Plan

Sequence Comparison

- Genome : all the information needed to build the basic building blocks and ensure the functioning of the organism.
- Sequencer : millions of sequences available including unknown functions
- Methods for inferring the role of each of the proteins deduced from sequencing ?
- The first ideas were based on the comparison of sequences.
  - 2 proteins with the same physico-chemical properties perform similar functions.
  - the three-dimensional form of proteins, which partly explains their physico-chemical properties, is not easily accessible.
  - If there are two similar (or identical) parts between 2 proteins, the 2 proteins will have similar three-dimensional conformations and participate in similar functions.
  - In other words, if the similar parts are not hidden in the hydrophobic part of the tertiary structure, these parts have the same physico-chemical characteristics, which will lead to similar functions.
- Here : one of the 1st methods for finding similar sub-seqs. Dynamic Programming

Definitions & Notations

Alphabets

- An alphabet A is a set of symbols from which words are written.
- Alphabet for DNA sequences : $\mathcal{A} = \{A, C, G, T\}$
  A : Adenine C : Cytosine G : Guanine T : Thymine.
- Alphabet for RNA sequences : $\mathcal{A}' = \{A, C, G, U\}$
  U : Uracil.
- Alphabet for proteins :
  $\mathcal{A}'' = \{A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V\}$

Other alphabets to take account of incomplete information.
$\mathcal{B} = \{A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V, B, Z, X, *\}$

letter	abbrev.	name	associated codons
B	Asx	Asparagine | Aspartic acid (Aspartate)	AAU AAC GAU GAC
Z	Glx	Glutamine | Glutamic acid (Glutamate)	CAA CAG GAA GAG
X	Xaa	Any amino acid	tous
*	END	Termination codon (translation stop)	UAA UAG UGA

Definitions & Notations

Notations : let us consider $A = A_1 A_2 \dots A_n$

- A_i or $A[i]$ is the i^{th} letter of sequence A,
- $|A|$ represents the length of the sequence,
- $A_{i,j}$ or $A[i,j]$: represents the subsequence of A between positions i and j,
- ϵ is the empty string,
- $\ll . \gg$ represents the concatenation operation between sequences.

$$\text{If } \begin{cases} S_1 = ACGTGGTC \\ S_2 = GTGCCA \end{cases} \quad \text{then} \quad S_1.S_2 = ACGTGGTCGTGCCA$$

Aligning 2 sequences

≪ mettre en évidence les similitudes et les différences entre les deux séquences ≫

Exemple : Soient $S_1 = ACGTCGTTC$ et $S_2 = AGGCCTCGC$.

L'alignement entre ces deux séquences est :

```
S1 : ACG--TCGTTC
     | |  |||  |
S2 : AGGCCTCG--C
```

- Une suite contigüe de «−» sera appelée un **décalage** (en anglais ***gap***, ***indel***)
- une **délétion** est un décalage dans la deuxième séquence (on passe de la première séquence à la seconde par la suppression de quelques lettres)
- une **insertion** est un décalage dans la première séquence (on passe de la première séquence à la seconde en ajoutant quelques lettres)
- différence insertion / délétion n'est que formelle
- *appariement* : mise en correspondance de deux lettres
  - deux lettres identiques : *appariement exact* (*match* en anglais.)
  - deux lettres diffèrent : *appariement non-exact* (*mismatch* en anglais)

Aligning 2 sequences

Pseudo-alignement

Un ***pseudo-alignement global*** L entre n séquences $\{S_1, S_2, \dots S_n\}$ est un ensemble ordonné de mots $(\{S'_1, S'_2, \dots S'_n\})$ construits sur l'alphabet $\mathcal{A}'$ ayant les contraintes suivantes :

- tous les mots ont la même longueur notée $|L|$,
- si on supprime dans chaque mot S'_i les symboles «−», on obtient la séquence S_i.

alignement global

Un ***alignement global*** entre n séquences $\{S_1, S_2, \dots S_n\}$ est un pseudo-alignement global dont aucune des colonnes n'est vide : pour tout indice $k \in [1..|L|]$, il existe une séquence S'_l dont le $k^{ième}$ caractère n'est pas le caractère «−» :

$$\forall k \in \{1, 2, \dots |L|\} \quad \exists l \in \{1, 2, \dots n\} \quad tel \ que \ \ S'_l[k] \neq \text{«−»}$$

Alignement local

Un ***alignement local*** entre n séquences $\{S_1, S_2, \dots S_n\}$ est un alignement global sur des sous-séquences $\{S_1[l_1, r_1], S_2[l_2, r_2], \dots, S_n[l_n, r_n]\}$.

