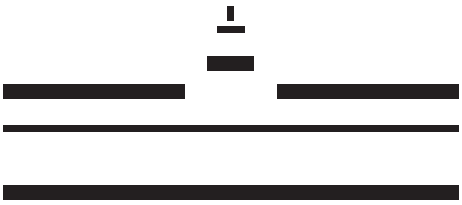


Westfälische
Wilhelms-Universität
Münster

An efficient error correction algorithm using FM-index

Seminararbeit

vorgelegt von:
Eike Gebauer

Matrikelnummer: 408126

Studiengang: MSc. Informatik

Zusammenfassung

An dieser Stelle soll eine Zusammenfassung der wissenschaftlichen Arbeit stehen. Idealerweise ist die Zusammenfassung genau eine Seite lang, was in etwa 350 - 450 Wörtern entspricht. Sie bildet zusammen mit dem Fazit einen Rahmen der gesamten wissenschaftlichen Arbeit und sollte auch für Außenstehende einfach zu verstehen und ansprechend formuliert werden.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 DNA-Sequenzierung 1
1.2 Fehlerkorrektur 3

2 Methodik **5**
2.1 Datenstrukturen 5
2.1.1 Patternsuche 6

3 Implementierung **11**
3.1 Quelltext-Abschnitte 11

4 Ergebnisse **13**

5 Diskussion **15**

6 Fazit **17**

1 Einleitung

DNA Sequenzierung ist aus der biologischen und medizinischen Forschung nicht mehr wegzudenken. Die Entwicklung immer günstigerer und leistungsstärkerer Sequenzierungsverfahren hat in den letzten Jahrzenten für einen rapiden Anstieg der zu verarbeitenden Datenmengen gesorgt. Hinzu kommt, dass die verwendeten Verfahren nicht fehlerfrei sind, was die Verarbeitung der Daten weiter erschwert.

In dieser Arbeit wird ein Paper behandelt, in dem der Fehlerkorrekturalgorithmus FMOE vorgestellt wird. Zu erst wird aber der Hintergrund in den Bereichen Biologie und Bioinformatik erläutert.

1.1 DNA-Sequenzierung

Das Ziel der de-novo DNA-Sequenzierung ist die Bestimmung der Nukleotidfolge in DNA ohne ein bereits bekanntes Referenzgenom. Wo bei alten Verfahren noch alle Basen nacheinander bestimmt wurden, werden seit dem sogenannten Shotgun Sequencing viele Teilstücke der DNA parallel analysiert. Das macht es möglich, lange DNA Stränge mit Verfahren zu sequenzieren, die nur kurze Fragmente verarbeiten können. Gleichzeitig ermöglicht es die parallele Sequenzierung von mehreren Fragmenten, was den Vorgang extrem beschleunigt.

Die Reihenfolge der DNA-Fragmente aufrecht zu erhalten, ist möglich, aber relativ langsam. Die sogenannten next-gen Sequenzierungsverfahren, welche den aktuell höchsten Durchsatz erzielen, verzichten daher darauf. Um trotzdem auf die ursprüngliche Reihenfolge und damit auf die gesamte DNA-Sequenz zurückschließen zu können, wird das Verfahren für mehrere Kopien desselben DNA-Strangs durchgeführt.

Wie in Abbildung 1.1 zu sehen, trennt man also mehrere Kopien des Strangs an zufälligen Positionen in ungefähr gleich lange Fragmente. Alle Fragmente können dann unabhängig von einander - und damit parallel - sequenziert werden. Das Ergebnis der Sequenzierung eines Fragments wird Read genannt. Aufgrund der zufälligen Trennungspositionen überlappen sich viele dieser Reads. Vorausgesetzt die Abdeckung[1] reicht an allen Stellen der Sequenz aus, kann man mit den Überlappungen die Reads in die ursprüngliche Reihenfolge bringen und so die gesuchte DNA-Sequenz bestimmen. Dieser Prozess heißt Assembly.

