Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Informatique

Bio ALGO

Devoir N°1 : Recherche exacte de motifs

Fait par :
Nom et prénom : ABDELMALEK BENMEZIANE
Matricule : 171731046778
Spécialité : M1 BIOINFO
Section : A

Contents

List of Figures

Chapter 1

Le principe de chaque algorithme

1.1 Boyer Moore

Robert Boyer et J Strother Moore l'ont créé en 1977. L'algorithme de recherche de chaînes B-M est un algorithme particulièrement efficace et a servi de référence standard pour l'algorithme de recherche de chaînes depuis lors.
L'algorithme B-M adopte une approche "en arrière": la chaîne de modèle (P) est alignée avec le début de la chaîne de texte (T), puis compare les caractères d'un modèle de droite à gauche, en commençant par le caractère le plus à droite.
Si un caractère est comparé qui n'est pas dans le modèle, aucune correspondance ne peut être trouvée en analysant d'autres aspects à cette position afin que le modèle puisse être entièrement modifié au-delà du caractère non concordant.
The two strategies are called heuristics of B - M as they are used to reduce the search. They are:

- ✓ Bad Character Heuristics
- ✓ Good Suffix Heuristics

1.1.1 Bad Character Heuristics

Cette heuristique a deux implications:

- Supposons qu'il y ait un caractère dans un texte qui n'apparaisse pas du tout dans un motif. Lorsqu'une incompatibilité se produit au niveau de ce caractère (appelé mauvais caractère), l'ensemble du modèle peut être modifié, commencez à faire correspondre la sous-chaîne de formulaire à côté de ce "mauvais caractère".
- Par contre, il se peut qu'un mauvais caractère soit présent dans le motif, dans ce cas, alignez la nature du motif avec un mauvais caractère dans le texte.

1.1.2 Good Suffix Heuristics

Un bon suffixe est un suffixe qui a été mis en correspondance avec succès. Après une incompatibilité qui a un décalage négatif dans l'heuristique des mauvais caractères, regardez si une sous-chaîne de motif correspondant jusqu'à ce que le mauvais caractère contienne un bon suffixe, si c'est le cas, nous avons un saut en avant égal à la longueur du suffixe trouvé.

1.2 Aho Corasick

L'algorithme d'Aho-Corasick est un algorithme de recherche de chaîne de caractères (ou motif) dans un texte dû à Alfred Aho et Margaret Corasick et publié en 1975. L'algorithme consiste à avancer dans une structure de données abstraite appelée dictionnaire qui contient le ou les mots recherchés en lisant les lettres du texte T une par une. La structure de données est déployée de manière efficace, ce qui garantit que chaque lettre du texte n'est lue qu'une seule fois. Généralement, le dictionnaire est appliqué à l'aide d'une trie ou arbre préfixe auquel on rajoute des suffixes de liens. Une fois le dictionnaire déployé, l'algorithme une complexité linéaire en la taille du texte

T et des chaînes recherchées. L'algorithme extrait toutes les occurrences des motifs. Il est donc possible que le nombre d'occurrences soit quadratique, comme pour un dictionnaire a, aa, aaa, aaaa et un texte aaaa. Le motif a apparaît à quatre reprises, le motif aa à trois reprises, etc.

1.3 Rabin Karp

L'algorithme de Rabin-Karp est un algorithme utilisé pour faire correspondre des modèles dans le texte à l'aide d'une fonction de hachage (filtre de Bloom). Contrairement à l'algorithme de correspondance de chaînes Naive, il ne parcourt pas chaque caractère de la phase initiale, mais filtre les caractères qui ne correspondent pas, puis effectue la comparaison.

1.4 Commentz Walter

L'algorithme de Commentz-Walter : est un algorithme inspiré de Boyer-Moore et Aho-Corasick, il utilise un « trie » inversé pour représenter les motifs à rechercher.

L'algorithme de Commentz-Walter utilise l'heuristique de décalage du mauvais caractère de Boyer Moore pour faire des décalages dans le texte. Il passe lui aussi par 02 étapes :

- Étape de prétraitement.
- Étape de recherche.

