

Comparaison de séquences

Eric Rivals

LIRMM - Méthodes Algorithmes pour la Bioinfo

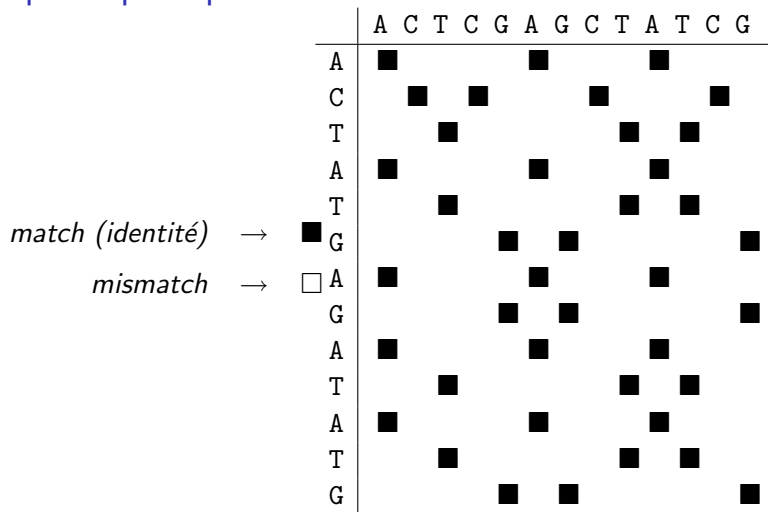
`www.lirmm.fr/~rivals`

Plan

- 1 Comparaison en images
- 2 Plus longue sous-séquence commune
 - Définitions
 - Algorithme quadratique
 - Algorithme linéaire en mémoire
 - Algorithmes paramétrés
 - Algorithme Hunt et Szymanski
 - Autres algorithmes paramétrés
 - Distance et cas multiples
- 3 Alignement
 - Types d'alignements
- 4 Bibliographie

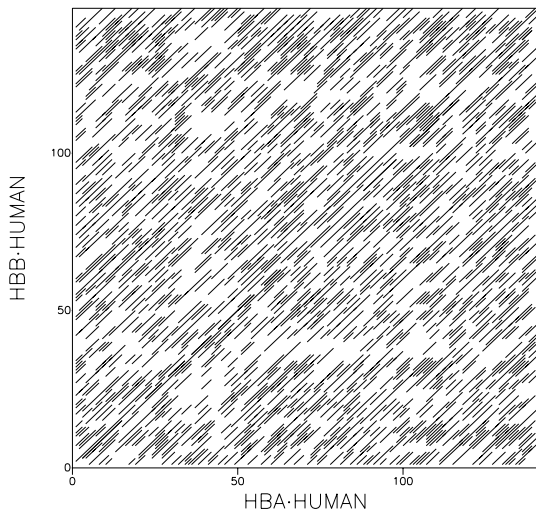
Comparaison de séquences avec le dotplot

