c1	-c1	-c2
G	-c1	-c1	c0	-c1	-c2
T	-c1	-c1	-c1	c0	-c2
-	-c2	-c2	-c2	-c2	

- Scorul alinierii:

$$S(i,j)=\begin{cases} M(-,b[1])+\ldots+M(-,b[j]),\ i=0 \\ M(a[1],-)+\ldots+M(a[i],-),\ j=0 \\ \max\{S(i-1,j-1)+M(a[i],b[j]),\\ \quad S(i,j-1)+M(-,b[j]),\\ \quad S(i-1,j)+M(a[i],-)\},\ i>0,j>0 \end{cases}$$

Algoritmul de construire a alinierii este similar celui pentru determinarea succesiunii de transformări dintr-o secvență in alta

Obs. In cazul alinierii secvențelor de aminoacizi matricea de scor are dimensiunea 21*21

Alinierea globala a secventelor

- Exemple de valori ale scorurilor:
  - Penalizare identică pentru toate nepotrivirile (varianta utilizată în BLAST)
  - Penalizare diferențiată în funcție de tipul nucleotidei (înlocuirea unei nucleotide cu una de același tip se penalizează mai puțin decât înlocuirea cu una de tip diferit)
    - Purina -> Purina,
      Pirimidina->Pirimidina
      penalizare:-1
    - Purina -> Pirimidina (si invers)
      penalizare: -5

Purine= {A,G}

Pirimidine= {C,T}

	A	C	G	T	-
A	1	-5	-5	-5	-c2
C	-5	1	-5	-5	-c2
G	-5	-5	1	-5	-c2
T	-5	-5	-5	1	-c2
-	-c2	-c2	-c2	-c2	

	A	C	G	T	-
A	1	-5	-1	-5	-c2
C	-5	1	-5	-1	-c2
G	-1	-5	1	-5	-c2
T	-5	-1	-5	1	-c2
-	-c2	-c2	-c2	-c2	

Alinierea globala a secventelor

- Scorul total al alinierii este:

S(a,b)=c0*#potriviri-c1*#nepotriviri-c2*#gap

- Algoritmul Needleman-Wunsch (1970) – permite alinierea globală a două secvențe
  - Se completează matricea S asociată alinierii pe baza matricii de scor M și a relației de recurență
  - Elementul din colțul din dreapta jos va indica scorul alinierii optime
  - Se construiește alinierea pornind de la elementul din dreapta jos urmând calea inversă folosită la construirea matricii

Alinierea globala a secventelor

- Exemplu

a = ATCTGA

b = TGCA

Matrice scor (c0=1, c1=1, c2=2)

	A	C	G	T	-
A	1	-1	-1	-1	-2
C	-1	1	-1	-1	-2
G	-1	-1	1	-1	-2
T	-1	-1	-1	1	-2
-	-2	-2	-2	-2	

- Matrice cu scoruri calculate pe baza relației de recurență

		T	G	C	A
	0	-2	-4	-6	-8
A	-2	-1	-3	-5	-5
T	-4	-1	-2	-4	-6
C	-6	-3	-2	-1	-3
T	-8	-5	-4	-3	-2
G	-10	-7	-4	-5	-4
A	-12	-9	-6	-5	-4

Aliniere: Scor: -4

ATCTGA

- T -GCA

Algoritmul Needleman-Wunsch

```
AliniereGlobalaNeedlemanWunsch (S[1:n,1:m],M[1:5,1:5])
i=n; j=m;
sa=c();  sb=c();    //secventele sa si sb se initializeaza ca liste vide
while i>0 and j>0
   if S[i,j]==S[i-1,j-1]+M(a[i],b[j])
       then sa=c(a[i], sa);  sb=c(b[j], sb);  i=i-1; j=j-1
                                        //se extind listele prin concatenare
   else
      if S[i,j]=S[i-1,j]+M(a[i],"-") then sa=c(a(i), sa); sb=c("-", sb);  i=i-1;
               else  sa=c("-", sa); sb=c(b[j], sb); j=j-1;
      end
   end
end
if i==0 then while j>0  sa=c("-", sa); sb=c(b[j], sb); j=j-1 end end
if j==0 then while i>0  sa=c(a[i], sa); sb=c("-", sb); i=i-1 end end
```

Algoritmul Needleman-Wunsch

- Este utilizat pentru alinierea globala a secvențelor (rezultatul este reprezentat de două secvențe având aceeași lungime obținute prin inserarea de “gap”-uri)

