



UNIVERSITÉ DE LIÈGE

---

## Recherche de motifs dans des séquences d'ADN

---

INFO0902-1 – Structure de données et algorithmes

François ROZET (s161024)

Bachelier Ingénieur civil  
Année académique 2018-2019

## Table des matières

<b>2</b>	<b>Analyse théorique</b>	<b>1</b>
2.1	Recherche dichotomique par force brute ou table de hachage . . . . .	1
2.2	Fonction de coût . . . . .	3
2.3	Programmation dynamique . . . . .	3
2.4	Complexité . . . . .	3
<b>3</b>	<b>Implémentation</b>	<b>5</b>
<b>4</b>	<b>Analyse empirique</b>	<b>6</b>

## 2 Analyse théorique

### 2.1 Recherche dichotomique par force brute ou table de hachage

Pour ne pas utiliser d'espace mémoire inutilement, les différents algorithmes retournent des triplets  $(l, i, j)$  tels que  $l$  est la longueur de la sous-séquence contiguë commune aux séquences  $X$  et  $Y$  la plus longue et  $X[i + k] = Y[j + k]$  pour tout  $0 \leq k < l$ .

---

**Pseudo-code 1** Recherche dichotomique par force brute
 

---

```

1 function DICHOTOMIC-SEARCH( $X, Y$ )
2    $t = NIL$ 
3    $p = 1; r = \min(X.length, Y.length)$ 
4   while  $p \leq r$  do
5      $q = \lfloor \frac{p+r}{2} \rfloor$ 
6      $s = \text{BRUTE-FORCE-AUX}(X, Y, q)$ 
7     if  $s \neq NIL$  then
8        $p = q + 1$ 
9     else
10       $r = q - 1$ 
11       $t = s$ 
12  return  $t$ 

```

---



---

**Pseudo-code 2** Recherche dichotomique par table de hachage
 

---

```

1 function HASH-TABLE-SEARCH( $X, Y$ )
2    $t = NIL$ 
3    $p = 1; r = \min(X.length, Y.length)$ 
4   while  $p \leq r$  do
5      $q = \lfloor \frac{p+r}{2} \rfloor$ 
6      $s = \text{HASH-TABLE-AUX}(X, Y, q)$ 
7     if  $s \neq NIL$  then
8        $p = q + 1$ 
9     else
10       $r = q - 1$ 
11       $t = s$ 
12  return  $t$ 

```

---



---

**Pseudo-code 3** Recherche auxiliaire par force brute
 

---

```

1 function BRUTE-FORCE-AUX( $X, Y, l$ )
2   for  $i = 1$  to  $X.length - l + 1$  do
3     for  $j = 1$  to  $Y.length - l + 1$  do
4       for  $k = 1$  to  $l$  do
5         if  $X[i + k - 1] \neq Y[j + k - 1]$  then
6            $k = 0$ 
7           break
8       if  $k \neq 0$  then
9         return  $(l, i, j)$ 
10  return  $NIL$ 

```

---

Pour ne pas perdre la trace de la position des sous-séquences dans  $X$ , il est nécessaire de la stocker dans la table de hachage. Pour ce faire, on utilise une *map* qui n'est autre qu'une table de hachage dont la fonction d'insertion HASH-INSERT prend en entrée une clé en plus de l'élément à stocker. C'est selon cette clé que l'élément est rangé dans la structure de données.

Dans notre cas, la clé est  $f$ , l'encodage des sous-séquences, et l'élément est  $i$ , la position de leur premier élément.