Dotplot : principe



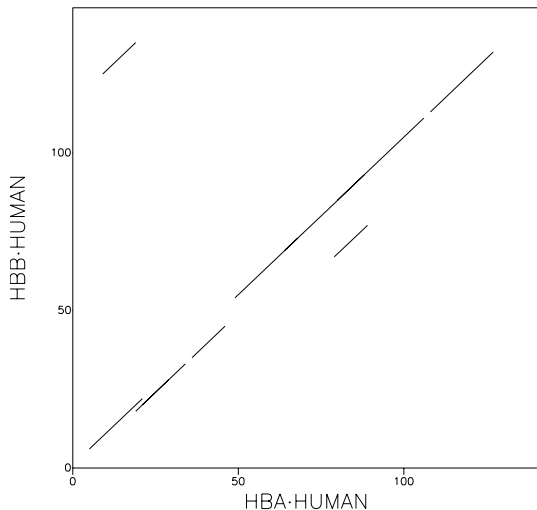
diagonale = région similaire

Dotplot : exemple



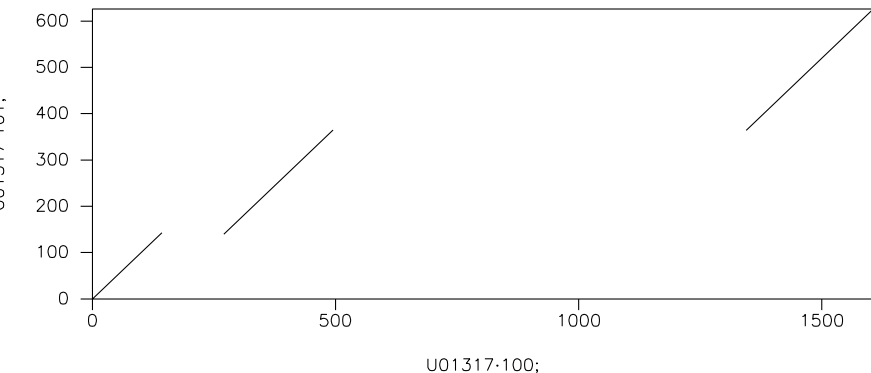
Protéines : chaînes alpha et beta de l'hémoglobine humaine
⇒ problème de bruit ; solution : filtrage

Dotplot avec filtrage



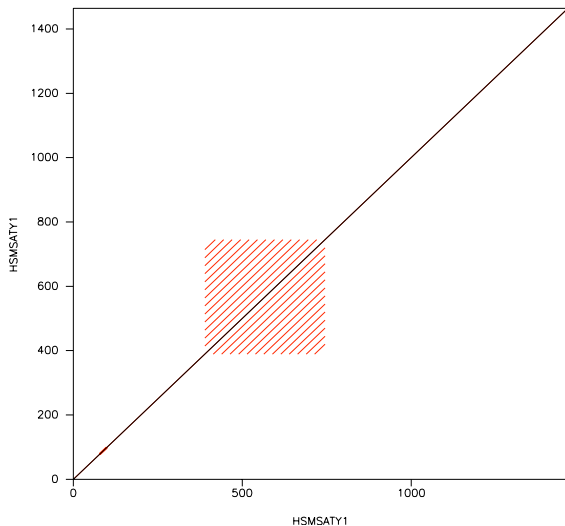
Idem ; filtre mot de longueur 10 de score 23.

Similarités locales



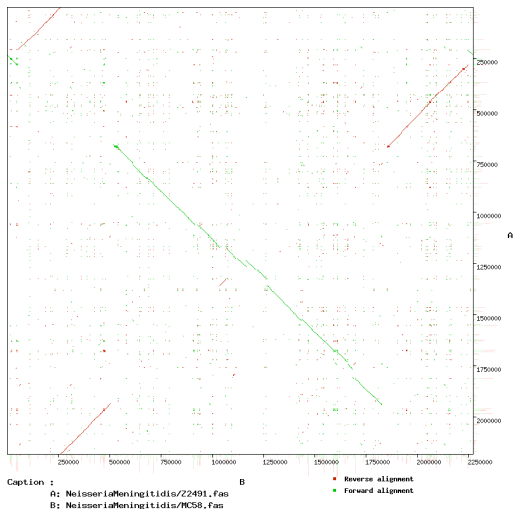
horizontalement : séquence du gène de hémoglobine beta humaine
verticalement : ARN épissé issu de ce gène

Répétitions



horiz. et vert. : une séquence contenant le minisatellite MSY1

Inversions génomiques



horiz. et vert. : génomes de 2 souches de bactérie *Neisseria meningitidis*

Avantages et inconvénients

- Avantages :

- ▶ simple, visuel
- ▶ très informatif

- Inconvénients :

- ▶ identification \Rightarrow pas de méthode de détection automatique
- ▶ interprétation \Rightarrow pas de mesure objective