- In secvențele biologice este mai plauzibil ca mutațiile să afecteze mai degrabă poziții succesive în loc de poziții izolate; din acest motiv “gap”-urile se penalizează diferențiat: o valoare mai mare asociată primului gap dintr-o secvență și valori mai mici pentru gap-urile următoare

- Tehnica alinierii poate fi extinsă în cazul secvențelor de aminoacizi, însă sunt necesare matrici de scor specifice (detalii în cursul următor)
- Ilustrare algoritm
  - implementare Java [http://baba.sourceforge.net/]

Penalizarea gap-urilor

Motivație:

- Mutațiile sunt determinate de erori în procesul de replicare a ADN-ului; adeseori aceste mutații nu afecteaza o singură poziție ci mai multe poziții succesive astfel că alinierile conțin nu doar spații individuale ci scurte secvențe de spații (numite gap-uri)

- Exemplu:

Aliniere 1	Aliniere 2
ATA- -GC	ATAG-GC
ATATTGC	AT- GTGC

Obs: prima aliniere este mai plauzibilă

Penalizarea gap-urilor

- Introducerea unui gap în oricare dintre secvențe este penalizată prin adăugarea unui scor negativ la scorul total al alinierii.

- Există două variante principale de penalizare
  - Liniară: fiecare gap introdus (indiferent de poziție) este penalizat cu aceeași valoare, d; o succesiune de k gap-uri va fi penalizată cu k*d
  - Afină: primul gap dintr-o secvență este penalizat cu o valoare diferită de cea aferentă gap-urilor următoare (acestea sunt penalizate cu o valoare mai mică). In cazul unei secvențe de k gap-uri penalizarea este: d+(k-1)*e
    - d = penalizare corespunzătoare inițierii secvenței de gap-uri
    - e = penalizare corespunzătoare extinderii secvenței de gap-uri (e<d)

Penalizarea gap-urilor

Implementarea penalizării liniare:

Este suficient să se existe în matricea de scor (M) o linie și o coloană având toate valorile egale cu -d

- Dacă se consideră că nucleotidele au probabilități diferite de eliminare/inserție atunci valorile de pe ultima linie/coloană pot fi diferite

Matricea de scor (M)

	A	C	G	T	-
A	1	-c1	-c1	–c1	–d
C	–c1	1	-c1	-c1	–d
G	–c1	–c1	1	-c1	–d
T	–c1	–c1	-c1	1	–d
-	–d	–d	–d	–d	

Calculul scorului alinierii

$$S(i,j)=\begin{cases} -j*d, & i=0 \\ -i*d, & j=0 \\ \max\{S(i-1,j-1)+M(a[i],b[j]),\ S(i,j-1)-d,\ S(i-1,j)-d\}, & i>0,j>0 \end{cases}$$

Penalizarea gap-urilor

- Implementarea penalizării afine (pt.alinierea șirurilor a[1:n] și b[1:m])

$$S(i,j)=\begin{cases} -d-(j-1)*e & i=0 \\ -d-(i-1)*e & j=0 \\ \max\{S(i-1,j-1)+M(a(i),b(j)), & \\ \quad \max\{S(i-k,j)-d-(k-1)*e, k=\overline{1,i}\}, & i>0,\ j>0 \\ \quad \max\{S(i,j-k)-d-(k-1)*e, k=\overline{1,j}\}\} & \end{cases}$$

Penalizarea gap-urilor

- Implementarea mai eficientă a penalizării afine: utilizarea a 3 matrici:

S(i,j): scorul unei alinieri în care pe poziția i și pe poziția j nu sunt gap-uri

G_b(i,j): scorul unei alinieri în care în secvența b (a doua) pe ultima poziție (j) se află un gap

G_a(i,j): scorul unei alinieri în care în secvența a (prima) pe ultima poziție (i) se află un gap

Relația de recurență:

$$S(i,j) = \max\{S(i-1,j-1), G_b(i-1,j-1), G_a(i-1,j-1)\} + M(a(i), b(j))$$

$$G_b(i,j) = \max\{S(i-1,j) - d, G_a(i-1,j) - d, G_b(i-1,j) - e\}$$

$$G_a(i,j) = \max\{S(i,j-1) - d, G_a(i,j-1) - e, G_b(i,j-1) - d\}$$

Penalizarea gap-urilor

Implementarea mai eficientă a penalizării afine: utilizarea a 3 matrici:

S(i,j): scorul unei alinieri în care pe poziția i și pe poziția j nu sunt gap-uri

