# IFT870/BIN710
# Forage de données

## Thème 6 : Forage de données séquentielles

Aida Ouangraoua
**Département d'informatique**

Université de Sherbrooke

# Données séquentielles

- Liste d'éléments ordonnées
  - Chaque élément est associé à une position spécifique

- **Exemples**:

  - Séries chronologiques : données numériques, intervalles de temps réguliers

  - Séquences symboliques : données catégorielles ou nominales, intervalles irréguliers

    - Séquences biologiques : ADN, ARN, protéines, sémantiques cachées, contient des motifs importants

  - Suite de mots dans un texte

# Forage de données pour séries chronologiques

- Prévision/Régression
  - Prédire les valeurs futures

- Description:
  - Comprendre les phénomènes cachés qui sous-tendent les observations
    - Mouvements à long terme, saisonniers, cycliques, aléatoires
  - Modèles additifs ou multiplicatifs à partir de ces 4 composants
  - Prévision en utilisant les modèles

- Recherche de séquences similaires
  - Similarité globale
  - Similarité locale
  - Recherche basée sur des motifs

- Réduction de données ou de dimensions
  - Transformée de Fourrier discrète, Transformée en ondelettes discrète

# Forage de données pour séquences symboliques

- Recherche de motifs fréquents
  - Recherche sans contrainte : motifs = sous-séquences
  - Recherche avec contrainte : exemple: motifs = sous-chaînes

- Classification de séquences
  - Basée sur des vecteurs d'attributs (**Exemple**: k-mers)
  - Basée sur des distances (**Exemple**: distance d'édition)
  - Basée sur des modèles (**Exemple**: Modèles de Markov cachés)

- Segmentation / Compression de séquences

# Recherche de motifs fréquents

- Séquence de transactions:
  [ $t_1$ $t_2$ $t_3$ … ] telles que chaque $t_i$ est un ensemble d'items $t_i = \{i_1, i_2, \ldots, i_k\}$

- K-sequence : sous-séquence de k items
  - Exemple: [ {2} {8} ] est une 2-sous-séquence de [ {2,3} {4,5,6} {7,8} ]

- K-chaîne : sous-chaîne de k items pris dans des transactions consécutives
  - Exemple : : [ {2} {4} {8} ] est une 3-sous-chaîne de [ {2,3} {4,5,6} {7,8} ]

- Score d'une sous-séquence : pourcentage de séquences la contenant dans un ensemble de séquences.

- Étant donné un seuil *min_score*, une sous-séquence est fréquente si son score est supérieur ou égal à *min_score*.

# Recherche de motifs fréquents

- Très grand nombre de sous-séquences dans un ensemble de séquences

- Besoin d'algorithmes efficaces qui explorent l'espace de recherche de façon intelligente, en tenant compte de sa structure, pour éviter de répéter les mêmes opérations.

  - Méthode Generalized Sequential Patterns (GSP)

  - Sequential Pattern Discovery using Equivallent Class (SPADE)

  - (Arbre de préfixe ou arbre de suffixe) PrefixSpan

# Algorithme GSP

- **Initialisation**:
  Lister toutes  les 1-sous-séquences fréquentes

- **k = 2;**

- **Tant que de nouveaux motifs fréquents sont trouvés**:

  - Générer des k-sous-séquences candidates en fusionnant des paires de (k-1)-sous-séquences fréquentes

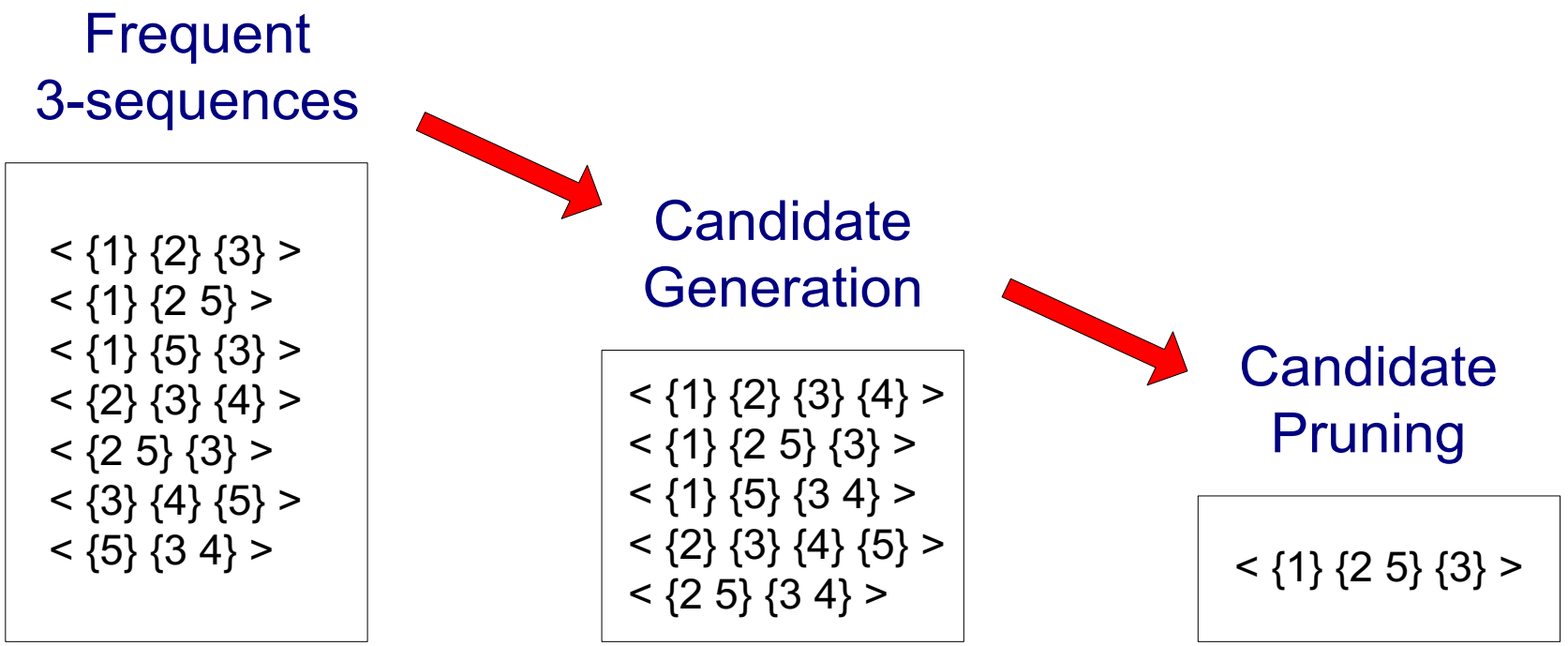
  - Retirer les k-sous-séquences contenant des (k-1)-sous-séquences non fréquentes

  - Calculer le score des candidats restants

  - Retirer les candidats de score inférieur à min_score

  - k = k+ 1

# Exemple GSP




Source: Data Mining: Concepts and Techniques

# GSP : inefficace pour de grands ensembles

- Très grand nombre de sous-séquences candidates générées à chaque étape
  - Si n k-sous-séquences ➔ (n * (n-1))/2 (k+1)-sous-séquences candidates
- À chaque étape, on doit recalculer les scores des candidats

# Méthode SPADE

- À chaque transaction, on associe une paire (SID, EID) telle que SID est l'identifiant de la séquence, et EID la position dans la séquence.

- Comme pour GSP, on génère les k-sous-séquences candidates à partir des sous-séquences plus courtes

# Méthode SPADE

| SID | EID | Items |
|---|---|---|
| 1 | 1 | a |
| 1 | 2 | abc |
| 1 | 3 | ac |
| 1 | 4 | d |
| 1 | 5 | cf |
| 2 | 1 | ad |
| 2 | 2 | c |
| 2 | 3 | bc |
| 2 | 4 | ae |
| 3 | 1 | ef |
| 3 | 2 | ab |
| 3 | 3 | df |
| 3 | 4 | c |
| 3 | 5 | b |
| 4 | 1 | e |
| 4 | 2 | g |
| 4 | 3 | af |
| 4 | 4 | c |
| 4 | 5 | b |
| 4 | 6 | c |

| a | | b | | ... |
|---|---|---|---|---|
| SID | EID | SID | EID | ... |
| 1 | 1 | 1 | 2 | |
| 1 | 2 | 2 | 3 | |
| 1 | 3 | 3 | 2 | |
| 2 | 1 | 3 | 5 | |
| 2 | 4 | 4 | 5 | |
| 3 | 2 | | | |
| 4 | 3 | | | |

| ab | | | ba | | | ... |
|---|---|---|---|---|---|---|
| SID | EID (a) | EID(b) | SID | EID (b) | EID(a) | ... |
| 1 | 1 | 2 | 1 | 2 | 3 | |
| 2 | 1 | 3 | 2 | 3 | 4 | |
| 3 | 2 | 5 | | | | |
| 4 | 3 | 5 | | | | |

| aba | | | | ... |
|---|---|---|---|---|
| SID | EID (a) | EID(b) | EID(a) | ... |
| 1 | 1 | 2 | 3 | |
| 2 | 1 | 3 | 4 | |

Source: Data Mining: Concepts and Techniques

# SPADE : inefficace pour de grands ensembles

- Consommation élevé de la mémoire
- Peu flexible pour les contraintes complexes

# PrefixSpan

Étant donnée une séquence [a abc ac d cf]

| Préfixe | Suffixe |
|---|---|
| [a] | [abc ac d cf] |
| [aa] | [bc ac d cf] |
| [a ab] | [c ac d cf ] |
| [a abc] | [ac d cf ] |
| [a abc a] | [c d cf ] |
| [a abc ac] | [d cf ] |
| [a abc acd] | [cf ] |
| [a abc acd c] | [f ] |