\Rightarrow besoin d'une mesure quantitative de similarité

Plus longue sous-séquence commune

Définitions

- Notations : Soit Σ un alphabet de cardinal ≥ 2 . Soient x, y deux séquences, chaînes ou mots sur Σ . On note :
 - ▶ $|x|$: longueur de x .
 - ▶ x est indicé de 1 à $|x|$. Soient 2 entiers i, j tq $1 \leq i \leq j \leq |x|$.
 - ▶ $x[i]$ est le caractère à la i^{e} position de x .
 - ▶ $x[i : j] := x[i] \dots x[j]$ est le facteur de x compris entre les positions i et j .
 - ▶ xy : concaténation de x et y .
- **Sous-séquence** : y est une sous-séquence de x ssi l'on peut obtenir y à partir de x en supprimant certaines positions de x .
- **Exemple** : Soient $x := agtatgct$, $y := agct$ et $z := catg$.
Alors $|x| = 8$, $x[4] = a$, $x[4 : 7] = atgc$ est un facteur de x
 y est une sous-séquence de x car $y = x[1]x[2]x[7]x[8]$;
on a aussi $y := x[4]x[6]x[7]x[8]$. z n'est pas une sous-séquence de x .

Plus longue sous-séquence commune (PLSC-LCS)

- Notions de **sous-séquence commune (SC)** à deux mots et de **plus longue sous-séquence commune (PLSC)**
- Exemple : Soient $x := agtatgct$, $y := agct$ et $z := catg$.
 - $PLSC(x, z) = PLSC(z, x) = atg$
car $atg = x[1]x[3]x[6]$ et $atg = z[2]z[3]z[4]$.
 - $PLSC(x, y) = agct = y$.
- PLSC : longueur de la PLSC = plus grand nombre d'identités entre les deux séquences, en respectant l'ordre des 2 séquences.

Algorithme quadratique

Calcul PLSC [Wagner et Fischer, 74]

- **Récurrence** : Calcul d'une matrice $L[0 : m, 0 : n]$ où $L[i, j]$ est la PLSC entre $x[1 : i]$ et $y[1 : j]$.

$$L[i, j] = \begin{cases} 0, & \text{si } i = 0 \text{ ou } j = 0 \\ L[i - 1, j - 1] + 1 & \text{si } x[i] = y[j] \\ \max(L[i - 1, j], L[i, j - 1]) & \text{sinon} \end{cases}$$

- **Bactracking** : mémoriser les “pas” sur le chemin en codant :
1 pour \leftarrow , 2 pour \uparrow et 3 pour \swarrow .
Mémoriser les chemins dans une matrice $P[0 : m, 0 : n]$.
- **Complexité** : $O(mn)$ en temps et en espace.

PLSC classique

In: séquence x , longueur m , séquence y , longueur n

Sortie : longueur de la PLSC, table C contenant une PLSC ;

Variables : $L[0 : m, 0 : n]$ et $P[0 : m, 0 : n]$: matrices d'entiers ;

Initialisation de L et P ;

FOR $i := 1$ to m DO

FOR $j := 1$ to n DO

IF $x[i] = y[j]$ THEN

$L[i, j] := L[i - 1, j - 1] + 1$; $P[i, j] := 3$;

ELSE

IF $L[i - 1, j] > L[i, j - 1]$ THEN

$L[i, j] := L[i - 1, j]$; $P[i, j] := 1$;

ELSE

$L[i, j] := L[i, j - 1]$; $P[i, j] := 2$;

$p := L[m, n]$;

Backtracking ;

Return p et C ;

PLSC backtracking

In: matrice P , entier p , séquence x , longueur m

Sortie : table C contenant une PLSC ;

$i := m$; $j := n$; $k := p$;

WHILE $k > 0$ DO

IF $P[i, j] = 3$ THEN

$C[k] := x[i]$; $i := i - 1$; $j := j - 1$; $k := k - 1$;

ELSE

IF $P[i, j] = 1$ THEN

$i := i - 1$;

ELSE

$j := j - 1$;

Retourner $C[1 : k]$;

Exemple

- Séquences :
 - ▶ *ACTCCATGCA* de longueur 10 et
 - ▶ *CATCAGTA* de longueur 8
- Matrice de calcul de dimension $(8 + 1) * (10 + 1)$
- Entrée $(9, 11)$: longueur de la PLSC

