Modellierung und Alignment von Genom-Sequenzdaten

Dissertation

zur Erlangung des akademischen Grades
Doktor der Ingenieurswissenschaften
(Dr.-Ing.)
der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel

Florian Schatz

Kiel
2014

1. Gutachter:	Prof. Dr. Manfred Schimmler
2. Gutachter:	Prof. Dr. Dirk Nowotka
Tag der mündlichen Prüfung:	25.11.2014

Zusammenfassung

Diese Arbeit betrachtet die zentralen Schritte eines Sequenzierungsprozesses und zeigt auf, wie diese modelliert und optimiert bzw. neu gestaltet werden können, um ein performanteres und qualitativ besseres Alignment von Short Read Daten umzusetzen.

Hierzu wird für die Modellierung von Short Reads ein Model hergeleitet, aus dem sich Formeln ableiten lassen, die sehr gute Abschätzungen über die zu erwartenden Ergebnisse eines Sequenzierungsprozesses zulassen. Es werden die Eigenschaften einer Sequenzierung wie die Coverage-Verteilung, Null-Coverage-Verteilung, Contig-Längen-Verteilung, N50 und Contig-Anzahl durch Formeln nachgebildet.

Weiterhin wird ein Algorithmus zum Alignment von Reads an ein Referenzgenom entwickelt. Der Fokus wird auf die statistischen Eigenschaften der Eingabedaten gelegt. Es wird der Schwerpunkt darauf gelegt, dass die Operationen durch die Zugriffszeiten von Arbeitsspeicher und Festplatten den Algorithmus nicht ausbremsen und die Prozessoren optimal genutzt werden. Es wird gezeigt, dass der entwickelte Algorithmus sowohl mehr Reads alignt als auch eine niedrigere Falsch-Positiv Rate als der Referenzalgorithmus Bowtie2 hat und dabei die Laufzeit deutlich unter der des Referenzalgorithmus liegt.

Zur Verbesserung der Eingabedaten wird eine Erweiterung des SHREC Algorithmus zur Fehlerkorrektur vorgestellt. Hierbei wird der Speicherbedarf durch Verwendung von Patricia Tries stark reduziert, wodurch größere Read-Sets verarbeitet werden können. Weiterhin wird die essentielle Parameterwahl von SHREC automatisiert, da sie für normale Anwender sehr schwierig zu bestimmen ist. Es wird gezeigt, dass diese Erweiterungen zu einer deutlichen Fehlerreduktion in den Reads führen.

Um den wachsenden Datenmengen gerecht zu werden, werden Cloud-Dienste analysiert und ein Kommunikations-Benchmark vorgestellt. Dieser hilft, die tatsächlich vorliegende Kommunikationsstruktur zu analysieren und vor der Ausführung eines kommunikationsintensiven Algorithmus zu entscheiden, wie die Verteilung von Aufgaben auf die verschiedenen Recheneinheiten zu erfolgen hat.

Abstract

This dissertation examines the main phases of a sequencing process. It shows how these phases can be modeled, optimized, or created differently to produce a short-read alignment that exhibits higher performance and quality.

To achieve this result, a model of short reads is deduced. With this model it is possible to derive formulas that allow good estimations of the result of a sequencing process. The properties' coverage distribution, null coverage distribution, contig length distribution, N50, and contig count are modelled from the formulas.

In addition, an algorithm for aligning reads to a reference genome is developed that focuses on the statistical properties of the input data. Computational operations are emphasized so that they are not limited by the access speed of the main memory or hard drive and the processors are being utilized optimally. The result shows that the algorithm aligns more reads and has a lower false positive rate than the reference algorithm Bowtie2, even with a much lower execution time.

To enhance the input data, an extension of the error correction algorithm SHREC is presented. Here, the needed memory is reduced by using patricia tries, allowing the algorithm to process bigger read sets. Beyond this, the needed parameters of SHREC are automated because they are hard to choose for a user. The results show that these extensions lead to a decreased error rate in reads.

Cloud services are analyzed to cope with the growing amount of data. A communication benchmark is described, which helps analyze the communication structure before running an algorithm. With the collected statistics of the benchmark it is possible to decide on the distribution of tasks to the different processing units.

Vorwort

Ich möchte mich herzlich bei Herrn Prof. Dr. Schimmler dafür bedanken, dass er meine Begeisterung für die Informatik wesentlich geprägt, mir die Möglichkeit zur Promotion gegeben und deren Betreuung übernommen hat.

Ein besonderer Dank gilt meinem Großvater, der mir Ehrgeiz, Strebsamkeit und Durchhaltevermögen vorgelebt hat.

Meiner Mutter möchte ich für die von ihr mitgegebene, positive Lebenseinstellung danken. Sie hat es mir in anstrengenden Phasen ermöglicht mit Freude weiter zu forschen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AMI	Amazon Machine Image
AWS	Amazon Web Service
BLAST	Basic Local Alignments Search Tool
bp	Basenpaar eines Genoms
BWA	Burrows Wheeler Aligner
BWT	Burrows Wheeler Transformation
de novo	(-Sequenzierung) Bestimmung einer Sequenz ohne Verwendung einer Referenzsequenz
DNA	engl., siehe DNS
DNS	Desoxyribonukleinsäure, sprachl. Verwendung meist DNA
EC2	Amazon Elastic Cloud Compute
FM	Full-text Minute (Index)
FP	Falsch-Positiv (Rate)
FPGA	engl. Field Programmable Gate Array, Wiederprogrammierbarer Chip
GC	GC-Anteil, Anteil der Nukleotide G und C in einem Genom
GPU	engl. Graphics Processing Unit, Grafikprozessor
HG18	engl. Human Genome, Humanes Referenzgenom in der Version 18
HG19	siehe HG18
HPC	engl. High Performance Computing, Hochleistungsrechnen
Indel	engl. Insertion or Deletion, Einfügung oder Löschung einer Base in einer Genomsequenz
MPI	Message Passing Interface
N50	N50 Statistik, Metrik zur Bestimmung der Güte einer Assemblierung
NGS	engl. Next Generation Sequencing
PCR	engl. Polymerase Chain Reaction, Methode zur Vervielfält. von DNA
PU	Processing Unit, Recheneinheit
S3	Amazon Simple Storage Service
SA	Spektrales Alignment
SIMD	Single Instruction Multiple Data
SNP	engl. Single Nucleotide Polymorphism, Unterschied eines bp im Vergleich zu einer Referenz

SW	Smith-Waterman (Algorithmus)
TP	engl. True-Positiv, Richtig-Positiv
VHDL	engl. Very High Speed Integrated Circuit Hardware Description Language, Hardwarebeschreibungssprache
WGS	engl. Whole Genome Shotgun
x86	x86-Prozessor-Architektur

1 Einleitung

1.1 Hintergrund

Jedes Lebewesen trägt die es bestimmenden Informationen in Form eines sehr langen Moleküls, der DNS, in jeder Zelle in sich. Die DNS legt hierbei fest, wie dieses Lebewesen aufgebaut ist und wie seine Steuerungsprozesse in ihm vorgehen. So sind z.B. die Augenfarbe, aber auch einige Verhaltensweisen im Menschen durch dessen DNS vorbestimmt.

Die Erforschung, wie und vor allem wo diese Eigenschaften in der DNS kodiert sind, ist sehr jung und wurde technologiebedingt überhaupt erst seit Ende des 20. Jahrhunderts möglich. Es eröffnet Chancen, wie z.B. genetisch bedingte Krankheiten zu heilen, ihnen vorzubeugen oder Nutzpflanzen so zu modifizieren, dass sie robuster sind und höhere Erträge liefern.

Um aber derartige Eigenschaften erforschen zu können, ist es als Erstes notwendig, dass das DNS-Molekül *gelesen* und so in eine digitale Form überführt wird, da die Sequenz eines komplexeren Lebewesens nur mit Hilfe eines Computers verarbeitet werden kann. Dieser Prozess des *Ablesens* wird als Sequenzierung bezeichnet.

Genauer betrachtet ist es notwendig, eine Vielzahl von Schritten im Sequenzierungsprozess durchzuführen. Zuerst wird in einem chemischen Ableseprozess eine sehr große Anzahl von sehr kurzen Abschnitten (Reads) der DNS gelesen und digital gespeichert. Das darauf folgende *Alignment* (Ausrichtung/Ordnung) der Reads an eine bekannte DNS Sequenz (Referenzsequenz) ist nun der zentrale Vorgang, um mit den sequenzierten Daten arbeiten zu können. Es ist notwendig, da die Reads an sich sehr kurz sind und nach der Sequenzierung nicht bekannt ist, wo in der DNS sie ihren Ursprung haben. Die Zuhilfenahme einer Referenzsequenz ist hierbei der Schlüssel dazu, dass man, plastisch gesprochen, von einem Puzzle aus 10^9 Teilen das eigentliche Bild dieses Puzzles schon grob kennt, anstatt ein Puzzle zu lösen, dessen dargestelltes Bild nicht bekannt ist. Die Zuhilfenahme einer Referenzsequenz ist hilfreich und verfälscht das Bild nicht, da z.B. zwischen verschiedenen Menschen nur 0,1% der Positionen der DNS (Basen) Mutationen, also Unterschiede, sind [85]. Innerhalb einer Generation sind es oft sogar nur 175 Mutationen [50].

Das Lösen des Puzzles ohne die Kenntnis des Gesamtbildes, also die Bestimmung einer DNS ohne Verwendung einer Referenzsequenz, wird als *de novo* Sequenzierung bezeichnet. Sie findet Anwendung bei vollständig unbekannten Sequenzen, wird aber in dieser Arbeit nicht weiter betrachtet, da der Fokus in den Bereich der Humangenetik gelegt wird und hier eine Vielzahl an Referenzsequenzen verfügbar ist.

1.2 Motivation und Aufbau dieser Arbeit

Die berechnungstheoretische Komplexität dieses Themengebiets wird deutlich, wenn man die Durchsätze aktueller Sequenzierer betrachtet. Der Illumina HiSeq 2500 sequenziert ein menschliches Genom in 27 Stunden und liefert über $1,2 * 10^9$ Reads, die alignt werden müssen, wobei die Kosten unter $1,000 liegen. [47]

Diese großen Datenmengen wiederum sind erst in den letzten Jahren durch neue Sequenziertechniken entstanden. Die zuvor fast ausschließlich verwendete Sanger Sequenzierung [67] aus dem Jahr 1977 hatte bis zur Entwicklung der sogenannten Next Generation Sequenzierung (NGS) nur einen Bruchteil der Datenmengen geliefert. Aktuell gibt es NGS Plattformen wie den Illumina Genome Analyzer, die HiSeq Plattform und weitere Angebote von Herstellern wie Ion Torrent und Pacific Biosciences. [94]

Um einen tieferen Einblick in die Grundlagen des in dieser Arbeit behandelten Gebietes und die Geschichte der Sequenzierung zu bekommen, werden diese im zweiten Kapitel behandelt. Der Leser mit Hintergrundwissen aus diesem Bereich kann dieses Kapitel überspringen, da ggf. unbekannte Begrifflichkeiten in späteren Kapiteln mit einer Referenz in das Grundlagenkapitel gekennzeichnet sind.

Betrachtet man das gesamte Forschungsgebiet des Alignments von Sequenziererdaten, kommen mehrere Fragestellungen auf. Es beginnt mit der Modellierung der Daten, also der Kenntnis der Daten und ihrer Eigenschaften. Das Wissen über die Daten ermöglicht es, schon a priori die Menge der benötigten Daten abzuschätzen und Aussagen über die Machbarkeit einer Sequenzierung zu treffen, wenn z.B. die Menge des genetischen Materials beschränkt ist und nicht bekannt ist, ob bei einer Sequenzierung eine vollständige Sequenz zu erwarten ist. Erste

wichtige Ergebnisse zur Modellierung stammen von Lander und Waterman. Sie entwickelten ein Model für die Lücken in einer Sequenzierung, die durch eine nicht ausreichende Datenmenge entstehen [37]. Über diese Modelle hinausgehend liefert das dritte Kapitel dieser Arbeit neue Beträge in dieser Forschungsrichtung. Es wird unter anderem beschrieben, wie sich die Überdeckung (Coverage) und Vollständigkeit (Contiglängen) mit einem Model fassen lassen.

Das bereits eingeführte Alignment ist der Ausgangspunkt, um Aussagen über Eigenschaften eines Menschen, wie z.B. äußere Merkmale oder vererbbare Krankheiten, treffen zu können. Hierzu werden die Unterschiede auf DNS-Ebene zwischen mehreren Menschen bestimmt, indem die Reads alignt werden. Nun betrachtet man die überlappenden Reads der verschiedenen Menschen und versucht Korrelationen zwischen den Unterschieden auf DNS-Ebene und dem betrachteten Merkmal zu finden. Die Herausforderung beim Alignment ist nun, dass man sehr viele Daten miteinander vergleichen, bzw. geschickte Datenstrukturen verwenden muss, um eine Zuordnung einer Sequenz zu einer Position innerhalb einer Referenzsequenz treffen zu können. Verbreitete Algorithmen zum Alignment sind Bowtie2 [38] und BWA [41]. Das im Fokus dieser Arbeit stehende Alignment wird im vierten Kapitel behandelt. Dabei werden die Ergebnisse des neu entwickelten Algorithmus zum Alignment vorgestellt und auf die Besonderheiten zur Arbeitsweise eingegangen, die es ermöglichen, besser und schneller als die Referenzalgorithmen zu alignen.

Beim biochemischen Lesevorgang der Reads treten verschiedene Arten von Fehlern auf, die den verarbeitenden Algorithmen das Arbeiten auf diesen Daten erschweren. So lag die Fehlerrate pro Base zwischen 1% und 2% für den Illumina Sequenzierer der ersten Generation [13]. Die Disziplin, die sich damit beschäftigt die Fehler schon vor einer Weiterverarbeitung aus den Reads zu entfernen, wird Error Correction (Fehlerkorrektur) genannt. Typische Algorithmen sind HiTeC [29], Reptile [95] und SHREC [75]. Im fünften Kapitel werden neue Erkenntnisse vorgestellt, wie sich der Speicherbedarf bei einem Fehlerkorrektur-Algorithmus reduzieren lässt und eine Parameteroptimierung durchgeführt werden kann.

In vielen Forschungsdisziplinen, wie auch dem Alignment, wachsen die Datenmengen durch neue Messtechniken sehr stark. Die Eingaben für die verarbeitenden Algorithmen sind so groß geworden, dass rechenintensive Abschnitte beschleunigt

werden müssen. Dabei werden häufig Algorithmenteile auf Rechen-Cluster oder spezielle Hardware (FPGAs, GPUs) ausgelagert. Der Nachteil eines Clusters ist allerdings, dass dieser neben den hohen Anschaffungskosten auch bei Nichtauslastung sehr hohe Betriebskosten hat. Ein FPGA oder GPU wiederum, ist nicht skalierbar und kann somit nicht beliebig viel Rechenleistung liefern. Es ist daher in einigen Anwendungsfällen sinnvoll Cloud Computing zu verwenden, also einige Teile der Berechnung auf kurzzeitig angemieteten Rechnern durchzuführen. Zu beachten ist allerdings, dass die Kommunikationsgeschwindigkeiten und Performanz einer hohen Varianz unterliegen. Die Untersuchung der Performanz für Bioinformatik-Anwendungen wurden in [27] untersucht. Deelman et al. untersuchten die Performanz von Simulationssoftware auf Cloud Systemen [10]. Im sechsten Kapitel werden neue Erkenntnisse zur Messung von Kommunikationsgeschwindigkeiten in Cloud-Plattformen vorgestellt und gezeigt, wie sich daraus ein Benchmark entwickeln lässt, der wiederum die Performanz einer Anwendung deutlich verbessern lässt.

Abschließend werden im letzten Kapitel eine Zusammenfassung der Ergebnisse dieser Arbeit gegeben und mögliche Anknüpfungspunkte aufgezeigt.