L'alignement de 2 séquences n'est qu'un cas particulier de l'alignement multiple. Les méthodes d'alignement multiple ($n > 2$) sont très différentes...

Aligning 2 sequences

«Aligner deux séquences, c'est mettre en correspondance des lettres de la première séquence avec des lettres de la seconde.»

On cherche donc à transformer la 1ère séquence en l'autre. Pour cela, nous disposons de trois opérations :

- opération de mutation : $a \longrightarrow a'$ (éventuellement $a = a'$)
- opération de délétion : on supprime une lettre,
- opération d'insertion : on insère une lettre.

Chacune de ces opérations a un score (coût, poids). Le «meilleur» alignement est celui pour lequel la somme des coûts est maximal. Le *score de l'alignement* est la somme totale des coûts élémentaires.

Les fonctions de coûts :

1. score de mutation entre 2 lettres : matrice symétrique $\sigma = (\sigma_{ij})_{i,j}$. Exemple de matrices : PAM, Dayhoff, Henikoff, Blosum...
2. scores des insertions délétions : $-g$ pour n'importe quelle lettre.

Formellement, si l'alphabet utilisé est $\mathcal{A}$, un alignement des séquences A et B est une suite de paires alignées sur l'alphabet $\mathcal{A} \cup \{-\}$: $\{(A'_k, B'_k)_{1 \leq k \leq K}\}$ tel que :

- si on supprime dans $A'_i A'_2 \dots A'_K$ les «−», on ré-obtient séquence A.
- si on supprime dans $B'_i B'_2 \dots B'_K$ les «−», on ré-obtient la séquence B.
- $\forall k \in [1, K]$, A'_k et B'_k ne sont pas simultanément égaux à «−»

Exemple : $L(A, B) = \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \begin{bmatrix} a_2 \\ b_2 \end{bmatrix} \begin{bmatrix} a_3 \\ b_3 \end{bmatrix} \dots \begin{bmatrix} a_{K'} \\ b_{K'} \end{bmatrix}$

Score de l'alignement : $score(L(A, B)) = \sum_{i=1}^{k} score(\begin{bmatrix} a'_i \\ b'_i \end{bmatrix})$

Score de l'alignement optimal : $score_{optimal}(A, B) = \max score(L(A, B))$

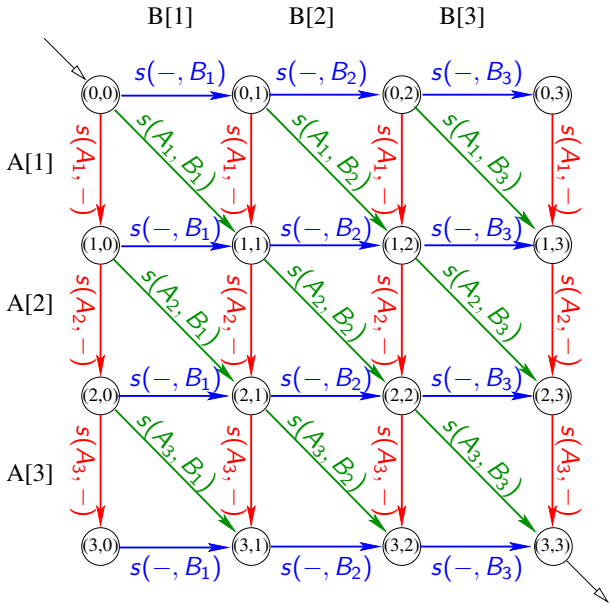
Reformulation using a directed graph

Ce problème peut se résoudre par les algorithmes de plus court chemin à travers un graphe orienté valué. Posons $\mathcal{G} = (E, V)$ le graphe orienté défini par :

- E est l'ensemble des sommets du graphe : $E = \{(i, j) | i \in [0, n], j \in [0, m]\}$
- V est l'ensemble des arcs :
  - si $i \in [1, n]$ et $j \in [1, m]$ alors il y a un arc partant du sommet $(i-1, j-1)$ vers le sommet (i, j) représentant la substitution de A_i en B_j notée $\begin{bmatrix} A_i \\ B_j \end{bmatrix}$. A cet arc est associé un coût de $score(\begin{bmatrix} A_i \\ B_j \end{bmatrix})$.
  - si $i \in [1, n]$ et $j \in [0, m]$ alors il y a un arc de $(i-1, j)$ vers (i, j) représentant la suppression de la lettre A_i notée $\begin{bmatrix} A_i \\ - \end{bmatrix}$. Cet arc reçoit un coût de $score(\begin{bmatrix} A_i \\ - \end{bmatrix})$.
  - si $i \in [0, n]$ et $j \in [1, m]$ alors il y a un arc de $(i, j-1)$ vers (i, j) représentant l'insertion de la lettre B_j notée $\begin{bmatrix} - \\ B_j \end{bmatrix}$. Cet arc reçoit un coût de $score(\begin{bmatrix} - \\ B_j \end{bmatrix})$.