1.4.1 Étape de prétraitement

Création du trie CW : un « trie » similaire à celui de Aho- Corasick mais en commençant par la fin du motif et en allant au 1er caractère du motif.

Exemple

K = cactaa, act, ata,actat,cctat inverser les motifs: K=aatcac, tca, ata,tatca,tatcc, puis chaque motif inversé sera inséré dans le « trie » (automate arbre) comme dans Aho-Corasick.

1.4.2 Définir les fonctions shift1 et shift2

Calcul de shift1(v):

1.4.3 Définir Set1 et Set2

- set1 (v) = v‘/ w (v) est un suffixe propre de w (v’).
- set2 (v) = v’ ; v ’est un élément de set1 (v) et ou (v’) $\neq \emptyset$.

Si nœud=racine alors shift1 = 1 et shift2 = wmin (où wmin est la longueur minimale des motifs à rechercher).

sinon, shift1 = min wmin, l; l = d (v ’) - d (v), où d (v) est la profondeur du nœud actuel v dans l’arbre, et v ’est un élément de set1 (v) et shift2 = min shift2 (nœud parent de v), l; l = d (v ’) - d (v), où v’ est un élément de set2 (v),

On commence à lire le texte d’entrée, en plaçant une fenêtre de longueur wmin et faire des comparaisons de droite à gauche dans le texte avec le « trie » construit avec les motifs inversés. Lorsqu’on ne trouve pas un nœud correspondant à un caractère, on fait un décalage dans le texte d’une valeur égale à

min (shift2 (v), max shift1 (v), char (inputText [pos- depth] - depth-1

Avec : pos est la position actuelle dans le texte. depth est la profondeur actuelle dans le « trie » (arbre).

La fonction char est définie comme char (c) = min wmin + 1, d (v), où v est le nœud de non correspondance (échec)

1.5 La complexité de chaque algorithme

En sachant que:

✓ n: la taille du texte

✓ m: la taille du motif

✓ z: le nombre total d'occurrences de motifs dans le texte

✓ wmax: la taille du plus long motif

complexité / Algorithme	meilleur des cas	pire des cas
Boyer Moore	$O(n+m)$	$O(n*m)$
Aho Corasick	$O(n+m)$	$O(n+m+z)$
Rabin Karp	$O(n+m)$	$O(n*m)$
Commentz Walter	$O(n+m)$	$O(n*wmax)$

Chapter 2

Les tableaux des tests

2.1 Boyer Moore

On a pris l'ensemble de tests suivant:

- Text = "The quick brown fox jumped over the lazy dog"
  Pattern = "dog"
- Text = "Hello world, how are you"
  Pattern = "you"
- Text = "Algorithms and Data Structure"
  Pattern = "Structure"
- text = "Walking along the beach, feeling the sand between my toes and listening to the waves crashing on the shore - this is my happy place."
  pattern = "place"
- text = "Lorem Ipsum has been the standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem

Ipsum."

pattern = "Lorem Ipsum"

On aura ce tableau :

Taille de text	24	29	44	100	287
Taille de motif	03	09	05	05	11
Nbr comparaisons	11	20	13	37	88
Temps exécution	0.459	0.417	0.432	0.618	0.798

2.2 Rabin Karp

On a pris l'ensemble de tests suivant:

- text = "The quick brown fox jumped over the lazy dog"
  patterns = ["the", "lazy", "dog"]
- text = "The cat in the hat"
  patterns = ["cat", "hat"]
- text = "ababcbababacbab"
  patterns = ["ababaca", "acbab"]
- text = "ACGTGCCTAGCTGCTA"
  patterns = ["ACGT", "CCTAG", "GCTA"]
- text = "AAACABBAAAABBAAABAAABAABB"
  patterns = ["AB", "AA", "AAB"]

On aura ce tableau :