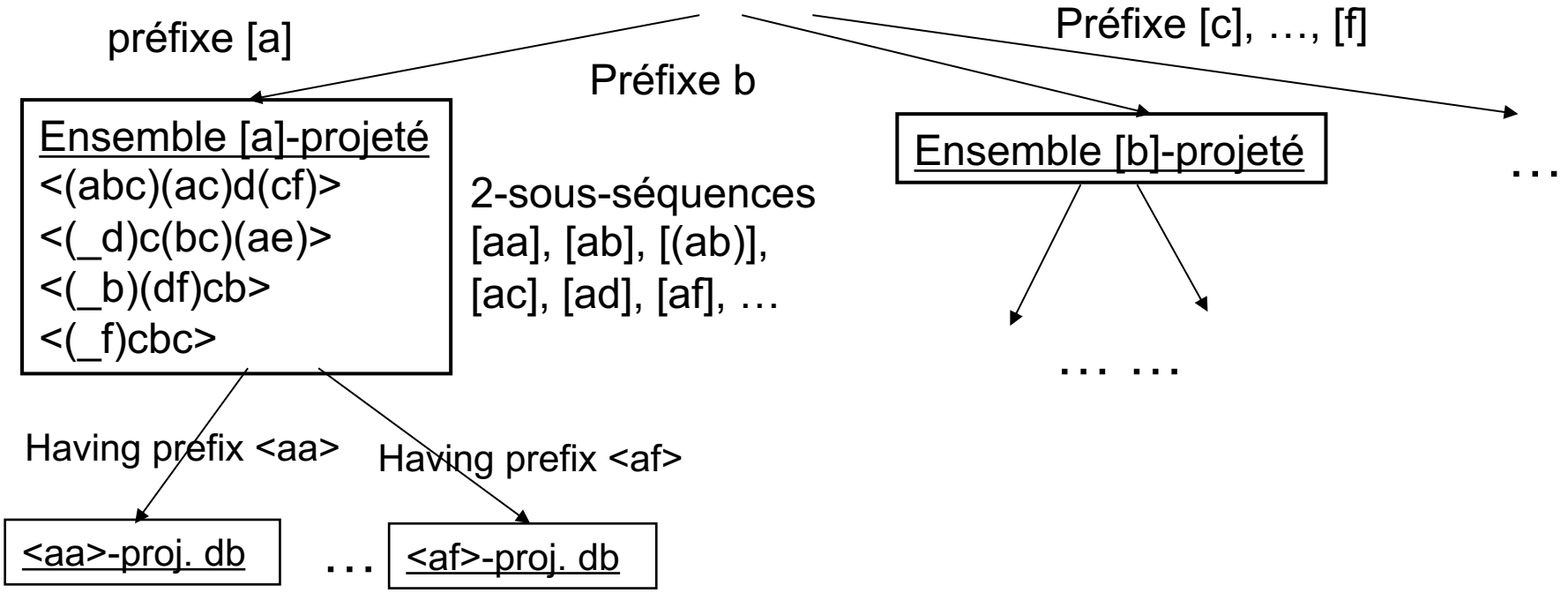
# PrefixSpan

- Partitionnement hiérarchique des suffixes en fonctions de leurs préfixes:
  - Suffixes commençant par [a]
    - Suffixes commençant par [aa]
    - Suffixes commençant par [ab]
    - Suffixes commençant par [ac]
    - …
  - Suffixes commençant par [b]
    - …
  - Suffixes commençant par [c]
    - …
  - Suffixes commençant par [d]
    - ...
  - Suffixes commençant par [e]
    - …
  - Suffixes commençant par [f]
    - …

# PrefixSpan

| SID | sequence |
|---|---|
| 10 | [a(abc)(ac)d(cf)] |
| 20 | [(ad)c(bc)(ae)] |
| 30 | [(ef)(ab)(df)cb] |
| 40 | [eg(af)cbc] |

1-sous-séquences
[a], [b], [c], [d], [e], [f]



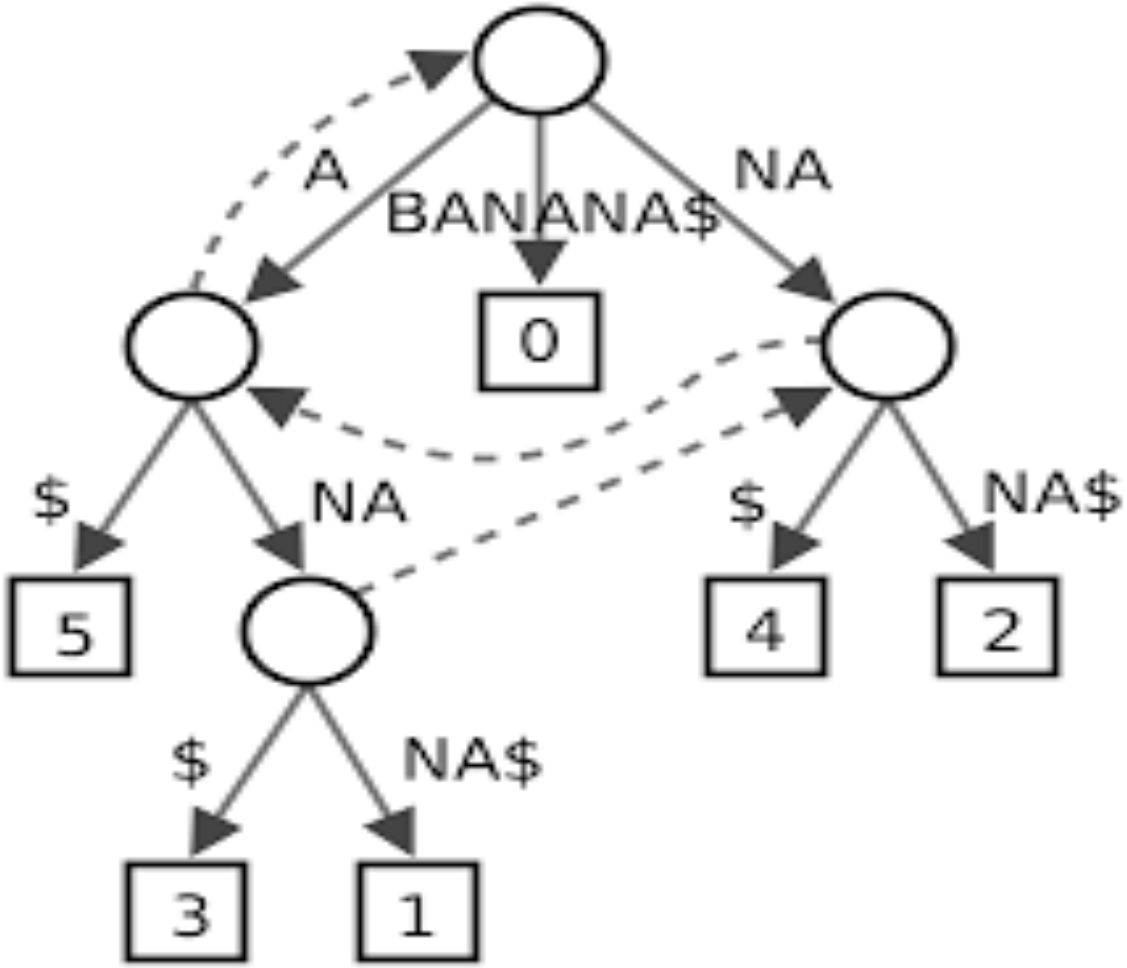

Source: Data Mining: Concepts and Techniques

# PrefixSpan

- Avantage:
  - On ne génère pas de sous-séquences candidates
- Limites:
  - Coût pour le calcul du sous-ensemble projeté à chaque nœud

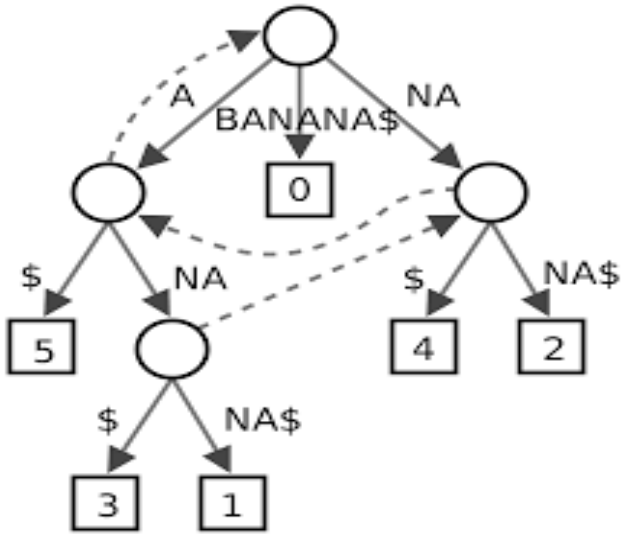
# Arbre de suffixe pour une chaîne (Ex: Proteine)

- Arbre enraciné binaire
- Bijection entre l'ensemble des chemins (racine → feuille) et les suffixes



# Arbre de suffixe pour une chaîne (Ex: ADN)

- Construction en O(n) pour une chaîne S telle que | S | = n
- Ajouter un caractère spécial à la fin
- Pour i de 0 à n-1:
  - Insérer un chemin racine → feuille correspondant à S[i:n]
- On obtient un « tri de suffixe »
- Pour obtenir l'arbre de suffixe, compresser le tri de suffixe:
  - pour tout nœud x qui n'a qu'un enfant, fusionner x et son enfant



# Implémentation : Arbre de suffixe

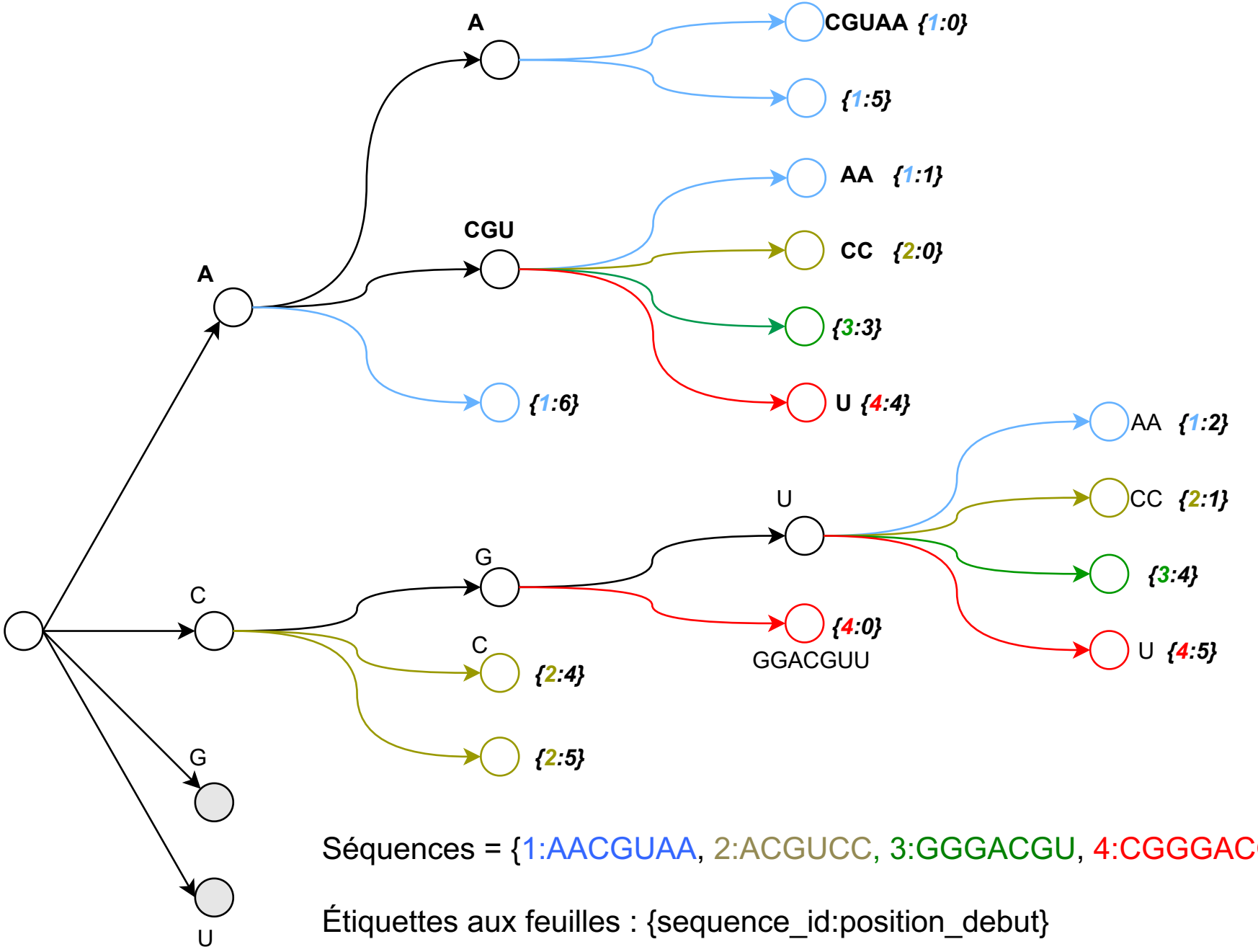
- Diverses structures de données pour l'implémentation
  - Liste
  - Tableau non trié
  - Tableau trié
  - Arbre équilibré
  - Table de hachage

