Alignement de séquences biologiques

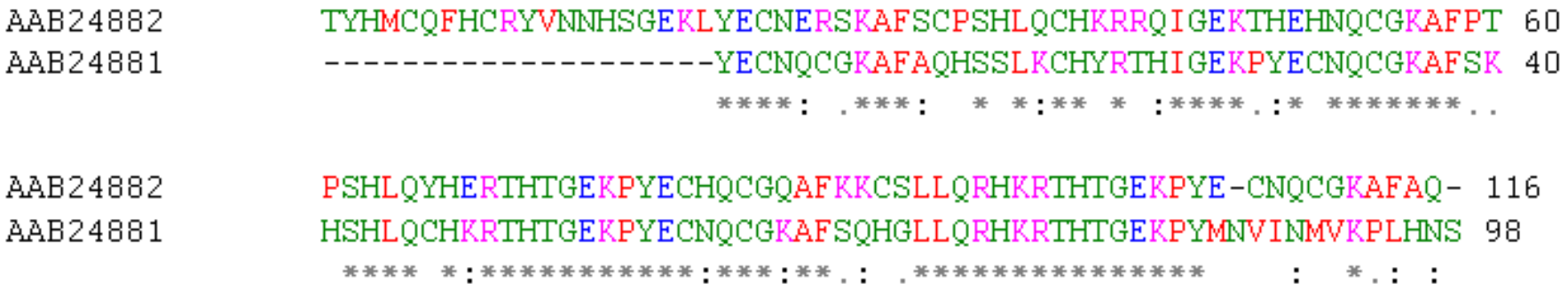
Nadia El-Mabrouk

Motivation

- Identification des gènes: Est-ce qu'un cadre de lecture est un gène? S'il existe un gène similaire dans un autre organisme, alors de fortes chances que la séquence corresponde à un gène « homologue ».
- Déduire la fonction d'un gène grâce à sa similarité avec un gène de fonction connue.
- Regrouper les gènes en familles d'homologues.
- Étudier l'évolution des gènes, des espèces.

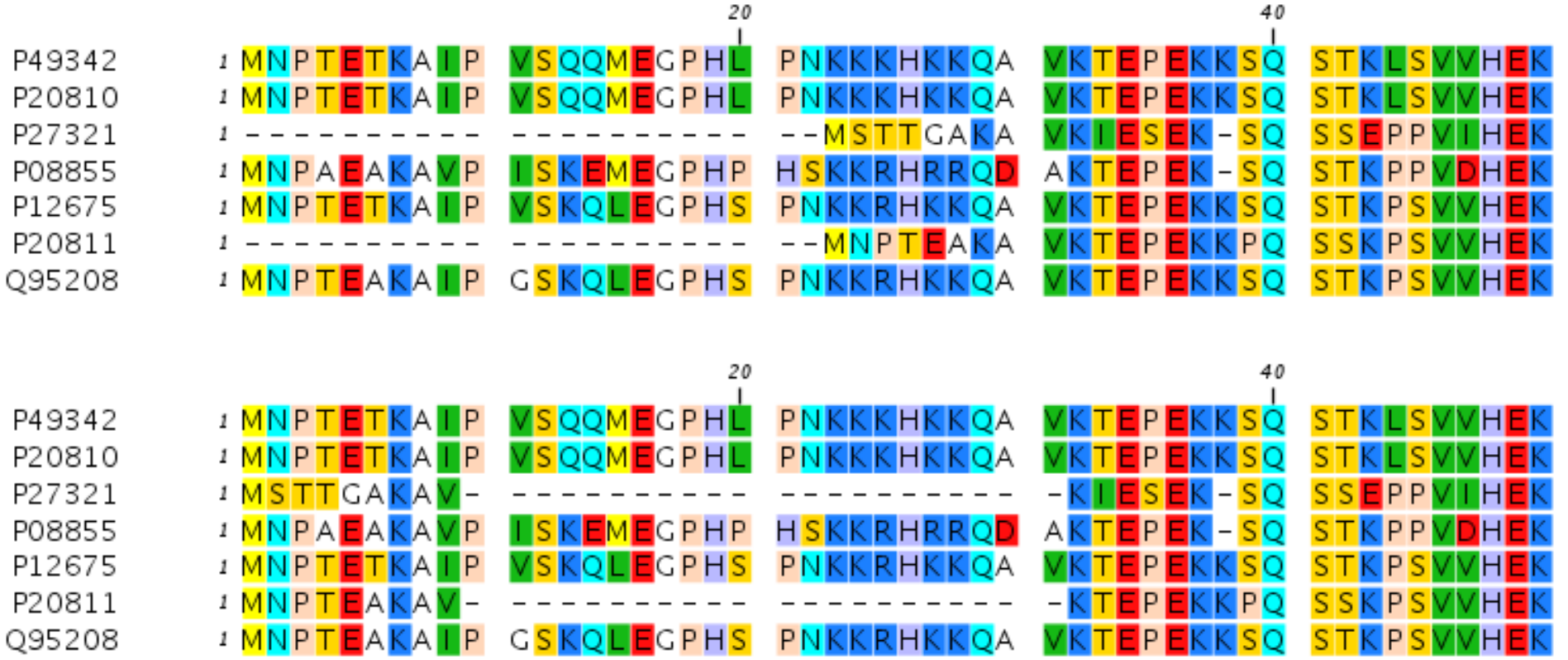
Exemple 1

- Un alignement de séquences réalisé par ClustalW entre deux protéines humaines.



http://fr.wikipedia.org/wiki/Alignement_de_s%C3%A9quences

Exemple 2



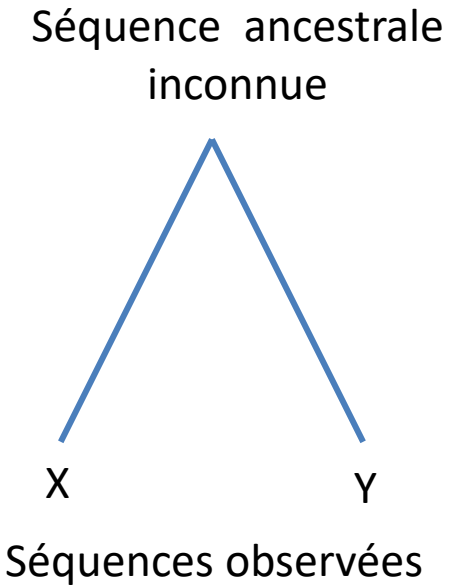
The first 50 positions of two different alignments of seven calpastatin sequences. The top alignment is made with cheap end gaps, while the bottom alignment is made with end gaps having the same price as any other gaps. In this case it seems that the latter scoring scheme gives the best result.

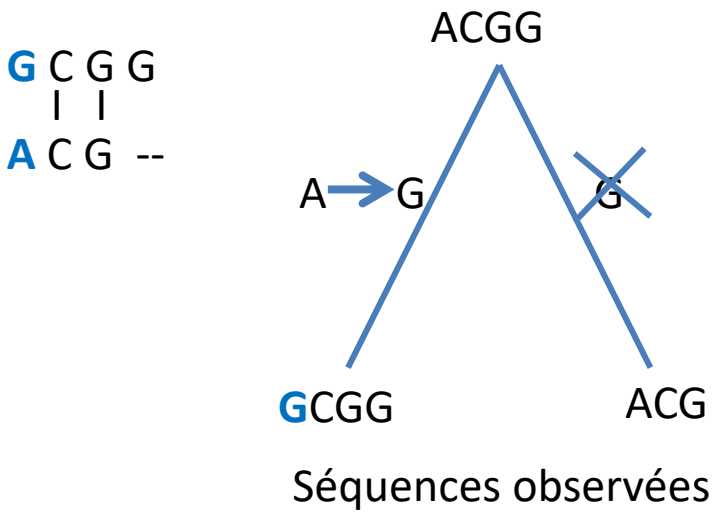
http://resources.qiagenbioinformatics.com/manuals/clcsequenceviewer/current/index.php?manual=_dup2Gap_costs.html

