Untersuchung von Komprimierungsverfahren für Nukleotidsequenzen

Maturaarbeit

Hristo Georgiev

30. Mai, 2024

Betreuer: Igor Režan

Zusammenfassung

Die Verbreitung von Next Generation Sequencing (NGS), welches die Parallelisierung von Genomkartierungen ermöglicht, führte zu einem enormen Wachstum genombezogener Daten. Dies hat einen Anreiz geboten zur Untersuchung von Methoden, mit denen einerseits die Kartierung von Genomen beschleunigt werden, und andererseits genetische Datenmaterialien komprimiert und so der Speicherplatz effizienter benutzt werden kann. Die biologische Struktur der Desoxyribonukleinsäure, die aus Wiederholungen der Nukleinbasen Adenin, Thymin, Guanin und Cytosin besteht, eignet sich wie wir sehen werden sehr gut für die Datenkompression. Dies wegen der Tatsache, dass es für jede Stelle in der Sequenz nur 4 (Anstatt 26) Möglichkeiten gibt, aber auch wegen der Redundanz in genetischem Code, der eine Wiederholung von Abschnitten aufweist.

In dieser Arbeit werden verschiedene Arten von Komprimierungsverfahren, unter anderem Entropiecodierer und Wörterbuchcodierer, hinsichtlich ihrer Effizienz und Geschwindigkeit bei der Kompression von Nukleotidsequenzen verglichen. Um die Funktion der Komprimierungsverfahren zu verstehen, werden einige Grundlagen der Infomationstheorie dargestellt, unter anderem die Bestandteile eines Komprimierungsverfahrens und ihre Funktion und Bedeutung beim Komprimieirungsprozess.

Ein zentrales Ergebnis dieser Arbeit ist die Bedeutung der Wahl des Komprimierungsverfahrens in Abhängigkeit von der Länge und Struktur der Nukleotidsequenzen. Verfahren, die kontextbasierte Entropiecodierer verwenden, wie PPM und Arithmetische Codierung, weisen bei längeren Sequenzen signifikante Vorteile auf. Die Ergebnisse zeigen, dass PPM das beste Kompressionsverhältnis sowohl bei kurzen als auch längeren Nukleotidsequenzen zeigt, während die Arithmetische Codierung durch ihre schnelle Laufzeit heraussticht. Wörterbuchcodierer wie Deflate sind zwar effizient bei kürzeren Sequenzen, nicht aber bei längeren Sequenzen. Die Untersuchung beleuchtet auch die Vielfalt und Komplexität der Komprimierung von Dateien mit genetischen Daten, da sie nicht nur Nukleotidsequenzen enthalten, sondern auch Metadaten verschiedener Art, die ebenfalls komprimiert werden müssen. Zusätzlich werden spezialisierte Komprimierungsverfahren wie Genozip analysiert, die solche Metadaten nutzen, um evolutionäre Verwandtschaften und biologische Funktionen zu analysieren, um weiteren Datenreduktion zu erreichen. Es wird auch das SPRING Komprimierungsverfahren untersucht, das auf Komprimierung von reinen Nukleotidsequenzen spezialisiert ist.

Die Arbeit schliesst mit einer Diskussion über die zukünftige Entwicklung und Optimierung von Komprimierungsverfahren, und die Notwendigkeit von interdisziplinärem Wissen bei der Konzeption und Entwicklung von Komprimierungsverfahren, die genetische Daten komprimieren sollen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Als ich die Grundlagen der Genetik gelernt habe, war ich fasziniert von der Eleganz, mit der Proteine synthetisiert werden. Aus lediglich vier Bauelementen, den Nukleinbasen der DNA, werden Proteine aller möglichen Funktionen und Rollen hergestellt: Strukturproteine, Enzymproteine, Rezeptorproteine, Antikörper, Viren und so weiter. Dabei ist mir aufgefallen, dass die Translation von DNA zu mRNA und die darauf folgende Transkription der Basentriplette zu Aminosäuren, aus denen die Proteinstrukturen entstehen, eine Art von biologischer Dekompression ist. Bei den Lebewesen werden die Nukleotiden eng in Chromosomen gepackt, sie stellen die komprimierte Information dar. Das Endprodukt ihrer Dekomprimierung sind die Proteine, die das Lebewesen aufbauen.

Darin sah ich einen Zusammenhang mit den Grundlagen der Diskrete Mathematik, genauer mit dem Teilbereich davon, der sich mit der Datenkomprimierung beschäftigt. Ähnlich wie bei der genetischen Information, wo es lediglich vier Nukleotide gibt, werden beim Computer Informationen durch sogar nur zwei Zeichen dargestellt, 0 und 1. Mit diesem Binärsystem kann ebenfalls eine Vielfalt von Daten repräsentiert werden. Statische Dateien wie Text und Bilder, aber auch Programm-Dateien, die kompiliert und ausgeführt werden können. Aus technischen Gründen besteht heutzutage Bedarf, die Dateien in einem möglichst effizienten Format vorliegen zu haben, um Speicherplatz zu sparen und ihre Übertragung zu beschleunigen. Dazu wurden verschiedene Komprimierungsverfahren entwickelt, die jeweils ihre Vor- und Nachteile haben und auf verschiedenen Datentypen mit unterschiedlicher Effizienz wirken.

Und dann kam ich zu meiner Idee: Wie wirken sich die verschiedenen Komprimierungsverfahren auf Dateien aus, die genetische Information darstellen? In wieweit kann man Nukleotidsequenzen noch effizienter komprimieren?

Als ich mit meiner Analyse begann, erkannte ich, dass es sich um ein aktuelles Problem handelt. Durch die Fortschritte der Sequenzierungsmethoden in der Gentechnik, vor allem die Verbreitung des Next-Generation Sequencing (NGS), konnte in kurzer Zeit ein grosses Volumen von Genomen kartiert werden und folglich haben sich vielen Daten angesammelt, die verarbeitet und analysiert werden sollen [1]. Die Menge der in der Forschung betrachteten genetischen Daten wächst exponentiell und schneller als der Preis für Datenspeicher [2].

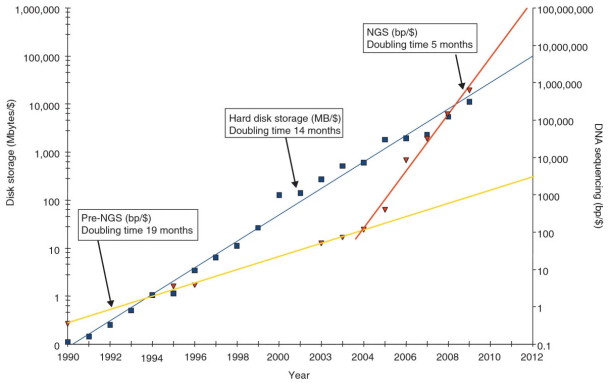

Abbildung 1.1: Die blauen Quadrate beschreiben die historischen Kosten der Festplattenpreise in Megabyte pro US-Dollar. Der langfristige Trend (blaue Linie, die hier eine gerade Linie ist, da die Darstellung logarithmisch ist) zeigt das exponentielle Wachstum der Speicherkapazität pro Dollar mit einer Verdoppelungszeit von etwa 1,5 Jahren. Die Kosten für die DNA-Sequenzierung, ausgedrückt in Basenpaaren pro Dollar, wird durch die roten Dreiecke dargestellt. Diese entspricht einer exponentielle Kurve (gelbe Linie) mit einer Verdoppelungszeit, die etwas langsamer ist als diejenige für Festplattenspeicherpreise. Deutlich zu erkennen ist der Sprung im Jahre 2004, als Next-Generation-Sequencing (NGS) verfügbar wurde. Dadurch fiel die Verdoppelungszeit auf weniger als 6 Monate (rote Linie). Diese Kurven sind nicht inflationsbereinigt oder korrigiert („voll ausgelastete“ Kosten für Sequenzierung und Plattenspeicherung, einschliesslich Personalkosten, Abschreibungen und Gemeinkosten) [2]

Die fortlaufende Sammlung und Klassifizierung genetischer Daten ist von wachsender Bedeutung sowohl in der Forschung als auch in der Industrie. In der Forschung werden die evolutionären Zusammenhänge zwischen Organismen untersucht, in der Industrie werden personalisierte Medikamente und Behandlungen entwickelt. Die COVID-19 mRNA Impfung zum Beispiel war nur dank dieses technologischen und wissenschaftlichen Fortschritts möglich [3].

Der Bereich der Komprimierung genetischer Daten ist als Gesamtes sehr umfangreich und komplex. Deshalb konzentriert sich diese Arbeit auf die Methodik, Auswirkungen und Leistung von Komprimierungsverfahren auf DNA-Sequenzen mit unterschiedlicher Länge und Herkunft.