[1] Abdeckung (englisch Coverage) beschreibt die Anzahl einzigartiger Reads, die dieselbe Base enthalten.

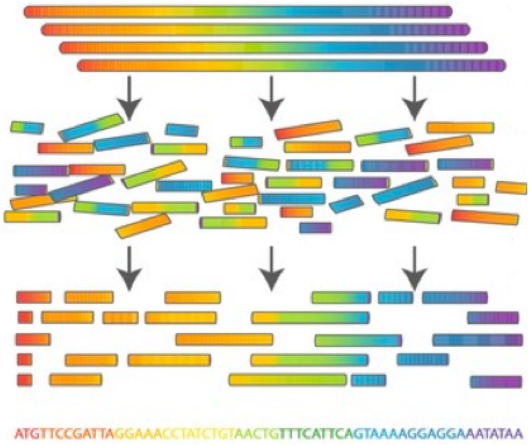


Abbildung 1.1: Shotgun Sequenzierung: Kopien des DNA-Strangs (oben) werden an zufälligen Positionen fragmentiert und sequenziert(darunter), in die korrekte Reihenfolge gebracht (3. von oben) und zur Gesamtsequenz zusammengesetzt (unten)

Wie bereits erwähnt, sind die aktuellen Verfahren zur Bestimmung der Reads nicht fehlerfrei. Es kann zum Überspringen (Deletion) oder Vertauschen (Substitution) richtiger Basen oder zur Einfügung (Insertion) falscher Basen kommen. Kollektiv werden diese Fehler Indels genannt.

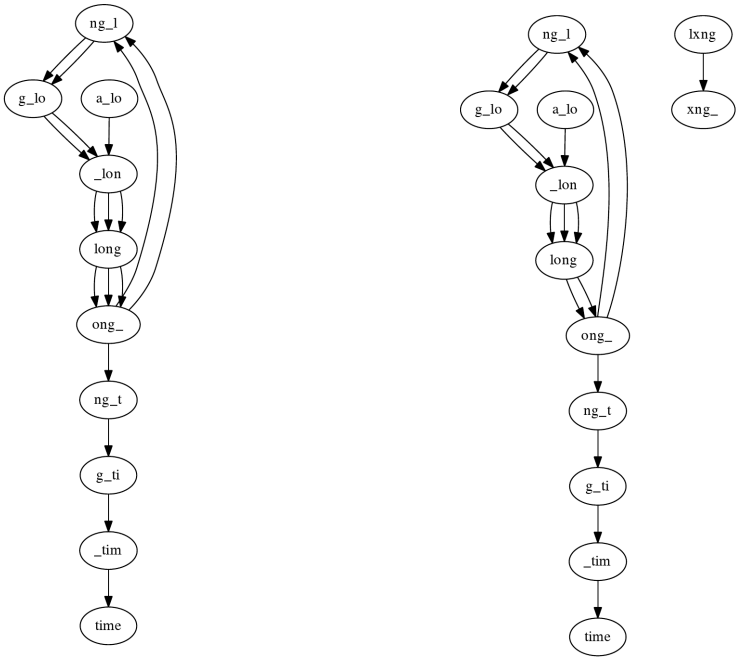
Die Assembly üblicherweise durch Überlappungsgraphen oder de Bruijn Graphen bewältigt. Überlappungsgraphen können Fehler bis zu einem gewissen Maß kompensieren, bringen allerdings hohe Speicher- und Rechenanforderungen mit sich. De Bruijn Graphen sind deutlich resourcensparender, tolerieren aber keine Fehler. Aus diesem Grund wird bei großen Datenmengen oft auf de Bruijn Graphen mit vorheriger Fehlerkorrektur zurückgegriffen.