Unique source : $(0, 0)$ Unique puits : (n, m).

Reformulation using a directed graph



Reformulation using a directed graph

Le chemin de plus grand coût donnera l'alignement de plus grand score.

- La méthode générale est celle de Bellman (prog. dyn.)
  complexité en $O(A \times S)$
  Le nombre de sommets $|S| = (n+1) \times (m+1)$
  Le nombre d'arcs $|A| = O(3mn)$
  $\Rightarrow$ L'algorithme de Bellman donne une complexité en $O(n^2 m^2)$.
  Améliorations possible (D'Esopo et Pape).
- Mais il n'y a pas de circuit...
  Décomposition en niveaux possible.
  Complexité en $O(A)$ donc en $O(3mn)$.
  On est donc amené à calculer l'ensemble des chemins optimaux partant du sommet initial et allant à chacun des autres sommets.

Iterative expression

$NW(i,j)$: score d'alignement des 2 préfixes $A[1,i]$ et $B[1,j]$.
On a la récurrence suivante :

$$NW(i,j) = max \begin{pmatrix} NW(i-1,j-1) + \sigma(A_i, B_j) \\ NW(i-1,j) - g \\ NW(i,j-1) - g \end{pmatrix}$$

On peut donc calculer l'ensemble des valeurs $NW(i,j)$ en $O(m \times n)$.
Pour la dernière paire de l'alignement de $A[1,i]$ et $B[1,j]$, on a 3 choix possibles :

- appariement : $NW = NW(i-1,j-1) + score(\begin{bmatrix} A_i \\ B_j \end{bmatrix})$
- délétion : $NW = NW(i-1,j) + score(\begin{bmatrix} A_i \\ - \end{bmatrix})$
- insertion : $NW = NW(i,j-1) + score(\begin{bmatrix} - \\ B_j \end{bmatrix})$

Rq : la complexité temporelle de l'algorithme récursif explose.
Est-on obligé de mémoriser tous les score intermédiaires ?

- Si on ne veut que le score : non. complexité mémoire en $O(min(m,n))$.
- Si on veut avoir le chemin qui maximise le score d'alignement, l'algorithme est très simple si on a stocké tous les résultats intermédiaires :
  - on calcule le score en stockant tous les résultats intermédiaires,
  - on "remonte" pour retrouver les sommets intermédiaires
  La complexité en temps : $O(m \times n)$
  La complexité en mémoire est en $O(m \times n)$.

First improvement

Le coût des insertions/délétions dépendant de la longueur. La récurrence devient un peu plus complexe :

$$NW(i,j) = max \begin{pmatrix} NW(i-1,j-1) + \sigma(A_i, B_j) \\ \max\limits_{1 \le k \le i} \{NW(i-k,j) - g(k)\} \\ \max\limits_{1 \le k \le j} \{NW(i,j-k) - g(k)\} \end{pmatrix}$$

La complexité dans le cas général est en $O(nm^2 + n^2 m)$.

≪ plus l'insertion est longue, plus le surcoût dû à son allongement est faible.≫
$\Rightarrow$ fonction $g(k)$ concave
On retrouve alors un algorithme de complexité inférieure.
Si la fonction $g(k)$ est linéaire : $g(k) = go + ge(k-1)$

$$\begin{aligned} S_2(i,j) &= \max_{1 \le k \le i} \{NW(i-k,j) - g(k)\} \\ &= \max\{\max_{2 \le k \le i} \{NW(i-k,j) - g(k)\}, NW(i-1,j) - g(1)\} \\ &= \max\{\max_{2 \le k \le i} \{NW(i-k,j) - g(\underbrace{k-1}_{k'})\} - ge, NW(i-1,j) - g(1)\} \\ &= \max\{\max_{1 \le k' \le i-1} \{NW(i-1-k',j) - g(k')\} - ge, NW(i-1,j) - g(1)\} \\ &= \max\{S_2(i-1,j) - ge, NW(i-1,j) - g(1)\} \end{aligned}$$

First improvement

On a donc besoin de 3 tableaux :

1. $NW(i,j)$: score d'alignement des préfixes $A[1,i]$ et $B[1,j]$
2. $D(i,j)$: score du meilleur alignement se terminant par une délétion
3. $I(i,j)$: score du meilleur alignement se terminant par une insertion.