Exemple : matrice

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

Exemple : solutions

- PLSC de longueur 6; Combien de PLSC ? 4
- Solution(s) :

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

Algorithme linéaire en mémoire

[Hirschberg 75]

Algorithme en espace linéaire [Hirschberg 75]

- Calcul de la **longueur** de la PLSC possible avec 2 colonnes ou 2 lignes de la matrice L **mais pas celui d'une PLSC**.
- Algorithme de calcul par ligne : $\text{AlgoLCS-Ligne}(x, m, y, n, L)$ où L est un vecteur représentant la ligne j de la matrice et AlgoLCS-ligne calcule puis retourne la ligne $j + 1$ dans L .
- Idée pour espace linéaire : approche « diviser pour régner »
Une PLSC optimale est faite d'un préfixe et d'un suffixe eux-mêmes optimaux.
- Méthode : trouver un point median sur le chemin optimal et calculer récursivement des chemins optimaux pour les préfixes et suffixes de x et y .
- Point médian : indice k tel que
 $PLSC(x[1..\lceil m/2 \rceil], y[1..k]) + PLSC(x[\lceil m/2 \rceil + 1..m], y[k + 1..n])$
soit maximal.

AlgoLCS-Milieu

- **Entrée** : séquence x , longueur m , séquence y , longueur n
- **Sortie** : point médian de la PLSC ;
- **Variables** :
 $L[0 : n]$, $L_n[0 : n]$, $P[0 : n]$, $P_n[0 : n]$: tableaux d'entiers ;

AlgoLCS-Milieu

```

L[0 : n] := 0;  mid := ⌈m/2⌉;
FOR i := 1..mid DO
  AlgoLCS-Ligne(x, mid, y, n, L);
FOR j := 0..n DO
  P[j] := j;
FOR i := mid + 1..m DO
  FOR j := 1..n DO
    IF x[i] = y[j] THEN
      Ln[j] := L[j - 1] + 1; Pn[j] := P[j - 1];
    ELSE
      IF Ln[j - 1] > L[j] THEN
        Ln[j] := Ln[j - 1]; Pn[j] := Pn[j - 1];
      ELSE
        Ln[j] := L[j]; Pn[j] := P[j];
  L[0 : n] := Ln[0 : n]; P[0 : n] := Pn[0 : n];
Retourner P[n];

```

AlgoLCS-Lineaire

Entrée : séquences x, y , longueurs m, n , table C

Sortie : une PLSC de x et y dans la table C , et sa longueur ;

Variables : $C_1[1 : \lceil n/2 \rceil]$, $C_2[1 : \lceil n/2 \rceil]$;

$mid := \lceil m/2 \rceil$;

IF $m = 0$ ou $n = 0$ **THEN**

retourner ϵ (le mot vide) ;

Calcul du point milieu de la PLSC ;

$k := \text{AlgoLCS-Milieu}(x, mid, y, n)$;

Calcul du préfixe de la PLSC ;

$p_1 := \text{AlgoLCS-Lineaire}(x[1..mid], mid, y[1..k], k, C_1)$;

Calcul du suffixe de la PLSC ;

$p_2 := \text{AlgoLCS-Lineaire}(x[mid + 1..m], m - mid, y[k + 1..n], n - k, C_2)$;

$C := C_1[1..p_1].C_2[1..p_2]$;

Return $p_1 + p_2$;

Algorithmes paramétrés

Algorithmes paramétrés pour la PLSC

Complexité en fonction d'autres paramètres que n et m :

- nombre de paires d'identité r
- nombre de paires d'identité dominante
- longueur de la PLSC.

Idée : algorithmes rapides si r est petit par rapport à n ou m .

