Recherche de motifs : Brute Force

Cours 4

Plan du cours

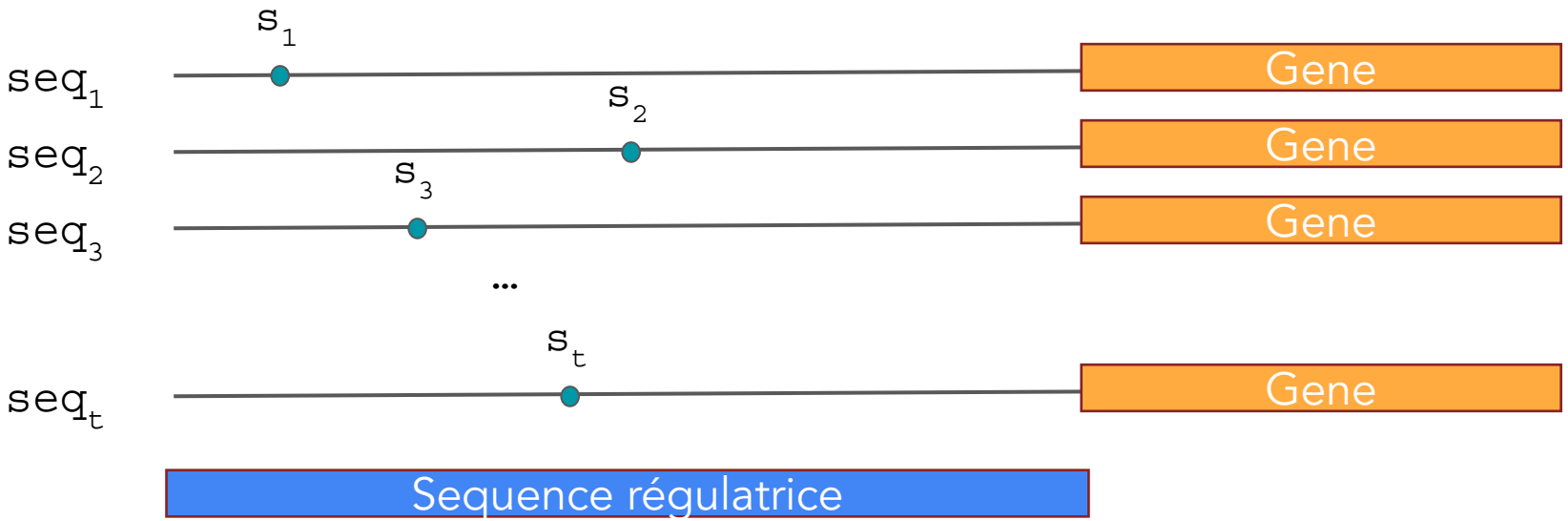
- Algorithme : Brute Force
- Exemple
- Complexité

Algorithme: Brute Force

- L'algorithme Brute Force (BF) peut trouver des motifs **variables** de taille k dans les séquences régulatrices.
  - BF explorent tous les positions
  - Complexité eleve

