Applications de la science des données et de l'intelligence artificielle

Algorithmes de programmation dynamique pour l'alignement de séquences

JOUR 2: modèle de coût affine pour les insertions-deletions

Nicolas Thierry-Mieg
CNRS / TIMC-IMAG / BCM, Grenoble, France

ENSIMAG 2A 2020-2021

Modèles de coût pour les indels

Quel coût pour une suite de N insertions-deletions successives ?

Modèle linéaire : **indel_SW(N) = N × indelOpen**

→ chaque indel a le même coût

→ solution : Smith-Waterman (1981) vu au premier cours

Modèle affine : **indel_AE(N) = indelOpen + (N-1) × indelExtend**

→ on peut pénaliser fortement l'ouverture d'une indel, et peu son extension

→ solution : idée de Gotoh (1982) améliorée par Altschul & Erickson (1986)

Coût linéaire - rappel

- **Smith-Waterman :**

$$S(i,j) = \max\{ S(i-1,j-1) + subst(s1[i-1],s2[j-1]) ,$$
$$S(i-1,j) + indel ,$$
$$S(i,j-1) + indel ,$$
$$0 \}$$

Coût linéaire - rappel

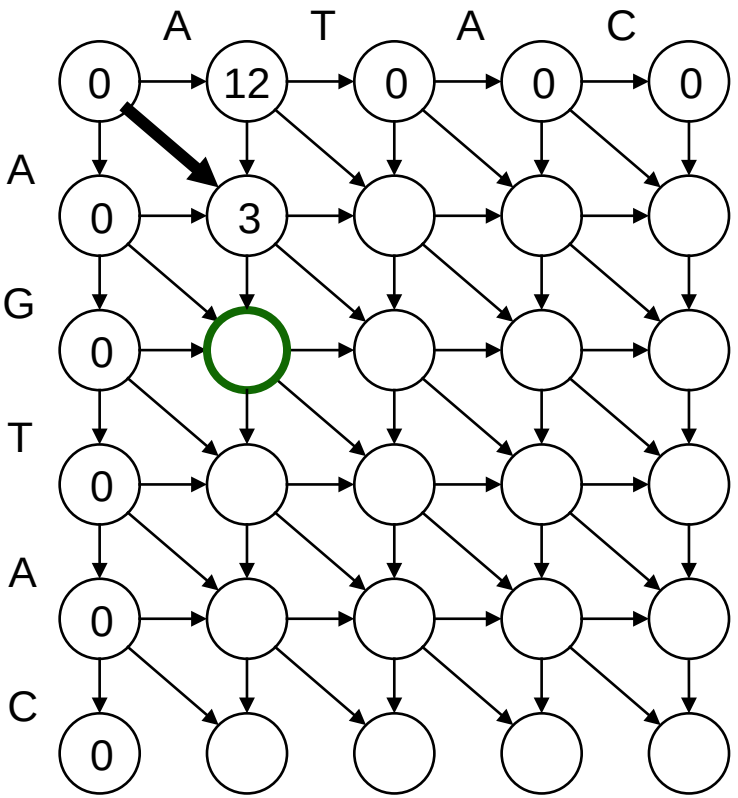
Exemple (partiel) avec :

match = +3

mismatch = -2

indel = -10

→ case **verte** : score= ?



Coût linéaire - rappel

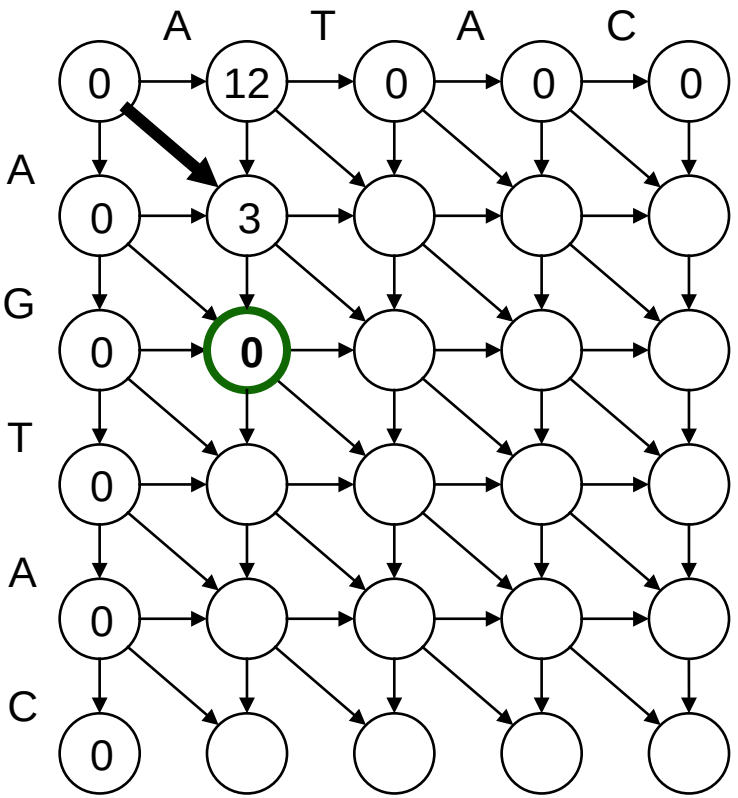
Exemple (partiel) avec :