Algorithmes paramétrés

- Hunt et Szymanski [[Hunt et Szymanski 77](#)]
- Hirschberg [[Hirschberg 77](#)]

Algorithme Hunt et Szymanski

[Hunt et Szymanski 77]

Algorithme Hunt et Szymanski

- étudie les propriétés de la matrice
- monotonie des valeurs par ligne/colonne
- positions des variations (+1) entre éléments successifs sur une ligne/colonne
- paramètre : nb de paires (i, j) telles que $x[i] = y[j]$
i.e., le nb de point dans le dotplot
- [Hunt et Szymanski 77]

Propriétés de la matrice L

- ① Sur une ligne, les valeurs de $L[i, \cdot]$ croissent avec j

$$\forall 0 < j \leq m, \quad L[i, j - 1] \leq L[i, j]$$

- ② La différence maximale entre deux entrées successives d'une ligne est de au plus 1

$$\forall 0 < j \leq m, \quad L[i, j] - L[i, j - 1] \leq 1$$

- ③ Ce n'est pas parce que $x[i] = y[j]$ que la différence est de 1, et inversement.

$$(x[i] = y[j]) \not\Rightarrow (L[i, j] - L[i, j - 1] = 1)$$

Propriétés 1, 2 et 3 aussi vraies sur les colonnes.

Pour quelles colonnes a-t'on $x[i] = y[j]$ et $L[i, j] - L[i, j - 1] = 1$?

Indices des changements sur une ligne

Definition

Pour $0 \leq i, l \leq n$ on note $k_{i,l} := \min\{j : 0 < j \leq m \text{ et } L[i, j] = l\}$

- ① Comme $k_{i,l}$ est la position minimale où le score de l est atteint, on a $L[i, k_{i,l}] = L[i, k_{i,l} - 1] + 1$.
- ② On a : $k_{i,l-1} < k_{i+1,l} \leq k_{i,l}$.

Preuve :

- de $k_{i+1,l} \leq k_{i,l}$. De par la récurrence, pour tout indice j on a $L[i+1, j] \geq L[i, j]$, et en particulier $L[i+1, k_{i,l}] \geq L[i, k_{i,l}]$. Donc $k_{i+1,l} \leq k_{i,l}$.
- de $k_{i,l-1} < k_{i+1,l}$. Par définition de $k_{i,l-1}$, on a $L[i, k_{i,l-1}] = l - 1$ et $L[i, k_{i,l-1} - 1] = l - 2$. D'après la propriété 2 (pour les colonnes), on a $L[i+1, k_{i,l-1} - 1] \leq L[i, k_{i,l-1} - 1] + 1 = l - 1$. D'après la récurrence pour L , on obtient que $L[i+1, k_{i,l-1}] \leq l - 1$. On en déduit que $k_{i,l-1} < k_{i+1,l}$. CQFD.

Récurrence sur les indices des changements sur une ligne

D'après la propriété précédente, on a deux cas pour calculer $k_{i+1,l}$.

- Soit il existe au moins un indice j' dans l'intervalle $k_{i,l-1} < j' < k_{i,l}$ tel que $x[i+1] = y[j']$. Dans ce cas, pour j le minimum parmi ces j' , on a $L[i+1,j] = L[i,j-1] + 1 = l$, car on sait que $L[i,j-1] = l-1$, $L[i+1,j-1]$ et $L[i,j] \leq l-1$. On a donc $k_{i+1,l} = j$.
- Soit il n'existe aucun indice j' vérifiant ces conditions et dans ce cas la récurrence implique que $k_{i+1,l} = k_{i,l}$.

On a donc la **récurrence** suivante pour $k_{i+1,l}$:

$$k_{i+1,l} := \begin{cases} \min\{j' : k_{i,l-1} < j' < k_{i,l} \text{ tel que } x[i+1] = y[j']\} \\ k_{i,l} \text{ s'il n'existe aucun } j' \text{ satisfaisant ces conditions} \end{cases}$$

Algorithme de Hunt et Szymanski : implémentation

- Entrée et sortie : identiques à celles de l'algorithme classique.
- Variables :
 - ▶ k : vecteur d'indices,
 - ▶ `matchlist` : table de listes de positions d'identités
 - ▶ lm : liste d'indices,
 - ▶ p : pointeur;
- [Hunt et Szymanski 77]