- Choix de la structure de données en fonction de la complexité pour la recherche d'un suffixe, l'insertion d'un suffixe, et le parcours de l'arbre pour lister les suffixes (d'un suffixe au suivant)

- Exemple (M étant le nombre de suffixes):
  - Liste, ou tableau non trié
    - Recherche en O(M) ; Insertion d'un suffixe en O(1) ; Parcours en O(1)
  - Tableau trié
    - Recherche en O(log(M)); Insertion en O(M) ; Parcours en O(1)
  - Arbre binaire équilibré
    - Recherche et insertion en O(log(M)) ; Parcours en O(1)
  - Table de hachage
    - Recherche et insertion en O(1) ; Parcours en O(M)

# Arbre de suffixe généralisé

- Arbre de suffixes pour plusieurs séquences




Séquences = {1:AACGUAA, 2:ACGUCC, 3:GGGACGU, 4:CGGGACGUU}

Étiquettes aux feuilles : {sequence_id:position_debut}

# Classification de séquences

- Basée sur des vecteurs d'attributs (**Exemple**: k-mers)
- Basée sur des distances (**Exemple**: distance d'édition)
- Basée sur des modèles (**Exemple**: Modèles de Markov cachés)

# Classification basée sur des vecteurs d'attributs

- Objectif : une représentation vectorielle des séquences pour utiliser des méthodes de classifications usuelles
- K-mer : sous-chaîne de longueur k
- Vecteurs des fréquences de k-mers utilisés pour résumer les séquences

| **Séquences** | **1-mer** | **2-mer** | **3-mer** |
|---|---|---|---|
| AACGUAA | A C G U | AA AC CG GU UA | AAC ACG CGU GUA UAA |
| ACGCC | A C G | AC CG CC GC | ACG CGC GCC |
| GGGACGU | A C G U | AC CG GA GG GU | ACG CGU GAC GGA GGG |
| CGGGCGUU | C G U | CG GC GG GU UU | CGG CGU GCG GGC GGG GUU |

# Classification basée sur des vecteurs d'attributs

- Distance euclidienne
  - 1-mer

D(S1,S2) = 3.74; D(S1,S3) = 4.24; D(S1,S4) = 5.09; D(S2,S3) = 3.74; D(S2,S4) = 3.46; D(S3,S4) = 1.41;

  - 1-mer + 2-mer

D(S1,S2) = 4.69; D(S1,S3) = 5.29; D(S1,S4) = 6.24; D(S2,S3) = 4.69; D(S2,S4) = 4.58; D(S3,S4) = 2.64;

| **Séquences** | **1-mer** | **2-mer** | **3-mer** |
|---|---|---|---|
| AACGUAA | A C G U | AA AC CG GU UA | AAC ACG CGU GUA UAA |
| ACGCC | A C G | AC CG CC GC | ACG CGC GCC |
| GGGACGU | A C G U | AC CG GA GG GU | ACG CGU GAC GGA GGG |
| CGGGCGUU | C G U | CG GC GG GU UU | CGG CGU GCG GGC GGG GUU |

| **Séquences** | **A** | **C** | **G** | **U** | **AA** | **AC** | **CG** | **CC** | **GA** | **GC** | **GG** | **GU** | **UA** | **UU** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S1 = AACGUAA | 4 | 1 | 1 | 1 | 2 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| S2 = ACGCC | 1 | 3 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| S3 = GGGACGU | 1 | 1 | 4 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 2 | 1 | 0 | 0 |
| S4 = CGGGCGUU | 0 | 2 | 4 | 1 | 0 | 0 | 2 | 0 | 0 | 1 | 2 | 1 | 0 | 1 |

# Classification basée sur des vecteurs d'attributs

- Avantages :
  - Simple et efficace
  - Coût de la transformation pour chaque nouvelle séquence en O(n)
  - Fonctionne bien pour des ensembles de séquences de taille similaire (possibilité de normaliser les valeurs)
  - Permet d'identifier les motifs fréquents
  - Modèle interprétable

- Limites :
  - Nécessite de sélectionner un nombre d'attributs pas trop élevé (pour éviter trop de 0 dans la matrice de données)
  - Ignore le contexte et l'ordre des k-mers : des séquences aléatoires très différentes peuvent avoir des fréquences de caractères similaires

# Classification basée sur des distances

- Objectif : utiliser une formule de distance pour calculer une matrice de distances entre les séquences et utiliser un algorithme de classification basée sur les distances (**exemple**: KNN)

  - Distance de Hamming
  - Distance d'édition (Levenshtein)
  - Distance d'édition pondérée
  - Distance d'édition généralisée

# Classification basée sur des distances

- Distance de Hamming

  - Distance = nombre de substitutions
    $D(S_1, S_2) = 6$

    AACTGGATGCTT
    ACTGGACTGAGT

  - Utile quand les séquences comparées sont de même longueur et le modèle de comparaison sous-jacent est sans suppression

  - Calcul de la distance en temps linéaire

# Classification basée sur des distances

- Distance d'édition (Levenshtein)

  - Distance = nombre minimum de substitutions, insertions et délétions nécessaire pour transformer $S_1$ en $S_2$
    $D(S_1, S_2) = 4$

```
AACTGGA-TGCTT
A-CTGGACTGAGT
```

  - Calcul de distance par programmation dynamique en temps quadratique

# Classification basée sur des distances

- Distance d'édition pondérée : attribution d'un coût aux substitutions/insertions/délétions
  - Coût d > 0 pour une délétion ou une insertion
  - Coût s > 0 pour une substitution

- **Exemple**: si d = 2 et s = 1.
  $D(S_1,S_2) = 2 \times d + 2 \times s = 6$

```
AACTGGA-TGCTT
A-CTGGACTGAGT
```

- **Exemple**: si d = 3 et s = 1.
  $D(S_1,S_2) = 7 \times s = 7$

```
AACTGGATGCTT
ACTGGACTGAGT
```

- Nécessite de fixer correctement les coûts

# Classification basée sur des distances

- Distance d'édition généralisée : définition d'une fonction de coût c qui associe à chaque paire de caractères différents un coût >= 0 qui dépend des caractères comparés

- Coût d'un alignement = somme des coûts des caractères alignés

- Exemple: $D(S_1,S_2) = 2 \times c(A,A) + c(A,-) + c(C,C) + 3 \times c(T,T) + 3 \times c(G,G) + c(-,C) + c(C,A) + c(T,G)$

```
AACTGGA-TGCTT
A-CTGGACTGAGT
```

- Coût c est généralement une distance, et donc D également.

- Distance d'édition simple : $c(x,x) = 0$ et $c(x,-) = c(-,x) = c(x,y) = 1$

- Distance d'édition pondérée : $c(x,x) = 0$ ; $c(x,-) = c(-,x) = d$ et $c(x,y) = s$

# Programmation dynamique pour calcul d'une distance d'édition

**Un alignement** :
Un chemin du point gauche-haut vers le point droit-bas.

```
AACTGGA-TGCTT
A-CTGGACTGAGT
```

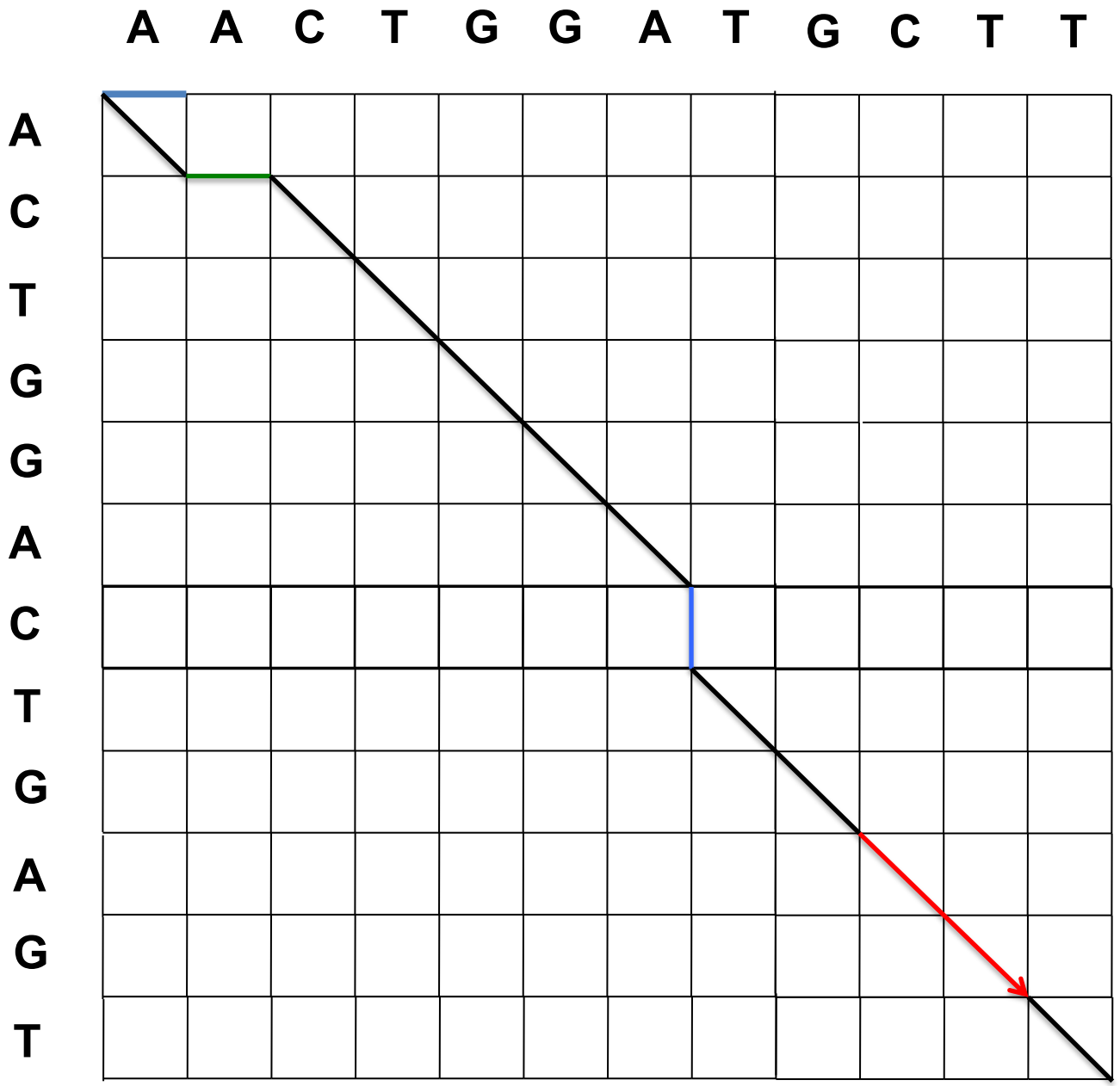
conservation

substitution

délétion

insertion

**Espace de recherche** :
Ensemble de tous les chemins possibles du point gauche-haut vers Le point droit-bas.



# Programmation dynamique pour calcul d'une distance d'édition

- Méthode de résolution de problème consistant à résoudre successivement des sous-problèmes par ordre d'inclusion croissante.

