# Alignement de séquences biologiques

Nadia El-Mabrouk

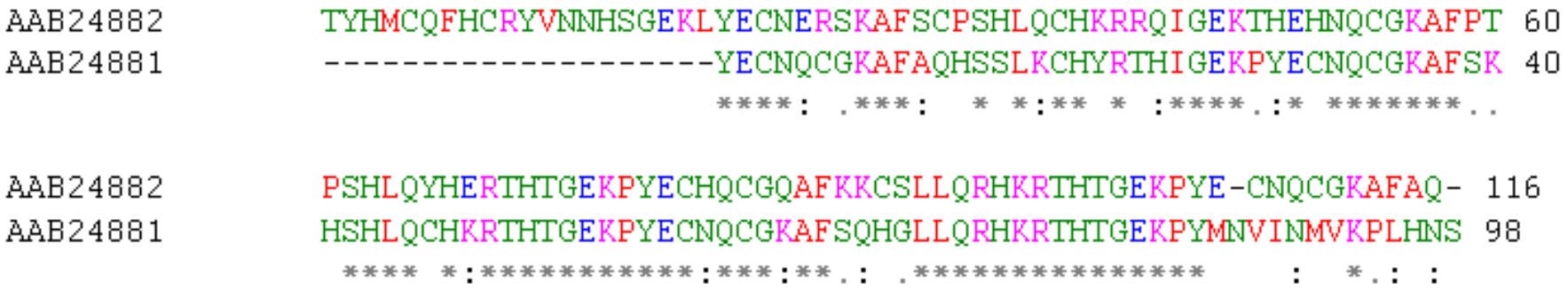
Inspiré de:
An introduction de Bioinformatics Algorithms – www.bioalgorithms.info
Neil C. Jones and Pavel A. Pevzner