2 Grundlagen der Genomsequenzierung

2.1 Einführung

In diesem Kapitel werden die biologischen und informatischen Grundlagen zusammengefasst, die für das Verständnis dieser Arbeit notwendig sind. Hierbei wird sich auf ein abstraktes, stark vereinfachtes Level, insbesondere im biologischen Bereich, reduziert. Ein Leser mit Grundkenntnissen der Bioinformatik kann dieses Kapitel überspringen oder zügig querlesen.

Da sich diese Arbeit mit der Modellierung und der Verarbeitung von digitalen Repräsentationen von physikalisch existierenden, mikrobiologischen Einheiten befasst (DNS, Genen,..), ist es sinnvoll, Grundkenntnisse dieser biologischen Abläufe und Einheiten zu besitzen. Entsprechend wird in Kapitel 2.2 eine Einführung gegeben.

Um von den tatsächlich greifbaren, physikalischen Entitäten die digitale Repräsentation zu erhalten, werden diese Teilchen mit Hilfe von Sequenzierern eingelesen. Da dieser Prozess ausschlaggebend für die Güte ist und vor allem weil die Aussagekraft der digitalen Sequenzen von den Sequenzierern abhängt, werden die Grundlagen dieses Prozesses in Abschnitt 2.3 erläutert.

Darauf folgend werden die informatischen Grundlagen in Kapitel 2.4 eingeführt, die sich mit den Repräsentationen der verarbeiteten Daten, üblichen Algorithmen und Datenstrukturen der Verarbeitung beschäftigen.

Dann folgen die technischen Grundlagen in 2.5, die im Wesentlichen die Grundlagen für den Einsatz von Cloud Computing darstellen.

2.2 Biologische Grundlagen

In diesem Kapitel werden die Grundlagen für das Verständnis der biologischen Abläufe erläutert. Hierbei ist die Vermittlung von abstrakten Funktionen auf biologischer Ebene das Ziel. Es wird sich daher zur Vereinfachung auf Prokarioten (zellulare Lebewesen ohne Zellkern) beschränkt, da diese die für diese Arbeit benötigten abstrakten Abläufe ausreichend beschreiben und die Eigenschaften der Eukarioten (zellulare Lebewesen mit Zellkern) die Abläufe deutlich verkomplizierten, ohne dass das Verständnis für diese Arbeit erhöht würde.

2.2.1 DNS

Die Desoxyribonukleinsäure (DNS) ist in jeder Zelle eines Lebewesens vorhanden. Sie trägt in ihr die Erbinformationen und damit die *Anleitung*, wie der jeweilige Organismus aufgebaut ist und wie die Abläufe in ihm stattfinden. Man kann sich die DNS als eine lange Kette bestehend aus den Nukleotiden Adenin (`A`), Thymin (`T`), Guanin (`G`) und Cytosin (`C`) vorstellen. Die Sequenz liegt als Doppelstrang vor, wobei sich immer `A` und `T` bzw. `G` und `C` gegenüber stehen. Diese zusammengehörigen, gegenüberstehenden Basen werden als Basenpaare (bp) bezeichnet. Die Verteilung der Vorkommen der Nukleotide ist in den meisten Organismen nicht gleichmäßig.

Der Doppelstrang wiederum bildet eine Doppelhelix (siehe Abbildung 2.1).[26]

In der DNS gibt es wiederum Abschnitte, die besondere Funktionen abbilden. Die bekanntesten und für diesen Kontext interessanten Abschnitte, sind die Gene, also Sequenzen, die in der Regel eine Länge von 1000-2000bp haben. Grob abstrahiert werden die Gene über einen Prozess in der Zelle direkt anhand einer biologischen Abbildungsvorschrift in Proteine übersetzt, die dann wiederum Funktionen ausführen.

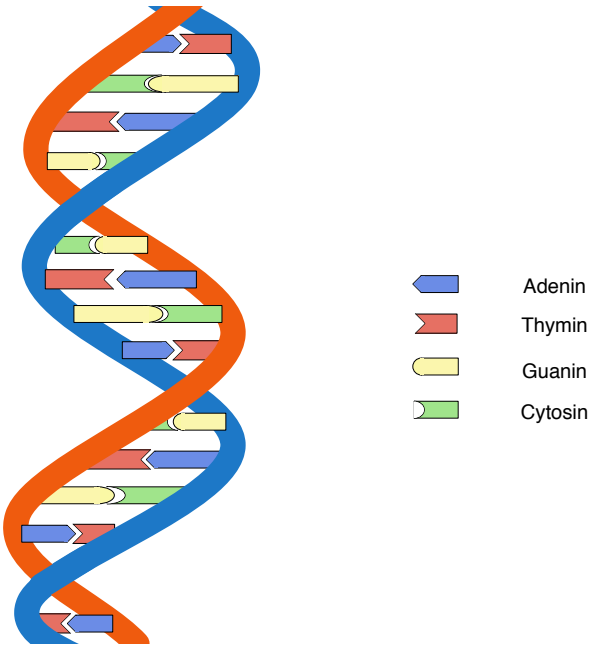

Abb. 2.1: DNS Helix
DNS bildet eine Helix, bei der sich zwei Stränge gegenüberstehen.

2.2.2 SNPs

Single Nukleotide Polymorphisms (SNPs) sind Unterschiede einzelner Nukleotide im Genom im Vergleich zu einem Referenzgenom. SNPs stellen einen Großteil der genetischen Variation innerhalb einer Population dar. Dies bedeutet, dass beispielsweise die Unterschiede zwischen zwei Menschen nach aktuellem Forschungsstand zum Großteil durch sehr wenige SNPs im Vergleich zur Gesamtgröße des Genoms entstehen. Die Unterschiede im Genom führen dann, z.B. bei der Übersetzung eines Gens in ein Protein, zu einem minimal veränderten Protein.

Abb. 2.2: SNPs
Einzelne Basenunterschiede, sog. SNPs, stellen einen Großteil der Variation innerhalb einer Population dar.

2.2.3 Repeat Regionen

Repeat Regionen sind Abschnitte in der DNS, die mehrfach wiederholt vorkommen. Dieses wiederholte Vorkommen kann direkt aufeinanderfolgend, aber auch mit größerem Abstand zu einander sein (siehe Abbildung 2.3). Hierbei ist die Häufigkeit deutlich höher als es durch eine statistische Verteilung der Fall wäre.

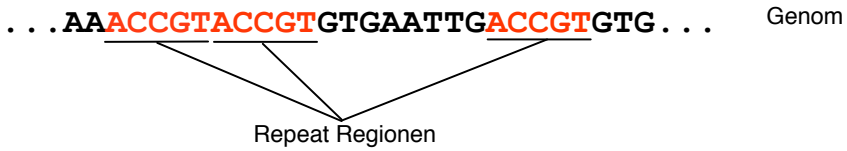

Abb. 2.3: Repeat Regionen
Repeat Regionen sind Sequenzen innerhalb eines Genoms, die sich in identischer oder sehr ähnlicher Form wiederholen.

2.3 Sequenzierungsplattformen

Die ersten Genome wurden 1977 von Maxam und Gilbert [48], bzw. im selben Jahr von Sanger und anderen [67], sequenziert. Letztere Methode war bis vor wenigen Jahren die Referenzmethode zur Sequenzierung, wobei bei dieser Art der Sequenzierung sehr viel manuelle Arbeit notwendig und das Verfahren sehr langsam und damit teuer war. Auf der anderen Seite ist die Basenfehlerrate sehr gering und ermöglicht es, sehr lange Sequenzen zu bestimmen. Es wird noch heute in der Sequenzierung verwendet, wenn es darum geht, einen im Vergleich zum Genom kleinen, aber dafür fest bestimmten Abschnitt, exakt zu sequenzieren.

Erst in den letzen Jahren mit der Einführung des sogenannten Next Generation Sequencing (NGS) oder auch Second Generation Sequencing (SGS) hat sich ein sehr großer Schritt in der Sequenzierung getan. Die neuen Sequenzierer verwenden Techniken, die eine sehr hohe Automatisierung ermöglichen und sehr hohe Datenmengen in kurzer Zeit generieren [3, 11, 66]. Die neuen Verfahren arbeiten so, dass sie basenweise die Sequenz bestimmen, indem sie zu der Sequenz komplementäre Basen an die Sequenz nacheinander anfügen. Je nach Verfahren werden bei diesem Prozess auf verschiedene Arten die bindenden Basen gemessen, indem vorher z.B. fluoreszierende Marker an diesen Basen angebracht wurden, die wiederum gemessen werden.

Der größte Unterschied der NGS ist, dass die Sequenzen deutlich kürzer sind. Dies kann allerdings durch viele kurze Sequenzen, die mit viel Rechenleistung zusammengefügt werden, kompensiert werden [90]. Die neuen Technologien haben weiterhin teilweise deutlich höhere Basenfehlerraten.

Es folgt ein kurzer Abriss der Entwicklung der NGS.

Die Sequenzierer von Life Sciences waren die Ersten, die mit sehr hohen Durchsatzraten den Markt revolutionierten, auch wenn die Basenfehlerrate hoch und die Readlänge extrem niedrig waren. Kurz darauf kamen Sequenzierer von Solexa und Applied Biosystems auf den Markt, wobei Solexa durch die Sequenziertechnik keine Basenfolge ausgab, sondern die Reads als Transitionsfolge im Colorspace lieferte. Die anfänglichen Solexa Reads hatten eine Länge von 30bp wohingegen 454

Sequenzierer von Roche schon bei 100bp lagen. Aktuell schaffen die Sequenzierer von 454 bereits bis zu 1000bp lange Reads. [58, 89]

Ebenfalls neu im Zuge des NGS war, dass es möglich war, durch einen Trick zwei Reads mit einem größeren Abstand (sog. Paired Reads) zu sequenzieren. Hierbei werden im Wesentlichen von einer beispielsweise 2000bp langen Sequenz die Enden, z.B. der Länge 250bp, sequenziert, indem an beiden Enden eine Markierungssequenz angebracht wird, von der aus die Sequenzierung startet. Diese Erweiterung ist besonders deshalb sehr hilfreich, da repetitive Regionen durch diese Distanzinformation einfacher bestimmt werden können und bei der Assemblierung längere Contigs erreicht werden können.

Weiterhin liefern die Sequenzierer mittlerweile neben der Basenfolge eine Folge von Qualitätswerten (sog. Qualityscores), die angeben, mit welcher Wahrscheinlichkeit die entsprechende Base tatsächlich korrekt bestimmt wurde.

Parallel wurden in den letzten Jahren neue Verfahren entwickelt, die u.a. mit Hilfe von Feldeffektransistoren und der pH-Wert-Bestimmung die Basen ermitteln. Dieses Verfahren ist vielversprechend, da die optische Bestimmung voraussichtlich teurer als die elektrische Bestimmung ist. [58]

2.3.1 Sequenzierte Genome

Das Humangenom ist aus Sicht der Informatik noch nicht vollständig sequenziert. Es fehlen immer noch kleinere Abschnitte, die nicht bestimmt sind. Aus diesem Grund werden regelmäßig neue Versionen des Humangenoms veröffentlicht. Diese Sequenzen sind frei verfügbar zum Download. Die aktuellste Version ist HG19 und wird vom UCSC zur Verfügung gestellt [25].

Es sei angemerkt, dass das Humangenom nicht aus einem Strang besteht, da es insgesamt 23 Chromosomen (Sequenzen) hat. Für die theoretischen Modelle dieser Arbeit ist es ausreichend, wenn man sich diese 23 Sequenzen direkt hintereinander geschrieben als eine Sequenz vorstellt.

2.4 Informatische Grundlagen

2.4.1 Allgemeine Definitionen

Wie im vorherigen Kapitel beschrieben, besteht die DNS aus den Nukleotiden `A`, `C`, `G`, `T` und kann als eine lange Kette betrachtet werden. Um nun eine digitale Repräsentation der DNS festzulegen, definiert man die DNS als eine Zeichenkette über dem Alphabet {`A`, `C`, `G`, `T`}. Die Länge der DNS, bzw. eines Genoms, sei im Folgenden mit G bezeichnet.

Entsprechend werden Reads als Sequenzen über dem selben Alphabet gebildet. Ihre Länge werde mit l bezeichnet, wobei die Mächtigkeit des Readsets mit n bezeichnet sei.

Das *Reverse Komplement* einer Sequenz ist die gegenüberliegende Sequenz in der Doppelhelix. Sie ist entsprechend der eigentlichen Sequenz rückwärts gelesen, wobei zusammengehörige Nukleotide `A` $\rightarrow$ `T`, `C` $\rightarrow$ `G` und `T` $\rightarrow$ `A`, `G` $\rightarrow$ `C` ausgetauscht sind. Da im Sequenzierungsprozess nicht bekannt ist, von welchem Strang eine Sequenz stammt, ist diese Transformation sehr oft notwendig.

2.4.2 k-mere

k-mere bezeichnen Zeichenketten der Länge k. Häufig werden sie im Kontext von Subsequenzen der Länge k einer Ursprungssequenz verwendet. Eine Sequenz der Länge l hat $l - k + 1$ k-mere, wie in Abbildung 2.4 zu sehen ist.

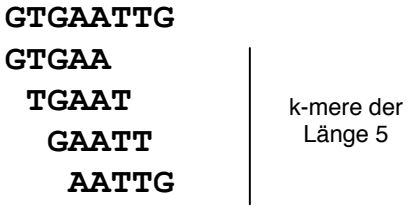

Abb. 2.4: k-mere
Häufig werden k-mere in Verbindung mit den Teilwörtern eines bestimmten Wortes verwendet.

2.4.3 Distanzfunktionen

Um den *Unterschied* oder Abstand zwischen zwei Wörtern zu bestimmen, verwendet man Distanzfunktionen. Eine Distanzfunktion gibt implizit auch an, wie die Wörter zueinander bestmöglich (nach der Distanzfunktion) ausgerichtet werden können. Wie Abbildung 2.5 zeigt, ist es nicht eindeutig, wie der Abstand zweier Wörter bestimmt werden muss und welche Operationen bei der Bestimmung zugelassen sind, also ob z.B. Leerstellen eingefügt werden dürfen.

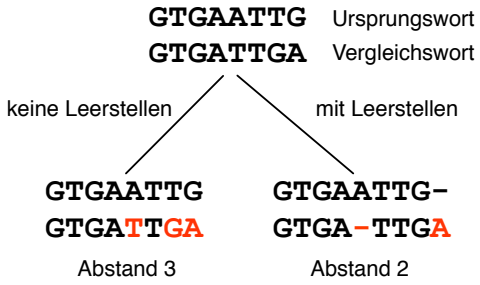

Abb. 2.5: Distanzfunktion
Verschiedene Distanzfunktionen ergeben verschiedene Distanzen und erlauben verschiedene Operationen.

Hammingdistanz

Die Hammingdistanz beschreibt die Anzahl der positionsweisen Buchstabenunterschiede zweier gleichlanger Sequenzen. Sie lässt keine Einfügungen oder Löschungen zu.

Die Hammingdistanz ist formal wie folgt zu beschreiben:

$HD : A^l \times A^l \to N_0^+$, mit $A = \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$

$HD(b_1 ... b_l, b'_1 ... b'_l) = \sum_{i=1}^{l} \delta(b_i, b'_i)$

mit

$$\delta(b_i, b'_i) := \begin{cases} 0, & \text{wenn } b_i = b'_i \\ 1, & \text{wenn } b_i \neq b'_i \end{cases}$$

In Abbildung 2.6 ist ein Beispiel zu sehen.

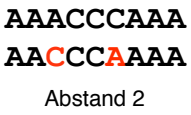

Abb. 2.6: Hammingdistanz Beispiel

Die Hammingdistanz ist sehr effizient für das Alphabet aus vier Buchstaben zu implementieren. Es ist möglich mit drei Operationen einen Bitvektor zu erzeugen, der nur an den unterschiedlichen Positionen eine Eins enthält. Weiterhin können mit 6 Operationen bei einem 64bit Prozessor 32 Nukleotide gleichzeitig berechnet werden. Die nötigen Operationen sind in einem Beispiel in Abbildung 2.7 zu sehen.