Les récurrences s'écrivent :

$$\begin{aligned} NW(i,j) &= max(\ NW(i-1,j-1)+\sigma(A_i,B_j),\ D(i,j),\ I(i,j)\ ) \\ D(i,j) &= max(\ NW(i-1,j)-go,\ D(i-1,j)-ge\ ) \\ I(i,j) &= max(\ NW(i,j-1)-go,\ I(i,j-1)-ge\ ) \end{aligned}$$

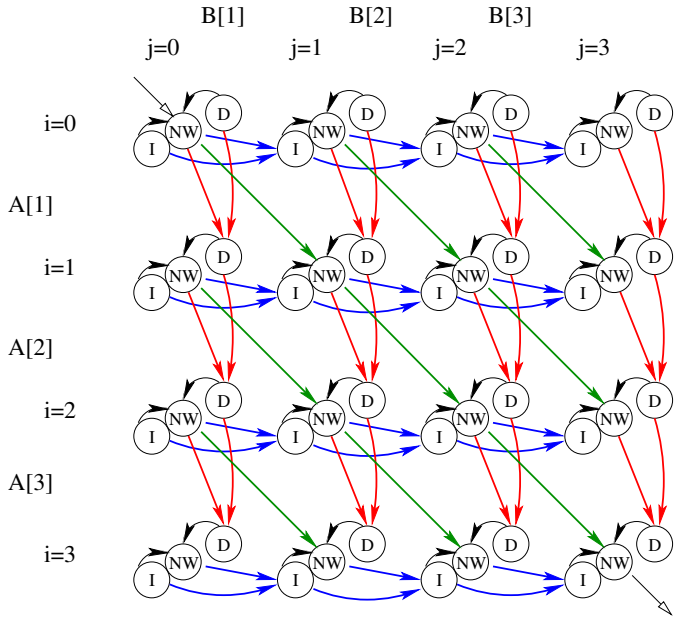
avec les initialisations :

$$\begin{aligned} NW(0,0) &= I(0,0)=D(0,0)=0 \\ D(i,0) &= I(i,0)=-g(i) \quad 1\le i\le n \\ D(0,j) &= I(0,j)=-g(j) \quad 1\le j\le m \end{aligned}$$

Reformulation using a directed graph

1. Alignement global avec pénalisation linéaire : déjà vu.
2. Alignement global avec affine : L'algorithme a besoin de 3 tableaux, ici, on va multiplier les sommets par trois, correspondant respectivement aux trois valeurs $NW(i,j)$, $D(i,j)$ et $I(i,j)$. Il y a six types d'arcs :
   - les arcs diagonaux : appariements $NW(i-1,j-1) \to NW(i,j)$ (score : $\sigma(A[i],B[j])$)
   - création d'une délétion : $NW(i,j) \to I(i+1,j)$ (score : go)
   - extension d'une délétion : $I(i,j) \to I(i+1,j)$ (score : ge)
   - création d'une insertion : $NW(i,j) \to D(i,j+1)$ (score : go)
   - extension d'une insertion : $D(i,j) \to D(i,j+1)$ (score : ge)
   - la fin d'une insertion/délétion : $D(i,j) \to NW(i,j)$ et $I(i,j) \to NW(i,j)$. On pourrait envisager d'avoir une pénalisation de fin d'insertion/délétion.

Reformulation using a directed graph



Local Alignment : Smith & Waterman

On cherche les 2 sous-séquences qui maximisent le score de Needleman et Wunsch :

$$SW(A[1,n],B[1,m]) = \max_{I,J/\,NW(I,J)\ge 0} NW(I,J)$$

où I et J sont 2 segments de $[1,n]$ et $[1,m]$.
On cherche 2 segments pour lesquels le score est positif.
Ces segments existent si on peut trouver $A[i]$ et $B[j]$ | $\sigma(A[i],B[j]) \ge 0$.
$SW(i,j)$: le meilleur score Needleman et Wunsch parmi toutes les sous-séquences de A et B de la forme $A[i_0,i]$ et $B[j_0,j]$.

$$SW(i,j) = max\begin{pmatrix} SW(i-1,j-1)+\sigma(A[i],B[j]) \\ SW(i-1,j)-g \\ SW(i,j-1)-g \\ 0 \end{pmatrix}$$

4 possibilités pour la fin de l'alignement :

- $A[i]$ et $B[j]$ sont en correspondance $\longrightarrow SW(i-1,j-1)+\sigma(A[i],B[j])$
- l'alignement se finit par une délétion $\longrightarrow SW(i-1,j)-g$
- l'alignement se finit par une insertion $\longrightarrow SW(i,j-1)-g$
- il n'existe pas de sous-séquences terminant par $A[i]$ et par $B[j]$ ayant une similarité positive.