# 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.

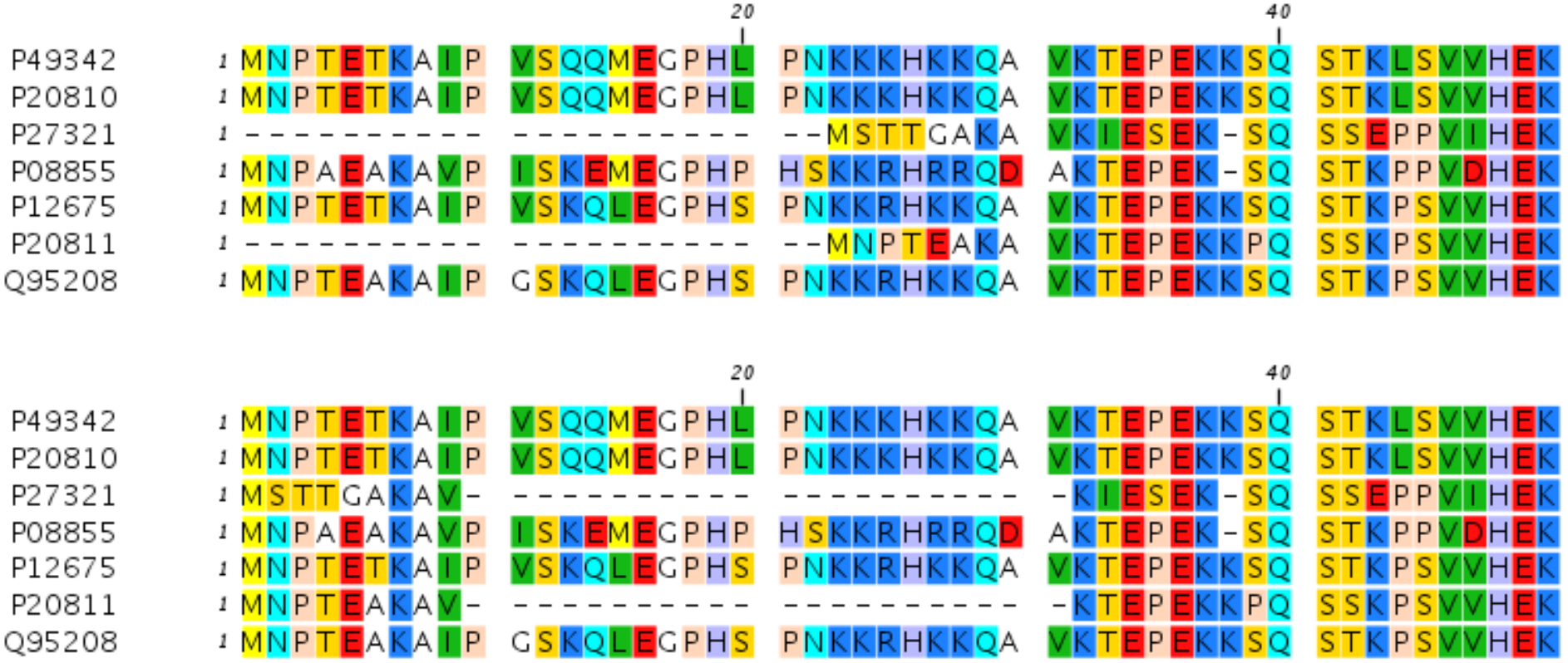
# Exemple 1

- 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*

# Alignement global/ local - Recherche

- Alignment Global

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

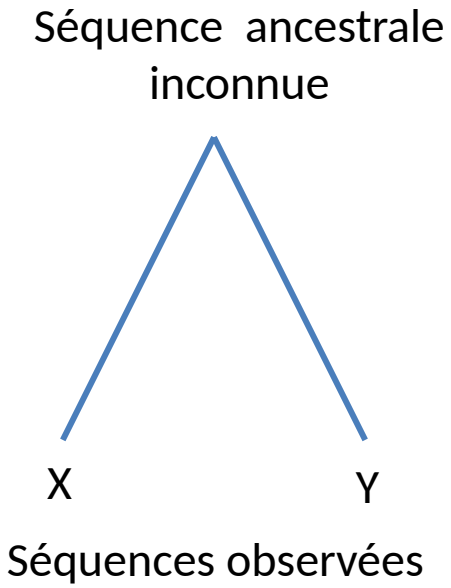
- Alignement local— trouver des régions conservées

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

- Recherche – trouver la position d'un gène

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

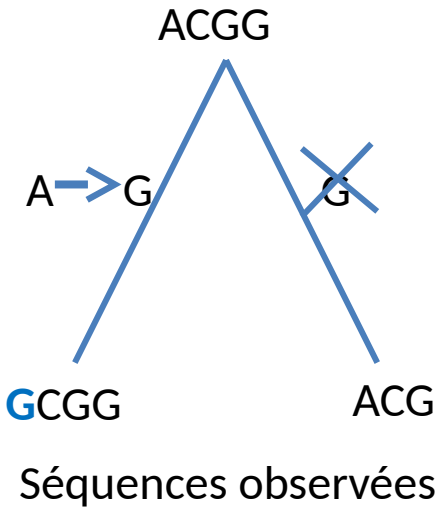
# Modèle sous-jacent: mutations ponctuelles




Exemple:

Substitution de caractères

G C G G
| |
A C G --



# 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

# Distance d'édition versus Hamming

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

$\mathbf{v}$ = ATAAATAT

$\mathbf{w}$ = TATATATA

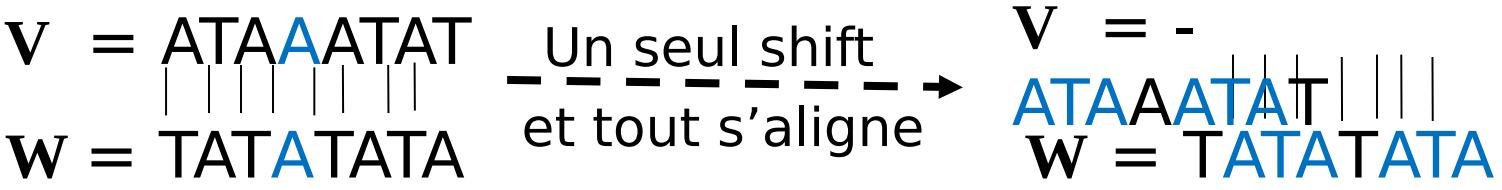
Dist. de Hamming: 7

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

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




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 *A*;
- Il n'existe aucune colonne j telle que $D[1,j]=D[2,j]=$"-".

# Alignement global

2 séquences v et w:

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

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

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ***v*** | **A** | **T** | **--** | **C** | **--** | **T** | **G** | **A** | **T** | **C** |
| ***w*** | **--** | **T** | **G** | **G** | **A** | **T** | **--** | **A** | **--** | **C** |

4 matches | 2 insertions | 3 suppressions | 1 mismatch

**Comment aligner de telle sorte à minimiser la distance d'édition?**

# 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

- Caractériser la structure d'une solution optimale: Qu'est-ce qu'on minimise?
- **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

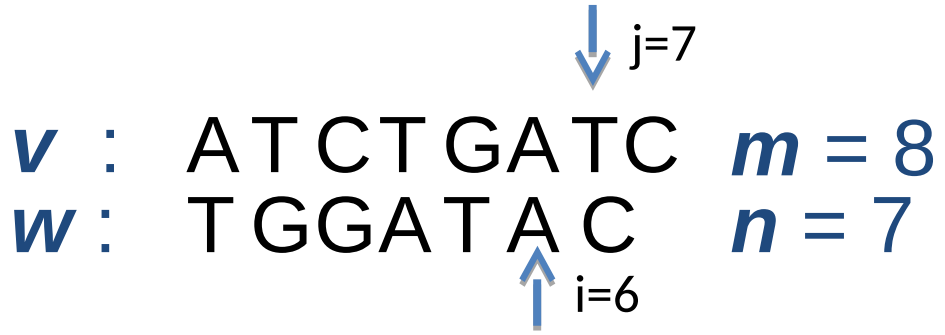
2 séquences v et w:

$v$ : 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]*

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

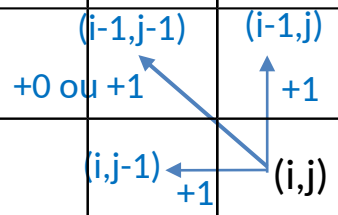
**Conditions initiales**: *D (i,0) =i; D(0,j) =j*

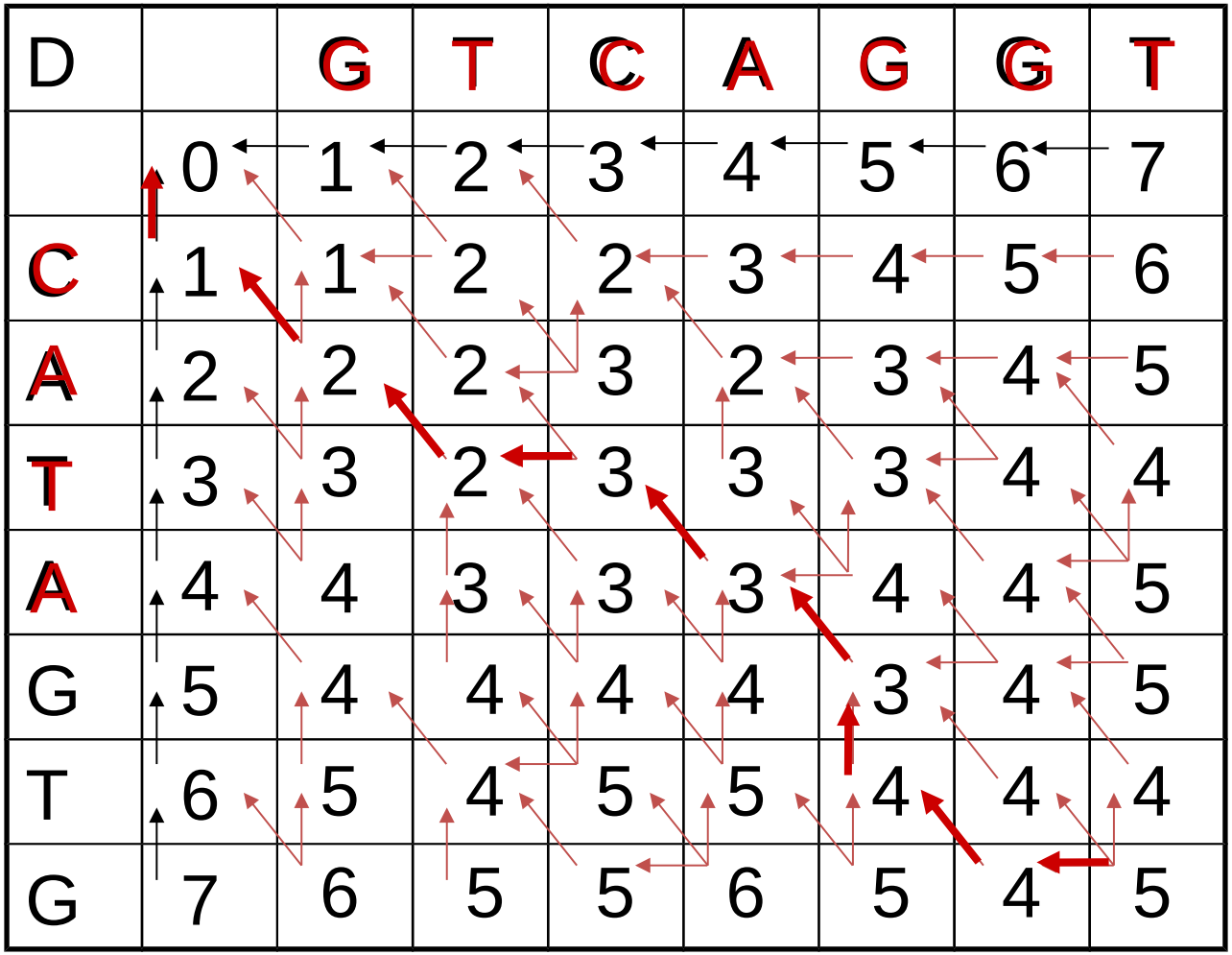
# Table de programmation dynamique

| i \ j | 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

| j / i | 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 $m$

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

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

**return** $(D(n,m), \boldsymbol{b})$

# 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 $\delta$ qui dépend des caractères. Par exemple, remplacer une purine par une pyrimidine plus coûteux que remplacer une purine par une purine
- Relations de récurrence:

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

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

$$D(i-1,j-1) + \delta(v_i, w_j)]$$

- *Si $\delta$ 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:

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

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

$$V(i-1,j-1) + P(v_i, w_j)]$$

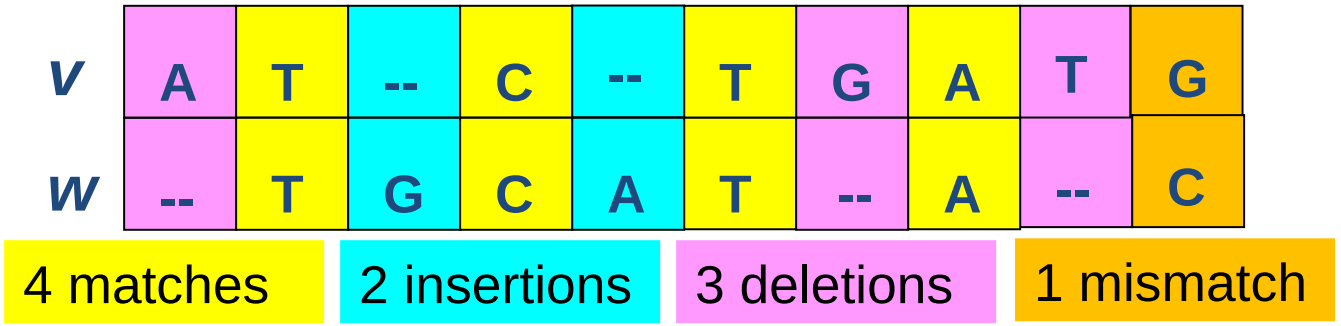
  - *Ça s'appelle: Algorithme de Needleman-Wunch.*

# 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

- Alignment Global

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

- Alignement local— trouver des régions conservées

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

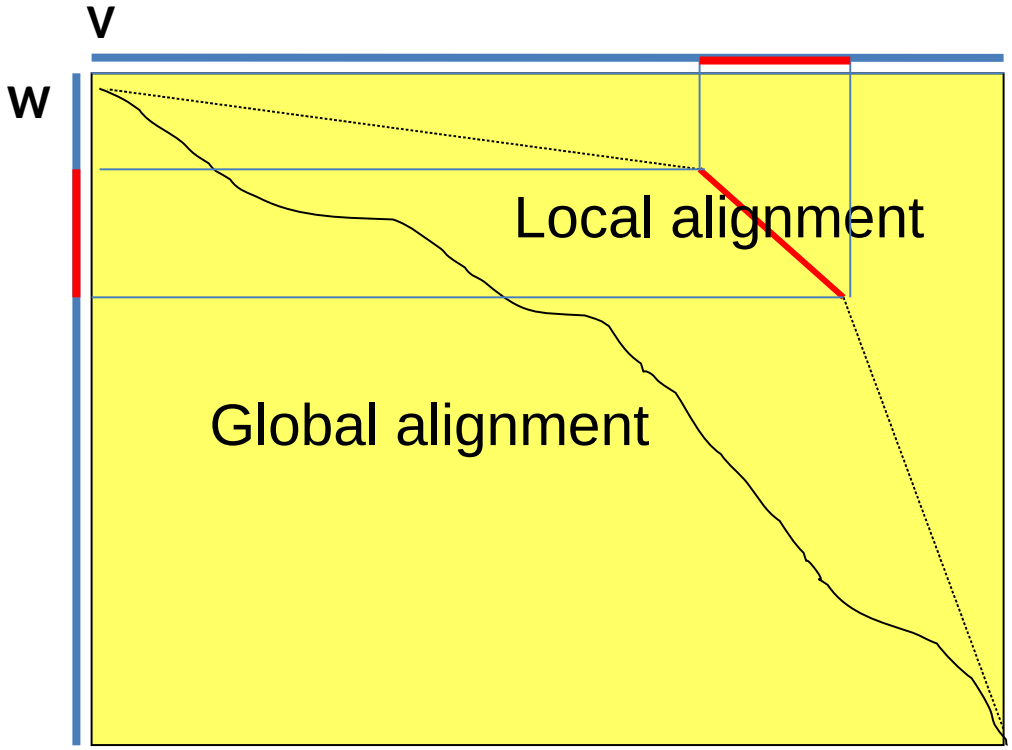
- Recherche – trouver la position d'un gène