Fehler in den Reads sorgen für verschiedene Probleme bei der Assembly mithilfe eines de Bruijn Graphen:

Fragmentierte Assembly
: Fehlerhafte Reads verhindern das Erkennen von Überlappungen und sorgen für Graphen, die keine valide Assembly zulassen. Ein Beispiel für eine fragmentierte Assembly ist in Abbildung 1.2(b) zu sehen.

Misassembly
: Fehlerhafte Reads überlappen sich in manchen Fällen mit korrekten Reads und sorgen für eine valide Assembly, die aber eine falsche Sequenz ergibt.
Dieser Fall ist im Nachhinein schwer zu erkennen.

Zusätzlich erhöht sich die Komplexität des Graphen bei fehlerhaften Reads unter Umständen enorm.



(a) Intakter de Bruijn Graph (b) Fragmentierter de Bruijn Graph

Abbildung 1.2: De Bruijn Graph des Texts „a long long time“. Oben der Graph aus fehlerfreien Fragmenten, unten der fragmentierte Graph, hervorgerufen durch Substitutionsfehler.

1.2 Fehlerkorrektur

Aufgabe der Fehlerkorrektur ist es, diese Komplikationen zu verhindern.
Da bei der de-novo Sequenzierung kein Referenzgenom zur Verfügung steht, kann die Fehlerkorrektur nur auf Grundlage der Reads geschehen. Die üblichen Verfahren stützen sich dabei auf statistische Modelle, um fehlerhafte Reads zu finden und zu verwerfen oder zu korrigieren.

Das hier behandelte Paper teilt Fehlerkorrekturalgorithmen in drei Kategorien:

k-mer Frequenzsprektrum Methoden erzeugen eine Erhebung aller k-mere[2] und ihrer Häufigkeiten aus den Reads und ersetzen seltene k-mere durch Häufige.
Diese Verfahren sind empfindlich gegenüber Wiederholungen in der DNA-Sequenz.

Suffix-Tree / -Array Methoden erzeugen einen Index der Suffixe der Reads in Form eines Arrays oder Baums und verhalten sich ansonsten ähnlich

[2]k-mer wird eine k Zeichen lange Teilkette einer Zeichenkette genannt. Beispiel: Die 3-mere von „ABCDE“sind „ABC“„BCD“und „CDE“.

zu den k-mer-basierten Methoden.
Die variable Länge der Suffixe macht diese Verfahren etwas resistenter gegen Wiederholungen.

Überlappungsbasierte Methoden suchen für jeden zu korrigierenden Read alle überlappenden Reads und korrigieren potenzielle Fehler mit einem MSA[3] der gefundenen Reads.
Durch die Verwendung der ganzen Reads sind diese Verfahren nicht anfällig für Wiederholungen, allerdings ist die berechnung des MSA Berechnung sehr langsam.

Weitere Vor- und Nachteile der verschiedenen Vorgehensweisen werden später noch erläutert.

[3]MSA ist die Abkürzung für Multiple Sequence Alignment und beschreibt das alignen von mehr als drei DNA-Sequenzen. Mit alignment ist an dieser Stelle der methodische Vergleich von Sequenzen gemeint.

2 Methodik

FMOE fällt in die dritte Kategorie und unterscheidet sich von anderen Verfahren derselben Kategorie hauptsächlich durch die Verwendung von FM-Index und damit verbundenen Optimierungen.

Im Folgenden werden daher zuerst die Datenstrukturen behandelt.

2.1 Datenstrukturen

Der FM-index ist eine Datenstruktur für Zeichenfolgen, die auf der Burrows Wheeler Transformation (BWT) basiert.

Die BWT ist ein Algorithmus, der aus einer beliebigen Zeichenfolge eine Permutation dieser Folge und einen Index erzeugt, mit dem Effekt, dass die erhaltene Permutation in der Regel leichter zu komprimieren ist. Eine Rücktransformation ermöglicht dann die Wiederherstellung der ursprünglichen Zeichenfolge.