Kapitel 2

Grundlagen der Komprimierung

Die Grundlagen der Komprimierung liegen in der Informationstheorie [4]. Der Begriff *Information* bedeutet Kenntnis über Sachverhalte oder Vorgänge und die Informationen werden mittels *Nachrichten* aus verschiedenen *Datentypen* zwischen einem Sender und einem Empfänger ausgetauscht.

Um diese Begriffe im Kontext dieser Arbeit zu veranschaulichen, werden zwei Abschnitte K und U aus der DNA des einsträngigen DNA-Virus Human Parvovirus B19 betrachtet [5]:

$$K = [C, A, A, G, G, T, G, T, T, T]$$
$$U = [T, C, A, A, G, G, C, A, A, C]$$

Es wird genetische Information untersucht, die Nachrichten werden also Nukleotidsequenzen sein. Eine Nukleotidsequenz besteht aus einer endlichen Folge von Zeichen aus einem Zeichenvorrat, der aus den Namen der Nukleotiden Adenin A, Cytosin C, Guanin G und Thymin T besteht.

Es gibt verlustfreie oder verlustbehaftete Komprimierungsverfahren. Die verlustfreie Kompression bei genetischen Daten ist unerlässlich, um ihre Integrität, Genauigkeit und Verlässlichkeit zu gewährleisten [6]. Sie stellt sicher, dass die Daten vollständig und unverändert bleiben, was für medizinische, wissenschaftliche und biotechnologische Anwendungen von höchster Bedeutung ist.

Komprimierungsverfahren können die sogenannte *Verzerrung* (Bias), die statistischen Eigenschaften und Besonderheiten der zu komprimierenden Daten ausnutzen [7]. Dazu müssen die Daten differenziert und analysiert werden, um Muster, Redundanzen und Wiederholungen zu identifizieren. Diese Analyse erfolgt durch stochastische Verfahren wie eine diskrete Wahrscheinlichkeitsverteilung, bei denen untersucht wird, welche Symbole oder

Symbolenfolgen in einer Nukleotidsequenz mit hoher Wahrscheinlichkeit als nächstes auftreten könnten. Diese Wahrscheinlichkeiten basieren auf der Häufigkeit und den Korrelationen der Zeichen innerhalb der Daten.

Strukturell besteht ein Komprimierungsverfahren aus zwei Bestandteilen: Dem *Modell* und dem *Coder*. Das Modell beschreibt die *Verzerrung*. Die Verzerrung entsteht durch bestimmte Muster oder Regelmässigkeiten in den Daten, die erkannt und ausgenutzt werden können. Der Coder nutzt diese Verzerrung, um die Daten effizient zu komprimieren. Dabei generiert er eine komprimierte Datei aus sogenannten Codewörtern. Ein *Codewort* ist eine eindeutige Folge von Bits oder Symbolen, die ein bestimmtes Muster oder Segment der Originaldaten beschreibt.

Durch die Verwendung von Wahrscheinlichkeitsmodellen kann das Komprimierungsverfahren effizientere Codierungen erstellen, indem es häufiger auftretenden Zeichen kürzere Codewörter zuweist und selteneren Zeichen längere Codewörter. Dies reduziert die durchschnittliche Länge der Darstellung der Daten und führt zu einer effektiven Komprimierung, wobei die codierten Daten weniger Speicherplatz benötigen.

2.1 Das Modell für die Komprimierung und der Begriff der Entropie

In der Datenkompression spielt das Modell eine zentrale Rolle, es beschreibt die statistischen Eigenschaften der zu komprimierenden Daten. Ein gutes Modell kann die Daten effizienter komprimieren, indem es die Wahrscheinlichkeitsverteilung der Symbole innerhalb der Daten berücksichtigt. Im Kontext der Komprimierung von Nukleotidsequenzen, hilft das Modell dabei, die Unterschiede und Muster in den Nukleotidsequenzen zu erkennen und zu nutzen.

Diese Unterschiede zwischen U und K werden durch die Wahrscheinlichkeitsverteilungen $P(X_K)$ beziehungsweise $P(X_U)$ ausgedrückt. Sie beschreiben die Wahrscheinlichkeit, mit der jede Nukleotid in den Sequenzen U und K auftritt.

Für K hat $P(X_K)$:

X_K	A	G	T	C
$P(X_K)$	0.2	0.3	0.4	0.1

Für U hat $P(X_U)$:

X_U	A	G	T	C
$P(X_U)$	0.4	0.2	0.1	0.2

Jede Nukleotidsequenzen hat ihre eigene Funktion, welche die verschiedenen Wahrscheinlichkeiten für jedes Nukleotid liefert. Daraus kann eine Wahrscheinlichkeitsverteilung erstellt werden, um die Unterschiede stochastisch zu betrachten. Eine solche Wahrscheinlichkeitsverteilung alleine gäbe jedoch nicht an, wie der Coder die Eingaben optimal codieren sollte, da sie lediglich die Wahrscheinlichkeiten der einzelnen Basen darstellt. Es wird der *Informationsgehalt* dieser Verteilung benötigt, um zu bestimmen, wie die Nukleotidsequenzen komprimiert werden können. Der Informationsgehalt hängt also direkt mit der Effizienz der Datenkomprimierung zusammen. Zur Beschreibung des Informationsgehalts wird das Konzept der *Entropie* aus der Physik benutzt [4]. Die Entropie steht für die Zufälligkeit und Unordnung eines Systems. Im Rahmen der Komprimierung genetischer Information beschreibt die Entropie das Mass des Informationsgehalts einer Nukleotidsequenz. Je homogener eine Nukleotidsequenz ist, desto niedriger ist ihre Entropie, weil es weniger Variationen der Nukleotiden innerhalb der Nukleotidsequenz gibt. Umgekehrt gilt: Je mehr Variabilität eine Nukleotidsequenz aufweist, desto grösser ist die Anzahl der möglichen Variationen der Nukleotiden und desto höher ist ihre Entropie.

Die Entropie $H(S)$ für eine diskret verteilte Zufallsvariable S einer Nukleotidsequenz ist wie folgt definiert:

$$H(S) = \sum_{s \in S} p(s) \log_2 \frac{1}{p(s)}$$

Der Informationsgehalt eines einzelnen Nukleotids oder eines Abschnitts der Nukleotidsequenz s wird wie folgt berechnet:

$$i(s) = \log_2 \frac{1}{p(s)}$$

Wobei $i(s)$ den Informationsgehalt eines einzelnen Nukleotids oder eines Abschnitts der Nukleotidsequenz s bezeichnet. Im Ganzen betrachtet, ist die Entropie das gewichtete Mittel des Informationsgehalts, abhängig davon, wie ein Kompromierungsverfahren die Information verwendet.

Der binäre Logarithmus log_2 und der Kehrwert der Wahrscheinlichkeit ergeben die anschliessende Codierung der Stelle $i(s)$ durch einen Binärcode mit N Stellen, wobei jede Stelle zwei Möglichkeiten (0 oder 1) hat. Somit gibt es 2^N mögliche Zustände für den Code dieser Stelle. Daraus kann der Coder abschätzen, wie lang das Codewort für die Stelle $i(s)$ im Durchschnitt sein könnte. Dadurch kann der Coder die Redundanz in den Daten reduzieren und eine effizientere Kompression erzielen.

Als Beispiel werden die Entropien von U und K berechnet:

$$H(K) = 0.1 \cdot \log_2 10 + 0.2 \cdot \log_2 5 + 0.3 \cdot \log_2 3.333 + 0.4 \cdot \log_2 2.5 = 2.046 \text{ bits}$$
$$H(U) = 0.1 \cdot \log_2 10 + 0.4 \cdot \log_2 2.5 + 0.2 \cdot \log_2 5 + 0.2 \cdot \log_2 5 = 1.790 \text{ bits}$$

2.2 Einfluss der Beziehungen zwischen Informationsgehalt und Entropie

Die Beziehungen zwischen den Nukleotiden innerhalb einer Nukleotidsequenz sind für Komprimierungsverfahren von Bedeutung. Durch die Berücksichtigung dieser Beziehungen können die Zusammenhänge zwischen den Informationsgehalten einzelner Basen abgeschätzt werden. Dies ermöglicht eine präzisere Modellierung der Datenstruktur und führt zu einer effizienteren Komprimierung, da Redundanzen und Muster innerhalb der Nukleotidsequenz besser erkannt und ausgenutzt werden können.

Um die Wahrscheinlichkeit $p(AG)$ abzuschätzen, dass zwei unabhängige Nukleotiden AG nacheinander auftreten, wird ihre Gesamtentropie als die Summe ihre einzelnen Entropien berechnet [4]:

$$i(AG) = \log_2 \tfrac{1}{p(AG)} = \log_2 \tfrac{1}{p(A)p(G)} = \log_2 \tfrac{1}{p(A)} + \log_2 \tfrac{1}{p(G)} = i(A) + i(G)$$

Die Logarithmenregel, dass der Logarithmus eines Produkts als die Summe der Logarithmen der Faktoren dargestellt werden kann, ermöglicht also die Abschätzung der Gesamtentropie einer Datenmenge.