match = +3

mismatch = -2

indel = -10

→ case **verte** : score=0



Coût affine : la fausse bonne idée…

- **Smith-Waterman :**

  S(i,j) = max{ S(i-1,j-1) + subst(s1[i-1],s2[j-1]) ,
  S(i-1,j) + indel ,
  S(i,j-1) + indel ,
  0 }

- **Une idée :**

  remplacer **S(i-1,j) + indel** par :

  { **S(i-1,j) + indelExtend** si (i-2,j) est prevs de (i-1,j) ,
  S(i-1,j) + indelOpen sinon

  et idem pour S(i,j-1) + indel

Coût affine : la fausse bonne idée…

Exemple (partiel) avec :

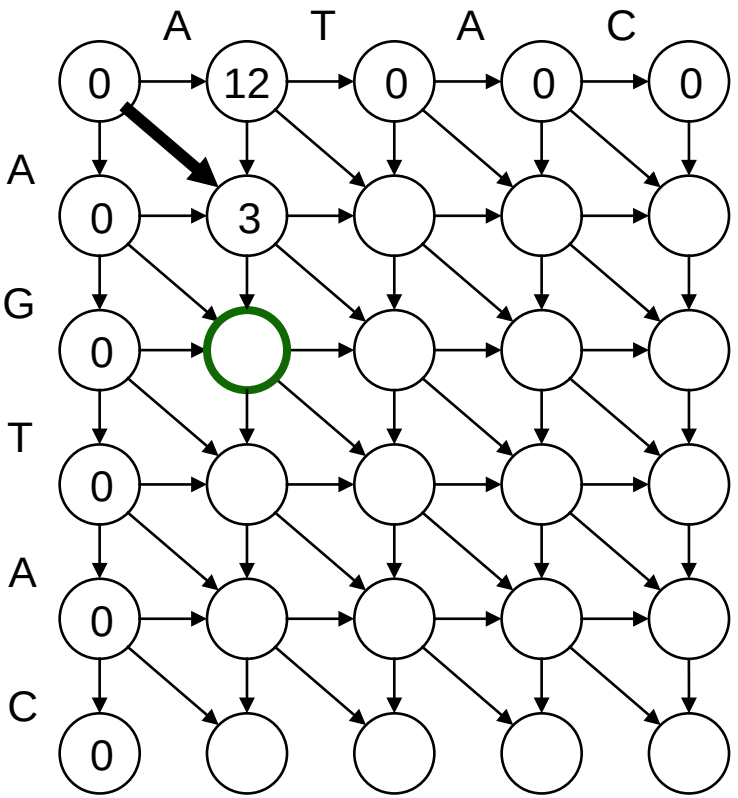
match = +3

mismatch = -2

indelOpen = -10

indelExtend = -1

→ case **verte** : score = ?