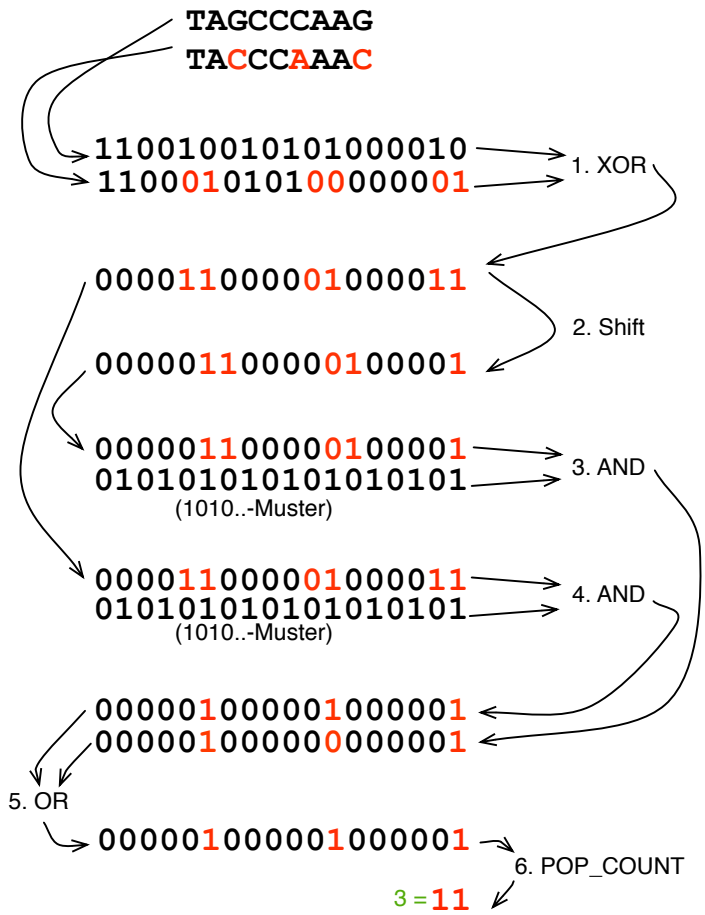

Abb. 2.7: Hammingdistanz mit sechs Operationen

Globales und lokales Alignment

Beim Alignment von Sequenzen wird zwischen lokalem und globalem Alignment unterschieden. Das globale Alignment wird verwendet, wenn die untersuchten Sequenzen die gleiche Länge haben und davon ausgegangen wird, dass die Sequenzen sehr ähnlich sind. Der Needleman-Wunsch-Algorithmus [52] ist der Algorithmus, der in $O(n^2)$ Zeit und Speicher das optimale globale Alignment bestimmen kann.

Das lokale Alignment ist ähnlich des globalen Alignments, findet aber das optimale Alignment zweier Teilsequenzen der ursprünglichen Eingabesequenzen nach einer Bewertungsfunktion. Der Smith-Waterman-Algorithmus [79] ist hier der Referenzalgorithmus, der ebenfalls in $O(n^2)$ Zeit und Speicher das optimale lokale Alignment findet.

Smith-Waterman

Der Smith-Waterman-Algorithmus ist ein Verfahren, um ein Maß für die Ähnlichkeit zweier Sequenzen zu bestimmen, wobei Einfügungen und Löschungen gestattet sind. Er arbeitet per dynamischer Programmierung, d.h., dass in einem Array iterativ aufbauend auf den vorherigen Ergebnissen das Alignment bestimmt wird, das mit der niedrigsten Anzahl von Einfügungen, Löschungen bzw. Ersetzungen erzeugt werden kann.

Die Definition des Algorithmus leitet sich aus der Berechnung einer $(l+1)\times(l+1)$ Matrix M ab, in der aus den Sequenzen $b = b_1 b_2 ... b_l, b' = b'_1 b'_2 .. b'_l$ die Unterschiede nach der Funktion δ' gewichtet werden.

Die Matrix M ist wiederum rekursiv definiert:

$SW : A^l \times A^l \rightarrow N_0^+$, mit $A = \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$

$SW := max(M(i,j))$ für alle $0 \leq i,j \leq l$.

$$M(i,j) := \begin{cases} M(i-1,j-1) + \delta'(b_i, b'_i) & \text{wenn Zeichen gleich oder ungleich} \\ M(i-1,j) + \delta'(b_i, -) & \text{wenn eine Löschung vorliegt} \\ M(i,j-1) + \delta'(-, b'_i) & \text{wenn eine Einfügung vorliegt} \\ 0 & \text{wenn leeres Suffix} \end{cases}$$

wobei $M(i,0) = 0$, $M(0,j) = 0$ und $\delta' : (A \cup \{-\}) \times (A \cup \{-\}) \to \{-1, 2\}$ mit

$$\delta'(b_i, b'_i) := \begin{cases} 2 & \text{wenn } b_i = b'_i \text{ (Identische Zeichen)} \\ -1 & \text{wenn } b_i = - \text{ oder } b'_i = - \text{ oder } b_i \neq b'_i \text{ (verschiedene Zeichen)} \end{cases}$$

wobei $\delta'(-,-)$ nach Definition von M nicht auftritt.

Abbildung 2.8 zeigt ein Beispiel, wie eine Matrix nach obiger Definition für zwei Sequenzen gefüllt aussieht.

Sequenz 1 **TAGC**
Sequenz 2 **TACA**

M =

	-	T	A	G	C
-	0	0	0	0	0
T	0	2	1	0	0
A	0	1	4	3	2
C	0	0	3	3	5
A	0	0	2	2	4

Abb. 2.8: Smith-Waterman-Matrix Beispiel

Aus der Matrix lässt sich per Backtracking vom maximalen Wert ein optimales Alignment berechnen. Dies ist in Abbildung 2.9 zu sehen.

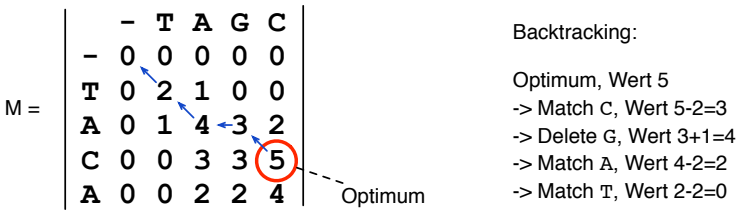

Abb. 2.9: Smith-Waterman-Matrix Backtracking Beispiel

2.4.4 Metriken

Es gibt viele Größen und Eigenschaften, die bei der Genomsequenzierung bestimmt werden. Es folgen einige Definitionen, die im Rahmen dieser Arbeit relevant sind.

Coverage

Das Maß der Coverage wird sehr häufig bei der *de novo* Sequenzierung und der Resequenzierung verwendet. Es beschreibt die durchschnittliche Anzahl von Reads, die eine beliebige Position innerhalb eines Genoms überdecken. Übliche Werte sind bei einer niedrigen Coverage 10. Bei einer Coverage von 100 würde man von einer hohen Coverage sprechen. Die Aussagekraft einer Coverage von C besagt nun in der Praxis, dass eine beliebige Stelle im Genom im Mittel C mal überdeckt wird und man hieraus Schlüsse auf die Aussagekraft eines bestimmten Nukleotids treffen kann. Beispielsweise ist es bei einem diploiden Organismus interessant, ob ein Nukleotid auf den verschiedenen Chromosomensätzen unterschiedlich ist. Bei einer niedrigen Coverage könnte eine Aussage hier nicht hinreichend sicher getroffen werden.

Abbildung 2.10 zeigt ein Beispiel in dem zu sehen ist, wie eine Coverage von 2 an der Position π entsteht.

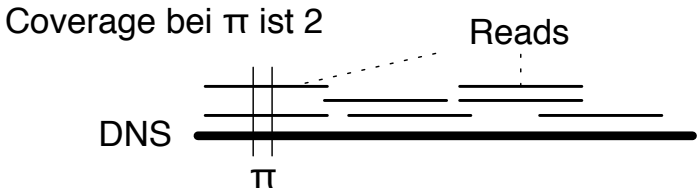

Abb. 2.10: Coverage Beispiel

In der Praxis sieht die Coverage hingegen auf den ersten Blick eher heterogen aus. Dies liegt an Regionen, die sehr oft im Genom vorkommen und daher bei der Sequenzierung überrepräsentiert werden. In Abbildung 2.11 ist ein Beispiel von Realdaten zu sehen. Neben den überrepräsentierten Werten ist auch der darunter liegende zufällige Prozess der Verteilung der Reads zu sehen.

Abb. 2.11: Coverage Realdaten Beispiel

Null-Coverage

Die Null-Coverage beschreibt die Verteilung der Längen der Bereiche eines Genoms, die nicht von Reads überdeckt sind. Sie ist aufschlussgebend dafür, wie gut die Verteilung der Reads, die vorher in einem chemischen Prozess erzeugt wurden, über das Genom war.

Abbildung 2.12 verdeutlicht einen Null-Coveragebereich.

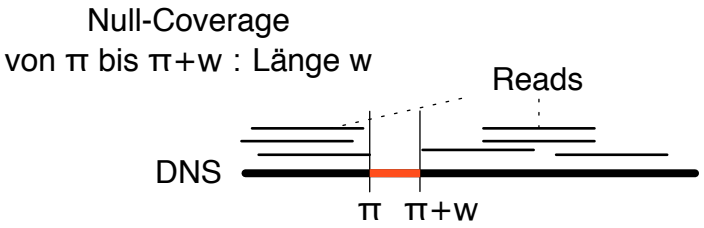

Abb. 2.12: Coverage Realdaten Beispiel

Contig und Contiglänge

Das Wort Contig kommt von dem Wort contiguous und bezeichnet eine Aufeinanderfolge von Nukleotiden, die aus mehreren überlappenden Reads entsteht. Ein Contig bezeichnet somit eine komplett überdeckte Sequenz, die vorne und hinten von nicht überdeckten Nukleotiden flankiert ist. Abbildung 2.13 illustriert verschiedene Contigs.

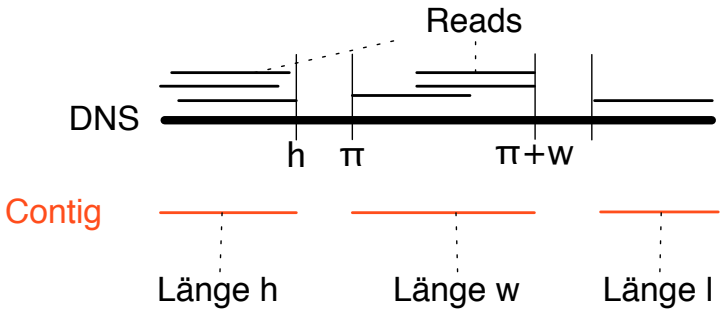

Abb. 2.13: Coverage Realdaten Beispiel

Die Contiglänge oder Contiglängenverteilung ist in der Praxis interessant für die Güte eines Sequenzierungsvorganges. Besonderes Interesse besteht darin, möglichst vollständige Überdeckungen zu erhalten. Bei der *de novo* Sequenzierung steht die Position eines Contigs nicht immer sicher fest, da z.B. eine fehlende Überdeckung ein Contig allein stehen lassen kann. Es kommt einem langen Contig eine sehr hohe Bedeutung zu, da ein langes Contig einfacher als viele kleine in einem Genom, mit Hilfe von z.B. ähnlichen Genomen, positioniert werden kann.

$N\alpha$ und $N50$

Anschließend an die soeben beschriebene Contiglänge bezeichnet der $N\alpha$-Wert die Länge des kleinsten Contigs, das genommen werden muss, wenn man $\alpha\%$ des Genoms mit den längsten Contigs überdecken möchte. Bedeutung hat dieser Wert daher, da es eine Aussage darüber ist, wie *zerstückelt* das Genom nach der Sequenzierung ist. Ein hoher $N\alpha$-Wert ist somit wünschenswert.

$N50$ ist entsprechend der Wert dafür, dass 50% des Genoms überdeckt werden sollen.

Alignmentqualität

Um ein Alignment bewerten zu können, ist es üblich, die Anzahl der alignten Reads zu verwenden. Darüber hinaus werden vor allem bei der Bewertung von Alignern, also Algorithmen die ein Alignment durchführen, die Raten für *Richtig-Positiv* (*true positive*, TP) und *Falsch-Positiv* (*false positive*, FP) herangezogen. Die TP Rate und FP Rate lassen sich allerdings nur bestimmen, wenn bekannt ist, wo ein Read seinen Ursprung hat. Dies ist bei Realdaten eines Sequenzierers nur schwer möglich, da in einem Genom viele sich wiederholende Regionen vorkommen und es somit für einen Read ggf. mehrere Positionen gibt, an die dieser *passt*. Aus diesem Grund verwendet man synthetische Daten, also generierte Daten mit bekannter Ursprungsposition, die zusammen mit den Reads abgespeichert werden. Diese synthetischen Daten werden dann bei der Bewertung eines Aligners verwendet.

Die TP-Rate ist nun der Quotient aus der Anzahl der Reads, die an die richtige Stelle alignt wurden und der Gesamtanzahl der Reads. Die FP Rate hingegen ist der Quotient aus der Anzahl der Reads, die an der falschen Stelle alignt wurden und der Gesamtzahl der Reads.

2.4.5 Datenstrukturen

Ein zentrales Element bei der Verarbeitung von Daten im Zusammenhang mit dem Sequenzieren sind die Datenstrukturen, in denen die Daten gespeichert werden. Dies ist besonders deshalb der Fall, da es sich bei der Sequenzierung um sehr viele Daten handelt und die Wahl der richtigen Datenstruktur im Fokus liegt. Im Folgenden wird auf Suchbäume und Hashtabellen eingegangen, da sie in dieser Arbeit verwendet werden.

Suchbäume

Ein Suchbaum ist eine baumartige Struktur, in der Daten gespeichert werden. Die Daten liegen hierbei in einer speziellen Sortierung im Speicher, so dass durch wenige Entscheidungen das gesuchte Element gefunden werden kann. Abbildung 2.14 zeigt ein Beispiel eines Binärbaumes. Sucht man ein Wort in dem Baum, geht man stellenweise von der Wurzel tiefer nach unten in den Baum und kommt schrittweise näher zum gesuchten Wort.

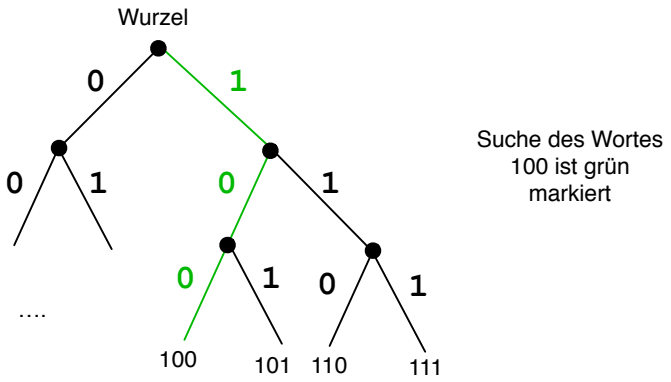

Abb. 2.14: Binärbaum

Speichert man beispielsweise alle Suffixe aller Wörter eines Textes in einem Baum, so kann man sehr schnell nach einem Wort oder Teilwort in einem Text suchen. Abbildung 2.15 zeigt ein Beispiel, in dem zu sehen ist, dass nach dem Einfügen des Wortes *Banane* sehr schnell geprüft werden kann, ob z.B. das Suffix *ane* in dem Wort enthalten ist und an welcher Position (Kreis) dieses vorkam.