2.2.1 Relative Entropie

Die Relative Entropie (auch Kullback-Leibler-Divergenz D_{KL}) gibt an, um wie viel sich die Wahrscheinlichkeitsverteilungen in einer Informationsquelle von einer Referenz- oder Zielverteilung unterscheiden [4]. Es wird gemessen, wie viel zusätzliche Information benötigt wird, um P anstelle der Verteilung Q zu beschreiben.

Dabei ist die mittlere Codelänge für die Zeichen der Wahrscheinlichkeitswerteilung P in Bezug auf die Wahrscheinlichkeitsverteilung Q:

$$D_{KL}(P||Q) = \sum P(i) \log \frac{P(i)}{Q(i)}$$

Die relative Entropie D_{KL} der Nukleotidsequenz K in Bezug auf U, die zusammen zum Genom des Parvovirus B19 gehören, wäre also:

$$D_{KL}(U||K) = 0.4 \cdot \log \frac{0.4}{0.2} + 0.2 \cdot \log \frac{0.2}{0.3} + 0.1 \cdot \log \frac{0.1}{0.4} + 0.2 \cdot \log \frac{0.2}{0.1} = 0.085 \,\text{bits}$$

2.2.2 Kreuzentropie

Die Kreuzentropie $H(P||Q)$ stellt eine Beziehung zwischen $D_{KL}(P||Q)$ und $H(K)$ her. Mittels dieser Angabe kann die mittlere Codelänge einer Entropiecodierung abgeschätzt werden [8]. Je stärker sich die Wahrscheinlichkeitsverteilungen (Q) und (P) voneinander unterscheiden, desto länger wird die

mittlere Codelänge der codierten Nukleotidsequenz sein, wenn Q anstelle von P verwendet wird.

$$\begin{aligned}\sum p_i \log \frac{1}{q_i} &= \sum p_i \log \frac{1}{q_i} \\ &= \sum p_i \log \frac{1}{p_i \cdot q_i} \\ &= \sum p_i \log \frac{1}{p_i} + \sum p_i \log \frac{1}{p_i \cdot q_i} \\ H(P||Q) &= H(P) + D(P||Q)\end{aligned}$$

$$C(K) = H(K) + D_{KL}(U||K) = 2.046 + 0.085 = 2.131 \text{ bits}$$

2.2.3 Bedingte Entropie

Die Bedingte Entropie gibt eine Angabe, wie die Wahrscheinlichkeit des Auftretens eines Symbols in einem Kontext die Entropie der gesamten Datei reduzieren kann. Der Kontext kann eine Menge von vorhergehenden oder benachbarten Symbolen sein.

Steht eine Nukleotidsequenz S im Kontext C, dann wird die bedingte Entropie für jedes Nukleotid $s \in S$ im Kontext von C so gemessen:

$$H(S|C) = \sum p(s) \sum p(s|c) \log_2 \frac{1}{p(s|c)}$$

Hierbei ist $P(s|c)$ die bedingte Wahrscheinlichkeit, dass das Nukleotid s im Kontext C auftritt. Durch die Nutzung dieser bedingten Wahrscheinlichkeiten kann der Informationsgehalt effizienter erfasst werden, was zur Verringerung der Entropie beiträgt. Falls S kontextunabhängig ist, dann würde $H(S|C) = H(S)$ gelten, sonst $H(S|C) < H(S)$, was eine Reduktion der Entropie bedeutet. Diese Eigenschaft der bedingten Entropie wird bei den Komprimierungsverfahren Prediction by Partial Matching (PPM) und LZ77 ausgenützt [8].

Das Markov Modell

Ein Diskrete-Zeit-Markov-Prozess ist ein stochastisches Modell, das die Abhängigkeiten zwischen den Daten beschreibt. Im Kontext dieser Arbeit bezieht sich „Daten“ auf die einzelnen Nukleotide in einer Nukleotidsequenz. Das Markov-Modell betrachtet hier also die Wahrscheinlichkeit, dass ein bestimmtes Nukleotid auftritt, basierend auf den vorhergehenden Nukleotiden

in der Sequenz. Sei $\{x_n\}$ eine Nukleotidsequenz. Die Nukleotidsequenz folgt einem Markov-Prozess k-ter Ordnung, wenn:

$$P(x_n|x_{n-1}, ..., x_{n-k}) = P(x_n|x_{n-1}, ..., x_{n-k}, ...)$$

In einem Markov-Modell k-ter Ordnung wird davon ausgegangen, dass das zukünftige Verhalten des Prozesses nur von den letzten k Nukleotiden abhängt und dass noch frühere Nukleotide keinen Einfluss mehr haben. Dies erleichtert die Modellierung und Vorhersage von Nukleotidsequenzen, da nur eine begrenzte Anzahl von vorherigen Nukleotiden berücksichtigt werden muss. Wenn der Zeichenvorrat aus l Symbolen besteht, dann ist die Anzahl der möglichen Nukleotiden ist l^k. Daher ist ein Markov Prozess erster Ordnung:

$$P(x_n|x_{n-1}) = P(x_n|x_{n-1}, x_{n-2}, x_{n-3}, ...)$$

Diese Gleichung beschreibt die Abhängigkeit zwischen dem Auftreten der Symbole der Nukleotidsequenz:

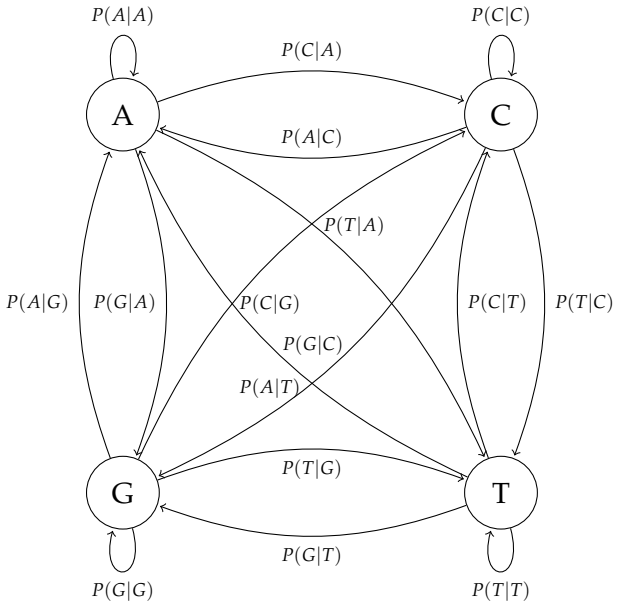

Abbildung 2.1: Markov-Kette zur Modellierung der Wahrscheinlichkeiten zwischen Nukleotiden.

2.3 Codierung und Decodierung

Ein Alphabet Z ist eine diskrete Menge von Symbolen:

$$Z = \{a_1, a_2, \ldots, a_n\}$$

Z ist das Alphabet und $a_1, a_2, \ldots, a_n$ sind die einzelnen Symbole des Alphabets. Z^+ bezeichnet die Menge aller endlichen, nicht-leeren Sequenzen von Symbolen aus Z:

$$Z^+ = \bigcup_{n=1}^{\infty} Z^n$$

Hierbei ist Z^n die Menge aller Sequenzen der Länge n, die aus Symbolen des Alphabets Z gebildet werden können.

Z^+ wird als Menge der *Wörter* über dem Alphabet Z bezeichnet.

Eine Codierung ist eine Funktion c, die eine injektive Abbildung zwischen einer Wortmenge Z^+ und einer Wortmenge B^+ beschreibt [8]:

$$c : Z^+ \implies B^+$$

Dabei bezeichnet B das sogenannte Codealphabet, und $c(Z^+) \subseteq B^+$ wird als Menge der Codewörter bezeichnet.

Zum Beispiel ist es möglich, die vier Nukleotiden in einer Codierung mit dem Quellaphabet Z mit 2 Bits aus dem Codealphabet B darzustellen (Binärcodierung), da 2 Bits eindeutig 2^2 Möglichkeiten darstellen können:

$$Z = \{A, G, C, T\}$$
$$B = \{0, 1\}$$

Dann ist Z^+ die Menge aller möglichen Nukleotidsequenzen, und B^+ die Menge aller möglichen Binärsequenzen, die zur Darstellung der codierten Nukleotidsequenzen verwendet werden.

A	00
G	01
C	11
T	10

Tabelle 2.1: Eine mögliche Codierung der Nukleotiden.