Taille de text	15	16	18	25	44
Taille des motifs	5|7	4|5	3	2|3	3|4
Nbr comparaisons	27|11	12|13	16	23|24	41|42
Temps exécution	0.709	0.965	0.717	1.330	1.532

2.3 Aho Corasick

On a pris l'ensemble de tests suivant:

- text = "ahishers"
  patterns = ["he", "she", "hers", "his"]
- text = "ababcbababacbab"
  patterns = ["ab", "abab", "bab", "aca"]
- text = "AAACABBAAAABBAAABAAABAABB"
  patterns = ["AB", "AA", "AAB"]
- text = "ACGTGCCTAGCTGCTA"
  patterns = ["ACGT", "CCTAG", "GCTA"]
- text = "Algorithms"
  patterns = ["Algo", "rithms"]

On aura ce tableau :

Taille de text	8	10	15	16	25
Taille des motifs	2|3|4	4|6	2|3|4	4|5	2|3
Nbr comparaisons	8	10	15	16	25
Temps exécution	0.925	0.619	2.186	0.801	4.985

2.4 Commentz Walter

On a pris l'ensemble de tests suivant:

- text = "hi hello world"
  patterns = ["hello", "world", "hi"]
- text = "The quick brown fox jumped over the lazy dog"
  patterns = ["the", "lazy", "dog"]
- text = "ACGTGCCTAGCTGCTA"
  patterns = ["ACGT", "CCTAG", "GCTA"]

- text = "AAACABBAAAABBAAABAAABAABB"
  patterns = ["AB", "AA", "AAB"]
- text = "ababcbababacbab"
  patterns = ["ab", "abab", "bab", "aca"]

  On aura ce tableau :

Taille de text	14	15	16	25	44
Taille des motifs	2|2|5	5|7	4|4|5	2|2|3	3|3|5
Temps exécution	0.529	0.518	1.187	0.704	4.555

2.4.1 Déroulement de Commentz Walter

voici un déroulement simple de l'algorithme Commentz-Walter sur le texte "hi hello world" et les motifs "hi", "world" et "hello".

1- Créer l'arbre des suffixes pour le texte T = "hi hello world" et l'initialiser avec les motifs S1 = "hi", S2 = "world" et S3 = "hello". L'arbre ressemble à ceci :