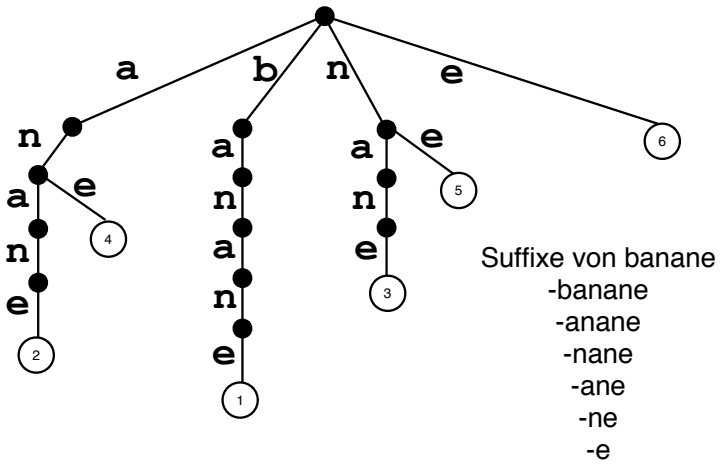

Abb. 2.15: Suffixbaum

Entsprechend lassen sich auch Genome oder Teile von Ihnen in einem Suffixbaum speichern. Sucht man dann eine Subsequenz, die z.B. aus einem anderen Genom stammt, ist die Suche sehr schnell, da nur buchstabenweise durch den Baum gegangen werden muss.

Hat man in einem Suffixbaum sehr lange Wörter, wird der Speicherverbrauch des Baumes sehr groß, da die Wörter buchstabenweise jeweils einen Knoten, und damit Speicher, benötigen. Bei langen Wörtern (in dem Beispiel das Wort Banane) gibt es aber häufig keine Abzweigungen im Baum. Um nun Speicher zu sparen, fasst man Knoten ohne Abzweigungen zusammen. Dies reduziert den Speicherbedarf enorm, dafür sind die Vergleiche, und damit die Suche im Baum, komplexer (s. Abbildung 2.16). Diese Speicherstruktur wird Patricia Trie genannt.

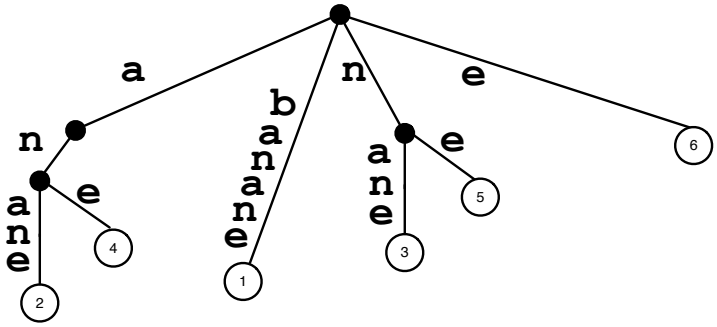

Abb. 2.16: Patricia Trie

2.4.6 Hashtabellen

Eine Hashtabelle ist eine Speicherstruktur, in der auf ein abzuspeicherndes oder gesuchtes Wort eine Funktion (Hash) angewendet wird, wodurch schnell in einer fixen Speicherstruktur nach diesem Wort gesucht werden kann. Eine Hashtabelle ist in der Regel so konzipiert, dass auf die Einträge *gemapped* werden kann, also direkt aus dem Wort der Speicherplatz des Wortes abzuleiten ist. Ein Beispiel ist in Abbildung 2.17 zu sehen.

Hashtabellen werden in dieser Arbeit verwendet, um in langen Texten Subsequenzen sehr schnell zu finden. Bei einer Hashtabelle wählt man in der Regel eine konstante Länge des Hashes. Der Vorteil bei einer Hashtabelle ist, dass in linearem Zeitaufwand in der Länge der Subsequenz bestimmt werden kann, ob, bzw. wo, diese in dem indizierten Text sind. Man bezeichnet die Hashtabelle als *Index* des Textes, der indiziert wurde.

Eine Suche in der Hashtabelle funktioniert wie folgt. Man wendet die Hashfunktion genau wie bei der Indizierung auf die gesuchte Subsequenz an. Aus dem Hashwert resultiert eine Position in der Tabelle, an der nachgeschlagen werden kann, ob ein Eintrag existiert, bzw. wo die zu dem Hashwert abgelegte Sequenz herkommt.

Da Hashfunktionen in der Regel nicht injektiv sind, kann es vorkommen, dass Hashwerte für verschiedenen Einträge aufeinander fallen. Hierbei sind dann mehrere Werte (z.B. Positionen innerhalb einer Sequenz) abzulegen bzw. ein Verhalten für Überschneidungen festzulegen.

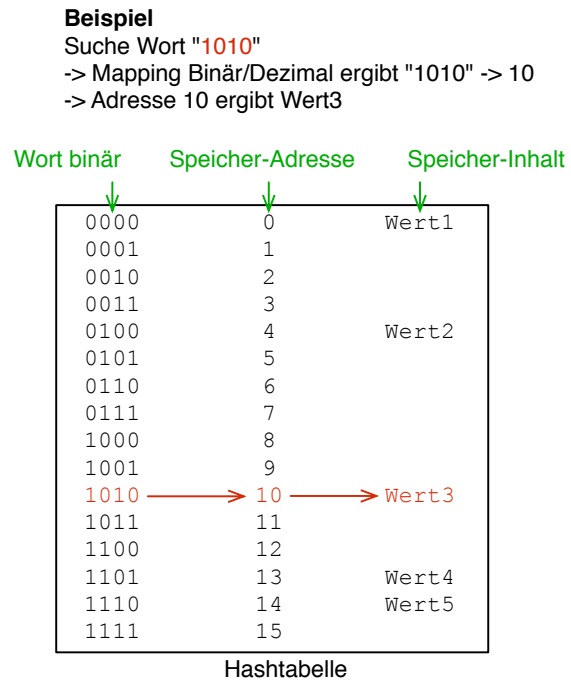

Abb. 2.17: Hashtabelle

Bei der Wahl der Hashfunktion es ist wichtig, dass die Funktion so gewählt wird, dass die Zielmenge möglichst wenige Elemente enthält, auf die mehrere Funktionswerte in Bezug auf die tatsächlichen Eingabedaten fallen.

2.4.7 FM-Index

Der Full-text Minute Index basiert auf der Burrows-Wheeler Transformation (BWT) [19], die zur Kompression von großen Texten entwickelt wurde und z.B. auch in bzip2 verwendet wird. Die Idee der BWT ist hierbei, dass durch eine reversible Transformation eines Textes Abschnitte innerhalb des Textes entstehen, die wiederholende Sequenzen enthalten. Diese wiederholenden Sequenzen können dann sehr gut komprimiert werden. Für den FM-Index hingegen bedeutet die Transformation (ohne die Kompression), dass ähnliche Sequenzen beieinander abgelegt sind und dadurch schnell die Vorkommen von ähnlichen Teilsequenzen und ihrer Positionen gefunden werden können.

Die Suche nach einer Zeichenkette im Index ist in logarithmischer Zeit möglich,

indem der Index mit Hilfe einer buchstabenweisen Schachtelung wie bei einer Suche in einer Baumstruktur durchgegangen wird. Existieren allerdings Unterschiede zwischen Suchwort und dem im Index abgelegten Wort, wird die Suche sehr rechen- und zeitaufwändig. Die Suche muss per Backtracking buchstabenweise rückwärts den oder die unterschiedlichen Buchstaben durch Substitution in alle übrigen Buchstaben des Alphabets durchgehen und wiederum eine Tiefensuche pro substituiertem Buchstaben durchführen. Dies benötigt exponentiellen Aufwand.

Um den exponentiellen Aufwand zu verdeutlichen, folgt ein sehr vereinfachtes Beispiel zur Konstruktion des aus der BWT abgeleiteten FM-Index. Es wird dabei nicht auf die Eigenschaften der BWT eingegangen. Sei $T = abbba\$$ das zu indexierende Wort, wobei \$ ein Wortendezeichen sei und lexikographisch vor allen anderen Zeichen stehe. Schreibt man nun alle Rotationen des Wortes auf erhält man R, wie in Abbildung 2.18 zu sehen. Nun werden die Zeilen sortiert und man erhält S. Der FM-Index besteht nun aus der ersten und letzen Spalte.

Rotation *R*	Sortierung *S*	FM-Index	*R1*	*R2*
abbba$	*$abbba*	*$a*	*$a*	*$a*
bbba$a	*a$abbb*	*ab*	*ab*	*ab*
bba$ab	*abbba$*	*a$*	*a$*	*a$*
ba$abb	*ba$abb*	*bb*	*bb*	*bb*
a$abbb	*bba$ab*	*bb*	*bb*	*bb*
$abbba	*bbba$a*	*ba*	*ba*	*ba*

Abb. 2.18: Beispiel FM-Index Erzeugung

Soll nun geprüft werden, ob das Wort T ein bestimmtes Infix $P = bba$ enthält, so wird die Eigenschaft ausgenutzt, dass durch die Rotation der erste und letzte Buchstabe (in der Rotation) aufeinander folgen. Es wird der Buchstabe b genommen und im Index nachgeschlagen. Es werden drei Zeilen gefunden, die mit b beginnen. Zwei der Zeilen enden mit einem b und eine mit einem a, das bedeutet, dass die Sequenz bb zweimal vorkommt ($R1$). Nun wird das zweite Zeichen b in der ersten Zeile gesucht und entsprechend vorgegangen. Es zeigt sich, dass es wieder drei Zeilen gibt, die mit b anfangen, aber nur eine, die mit a, dem dritten

Zeichen, enden ($R2$).

Nimmt man nun an, dass kein a gefunden worden wäre und ein Fehler in dem Infix P wäre, so müsste man nun Buchstabenweise zurück gehen und jeweils durch Substitutionen der Buchstaben prüfen, ob eine gültige Sequenz gefunden wurde.

2.5 Technische Grundlagen

2.5.1 Cloud Computing

Cloud Computing im eigentlichen Sinne bezeichnet das *Rechnen* (Computing) in einem unbekannten Umfeld (der Cloud). Seine Ursprünge hat es darin, dass man Ressourcen, die nicht permanent genutzt werden, besser ausnutzen wollte, bzw. umgekehrt, große Rechenkapazitäten kurzfristig zur Verfügung haben wollte. Hierbei ist die Idee, dass sich der Anwender (Programmierer) keine Gedanken darüber machen muss, wo dieser Rechner steht, bzw. wie sein Umfeld gerade beschaffen ist, die Infrastruktur und die Zugriffsmöglichkeiten also abstrahiert werden. So weiß der Anwender nicht, ob er gerade auf einem einfachen Single-Core Rechner arbeitet oder ihm ein kleines Zeitfenster eines Multi-Core Rechners zugewiesen wurde. Es wird dem Anwender bei den meisten kommerziellen Anbietern lediglich eine gewisse Anzahl an Prozessoren, eine fixe Arbeitsspeichergröße, bzw. ein fixer persistenter Speicher, garantiert.

2.5.2 MPI

MPI steht für Message Passing Interface und ist ein Standard, der für die Kommunikation in verteilten Systemen verwendet wird. In Form von Bibliotheken vereinfacht er die Programmierung, da die Kommunikation in Anwendungen auf verteilten Systemen in einem Netzwerk abstrahiert werden kann. Die Bibliotheken bieten dabei unter anderem Funktionalitäten, um Nachrichten im Netz an mehrere Prozesse zu senden oder verteilte Werte zu aggregieren und daraus z.B. die Summe zu bilden.

3 Modellierung von Short Read Sequenziererdaten

3.1 Einführung und Motivation

Aktuelle Sequenzierer erzeugen riesige Datenmengen, wobei Zeit und Geld immer noch stark begrenzende Faktoren sind. Ist ein DNS Molekül vorhanden, das sequenziert werden soll, ist die höchste Priorität, eine genügend große Menge an Reads zu produzieren, um die digitale Repräsentation der DNS wiederherstellen zu können. [90]

Auf der anderen Seite soll aber die Menge an zu produzierenden Reads minimiert werden, damit die Kosten für die Sequenzierung möglichst gering gehalten werden. Da auch die Rechenleistung beim Alignment linear [41], bei der de novo Assemblierung exponentiell [54], wächst, ist es sinnvoll, die Anzahl der Reads möglichst passend für den jeweiligen Anwendungsfall zu wählen. Ein weiterer Punkt ist, dass bei vielen Anwendungsfällen beschränkt viel genetisches Material vorhanden ist. Beispiele hierfür sind vor allem bei diagnostischen oder therapeutischen Verfahren zu finden.

In diesem Kapitel der Arbeit wird sich damit beschäftigt, die benötigte Datenmenge schon vor einem Sequenziervorgang zu ermitteln. In der Praxis kann dies zu einer Reduzierung der Kosten und des Aufwandes oder auch zur Erkenntnis führen, dass andere Analysemethoden oder Technologien als die ursprünglich beabsichtigte zur Bestimmung einer Genomsequenz oder einer Subsequenz nötig sind. Beispielsweise kann bei der vorherigen Analyse heraus kommen, dass es statistisch gar nicht möglich ist, die eine konkrete Sequenz in der gewünschten Überdeckung mit einem vorgegebenen Budget zu sequenzieren.

Darüber hinaus ist die Analyse und Modellierung von Short Read Sequenziererdaten für die Weiterentwicklung der Sequenziertechnologie essentiell. Ist das theoretische Modell bekannt, können die Hersteller der Sequenzierer bei der Entwicklung neuer oder Verbesserung bestehender Sequenzierverfahren prüfen, ob die erzeugten Daten mit dem theoretischen Modell übereinstimmen und ggf. systembedingte Fehler finden.

Die in Abschnitt 3.3 dargestellten Formeln wurden entwickelt, um theoretische Schranken eines Alignments von Short Reads für die folgenden Eigenschaften annähern zu können: Verteilung der Coverage, Verteilung der Null-Coverage, Ver-

teilung der Contig-Längen sowie Abschätzungen für $N\alpha$ (z.B. $N50$). Definitionen dieser Eigenschaften finden sich im Kapitel Grundlagen (Kapitel 2.4.1).

Auf Grund der Größe von Genomen höherer Lebewesen, wie des Genoms des Menschen (Genomlänge ca. $3 * 10^9$), sind die *informationstechnischen Probleme* in einem sehr großen Eingaberaum zu lösen. Man bewegt sich des Weiteren in Bereichen des Gesetzes der Großen Zahlen, also in Bereichen, in denen sich die relative Häufigkeit eines Zufallsergebnisses seiner Wahrscheinlichkeit annähert.

Nach einer Aufführung der in diesem Bereich veröffentlichten Arbeiten folgen in Kapitel 3.3 eigene Arbeiten, deren erste Ergebnisse in [71] und [72] vorgestellt und gesammelt in [73] veröffentlicht wurden.