Der FM-Index wird ähnlich zur BWT konstruiert, speichert aber zusätzliche Daten, um schnelle Patternsuche bei immer noch sehr geringer Speicherkomplexität zu ermöglichen. Die Funktionsweise und die Gründe für die Resourcensparsamkeit lassen sich am besten im Vergleich zum Suffix-Array erläutern.

Wie in den Tabellen 2.1 und 2.2 zu sehen ist, beginnt man bei beiden Datenstrukturen damit, alle Suffixe der Zeichenfolge zu erzeugen. Im Fall des FM-Index wird jede Zeile mit dem Anfang der Zeichenfolge aufgefüllt, sodass sich die Rotationen der Zeichenfolge ergeben. Zusätzlich werden die Suffixpositionen in der Zeichenfolge in einer extra Spalte festgehalten.

Das Ergebnis dieser Operation wird dann lexikographisch sortiert. In Tabelle 2.4 ist der fertige Suffix Array zu sehen. Während bei dem Suffix-Array das gesamte Array gespeichert werden muss, kann beim FM-Index auf einen großen Teil der Daten verzichtet werden.

- Aus den Rotationen müssen nur die Spalten F und L gespeichert werden.
- Die Spalte F ist aufsteigend sortiert und kann daher bei einem kleinen Alphabet wie $\{A, C, G, T, \$\}$ sehr stark komprimiert werden.
- Für den Suchalgorithmus müssen die Ränge der Zeichen in der letzten Zeile (in Tabelle 2.5 als tiefgestellte Indizes sichtbar) bekannt sein. Diese kann man bei jeder Suche neu berechnen oder speichern. In den meisten Fällen wird ein hybrider Ansatz verwendet, bei dem ein Bruchteil

0	a	t	g	a	t	t	a	t	c	$
1	t	g	a	t	t	a	t	c	$	
2	g	a	t	t	a	t	c	$		
3	a	t	t	a	t	c	$			
4	t	t	a	t	c	$				
5	t	a	t	c	$					
6	a	t	c	$						
7	t	c	$							
8	c	$								
9	$									

Abbildung 2.1: Suffixe

0	a	t	g	a	t	t	a	t	c	$
1	t	g	a	t	t	a	t	c	$	a
2	g	a	t	t	a	t	c	$	a	t
3	a	t	t	a	t	c	$	a	t	g
4	t	t	a	t	c	$	a	t	g	a
5	t	a	t	c	$	a	t	g	a	t
6	a	t	c	$	a	t	g	a	t	t
7	t	c	$	a	t	g	a	t	t	a
8	c	$	a	t	g	a	t	t	a	t
9	$	a	t	g	a	t	t	a	t	c

Abbildung 2.2: Rotationen

Abbildung 2.3: Vorstufen von Suffix-Array und FM-Index für die Zeichenfolge „atgattatc$“

der Ränge vorberechnet und gespeichert wird, um die Berechnung der Übrigen zu beschleunigen.

- Die Positionen in der usprünglichen Zeichenfolge (in den Tabellen 2.6 als getrennte Spalte sichtbar) können genau wie die Ränge teilweise vorberechnet werden, um einen guten Kompromiss zwischen Speicherverbrauch und Rechenaufwand zu erhalten.

Diese Optimierungen sorgen dafür, dass ein Suffix-Array in der Regel ein vielfaches an Speicher benötigt als ein FM-Index.