Initialisation (ne pas pénaliser les indels du début) :

$$\begin{aligned} SW(0,0) &= 0 \\ SW(i,0) &= 0 \quad 1\le i\le m \\ SW(0,j) &= 0 \quad 1\le j\le n \end{aligned}$$

Any penalty function

Fonction quelconque de pénalisation des insertions/délétions

$$SW(i,j) = max \begin{pmatrix} SW(i-1,j-1) + \sigma(A[i],B[j]) \\ max_{1\leq k\leq i}\{SW(i-k,j) - g(k)\} \\ max_{1\leq k\leq j}\{SW(i,j-k) - g(k)\} \\ 0 \end{pmatrix}$$

Cas d'une pénalisation affine. Posons $g(k) = go + ge(k-1)$.

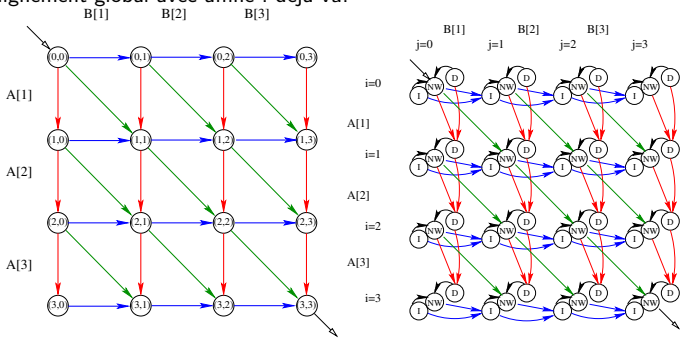
$$\begin{cases} Diag(i,j) & = & SW(i-1,j-1) + \sigma(A[i],B[j]) \\ Del(i,j) & = & max(\ SW(i-1,j) - go\ ,\ Del(i-1,j) - ge\ ) \\ Ins(i,j) & = & max(\ SW(i,j-1) - go\ ,\ Ins(i,j-1) - ge\ ) \\ SW(i,j) & = & max(\ Diag(i,j)\ ,\ Del(i,j)\ ,\ Ins(i,j)\ ,\ 0) \end{cases}$$

Initialisation : première ligne à 0 ; première colonne à 0.

Reformulation using a directed graph

!

1. Alignement global avec pénalisation linéaire : déjà vu.
2. Alignement global avec affine : déjà vu.



3. Alignement local : graphes identiques.
   Ce qui change : ensemble des sommets initiaux et finaux.

Algorithme	Pénalisation	initiaux	finaux
NW	linéaire	$(0,0)$	(m,n)
	affine	$NW(0,0)$	$NW(m,n)$
SW	linéaire	tous	tous
	affine	$SW(i,j)$	$SW(i,j)$

Multiple alignment extension

L'alignement de n séquences (ou alignement multiple) est un problème NP-difficile (quel que soit n). Le problème de la recherche d'une sous-séquence commune de longueur maximale est déjà un problème NP-difficile.

Fonction de score d'un alignement : SP-score (*Sum of Pair*)
Programmation dynamique : généralisation de l'algorithme SW Soit $\mathcal{S} = (S_1, S_2, \ldots, S_n)$, l'ensemble des séquences à aligner.

- Notons $\vec{i} = (i_1, i_2, \ldots, i_n)$, un vecteur de $\mathbb{N}^n$.
- Notons $C(\vec{i}) = C(i_1, i_2, \ldots, i_n)$ le coût minimal de l'alignement des préfixes $S_k[1, i_k]$ de longueur i_k de chaque séquence S_k, pour $k = 1, 2, \ldots, n$.
- Pour l'alignement de deux séquences, on procède par colonne de l'alignement : on rajoute une paire d'alignement (du type $\begin{bmatrix} a_i \\ b_j \end{bmatrix}$, $\begin{bmatrix} a_i \\ - \end{bmatrix}$ ou $\begin{bmatrix} - \\ b_j \end{bmatrix}$) à chaque itération. Ici la hauteur de la colonne ne sera pas 2 mais n.
  La coût d'édition correspondant à une colonne ayant les lettres ou «−» $(a_1, a_2, \ldots, a_n)$ sera donné en vertu du modèle SP-score par :

$$d(a_1, a_2, \ldots, a_n) = \sum_{1\leq i<j\leq n} d(a_i, a_j)$$

où $d(a_i, a_j)$ est le coût d'édition habituel pour l'alignement de deux séquences.