Modèle sous-jacent: mutations ponctuelles

Exemple:

Substitution de caractères





Alignement global/ local - Recherche

1. Alignment Global

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
  |  || |  ||  |  || |||||| || |  | |  | | ||   |
AATTGCCGCC-GTCGT--GTTCAGGGGTCA-GTTATG--T-CTGAT--C
```

2. Alignement local— trouver des régions conservées

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
               || |||||| || |
AATTGCCGCC-GTCGT--GTTCAGGGGTCA-GTTATG--T-CTGAT--C
```

3. Recherche – trouver la position d'un gène

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
               || |||||| ||
               GTTCAGGGGTCA
```

1 - Alignement global

Alignement sans indels

v : A T A A A T A T
w : T A T A T A T A

- Distance de Hamming : $d_H(v, w) = 7$

C'est beaucoup, bien que les séquences soient très similaires.

Alignement avec indels

En décalant d'une seule position:

v : A T A A A T A T --
w : -- T A T A T A T A

- Distance d'édition ou de Levenshtein (1966)

$$D(v, w) = 3.$$

- $D(v, w)$ = MIN d'insertions, suppressions, substitutions nécessaire pour transformer v en w

Alignement global

2 séquences v et w:

v : A T C T G A T C $m = 8$
w : T G C A T A G $n = 7$

Alignement : matrice 2 * k ($k \geq m, n$)

v	A	T	--	C	--	T	G	A	T	C
w	--	T	G	C	A	T	--	A	--	G

4 matches | 2 insertions | 3 suppressions | 1 mismatch

Distance d’édition versus Hamming

Dist. de Hamming compare toujours *i*-ème lettre de v et *i*-ème lettre de w

```
v = ATAAATAT
    | | | | | |
w = TATATATA
```

Dist. de Hamming: 7

Calculer distance de Hamming : **Trivial**.

Distance d'édition versus Hamming

Dist. de Hamming compare toujours *i*-ème lettre de v et *i*-ème lettre de w

```
v = ATAAATAT
    | | | | | | | |
w = TATATATA
```

Un seul shift et tout s'aligne

Dist. d'édition peut comparer *i*-ème lettre of v et *j*-ème lettre de w

```
v = - ATAAATAT
      | | | | | | |
w = TATATATA -
```

Dist. de Hamming:
7

Calculer Hamming distance
tâche: **Triviale**

Distance d'Édition:
3

Calculer dist. d'édition
tâche **non-triviale**

Dans la suite D(v,w) désigne la distance d'édition entre v et w

Alignement global

V:	A	T	--	C	--	T	G	A	T	C
W:	--	T	G	G	A	T	--	A	--	C

Un alignement de *v* et *w* est une matrice *D* de 2 lignes et *k* colonnes, avec $k \geq max(n,m)$ telle que

- Pour tous $1 \leq i \leq 2$ et $1 \leq j \leq k$, $D[i,j]$ est dans $\{A,C,G,T,-\}$;
- *v* (respectivement *w*) est obtenu en concaténant, dans l'ordre, les lettres $\{A,C,G,T\}$ de la 1ère (respec. la 2ème) ligne de *D*;
- Il n'existe aucune colonne j telle que $D[1,j]=D[2,j]=$"-".

Programmation dynamique

- Méthode algorithmique pour résoudre un problème d'optimisation – Ici: minimum d'opérations pour passer d'une séquence à l'autre.
- Introduite au début des années 1950 par Richard Bellman.
- **Consiste à résoudre un problème en le décomposant en sous-problèmes.** Problèmes emboités→ différent de diviser pour règner.
- **Chaque sous-problème est résolu une seule fois et mémorisé dans un tableau.**

Programmation dynamique

- **Relations de récurrence** : Définir, de manière récursive, la valeur d'une solution optimale.
- Calculer la valeur d'une solution optimale du plus petit au plus grand sous-problème
- Construire une solution optimale à partir des informations calculées (backtracking)

Alignement global, distance d'édition

2 séquences v et w:

v : A T C T G A T C $m = 8$
w : T G G A T A C $n = 7$

Trouver un alignement qui minimise indels+mismatchs

Alignement global, distance d'édition

- *D(i,j)* = MIN d'erreurs (substitutions, insersions, suppressions) entre *v[1,j]* et *w[1,i]*

Alignement global, distance d'édition

- *D(i,j)* = MIN d'erreurs (substitutions, insersions, suppressions) entre *v[1,j]* et *w[1,i]*
- *3 cas possibles:*
  1. V_j est impliqué dans un indel:

Alignement global, distance d'édition

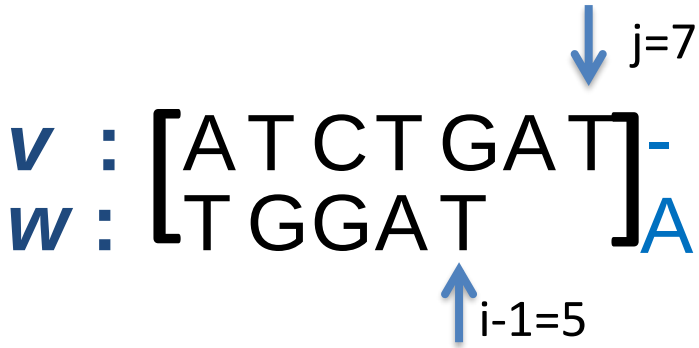
- *D(i,j)* = MIN d'erreurs (substitutions, insersions, suppressions) entre *v[1,j]* et *w[1,i]*
- *3 cas possibles:*
  1. V_j est impliqué dans un indel:

Alignement global, distance d'édition

- *D(i,j)* = MIN d'erreurs (substitutions, insersions, suppressions) entre *v[1,j]* et *w[1,i]*
- *3 cas possibles:*
  2. W_i est impliqué dans un indel:

Alignement global, distance d'édition

- *D(i,j)* = MIN d'erreurs (substitutions, insersions, suppressions) entre *v[1,j]* et *w[1,i]*
- *3 cas possibles:*
  2. W_i est impliqué dans un indel:



Alignement global, distance d'édition

- *D(i,j)* = MIN d'erreurs (substitutions, insersions, suppressions) entre *v[1,j]* et *w[1,i]*
- *3 cas possibles:*
  3. V_i *et* W_i sont alignés:

Alignement global, distance d'édition

- *D(i,j)* = MIN d'erreurs (substitutions, insersions, suppressions) entre *v[1,j]* et *w[1,i]*
- *3 cas possibles:*
  3. V_i *et* W_i sont alignés:

Alignement global, distance d'édition

- *D(i,j)* = MIN d'erreurs (substitutions, insersions, suppressions) entre *v[1,i]* et *w[1,j]*

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ \begin{cases} D(i-1, j-1) + 1 & \text{si } v_i \neq w_j \\ D(i-1, j-1) & \text{si } v_i = w_j \end{cases} \end{cases}$$

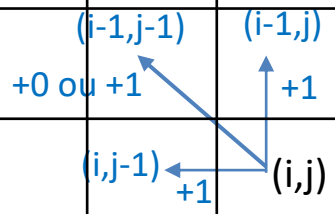
Conditions initiales: *D (i,0) =i; D(0,j) =j*

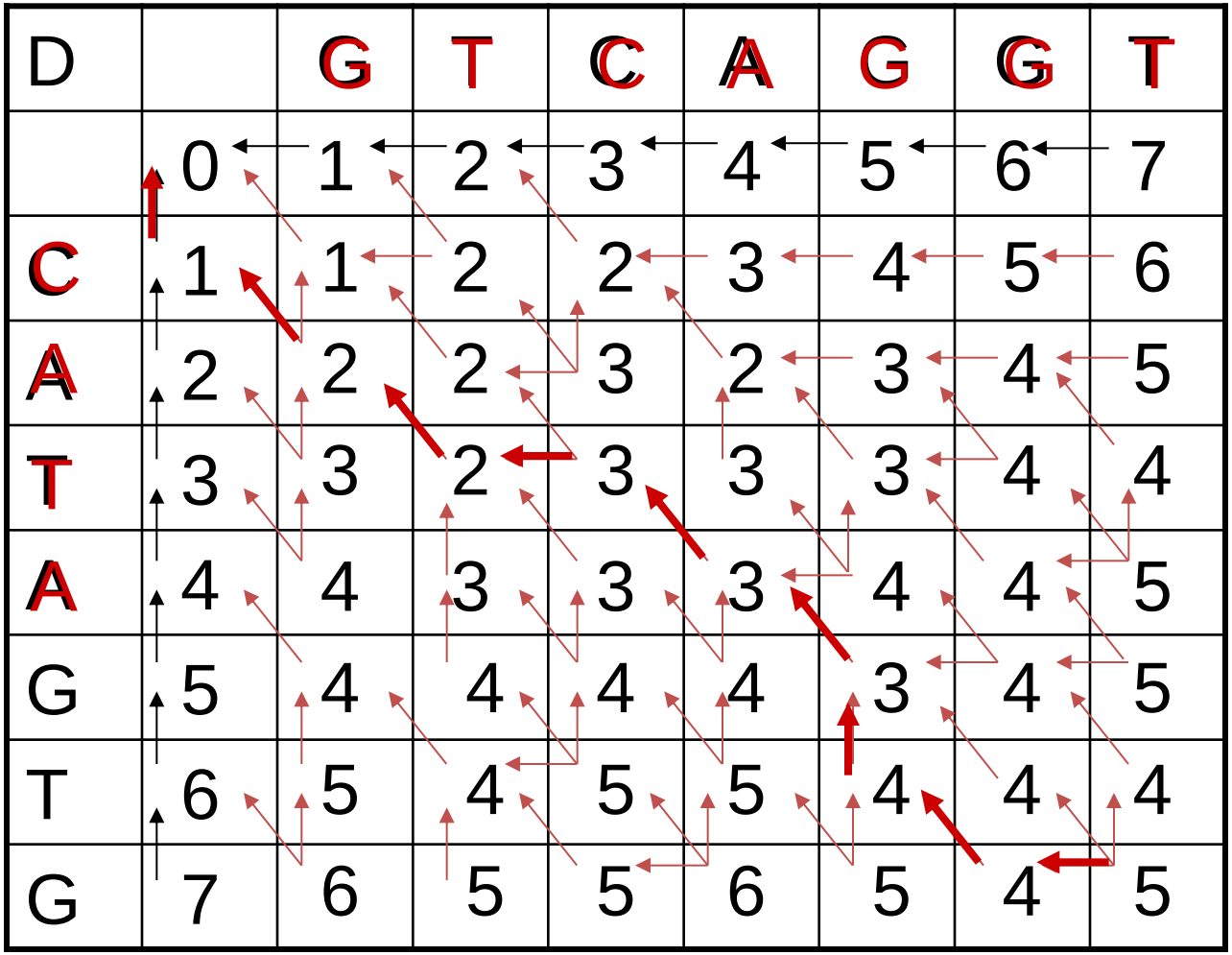
Table de programmation dynamique

j / i	0	A 1	T 2	C 3	T 4	G 5	A 6	T 7	C 8
0	0	1							
T 1	1	1							
G 2									
G 3									
A 4									
T 5									
A 6									
C 7									

Table de programmation dynamique

i \ j	0	A 1	T 2	C 3	T 4	G 5	A 6	T 7	C 8
0									
T_1									
G_2									
G_3									
A_4									
T_5									
A_6									
C_7									





C A T - A G T G -

- G T C A G - G T

Algorithme

distEdit(v,w)