2.1.1 Patternsuche

Die Patternsuche im Suffix-Array ist trivial, da es für jedes Vorkommen des Patterns einen Suffix geben muss, der mit dem Pattern beginnt. Weil alle

9	$
6	a t c $
0	a t g a t t a t c $
3	a t t a t c $
8	c $
2	g a t t a t c $
5	t a t c $
7	t c $
1	t g a t t a t c $
4	t t a t c $

Abbildung 2.4: Suffix-Array

	F								L
9	$	a	t	g	a	t	t	a	c_1
6	a_1	t	c	$	a	t	g	a	t_1
0	a_2	t	g	a	t	t	a	t	c
3	a_3	t	t	a	t	c	$	a	g_1
8	c_1	$	a	t	g	a	t	t	t_2
2	g_1	a	t	t	a	t	c	$	t_3
5	t_1	a	t	c	$	a	t	g	t_4
7	t_2	c	$	a	t	g	a	t	a_1
1	t_3	g	a	t	t	a	t	c	a_2
4	t_4	t	a	t	c	$	a	t	g	a_3

Abbildung 2.5: FM-Index

Abbildung 2.6: Resultierender Suffix-Array und FM-Index für die Zeichenfolge „atgattatc$“

Suffixe lexikographisch sortiert sind, liegen alle Suffixe, die mit dem Pattern beginnen hintereinander im Array. Es genügt also den Ersten und den Letzten der Suffixe durch wiederholte binäre Suche zu finden und mithilfe der ersten Spalte die Position aller dazwischen liegenden herauszufinden.
Beispiel: Suche nach „att“(siehe auch Tabelle 2.7)

1. Durch binäre Suche das Intervall der Suffixe finden, dass mit „a“beginnt (blau). Wenn die erste Spalte des Arrays durch RLE[1] komprimiert wurde, kann dieser Schritt ohne binäre Suche in konstanter Zeit stattfinden.

2. Mit binärer Suche das Teilintervall finden, dessen zweites Zeichen „t“ist (orange).

[1]RLE steht für Run Length Encoding und ist eine Kompression, die bei Daten mit vielen Symbolwiederholungen eingesetzt wird. Beispiel: „aaaaabbb“wird komprimiert zu „a5b3“

3. Schritt 2 für das nächste Zeichen wiederholen (rot).

4. Den gefundenen Suffix anhand der extra Spalte seiner Position in der Zeichenfolge zuordnen (Index 3)

9	$									
6	a	t	c	$						
0	a	t	g	a	t	t	a	t	c	$
3	a	t	t	a	t	c	$			
8	c	$								
2	g	a	t	t	a	t	c	$		
5	t	a	t	c	$					
7	t	c	$							
1	t	g	a	t	t	a	t	c	$	
4	t	t	a	t	c	$				

Abbildung 2.7: Patternsuche in Suffix-Array

Die Patternsuche in einem FM-Index ist etwas komplizierter. Man sucht das Pattern von hinten nach vorne durch ein sogennantes LF-Mapping. Die Eigenschaft, die dieses Mapping ermöglicht, ist, dass der Zeichenrang in der letzten Zeile, dem der ersten Zeile entspricht. Das dritte „a“in der Spalte L ist in der ursprünglichen Folge also dasselbe Zeichen, wie das dritte „a“in der Spalte F.

Auch im FM-Index liegen alle Vorkommen des Patterns in einem zusammenhängenden Intervall.
Das selbe Beispiel: Suche nach „att“(siehe auch Tabelle 2.8)