Multiple alignment extension

La récurrence centrale de la programmation dynamique s'écrit :

$$C(\vec{i}) = max\left\{ C(\vec{i} - \vec{e}) + d\Big(f(e_1, S_1[i_1]), f(e_2, S_2[i_2]), \ldots, f(e_n, S_n[i_n]) \Big) \right\}$$

$$avec\ \ f(e_k, S_k[i_k]) = \begin{cases} S_k[i_k] & si\ \ e_k = 1 \\ \text{«−»} & sinon\ . \end{cases}$$

Le maximum est pris sur l'ensemble des vecteurs $\vec{e}$ de $\{0,1\}^n$ qui ont au moins une coordonnée non nulle.
Reformulation à l'aide d'un graphe orienté (S, A) **:**

- S : ensemble des vecteurs e de $\mathbb{N}^n$ avec $e_i \leqslant |S_i|$
- $\vec{I}_1 \to \vec{I}_2$ si $\forall i \in \{1, 2, \ldots, n\}$, $0 \leq I_2[i] - I_1[i] \leq 1$.
  Chaque sommet (dans l'intérieur du graphe) est atteint par $(2^n - 1)$ arcs.
  De chaque sommet partent $(2^n - 1)$ arcs.
- La valeur associée à un arc entre les sommets $\vec{I}_1$ et $\vec{I}_2$ est :

$$d\begin{pmatrix} f(\vec{I}_2[1] - \vec{I}_1[1], S_1[i_1]), \\ f(\vec{I}_2[2] - \vec{I}_1[2], S_2[i_2]), \\ \vdots \\ f(\vec{I}_2[n] - \vec{I}_1[n], S_n[i_n]) \end{pmatrix} \text{ avec } f(I[k], S_k[i_k]) = \begin{cases} S_k[i_k] & si\ \ I[k] = 1 \\ \text{«−»} & sinon\ . \end{cases}$$

- Problème : trouver les deux sommets tels que le chemin allant de l'un à l'autre ait un coût maximum.

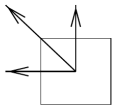
Complexité en $0(\Pi_{i=1}^n |S_i|)$ **L'algorithme** est en $O\Big((2^n - 1).\prod_{i=1}^n l_i\Big)$.

Memory space reduction : Hirschberg

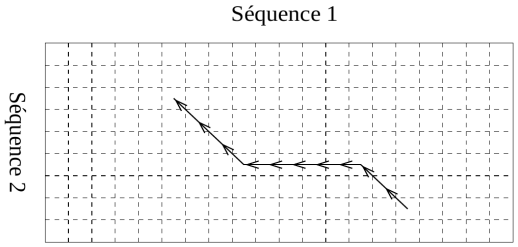
- La méthode de la programmation dynamique donne juste le score en $O(mn)$
- l'alignement n'est pas explicitement calculé : seuls le score la position de la fin de l'alignement sont donnés.

Méthode la plus simple pour calculer l'alignement :

- calculer toute la matrice des les scores intermédiaires
- à partir de la position de la fin de l'alignement :
  - retrouver la case qui a permis d'obtenir ce meilleur score.
  - stocker l'appariement associé
  - recommencer à partir de la nouvelle case



Pointeur vers la case ayant permis le score maximum



Phase de remontée SW

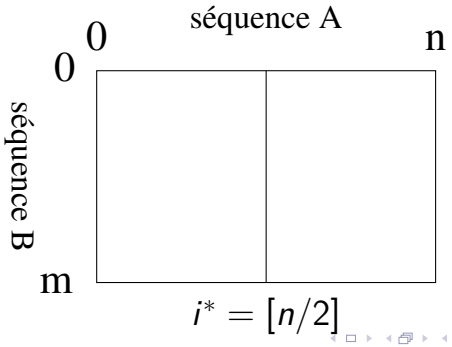
Memory space reduction : Hirschberg

- Complexité en espace : $O(m \times n)$
- Idée de Hirschberg : *diviser pour régner.*
  implémentée par Miller et Myers en 1988
- algorithme récursif qui divise la première séquence en deux, puis cherche le résidu de la 2de séquence qui permet d'optimiser les 2 alignements...