Coût affine : la fausse bonne idée…

Exemple (partiel) avec :

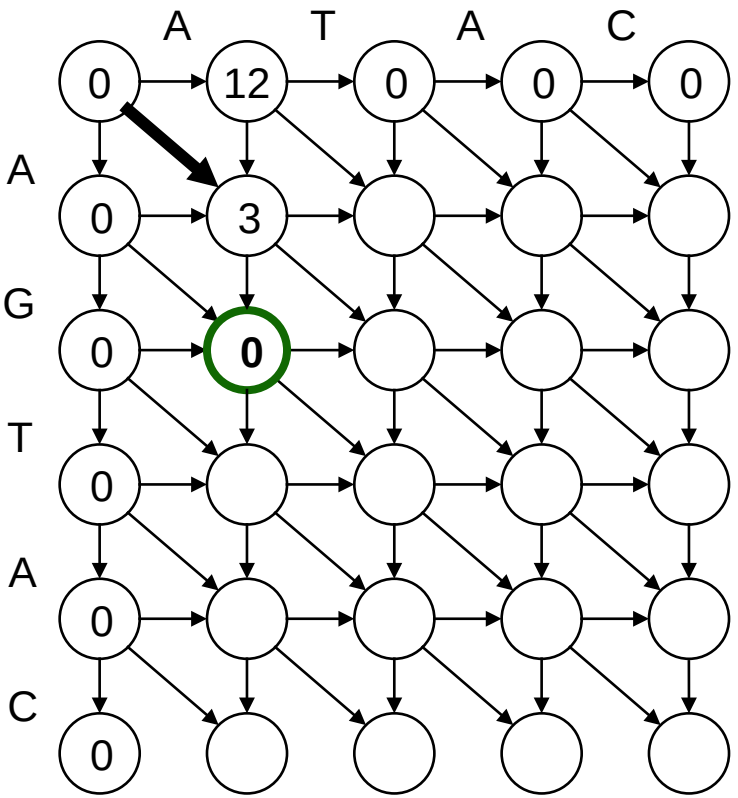
match = +3

mismatch = -2

indelOpen = -10

indelExtend = -1

→ case **verte** : score = max(-2,-10,-7,0) = 0 ?



Coût affine : la fausse bonne idée…

Exemple (partiel) avec :

match = +3

mismatch = -2

indelOpen = -10

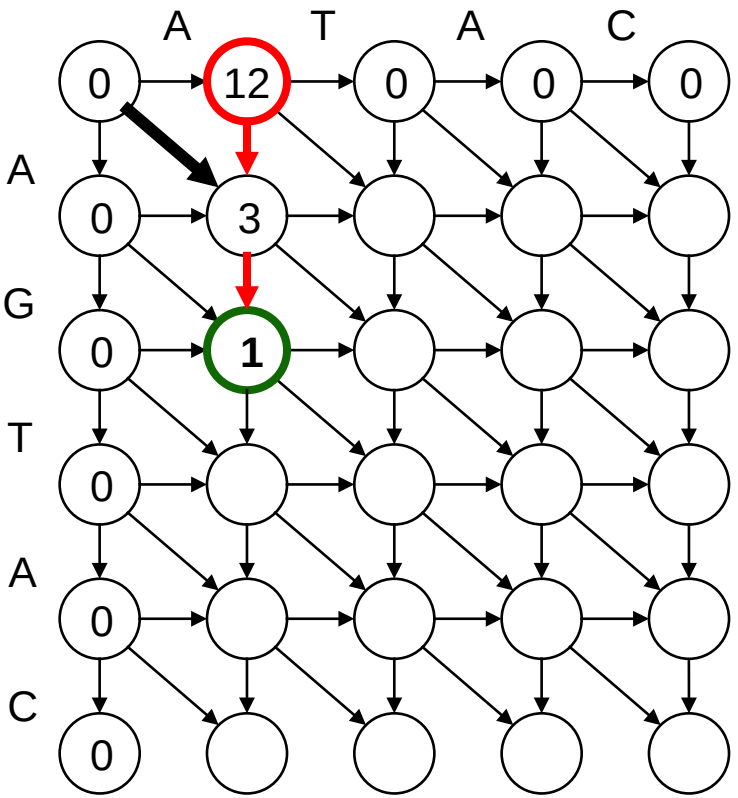
indelExtend = -1

→ case **verte** : score = max(-2,-10,-7,0) = 0 ?