---

**Pseudo-code 4** Recherche auxiliaire par table de hachage sans calcul incrémental

---

```

1 function HASH-TABLE-AUX1( $X, Y, l$ )
2   Let  $H$  be a new hash table
3   for  $i = 1$  to  $X.length - l + 1$  do
4      $f = \text{ENCODE}(X, i, l)$ 
5     HASH-INSERT( $H, f, i$ )
6   for  $j = 1$  to  $Y.length - l + 1$  do
7      $f = \text{ENCODE}(Y, j, l)$ 
8      $i = \text{HASH-SEARCH}(H, f)$ 
9     if  $i \neq \text{NIL}$  then
10      return  $(l, i, j)$ 
11  return  $\text{NIL}$ 

```

---



---

**Pseudo-code 5** Recherche auxiliaire par table de hachage avec calcul incrémental

---

```

1 function HASH-TABLE-AUX2( $X, Y, l$ )
2   Let  $H$  be a new hash table
3    $b = 4; p = b^{l-1}$ 
4    $f = \text{ENCODE}(X, 1, l)$ 
5   HASH-INSERT( $H, f, 1$ )
6   for  $i = 2$  to  $X.length - l + 1$  do
7      $f = (f - X[i - 1] \cdot p) \cdot b + X[i + l - 1]$ 
8     HASH-INSERT( $H, f, i$ )
9    $f = \text{ENCODE}(Y, 1, l)$ 
10   $i = \text{HASH-SEARCH}(H, f)$ 
11  if  $i \neq \text{NIL}$  then
12    return  $(l, i, 1)$ 
13  for  $j = 2$  to  $Y.length - l + 1$  do
14     $f = (f - Y[j - 1] \cdot p) \cdot b + Y[j + l - 1]$ 
15     $i = \text{HASH-SEARCH}(H, f)$ 
16    if  $i \neq \text{NIL}$  then
17      return  $(l, i, j)$ 
18  return  $\text{NIL}$ 

```

---

---

**Pseudo-code 6** Fonction d'encodage

---

```

1 function ENCODE( $X, i, l$ )
2    $b = 4$ 
3    $f = 0$ 
4   for  $j = 1$  to  $l$  do
5      $f = f \cdot b + X[i + j - 1]$ 
6   return  $f$ 

```

---

**2.2 Fonction de coût**

De façon similaire à l'exemple donné dans le cours, la fonction de coût est

$$C[i, j] = \begin{cases} C[i-1, j-1] + 1 & \text{si } i, j > 0 \text{ et } X[i] = Y[j] \\ 0 & \text{sinon} \end{cases} \quad (1)$$

On notera que le cas  $X[i] \neq Y[j]$  est absorbé dans le cas de base.

**2.3 Programmation dynamique**

Le maximum de la fonction de coût indiquant le dernier élément de la sous-séquence recherchée, il est aisé de trouver cette dernière à partir de la matrice. Cependant, pour éviter de devoir la (re-)parcourir pour trouver son maximum, ce dernier est déterminé pendant la construction.

---

**Pseudo-code 7** Recherche par programmation dynamique

---

```

1 function DYNAMIC-SEARCH( $X, Y$ )
2   Let  $C[1 \dots X.length + 1, 1 \dots Y.length + 1]$  be a new table
3    $l = 0$ 
4   for  $i = 1$  to  $X.length + 1$  do
5     for  $j = 1$  to  $Y.length + 1$  do
6       if  $i > 1$  and  $j > 1$  and  $X[i-1] == Y[j-1]$  then
7          $C[i, j] = C[i-1, j-1] + 1$ 
8         if  $C[i, j] > l$  then
9            $l = C[i, j]$ 
10           $i_{end} = i - 1$ 
11           $j_{end} = j - 1$ 
12        else  $C[i, j] = 0$ 
13  if  $l \neq 0$  then
14    return  $(l, i_{end} - l + 1, j_{end} - l + 1)$ 
15  return  $NIL$ 

```

---

**2.4 Complexité**

Soit  $n$  la longueur des deux séquences et  $m$  la longueur de leur sous-séquence contiguë commune la plus longue.