Algorithme de Hunt et Szymanski

```

//Initialise vecteur k, liste lm, et matchlist;
k[0] := 0; lm := NULL;
FOR i := 1..n DO
  k[i] := n + 1;
  matchlist[i] := (j1, ..., jq) tels que
    j1 > j2 > ... > jq et x[i] = y[jl] pour tout l ∈ [1, q];
//Calcul des valeurs successives de k;
FOR i := 1..n DO
  FOR j dans matchlist[i] DO
    cherche index l tel que k[l - 1] < j ≤ k[l];
    IF j < k[l] THEN
      k[l] := j; ajouter (i) en tête de lm;
j := max{l tel que k[l] < n + 1}; //Calcul longueur PLSC;
i := j; p := lm; //Extraction PLSC;
WHILE p ≠ null DO
  C[i] := x[p → i]; p := p → suiv;
Return j and C;

```

Autres algorithmes paramétrés

Liste d'algorithmes paramétrés

- Algorithme classique [Wagner et Fischer 74]
- Algorithme linéaire en espace [Hirschberg 75]
- Algorithme en $O(pn + n \log(n))$ où p est la longueur de $PLSC(x, y)$ et n la longueur de la plus petite des séquences [Hirschberg 77]
- Algorithme en $O((r + n) \log(n))$ où r est le nombre de paires d'identités [Hunt et Szymanski 77]
Implémentation de la commande Unix *diff*
- Algorithme en $O(n(m - p))$ [Nakatsu et al. 82]
- Algorithme en $O(n(m - p))$ [Myers 86]

LCS : distance et cas multiple séquences

Distance d'édition et PLSC

- **Distance d'édition** : notée $d_E(x, y)$, est le nombre minimal d'insertions et de délétions nécessaires pour transformer x en y
- Problème dual de la PLSC, relation entre $PLSC(x, y)$ et $d_E(x, y)$

$$|x| + |y| - d_E(x, y) = 2 * PLSC(x, y)$$

Autres distances

Soient s, t deux séquences.

Définition : La **distance de Hamming** de s à t est le nombre minimal de substitutions nécessaires pour transformer s en t . s et t doivent avoir la même longueur.

Définition : La **distance de Levenshtein** entre s et t est le nombre minimal d'insertions, de délétions et de **substitutions** pour transformer s en t . On la note $D_L(s, t)$.

Cas multiple

Lorsque le nombre de chaînes en entrée est non borné, c.à.d. un paramètre $k \geq 2$, alors le problème PLSC est :

- NP-dur même si l'alphabet est binaire [Maier 78]
réduction à Vertex-Cover
- W[1]-dur quant au paramètre du nombre de chaînes en entrée
complexité paramétrique [Pietrzak 03]
- difficile à approximer pour des alphabets non bornés
[Jiang et Li 95].
- NP- et W[1]-dur lorsque l'on considère des chaînes circulaire et/ou
non orientées, même si l'alphabet est binaire
[Nicolas et Rivals 07]

Alignement de séquences

Alignement

- Données :
 - ▶ une paire de séquences (ADN / protéine)
 - ▶ une méthode de score
- Prise en compte des mutations ponctuelles : 3 opérations
insertion, délétion, substitution
- But :
 - ▶ quantifier et localiser la similarité : score + alignement
 - ▶ trouver la meilleure mise en correspondance des résidus qui conserve l'ordre des séquences : meilleur score

```

R D I S L V - - - K N A G I
|   |   | |           | |   | |
R N I - L V S D A K N V G I

```

Définition d'un alignement

Soient s et t deux séquences respectivement de longueur n et m . Soit un entier i dans $[1, n]$ alors s_i est le i ème symbole de s .