- Utilisation d'un tableau de programmation dynamique pour stocker les solutions des sous-problèmes afin de ne pas les recalculer deux fois.

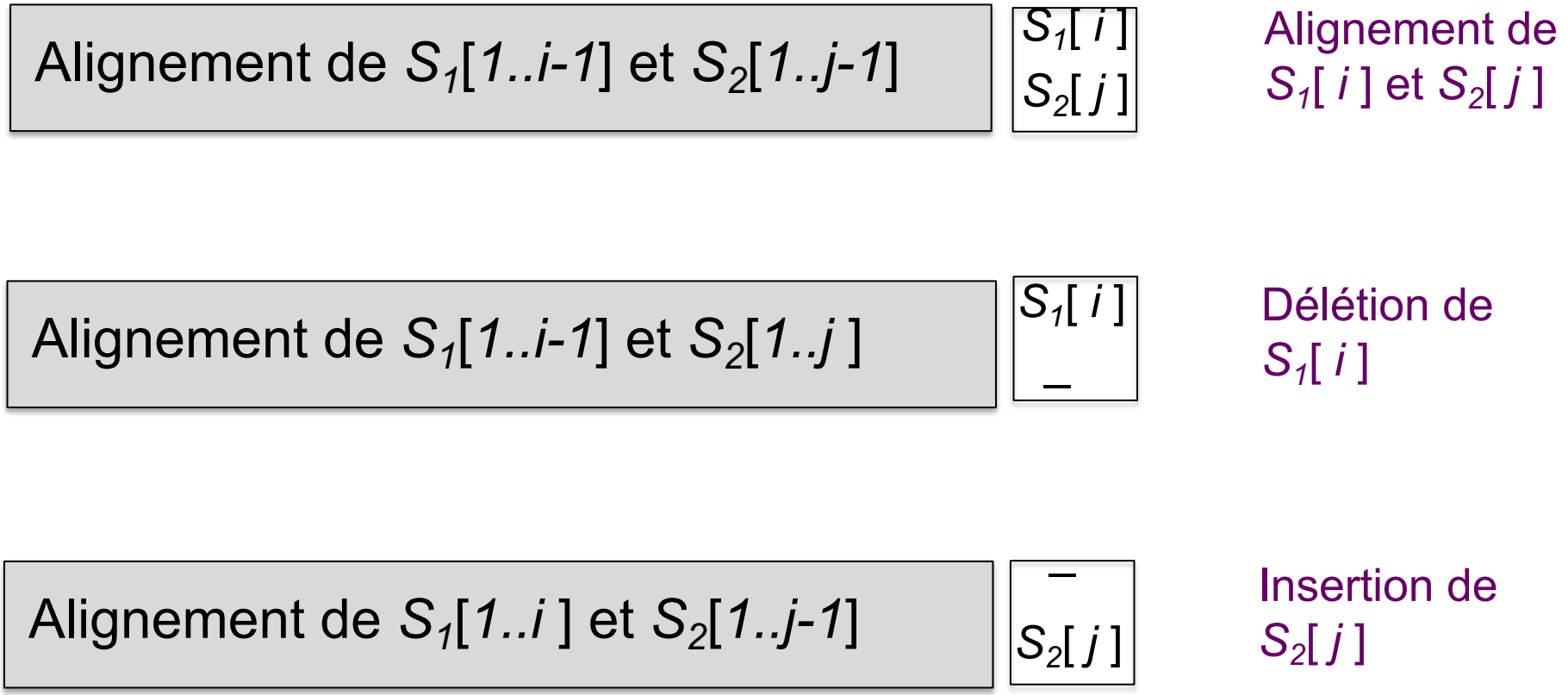
- Problème : calculer $D(S_1,S_2)$
  - Calculer $D(n,m) = D(S_1[1..n], S_2[1..m])$ ; $n = |S_1|$ et $m = |S_2|$

- Sous-problèmes : calculer les distances entre les préfixes
  - Calculer $D(i,j) = D(S_1[1..i], S_2[1..j])$ *; 1 <= i <= n et 1 <= j <= m*

# Programmation dynamique pour calcul d'une distance d'édition

- Expression de $D(i,j)$ en fonction de distances entre préfixes plus petits que $S_1[1..i\,]$ et $S_2[1..j\,]$

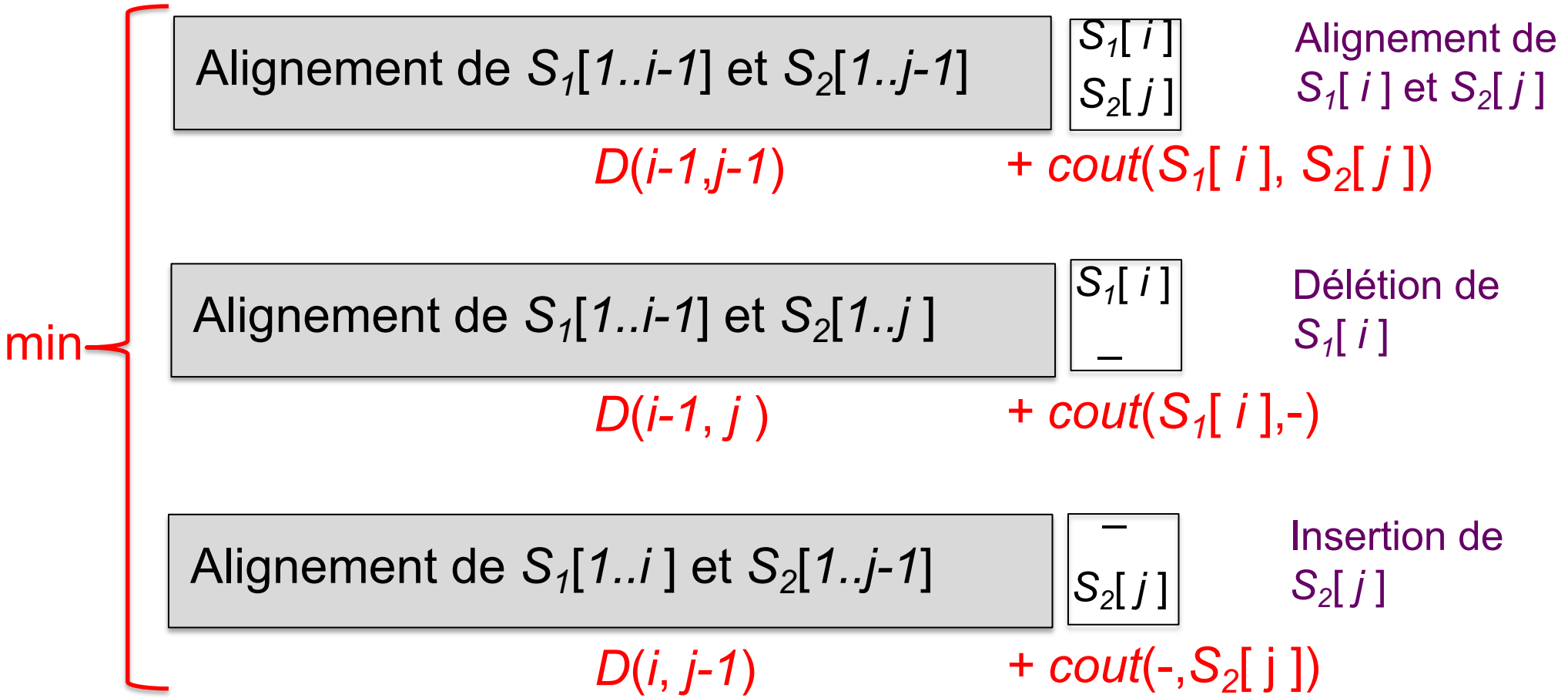
- Trois cas de figures pour un alignement entre $S_1[1..i\,]$ et $S_2[1..j\,]$



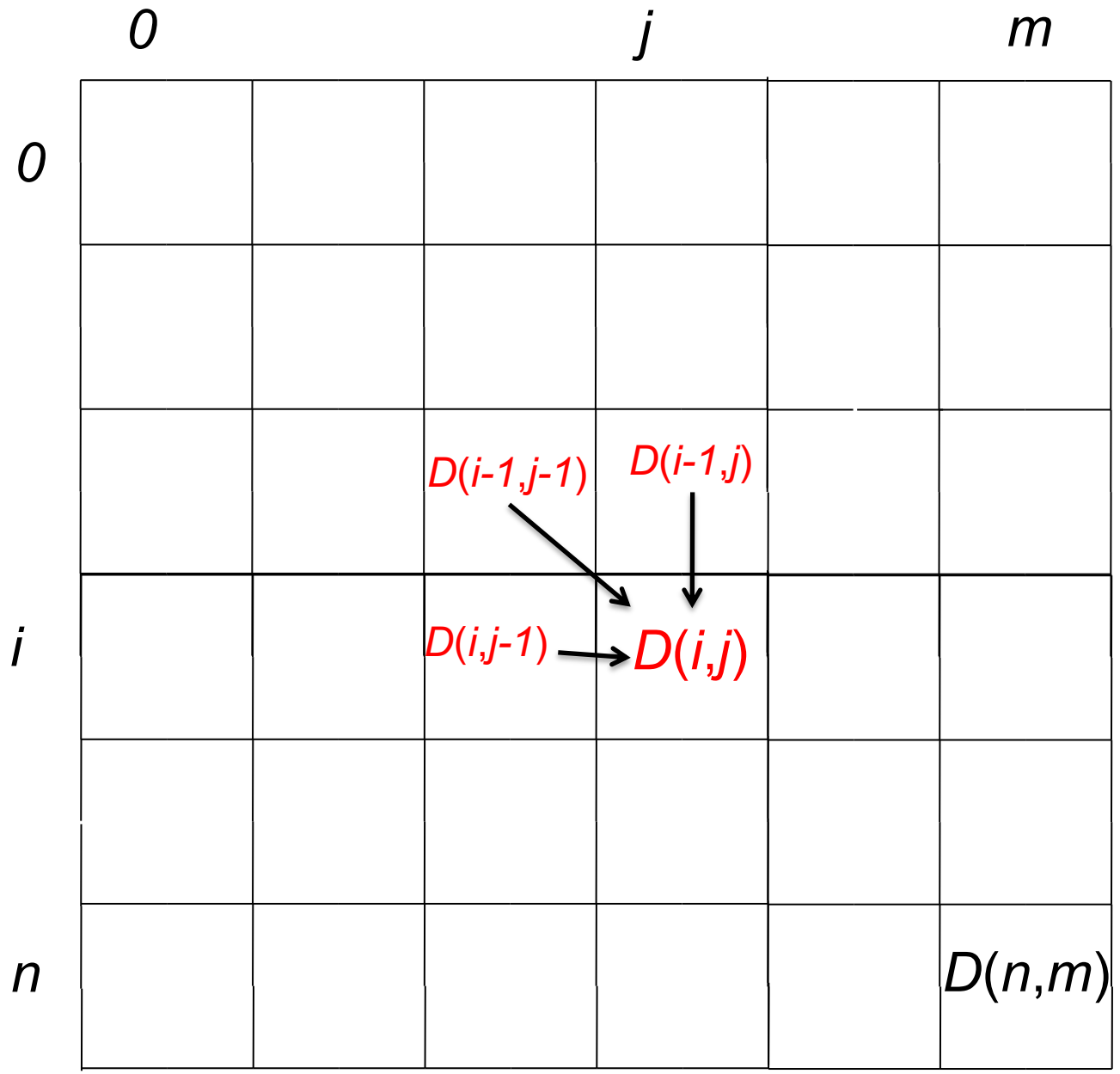
# Programmation dynamique pour calcul d'une distance d'édition