3.2 Bestehende Modelle

Seit der Entwicklung der Sequenzierung mit Hilfe der Sanger-Methode (siehe Kapitel 2.3) wird sich damit beschäftigt, die theoretischen Grundlagen zu schaffen, die diesen Prozess modellieren. Anfangs wurden nur kurze Moleküle mit der Sanger-Methode in der Größenordnung der Readlänge analysiert. Da diese Länge aber um ein vielfaches kleiner war als die Länge von ganzen DNS Molekülen, wurde das Verfahren weiterentwickelt und die sogenannte *Shotgun* Sequenzierung [3, 11, 66] erfunden. Dieses Verfahren ermöglicht, vor allem durch seine deutlich niedrigeren Kosten, dass auch längere Moleküle bestimmt werden können, indem viele überlappende Teilsequenzen gemessen werden. [90]

Bei der Shotgun Sequenzierung wird ein Sequenzstück vervielfacht und zufällige Teilstücke dieser Sequenz werden sequenziert, da die Sequenzierung kleiner Sequenzen um ein vielfaches einfacher und damit kostengünstiger ist. Aus Überlappungen zwischen den Reads wird dann die ursprüngliche Sequenz wiederhergestellt. Dieses Verfahren wurde soweit weiterentwickelt, dass ganze Genome per *Whole Genome Shotgun* (WGS) Sequenzierung assembliert werden können. WGS ist mittlerweile die Basis der meisten größeren Genomsequenzierungsprojekte [88]. Der Prozess des WGS ist ein zufälliges Überdeckungsproblem (siehe Seite 13) der Größe eines Genoms. Wissenschaftler haben hierzu theoretische Modelle des Prozesses zur Vorhersage als auch zur Analyse (z.B. [21]) entwickelt. Hierbei wird ein allgemeines Modell bzw. die Problemstellung so definiert, dass gewisse Qualitätsmetriken in Abhängigkeit zu Eingabeparametern approximiert werden müssen. Die Parameter des Modells enthalten die Anzahl der Reads n, die Länge der Reads l und die Länge der Sequenz G. Bei den Qualitätsmetriken sind vor allem die Anzahl der Basen einer Sequenz, die bei einer Sequenzierung von mindestens einem Read überdeckt werden γ, sowie die Anzahl der alleinstehenden, aufeinander folgenden Sequenzen (Contigs) κ von Interesse. γ schätzt hierbei die Anzahl der Basen ab, die durch eine Assemblierung von überlappenden Reads entschlüsselt werden können, wobei κ ein Maß für die Arbeit ist, die noch nötig ist, um das Sequenzierprojekt abzuschließen. Da die unterliegenden Abhängigkeiten dieser Metriken nicht linear sind [93], werden deterministische Methoden normalerweise nicht verwendet um diese Werte zu bestimmen. Stattdessen wer-

den κ [Contigs] und γ [Überdeckung] in einem probabilistischen Zusammenhang als unabhängige und gleichverteilte Zufallsvariablen modelliert. [90]

3.2.1 Klassische Sequenziermodelle

Einige weit verbreitete, ältere Modelle werden noch heute in der Forschung verwendet. Hier sind vor allem die Coverage Gleichung

$$E(G) = ne - \rho \tag{3.1}$$

und das Lander-Waterman Model für Lücken [37] zu nennen, die im Folgenden erläutert werden. [90]

$$E(C) \approx ne^{-c\sigma} \tag{3.2}$$

$E(X)$ ist hierbei der Erwartungswert einer Zufallsvariable X. [90]

Die erste Gleichung kann aus elementaren wahrscheinlickeitstheoretischen Zusammenhängen über die Überdeckung einer Base hergeleitet werden, wie z.B. gezeigt durch Clarke und Carbon 1976 in [8]. Sie wird auf Seite 31 näher erläutert. [90]

Beide Gleichungen können direkt verwendet werden, um individuelle Projekte zu evaluieren. Allerdings ist es nicht möglich, die unterliegenden Wahrscheinlichkeitsverteilungen aus ihnen abzuleiten. [90]

Coverage Gleichung

Erste Modelle für Sequenzierprozesse wurden schon 1976 verwendet. Diese dienten zur Abschätzung der nicht überdeckten Basen [8]. Die Modelle bedienen sich der elementaren Wahrscheinlichkeitstheorie. Bezeichnet man die Länge einer zu sequenzierenden Sequenz mit G und die Länge des Reads mit l, so gilt, dass ein Read eine Position der Sequenz mit der Wahrscheinlichkeit $\frac{l}{G}$ überdeckt, solange $l << G$ und die Reads über das Genom gleichverteilt sind (vgl. Annahmen,

Kapitel 3.3.2). Mit Hilfe der Binomialverteilung und aus ihr resultierenden Eigenschaften [18] kann nun gezeigt werden, dass eine beliebige Position der Sequenz von mindestens einem von n Reads mit einer Wahrscheinlichkeit

$$P_{yes} = 1 - (1 - \frac{l}{G})^n \tag{3.3}$$

überdeckt wird.

Ist $n >> 1$, kann man obige Gleichung durch die Exponentialfunktion approximieren und es gilt:

$$P_{yes} \approx 1 - exp(1 - n\frac{l}{G}) \tag{3.4}$$

Hierbei ist der Term $n\frac{l}{G}$ genau die durchschnittliche Coverage einer Base, da $n * l$ Basen auf G Positionen verteilt werden. Diese einfache Formel kann verwendet werden, um eine grobe Abschätzung der prozentualen Menge der Positionen einer Sequenz, die überdeckt sind, zu bestimmen, wenn angenommen wird, dass die Wahrscheinlichkeit für alle Positionen dieselbe ist. Anders formuliert: Der Erwartungswert der Überdeckung C' der Sequenz ist

$$E(C') \approx 1 - exp(1 - n\frac{l}{G}) \tag{3.5}$$

Lander-Waterman Modell

Eric Lander und Michael Waterman entwickelten 1988 ein Modell zur Charakterisierung der Eigenschaften von sogenannten *Islands* (Contigs) in Mapping-Projekten [37]. Auch wenn in ihrer Arbeit die Rede von Mapping ist, können die Ergebnisse analog für de novo Sequenzierungsprojekte verwendet werden, um die Durchführbarkeit zu ermitteln. Ansatzpunkt der Arbeit waren die Bereiche ohne Überdeckung. Es wurden in der Arbeit die Anzahl der Contigs, die Anzahl der Contigs bestehend aus genau j Reads sowie einige weitere Ergebnisse gezeigt. Die Formeln sind hierbei in Abhängigkeit zu der minimal notwendigen prozentualen Überlappung θ zwischen zwei Reads formuliert.

Als Auszug folgen nun die beiden genannten Ergebnisse:

(i) Die erwartete Anzahl an Contigs C ist

$$E(C) \approx ne^{-c\sigma} \tag{3.6}$$

(ii) Die erwartete Anzahl an Contigs C_j, bestehend aus genau j Reads ($j \geq 1$), ist

$$E(C_j) \approx ne^{-2c\sigma}(1 - e^{-c\sigma})^{j-1} \tag{3.7}$$

Wobei $c = l\frac{n}{G}$ und $\sigma = 1-\theta$ mit $\theta = \frac{T}{l}$. T ist hierbei die Anzahl der Basen, mit der zwei Reads mindestens überlappen müssen, damit diese Überlappung eindeutig identifiziert werden kann.

Weiterentwicklungen und aktuelle Forschungsergebnisse

1995 veröffentlichte Roach [62] Verbesserungen zu obigen Theorien, die es ermöglichen, die Modelle für Sequenzierprozesse zu verwenden, deren Ziel die vollständige Sequenzierung eines Genoms waren.

Wendl und Waterston veröffentlichten in 2002 die Verteilung der Lücken basierend auf Stevens Theorem [20, 81]. Siegel leitete die genaue Verteilung der Coverage für das allgemeine Geometrische Coverage Problem[1] in [76] her. Es ist allerdings berechnungstheoretisch zu komplex, um es für WGS Sequenzierungsprojekte anzuwenden. Beispielsweise ist die Anwendung für $n > 104$ für vergleichsweise kleine mikrobakterielle WGS Projekte (z.B. [21]) nicht möglich, da jeder Wert in der Verteilungsfunktion dazu führt, dass n^2 Terme addiert werden, die wiederum jeweils eine sehr hohe numerische Genauigkeit haben müssen, um die Produkte von drei Binomialkoeffizienten der Ordnung n sowie zwei extrem kleine Kernel-Wahrscheinlichkeiten zu lösen. [90]

[1] Geometrisches Coverage Problem, auch Geometic Set Cover: Es beschreibt das geometrische Pendant des Minimum Set Cover Problems. In $\mathbb{R}^2$ wäre dies das Problem welche Teilmenge einer Gesamtmenge von Kreisen oder Rechtecken eine gegebene Fläche überdeckt.

Die WGS Methode wird mittlerweile für groß angelegte Säugetier-Genom Sequenzierungsprojekte verwendet [9, 22, 86, 88]. Derartige Projekte sind etwa eintausendmal größer als die vorherig beschriebenen mikrobiellen Vorgänger. [90]

Wendl entwickelte 2006 ein Model der Coverage-Verteilung für WGS Sequenzierung. Ziel dieses Modells ist es, weitere Aspekte des Sequenzierungsprozesses zu erfassen [90].

Die Grundidee der Lander-Waterman Theorie wurde in mehreren Facetten für verschiedene Mapping Techniken durch Arbeiten von [4, 60, 96] weiterentwickelt.

Von Balzer et al. [5] wurde ein anderer Ansatz verfolgt. Hier wird versucht, die Verteilung von Reads durch Simulation nachzubilden. Im Fokus steht hier also die Beschreibung des Sequenzierungsprozesses.

3.3 Statistische Analyse und Modellierung von Sequenziererdaten

3.3.1 Definitionen

Die Länge einer DNS Sequenz wird mit G bezeichnet. Die Menge der Reads eines Sequenzierungsprojektes wird im Folgenden Readset genannt, wobei die Anzahl der Reads mit n bezeichnet wird. Ein Read hat die Länge l.

3.3.2 Annahmen

Für die folgenden Modelle wird die allgemein anerkannte und verwendete Annahme getroffen, dass die Positionen der Reads innerhalb des Genoms eines Sequenzierungsvorganges gleichverteilt sind [91]. Obwohl diese Annahme auf die meisten Sequenzierungstechniken nicht vollständig zutrifft, erlaubt diese Vereinfachung Modelle zu entwickeln, die zu handhabbaren Formeln führen, die wiederum sehr gute Grenzen für die Parameter eines Sequenzierungsprozesses ergeben. Des Weiteren ist die Sequenzierung eines Genoms ein individueller Prozess, der sich mit jeder Durchführung und jedem Genom unterscheidet. Es wird im Folgenden nicht auf strukturelle Eigenschaften, wie z.B. Repeat-Regionen oder den GC-Anteil im Genom, eingegangen, obwohl diese in der Praxis eine Herausforderung für die de novo Assemblierung darstellen. Diese Eigenschaften sind allerdings auch genomspezifisch, weshalb für sie keine Verallgemeinerungen möglich sind. [91]

3.3.3 Verteilung der Coverage

Es wurde das folgende Modell entwickelt, um die Wahrscheinlichkeit zu berechnen, dass an einer beliebigen Position k eine Coverage von y, bezeichnet mit $C(k, y)$, vorhanden ist. Dabei ist $C(k, y)$ die Wahrscheinlichkeit, dass y Reads die Position k überdecken. Um $C(k, y)$ zu berechnen ist es notwendig, die Wahrscheinlichkeiten der Kombinationen aufzusummieren, die eine Coverage y bilden können. Abbildung 3.1 zeigt die Konstruktion.

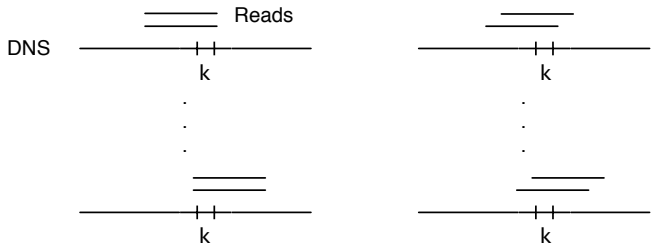

Abb. 3.1: Möglichkeiten der Anordnung von zwei Reads, um über einer Position k eine Coverage von Zwei zu erzeugen

Hierbei ist die Anzahl der Möglichkeiten der Anordnung von Reads genau gleich mit der Anzahl, die die Partitionsfunktion[2] aus der Zahlentheorie liefert, wobei die Funktion exponentiell wächst.

Es folgt die Situation für $y = 2$. Unten, in Formel (3.9), ist eine Vereinfachung zu sehen.

Es wurde die folgende Formel entwickelt, die die beschriebene Wahrscheinlichkeit modelliert, wobei $B(k|p, m)$ die Binomialverteilung ist. Die Erläuterung der Terme folgt nach der folgenden Formel.

$$\begin{aligned} C(k,2) = l * B(2|\frac{1}{G},n) * B(0|\frac{1}{G},n)^{l-1} \\ + \binom{l}{2} * B(1|\frac{1}{G},n)^2 * B(0|\frac{1}{G},n)^{l-2} \end{aligned} \tag{3.8}$$

Da die Reads die Länge l haben, gibt es l Möglichkeiten, die Position k mit zwei Reads zu überdecken, die an der selben Position starten. Dies geschieht genau mit der Wahrscheinlichkeit $B(2|\frac{1}{G}, n)$. Um nur genau diese zwei Reads an der Position zu haben und keine weiteren anderen, dürfen an den übrigen $l-1$ Positionen keine Reads sein: $B(0|\frac{1}{G}, n)^{l-1}$

[2]Die Partitionsfunktion einer positiven, ganzen Zahl bestimmt die Anzahl der Möglichkeiten, mit der diese Zahl in positive, ganze Summanden zerlegt werden kann. Ein Näherungswert des Funktionswerts ist, dass die Anzahl der Dezimalstellen der Partitionsfunktion einer Zahl, der Wurzel aus dieser Zahl entspricht.

Der zweite Term ist die Anzahl der Kombinationen, die zwei verschiedene Vorkommen von Reads an verschiedenen Positionen bilden können; es sind $\binom{l}{2}$ multipliziert mit der Wahrscheinlichkeit, dass sie auftreten $B(1|\frac{1}{G},n)^2$, multipliziert mit der Wahrscheinlichkeit, dass keine weiteren Reads an den übrigen $l-2$ Positionen auftreten $B(0|\frac{1}{G},n)^{l-2}$.

Diese Formel kann auf Grund der hohen Anzahl an zufälligen Ereignissen, hier das zufällige *Ziehen* der Readpositionen, vereinfacht werden. Da die Binomialverteilung $B(k|p,m)$ gegen die Poissonverteilung $P_\lambda(k)$ für kleine p und große m konvergiert, gilt: $B(k|p,m) \approx P_\lambda(k) = \frac{\lambda^k}{k!}e^{-\lambda}$. Hierbei ist $\lambda = \frac{m}{G}$.

Die Konvergenz der Verteilungen kann wie folgt interpretiert werden. Die Readmenge, bestehend aus $M = n * l$ Nukleotiden, kann als unabhängige Menge angesehen werden, die das Genom überdeckt. Sie kann somit behandelt werden, als wenn diese Nukleotide unabhängig über das Genom verteilt wären, ohne die Bedingung, dass sie ursprünglich als Sequenz von l Nukleotiden sequenziert wurden. Hierdurch kann das komplexe Wahrscheinlichkeitsmodell, bestehend aus exponentiell vielen Kombinationen, die das Genom überdecken können, wie folgt vereinfacht werden. Der Beweis folgt direkt im Anschluss.

$$C(k,y) < B(y|\frac{1}{G}, n*l), \tag{3.9}$$

für $0 < y \leq l$

Für eine praktische Anwendung bedeutet dies, dass $C(k,y)$ die genomweite Coverageverteilung angibt, wobei y die Coverage ist. Die Formel (3.9) kann dann die prozentuale Anzahl der Nukleotide liefern, die mit y Coverage überdeckt sind.

Der nächste Abschnitt zeigt die Konvergenz.

Untergrenze der Wahrscheinlichkeitsfunktion

Theorem: Die Verteilung der Coverage nach dem Modell bestehend aus zusammenhängenden Reads, die nach der Binomialverteilung $C(k,y)$ verteilt sind, kann

als untere Schranke das Modell aus unabhängigen Nukleotiden, die nach der Binomialverteilung verteilt sind, abgeschätzt werden: $C(k,y) \geq B(y|\frac{1}{G}, n*l)$.

Beweis: Der Beweis wird per Induktion geführt.

Da nach dem *Poissonschen Grenzwertsatz* gilt

$$B(k|p,n) \approx \frac{\lambda^k}{k!}e^{-\lambda} \tag{3.10}$$

für $n \to \infty$ und damit Folgendes gilt:

$$B(y|\frac{1}{G}, n*l) \approx \frac{(\lambda*l)^y}{y!} * e^{-l\lambda} \tag{3.11}$$

mit $\lambda = \frac{n}{G}$, wird im Folgenden gezeigt, dass die Behauptung gilt

$$C(k,y) \geq \frac{(\lambda*l)^y}{y!} * e^{-l\lambda} \tag{3.12}$$

Induktionsanfang: Sei $y = 0$, dann gilt.

$$\begin{aligned} C(k,0) &= B(0|\frac{1}{G}, n)^l && (3.13)\\ &\overset{(3.10)}{=} e^{-\lambda*l} && (3.14)\\ &= \frac{(l*\lambda)^0}{0!}e^{-\lambda*l} && (3.15)\\ &\overset{(3.11)}{=} B(0|\frac{1}{G}, n*l) && (3.16) \end{aligned}$$

Induktionsannahme: Es gelte die Behauptung für ein $0 \leq y < l$.