1. In konstanter Zeit das Intervall von F suchen, dass mit „t“beginnt (blau).

2. In dem selben Intervall in L linear nach dem nächsten Zeichen „t“suchen (orange).

3. Das gefundene „t“s anhand des Rangs in F wieder finden.

4. Schritt 2 und 3 mit dem Zeichen „a“wiederholen (rot).

5. Den gefundenen Eintrag anhand der extra Spalte seiner Position in der Zeichenfolge zuordnen (Index 3)

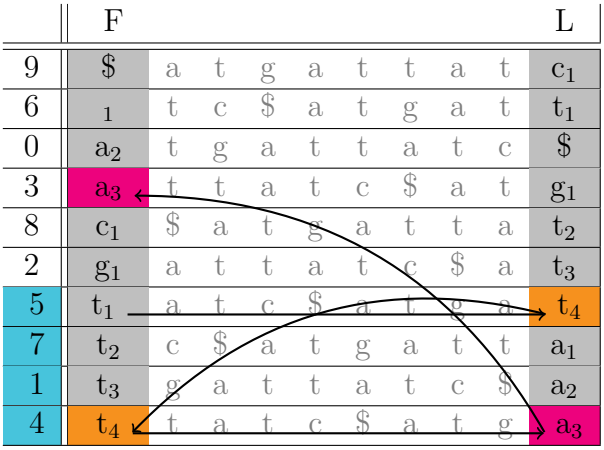


Abbildung 2.8: Patternsuche in FM-Index

3 Implementierung

In diesem Kapitel werden Technische Details, wie zum Beispiel die Verwendete Laufzeitumgebung, Laufzeitanalyse oder wichtige Implementierungsdetails behandelt. Dabei sollte beachtet werden inwiefern diese Angaben für die Arbeit von Bedeutung sind. Die Namen einzelner Funktionsaufrufe oder Klassendiagramme sind zum Beispiel im Allgemeinen für den Leser uninteressant.

3.1 Quelltext-Abschnitte

Eigene Auszüge aus dem Quelltext bindet man am Einfachsten mit dem Befehl `\lstinputlisting{}` ein. Das Ergebnis ist in 3.1 zu sehen. Optionen für die Sprache, Tab-Breite etc. von `\lstinputlisting` können auch am Anfang des Dokuments mit `\lstset{}` gesetzt werden. Das Paket erlaubt mit `firstline=...` und `lastline=...` auch die Einbindung von einzelnen Zeilen einer Datei.

Quelltext 3.1: Codebeispiel

```
int myTest(n) {
  a = 0;
  for (int i = 0; i <= n; i++)
      a = a + 1;
  return a;
}
```

4 Ergebnisse

Dieses Kapitel sollte die Ergebnisse beinhalten, die mit den Methoden aus Kapitel 2 erstellt wurden.

Tabelle 4.1: Beispieltabelle

Spalte1	Spalte2	Spalte3
1	2	3

5 Diskussion

In diesem Kapitel werden die zuvor vorgestellten Ergebnisse der Arbeit diskutiert. Häufig wird es mit dem Fazit zusammengelegt.

6 Fazit

Dieses Kapitel bildet die abschließende Zusammenfassung der Arbeit. Dazu können die folgende Punkte behandelt werden:

- Reflexion: wurden die Ziele der Arbeit erreicht?
- mögliche Erweiterungen und Verbesserungen („future work“)

Abbildungsverzeichnis

1.1 Shotgun Sequenzierung: Kopien des DNA-Strangs (oben) werden an zufälligen Positionen fragmentiert und sequenziert(darunter), in die korrekte Reihenfolge gebracht (3. von oben) und zur Gesamtsequenz zusammengesetzt (unten) 2
1.2 De Bruijn Graph des Texts „a long long time“. Oben der Graph aus fehlerfreien Fragmenten, unten der fragmentierte Graph, hervorgerufen durch Substitutionsfehler. 3

2.1 Suffixe 6
2.2 Rotationen 6
2.3 Vorstufen von Suffix-Array und FM-Index für die Zeichenfolge „atgattatc$“ 6
2.4 Suffix-Array 7
2.5 FM-Index 7
2.6 Resultierender Suffix-Array und FM-Index für die Zeichenfolge „atgattatc$“ 7
2.7 Patternsuche in Suffix-Array 8
2.8 Patternsuche in FM-Index 9

Tabellenverzeichnis

4.1 Beispieltabelle 13

Literaturverzeichnis

[1] Jele, Harald: *Wissenschaftliches Arbeiten: Zitieren.* R. Oldenbourg Verlag, 2010

Eidesstattliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit über *„Titel“* selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

Vorname Nachname, Münster, 14. Januar 2019

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

Vorname Nachname, Münster, 14. Januar 2019