- Expression de $D(i,j)$ en fonction de distances entre préfixes plus petits que $S_1[1..i\,]$ et $S_2[1..j\,]$

- Trois cas de figures pour un alignement entre $S_1[1..i\,]$ et $S_2[1..j\,]$



# Programmation dynamique pour calcul d'une distance d'édition

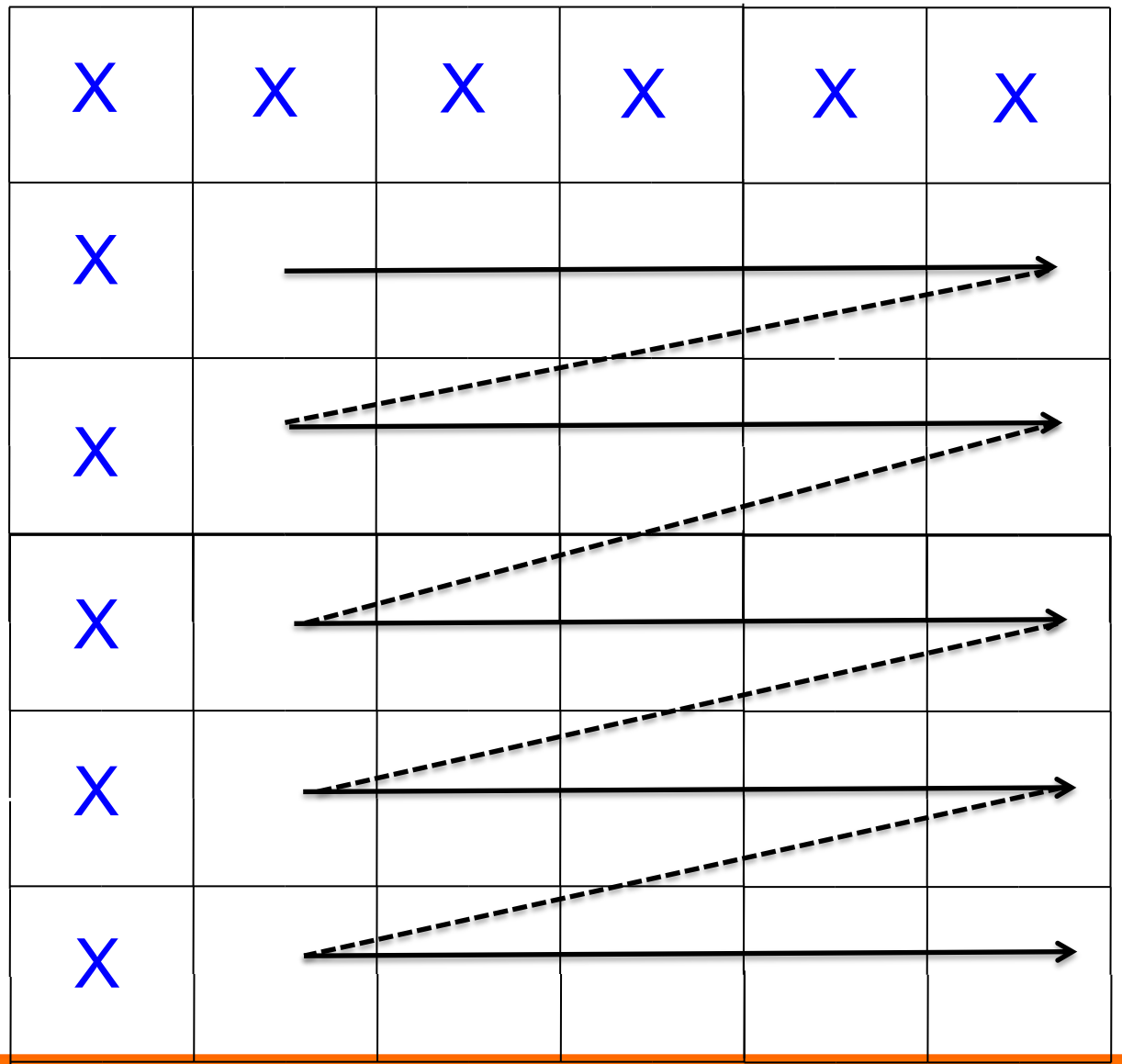


# Programmation dynamique pour calcul d'une distance d'édition

$$D(i,0) = \begin{cases} D(i\text{-}1, 0) + cout(S_1[\,i\,], \text{-}) & \text{si } i > 0 \\ 0 & \text{sinon} \end{cases}$$

$$D(0,j) = \begin{cases} D(0, j\text{-}1) + cout(\text{-}, S_2[\,j\,]) & \text{si } j > 0 \\ 0 & \text{sinon} \end{cases}$$

Initialisation




Complexité:
$O(n^2)$

# Classification basée sur des distances

- Calculer un score de similarité plutôt qu'une distance.
  - $D(S_1,S_2) = 0$ ; on cherche la distance minimum
  - $Sim(S_1,S_2) > 0$ ; on cherche le score maximum

- Similarité globale : duale de la distance globale : algorithme similaire

- Similarité locale: calculer le score maximum entre deux sous-chaînes de $S_1$ et $S_2$

# Classification basée sur des distances

- Avantages :
  - Fonctionne bien si la distance évalue bien les différences entre les séquences
  - Tient compte de l'ordre des caractères
  - Permet de comparer des séquences de tailles différentes

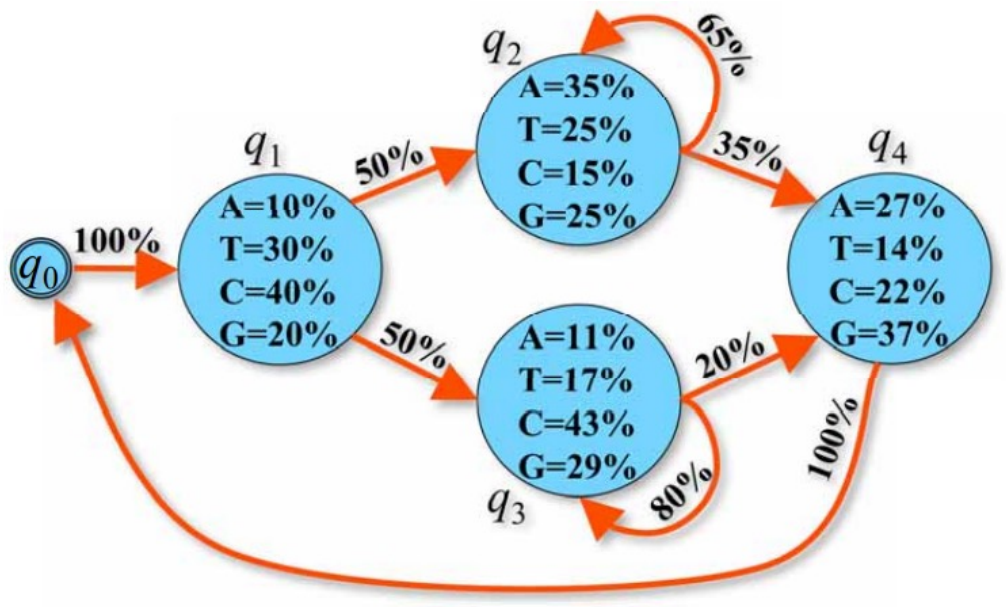
- Limites :
  - Nécessite d'aligner les séquences pour interpréter les modèles
  - Nécessite une connaissance a priopri des séquences pour bien définir la distance
  - Coût du calcul des distances pour chaque nouvelle séquence en O(n^2 x m), m étant le nombre de séquences d'entraînement

# Classification basée sur des modèles

- Modèle de Markov Caché
  - Modèle statistique dans lequel les séquences sont modélisées comme des processus de Markov (processus sans mémoire) avec des états cachés.
  - Décrit l'évolution d'un ensemble d'évènements observables (la séquence de caractères) qui dépendent de facteurs non observables (une séquence d'états cachés).
  - Les états cachés forment une chaîne de Markov, et la distribution des caractères observables dépend des états cachés
  - Chaîne de Markov
    - Pour une séquence d'états {s1,s2,s3,….,sn}:
      $P(sn \mid sn-1,sn-2,\ldots,s1) = P(sn \mid sn-1)$

# Classification basée sur des modèles

## ❑ Modèle de Markov Caché caractérisé par:




- Un ensemble d'états
  $Q = \{q_0, q_1, q_2, q_3, q_4\}$

- La distribution initiale des états:
  $\prod$ tel que la somme des probas initiales = 1