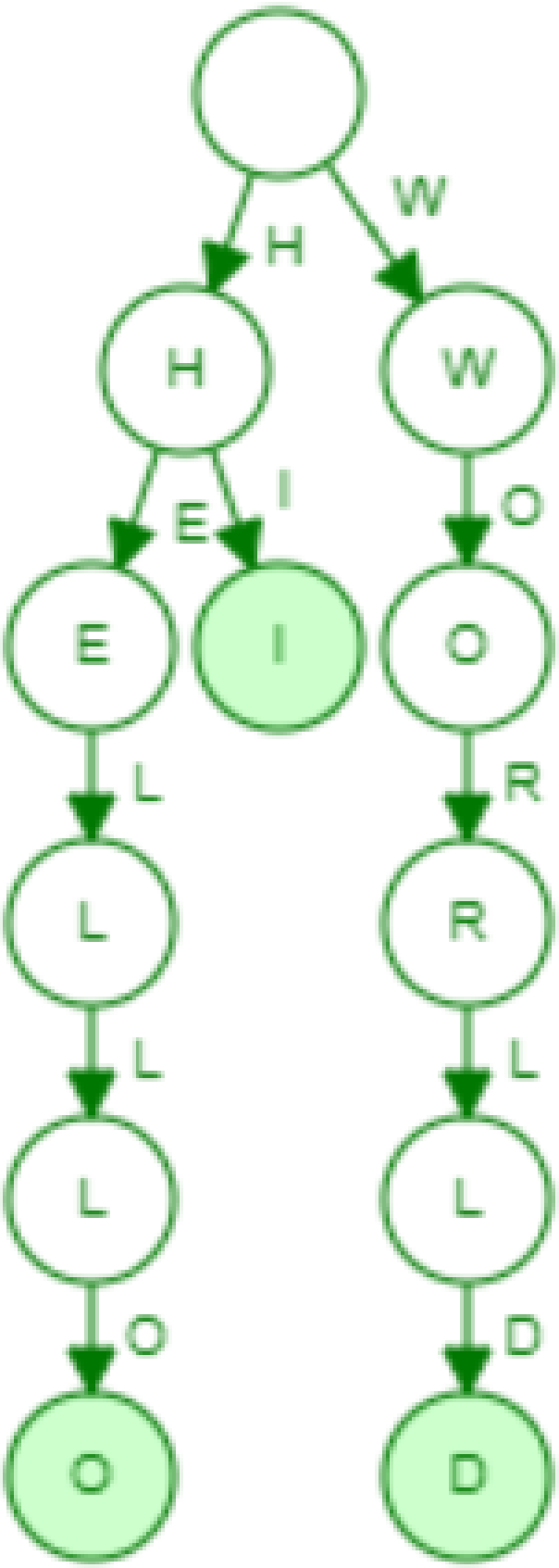

Figure 2.1: Trie

2- Pour chaque nœud de l'arbre, déterminer les ensembles Set1 et Set2 en utilisant les fonctions find_set1() et find_set2(). Par exemple, pour le nœud "h", Set1 contient le nœud "i" (car "hi" est un suffixe de "hi hello world") et Set2 est vide.

3- Calculer les valeurs de shift1 et shift2 pour chaque nœud de l'arbre, en utilisant les ensembles Set1 et Set2. Les valeurs de shift1 et shift2 indiquent de combien de caractères le texte peut être décalé à partir de ce nœud sans risquer de manquer un motif. Par exemple, pour le nœud "h", shift1 = 1 (car le suffixe "i" a une longueur de 1 caractère) et shift2 = 9 (car il faut décaler le texte de 9 caractères pour passer de "hi" à "hello world").

4- Utiliser les valeurs de shift1 et shift2 pour parcourir le texte T à la recherche des motifs. Pour chaque itération, commencer à la racine de l'arbre et descendre dans l'arbre en suivant les caractères du texte T. Si un nœud est atteint et qu'il a un match avec l'un des motifs, signaler une occurrence. Sinon, décaler le texte T de shift1 caractères si possible, ou de shift2 caractères sinon, et recommencer la recherche à partir de la nouvelle position.

Par exemple, pour le premier motif "hi", on commence à la racine de l'arbre et on suit le caractère "h" de T jusqu'au nœud "h". Comme ce nœud a un match avec le motif "hi", on signale une occurrence. Pour le deuxième motif "world", on commence à la racine de l'arbre et on suit le caractère "w" de T jusqu'au nœud "w". Ensuite, on décale T de 9 caractères (valeur de shift2 pour le nœud "w"), ce qui nous amène à la position juste après le mot "hello" dans T. On recommence la recherche à partir de cette nouvelle position, en suivant le caractère "o" de T jusqu'au nœud "o", puis le caractère "r" jusqu'au nœud "r", etc. Finalement, on trouve un match avec le motif "world" au nœud "d".