```
for i ← 1 to n
    D(i,0) ← i
for j ← 1 to m
    D(0,j) ← j
for i ← 1 to n
    for j ← 1 to m
        D(i,j) ← min { D(i-1,j) +1
                     { D(i,j-1) +1
                     { { D(i-1, j-1) if v_i = w_j
                     { { D(i-1, j-1) + 1, if v_i ≠ w_j
```

$$b_{i,j} \leftarrow \begin{cases} \text{“}\leftarrow\text{“} & \textbf{if } D(i,j) = D(i-1,j) + 1 \\ \text{“}\uparrow\text{“} & \textbf{if } D(i,j) = D(i,j-1) + 1 \\ \text{“}\nwarrow\text{“} & \textbf{otherwise} \end{cases}$$

```
return (D(n,m), b)
```

Complexité

- Temps constant pour chaque chaque (i,j) avec $1 \leq i \leq n$ et $1 \leq j \leq m$
- Temps proportionnel à $O(nm)$ pour remplir la table de n lignes et m colonnes.
- Complexité en espace: également $O(nm)$.

Distance d'édition avec pondération des opérations

- On peut associer un score à chaque opération:
  - *d* pour une insertion/délétion
  - *r* pour une substitution
  - *e* pour un match
- *d>0, r>0 et e≥0*. En général *e=0*.
- Il faut que r<2d, sinon jamais de substitutions.
- Relations de récurrence:

$$D(i,0) = i \times d;\ D(0,j) = j \times d$$

$$D(i,j) = \min [D(i,j-1)+d,\ D(i-1,j)+d,\ D(i-1,j-1)+p(i,j)]$$

où $p(i,j) = e$ si $v_i = w_j$ et $p(i,j) = r$ sinon.

Distance d'édition généralisée

- Score δ qui dépend des caractères. Par exemple, remplacer une purine par une pyrimidine plus coûteux que remplacer une purine par une purine
- Relations de récurrence:

$$D(i,0) = \Sigma_{1 \le k \le i} \delta(v_k, -); \; D(0,j) = \Sigma_{1 \le k \le j} \delta(-, w_k)$$

$$D(i,j) = \min [D(i,j-1) + \delta(-, w_j), \; D(i-1,j) + \delta(v_i, -),$$
$$D(i-1,j-1) + \delta(v_i, w_j)]$$

- *Si δ est une distance, alors D est une distance* *(séparation, symétrie et inégalité triangulaire)*

Similarité entre deux séquences

- Plutôt que de mesurer la différence entre deux séquences, mesurer leur degré de similarité
- *P(a,b)*: score de l'appariement *(a,b):* Positif si *a=b* et *≤0* sinon. *V(i,j)*: valeur de l'alignement optimal de *v[1,i]* et *w[1,j]*
- Relations de récurrence:

$$V(i,0) = \sum_{1 \le k \le i} P(v_k, -); \; V(0,j) = \sum_{1 \le k \le j} P(-, w_k)$$

$$V(i,j) = \max [V(i,j-1) + P(-, w_j), \; V(i-1,j) + P(v_i, -), \; V(i-1,j-1) + P(v_i, w_j)]$$

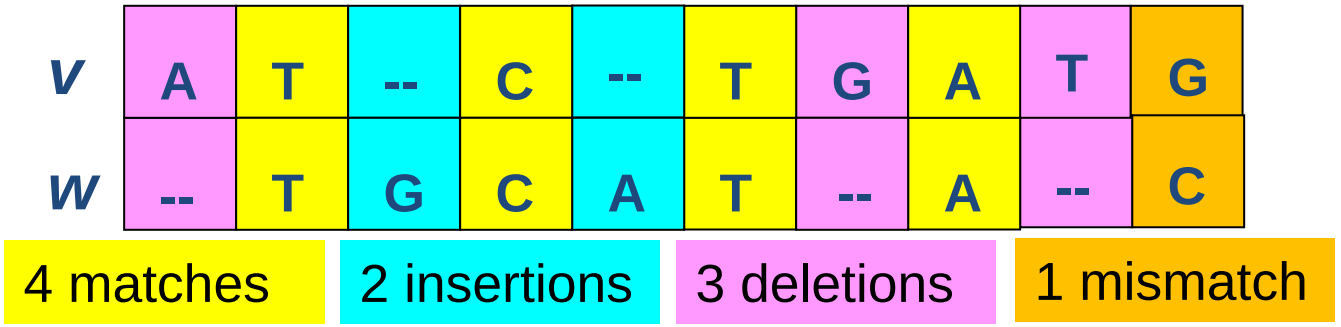
- *Ça s'appelle: Algorithme de Needleman-Wunch.*

Score simple

- Lorsque mismatches pénalisés par $-\mu$, indels pénalisés par $-\sigma$, et matches gratifiés d'un $+\varepsilon$, le score d'un alignement est:

  $\varepsilon(\#matches) - \mu(\#mismatches) - \sigma\ (\#indels)$

- **Exemple**: $\varepsilon = 2;\ \mu = \sigma = 1;$



Score = 2x4-1x6=2

Matrice de score pour les AA: Blosum50

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-2	-1	-1	-3	-1	1	0	-3	-2	0	-2	-1	-1	-5
R	-2	7	-1	-2	-4	1	0	-3	0	-4	-3	3	-2	-3	-3	-1	-1	-3	-1	-3	-1	0	-1	-5
N	-1	-1	7	2	-2	0	0	0	1	-3	-4	0	-2	-4	-2	1	0	-4	-2	-3	4	0	-1	-5
D	-2	-2	2	8	-4	0	2	-1	-1	-4	-4	-1	-4	-5	-1	0	-1	-5	-3	-4	5	1	-1	-5
C	-1	-4	-2	-4	13	-3	-3	-3	-3	-2	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1	-3	-3	-2	-5
Q	-1	1	0	0	-3	7	2	-2	1	-3	-2	2	0	-4	-1	0	-1	-1	-1	-3	0	4	-1	-5
E	-1	0	0	2	-3	2	6	-3	0	-4	-3	1	-2	-3	-1	-1	-1	-3	-2	-3	1	5	-1	-5
G	0	-3	0	-1	-3	-2	-3	8	-2	-4	-4	-2	-3	-4	-2	0	-2	-3	-3	-4	-1	-2	-2	-5
H	-2	0	1	-1	-3	1	0	-2	10	-4	-3	0	-1	-1	-2	-1	-2	-3	2	-4	0	0	-1	-5
I	-1	-4	-3	-4	-2	-3	-4	-4	-4	5	2	-3	2	0	-3	-3	-1	-3	-1	4	-4	-3	-1	-5
L	-2	-3	-4	-4	-2	-2	-3	-4	-3	2	5	-3	3	1	-4	-3	-1	-2	-1	1	-4	-3	-1	-5
K	-1	3	0	-1	-3	2	1	-2	0	-3	-3	6	-2	-4	-1	0	-1	-3	-2	-3	0	1	-1	-5
M	-1	-2	-2	-4	-2	0	-2	-3	-1	2	3	-2	7	0	-3	-2	-1	-1	0	1	-3	-1	-1	-5
F	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	1	-4	0	8	-4	-3	-2	1	4	-1	-4	-4	-2	-5
P	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	10	-1	-1	-4	-3	-3	-2	-1	-2	-5
S	1	-1	1	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	5	2	-4	-2	-2	0	0	-1	-5
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	2	5	-3	-2	0	0	-1	0	-5
W	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	1	-4	-4	-3	15	2	-3	-5	-2	-3	-5
Y	-2	-1	-2	-3	-3	-1	-2	-3	2	-1	-1	-2	0	4	-3	-2	-2	2	8	-1	-3	-2	-1	-5
V	0	-3	-3	-4	-1	-3	-3	-4	-4	4	1	-3	1	-1	-3	-2	0	-3	-1	5	-4	-3	-1	-5
B	-2	-1	4	5	-3	0	1	-1	0	-4	-4	0	-3	-4	-2	0	0	-5	-3	-4	5	2	-1	-5
Z	-1	0	0	1	-3	4	5	-2	0	-3	-3	1	-1	-4	-1	0	-1	-2	-2	-3	2	5	-1	-5
X	-1	-1	-1	-1	-2	-1	-1	-2	-1	-1	-1	-1	-1	-2	-2	-1	0	-3	-1	-1	-1	-1	-1	-5
*	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	1

Alignement global/ local - Recherche

1. Alignment Global

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
  |   || |   ||   |   || |||||| || |   | |   | | ||    |
AATTGCCGCC-GTCGT--GTTCAGGGGTCA-GTTATG--T-CTGAT--C
```

2. **Alignement local— trouver des régions conservées**

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
                  || |||||| || |
AATTGCCGCC-GTCGT--GTTCAGGGGTCA-GTTATG--T-CTGAT--C
```

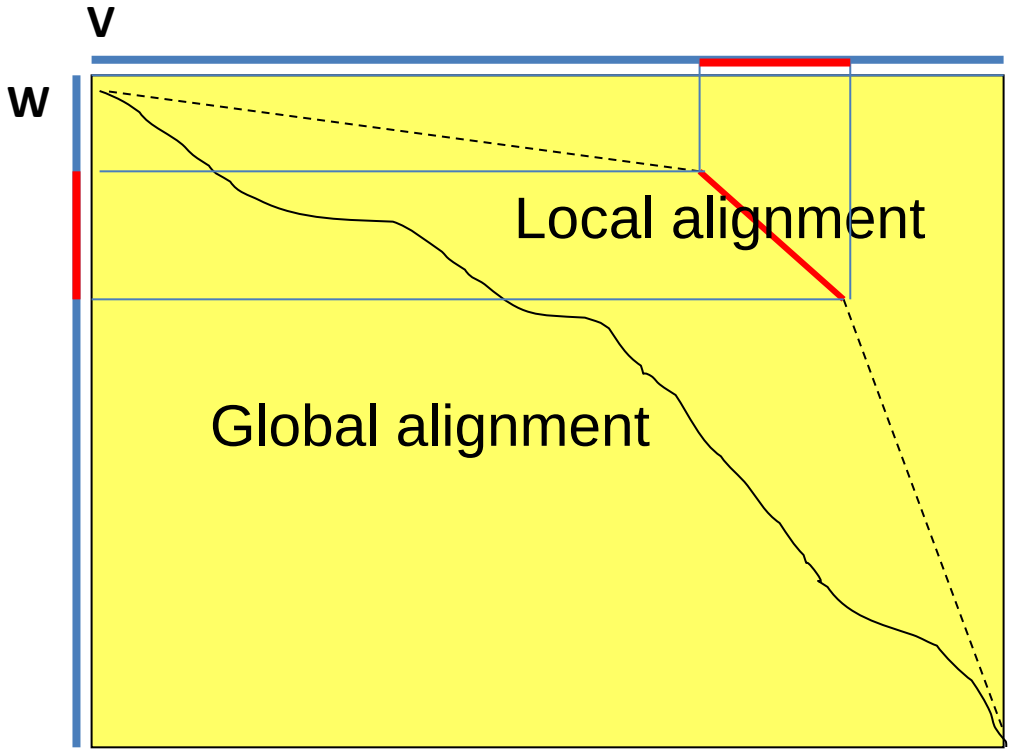
3. Recherche – trouver la position d'un gène

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
                  || |||||| ||
                  GTTCAGGGGTCA
```