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

# Alignement local: Exemple

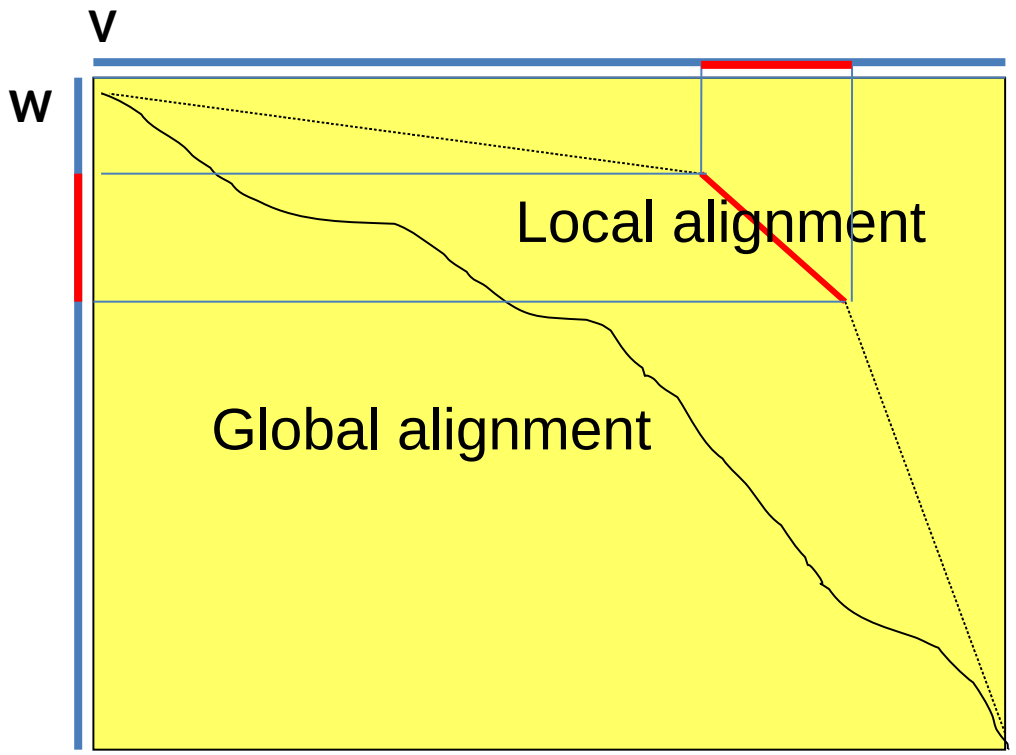
Input : Deux séquences **v, w** et une matrice de scores de similarité δ.

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

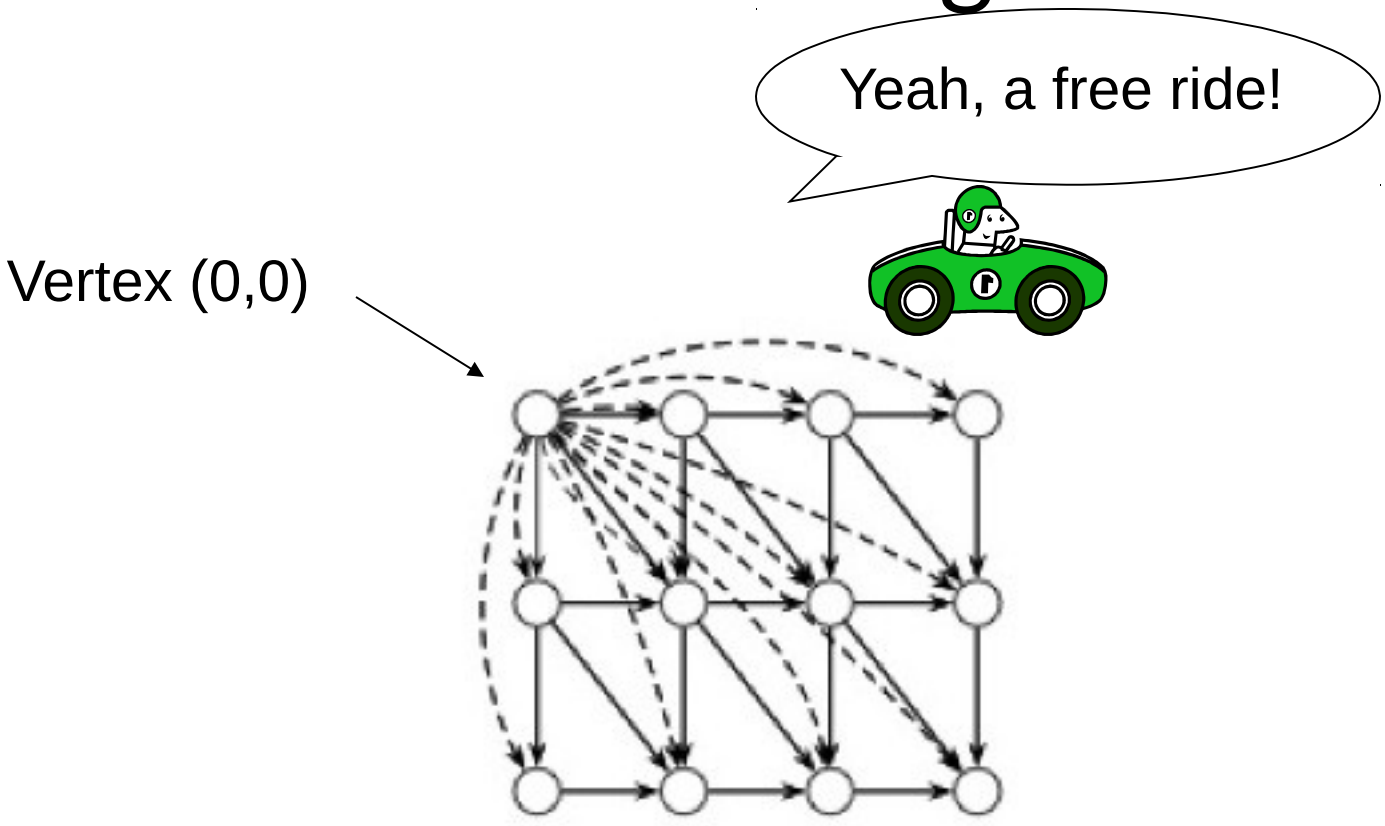


# Solution directe

En temps $O(n^6)$: $n^2$ arêtes sources et $n^2$ arêtes cible. 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 arête par un parcours gratuit!