Induktionsschritt: Es ist zu zeigen: Wenn

$$C(k,y) \geq \frac{(\lambda * l)^{(y)}}{(y)!} * e^{-l\lambda} \tag{3.17}$$

dann

$$C(k,y+1) \geq \frac{(\lambda * l)^{(y+1)}}{(y+1)!} * e^{-l\lambda} \tag{3.18}$$

Gezeigt wird dies durch folgende Konstruktion. Wenn an einer Position k die Wahrscheinlichkeit für eine Coverage $C(k,y)$ existiert, so ist an der selben Position die Coverage $C(k,y+1)$, also um eins höher, wenn dort ein weiterer Read zu finden ist. Dies geschieht mit Wahrscheinlichkeit $l * B(1|\frac{1}{G},n) * B(0|\frac{1}{G},n)^{-1} = l * \lambda$, da über der Position k oder einer der $l-1$ Vorherigen ein Read liegen muss. Die Wahrscheinlichkeit ist also $C(k,y+1) = C(k,y) * (l * \lambda)$.

$$\begin{aligned}
C(k,y+1) &= C(k,y) * (l * \lambda) && (3.19)\\
&\overset{IV or.}{\geq} \frac{(\lambda * l)^{(y)}}{(y)!} * e^{-l\lambda} * (l * \lambda) && (3.20)\\
&= \frac{(\lambda * l)^{(y+1)}}{(y)!} * e^{-l\lambda} && (3.21)\\
&\overset{y\geq 0}{\geq} \frac{(\lambda * l)^{(y)}}{(y)!} * e^{-l\lambda} * \frac{1}{y+1} && (3.22)\\
&= \frac{(\lambda * l)^{(y+1)}}{(y+1)!} * e^{-l\lambda} && (3.23)
\end{aligned}$$

Die Konstruktion lässt sich auch anschaulich darlegen. Da, wie bereits erwähnt, die Anzahl der Terme sehr stark wächst, folgt ein Beispiel für $y = 2$:

$$
\begin{aligned}
C(k,2) &= l * B(2|\frac{1}{G},n) * B(0|\frac{1}{G},n)^{l-1} \\
&+ \binom{l}{2} * B(1|\frac{1}{G},n)^2 * B(0|\frac{1}{G},n)^{l-2} && (3.24)\\
&\approx l * \frac{\lambda^2}{2!} * e^{-\lambda} * \frac{\lambda^0}{0!} * e^{-\lambda*(l-1)} \\
&+ \binom{l}{2} * (\frac{\lambda^1}{1!})^2 * e^{-\lambda^2} * \frac{\lambda^0}{0!} * e^{-\lambda*l+\lambda} && (3.25)\\
&= l * \frac{\lambda^2}{2} * e^{-\lambda} * e^{-\lambda*(l-1)} \\
&+ \binom{l}{2} * \lambda^2 * e^{-2\lambda} * e^{-\lambda*(l-2)} && (3.26)\\
&= l * \frac{\lambda^2}{2} * e^{-l\lambda} + \binom{l}{2} * \lambda^2 * e^{-l\lambda} && (3.27)\\
&= \frac{(\lambda*l)^2}{2} * e^{-l\lambda} && (3.28)\\
&= B(2|\frac{1}{G}, n*l) && (3.29)
\end{aligned}
$$

3.3.4 Verteilung der Null-Coverage

Null-Coverage bezeichnet Bereiche, die von keinem Read überdeckt sind, wobei diese Bereiche konsekutiv nicht überdeckt sein dürfen. Ein Bereich mit Null-Coverage der Länge 5 hat also keine Reads an 5 aufeinander folgenden Basen im Genom.

Die Wahrscheinlichkeit $P_{no}(k)$, dass kein Read an einer beliebigen Position k startet, ist:

$$P_{no}(k) = \binom{n}{0} * (\frac{1}{G})^0 * (1-\frac{1}{G})^n \quad (3.30)$$

welches der Binomialverteilung $B(0|\frac{1}{G},n)$ entspricht.

Hieraus folgt, dass die Wahrscheinlichkeit einer Null-Coverage-Region der Länge

w, startend an Position k, wie folgt ist:

$$P_0(k,w) = (1 - P_{no}(k+w)) * \prod_{i=1}^{l-1+w} P_{no}(k+i) *(1 - P_{no}(k-l)) \quad (3.31)$$

Dies entspricht der Wahrscheinlichkeit, dass für die Position k gilt: An Position $k+w$ starten ein oder mehrere Reads, ein oder mehrere Reads enden an Position $k-1$ und keine Reads starten an Position k und den folgenden $w-1$ Positionen. Abbildung 3.2 zeigt diese Struktur.

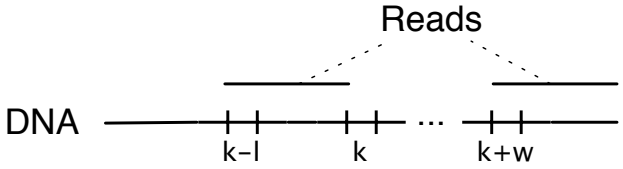

Abb. 3.2: Konstruktion eines Bereichs ohne Überdeckung eines Reads

Diese Formel kann vereinfacht werden, da $P_{no}(k) = P_{no}(\tau), \forall k, \tau$

$$P_0(k,w) = (1 - P_{no}(k))^2 * P_{no}(k)^{l-1+w} \quad (3.32)$$

Anmerkung: Multipliziert man die Formel (3.32) mit G, erhält man die absolute Anzahl von Null-Coverage Regionen der Länge w.

3.3.5 Verteilung der Contig-Längen

Das folgende Modell kann verwendet werden, um die Contiglängenverteilung $P_{len}(x)$ zu erhalten. Dies ist die Verteilung der Längen von Contigs, wenn n Reads sequenziert werden.

Ein Contig der Länge x kann konstruiert werden, indem ein Contig κ mit einer Länge kleiner als x durch einen Read der Länge l verlängert wird. Dies kann für alle κ der Länge $|\kappa|$ mit $x > |\kappa| > x - l$ geschehen, da mindestens eine

Überlappung von einer Base stattfinden muss. Die Überlappung um eine Base ist hierbei ein theoretisches Minimum um die oberen theoretischen Grenzen zu berechnen. Siehe Abbildung 3.3 für die schematische Struktur.

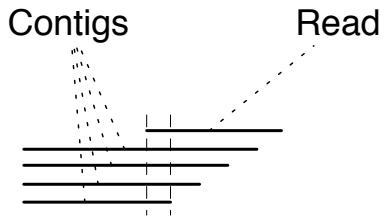

Abb. 3.3: Konstruktion der Verlängerung eine Contigs durch einen Read und dessen minimale Überlappung

In der Praxis, z.B. für de novo Assembly, muss die Überlappung eine gewisse Länge aufweisen, um eine Einzigartigkeit der Sequenz sicherzustellen, wie z.B. in [37].

Für den Sonderfall $x < l$ gilt, dass es keine Contigs der Länge $l - 1$ gibt, da ein Contig mindestens die Länge l haben muss. Daher sei hierfür die Wahrscheinlichkeit null: $P_{len}(x) = 0$ für $x < l$.

Um die Contiglängenverteilung zu berechnen, wird als erstes die Binomialverteilung $B(k|p, m)$ verwendet, um die Wahrscheinlichkeit zu errechnen, dass kein Read an einer beliebigen Position k startet:

$$P_{no}(k) = B(0|\frac{1}{G}, n) \tag{3.33}$$

wobei G die Länge des Genoms ist und n die Anzahl der Reads.

Als Nächstes wird die Wahrscheinlichkeit benötigt, mit der ein oder mehrere Reads an einer zufälligen Position k starten. Dies ist die Gegenwahrscheinlichkeit zu obiger Formel:

$$P_{yes} = 1 - P_{no} \tag{3.34}$$

Hieraus kann nun die Contiglängenverteilung berechnet werden:

$$P_{len}(x) = \sum_{i=1}^{l-1} P_{len}(x-i) * (P_{no})^{-1} \tag{3.35}$$

$$*P_{yes} * (P_{no})^{i} \tag{3.36}$$

$$= \sum_{i=1}^{l-1} P_{len}(x-i) * P_{yes} * (P_{no})^{i-1} \tag{3.37}$$

Die Formel ist wie folgt aufgebaut. Es wird die Wahrscheinlichkeit eines Contigs aus dem obig beschriebenen Längenbereich genommen, dann wird die Bedingung keines weiteren Contigs an der gesuchten Stelle entfernt und dann ein Read überlappend hinzugefügt, wobei die Bedingung hinzugefügt werden muss, dass keine weiteren Reads den gerade hinzugefügten Read an den i Positionen überlappen dürfen. Es sei angemerkt, dass die Bedingung, keinen Read am Ende zu haben, bereits in $P_{len}(x-i)$ enthalten war. Dies kann also für das neue Ende des Contigs als Bedingung verwendet werden, da $P_{no}(a) = P_{no}(b), \forall a, b$

Weiterhin gilt $P_{len}(x) = 0$, $\forall x < l$ und $P_{len}(l) = P_{yes} * (P_{no})^{2l}$, da ein oder mehrere Reads genau an einer Position starten müssen und keine weiteren Reads diesen Read überlappen dürfen.

Für den Fall, dass wenige Reads in der Menge sind, gilt: $x < n * l : P_{len} = 0$.

Um die absolute Anzahl an Contigs einer bestimmten Länge zu berechnen, muss die Formel analog zu den vorherigen Formeln mit G multipliziert werden: $G * P_{len}(x)$. Um für ein spezielles x die Anzahl der Contigs zu berechnen, müssen vorher alle Contiglängen der Länge $x' < x$ berechnet werden.

Es sein angemerkt, dass Contigs in diesem Kontext maximale Regionen sind, die auf einer theoretischen Referenz aus überlappenden Reads bestehen.

3.3.6 Abschätzung von $N\alpha$

Um $N\alpha$, z.B. $N50$, zu berechnen ist es notwendig, dass die Anzahl der Contigs einer bestimmten Länge bekannt ist. Dies sei im Folgenden mit $C_{len}(u)$ bezeichnet,

wobei u die Länge eines Contigs ist. $C_{len}(u)$ kann nun berechnet werden, indem $P_{len}(u)$ mit G multipliziert wird:

$$C_{len}(u) = G * P_{len}(u) \tag{3.38}$$

Nach Definition ist $N\alpha$ das größte y, für das gilt:

$$G * \frac{\alpha}{100} \leq \sum_{i=y}^{G} C_{len}(i) \tag{3.39}$$

3.3.7 Vereinfachung von $N\alpha$

Die Berechnung von (3.39) ist sehr aufwändig, da in der Regel gilt $0 < y << G$ und somit sehr viele Terme addiert werden müssen. Es ist aus Gründen der Berechnungskomplexität vorteilhaft, die Definition von $N\alpha$ umzukehren. $N\alpha$ kann definiert werden als das kleinste y für das gilt:

$$G * \frac{\alpha}{100} > \quad \sum_{i=1}^{y} C_{len}(i) + \sum_{k=1}^{G} P_{no}(k)^l \tag{3.40}$$

$$\leftrightarrow \frac{\alpha}{100} > \quad \sum_{i=1}^{y} \frac{C_{len}(i)}{G} + P_{no}(k)^l, k \in [G] \tag{3.41}$$

Hierbei entspricht der letzte Term der Anzahl der Positionen mit einer Coverage von Null, also Positionen, bei denen an l aufeinander folgenden Stellen kein Read startet.

Auf Grund der rekursiven Definition von C_{len} ist nun $N\alpha$ berechenbar. Die vereinfachte Form von $N\alpha$ hat $O(y)$ Terme an Stelle von $O(G)$ Termen in der normalen Form, da normalerweise $y << G$.

3.3.8 Abschätzung von $N50$

$N50$ kann mit Formel 3.41 berechnet werden. Hierbei kann ähnlich wie bei der Berechnung von $C_{len}(x)$ vorgegangen werden, die am Ende des Abschnitts zur Contiglängenverteilung beschrieben ist.

3.3.9 Abschätzung der Contig-Anzahl

Unter Verwendung von Formel 3.32 ist es möglich, die erwartete Anzahl an Contigs zu berechnen. Weiterhin wird die Anzahl von Null-Coverage Regionen $C_0(z)$ benötigt, wobei z die Länge der Region bezeichnet:

$$C_0(z) = \sum_{i=1}^{G-z} P_0(i,z) \approx \sum_{i=1}^{G} P_0(i,z) = G * P_0(1,z) \tag{3.42}$$

Die Formel ist folgendermaßen zu interpretieren. An jeder Position besteht die Wahrscheinlichkeit $P_0(k,z)$, dass dort eine Null-Coverage Region der Länge z existiert.

Hieraus kann nun die Anzahl an Contigs CC abgeleitet werden, da jede Null-Coverage Region genau zwei adjazente Contigs teilt. Die Summe der Unterbrechungen entspricht der Gesamtanzahl an Contigs minus Eins:

$$CC = 1 + \sum_{i=1}^{G} C_0(i) \tag{3.43}$$

Wie in Formel (3.32) beschrieben, ist es möglich, die absoluten Werte für spezifische G, n, l zu berechnen. Analog kann dies auch für CC durchgeführt werden.

3.3.10 Anwendung und Grenzen der Modelle

Die Verteilung der Reads über das Genom ist nicht, wie in den einführenden Annahmen beschrieben, vollständig gleichverteilt. Es existieren innerhalb einer Sequenziertechnologie und eines Genoms gewisse Verzerrungen in Bezug auf die Positionsverteilung der Reads. Analysen im Zusammenhang mit dieser Arbeit haben gezeigt, dass eine höhere Anzahl an Reads im Verhältnis eine kleinere Menge an neuen Informationen liefert. Diese Korrelation bedeutet z.B., dass bei der Sequenzierung des selben Genoms zum zweiten Mal nicht doppelt so viel Information enthalten ist, als statistisch in Abhängigkeit zur Readverteilung vorkommen müsste. Diese Feststellung wird mit Effizienz des Readsets ρ bezeichnet.

In praktischen Versuchen mit dem HG18 zeigte sich, dass $\rho \approx 0.40$, d.h. dass für die Berechnungen der Formeln mit gemessenen Daten im Vergleich zu zufällig generierten Daten die Anzahl der gemessenen Reads n mit einem Faktor von 0.40 multipliziert werden muss, wenn man die beobachteten Verteilungen mit den tatsächlichen vergleicht.

Um die obigen Formeln für ein beliebiges anderes Sequenzierungsprojekt anzuwenden, wird folgendes Vorgehen vorgeschlagen. Die Größe des Genoms G sollte entweder bekannt sein oder anhand von ähnlichen Genomen abgeschätzt werden. Der Parameter l, der die Readlänge angibt, kann von einem Sequenzierungslabor ermittelt werden. Schließlich ist es nötig, die Anzahl der Reads n zu ermitteln. Hierzu kann man z.B. die maximale Anzahl der Reads, die ein Sequenzierungslabor für ein festes Budget erzeugt, erfragen. Alternativ kann man n variieren lassen und das für das betrachtete Szenario beste n anhand der Ergebnisse der Formeln ermitteln, indem man z.B. $N50$ als unteren Grenzwert festlegt. Hierzu bietet sich die Verwendung einer Computeralgebrasoftware an.

In Bezug auf die Abschätzung der Effizienz der Reads wird folgendes Vorgehen vorgeschlagen. Es wird ein einfaches Alignment von n Reads, z.B. $n = 1.000.000$, gegen eine bekannte Referenz der verwendeten Sequenziertechnologie durchgeführt. Das Ergebnis wird mit den theoretischen Werten verglichen, wobei $\rho \in [0,3\ ,\ 0,5]$ variiert wird. Dies kann umgesetzt werden, indem die obigen Formeln mit den Werten G, n_ρ, l von den Reads berechnet werden, z.B. $n_{0,5} = 1.000.000 * 0,5 = 500.000$. Hieraus würde ρ angepasst, bis der gemessene Wert, z.B. $N50$, dem theoretischen Wert entspricht.