2 - Alignement local

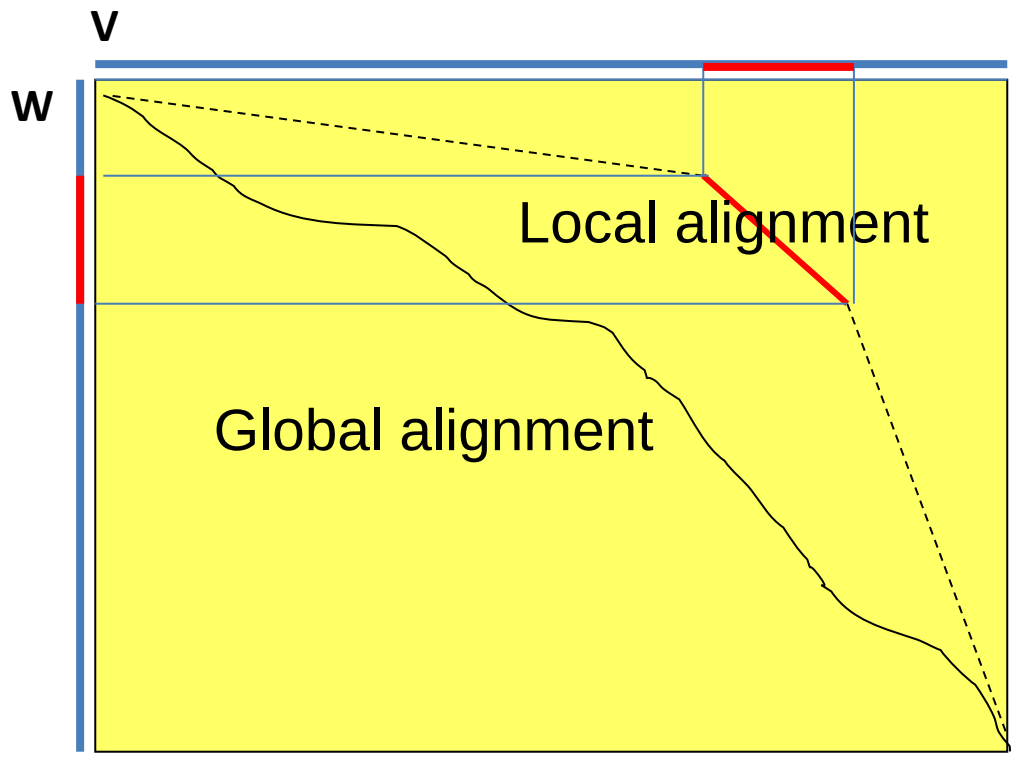
Input : Deux séquences **v, w** et une matrice de scores de similarité δ.

Output : Trouver deux facteurs de v et w dont le score de similarité est maximal parmi tous les facteurs possibles.

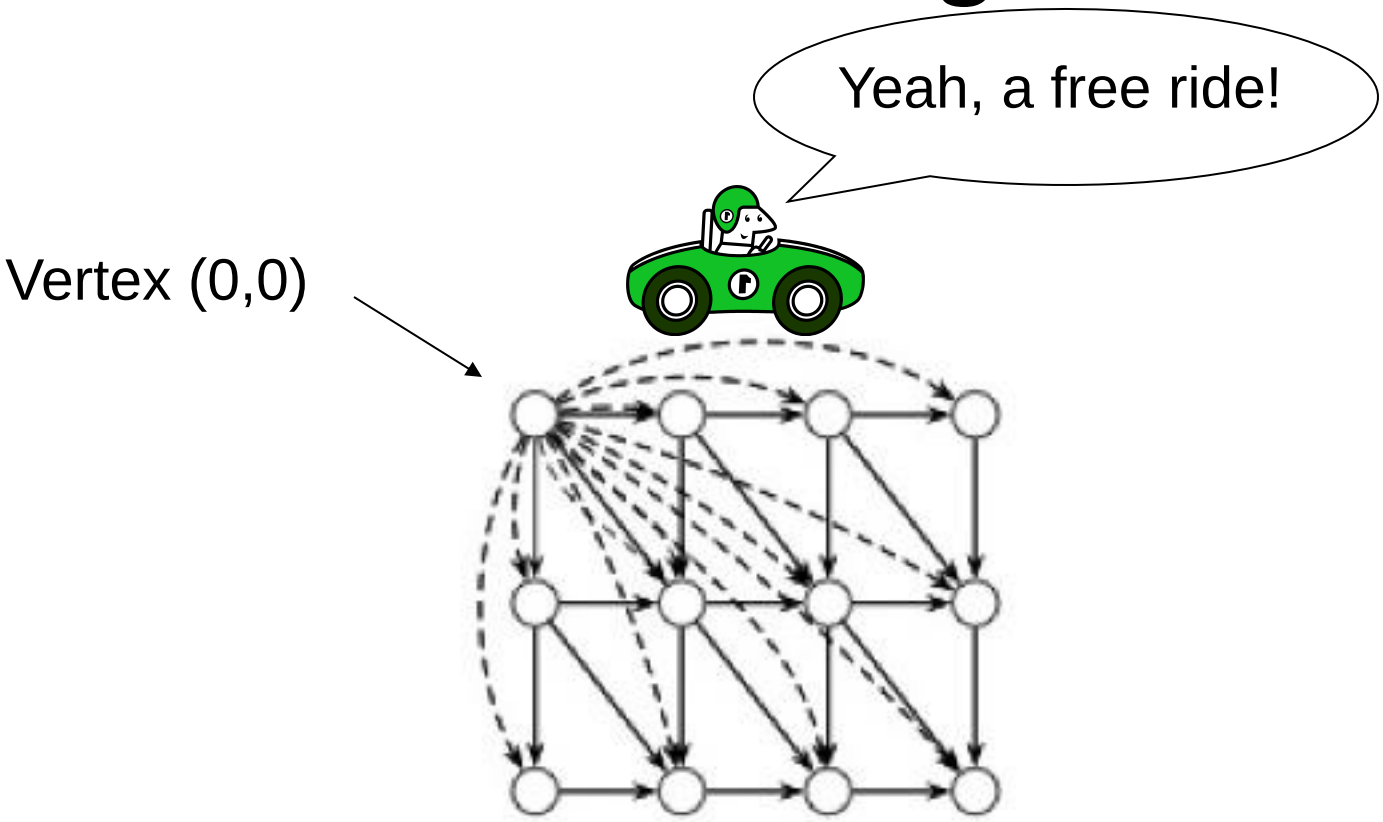


Solution directe

En temps $O(n^6)$: n^2 positions de début et n^2 positions de fin. Calculer la valeur de similarité maximale d'un chemin prend un temps $O(n^2)$.



Solution: Parcours gratuits



The dashed edges represent the free rides from (0,0) to every other node.

Alignement local: Récurrences

- La plus grande valeur *V(i,j)* est le score du meilleur alignement local.
- Récurrences:

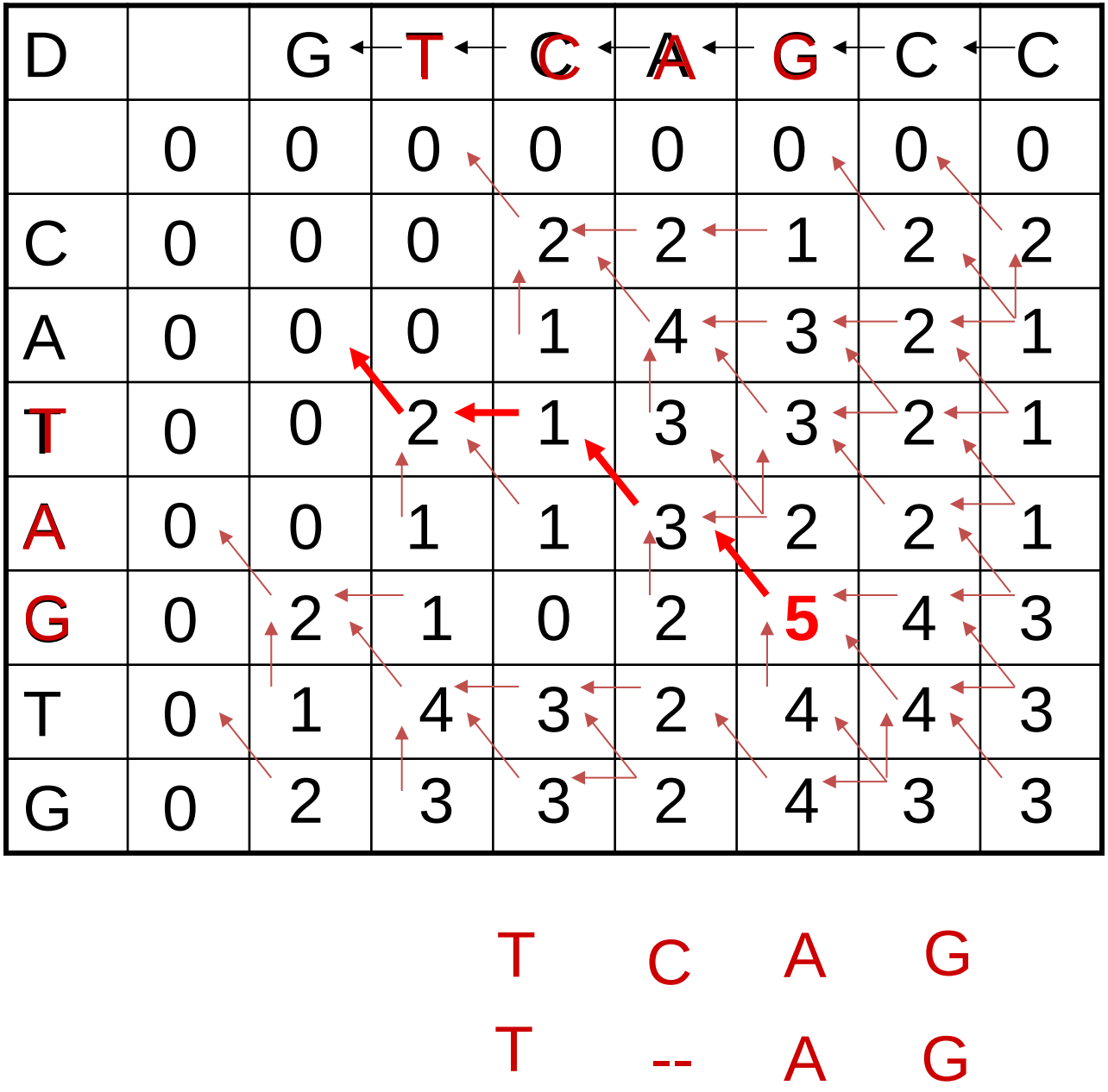
$$V(i,0) = V(0,j) = 0$$

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

Seules différences avec l'alignement global. Réinitialisation à 0. Possibilité d'arriver à chaque case par un parcours "gratuit"!

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

Match =2; Mismatch, indel = -1


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

Alignement local: Récurrences

- Récurrences:

$$V(i,0) = V(0,j) = 0$$

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

- Remplir la table de programmation dynamique
- Rechercher une case (i,j) contenant une valeur maximale.
- Démarrer à (i,j) et remonter les pointeurs jusqu'à tomber sur un 0.

Alignement global/ local - **Recherche**

1. Alignment Global

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
  |   || |   ||   |   || |||||| || |   | |   | | ||    |
AATTGCCGCC-GTCGT--GTTCAGGGGTCA-GTTATG--T-CTGAT--C
```

2. Alignement local— trouver des régions conservées

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
                 || |||||| || |
AATTGCCGCC-GTCGT--GTTCAGGGGTCA-GTTATG--T-CTGAT--C
```

3. **Recherche** – trouver la position d'un gène

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
                 || |||||| ||
                 GTTCAGGGGTCA
```

3 - Recherche

- Trouver toutes les occurrences d'un mot *P* dans un texte *T* d'une valeur supérieure à un certain score, ou d'un nombre d'erreurs inférieur ou égal à une certaine valeur *k*.
- Exemple: *T* = …GTCAGTTT…; *P* = GTT; k = 1.