# Alignement local: Récurrences

- Récurrences:

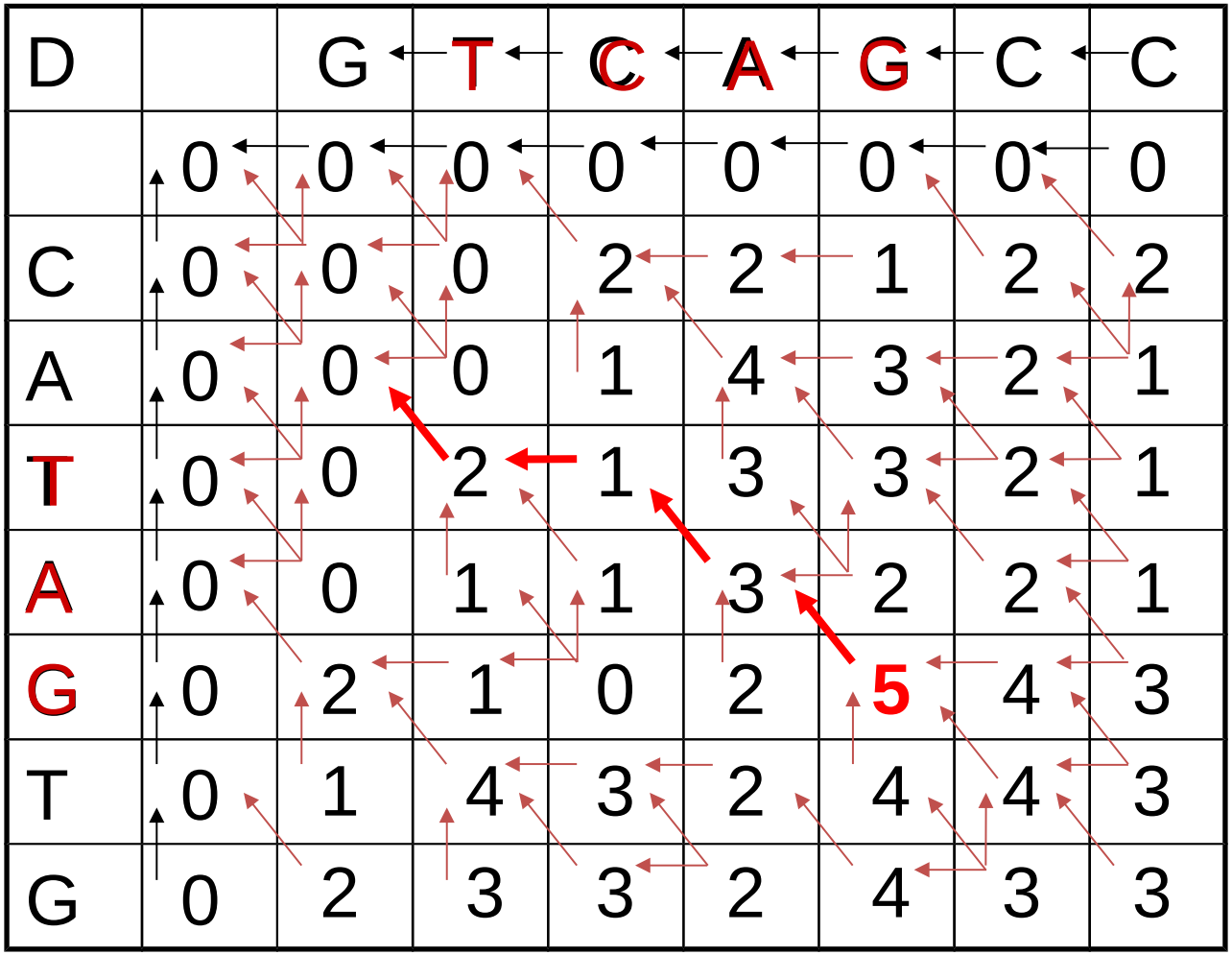
$$V(i,0) = V(0,j) = 0$$

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

- 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.