Chapter 3

Les courbes

3.1 Boyer Moore

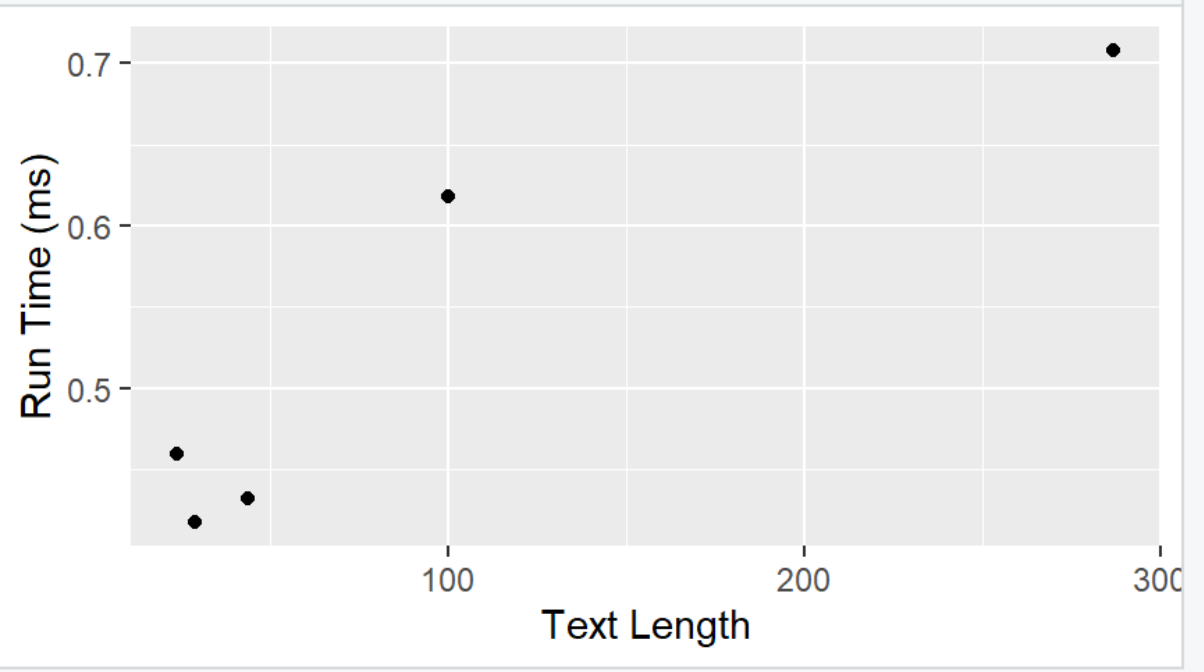

Figure 3.1: Courbe Boyer Moore

3.2 Rabin Karp

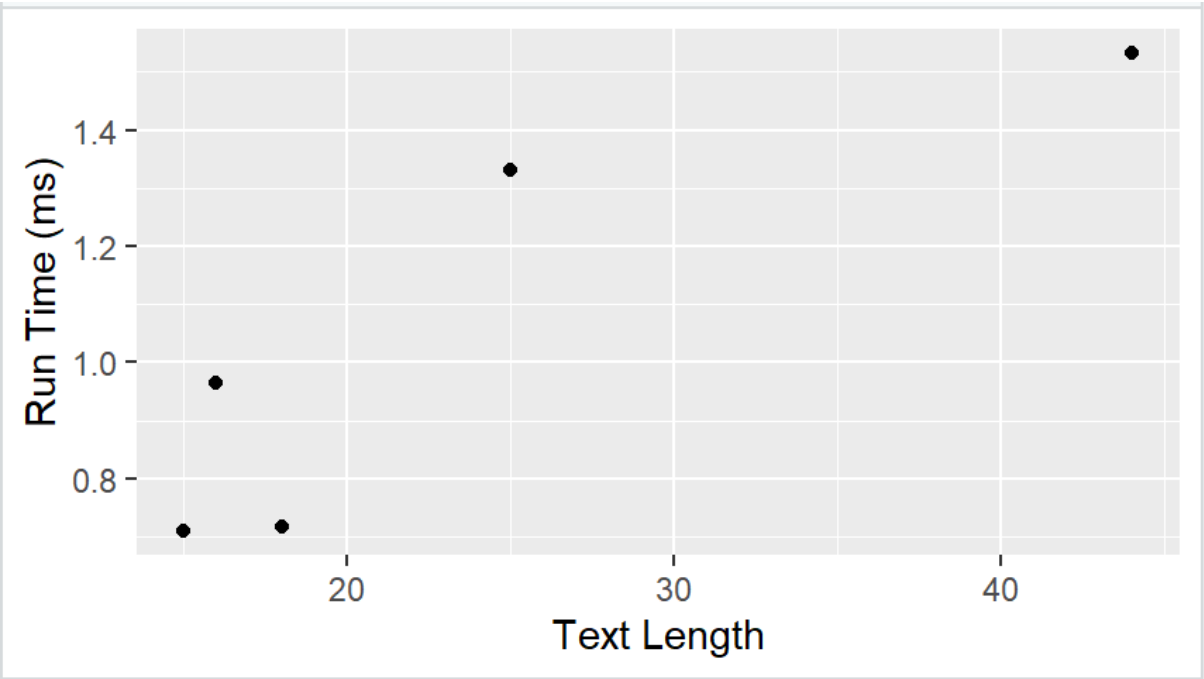

Figure 3.2: Courbe Rabin Karp

3.3 Aho Corasick

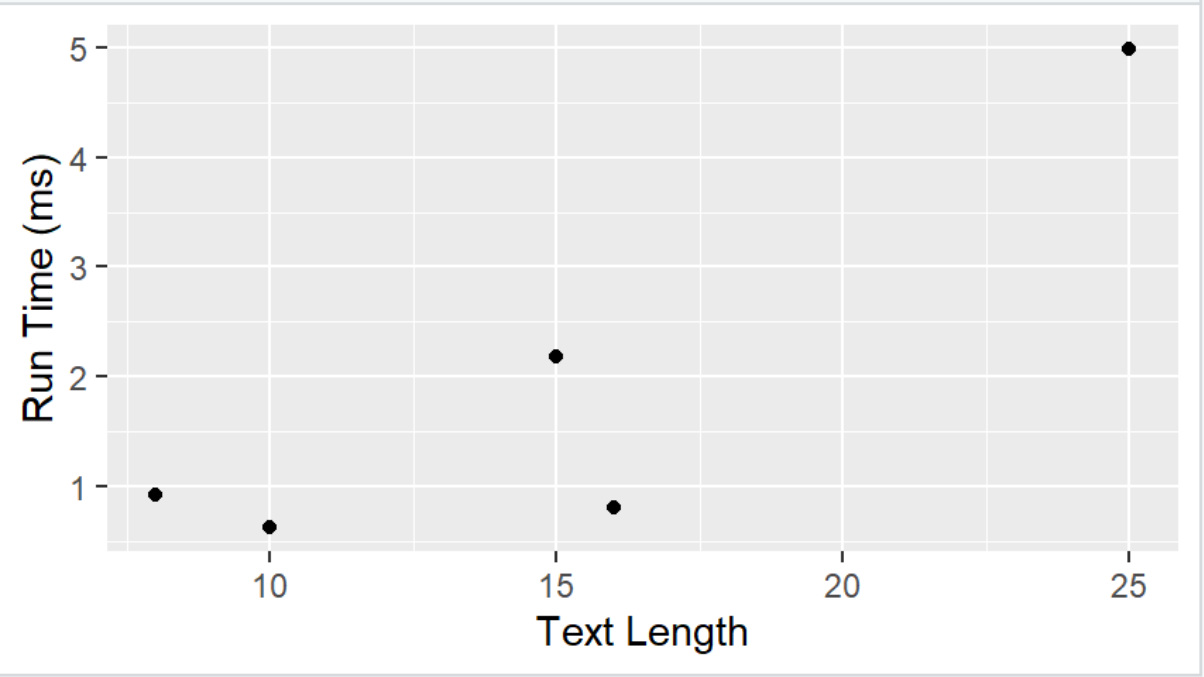

Figure 3.3: Courbe Aho Corasick

3.4 Commentz Walter

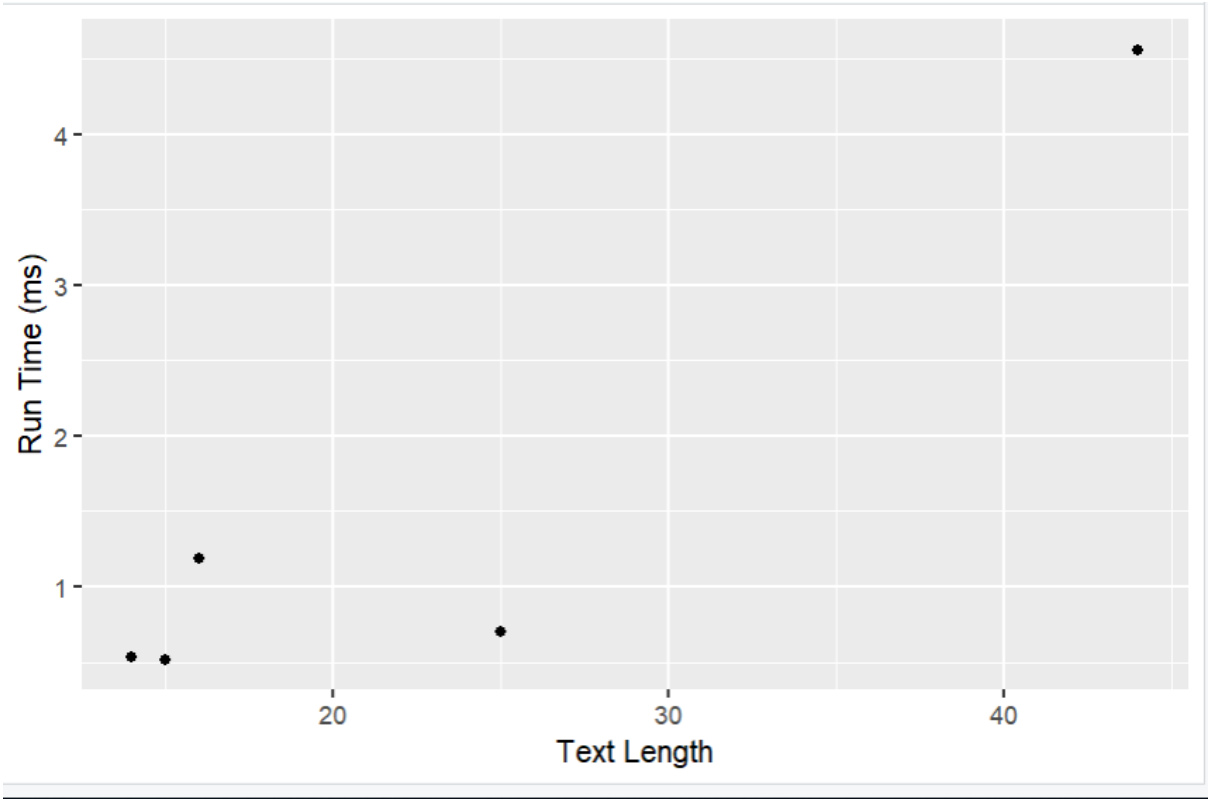

Figure 3.4: Courbe Commentz Walter

Chapter 4

Conclusion

4.1 Conclusion

Les résultats de test des algorithmes précédents sont presque en accord avec la complexité théorique mais pas exactement, car la complexité théorique peut être trop optimiste car elle ne prend pas en compte les facteurs tels que la taille de la mémoire, la latence d'accès à la mémoire et les limitations matérielles. En revanche, la complexité expérimentale peut être trop pessimiste car elle peut inclure des facteurs tels que des opérations supplémentaires qui n'ont pas été considérées dans la complexité théorique.

4.2 Etude comparative entre les algorithmes de texte

Algorithme	complexité	type de recherche	plusieurs motifs
Boyer Moore	$O(n * m)$	suffixe	Non
Rabin Karp	$O(n * m)$	préfixe	Oui
Aho Corasick	$O(n + m + z)$	préfixe	Oui
Commentz Walter	$O(n + m + wmax)$	suffixe	Oui

Algorithme	approche	idée clé
Boyer Moore	approches heuristiques	heuristique des mauvais caractères et des bons suffixes pour déterminer la distance de décalage
Rabin Karp	approche de hachage	comparer le texte et le motif à partir de leurs fonctions de hachage
Aho Corasick	approche d'automate	l'utilisation d'une Trie
Commentz Walter	approche d'automate	l'utilisation d'une Trie