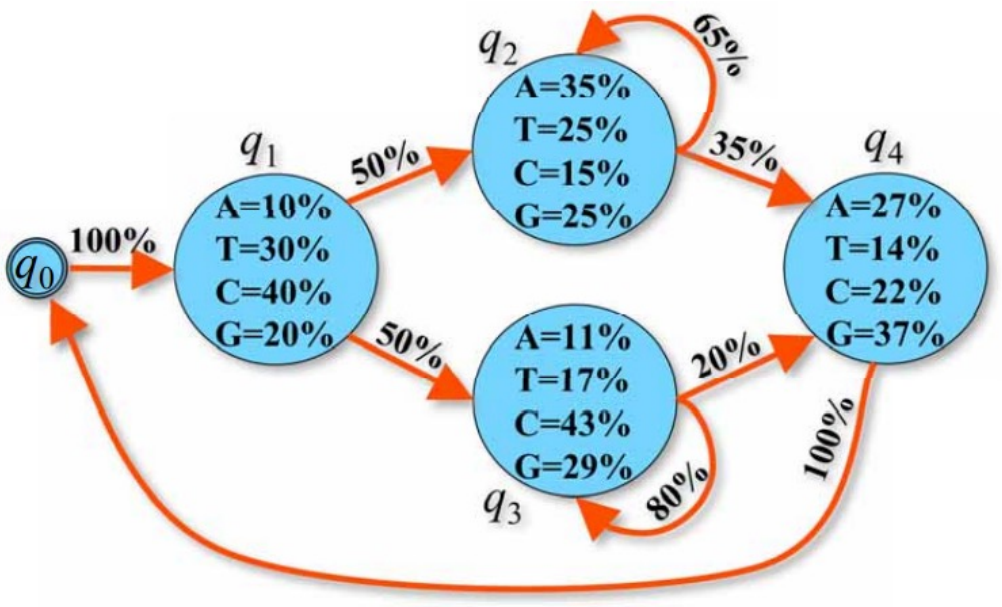
- Une matrice |Q| x |Q| des probabilités de transition entre états
  $a_{ij}$ : probabilité de la transition $q_i \rightarrow q_j$ ;
  (somme des probabilités sortantes = 1)

- Un alphabet des caractères émis $\sum = \{A,T,C,G\}$

- Une matrice |Q| x |$\sum$| des probabilités d'émission
  $b_{ik}$ : probabilité d'émettre un caractère k dans l'état $q_i$ ;

# Classification basée sur des modèles

- Modèle de Markov Caché
  Trois principales questions:



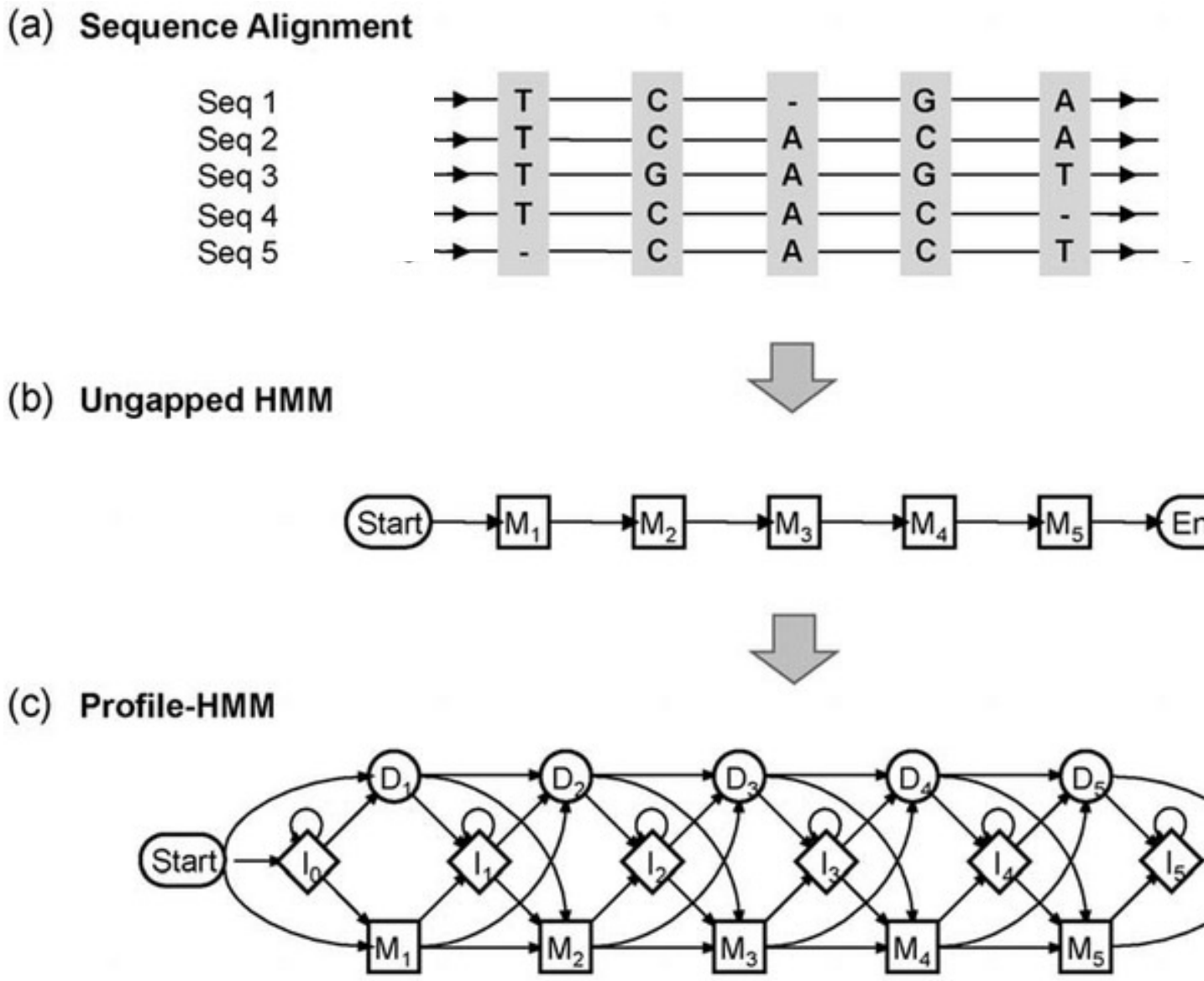

- **Apprentissage** : Étant donné un ensemble de séquences observées, trouver le modèle le plus probable
  - **Apprentissage supervisé** si les séquences d'états associées aux séquences observées sont données
  - Sinon **apprentissage non-supervisé** (algorithme de Baum-Welch)

- **Évaluation :** Étant donnés un modèle M et une séquence observée S, calculer la probabilité que la séquence S ait été produite par le modèle M (algorithme Forward)

- **Décodage** : Étant donnés un modèle M et une séquence observée S, calculer la séquence d'états la plus probable qui aurait pu générer la séquence S. (algorithme de Viterbi)

# Classification basée sur des modèles

- **Profile-HMM**: Modèle de Markov Caché pour représenter un alignement de séquences




Source: Byung-Jun Yoon, Current Genomics, 2009, 10, 402-415.