Strukturelle Eigenschaften, z.B. viele Repeat-Regionen oder ein hoher GC-Anteil einiger Genome, können die Ergebnisse beeinflussen. Eine Konsequenz ist z.B. die Erhöhung der Anzahl der Contigs und damit eine Verringerung des $N50$ Wertes.

3.4 Ergebnisse aus Messreihen

3.4.1 Einführung

Es wurden Reads des *1000 Genome Project* (www.1000genomes.org, Individuum NA12878, sequenziert mit dem Illumina Genome Analyser) genommen und mit den theoretischen Werten der entwickelten Formeln aus Kapitel 3.3 verglichen. Entsprechend ergaben sich theoretische und gemessene Werte für die Verteilung der Coverage, die Verteilung der Null-Coverage, die Verteilung der Contiglängen, des $N50$-Wertes sowie der Contig-Anzahl, die jeweils gegenübergestellt wurden.

Um die gemessenen Daten mit den theoretischen zu vergleichen, wurden etwa $3.5 * 10^6$ Reads der Länge 47bp aus dem angegebenen Projekt genommen und mit dem Alignmentalgorithmus Bowtie[39] (mode -m 1, genau eine Ergebnis) an die Referenz HG18 des Humangenoms alignt. Auf Grund der großen Datenmenge wurde sich dabei auf Chromosom 21 beschränkt.

Die Ergebnisse der Vergleiche folgen in den anschließenden Unterabschnitten.

3.4.2 Verteilung der Coverage

Die Ergebnisse der theoretischen und der gemessenen Werte korrelieren und geben gute Abschätzungen. Abbildung 3.4 zeigt den prozentualen Anteil von Basenpaaren zu den dazugehörigen Coverage-Werten. Man kann erkennen, dass die entwickelte Formel den exponentiellen Abfall der gemessenen Daten gut modelliert. Unterschiede zwischen theoretischem Modell und gemessenen Daten sind im Bereich der sehr niedrigen und teilweise bei sehr hohen Coverage-Werten zu erkennen. Hier ist zu erkennen, dass die niedrige Coverage im theoretischen Modell niedriger als gemessen vorhergesagt wird. Entsprechend geht dies zu Lasten der Vorhersage von hoher Coverage mit umgekehrtem Effekt, allerdings mit deutlich niedrigeren relativen Werten, aber über einen breiteren Bereich.

Es ist zu erkennen, dass es notwendig ist, mehr Reads zu haben, als der Wert *durchschnittliche Coverage*, der oft verwendet wird, suggeriert.

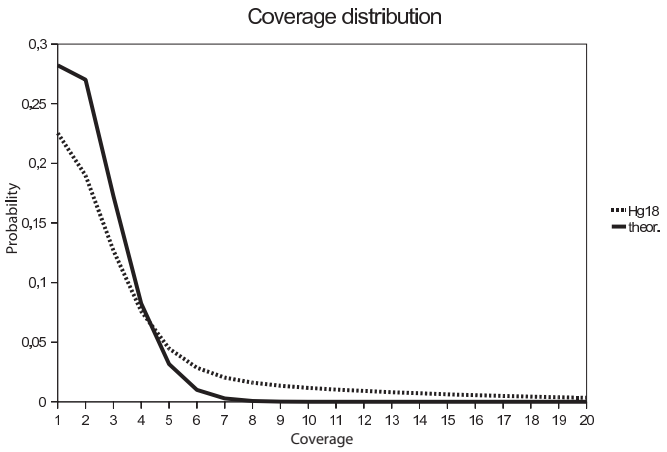

Abb. 3.4: Coverage-Verteilung. Die Verteilung der Coverage einer festen Anzahl von Reads des Humangenoms im Vergleich zur theoretischen Verteilung nach der entwickelten Formel. Der Unterschied liegt in einer zu klein erwarteten niedrigen Coverage und einer zu hoch erwarteten hohen Coverage.

3.4.3 Verteilung der Null-Coverage

Die Modellierung der Null-Coverage ist sehr präzise. Wie in Abbildung 3.5 zu sehen ist, wird die Verteilung sehr genau approximiert. Ein kleiner Unterschied ist bei den sehr kurzen Null-Coverage-Regionen zu sehen.

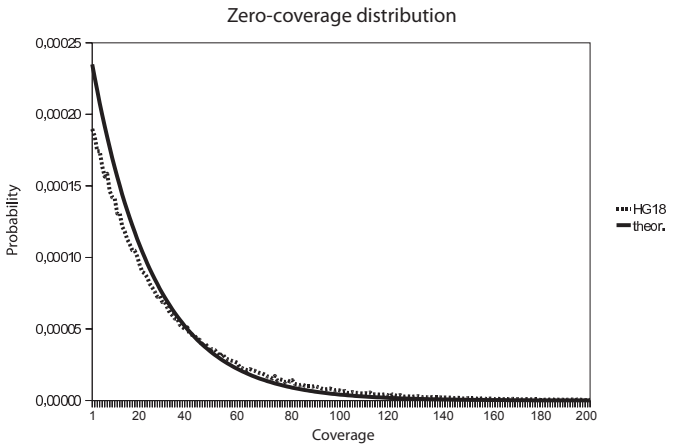

Abb. 3.5: Null-Coverage-Verteilung. Die Verteilung der Null-Coverage einer festen Anzahl von Reads des Humangenoms im Vergleich zur theoretischen Verteilung nach der entwickelten Formel. Ein kleiner Unterschied ist bei den sehr kurzen Null-Coverage-Regionen zu sehen.

3.4.4 Verteilung der Contig-Längen

Auf Grund des charakteristischen Verlaufs der Verteilung der Contig-Längen zeigt dieser Vergleich in besonderem Maße die Aussagekraft des verwendeten Modells zur Modellierung der verschiedenen Verteilungen. Die gemessenen Daten werden sehr genau von der Formel zur Berechnung der Verteilung der Contig-Längen approximiert, wie Abbildungen 3.6 und 3.7 zeigen. Abbildungen 3.6 und 3.7 unterscheiden sich in darin, dass aus Skalierungsgründen Abbildung 3.7 keine Contigs der Länge 47 enthält, dies sind Contigs, die aus einem Read bestehen. In Abbildung 3.6 ist zu sehen, dass es sehr viele Contigs dieser Länge gibt, was die genaue Betrachtung des Verlaufes der Verteilung in dieser Abbildung erschwert.

Weiterhin lässt sich an diesen Abbildungen erkennen, dass die Wahrscheinlichkeit des Auftretens von Contigs, die kürzer oder gleich der zweifachen Readlänge sind, identisch ist. Dies hat seinen Grund darin, dass in diesem Intervall die Wahrscheinlichkeit längenunabhängig ist. Der starke Abfall bei 94bp wird dadurch verursacht, dass die Wahrscheinlichkeitsfunktion die Bedingung eines dritten Reads in diesem Intervall bekommt. Wie die theoretische Kurve zeigt, ist dieser *Schritt* auf die rekursive Definition der Wahrscheinlichkeitsfunktion zurückzuführen.

Im Vergleich der Kurven zeigen die gemessenen Daten eine geringfügig höhere Anzahl von kurzen Contigs. In Abbildung 3.7 ist aber zu sehen, dass es sich um einen relativ kleinen Unterschied handelt.

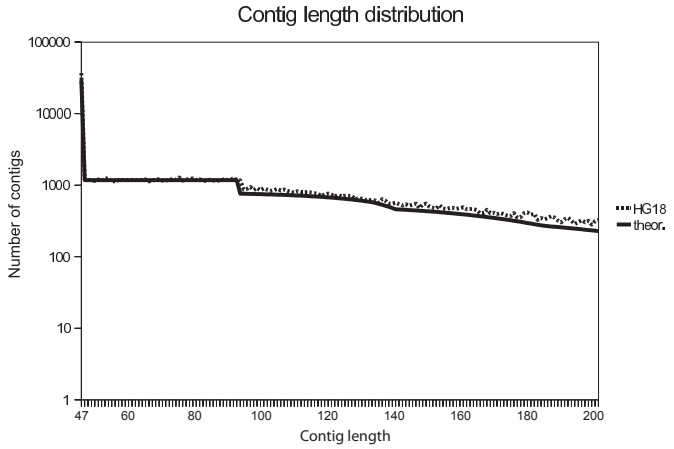

Abb. 3.6: Verteilung der Contig-Längen. Die Verteilung der Contig-Coverage, einer festen Anzahl von Reads des Humangenoms, im Vergleich zur theoretischen Verteilung nach der entwickelten Formel. Im Vergleich der Kurven zeigen die gemessenen Daten eine geringfügig höhere Anzahl von kurzen Contigs. Abbildung 3.7 zeigt dieselben Daten in einer anderen Skalierung.

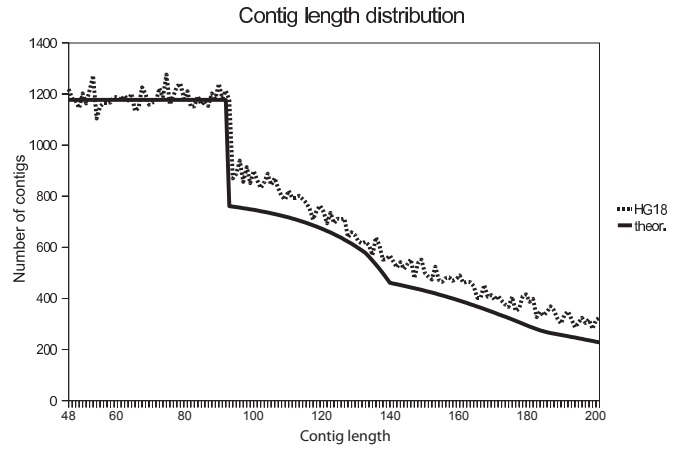

Abb. 3.7: Verteilung der Contig-Längen. Wie Abbildung 3.6, aber ohne Contigs der Länge 47bp, die nur aus einem Read bestehen.

3.4.5 Abschätzung von $N50$

Die Messungen haben ergeben, dass es möglich ist, den $N50$-Wert genau abzuschätzen. Die gemessenen Daten ergaben einen $N50$-Wert von 172 und einen dazugehörigen, theoretischen $N50$-Wert von 195.

Dies ist so zu interpretieren, dass das theoretische Modell mehr lange Contigs prognostiziert, als später gemessen werden. Dies ist passend zu der Beobachtung, dass die Contiglängen im unteren Bereich zu niedrig berechnet werden (siehe Abbildung 3.7), wofür im nicht dargestellten Bereich jenseits der 200bp höhere Contig-Anzahlen vorhanden sein müssen, da die absolute Coverage die selbe ist.

3.4.6 Abschätzung der Contig-Anzahl

Das theoretische Modell berechnet eine Contig-Anzahl von 200.067 Contigs, wohingegen die gemessenen Daten 189.753 Contigs wiedergeben.

3.5 Zusammenfassung

Die entwickelten Formeln liefern sehr gute Abschätzungen für die zu erwartenden Ergebnisse eines Sequenzierungsprozesses. Die Charakteristika eines Sequenzierungsprozesses wie die Coverage-Verteilung, Null-Coverage-Verteilung, Contig-Längen-Verteilung, $N50$ und Contig-Anzahl, können durch diese Formeln nachgebildet werden. Es wurde gezeigt, dass gemessene Daten die theoretischen Ergebnisse unterstützen.

Weiterhin sind die entwickelten Formeln insofern unabhängig von Sequenzierfehlern und Biases der Sequenzierer, dass die theoretischen Werte die messbaren Daten ohne Betrachtung dieser Eigenschaften nachbilden. Die Ursache hierfür ist, dass die Betrachtung globaler Verteilungen bei sehr großen Eingabemengen kleinere Besonderheiten und lokale Fehler statistisch herausfallen lassen. Sowohl diese Fehler als auch Repeat-Regionen beeinflussen vermutlich lediglich die *Effektivität* der Reads.

4 Alignment von Sequenziererdaten

4.1 Einführung und Motivation

Das Alignment von Reads an eine Referenzsequenz bezeichnet das Finden einer *bestmöglichen* Position eines Reads innerhalb einer bekannten Genomsequenz, wobei *bestmöglich* einer Metrik folgt, die genau besagt, wie gut ein Read an eine bestimmte Position passt.

Möchte man beispielsweise erforschen, auf welche Art sich äußerliche Eigenschaften eines Menschen oder vererbbare Krankheiten im Genom darstellen, wird so vorgegangen, dass Unterschiede zwischen den Genomen mehrerer Individuen identifiziert werden, um Rückschlüsse auf die Eigenschaften ziehen zu können. Da die Genome dieser Individuen zu über 99% identisch sind, reicht es meist ein Genom zu assemblieren und die Reads der anderen Individuen an einem Genom (dem Referenzgenom) zu alignen (auszurichten), um die Unterschiede zu finden.

In Abbildung 4.1 ist ein Beispiel zu sehen.

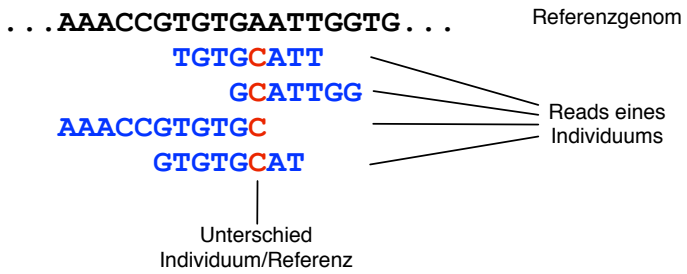

Abb. 4.1: Unterschiede einzelner Basen von Individuen
Reads eines Individuums alignt an einem Referenzgenom. Die rote Base ist der Unterschied.

Das Alignment von Reads an ein Referenzgenom ist weiterhin der erste Schritt in den meisten vielstufigen Abläufen (Pipelines) in der komparativen Genomforschung, zu dessen Disziplinen das Variant Calling[1], die Isoform Quantifizierung[2] und die differenzielle Gen Expression[3] gehören. Bei vielen Pipelines ist hierbei das Alignment der Schritt, der am längsten dauert, da der jeweilige Alignmentalgorithmus (Aligner) ein kompliziertes berechnungstheoretisches Problem lösen

[1]Variante Calling: Bestimmung von Unterschieden innerhalb eines Genoms
[2]Isoform Quantifizierung: Bestimmung der Häufigkeit der Expression von Isoformen
[3]Differenzielle Gen Expression: Bestimmung der Häufigkeit der Transkription eines Gens

muss. Dies ist die Berechnung der Wahrscheinlichkeiten aller Alignments (also der möglichen Positionen) und darauf folgend, der Ermittlung der Position an die der betrachtete Read mit der höchsten Wahrscheinlichkeit gehört.[38]

Weiterhin führt die enorm schnelle Weiterentwicklung der Next-Generation Sequenzierer dazu, dass riesige Datenmengen verarbeitet werden müssen. So sequenziert der Illumina HiSeq 2500 ein menschliches Genom mit 600 Millionen Paired Reads der Länge 100bp (insgesamt 120 Gigabasen) in nur 27 Stunden. Die Kosten sinken hierbei stark, so dass zum Stand dieser Arbeit die Kosten für die Sequenzierung eines menschlichen Genoms auf unter $1000 geschätzt werden. [47]

Mathematisch lässt sich das Alignment als Optimierungsproblem definieren. Hierbei ist das Optimierungsproblem die Bestimmung einer Position p eines Reads r der Länge l im Genom G der Länge n für die gilt:

$min(d(G_{p,l}, r))$

wobei d eine Distanzfunktion ist mit $d : (a, b) \rightarrow \mathbb{R}$ und $G_{p,l}$ die Subsequenz von G beginnend an der Stelle p mit der Länge l.