Die Decodierung ist die Umkehrabbildung der Codierung:

$$c^{-1} : c(Z^+) \implies Z^+$$

Die Decodierungsfunktion c^{-1} nimmt die codierte Nukleotidsequenz aus B^+ und stellt die ursprüngliche Nukleotidsequenz aus Z^+ wieder her. Die Eindeutigkeit der Codierung und Decodierung wird durch die Injektivität der Abbildung sichergestellt, wodurch eine verlustfreie Übertragung und Speicherung von Informationen ermöglicht wird.

Um eine optimale und eindeutig decodierbare Zuweisung zwischen Z^+ und B^+ zu erreichen, müssen die zugewiesenen Codewörter *Präfixcodes* sein [8]. Ein Präfixcode ist eine spezielle Art von Codierung, bei dem kein Codewort den Anfang eines anderen Codewortes bildet. Diese Eigenschaft stellt sicher, dass die Codierung eindeutig und ohne Mehrdeutigkeiten decodiert werden kann. Um dies zu verdeutlichen, betrachten wir ein Beispiel für einen Nichtpräfixcode und die Probleme, die dabei auftreten können:

Angenommen, wir weisen 10 dem Symbol a_1, 1 dem Symbol a_2 und 101 dem Symbol a_3 zu. Wenn wir nun das Codewort 101 decodieren müssen, ist nicht klar, ob der codierte Text a_1a_2 (also 10 gefolgt von 1) oder a_3 entspricht. Dies liegt daran, dass 10 ein Präfix von 101 ist, was zu dieser Mehrdeutigkeit bei der Decodierung führt. Ein solcher Codierung ist *nicht* eindeutig decodierbar, was ein Problem für die korrekte Wiederherstellung der ursprünglichen Nukleotidsequenz darstellt.

Präfixcodes erfüllen die Kraftsche Ungleichung [8]. Diese lautet folgendermassen: Angenommen, dass es Codewörter für $a_1, a_2, ..., a_n$ gibt, die jeweils Längen von $l_1, l_2, ..., l_n$ haben. Dann gilt:

$$\sum_{i=1}^{m} 2^{-l_i} \leq 1$$

Diese Ungleichung stellt sicher, dass die Gesamtlänge der möglichen Codewörter in einem binären Baum nicht grösser ist als die Anzahl der verfügbaren Plätze im Baum. Dadurch wird garantiert, dass kein Codewort das Präfix eines anderen ist. Umgekehrt gilt: Wenn eine Menge $O = l_1, l_2, ..., l_i$ die Kraftsche Ungleichung erfüllt, dann existiert ein Präfixcode mit diesen Längen.

Die mittlere Codelänge l hängt von der Entropie ab. Für ein Alphabet Z mit m Symbolen kann mit der folgenden Formel berechnet werden [8]:

$$l_Z = \sum_{n=1}^{m} p(a_i) \cdot n(a_i)$$

Wo $p(a_i)$ die Wahrscheinlichkeit des Auftretens von Symbol a_i, und $n(a_i)$ die Anzahl von Bits für das Codewort a_i bezeichnen.

Obwohl daher die Entropie im obigen Beispiel ungefähr 2 Bits betrug, $H(U) = 2.046$ Bits, können einheitlichere Nukleotide eine niedrigere Entropie aufweisen und daher wären 2.046 Bits im Durchschnitt zu viel für die Codierung und es wird sich eine suboptimale Kompression ergeben. Die Präfixcodes ermöglichen es, häufig auftretenden Nukleotiden kürzere Codewörter und selten auftretenden Nukleotiden längere Codewörter zuzuweisen, was zu einer effizienteren Codierung und geringeren durchschnittlichen Codelängen führt.

2.3.1 Arten von Codierung

Es gibt drei Hauptgruppen von Codierungsstrategien für verlustfreie Komprimierung – *Lauflängencodierung*, *Wörterbuchcodierung* und *Entropiecodierung* [9]. Alle Codierungsstrategien bestehen aus einem Encoder und einem Decoder. Ein Encoder ist der Teil, der eine Nukleotidsequenz codiert, indem er jedem Nukleotid der Nukleotidsequenz ein entsprechendes Codewort aus dem Codealphabet zuweist, oft mit dem Ziel, die Datenmenge zu reduzieren. Der Decoder ist der Gegenpart, der die codierte Nukleotidsequenz wieder in die originale Nukleotidsequenz zurückverwandelt, indem er die Codewörter erkennt und sie zurück in die ursprünglichen Nukleotide des Quellalphabets umwandelt.

Lauflängencodierung

Eine einfache Form von Codierung ist die Lauflängencodierung (Englisch: Run Length Encoding, RLE) [10]. Mehrmals nacheinander auftretende Symbole werden durch die Anzahl ihres Auftretens ersetzt.

Die Nukleotidsequenzen Y hat zwei Nukleotide, Guanin und Cytosin, die jeweils zweimal und fünfmal nacheinander auftreten. Die Nukleotide Adenin und Thymin treten nur einzeln auf.

$$Y = [AGGATATATCCCCC]$$

Nach der Lauflängecodierung werden die sich wiederholenden Symbole gelöscht und die Nukleotidsequenz wird mit den Anzahlen ergänzt:

$$Y = [A1G2A1T1A1T1C5]$$

Wörterbuchcodierung

Wörterbuchcodierer suchen nach Repetitionen in den zu komprimierenden Daten und speichern sie in einem „Wörterbuch“. Wenn der Encoder ei-

ne Übereinstimmung findet, ersetzt er einen Verweis auf die Position der Repetition in der Datenstruktur, welche das Wörterbuch enthält [11]. Der LZ77-Algorithmus verwendet ein dynamisches Wörterbuch, das die repetierende Symbolfrequenzen mithilfe eines Schiebefenster (Englisch: Sliding Window) aufnimmt und komprimiert. Das Schiebefenster sucht nach repetierenden Sequenzen von Symbolen innerhalb eines Bereichs. Zum Beispiel hat ein 32-Kilobyte Schiebefenster einen Bereich von 32'768 Bits. Wenn der LZ77-Algorithmus eine bereits aufgetretene Sequenz findet, wird sie durch 2 Angaben ersetzt: ihren Startindex, der vom Anfang der ganzen Sequenz gezählt wird, und die Länge der sich wiederholenden Sequenz.

Als Beispiel wird die Nukleotidsequenz X mit einem Wörterbuchcodierer komprimiert:

$$X = [TGCATGCATGCATGCATGCA]$$

Wenn das Fenster auf dem fünften Symbol ist, ist die Nukleotidsequenzen F im Schiebenfenster $F = [TGCA]$. Daher kann der Algorithmus erkennen, dass die nächsten 5 Symbole genau gleich sind und die Sequenz wie folgt modifizieren:

$$X = [TGCAT(D = 5, L = 5)GCATGCATGCA]$$

Während das Fenster expandiert, kommt es zu einer wachsenden Komprimierung der Sequenz:

$$X = [TGCAT(D = 5, L = 18)]$$

In dieser Arbeit wird der Deflate Algorithmus untersucht, der eine Kombination von Wörterbuchcodierer (LZ77) und Entropiecodierer (Huffman-Codierung) verwendet. [12].

Entropiecodierung

Entropiecodierung, auch Codierung variabler Länge genannt, betrachtet die Symbole in einer Menge $S = \{a_1, a_2, a_3, ..., a_m\}$ mit Kardinalität $|S| = m$ zusammen mit den Wahrscheinlichkeiten p_i jedes Symbols a_i aus dem Wahrscheinlichkeitsmodell. Jedes Symbol a_i wird dann binär codiert, wobei die Länge des Codeworts $q_0 = \log_2(m)$ Binärzeichen ist.

Der Decoder enthält das Mapping der Symbole auf Binärcodes und die jeweiligen Längen q_i der codierten Symbole. Durch die Kenntnis dieses Mappings und der Längen kann der Decoder die binäre Zeichenfolge korrekt

decodieren. Anstatt nur die Länge der Symbole zu verwenden, nutzt der Decoder die gesamte Zuordnungstabelle (Mapping) zwischen Symbolen und Binärcodes, um die Originalsymbole wiederherzustellen.

Zum Beispiel: Angenommen, die Symbole a_1, a_2 und a_3 haben die Wahrscheinlichkeiten $p_1 = 0.5$, $p_2 = 0.25$ und $p_3 = 0.25$. Dann könnte das Symbol a_1 mit einem kürzeren Binärcode wie 0 codiert werden, während a_2 und a_3 längere Binärcodes wie 10 und 11 erhalten.

Dieses Prinzip der Entropiecodierung führt zu einer effizienteren Nutzung der Datenübertragungsrate und des Speicherplatzes, da die durchschnittliche Codelänge minimiert wird, indem die Häufigkeit der Symbole berücksichtigt wird. Die meisten Codierungsstrategien in einem verlustfreien Komprimierungsverfahren basieren auf der Erstellung von Codewörtern mit variabler Länge, deren Quellensymbole aus Binärzeichen bestehen. Huffman-Codierung, Arithmetische Codierung und Prediction by Partial Matching sind bekannte Codierungsstrategien mit variabler Länge, die in dieser Arbeit betrachtet werden.