Définition : un **Alignement** global de s et t :

- une matrice à 2 lignes, et entre $\max(m, n)$ et $n + m$ colonnes,
- où chaque colonne met en correspondance deux symboles :
soit $\begin{pmatrix} s_i \\ t_j \end{pmatrix}$ ou $\begin{pmatrix} s_i \\ - \end{pmatrix}$ ou $\begin{pmatrix} - \\ t_j \end{pmatrix}$,
- où les colonnes doivent respecter l'ordre des séquences,
i.e., par exemple si une colonne contient $\begin{pmatrix} s_i \\ t_j \end{pmatrix}$
la suivante peut contenir $\begin{pmatrix} s_{i+1} \\ - \end{pmatrix}$, $\begin{pmatrix} s_{i+1} \\ t_{j+1} \end{pmatrix}$ ou $\begin{pmatrix} - \\ t_{j+1} \end{pmatrix}$

Remarque : suivant le nombre de symboles $-$, le nombre de colonnes d'un alignement varie.

Combinatoire des alignements

Exemple : Séquences $s := I$, $t := L$

Alignement 1	Alignement 2	Alignement 3
$\begin{pmatrix} I & - \\ - & L \end{pmatrix}$	$\begin{pmatrix} - & I \\ L & - \end{pmatrix}$	$\begin{pmatrix} I \\ L \end{pmatrix}$

Soient $s := IL$ et $t := LV$. **13** alignements possibles.

$\begin{pmatrix} I & L \\ L & V \end{pmatrix}$	$\begin{pmatrix} I & L & - \\ L & - & V \end{pmatrix}$	$\begin{pmatrix} I & L & - & - \\ - & - & L & V \end{pmatrix}$	$\begin{pmatrix} - & - & I & L \\ L & V & - & - \end{pmatrix}$
$\begin{pmatrix} I & - & L \\ - & L & V \end{pmatrix}$	$\begin{pmatrix} I & - & L \\ L & V & - \end{pmatrix}$	$\begin{pmatrix} I & - & L & - \\ - & L & - & V \end{pmatrix}$	$\begin{pmatrix} - & I & - & L \\ L & - & V & - \end{pmatrix}$
$\begin{pmatrix} I & L & - \\ - & L & V \end{pmatrix}$	$\begin{pmatrix} - & I & L \\ L & - & V \end{pmatrix}$	$\begin{pmatrix} I & - & - & L \\ - & L & V & - \end{pmatrix}$	$\begin{pmatrix} - & I & L & - \\ L & - & - & V \end{pmatrix}$
	$\begin{pmatrix} - & I & L \\ L & V & - \end{pmatrix}$		

Composantes d'un schéma de scores

- identité/substitution :
 - ▶ matrice s de score de similarité (PAM, BLOSUM, etc.)
 - ▶ $s(a, b)$ = score d'alignement des résidus a et b
- insertion/délétion ou *indel* :
fonction de pénalité $\left\{ \begin{array}{l} \text{élémentaire : coût unitaire pour un indel} \\ \text{complexe : coûts affine, logarithmique} \end{array} \right.$
- le score de l'alignement : somme des scores des événements élémentaires

Algorithme

création d'une table indexée par les deux séquences

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

Case (i, j) : score alignement entre les i premières bases de ACTGTAATG et les j premières bases de ACGGCTATC.

Formellement

- s : une matrice de score ; g : pénalité associée à un indel
- initialisation : $M(0, 0) = 0$, $M(0, j) = g \times j$, $M(i, 0) = g \times i$
- remplissage

$$M(i, j) = \max \begin{cases} M(i-1, j-1) + s(x_i, y_j) & \text{match ou mismatch} \\ M(i-1, j) + g & \text{insertion} \\ M(i, j-1) + g & \text{délétion} \end{cases}$$

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

Coûts : $s(a, b) = -1$ avec $a \neq b$ et 2 sinon ; $g = -1$

Case (9, 9) : score de l'alignement global entre ACGGCTATC et ACTGTAATG.