```
          *   *           *   *   *
T:  G   T   C   A   G   T   T   T
```

Recherche

- *Distance d'édition; T* = GTCAGTTT; *P* = GTT; k = 1.

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

Recherche

- *Distance d'édition; T* = GTCAGTTT; *P* = GTT; k = 1.
  - *D[0,i] = 0*

D		G	T	C	A	G	T	T	T
	0	0	0	0	0	0	0	0	0
G									
T									
T									

Recherche

- *Distance d'édition; T* = GTCAGTTT; *P* = GTT; k = 1.
  - *D[0,j] = 0*
  - *D[i,0] = i*

D		G	T	C	A	G	T	T	T
	0	0	0	0	0	0	0	0	0
G	1								
T	2								
T	3								

Recherche

- *Distance d'édition; T* = GTCAGTTT; *P* = GTT; k = 1.
  - *D[0,j] = 0*
  - *D[i,0] = i*
  - *D[i,j] = min (D[i-1,j]+1, D[i,j-]+1, D[i-1,j-1]+e)*

    où e = 0 si P[i] = T[j] et e=1 sinon.

D		G	T	C	A	G	T	T	T
	0	0	0	0	0	0	0	0	0
G	1	0	1	1	1	0	1	1	1
T	2	1	0	1	2	1	0	1	1
T	3	2	1	1	2	2	1	0	1

Recherche

- *Distance d'édition; T* = GTCAGTTT; *P* = GTT; k = 1.
  - *Explorer la dernière ligne et retenir toutes les cases de valeur ≤ k.*

D		G	T	C	A	G	T	T	T
	0	0	0	0	0	0	0	0	0
G	1	0	1	1	1	0	1	1	1
T	2	1	0	1	2	1	0	1	1
T	3	2	**1**	**1**	2	2	**1**	**0**	**1**

Autres variantes possibles

C T T T C A C C -

C A T - C T T C T

- Alignement préfixe-suffixe

C T T T C A C C

C A T C C T T - C T

- Alignement suffixe-préfixe
- Alignement préfixe-préfixe

C T T T C A C C

C A T - C T T T C T

- Alignement suffixe-suffixe

4 - Pondération des indels

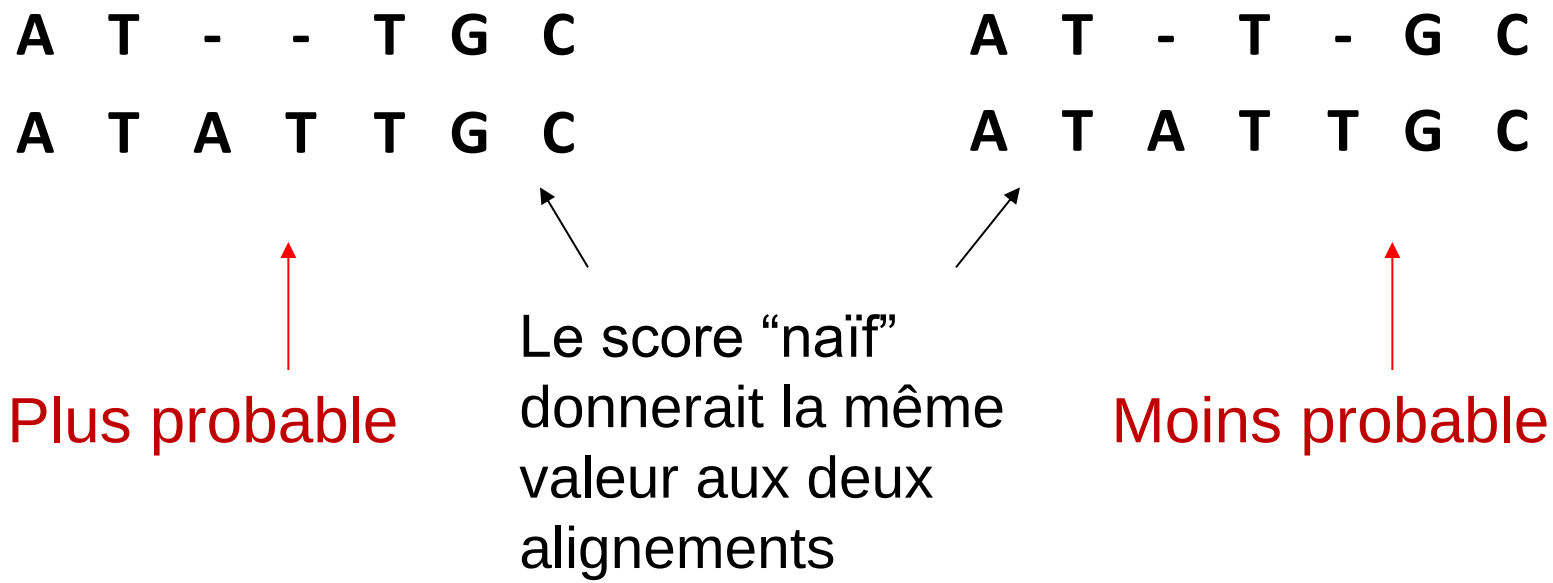
Approche naïve

- Un score de pénalité σ pour chaque indel:
  - $-\sigma$ pour 1 indel,
  - -2σ pour 2 indels consécutifs
  - -3σ pour 3 indels consécutifs, etc.

Peut être trop sévère pour une suite de 100 indels consécutifs

Considérer les gaps

- Des indels consécutifs dans les séquences biologiques sont plutôt dus à un événement unique:



Considérer les gaps

- *Gaps*: séquence d'indels consécutifs sur une ligne de l'alignement

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

- Contient 7 indels, mais seulement 4 gaps.
- Score particulier pour les gaps: influence la distribution des indels.

Pondération constante

- Score d'un gap indépendant de sa taille: pénalité constante ρ.
- Score d'un alignement entre v et w contenant k gaps $\Sigma_{1 \le i \le \tau} \delta(v_i, w_j) - k\rho$

Exemple:

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

score = $3\delta(C,C) + 2\delta(A,A) + \delta(A,C) - 4\rho$

Pondération affine

- Pénalités pour les gaps:
  - $-\rho-\sigma$ pour 1 indel
  - $-\rho-2\sigma$ pour 2 indels
  - $-\rho-3\sigma$ pour 3 indels, etc.
- ρ : pénalité d'ouverture d'un gap

  σ : pénalité d'extension.
- Score d'un gap de taille t: $-\rho - t.\sigma$
- Score d'un alignement de taille l contenant k gaps et q indels: $\Sigma_{1\leq i \leq l}\delta(v_i,w_j)-k\rho - q\sigma$
- Permet une pénalité réduite pour les grands gaps.
- Exemple:

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

$$\text{score} = 3\delta(C,C) + 2\delta(A,A) + \delta(A,C)-4\rho-7\sigma$$

Autres pondérations

- Pondération convexe: chaque indel supplémentaire est moins pénalisé que le précédent.