Arithmetische Codierung Die Arithmetische Codierung wird in dieser Arbeit als eine alternative Entropiecodierung zur Huffman-Codierung untersucht. Im Gegensatz zur Huffman-Codierung kann die Arithmetische Codierung an veränderliche Wahrscheinlichkeitsverteilungen angepasst werden. Dies ist besonders nützlich bei Quellen mit dynamischen oder adaptiven Wahrscheinlichkeiten, da die Arithmetische Codierung kontinuierlich die Wahrscheinlichkeiten während des Codierungsprozesses aktualisieren kann.

Der Komprimierungsprozess der Arithmetischen Codierung beginnt mit der Initialisierung des Intervalls $[0, 1)$. Dieses Intervall repräsentiert den gesamten Wertebereich, innerhalb dessen die gesamte zu codierende Datei liegt [13]. Für jedes Symbol $a_i \in S$ wird das aktuelle Intervall basierend auf der Wahrscheinlichkeit des Symbols p_i in kleinere Teilintervalle unterteilt. Angenommen, $P(a_1), P(a_2), \ldots, P(a_n)$ sind die Wahrscheinlichkeiten der Symbole $a_1, a_2, \ldots, a_n$, dann wird das Intervall $[l, r)$ in Teilintervalle aufgeteilt, wobei jedes Teilintervall die Breite $P(a_i)$ hat. Das neue Intervall $[l', r']$ für das Symbol a_i wird durch die folgenden Formeln berechnet:

$$l' = l + (r - l) \cdot \sum_{j=1}^{i-1} P(a_j)$$

$$r' = l + (r - l) \cdot \sum_{j=1}^{i} P(a_j)$$

Dieser Prozess wird für jedes nachfolgende Symbol der Datei wiederholt, wobei das Intervall immer weiter eingeengt wird. Am Ende des Kompri-

mierungsprozesses repräsentiert ein Punkt innerhalb des finalen Intervalls die gesamte Datei. Um die Datei tatsächlich zu übertragen oder zu speichern, wird ein Wert innerhalb des finalen Intervalls ausgewählt und in eine Binärdarstellung umgewandelt. Diese Darstellung ist so gewählt, dass sie das Intervall eindeutig identifiziert und die Datei effizient codiert.

Die Fähigkeit der Arithmetischen Codierung, das Intervall kontinuierlich anzupassen und zu verfeinern, ermöglicht eine extrem effiziente Nutzung der verfügbaren Bitlänge, besonders bei Quellen mit nicht-statischen Wahrscheinlichkeiten. Dies führt zu einer Codierung, die näher an der theoretischen Entropiegrenze liegt, als dies mit der Huffman-Codierung möglich ist [13].

Das Wahrscheinlichkeitsmodell der Nukleotidsequenz $K = [C, A, A, G, G, T, G, T, T, T]$ hat die folgende diskrete Wahrscheinlichkeitsverteilung $P(X_K)$:

X_K	A	G	T	C
$P(X_K)$	0.2	0.3	0.4	0.1

Die Codierung beginnt mit der Erstellung des Intervalls, wobei jedem Symbol ein Teilintervall zugewiesen wird:

$$
\begin{aligned}
C &\in [0; 0.1) \\
A &\in [0.1; 0.3) \\
G &\in [0.3; 0.6) \\
T &\in [0.6; 1)
\end{aligned}
$$

Dann wird das entsprechende Intervall iterativ für jedes Nukleotid in der Sequenz K partitioniert. Das erste Nukleotid in K ist C, $C \in [0.9; 1)$. Das nachfolgende Nukleotid ist A, entsprechend heisst das neue Intervall $[0.2 \cdot 0.9; 0.2 \cdot 1] = [0.18; 0.1]$ und so weiter. Die Sequenz wird als Bruch 0.013861017600000003 codiert und die Dezimalzahl binär dargestellt. Das Intervall erhält somit die folgende Verteilung:

$[0.0, 0.1, 0.30000000000000004, 0.6000000000000001, 1.0)$.

Diese Dezimalzahlen werden im nächsten Schritt in Binärzahlen umgewandelt. Bei der Decodierung werden dasselbe Modell verwendet, der Vorgang jedoch rückwärts ausgeführt. Beim ersten Schritt der Decodierung wird der Bruch 0.013861017600000003 betrachtet. Da der Wert zwischen 0.0 und 0.1 liegt, sollte das erste codierte Symbol C sein, was dem ursprünglichen Anfang der Sequenz entspricht.

Prediction by Partial Matching Als Letztes und eines der neusten verlustfreien Komprimierungsverfahren folgt Prediction by Partial Matching (PPM). Der bedeutendste Unterschied zwischen diesem Verfahren und den anderen Verfahren, die in dieser Arbeit untersucht werden, ist die Beachtung des Kontexts anhand eines Markov-Modells [14]. Im Gegensatz zu LZ77, wo der Kontext innerhalb eines Schiebefenster beachtet wird, wird bei PPM ein Markov-Modell verwendet [8]. Das Markov-Modell innerhalb von PPM versucht stochastisch, basierend auf den Frequenzen und der Struktur der bisher betrachteten Symbole, das nächste Symbol oder die nächste Teilsequenz vorauszusagen [14].

2.3.2 Evaluation eines Komprimierungsverfahrens

Ein Komprimierungsverfahren wird hauptsächlich durch sein Kompressionsverhältnis beurteilt. Das Kompressionsverhältnis R ist das Verhältnis zwischen den Grösse der unkomprimierten Version n_1 zur Grösse der komprimierten Version n_2 einer Datei [8]. Da die Grösse der komprimierten Version kleiner sein sollte als die Grösse der unkomprimierten Version, wurde die Reihenfolge in dieser Definition so gewählt, dass gilt: Je *grösser* R ist, desto besser ist die erreichte Kompression.

$$R = \frac{n_1}{n_2}$$

Zusätzlich werden Kompressions- und Dekompressionsgeschwindigkeit und Speicherverbrauch betrachtet. Diese können jedoch stark vom Betriebssystem, dem Prozessor und dem verfügbarem Speicherplatz abhängig sein. Die Fähigkeit eines Algorithmus, sich an verschiedene Datentypen und statistische Eigenschaften anzupassen sowie seine Komplexität und Implementierbarkeit werden ebenfalls berücksichtigt. Bei genetischen Daten ist eines der wichtigsten Kriterien bei der Evaluation die *Fehlerresistenz und Robustheit* des Algorithmus. Denn kleine Fehler in Erbdaten sind Mutationen und können im Endeffekt Proteine beschreiben, die ganz unterschiedliche Eigenschaften haben [15].

Kapitel 3

Forschung

3.1 Informationsquellen und Informationsverarbeitung

Als erster Schritt der Forschung wurden passende Informationsquellen gesucht. Damit die Resultate vertretbar sind, wurden echte Informationsquellen gewählt und es wurde berücksichtigt, dass die Länge der Informationsquellen variieren soll. Konkret wurde das Genom des Parvovirus B19, ein einsträngiger DNA-Virus der aus $5,6kb$ (kb: Kilobasen = $1 \cdot 10^3$ Nukleotiden), als eine sehr einfache Informationsquelle gewählt [5]. Als eine grössere Informationsquelle wurde das Genom des Human Alphaherpesvirus 2 (HSV-2) gewählt. HSV-2 ist ein doppelsträngiger DNA-Virus, der aus $156kb$ besteht und entsprechend grösser und komplizierter als das Parvovirus B19 ist [16].

In der Praxis werden Daten stückweise konstruiert oder mit zusätzlichen Metadaten zusammengestellt. Es existiert eine Vielfalt von Datenformaten, jeweils mit verschiedener Struktur und Zweck [2]. Die Dateien bestehen daher nicht nur aus reinen Nukleotidsequenzen, sondern enthalten mehrere Meta-Informationen, wie Verwandschaften innerhalb der Sequenz mit anderen Sequenzstücken des Genoms, Stämme, oder anderen Genomen. Zum Beispiel werden die Nukleootidsequenzen nach der Sequenzierung in FASTA/FASTQ gespeichert.

Danach werden die FASTA/FASTQ Dateien in SAM (Sequence Aligment Map) oder BAM-Dateien (Binary Alignment Map) gespeichert, die auch Information betreffend der Ausrichtung zum Referenzgenom enthalten. Ein *Referenzgenom* ist eine etablierte Nukleotidsequenz, die als Basis genommen wird und mit anderen sequenzierten Genomen verglichen werden kann. [17].