Structure linéaire

$M_k$ ***Match*** *states*

3 types d'états

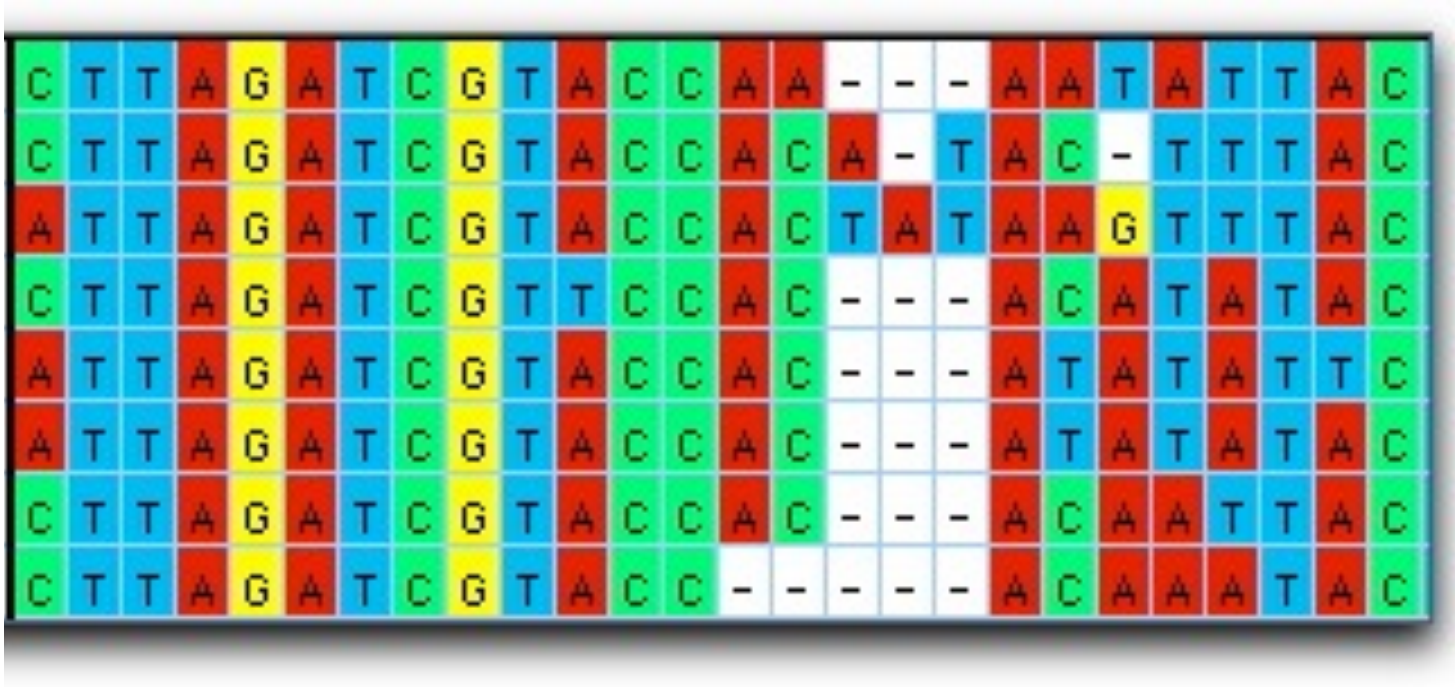
$M_k$ ***Match*** *states*

$I_k$ ***Insert*** *states*

$D_k$ ***Delete*** *states*

# Classification basée sur des modèles

- Alignement multiple de séquences

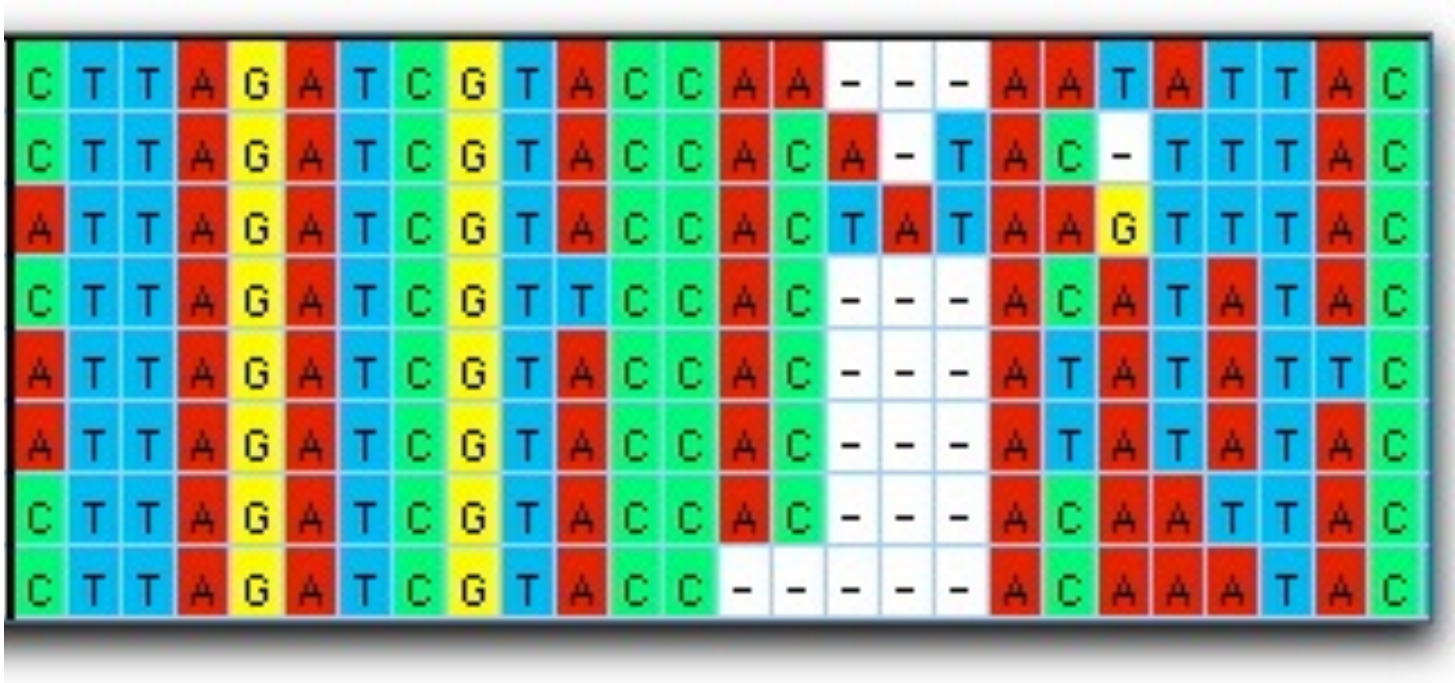



M M M M M M M M M M M M M M M I I I M M M M M M M M

Déterminer les états correspondants Aux colonnes de l'alignement

# Classification basée sur des modèles

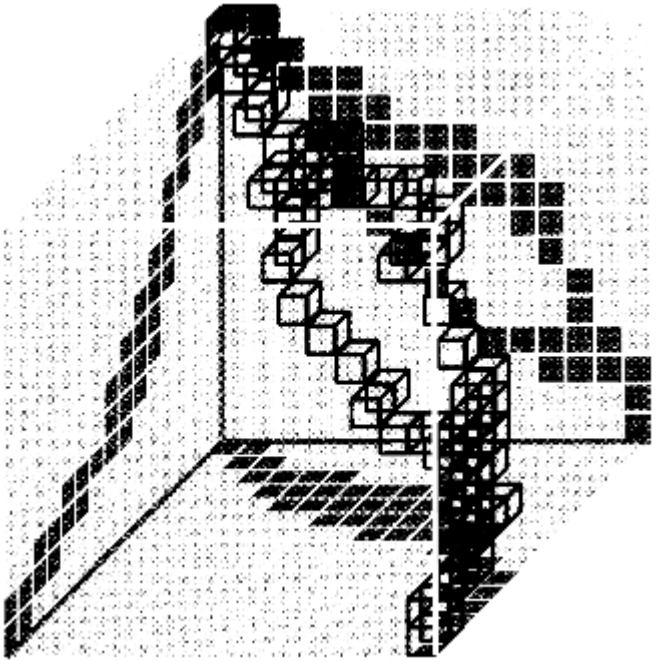
- Alignement multiple de séquences




- **Problème**: pour un ensemble de séquences, trouver un alignement multiple qui minimise un coût ou maximise un score définie sur l'ensemble des alignements multiples possibles
**Exemple de score**: la somme des scores deux-à-deux induits

# Classification basée sur des modèles

- Alignement multiple de séquences



Programmation dynamique multi-dimensionnelle.
Exemple: 3 dimensions pour 3 séquences

- Algorithme de programmation dynamique exacte en temps O(n^k), n étant la longueur maximum des séquences et k le nombre de séquence ➔ Algorithme exponentiel
- Utilisation d'algorithmes heuristiques pour le calcul de l'alignement multiple à partir d'alignement deux à deux

# Classification basée sur des modèles

- Pour chaque classe de séquences Ci : {C1,C2,…Cm} :
  - Construire un alignement multiple Ai de séquences
  - Construire un Profile-HMM Mi correspondant

- Pour chaque nouvelle séquence S:
  - Pour chaque Profile-HMM Mi
    - Évaluer la probabilité Pi que S soit produite par Mi
    - Attribué à S la classe correspondant au Pi maximum

# Classification basée sur des modèles

- Avantages
  - Fonctionne bien si le modèle représente bien les caractéristiques conservées entre les séquences
  - Tient compte de l'ordre des caractères

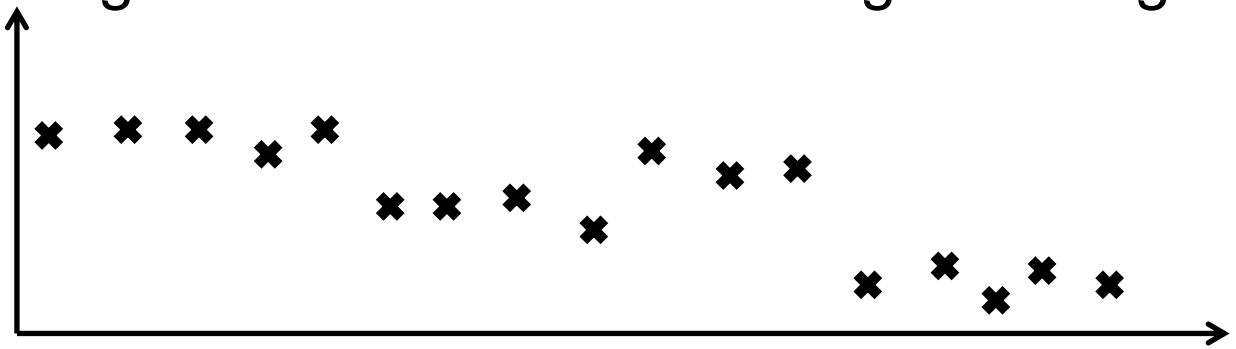
- Limites
  - Nécessite de calculer des alignements
  - Nécessite une connaissance a priori des séquences pour bien fixer les paramètres de l'alignement
  - La performance dépend de la qualité de l'alignement

# Segmentation de séquences

- **Objectif**:
  - Découper la séquence en segments d'éléments similaires
  - Compresser la séquence

- **Problème de k-segmentation**: étant donné une séquence S = $s_1$ $s_2$ … $s_n$, trouver un découpage de S en k segments [$s_{i\_1}$ $s_{i\_2}$[, [$s_{i\_2}$ $s_{i3}$[, …, [$s_{ik}$ $s_{ik+1}$[ tel que la somme des carrés de la distance de chaque élément à la moyenne de son segment est minimale.
  (SSE = sum of square errors)

- Similaire au clustering mais les parties doivent être des segments aux éléments contigus le long de la séquence.

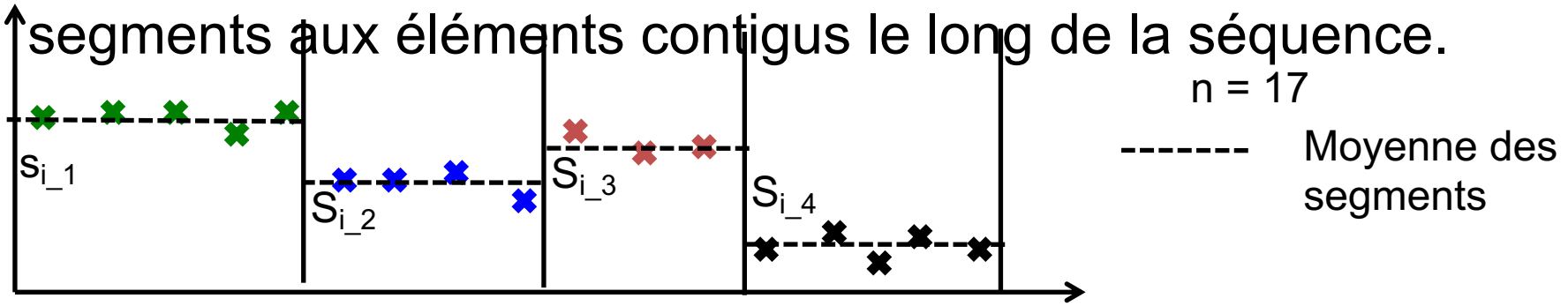


# Segmentation de séquences

- **Objectif**:
  - Découper la séquence en segments d'éléments similaires
  - Compresser la séquence

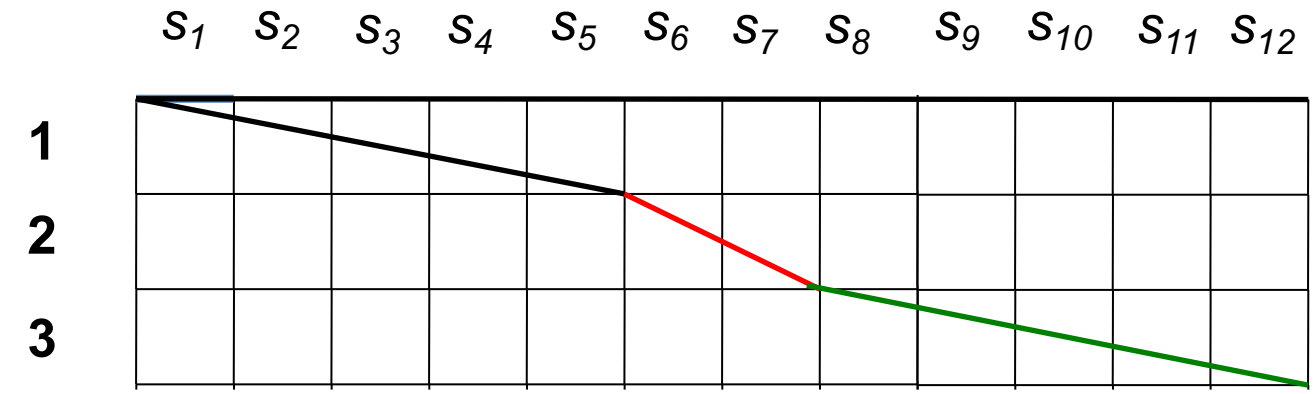
- **Problème de k-segmentation**: étant donné une séquence S = $s_1\ s_2\ s_3\ \ldots\ s_n$, trouver un découpage de S en k segments $[s_{i\_1}=s_1, s_{(i\_2)-1}]$, $[s_{i\_2}, s_{(i\_3)-1}]$, …, $[s_{i\_(k-1)}, s_{(i\_k)-1}]$, $[s_{i\_k}, s_{i\_(k+1)}=s_n]$ tel que le coût de la segmentation est minimal:
  Exemple : coût d'une segmentation = la somme des carrés de la distance de chaque élément à la moyenne de son segment (SSE = sum of square errors)

- Similaire au k-clustering mais les parties doivent être des segments aux éléments contigus le long de la séquence.