- (a) La solution par force brute est composée de quatre boucles imbriquées. Les boucles primaire et intermédiaires exécutent toujours de l'ordre de  $n - m$  itérations. La boucle finale, quant à elle, en exécute de l'ordre de  $m$ . Les opérations à l'intérieur de ces boucles étant toutes  $\mathcal{O}(1)$ , la complexité en temps est

$$\mathcal{O}(m(n - m)^3) \quad (2)$$

Néanmoins, lorsque  $m$  tend vers  $n$ , les trois premières boucles voient leur nombre d'itérations tendre vers 1. Ainsi, la complexité en temps est  $\Omega(n)$ . On trouve aussi que  $m = \frac{n}{4}$  est le pire cas, bien qu'il soit équivalent au cas moyen en terme de complexité.

En terme d'espace, cet algorithme n'utilise aucune structure de données supplémentaire aux séquences testées. Dès lors, sa complexité en espace est  $\Theta(1)$ .

- (b) Comme la solution par force brute, la recherche dichotomique est composée de quatre boucles imbriquées. Cependant, ici, la boucle primaire s'arrête lorsque un interval de taille  $n$  a été divisé *suffisamment* de fois, c.-à-d.  $\log_2(n)$  fois. De plus, les boucles intermédiaires sont maintenant  $\mathcal{O}(n)$ . Ainsi, la complexité en temps est

$$\mathcal{O}(mn^2 \log(n)) \quad (3)$$

Pour cette solution, lorsque  $m$  tend vers  $n$ , les boucles intermédiaires ne voient pas leur nombre d'itérations tendre vers 1, sauf si, en plus, la sous-séquence commune se situe au début des deux séquences testées. Ainsi, la complexité en temps est  $\Omega(n \log(n))$ . Le pire cas est équivalent au cas moyen.

Concernant la complexité en espace, cette solution est équivalente à la force brute.

- (c) La solution par table de hachage ne possède que deux boucles imbriquées qui correspondent aux deux premières de la recherche dichotomique. Néanmoins, sans calcul incrémental, l'opération d'encodage exécute  $\mathcal{O}(n)$  opérations à chaque appel. En supposant que l'insertion et la lecture dans la table de hachage sont faites en  $\mathcal{O}(1)$ , la complexité en temps est

$$\Theta(n^2 \log(n)) \quad (4)$$

En terme d'espace, à chaque appel de la fonction auxiliaire pour une taille  $l$  de sous-séquence, il est nécessaire d'initialiser une table de hachage sur un univers  $U = \{0, 1, \dots, 4^l - 1\}$  afin d'y stocker  $n - l + 1$  sous-séquences. Pour conserver un accès (insertion, recherche et suppression) aux éléments de la table en  $\mathcal{O}(1)$ , il est nécessaire d'avoir  $p = \Omega(n - l)$  où  $p$  est sa taille<sup>1</sup>. Or, la valeur de départ de  $l$  est  $\frac{n}{2}$ , dès lors, la complexité en espace de la solution par table de hachage est  $\Theta(n)$ .

- (d) Avec le calcul incrémental, l'encodage ne demande plus que  $\mathcal{O}(1)$  opérations par appel. Ainsi, la complexité en temps devient

$$\Theta(n \log(n)) \quad (5)$$

La complexité en espace n'a, quant à elle, pas changé.

---

1. Dans le cas d'une table de hachage par adressage direct, la condition  $p \geq n - l + 1$  doit aussi être respectée.

- (e) La solution par programmation dynamique n'utilise que deux boucles imbriquées itérant sur l'entière des séquences. Dès lors, la complexité en temps est

$$\Theta(n^2) \tag{6}$$

De plus, la matrice de coût possédant  $(n + 1)^2$  éléments, la complexité spatiale est aussi  $\Theta(n^2)$ .

### 3 Implémentation

Les algorithmes ayant été implémentés à l'aide du langage C++, il est nécessaire de les compiler pour obtenir des programmes exécutables. Par simplicité, le processus de compilation a été automatisé à l'aide d'un `Makefile` qui construit, par défaut, les exécutables `lcsdicho`, `lcshash` et `lcsdp`.