Ferner können die SAM/BAM Dateien in VCF (Variant Call Format) Dateien umgewandelt werden und mit zusätzlichen Metainformationen wie genetische Variationen, Insertion- und Deletion-Mutationen gegenüber dem Referenzgenom angereichert werden [2].

Es gibt daher verschiedene Komprimierungsverfahren, die je nach Datentyp entwickelt wurden. Das SPRING Komprimierungsverfahren ist auf FASTA/FASTQ Dateien spezialisiert [18]. Genozip hingegen wurde zunächst als ein Komprimierungsverfahren für VCF-Dateien entwickelt, und im Laufe der Zeit wurden Optionen für SAM/BAM-Dateien hinzugefügt [19].

Besonders auffällig ist, dass Genozip und SPRING unterschiedliche Codierer für jede Datenstruktur *innerhalb* der Dateien verwenden. Im Hinblick auf die Informationstheorie können die verschiedenen Datenstrukturen als verschiedene Datenquellen betrachtet werden, jede mit eigenenWahrscheinlichkeitsverteilungen und Entropien, deren Besonderheiten für unterschiedliche Codierer passen. Für FASTQ Dateien verwendet Genozip seinen eigenen ‚acgt'-Codierer für die Nukleotidsequenz-Felder, und ‚domqual' für die Qualitätskennzahlen [20]. Weil diese Arbeit auf reine Nukleotidsequenzen ausgerichtet ist, habe ich die rohen Nukleotidsequenzen aus den FASTA-Dateien der Genome der beiden Viren gewonnen und mit diesen weiter gearbeitet.

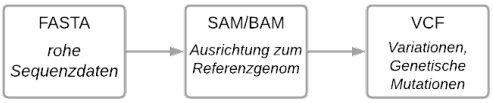

Abbildung 3.1: Addition zusätzlicher Metadaten in genetischen Dateien

Für die Ausführung der Experimente habe ich meinen privaten Laptop verwendet. Die Implementationen der Algorithmen, die die Information verarbeiteten, sind in C++ geschrieben und verwenden Open-Source-Software (OSS). OSS bezeichnet Software, deren Quellcode öffentlich zugänglich ist und von jedermann eingesehen, verändert und weiterverbreitet werden darf.

3.2 Erste Komprimierungsversuche

Als Ausgangslage hatte ich nur eine grobe Idee, was verlustfreie Algorithmen sind, und noch keine Vorkenntnisse im Bereich Informationstheorie oder spezifische Erfahrung mit der Implementation von Komprimierungsprozessen. Um eine grobe Übersicht über die verschiedenen Komprimierungsstrategien zu bekommen, habe ich je einen Algorithmus für jede Komprimierungsstrategie gewählt, was auch tatsächlich zu aussagekräftigen Variationen in den Resultaten geführt hat.

Da Lauflängecodierung die einfachste Form von Codierung ist, habe ich zunächst anhand dieser Methode die Nukleotidsequenz des Parvovirus B19 komprimiert. Die Resultate waren unerwartet – das Kompressionsverhältnis lag bei 0.734, tatsächlich ergab sich keine Komprimierung, sondern eine Vergrösserung der Ausgabe.

Um besser zu verstehen, warum das überhaupt passieren konnte, habe ich eine Datei erstellt, in der die Sequenz TGCA 1200-Mal repetiert wird. Nach der Lauflängencodierung dieser Datei war das Kompressionsverhältnis 0.5, genau zweimal die Grösse der ursprünglichen Datei. Dann wurde mir klar, dass die Lauflängencodierung lediglich kontextlos codiert, und falls die Nukleotiden in der Datei nicht gruppiert nacheinander auftreten, die einzelnen Symbole jedes mal mit ihrer entsprechenden Frequenz (1) bezeichnet werden. Wenn also die Symbole *niemals* mehrmals hintereinander auftreten, wird neben *jedem* Symbol noch ein zusätzliches Zeichen hinzugefügt.

Nach weiterer Analyse habe ich festgestellt, dass die Datei erst mit Burrows-Wheeler-transform (BWT) verarbeitet werden muss. BWT gruppiert die Sequenzstücke, die eine ähnliche Struktur haben, und sortiert sie alphabetisch [21]. Nach der Burrows-Wheeler-Transformation ergab sich ein Kompressionsverhältnis von 1.389, was einer vernünftigen Kompression entspricht.

Mit Huffman-Codierung habe ich eine ähnliche Erfahrung gemacht. Wenn die reine Nukleotidsequenz mit Huffman-Coding codiert wurde, ergab sich wieder eine Vergrösserung: Die Ausgabedatei war 1.967 länger als die Eingabedatei. Auch mit der künstlichen Sequenz von 1200 Repetitionen von TGCA verdoppelte sich die Länge der mit Huffman-Codierung codierten Datei. Die Huffman-Bäume waren zu flach, weil es sich nur um Sequenzen von vier Symbolen - A, G, T, und C - handelt. Ähnlich wie die Notwendigkeit von BWT vor der Lauflängencodierung, wurde eine LZ77-Codierung vor der Erstellung des Huffman-Baumes benötigt, um überhaupt eine Komprimierung zu erreichen. Genau diese Komprimierungsstrategie verwendet auch der Deflate Algorithmus [12].

Dieser Ansatz lässt sich mit Hilfe der Informationstheorie so erklären: Erst durch LZ77 sinkt die Entropie der Information innerhalb der Datei, und daraus werden optimale Präfix-Codes während der Huffman-Codierung erstellt, die den Informationsgehalt besser darstellen. Damit erreicht der Deflate-Algorithmus ein Kompressionsverhältnis von 3.386.

3.3 Evaluation der Ergebnisse

Aus diesen Versuchen ergab sich, dass die Komprimierungsverfahren in der Regel aus einer geschickten Kombination verschiedener Komprimierungsstrategien bestehen und ein statistisches Modell oder eine geeignete

Datenstruktur erfordern, um die Informationen und Zusammenhänge innerhalb der Datei besser darzustellen. Darüber hinaus habe ich festgestellt, dass ein Komprimierungsverfahren nicht zwangsläufig zu einer tatsächlichen Reduktion der Dateigrösse führen muss. Bei der Komprimierung kann entweder die Struktur der Eingabedatei nicht zum Algorithmus passen, oder eine Kanalcodierung erforderlich sein, bei der zusätzliche Daten zur Fehlerkorrektur hinzugefügt werden, was zu einer Vergrösserung der Datei führen kann [8]. *Kanalcodierung* ist ein Verfahren, bei dem zusätzliche Redundanz zu den Daten hinzugefügt wird, um Fehler bei der Decodierung zu vermeiden.

Die beste Komprimierung lieferte das PPM-Verfahren (Prediction by Partial Matching) mit einem Kompressionsverhältnis von 3.446 und einer Laufzeit von 1946*ms*. Die schnellste Komprimierung wurde von Deflate erreicht, mit einer Laufzeit von 783*ms* bei einem Kompressionsverhältnis von 3.386. Beide Komprimierungsverfahren sind kontextbasierte Entropiecodierer [8]. Die Unterschiede in der Laufzeit resultieren wahrscheinlich aus den unterschiedlichen Ansätzen zur Kontextbildung: Deflate verwendet ein Wörterbuch mit Schiebefenster (LZ77), während PPM ein Markov-Modell nutzt.

Die Arithmetische Codierung erreicht ein Kompressionsverhältnis von 3.417 und eine Laufzeit von 1818*ms*. Der Grund für eine solche Leistung könnte die Tatsache sein, dass die Nukleotidsequenz lediglich aus vier Zeichen (G, C, A, T) besteht. Daher werden weniger Unterintervalle während der Codierung erzeugt und die Aufteilung zwischen den Intervallen ist mit genauereren Dezimalzahlen angegeben. Die hohe Genauigkeit der Dezimalzahlen bedeutet, dass sie weniger Zahlen nach dem Komma enthalten, was ermöglicht, dass sie auch in kürzeren Binärzahlen umgewandelt werden können. Da es an jeder Stelle nur 2^2 Möglichkeiten gibt, die nur 2 Bits benötigen, haben die generierten Dezimalzahlen auch einen Wert, der nach Umwandlung in die Basis 2 kürzer ist. In den meisten anderen Fällen wie zum Beispiel bei der Textencodierung, werden jedoch 8 Bits für ASCII-Zeichen benötigt. Die Leistung der Arithmetischen Codierung bei der Kompression von Nukleotidsequenzen wird auch von spezialisierten Komprimierungsverfahren in Anspruch genommen: SPRING verwendet Arithmetische Codierung für die Komprimierung und liefert ebenfalls ein besseres Kompressionsverhältnis im Vergleich zu Wörterbuch-basierten Komprimierungsverfahren wie pigz [18].