G_b(i,j): scorul unei alinieri în care în secvența b (a doua) pe ultima poziție (j) se află un gap

G_a(i,j): scorul unei alinieri în care în secvența a (prima) pe ultima poziție (i) se află un gap

Inițializare matrici:

$$S(0,0) = 0, S(i,0) = -\infty, S(0,\ j) = -\infty$$

$$G_b(0,0) = 0, G_b(i,0) = -\infty, G_b(0,\ j) = -d - (j-1)e$$

$$G_a(0,0) = 0, G_a(0,\ j) = -\infty, G_a(i,0) = -d - (i-1)e$$

Penalizarea gap-urilor

- Construirea alinierii:
  - se pornește de la max{S(n,m),G_a(n,m),G_b(n,m)}; matricea în care se află valoarea maximă este matricea curentă
  - Se identifică varianta prin care s-a obținut valoarea curenta (i,j) în matricea curentă și se trece la una dintre matricile S, G_a, G_b; în funcție de matricea care devine curentă se actualizează alinierea:
    - S : se preiau a_i și b_j;
    - G_a: se introduce gap în a pe pozitia i și se preia b_j
    - G_b: se preia a_i și se introduce gap în b pe poziția j
  - Procesul continuă până se ajunge la i=0 si j=0

$$S(i,j) = \max\{S(i-1,j-1), G_b(i-1,j-1), G_a(i-1,j-1)\} + M(a(i), b(j))$$

$$G_b(i,j) = \max\{S(i-1,j) - d, G_a(i-1,j) - d, G_b(i-1,j) - e\}$$

$$G_a(i,j) = \max\{S(i,j-1) - d, G_a(i,j-1) - e, G_b(i,j-1) - d\}$$

Alinierea locala a secventelor

Context:

- Alinierea globală (ce permite identificarea de similarități între secvențe întregi) este utilă în cazul în care similaritatea este valabilă pentru întreaga secvență (de exemplu pentru proteine făcând parte din aceeași familie)

- Adeseori scorul potrivirii a două subsecvențe poate fi mai mare decât scorul potrivirii secvențelor în întregime (se întâmplă în cazul în care matricea de scor conține și valori negative)

- De exemplu genele “homeobox” care reglează dezvoltarea embrionară diferă mult de la o specie la alta însă conțin o regiune (homeodomain) care se conservă în mare parte între specii; problema este de a identifica această regiune și de a le ignora pe celelalte

Alinierea locala a secventelor

Exemplu:

Aliniere globala:

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

Aliniere locala:

```
                    tccCAGTTATGTCAGgggacacgagcatgcagagac
                       ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

Regiune conservata

Alinierea locala a secventelor

- Scop: Să se determine cea mai bună aliniere locală a două secvențe

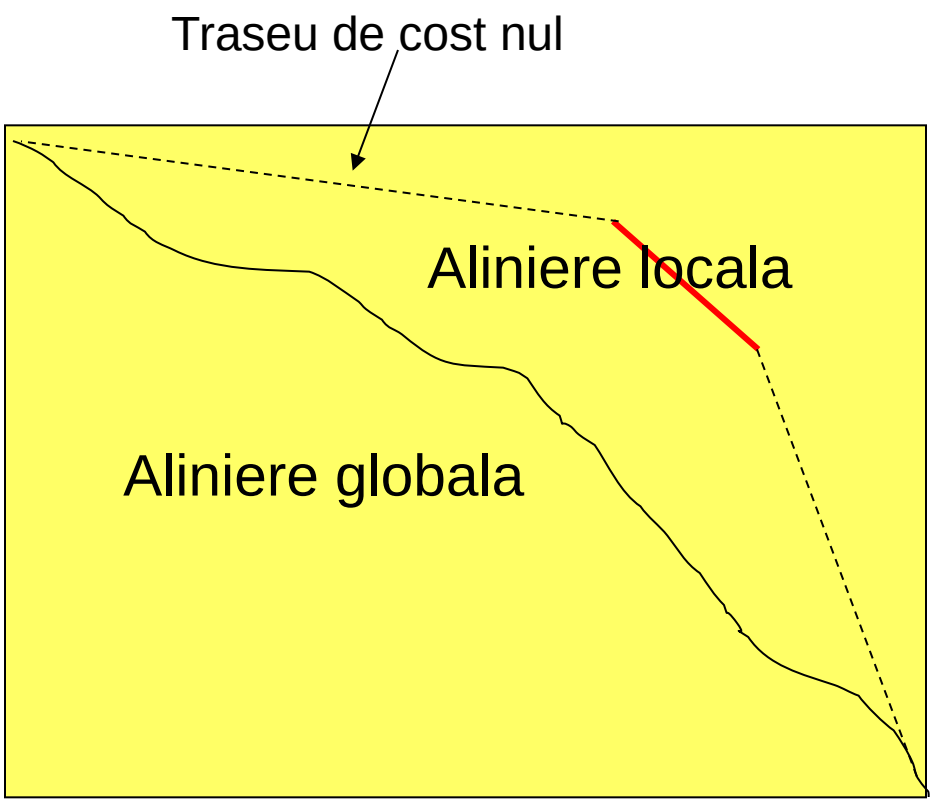
- Input : Secvențele **a, b** și o matrice de scor *M* (care conține atât scoruri pozitive cât și scoruri negative)
- Output : Alinierea unor subsecvențe ale lui a și b pentru care scorul este maxim în raport cu alinierile tuturor celorlalte subsecvențe.

Obs:

- alinierea tuturor perechilor de subsecvențe este ineficientă
- Algoritm mai eficient: **Smith-Waterman (1981)**

Alinierea locala a secventelor

- Ilustrare



- Idee
  - Se foloseste analogia cu problema turistului însă se considera posibil “saltul” de cost nul intre colțul din stânga sus și o pozitie oarecare în matrice precum și “saltul” de la o poziție oarecare și colțul din dreapta jos
  - Ideea este implementată prin restricționarea scorurilor parțiale la valori pozitive

Alinierea locala a secventelor

Diferențe față de alinierea globală: relația de recurență devine (in cazul penalizării liniare a gap-urilor cu valoarea d)

$$S(i,j)=\begin{cases} 0, & i=0 \\ 0, & j=0 \\ \max\{0, S(i-1,j-1)+M(a[i],b[j]), S(i,j-1)-d, S(i-1,j)-d\}, & i>0, j>0 \end{cases}$$

- Se determină valoarea maximă din matricea S: S(i*,j*);a(i*) și b(j*) vor reprezenta ultimele două elemente din subșirurile aliniate
- Alinierea se determină pornind din acest punct și urmând calea inversă a obținerii lui S(i*,j*) până când se ajunge la o valoare S(i0,j0) egală cu 0.

Algoritmul Smith Waterman

- Exemplu

a = ATCTGA

b = TGCA

Matrice scor (c1=1, c2=2)

	A	C	G	T	-
A	1	-1	-1	-1	-2
C	-1	1	-1	-1	-2
G	-1	-1	1	-1	-2
T	-1	-1	-1	1	-2
-	-2	-2	-2	-2	

- Matrice cu scoruri calculate pe baza relației de recurență
-

		T	G	C	A
	0	0	0	0	0
A	0	0	0	0	1
T	0	1	0	0	0
C	0	0	0	1	0
T	0	1	0	0	0
G	0	0	2	0	0
A	0	0	0	1	1

Aliniere: Scor: 2

ATC TG A

TG CA

Algoritmul Smith Waterman

```
AliniereLocalaSmithWaterman (S[1:n,1:m],M[1:5,1:5])
[imax,jmax]=pozitieMaxim(S)
i=imax; j=jmax;
sa=c();  sb=c();          // secventele sa si sb se initializeaza ca liste vide
while S[i,j]>0
   if S[i,j]==S[i-1,j-1]+M[a[i],b[j]]
       then sa=c(a[i], sa);  sb=c(b[j], sb);  i=i-1; j=j-1  //se extind listele
   else
      if S[i,j]=S[i-1,j]+M(a[i],"-") then sa=c(a[i],sa); sb=c("-", sb);  i=i-1;
                else  sa=c("-", sa); sb=c(b[j], sb); j=j-1;
      end
   end
end
```

Exemplu implementare: http://baba.sourceforge.net/

Observatii

- Algoritmii de aliniere bazați pe tehnica programării dinamice (Needleman-Wunsch și Smith-Waterman) generează alinieri optime însă sunt costisitori din punct de vedere computațional: pentru două secvențe de lungimi m respectiv n ordinul de complexitate al implementărilor tradiționale este O(mn) atât din punct de vedere al timpului de calcul cât și din punctul de vedere al spațiului de memorie necesar

- In cazul în care se pune problema alinierii unor secvențe de dimensiuni mari se folosesc
  - Implementari paralele
  - algoritmi sub-optimali (euristici) : FASTA, BLAST (cursul urmator)