  Exemple: score d'un gap de taille t: $-\rho - log_e(t)$

- Pondération quelconque ω:

  Fonction quelconque de la taille du gap.

  Exemple: score d'un gap de taille t: $-\omega(t)$

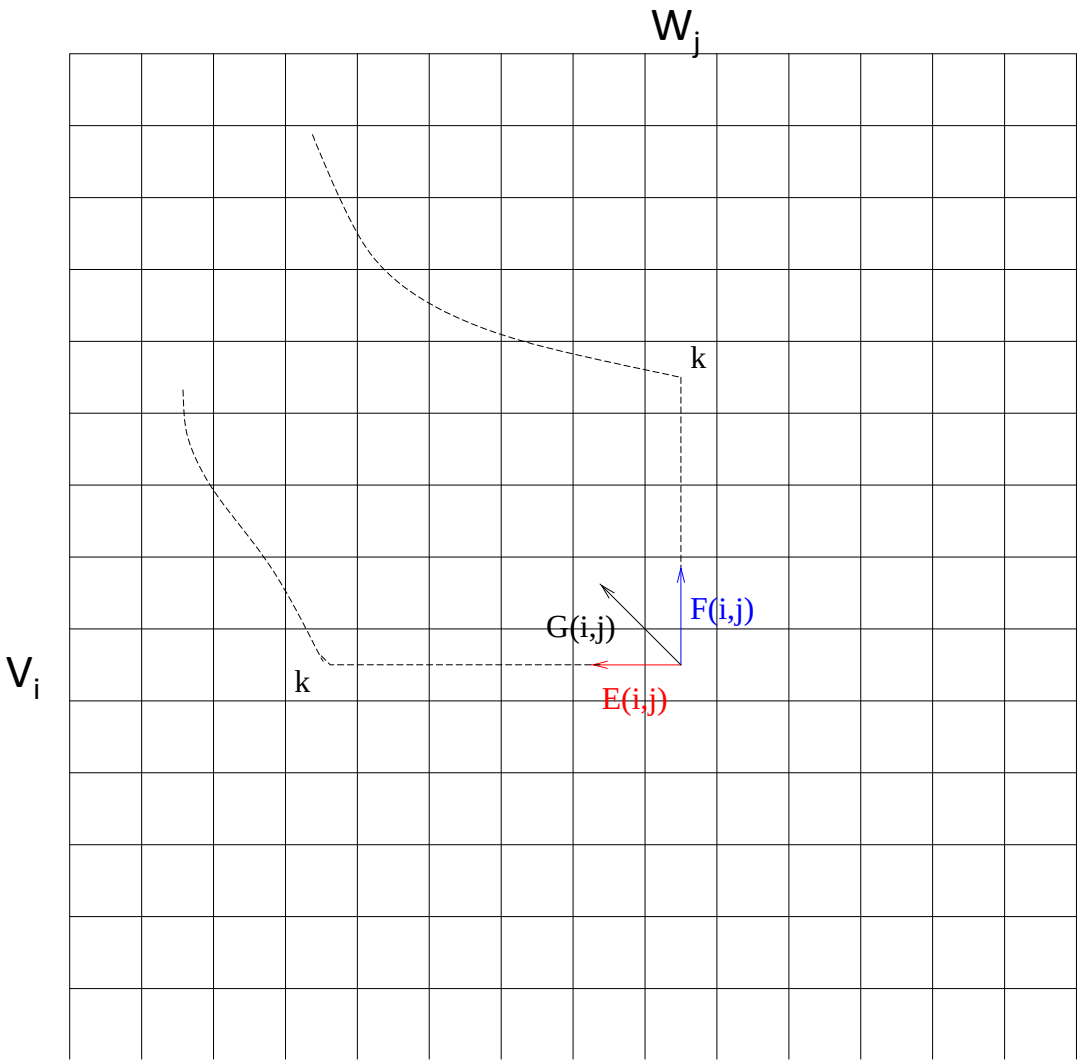
Alignement avec gap – Pondération quelconque

- Trois alignements possibles de *v[1,i]* avec *w[1,j]*:
  1. Alignement de *v[1,i]* avec *w[1,j-1]* suivit de $(-, w_j)$
  2. Alignement de *v[1,i-1]* avec *w[1,j]* suivit de $(v_i, -)$
  3. Alignement de *v[1,i-1]* avec *w[1,j-1]* suivit de (v_i, w_j)

- *E(i,j):* Valeur maximale d'un alignement de type 1.
- *F(i,j):* Valeur maximale d'un alignement de type 2.
- *G(i,j):* Valeur maximale d'un alignement de type 3.

$$V(i,j) = \max [E(i,j), F(i,j), G(i,j)]$$

Alignement avec gap – Pondération quelconque



Alignement global avec gap – Pondération quelconque

- Conditions initiales:
  - $V(i,0) = F(i,0) = -\omega(i)$
  - $V(0,j) = E(0,j) = -\omega(j)$
- Relations de récurrence:
  - $G(i,j) = V(i-1,j-1) + \delta(v_i, w_j)$
  - $E(i,j) = max_{0 \le k \le j-1}[V(i,k) - \omega(j-k)]$
  - $F(i,j) = max_{0 \le k \le i-1}[V(k,j) - \omega(i-k)]$
- Valeur optimale: $V(m,n)$
- Complexité: $O(m^2n+mn^2)$

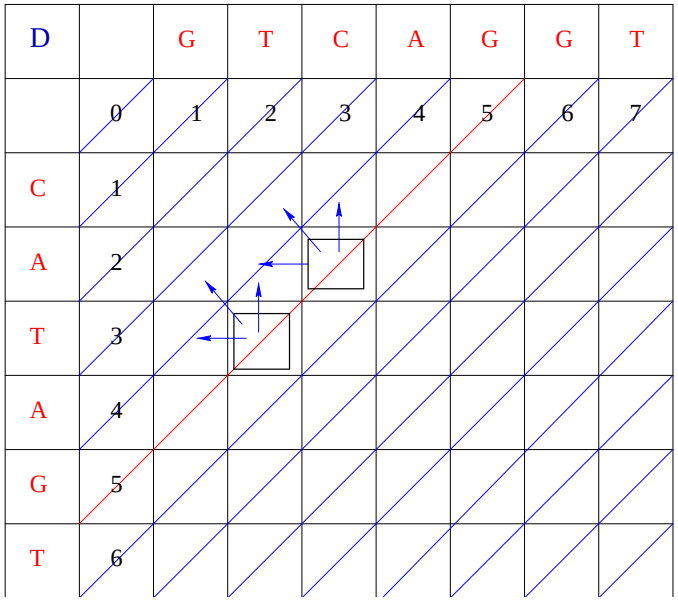
Alignement avec gap – Pondération affine

- Conditions initiales:
  - $V(i,0) = F(i,0) = -\rho - i.\sigma$
  - $V(0,j) = E(0,j) = -\rho - j.\sigma$
- Relations de récurrence:
  - $G(i,j) = V(i-1,j-1) + \delta(v_i, w_j)$
  - $E(i,j) = max[\ E(i,j-1),\ V(i,j-1) - \rho\ ] - \sigma$
  - $F(i,j) = max[\ F(i-1,j),\ V(i-1,j) - \rho\ ] - \sigma$
- Complexité: $O(mn)$

5 - Parallélisme

- Table de programmation dynamique pour l'alignement: Pour calculer une case (i,j), on a besoin des 3 cases voisines (i-1,j), (i-1,j-1), (I,j-1)

  → Remplissage ligne par ligne, colonne par colonne, ou anti-diagonale par anti-diagonale:



Parallélisme

- Pour chaque anti-diagonale *k*, on a besoin des anti-diagonales *k-1* et *k-2*.
- Observation clef: Chaque case d'une anti-diagonale *k* est calculée indépendamment des autres cases de l'anti-diagonale *k*

  → Un processeur peut-être assigné au calcul de chaque case
- Complexité en temps: O(n)

Références

- ***An introduction de Bioinformatics Algorithms***, Neil C. Jones and Pavel A. Pevzner, The MIT Press, 2004.
- ***Algorithms on Strings, Trees and Sequences – Computer science and Computational biology***, Dan Gusfield, Cambridge University Press, 1997.
- **Handbook of Computational Molecular Biology**, Srinivas Aluru ed., Chapman & Hall/CRC Computer and Information Science Series, 2005.
- **Pour l'alignement local normalisé:**
  - A.N. Arslan, O. Egecioglu and P.A. Pevzner. *A new approach to sequence comparison: normalized sequence alignment,* Bioinformatics, 17 (4): 327-337, 2001.

(Supplément): Alignement local normalisé

- Un problème avec l'alignement local est que la taille des deux séquences *S* (taille *n*) et *T* (taille *m*) (n>m) comparées n'est pas pris en compte :