3.4 Unterschiede bei den Ergebnissen bei längeren Sequenzen

Die Länge der Nukleotidsequenzen der Genome variiert *erheblich* zwischen verschiedenen Organismen, von einfachen Viren bis hin zu komplexen Säugetieren [22]. Virale Genome, zu denen Parvovirus B19 und HSV-2 gehören und die in dieser Arbeit untersucht werden, gehören zu den klein-

sten Genomen. Bakterielle Genome sind in der Regel grösser und reichen von etwa $5 \cdot 10^5$ Basenpaaren (bp) bis über $10 \cdot 10^6 bp$ [23]. Die Genome von Pilzen variieren stark und liegen typischerweise zwischen $10 \cdot 10^6 bp$ und $10 \cdot 10^7 bp$ [24]. Das Genom der Hefe Saccharomyces cerevisiae hat beispielsweise eine Grösse von etwa $1,2 \cdot 10^7 bp$. Pflanzengenome können sehr gross sein und reichen von $1.0 \cdot 10^6 bp$ bis über $1.0 \cdot 10^{11} bp$. Das Genom von Arabidopsis thaliana beträgt etwa $1.38 \cdot 10^8 bp$, während das Weizengenom ungefähr $1.7 \cdot 10^{10} bp$ umfasst. Auch die Genome von Tieren weisen eine grosse Bandbreite auf. Das menschliche Genom als letztes Beispiel hat eine Länge von etwa $3 \cdot 10^9 bp$.

Bei der Komprimierung vom HSV-2 sind die Unterschiede zwischen den Komprimierungsverfahren und ihre Besonderheiten stärker ausgeprägt. PPM erreichte ein Kompromierungsverhältnis von 4.004, und damit die grösste Verbesserung im Vergleich zu seiner Komprimierung des Parvovirus B19 (3.446). Dies liegt anscheinend an der Verfügbarkeit von mehr Kontext bei der Komprimierung. Arithmetic Coding und Deflate zeigen jeweils ebenfalls bessere Komprimierungsverhältnisse (3.604 für Deflate und 4.024 für Arithmetische Codierung) im Vergleich zu ihren Leistungen bei den kürzeren Nukleotidsequenzen.

Die Laufzeiten der Komprimierungsverfahren stiegen jedoch nicht proportional zu ihren verbesserten Komprimierungsverhältnissen. Deflate benötigte $35370ms$, während PPM mit $18530ms$ nahezu die Hälfte dieser Zeit in Anspruch nahm. Am effizientesten war die Arithmetische Codierung, die die Komprimierung in $9250ms$ bewältigte und damit fast viermal schneller war als Deflate und etwa zweimal schneller als PPM. Diese Unterschiede in der Laufzeit könnten an den unterschiedlichen algorithmischen Komplexitäten und Optimierungsgrade der Verfahren liegen.

Zusammengefasst illustrieren die Ergebnisse, dass die Wahl des optimalen Komprimierungsverfahrens stark von der Länge und Struktur der zu komprimierenden Sequenz abhängt. Während Verfahren wie PPM und Arithmetische Codierung bei längeren Sequenzen signifikante Vorteile in der Komprimierungseffizienz aufweisen, müssen auch die Laufzeiten und Ressourcenanforderungen berücksichtigt werden, um eine optimale Balance zwischen Komprimierungsverhältnis und Verarbeitungszeit zu erreichen. Deflate zeigte hingegen eine erhebliche Verschlechterung bei längeren Sequenzen. Dies unterstreicht die Notwendigkeit einer sorgfältigen Analyse und Auswahl des geeigneten Komprimierungsalgorithmus in Abhängigkeit von den spezifischen Eigenschaften der zu verarbeitenden Daten.

3.5 Beispiel aus der Praxis: Genozip

Ich konnte den Gründer von Genozip erreichen, um mehr Informationen über die aktuellen Forschungen im Bereich der Bioinformatik zu sammeln [25]. Genozip wurde im Jahr 2019 unter dem Namen *vczip* als spezialisierte Komprimierungslösung für VCF-Dateien veröffentlicht [19]. Es bietet eine bessere Komprimierung als gzip, bgzip und bzip2. Genozip verwendet seinen eigenen, massgeschneiderten Algorithmus namens *acgt*, der nur den Unterschied zwischen der zu komprimierenden Sequenz und einem Referenzgenom komprimiert [20]. Der Algorithmus arbeitet exklusiv mit Binärzahlen - die ASCII-Zeichen A, C, G, T werden vor der Verarbeitung in Binärdarstellung umgewandelt.

Genozip zeigt auch die Vielfältigkeit des Komprimierungsproblems der genetischen Dateien. Die Dateien werden in VBlocks, eine eigene Datenstruktur, aufgeteilt, und in parallelen Threads segmentiert, gelesen, komprimiert und zusammengestellt [20]. Ein *Thread* ist eine Ausführungseinheit innerhalb eines Prozesses, die in einem parallelen Ausführungsstrang operiert und dabei gemeinsam genutzte Ressourcen wie Speicher und Variablen des übergeordneten Prozesses verwendet. Threads ermöglichen die gleichzeitige Ausführung mehrerer Berechnungen oder Aufgaben innerhalb eines einzelnen Programms, was zu einer effizienteren Nutzung der Rechenressourcen eines Computers führt.

Dieser Vorgang benötigt Anpassungen im Komprimierungsverfahren für die stückweise Verarbeitung der Dateien. Es wird je nach Datentyp ein anderes Komprimierungsverfahren benutzt. Weiter wurden zusätzlich Features entwickelt, die die Anwendung in einer Data-Pipeline ermöglichen und die Erstellung diverser Statistiken für die Analyse der Daten. Es lässt sich also sagen, dass für die Entwicklung einer effektiven Lösung für das Komprimieren genetischer Dateien, neben gutem Verständnis von Stochastik und Gentechnik auch Erfahrung mit Rechnerarchitekturen benötigt wird, um die Leistung eines Computers maximal auszunutzen.

Kapitel 4

Diskussion

Die Verwendung von Komprimierungsverfahren mit kontextbasierten Modellen, die Entropiecodierer verwenden, weist eine erheblich bessere Leistung bei der Komprimierung von Nukleotidsequenzen auf im Vergleich zu Komprimierungsverfahren, die Lempel-Ziv Wörterbuchcodierung verwenden, vor allem bei längeren Sequenzen. PPM leistet das beste Komprimierungsverhältnis, wobei Arithmetische Codierung die niedrigste Laufzeit hat. Bei längeren Sequenzen nähert sich die Arithmetische Codierung der Komprimierungsleistung von PPM, braucht aber lediglich die Hälfte der Zeit. Wörterbuchcodierer wie LZ77, die bei der Deflate-Komprimierung benutzt werden, sind sehr langsam und bei längeren Sequenzen braucht Deflate doppelt so viel Zeit für die Komprimierung.

Moderne Komprimierungsverfahren wie LZMA und PPM, die eine Kombination aus verschiedenen Komprimierungsverfahren benutzen, sind schneller und komprimieren besser, wobei die älteren Komprimierungsverfahren eine mittelmässige Laufzeit aufweisen und in gewissen Fällen gar keine Komprimierung erreichen. Dafür sind die älteren Komprimierungsverfahren einfacher zu implementieren als die modernen.

Die Speicherung der Sequenz-Daten in binärer Form spielt eine entscheidende Rolle beim Komprimieren von DNA-Sequenzen. Weil die Symbole einer DNA-Sequenz nur 2 Bits zur Darstellung benötigen, ist die Genauigkeit der Dezimalzahlen bei der Arithmetischen Codierung höher. BWT braucht auch weniger Transformationen und entsprechend weniger Zeit. Die Redundanz innerhalb der DNA-Sequenzen, die Aufgrund der biologischen Fehlerkorrektur eine wichtige Besonderheit ist, wird bei Kontext-basierten Komprimierungsverfahren wie PPM und Deflate ausgenutzt und sie zeigen daher eine hervorragende Leistung, vor allem bei längeren Nukleotidsequenzen, bei denen mehr Kontext aufgebaut werden kann.

Es kann davon ausgegangen werden, dass Komprimierungsverfahren, die einen Wörterbuchcodierer verwenden, benachteiligt sind gegenüber anderen Komprimierungsverfahren. Nicht nur die Ergebnisse dieser Arbeit bestätigen das, sondern auch die spezialisierten Kompromierungsverfahren, die genetische Dateien komprimieren: Sowohl die Autoren von Genozip als auch die Autoren von SPRING vergleichen die Leistung ihrer Komprimierungsverfahren mit pigz, gzip und lzma, die ein Lempel-Ziv Wörterbuchcodierung verwenden.