| 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

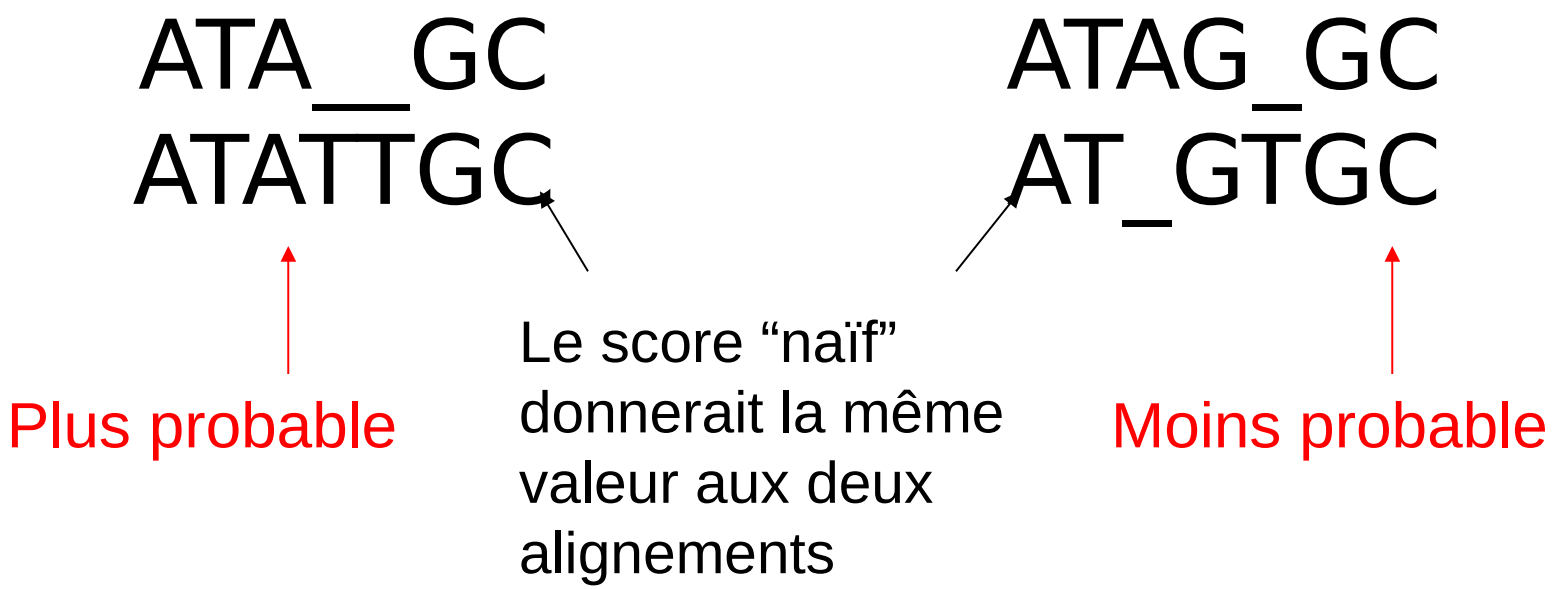
# Scores des indels: Approche naïve

- Un score de pénalité $\sigma$ pour chaque indel:
  - $-\sigma$ pour 1 indel,
  - $-2\sigma$ pour 2 indels consécutifs
  - $-3\sigma$ 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 $\Sigma_{1\leq i \leq \tau}\delta(v_i,w_j)-k\rho$

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

  $\sigma$ : 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: $\Sigma_{1\le i \le l}\delta(v_i,w_j)-k\rho -q\sigma$
- 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 |

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 $\omega$:

  Fonction quelconque de la taille du gap.