  - Un alignement de taille 100 et de valeur 51 est considéré meilleur qu'un alignement de taille 50 et de valeur 50, alors que le second alignement a une valeur moyenne par base beaucoup plus élevée.
- Solutions peu satisfaisantes:
  - Procéder à un post-traitement → Un alignement long peut masquer un alignement plus court mais plus significatif
  - Calculer l'alignement global entre chaque paire de sous-séquences de S et T → $O(n^2m^2)$ combinaisons possibles, d'où une complexité en $O(n^3m^3)$

Alignement local normalisé

Pevzner et al. : algorithme en *O(nm.log(n))*

A.N. Arslan, O. Egecioglu and P.A. Pevzner. A new approach to sequence comparison: normalized sequence alignment. Bioinformatics, 17 (4): 327-337, 2001.

- La valeur d'un alignement local optimal *LA(S',T')* entre deux sous-séquences *S'* et *T' de S et T* est $a\alpha+b\beta+g\gamma$ où:
  - *a*: nbre de match, *b*: nbre de mismatch; *g*: nbre d'indel.

$$LA(S,T) = max_{(S',T')}\ a\alpha+b\beta+g\gamma$$

- On a nécessairement *n'+m' = 2a+2b+g où n' est la taille de S' et m' la taille de S'.*
- Pevzner propose de mesurer la taille de l'alignement comme *n'+m'+L* où *L* est une constante positive.

Alignement local normalisé

- L'alignement local normalisé optimal est:

$$NLA(S,T) = \max_{(S',T')} (a\alpha+b\beta+g\gamma) / (2a+2b+g+L)$$

- Définissons l'alignement local paramétré pour un certain paramètre λ:

$$PA(S,T, \lambda) = \max_{(S',T')} (a\alpha+b\beta+g\gamma) - \lambda(2a+2b+g+L)$$

- *Résultat de W. Dinkelbach :*

« On nonlinear fractional programming ». Management Science, 13: 492-498, 1967

$$\lambda = NLA(S,T) \text{ ssi } PA(S,T, \lambda) = 0$$

Autrement dit, λ est le score de l'alignement local normalisé optimal ssi l'alignement local paramétré par λ est de score 0.

Alignement local normalisé

- Dinkelbach propose une méthode de recherche itérative pour trouver le 0 de *PA(S,T, λ)* (complexité inconnue mais fonctionne bien en pratique). D'abord, Initialiser λ en trouvant l'alignement local *LA(S,T)* et en posant

$$\lambda = (a\alpha+b\beta+g\gamma) / (2a+2b+g+L).$$

Répéter ensuite les deux étapes suivantes jusqu'à ce que λ ne change plus:

  - Trouver *PA(S,T, λ)* → *donne un vecteur (a,b,g)*
  - Poser λ'= $(a\alpha+b\beta+g\gamma) / (2a+2b+g+L)$, *et donner à* λ la valeur de λ'

- L'étape clef dans l'algorithme est de trouver *PA(S,T, λ). Or, le problème peut être exprimé* sous forme d'un problème d'alignement local de la façon suivante:

$$PA(S,T, \lambda) = (max_{(S',T')} a(\alpha-2\lambda)+b(\beta-2\lambda)+g(\gamma-\lambda))-L\lambda$$

Alignement local normalisé

- *Il suffit de résoudre l'alignement local avec $\alpha'=\alpha-2\lambda$, $\beta'=\beta-2\lambda$ et $\gamma'=\gamma-\lambda$*

Cette résolution de PA(S,T, λ) prend un temps O(nm). Par ailleurs, si α, β, γ sont des nombres rationnels, alors le temps total de l'algorithme est en O(nm. log(n))

- *Utilisation de la technique de N. Megiddo*

Combinatorial optimization with rational objective functions. Mathematics of Operations Research, 4(4): 414-424, 1979.