Soit deux séquences A et B de longueur respective n et m. L'algorithme se décompose en six phases :

1. On commence par diviser en deux la séquence A : on s'intéresse au résidu $i^* = \lfloor \frac{n}{2} \rfloor$

   Etape 1 :



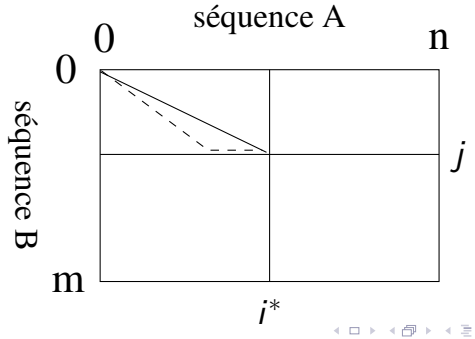
Memory space reduction : Hirschberg

- Complexité en espace : $O(m \times n)$
- Idée de Hirschberg : *diviser pour régner.*
  implémentée par Miller et Myers en 1988
- algorithme récursif qui divise la première séquence en deux, puis cherche le résidu de la 2de séquence qui permet d'optimiser les 2 alignements...

Soit deux séquences A et B de longueur respective n et m. L'algorithme se décompose en six phases :

2. on stocke 2 scores d'alignements de $A[1, i^*]$ avec $B[1, j]$:
   - les scores des chemins finissant par un appariement ou par insertion,
   - les scores des chemins finissant par une délétion.

   Etape 2 :



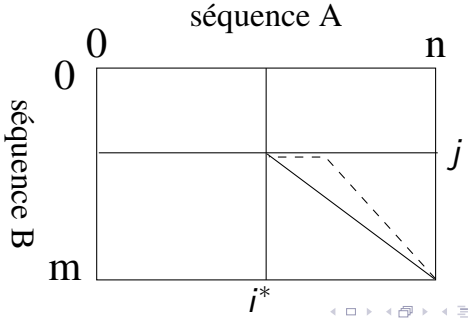
Memory space reduction : Hirschberg

- Complexité en espace : $O(m \times n)$
- Idée de Hirschberg : *diviser pour régner.*
  implémentée par Miller et Myers en 1988
- algorithme récursif qui divise la première séquence en deux, puis cherche le résidu de la 2de séquence qui permet d'optimiser les 2 alignements...

Soit deux séquences A et B de longueur respective n et m. L'algorithme se décompose en six phases :

3. on stocke 2 scores d'alignements de $A[i^* + 1, n]$ avec $B[j + 1, m]$:
   - les scores des chemins commençant par un appariement ou par une insertion,
   - les scores des chemins commençant par une délétion.

   Etape 3 :

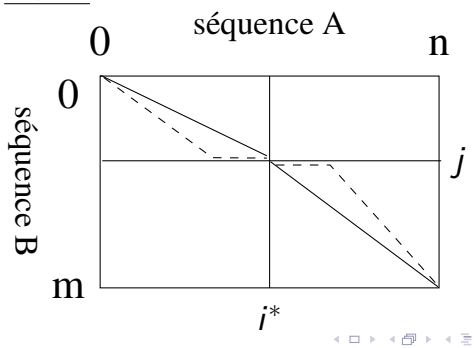


Memory space reduction : Hirschberg

- Complexité en espace : $O(m \times n)$
- Idée de Hirschberg : *diviser pour régner.*
  implémentée par Miller et Myers en 1988
- algorithme récursif qui divise la première séquence en deux, puis cherche le résidu de la 2de séquence qui permet d'optimiser les 2 alignements...

Soit deux séquences A et B de longueur respective n et m. L'algorithme se décompose en six phases :

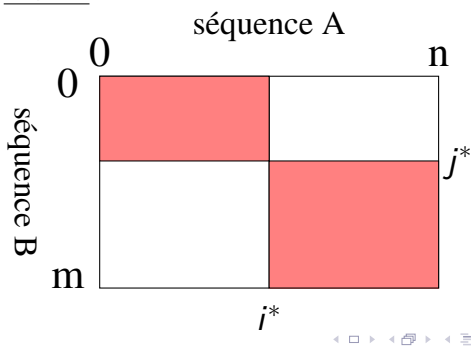
4. Le chemin optimal global passe forcément par la colonne i^*.
   Recherche du maximum parmi toutes les sommes entre un score de l'étape 2 et un score de l'étape 3. Soit j^* l'indice qui maximise cette somme.
   Etape 4 :



CÔTE D'AZUR
Algorithmics for Biology
Jean-Paul Comet
Complexity
Pat. Matching
Graphs
Dyn.Prog.
Sequences
2 seq. align.
N&W
S&W
Mult. Align.
Hirschberg
Memory space reduction : Hirschberg
- Complexité en espace : $O(m \times n)$
- Idée de Hirschberg : *diviser pour régner.*
  implémentée par Miller et Myers en 1988
- algorithme récursif qui divise la première séquence en deux, puis cherche le résidu de la 2de séquence qui permet d'optimiser les 2 alignements...