Ces derniers prennent en entrée, comme demandé, deux fichiers contenant des séquences génomiques et le nombre de nucléotides à prendre en compte. Cependant, afin de pouvoir mesurer plus facilement les performances des différents algorithmes, le nombre de nucléotides considérés `NNN` ainsi que le temps de calcul `TTT` ont été rajoutés à l'entête des fichiers FASTA renvoyés.

```
> L: XXX G1: YYY G2: ZZZ N: NNN T: TTT
```

De plus, si le fichier FASTA ciblé existe déjà, la nouvelle sous-séquence sera ajoutée à la fin du fichier.

Aussi, pour ne pas restreindre l'application des algorithmes aux séquences génomiques, ils ont été implémentés de manière *générique*, c.-à-d. en tant que *template*.

#### Table de hachage

Comme il a été discuté dans la section 2.1, la table de hachage utilisée est une *map*. Dans le langage C++, cette dernière est pré-implémentée dans la bibliothèque standard sous le nom de `unordered_map`. De plus, la longueur  $p$  de cette table est fixée à  $n - l + 1$  ce qui vérifie toujours  $p = \Omega(n - l)$  (cf. section 2.4).

#### Programmation dynamique

Pour ne pas devoir gérer manuellement l'allocation (et désallocation) de mémoire de la table  $C$ , cette dernière est initialisée en tant que pointeur intelligent (*smart pointer*), lui aussi pré-implémenté dans la bibliothèque standard.

## 4 Analyse empirique

### Recherche dichotomique

Bien que la courbe ne soit pas parfaitement lisse, on devine à la Figure 1 une relation d'ordre 2.

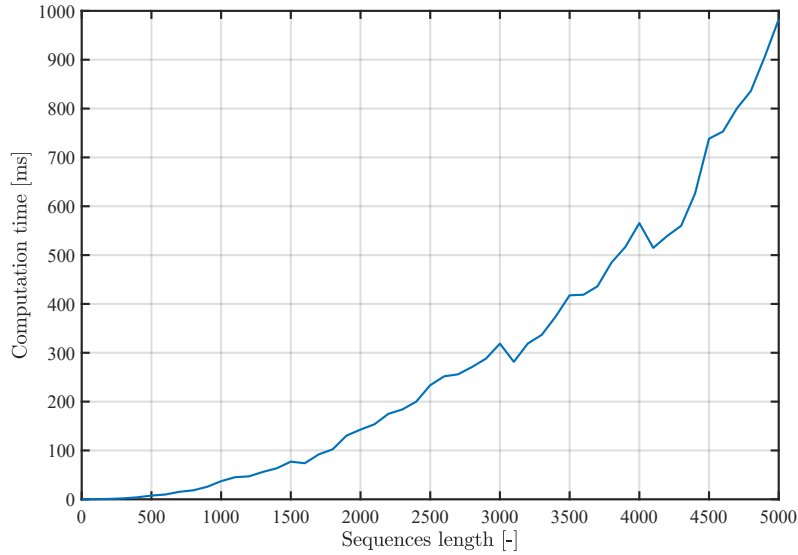


Figure 1 – Évolution du temps de calcul de la recherche dichotomique par force brute en fonction de la taille des séquences.

Or, d'après l'analyse théorique (cf. section 2.4), la complexité de la recherche dichotomique par force brute devrait être  $\mathcal{O}(mn^2 \log(n))$ . Cependant, si l'on se penche sur les données collectées, il apparaît que le facteur  $m$  est quasiment constant. De plus,  $\log(n)$  varie assez peu sur l'intervalle  $[0, 5000]$ . Il ne serait donc pas étonnant de confondre les deux relations.

### Table de hachage

Encore une fois, si la courbe comporte beaucoup de bruit<sup>2</sup>, une relation d'ordre 1 reste discernable à la Figure 2.

---

2. L'algorithme étant particulièrement rapide, la mesure de son temps de calcul est fort sensible aux perturbations causées par les autres processus du système.