NON !!

en arrivant de la case **rouge :**

score = 12-10-1 = 1



Coût affine : la fausse bonne idée…

Exemple (partiel) avec :

match = +3

mismatch = -2

indelOpen = -10

indelExtend = -1

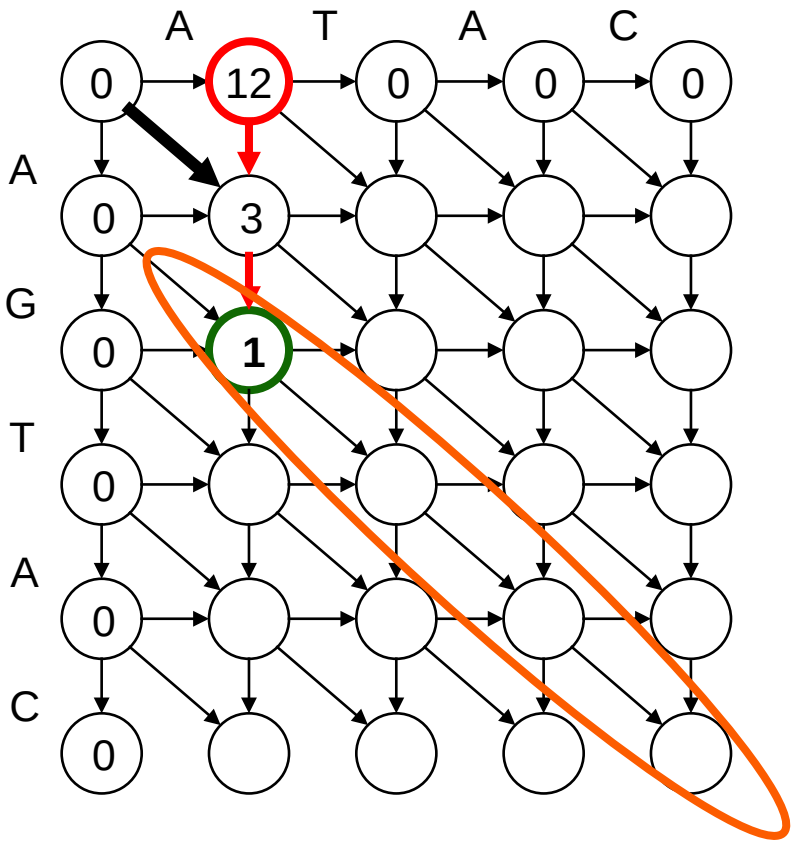
→ case **verte** : score = max(-2,-10,-7,0) = 0 ?

NON !!

en arrivant de la case **rouge :**

score = 12-10-1 = 1

→ si la case verte est le début d'un alignement optimal (eg **diagonale orange**), on aura un score trop faible (de 1) et un alignement optimal tronqué : il manquera l'alignement jusqu'à la case **rouge** ainsi que les 2 indels de la **rouge** à la **verte**



Coût affine : la vraie bonne idée !

Exemple (partiel) avec :

match = +3

mismatch = -2

indelOpen = -10

indelExtend = -1

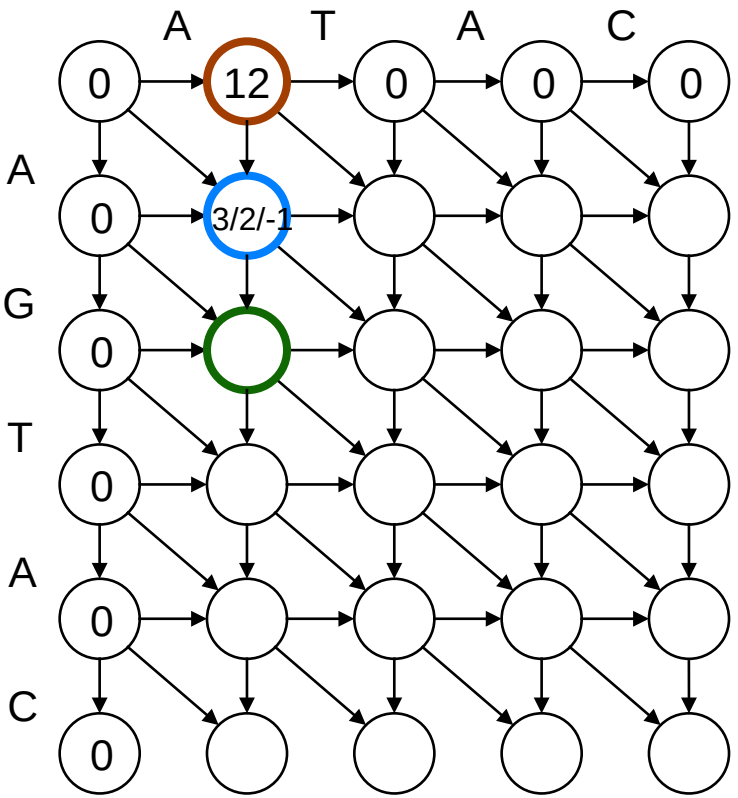
Ce qu'il manque : retenir plusieurs scores dans chaque case, selon le dernier déplacement…