Algorithme: Brute Force

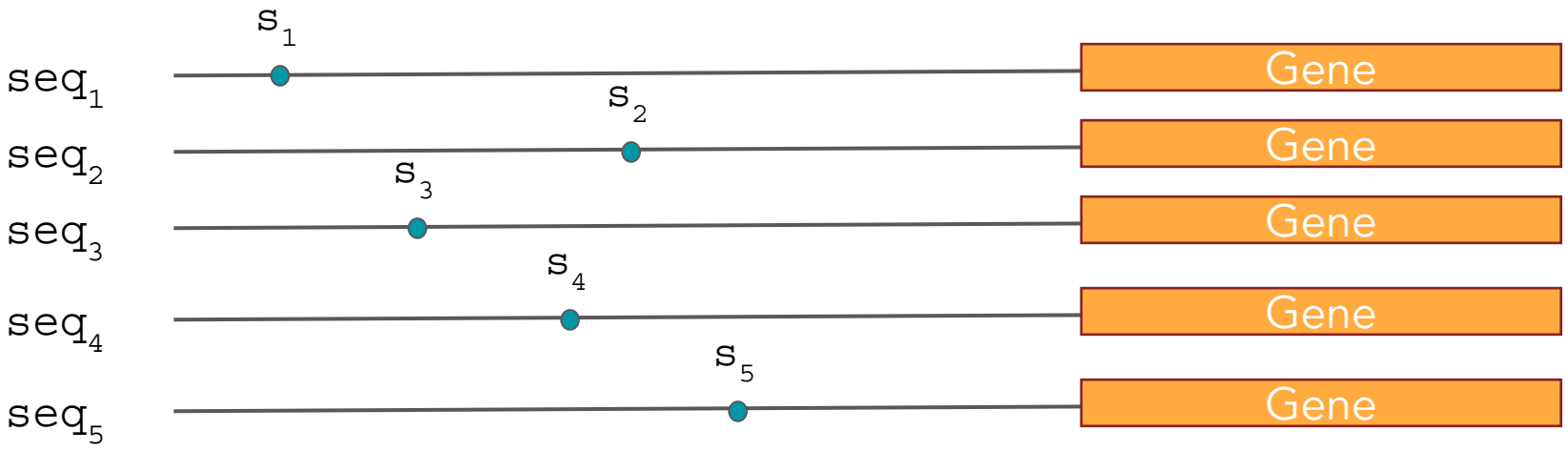
- Comment trouver des motifs variables?
  - Disons que nous savons où le motif commence dans chaque séquence.
    - Les positions de début de motif dans leurs séquences peuvent être représentées par un vecteur s = (s_1, s_2, …, s_t)



- Comment trouver des motifs variables?

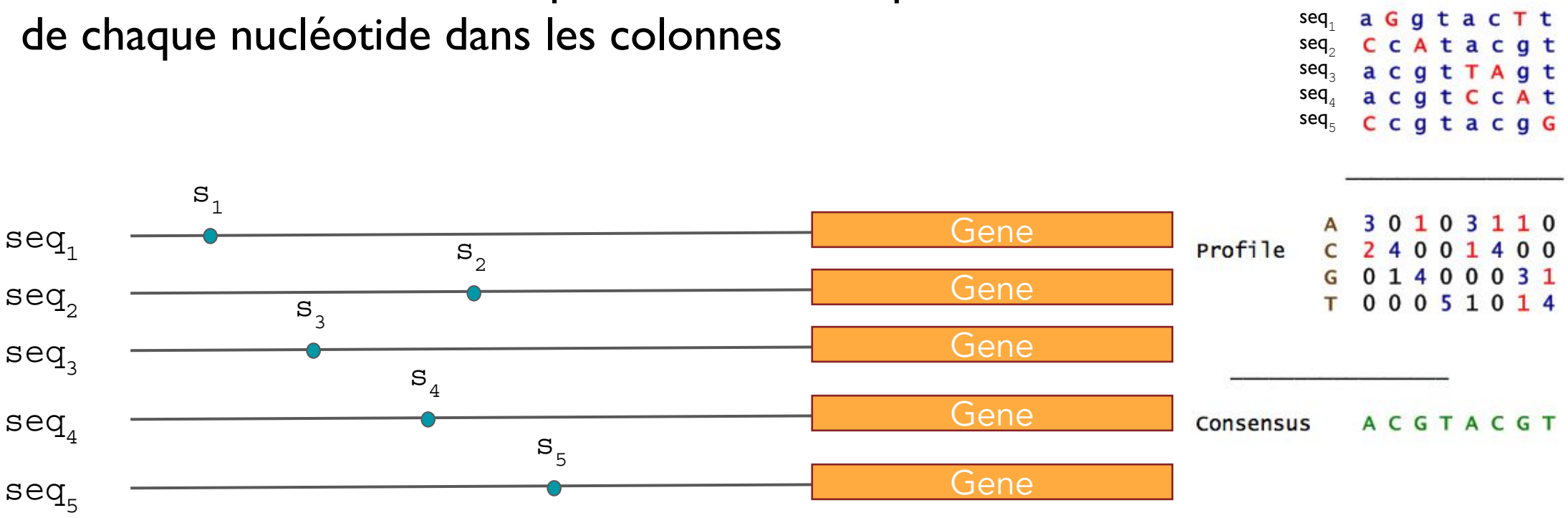
- Extraire et aligner les modèles à partir de leurs index de départ s = (s_1, s_2, …, s_t)

seq_1 a G g t a c T t
seq_2 C c A t a c g t
seq_3 a c g t T A g t
seq_4 a c g t C c A t
seq_5 C c g t a c g G



- Comment trouver des motifs variables?