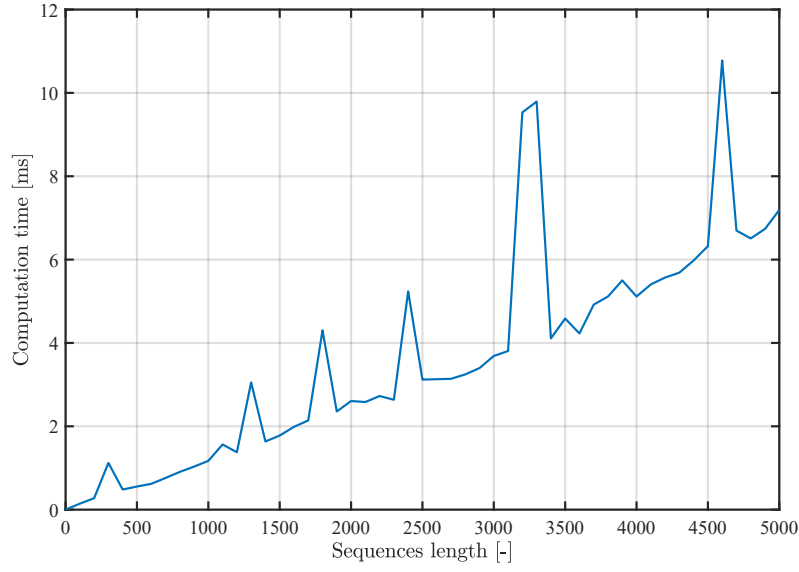


Figure 2 – Évolution du temps de calcul de la recherche dichotomique par table de hachage en fonction de la taille des séquences.

Selon l'analyse théorique, la complexité de la recherche dichotomique par table de hachage (avec calcul incrémental) devrait être  $\Theta(n \log(n))$ . Mais, puisque ici aussi, le facteur  $\log(n)$  varie peu sur l'intervall  $[0, 5000]$ , il n'est pas surprenant d'observer ce qui semble être une relation d'ordre 1.

### Programmation dynamique

Une fois de plus, une relation d'ordre 2 est aisément reconnaissable à la Figure 3.

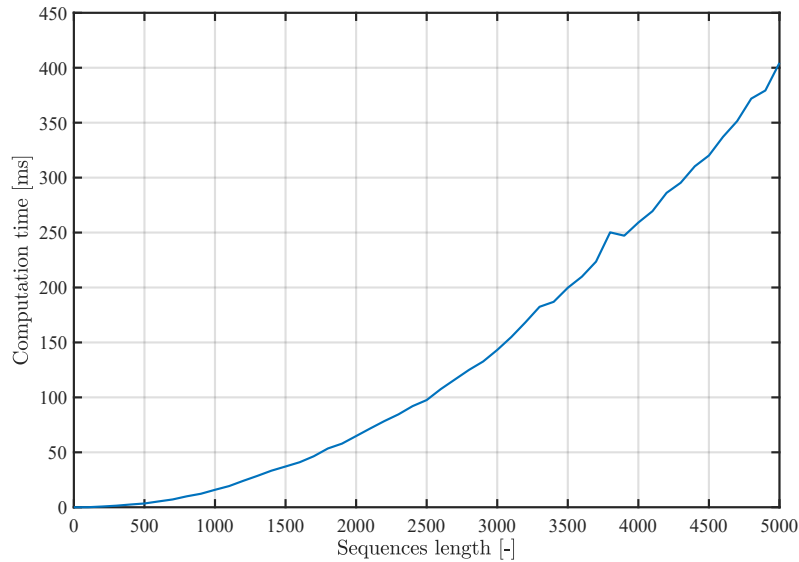


Figure 3 – Évolution du temps de calcul de la recherche par programmation dynamique en fonction de la taille des séquences.

Cette observation correspond à la complexité théorique, à savoir  $\Theta(n^2)$ .