Résultat :

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

Types d'alignements

Types d'alignement

global : alignement des séquences complètes
[Needleman & Wunsch, 70]

local : alignement des **meilleures sous-séquences**
[Smith & Waterman, 81]

... gqvaryag	<table style="border-collapse: collapse; text-align: center;"> <tr><td>E</td><td>K</td><td>L</td><td>F</td><td>H</td><td>S</td><td>I</td><td>F</td><td>V</td><td>E</td></tr> <tr><td> </td><td> </td><td> </td><td> </td><td></td><td></td><td></td><td></td><td> </td><td> </td></tr> <tr><td>E</td><td>K</td><td>L</td><td>F</td><td>V</td><td>-</td><td>L</td><td>R</td><td>V</td><td>E</td></tr> </table>	E	K	L	F	H	S	I	F	V	E											E	K	L	F	V	-	L	R	V	E	qnislt...
E	K	L	F	H	S	I	F	V	E																							
E	K	L	F	V	-	L	R	V	E																							
... teqlinyi		laesas...																														

semi-global : inclusion, chevauchements (modification de l'algo global)



Alignement local [Smith & Waterman, 1981]

- évaluation d'une ressemblance locale entre deux séquences
- recherche de la région de plus forte similarité

Exemple

alignement **global**

```

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

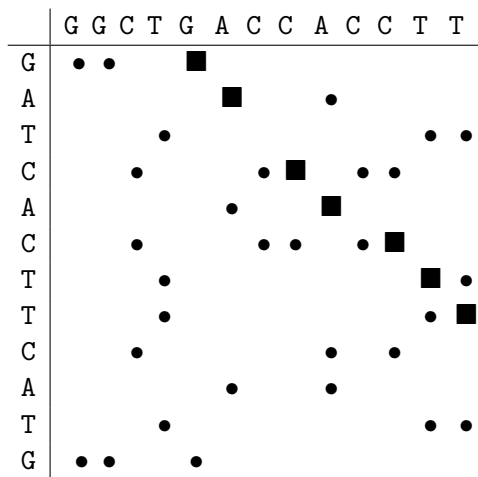
```

alignement **local**

```

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

```



les séquences présentent une similarité que l'alignement global ne révèle pas

Formellement

- s : une matrice de score
- g : pénalité associé à un indel
- initialisation

$$M(0, 0) = 0$$

$$M(0, j) = 0 \longleftarrow$$

$$M(i, 0) = 0 \longleftarrow$$

- remplissage

$$M(i, j) = \max \begin{cases} M(i-1, j-1) + s(x_i, y_j) & \text{match ou mismatch} \\ M(i-1, j) + g & \text{insertion} \\ M(i, j-1) + g & \text{délétion} \\ 0 & \longleftarrow \end{cases}$$

Bibliographie LCS

- Wagner R., Fischer M.J. (1974) The string-to-string correction problem. J. ACM, 21(1) :168-173, 1974.
- Hirschberg, D. S. (1975). "A linear space algorithm for computing maximal common subsequences". Communications of the ACM 18 (6) : 341-343. doi :10.1145/360825.360861
- Hunt J.W., Szymanski T.G. (1977) A fast algorithm for computing longest common subsequences, Comm. ACM 20 350-353.
- Hirschberg D.S. (1977) Algorithms for the longest common subsequence problem. J. ACM 24 664-675.
- David Maier (1978). "The Complexity of Some Problems on Subsequences and Supersequences". J. ACM (ACM Press) 25 (2) : 322-336. doi :10.1145/322063.322075.

Bibliographie LCS & alignement

- Pietrzak K. (2003), On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems, *J. Computer System Sciences* 67 (4) 757-771.
- Nicolas F, Rivals. E (2007), Longest Common Subsequence Problem for Unoriented and Cyclic Strings, *Theoretical Computer Science (TCS)*, Vol. 370, p. 1-18.
- Jiang T., Li M. (1995) On the approximation of shortest common supersequences and longest common subsequences, *SIAM Journal on Computing* 24 (5) 1122-1139.
- Smith T.F., Waterman M.S. (1981). Identification of Common Molecular Subsequences. *J. Molecular Biology* 147 : 195-197.
doi :10.1016/0022-2836(81)90087-5