  - Construire une matrice de profil avec les fréquences de chaque nucléotide dans les colonnes



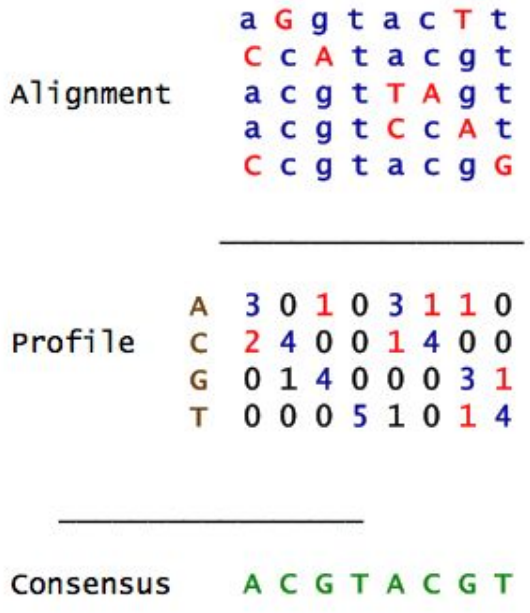
  - Le nucléotide consensus dans chaque position a le plus haut score dans la colonne

- Comment trouver des motifs variables?

- Extraire et aligner les modèles à partir de leurs index de départ $s = (s_1, s_2, \ldots, s_t)$

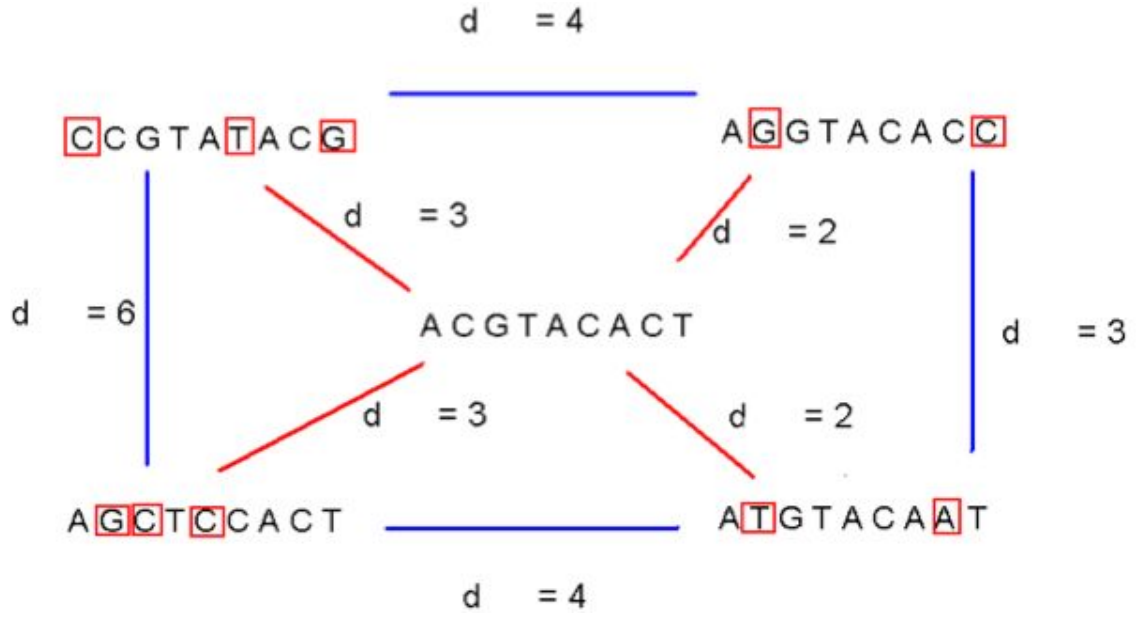
- Construire une matrice de profil avec les fréquences de chaque nucléotide dans les colonnes