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

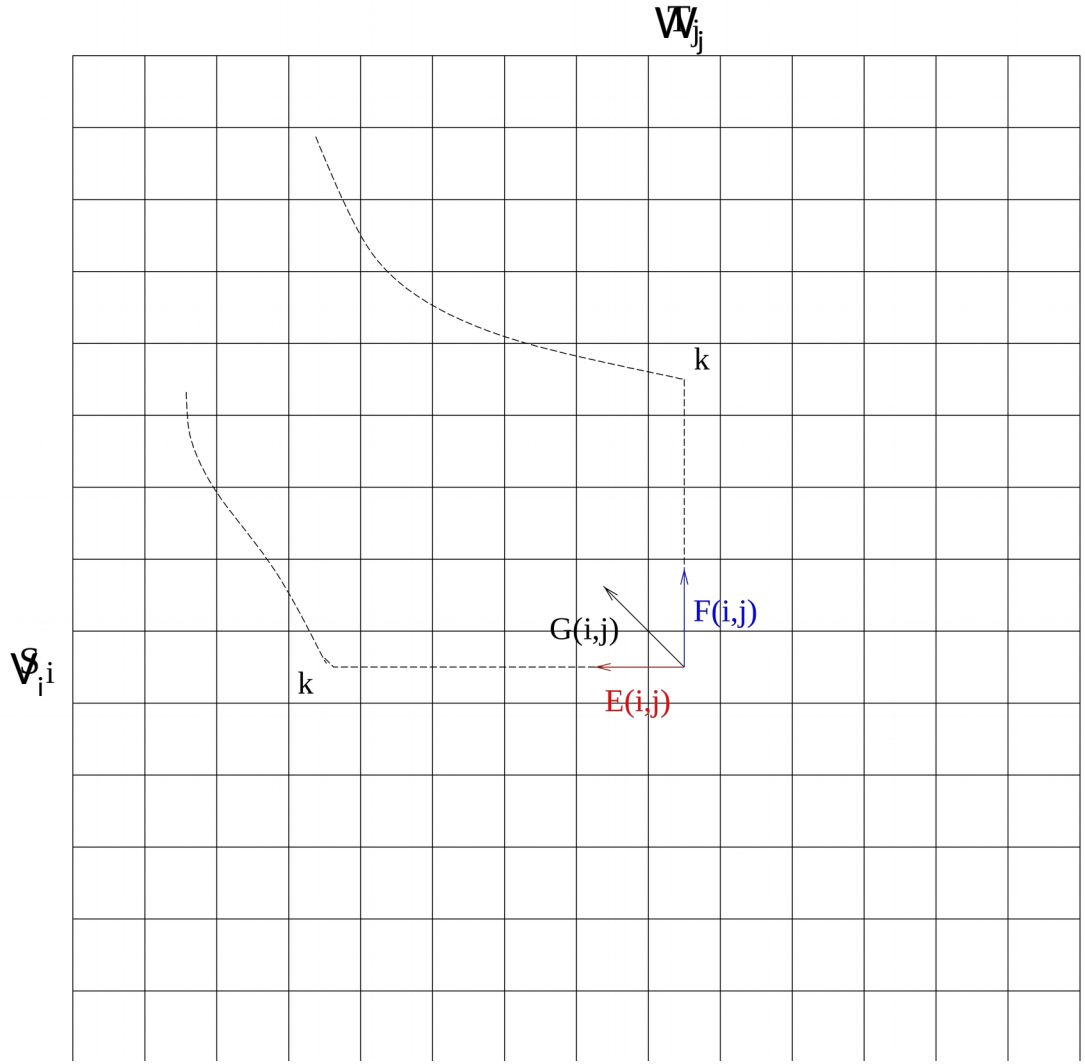
# Alignement avec gap – Pondération quelconque

- 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 $(v_j,-)$
  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: $O(m^2n+nm^2)$

# 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: $O(mn)$

# Recherche approchée d'un mot

- Mot *P* de taille *m*, séquence *T* de taille *n*, entier *k*; *k*<*m*; *m*<< *n*
- **Trouver toutes les occurrences de *P* dans *T* à *k* erreurs près.**

Ex: Chercher toutes les occurrences de CAT à 1 erreur près dans G T C T T C A T ….

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| **G** | **T** | **C** | **T** | **T** | **C** | **A** | **T** |
| | | | * | * | | * | * |

# Recherche approchée d’un mot

| D(I,j) | | G | T | C | T | T | C | A | T |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| C | | | | | | | | | |
| A | | | | | | | | | |
| T | | | | | | | | | |

# Recherche approchée d'un mot

| D(I,j) | | G | T | C | T | T | C | A | T |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | | | | | | | | | |
| A | | | | | | | | | |
| T | | | | | | | | | |

- Initialiser la première ligne à 0

# Recherche approchée d'un mot

| D(I,j) | | G | T | C | T | T | C | A | T |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 1 | | | | | | | | |
| A | 2 | | | | | | | | |
| T | 3 | | | | | | | | |

- Initialiser la **première ligne à 0.**
- Mêmes relations de récurrence que pour l'alignement global.

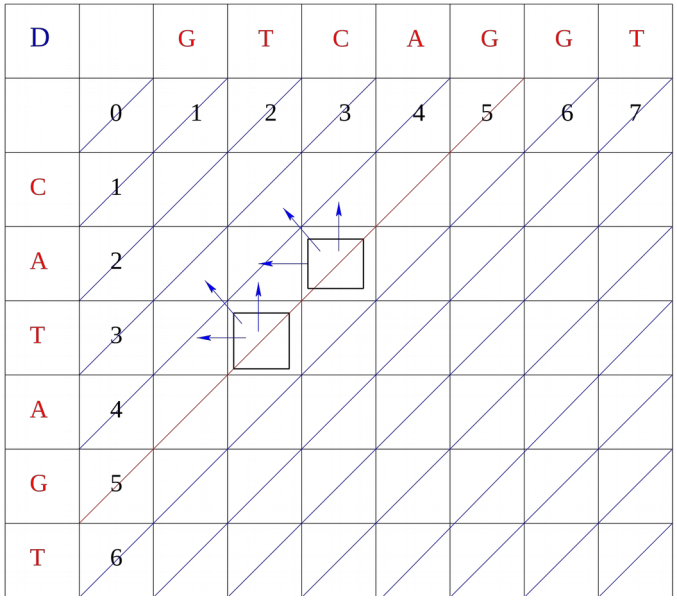
# Recherche approchée d'un mot

| D(I,j) | | G | T | C | T | T | C | A | T |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 1 | | | | | | | | |
| A | 2 | | | | | | | | |
| T | 3 | | | | | | | | |

- Rechercher **à la ligne m** toutes les cases contenant des valeurs ≤ k.
- Suivre les pointeurs **jusqu'à la 1ière ligne**.

# 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)!