Soit deux séquences A et B de longueur respective n et m. L'algorithme se décompose en six phases :

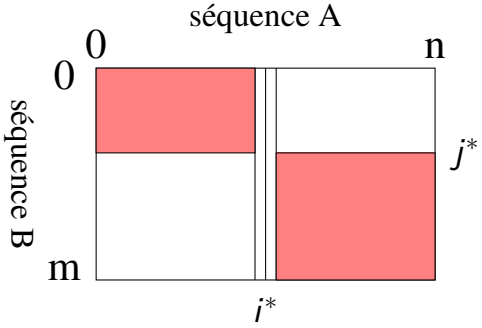
5. on recommence récursivement sur les sous-séquences $A[1, i^*]$ et $B[1, j^*]$ puis sur $A[i^*, n]$ et $B[j^*, m]$.
   On mémorise que la lettre $A[i^*]$ est alignée avec $B[j^*]$.
   Etape 5 :



CÔTE D'AZUR
Algorithmics for Biology
Jean-Paul Comet
Complexity
Pat. Matching
Graphs
Dyn.Prog.
Sequences
2 seq. align.
N&W
S&W
Mult. Align.
Hirschberg
Memory space reduction : Hirschberg
- Complexité en espace : $O(m \times n)$
- Idée de Hirschberg : *diviser pour régner.*
  implémentée par Miller et Myers en 1988
- algorithme récursif qui divise la première séquence en deux, puis cherche le résidu de la 2de séquence qui permet d'optimiser les 2 alignements...

Soit deux séquences A et B de longueur respective n et m. L'algorithme se décompose en six phases :

6. Si les résidus $A[i^*]$ et $A[i^*+1]$ sont dans un gap, on recommence sur les sous-séquences $A[1, i^*-1]$ et $B[1, j^*]$ puis sur $A[i^*+1, n]$ et $B[j^*, m]$.
   Etape 6 :



CÔTE D'AZUR
Algorithmics for Biology
Jean-Paul Comet
Complexity
Pat. Matching
Graphs
Dyn.Prog.
Sequences
2 seq. align.
N&W
S&W
Mult. Align.
Hirschberg
Memory space reduction : Hirschberg

- Pas à pas, on reconstruit l'alignement : à chaque étape, on stocke un appariement ou une délétion.
- L'espace nécessaire pour les étapes 5 et 6, peut être récupérer dans l'espace utilisé pour les étapes 1 à 4.
- la complexité en espace est $O(m)$. En effet, il faut l'espace pour mémoriser les scores intermédiaires (deux colonnes de taille m) et il faut mémoriser à chaque étape soit un appariement, soit une délétion.

Cet algorithme permet de réduire l'espace mémoire nécessaire pour le calcul de l'alignement, au prix d'une augmentation du temps d'exécution.

- à l'étape 5 : la zone hachurée est exactement la moitié de la zone initiale.
- à l'étape 6 : légèrement inférieure.

Le temps nécessaire pour déterminer les deux paires d'alignements suivantes est donc la moitié du temps nécessaire pour déterminer la première paire.

$$\text{Complexité totale :} \quad O\Big(m \times n \times \underbrace{(1 + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} \ldots)}_{A}\Big).$$

D'après l'identité remarquable :
$a^n - b^n = (a-b)(a^{n-1}b^0 + a^{n-2}b^1 + \ldots + a^{n-k-1}b^k + \ldots + a^0 b^{n-1})$
et puisque le nombre de découpage est $\sim k = log_2(n)$, le terme A devient :

$$A = \frac{1 - (\frac{1}{2})^k}{\frac{1}{2}} = 2\left(1 - (\frac{1}{2})^k\right) = 2\left(\frac{1}{2}\right)^k \left(2^k - 1\right) = \frac{2^k - 1}{2^{k-1}} < 2$$

cette méthode nécessite la connaissance des bornes de l'alignement local. Ces bornes sont calculables par l'exécution de l'algorithme de SW sur les séquences lues, en sens inverse, à partir des coordonnées de la fin de l'alignement.