- Le nucléotide consensus dans chaque position a le plus haut score dans la colonne



- **L'idée principale est : nous voulons choisir les positions de départ que nous donner la meilleure séquence consensus**

- <u>Sequence consensus</u>
  - Pensez à la séquence consensus comme un motif «ancêtre», à partir duquel des motifs mutés ont émergé
  - La distance entre un motif réel et la séquence consensus est généralement inférieure à la distance entre deux motifs réels



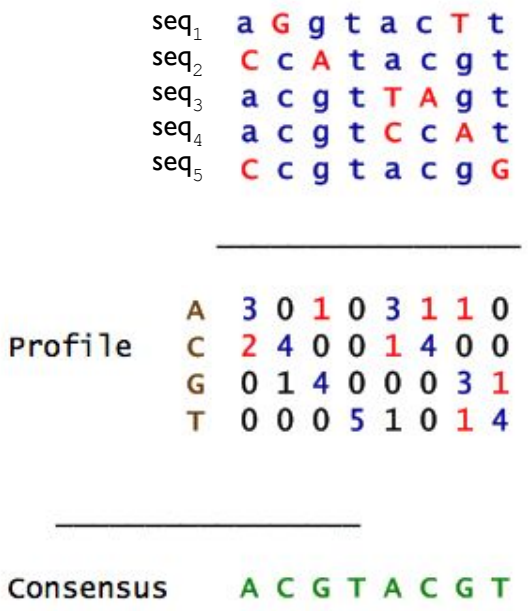
- Evaluer la séquence consensus

  - Nous avons une estimation de la séquence consensus, mais à quel point ce consensus est-il «bon»?

  - Nous pouvons définir une fonction d'évaluation pour comparer différentes séquences consensus.

  - **L'idée principale est : nous voulons choisir les positions de départ que nous donner la meilleure séquence consensus**

- # Evaluer la séquence consensus : exemple



Score = 3+4+4+5+3+4+3+4 = 30

Calculer un score, la somme de fréquences de chaque nucléotide de la séquence consensus

- Algorithme: Brute Force - Definition formelle

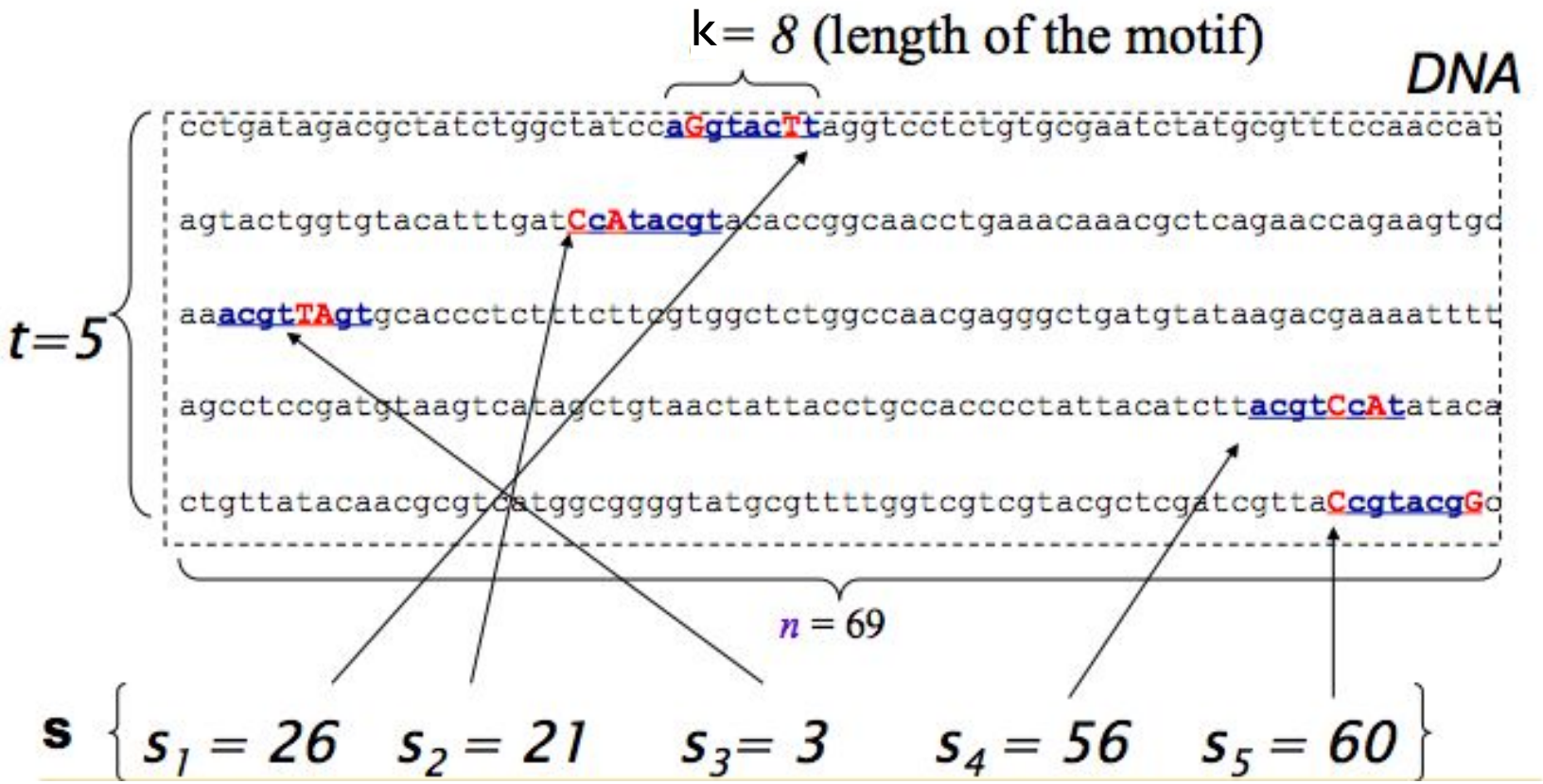
  - But: À partir d'un ensemble de séquences d'ADN (t x n), trouver un ensemble de positions de départs, une pour chaque séquence, qui maximise le score de la séquence consensus