Es geht aber nicht nur um die Kompression einer Datei von Nukleotidbasen, sondern um die parallele Kompression von mehreren verwandten Dateien, die neben Nukleotidsequenzdaten auch andere Metadaten enthalten. Aus diesem Grund verwenden aktuelle Lösungen wie Genozip ihre eigenen massgeschneiderten Algorithmen, um solche Dateien effizient zu komprimieren [20].

Kapitel 5

Ausblick

Welche weiteren Themen würden sich lohnen, angesichts dieser Ergebnisse weiter zu erforschen? Die Dekompression und die Dekompressionszeit der Komprimierungsverfahren konnten im Rahmen dieser Arbeit nicht berücksichtigt werden, trotzdem sind sie eine wichtige Angabe für die Auswertung ihrer Nutzung. In dieser Arbeit wurde eine kurze Einführung in die Informationstheorie gemacht und eine breite Auswahl von verschiedenen Typen von Komprimierungsverfahren betrachtet. Eine genauere Analyse der Entropiecodierer, wie zum Beispiel das Erstellen von Präfixcodes, die Kraft-Ungleichung und Optimierung von Codes mit variablen Codelängen, oder die Auswirkungen der Fehlerkorrektur (Kanalcodierung) wären auch interessant zu untersuchen.

Das Problem der Kompression von genetischen Daten ist ein komplexes wissenschaft-technisches Unternehmen, das interdisziplinäres Wissen und Zusammenarbeit benötigt: Es werden Kenntnisse in der Biologie benötigt, um die Daten zu strukturieren und zu interpretieren. Kenntnisse in der Mathematik sind erforderlich, um passende stochastische Prozesse und Algorithmen zu finden, die diese Daten effizient komprimieren können, und Kenntnisse in der Informatik, um die rechnerischen Ressourcen beim Einsatz der Algorithmen maximal auszunutzen.

Literatur

[1] *What is Genomic Data? - Genomic Data Explained - AWS*, 2023. Adresse: https://aws.amazon.com/what-is/genomic-data/.

[2] L. D. Stein, «The case for cloud computing in genome informatics,» *Genome Biology*, Jg. 11, Nr. 5, S. 207, 2010. DOI: 10.1186/gb-2010-11-5-207. Adresse: https://doi.org/10.1186/gb-2010-11-5-207.

[3] N. Pardi, M. J. Hogan, F. W. Porter und D. Weissman, «mRNA vaccines — a new era in vaccinology,» *Nature Reviews Drug Discovery*, Jg. 17, Nr. 4, S. 261–279, Apr. 2018, ISSN: 1474-1784. DOI: 10.1038/nrd.2017.243. Adresse: https://doi.org/10.1038/nrd.2017.243.

[4] C. E. Shannon, «A Mathematical Theory of Communication,» *The Bell System Technical Journal*, Jg. 27, S. 379–423, 1948. Adresse: http://plan9.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf (besucht am 22. 04. 2003).

[5] Y. Qiu, Z. Zhao und J. Qiu, «Sequences of Seven Complete Genomes of Human Parvovirus B19,» en, *Microbiol Resour Announc*, Jg. 7, Nr. 11, Sep. 2018.

[6] S. Deorowicz und S. Grabowski, «Genome compression: a novel approach for large-scale data management,» *Bioinformatics*, Jg. 29, Nr. 22, S. 2881–2887, 2013.

[7] G. E. Blelloch, *Introduction to data compression*, 2001.

[8] K. Sayood, *Introduction to data compression*. Elsevier, 2006. Adresse: https://www.mbit.edu.in/wp-content/uploads/2020/05/data_compression.pdf.

[9] D. R. Bull, «Chapter 7 - Lossless Compression Methods,» in *Communicating Pictures*, D. R. Bull, Hrsg., Oxford: Academic Press, 2014, S. 213–253, ISBN: 978-0-12-405906-1. DOI: https://doi.org/10.1016/B978-0-12-405906-1.00007-6. Adresse: https://www.sciencedirect.com/science/article/pii/B9780124059061000076.

[10] S. Golomb, «Run-length encodings (Corresp.),» *IEEE Transactions on Information Theory*, Jg. 12, Nr. 3, S. 399–401, 1966. DOI: 10.1109/TIT.1966.1053907.

[11] J. Ziv und A. Lempel, «A universal algorithm for sequential data compression,» *IEEE Transactions on Information Theory*, Jg. 23, Nr. 3, S. 337–343, Mai 1977, ISSN: 1557-9654. DOI: 10.1109/TIT.1977.1055714.

[12] L. P. Deutsch, *DEFLATE Compressed Data Format Specification version 1.3*, RFC 1951, Mai 1996. DOI: 10.17487/RFC1951. Adresse: https://www.rfc-editor.org/info/rfc1951.

[13] I. H. Witten, R. M. Neal und J. G. Cleary, «Arithmetic coding for data compression,» *Communications of the ACM*, Jg. 30, Nr. 6, S. 520–540, 1987.

[14] J. Cleary und I. Witten, «Data Compression Using Adaptive Coding and Partial String Matching,» *IEEE Transactions on Communications*, Jg. 32, Nr. 4, S. 396–402, Apr. 1984, ISSN: 1558-0857. DOI: 10.1109/TCOM.1984.1096090.

[15] L. A. Urry, M. L. Cain, S. A. Wasserman, P. V. Minorsky und J. B. Reece, *Campbell Biologie*. Pearson Deutschland, 2019, S. 1744, ISBN: 9783868943665. Adresse: https://elibrary.pearson.de/book/99.150005/9783863268671.

[16] J. D. Baines und P. E. Pellett, «Genetic comparison of human alphaherpesvirus genomes,» en, in *Human Herpesviruses: Biology, Therapy, and Immunoprophylaxis*, Cambridge: Cambridge University Press, 2007.

[17] N. H. S. England, «Reference genome,» in *Genomics Education Programme Glossary*. Adresse: https://www.genomicseducation.hee.nhs.uk/glossary/reference-genome/ (besucht am 14. 03. 2024).

[18] S. Chandak, K. Tatwawadi, I. Ochoa, M. Hernaez und T. Weissman, «SPRING: a next-generation compressor for FASTQ data,» *Bioinformatics*, Jg. 35, Nr. 15, S. 2674–2676, Dez. 2018, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty1015. eprint: https://academic.oup.com/bioinformatics/article-pdf/35/15/2674/50722542/bioinformatics_35_15_2674.pdf. Adresse: https://doi.org/10.1093/bioinformatics/bty1015.

[19] L. Divon, *Genozip*, https://github.com/divonlan/genozip, 2024.

[20] D. Lan, R. Tobler, Y. Souilmi und B. Llamas, «Genozip: a universal extensible genomic data compressor,» *Bioinformatics*, Jg. 37, Nr. 16, S. 2225–2230, Feb. 2021, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btab102. eprint: https://academic.oup.com/bioinformatics/article-pdf/37/16/2225/50339073/btab102.pdf. Adresse: https://doi.org/10.1093/bioinformatics/btab102.

[21] M. Burrows und D. Wheeler, «A Block-sorting Lossless Data Compression Algorithm,» Digital Equipment Corporation, SRC Research Report 124, Mai 1994, Also republished in 1996.

[22] Z. u. C. u. o. Luo Ming-Cheng und Wang, «An overview of the BGI (Beijing Genomics Institute),» *Science*, Jg. 291, Nr. 5507, S. 1663–1666, 2001.

[23] E. A. u. S. Williams u. a., «The genome sequence of the radioresistant bacterium Deinococcus radiodurans R1,» *Science*, Jg. 286, Nr. 5444, S. 1571–1577, 2002.

[24] P. u. S. Paux Etienne und Sourdille u. a., «In-silico ordering of wheat chromosome 3D,» *Genome biology*, Jg. 9, Nr. 3, S. 1–14, 2008.

[25] L. Divon, Private Communication, divon@genozip.com, 2024.

Anhang A

Appendix A: Kompressionsergebnisse

Die Versuche wurden auf einem Macbook Air, Modell 2023, mit Apple M2 Prozessor und 16 GB Datenspeicher ausgeführt. Das benutzte Betriebssystem war Mac OS Sonoma Version 14.0.

Genom	Verfahren	Compression (Bytes)	Dateigrösse (%)	Komprimierungs-verhältnis	Laufzeit (ms)
Parvovirus B19					
	Huffman Coding	11192	196.731	0.508	2405
	RLE	7750	136.228	0.734	1107
	BWT + RLE	4096	72.000	1.389	-
	Arithmetic Coding	1665	29.267	3.417	1818
	Deflate	1680	29.531	3.386	783
	PPM	1651	29.021	3.446	1946
HSV-2					
	Huffman Coding	302206	191.943	0.521	54602
	RLE	215197	136.680	0.732	13553
	Arithmetic Coding	39125	24.850	4.024	9250
	Deflate	43688	27.749	3.604	35370
	PPM	39314	24.970	4.004	18530