# Programmation dynamique pour le calcul d'une k-segmentation optimale

**Une k-segmentation** :
Un chemin du point gauche-haut vers le point droit-bas composé de k segments.




segment 1

segment 2

segment 3

**Espace de recherche** : Ensemble de tous les chemins composés de k segments, du point gauche-haut vers le point droit-bas.

# Programmation dynamique pour le calcul d'une k-segmentation optimale

- Méthode de résolution de problème consistant à résoudre successivement des sous-problèmes par ordre d'inclusion croissante.

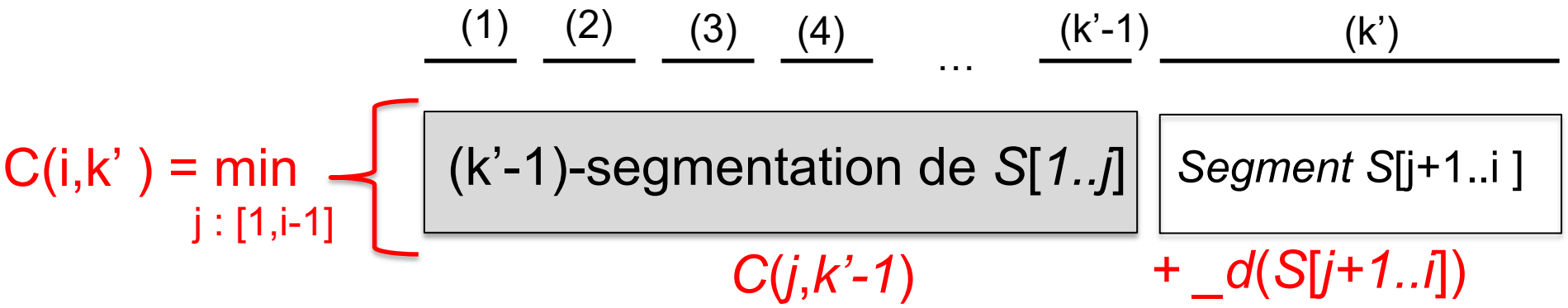
- Utilisation d'un tableau de programmation dynamique pour stocker les solutions des sous-problèmes afin de ne pas les recalculer deux fois.

- Problème : calculer C($S$,$k$)
  - Calculer C($n$,$k$) = $C(S[1..n], k)$ ; $n = |S|$

- Sous-problèmes : calculer les coûts pour les préfixes de $S$
  - Calculer $C(i,k') = C(S[1..i],k')$ *; 1 <= i <= n et 1 <= k' <= k*

# Programmation dynamique pour le calcul d'une k-segmentation optimale

- Expression de $C(i,k')$ en fonction des coûts de (k'-1)-segmentation de préfixes de S[1..i]



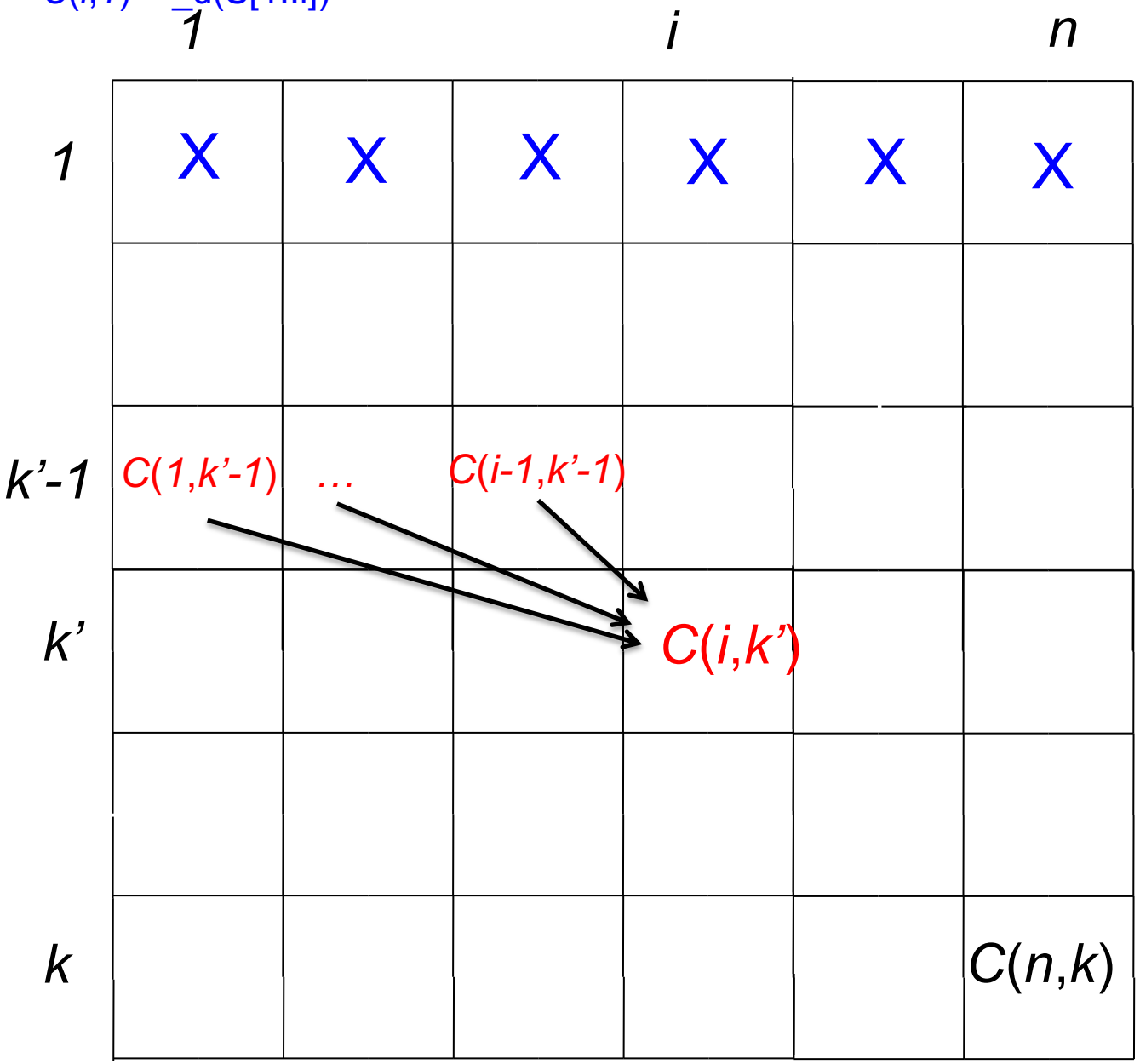

$d(s_i , s_j)$ = *distance entre deux éléments*

$\_d(s_1\ s_2\ s_3\ \ldots\ s_n) = \text{Somme}_{i:[1..n]} \{ d(s_i , \text{moyenne}(s_1,s_2,s_3,s_n))^2 \}$

# Programmation dynamique pour le calcul d'une k-segmentation optimale

Initialisation

$C(i,1) = \_d(S[1..i])$

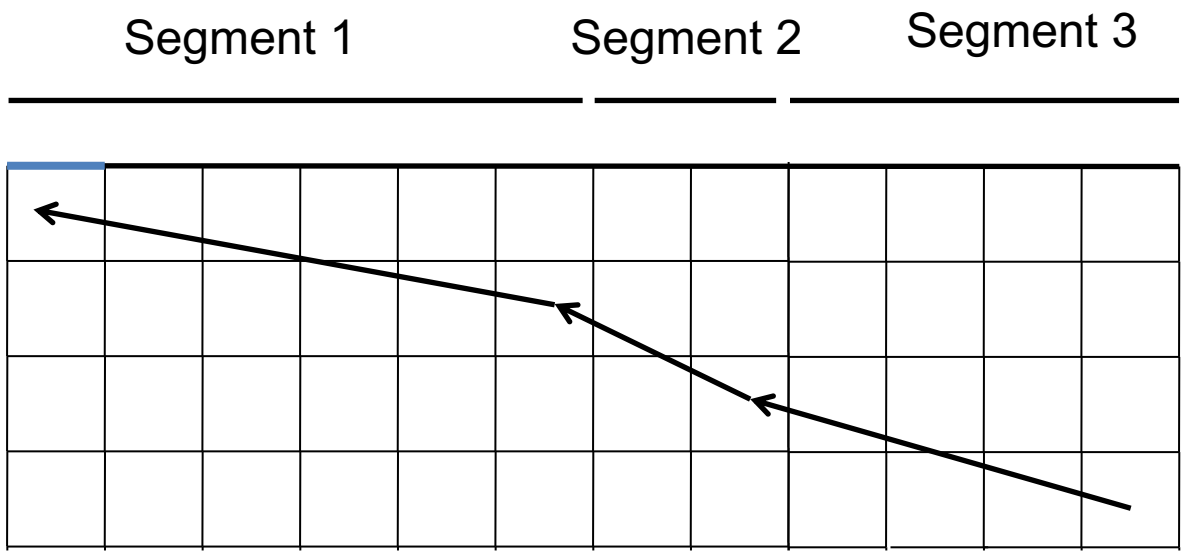



Complexité: $O(kn^2)$

- pré-calcul de _d(S[i..j]) pour tout i, j tels que $1 <= i <= j <= n$ : $O(n^2)$
- k x n cellules : O(kn)
- Calcul pour chaque cellule : O(n)

# Programmation dynamique pour le calcul d'une k-segmentation optimale

Trace retour:
Pour retrouver la
K-segmentation



# Algorithmes heuristiques pour la k-segmentation

- Approche hiérarchique agglomérative: O(nlog(n))
  - $S = s_1\ s_2\ s_3\ \ldots\ s_n$
  - Commencer avec n segments contenant chacun un élément
  - De façon itérative, fusionner deux segments consécutifs tels que l'augmentation du coût est minimal jusqu'à obtenir k segments

- Approche hiérarchique divisive: O(nk)
  - $S = s_1\ s_2\ s_3\ \ldots\ s_n$
  - Commencer avec 1 seul segment contenant n éléments
  - De façon itérative, découper un segment en deux segments tels que la diminution du coût est maxiimal jusqu'à obtenir k segments

- Recherche locale O(nk)
  - $S = s_1\ s_2\ s_3\ \ldots\ s_n$
  - Fixer $s_{i\_1}, s_{i\_2}, s_{i\_3}, \ldots, s_{i\_k}$ aléatoirement et les déplacer pour réduire le côut.

# Références

[1] Guozhu DONG, Jian PEI : *Sequence Data Mining*, Advances in Database Systems (Volume 33), 2007.

[2] Jiawei HAN, Micheline KAMBER, Jian PEI. *DataMining: Concepts and Techniques (Third edition)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.

[3] Adreas C. MÜLLER et Sarah GUIDO : *Introduction to Machine Learning with Python*. O'Reilly Media, Inc., Sebastopol, CA, 2017.