  - Entrée: une matrice t x n de séquences d'ADN, et k, la longueur du motif à trouver

  - Sortie: un vecteur de t positions de départ $s = (s_1, s_2, \ldots, s_t)$ maximisant le score (s, profile)

- Paramètre du algorithme
  - t - nombre de séquences d'ADN
  - n - longueur de chaque séquence d'ADN
  - ADN - échantillon de séquences d'ADN (stocké sous la forme d'un tableau t x n)
  - k - longueur du motif
  - s_i - position de départ d'un motif dans la séquence i
  - s = (s_1, s_2, …, s_t) positions de départ de chaque motif

- Paramètre du algorithme



Brute Force Method

- Calculer les scores pour chaque combinaison possible de positions de départ s.
- Le meilleur score déterminera le meilleur profile et la meilleure séquence consensus.
- Le but est de maximiser Score (s, profile) en faisant varier les positions de départ s_i, où:

$$1 <= s_i <= n-k+1$$

Brute Force

```
1.  BruteForceMotifSearch(DNA, t, n, k)
2.  bestScore ← 0
3.  for each s=(s_1,s_2 , . . ., s_t) from (1, 1, …, 1)
                                      to (n - k +1, …, n - k +1)
4.      if (Score(s,DNA) > bestScore)
5.          bestScore ← score(s, DNA)
6.          bestMotif ← (s_1,s_2 , . . . , s_t)
7.  return bestMotif
```

Brute Force Method : complexité

- Variant (`n - k + 1`) positions dans chacune des t séquences, nous regardons $(n - k + 1)^t$ ensembles s.

- Pour chaque s, la fonction d'évaluation fait k opérations, donc la complexité de l'algorithme est

- $k\,(n - k + 1)^t = O(kn^t)$

- Cela signifie que pour t = 8, n = 1000, k = 10, nous devons effectuer environ $10*1000^8 = 10^{25}$ calculs - l'algorithme prendra des milliards d'années pour se terminer sur une telle instance de problème

A retenir

- L’algorithme BF peut trouver des motifs **invariables** de taille **k** dans les séquences régulatrices.
- Il explore tout l’espace de recherche en variant toutes les positions de chaque séquence.
- Complexité ~ $O(kn^t)$