Ici dans la case **bleue** on a :

→ **score_D**=3 après move DIAGONAL

→ **score_V**=2 après move VERTICAL

→ **score_H**=N/A=-1 après move HORIZONTAL



Coût affine : la vraie bonne idée !

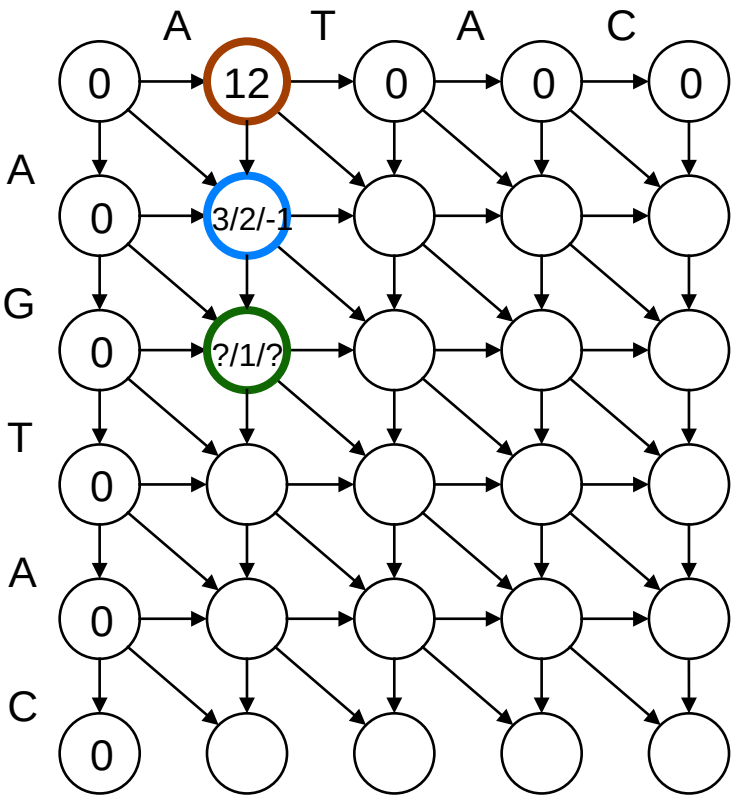
Exemple (partiel) avec :

match = +3

mismatch = -2

indelOpen = -10

indelExtend = -1



Ce qu'il manque : retenir plusieurs scores dans chaque case, selon le dernier déplacement…

Ici dans la case **bleue** on a :

→ **score_D**=3 après move DIAGONAL

→ **score_V**=2 après move VERTICAL

→ **score_H**=N/A=-1 après move HORIZONTAL

Puis dans la case **verte** pour calculer **score_V** on va regarder dans la case **bleue** (car score_V stocke le meilleur score quand le dernier move est VERTICAL) :

score_V = max(**score_D** – 10, **score_V** – 1, **score_H** – 10)

→ **score_V** = 1

Coût affine : la vraie bonne idée - Altschul-Erickson via Gotoh

- Il faut donc 3 scores par coordonnée (i,j) : D, V, H

- Chaque score est obtenu en provenant d'une **case** fixe :
  - (i-1,j-1) pour D
  - (i-1,j) pour V
  - (i,j-1) pour H

mais peut être obtenu avec n'importe lequel des 3 scores (D, V, H) de la case en question !

- Formellement, en notant s=subst(s1[i-1],s2[j-1]) , o=indelOpen, e=indelExtend :

D(i,j) = max[0, D(i-1,j-1)+s, V(i-1,j-1)+s, H(i-1,j-1)+s]

V(i,j) = max[0, D(i-1,j)+o, V(i-1,j)+e, H(i-1,j)+o]

H(i,j) = max[0, D(i,j-1)+o, V(i,j-1)+o, H(i,j-1)+e]

→ Il faut donc retenir 3 bits de *prevs* par score : 9 bits au total