Im Rahmen dieser Arbeit wurde ein neuer Aligner, *Smarti*, entwickelt. Der Kerngedanke ist, dass die Sequenzierer immer längere Reads liefern und ihre Fehlerrate, die das Hauptproblem des Alignments ist, sinkt. Dies bedeutet, dass die Wahrscheinlichkeit, eine konsekutive Sequenz von mehreren fehlerfreien Basen (Seed) zu finden, steigt. Damit bietet sich die Möglichkeit, einen Read zu einer Referenzsequenz mit Hilfe eines Seeds schnell, mit wenigen Rechenoperationen und Speicherzugriffen im Vergleich zu existierenden Algorithmen, zu finden. Dieses Verfahren wird seed-and-extend genannt und bezeichnet Algorithmen, die ausgehend von einem kleinen Abschnitt (Seed) eine Lösung erweitern (Extend), bzw. optimieren. Sie gehören somit zu den Greedy Algorithmen.

Einige in diesem Abschnitt der Arbeit entwickelten theoretischen Konzepte und Ideen wurden mit Sascha Möller und Niklas Paulsen in deren vom Autor betreuten, studentischen Arbeiten ([49, 58]) hinsichtlich ihrer Machbarkeit evaluiert, umgesetzt und weiterentwickelt. Die erste Veröffentlichung der Ideen war in [70],

eine erweiterte Version wurde mit Niklas Paulsen in [57] vorgestellt. In [92] erfolge eine weitere Veröffentlichung im Bereich Protein Alignment auf FPGAs.

Der Schwerpunkt der Forschungsarbeit waren die theoretischen Grundlagen und Ideen, die zu der Entwicklung des Algorithmus notwendig waren. Hierbei wurde besonders auf die algorithmische Sicht Wert gelegt. Das Verständnis der Daten und der berechnungstheoretischen Probleme waren essentielle Punkte, auf die sich diese Arbeit fokussiert.

4.2 Bestehende Algorithmen

Viele Aligner verwenden einen Genomindex um ein Alignment durchzuführen. Hierbei indizieren sie die jeweilige Referenzsequenz und speichern gewisse Teilsequenzen (z.B. k-mere) des Genoms mit deren Positionen innerhalb des Genoms ab oder generieren eine reversible Permutation der Teilsequenz, um schnell darin suchen zu können (s. Abb. 4.2).

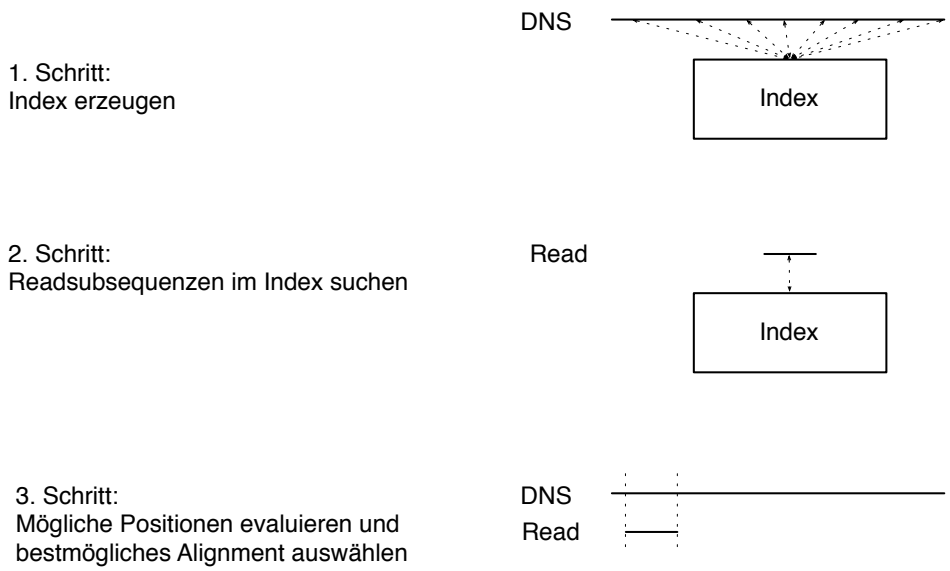

Abb. 4.2: Idee des Alignments
Die Idee des Alignments ist es, einen Index der Referenz zu erzeugen, um danach für alle Reads alle möglichen Positionen schnell zu ermitteln.

Auf diese Art und Weise kann die Anzahl der möglichen Positionen stark eingeschränkt werden, da nicht das gesamte Genom Position für Position durchsucht werden muss. Als Index wird häufig der Full-text Minute Index (FM-Index, siehe 2.4.7) [19] verwendet ([36, 41, 42, 43]), da er gemessen an seinem geringen Speicherbedarf relativ schnell ist. Index basierte Aligner arbeiten weiterhin so, dass sie den Read auf verschiedene definierte Arten permutieren, so wie die Teilsequenzen bei der Indexerzeugung permutiert wurden, und diese Permutation dann in dem Index suchen. Diese Permutation ist notwendig, um Unterschiede im Genom zulassen zu können. Im einfachsten Fall werden nur die k-mere indiziert. Die im Index angegebenen Positionen ergeben dann mögliche Kandidaten für die richtige Position. Nun werden mit einer vorgegebenen Distanzfunktion die möglichen Positionen verglichen. Obwohl der Suchraum sehr gross ist, können so viele mögliche Positionen ausgelassen werden, ohne dass sich die Sensitivität verringert.[38]

Der Seed-and-Extend Ansatz wurde bereits 1990 in BLAST (Basic Local Alignments Search Tool) verwendet (siehe [2]), der heute immer noch bei der Suche in Sequenzdatenbanken Anwendung findet.

In Abhängigkeit der Eingabedaten können Index-basierte Aligner aber auch ineffizient sein, wenn beim Alignment Lücken oder Einfügungen (Indels) erlaubt sein sollen. Indels können entweder durch Sequenzierfehler oder echte Variationen in dem sequenzierten Genom entstanden sein. Aligner, die den FM-Index verwenden, wie z.B. Bowtie2, sind hier meist erfolglos, wenn Indels in Reads oder der Referenz überbrückt werden sollen. Weiterhin vergrößern Indels den Suchraum sehr stark und verringern die Effektivität der Suchraumeinschränkung enorm. Hierdurch verringern sie die Geschwindigkeit der Index-basierten Aligner beachtlich.[38]

Alle hier vorgestellten Aligner unterstützen Paired end Reads (siehe 2.3).

4.2.1 Bowtie2

Bowtie2 ([38, 39]) generiert zuerst aus dem Read und dessen reversem Komplement mehrere Seeds. Nun sucht er im FM-Index nach dem Seed, wobei keine

Fehler enthalten sein dürfen. Danach erweitert er den Seed per dynamischer Programmierung unter Verwendung von besonders schnellen SIMD Prozessorbefehlen, um das Alignment zu prüfen. Der erste Schritt unter Verwendung des FM-Index erlaubt eine sehr schnelle Suche, da hier sehr speichereffizient gearbeitet werden kann. Allerdings sind in diesem Schritt keine Fehler im Seed zugelassen. Der zweite Schritt ermöglicht dann ein Alignment mit Fehlern außerhalb des Seed.[38]

Der von Bowtie2 verwendete FM-Index basiert auf der Burrows-Wheeler Transformation (BWT), ist ursprünglich für die Teilsequenzsuche in Texten entwickelt worden und ähnelt einer Baumstruktur, in der buchstabenweise gesucht wird. Er ist dabei aber in der Regel platzsparend linear im Speicher abgelegt. Bowtie2 sucht hierbei standardmäßig nach Seeds der Länge 28. Sind Fehler im Seed enthalten, wird der FM-Index allerdings sehr schnell sehr langsam, da per Backtracking (wie in einem Baum) weitergesucht werden muss.

Da Bowtie2 sehr schnell ist, erfreut er sich einer hohen Verbreitung und wird in vielen weiterführenden Anwendungen verwendet ([44, 84]). Er war der erste Algorithmus, der die BWT zum Short Read Alignment verwendet hat.

4.2.2 SOAP2

SOAP2 verwendet ebenfalls einen BWT-basierten Index, wobei die Suche durch einen Hash zusätzlich beschleunigt wird. Um Fehler zu tolerieren, werden die Reads in drei Teile geteilt und in jedem Drittel sind bis zu 2 Fehler erlaubt. Ansonsten arbeitet SOAP2 ähnlich Bowtie2. [43]

4.2.3 BWA

BWA [41] ist der Aligner, der in der wissenschaftlichen Community für Qualität steht, wobei bekannt ist, dass dies mit einem hohen Rechenaufwand einhergeht. Er arbeitet mit einem FM-Index, wobei Indels zugelassen werden. Hierbei wird allerdings der Suchraum beschränkt, indem zur Laufzeit das Abzweigen im Index bei Unterschieden abgespeichert wird und so zur Laufzeit Sequenzen, die zu weit

von der Originalsequenz abweichen, ausgeschlossen werden. Durch die Möglichkeit Indels zu tolerieren, ist er bei hohen Fehlerraten und langen Reads deutlich anfälliger und wird extrem langsam.

4.2.4 Weitere Aligner

Es gibt zum Stand dieser Arbeit eine Vielzahl von weiteren Alignern, die allerdings entweder nur zur Lösung von Spezialproblemen entwickelt wurden oder auf spezielle Hardware zugeschnitten sind. In diesem Kontext sei SOAP3 genannt, der GPU basiert Reads alignt [45]. Weiterhin seien für die Implementierung auf GPUs BarraCuda [35] und CUSHAW [46] genannt.

4.2.5 Bewertungskriterien für Aligner

Als Bewertungskriterien für Aligner sind im Wesentlichen die Alignmentqualität und die Geschwindigkeit zu betrachten. Die Alignmentqualität bezeichnet hierbei die Anzahl der Richtig-Positiv (TP) alignten Reads als auch die Anzahl der Falsch-Positiv (FP) alignten Reads (s. 2.4.4). Dabei hängt die Wertung der Wichtigkeit von TP und FP von der Anwendung ab, da eine höhere FP Rate in der Regel mit einer höheren TP Rate einhergeht, da der Aligner sich nicht so stark einschränken muss. Ist es aber wichtig, dass keine Fehler im Alignment sind, muss die FP Rate niedrig sein, was auf Kosten der Gesamtanzahl der alignten Reads gehen kann.

Die Geschwindigkeit bezeichnet die Anzahl der Alignments pro Zeiteinheit, wobei zu beachten ist, welche Rahmenparameter angenommen werden. Vor allem ist bei einem Vergleich die verwendete Hardware zu beachten, da nicht alle Aligner gleich gut die vorhandene Hardware ausnutzen.

Weiterentwicklungen und aktuelle Forschungsergebnisse

Es wurde im Rahmen dieser Arbeit zudem untersucht, ob bestehende Algorithmen verbessert werden könnten bzw. ob die Forschung an der Verbesserung eines bestehenden Algorithmus sinnvoller wäre, als einen *neuen* Algorithmus zu schaffen.

Da es allerdings bei einem Alignmentalgorithmus massiv auf das effiziente Zusammenspiel aller Komponenten in der späteren Implementierung ankommt, wurde sich dagegen entschieden. Das Alignmentproblem ist extrem rechen- und speicherintensiv und die Kontrolle über alle beteiligten Komponenten ist essentiell. Angefangen beim effizienten Einlesen, über das Verarbeiten, bis hin zur notwenigen Prozessverwaltung und schließlich zur Steuerung des Ausschreibens, ist es notwendig, dass alle Komponenten perfekt miteinander arbeiten. Bei der Verwendung von existierenden Lösungen entständen Schnittstellen, die eine Transformation der Daten erfordern würden, welche in der hochperformanten Implementierung Zeitverluste bedeutet hätte.

4.3 Der Smarti Algorithmus

Die grobe Idee des Smarti Algorithmus ist, aus einem Referenzgenom kurze Teilsequenzen (Seeds) und deren Herkunft im Genom in einem Index abzuspeichern. Um dann die Position eines Reads zu finden, werden Subsequenzen des Reads im Index *nachgeschlagen*. Existiert diese Subsequenz im Index, ist eine mögliche Position gefunden und der komplette Read kann auf seiner Gesamtlänge mit der Ursprungsregion verglichen werden (vgl. Abb. 4.3).

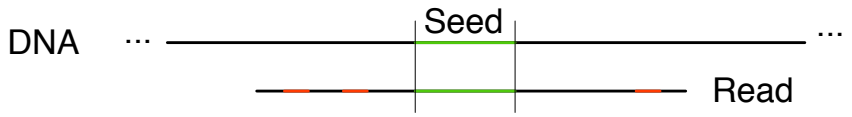

Abb. 4.3: Seed and extend Ansatz
Ist ein Seed (grün) gefunden, so kann der Rest des Reads verglichen werden. Unterschiede sind mit rot gekennzeichnet.

Bei der Entwicklung wurde der Fokus auf die theoretischen Eigenschaften der Genome, der Reads und der Rechnerarchitektur gelegt. Im Zentrum der Idee des Algorithmus steht, dass die Sequenzierungstechnologien sich weiterentwickeln und die Reads immer länger werden, wobei gleichzeitig die Fehlerrate sinkt, aktuelle Aligner aber hiermit nicht mithalten können. Dies hat seinen Ursprung darin, dass längere Reads, absolut gesehen, mehr Fehler enthalten, die wiederum dem häufig verwendeten FM-Index Schwierigkeiten bereiten. Beim Backtracking des FM-Index müssen im Fehlerfall jeweils drei weitere Positionen ausprobiert werden. Einige Aligner haben diesen Trend aufgegriffen und sich auf Seeds erweitert, sind aber trotzdem noch langsam.

Es ist also die Beobachtung, dass ein *schnelles* Finden der Seeds notwendig ist, um dann *schnell* evaluieren zu können, wie viele Fehler im möglichen Alignment sind. *Schnell* bedeutet in diesem Zusammenhang mit wenigen Rechen- und Speicherzugriffsoperationen. Gegebenenfalls kann erst bei Bestätigung einer Position, also wenn die Wahrscheinlichkeit sehr hoch ist, dass das aktuelle Alignment richtig ist, aufwändiger gerechnet werden. D.h., dass mit mehr Operationen, bzw. Rechenleistung, ein Read mit einer Position im Genom verglichen wird. Weiterhin wurde beachtet, dass bei der Entwicklung die x86 Architektur, mit den üblichen

Einheiten wie CPUs, RAM und Festplatten optimal verwendet wurde. Hierbei hat besonders die Zugriffsgeschwindigkeit des RAM Beachtung gefunden, da hier die häufig zugegriffenen Daten liegen müssen.

Nach theoretischen Vorüberlegungen in 4.3.2 folgt eine genaue Beschreibung des Algorithmus in 4.3.3.

4.3.1 Definitionen

Die Länge eines Genoms sei mit G bezeichnet. Ein Read habe die Länge l und das Readset, also die Menge der Reads, habe die Mächtigkeit n. Die Fehlerrate des Readsets sei mit e bezeichnet. Der Index nehme Seeds der Länge k für Anfragen entgegen.

Wie bereits im Kapitel zu den theoretischen Modellen von Readsets ausführlicher erläutert (vgl. 3.3.2), wird die Annahme getroffen, dass die Positionen der Reads über das Genom gleichverteilt sind und die Vorkommenshäufigkeiten der Nukleotide ebenfalls gleichverteilt sind.

4.3.2 Theoretische Vorüberlegungen

Mehrere theoretische Vorüberlegungen und Beobachtungen haben zu der Entwicklung des Alignmentalgorithmus Smarti geführt. Eine Beobachtung war, dass im Wesentlichen nur Aligner mit FM-Index verbreitet sind, wobei der FM-Index jedoch langsam bei Fehlern ist (siehe 2.4.7). Die Überlegung war nun, dass ein schneller Zugriff auf fehlerfreie Sequenzen notwendig ist, da für jeden Seed bis zu $2 * (l - k)$ Versuche unternommen werden müssen, um einen möglichen Seed in dem Index zu finden. Dies bedeutet, dass sehr viele Anfragen an den Index erfolgen, die nicht erfolgreich sind. Im Vergleich dazu sind nur wenige Operationen notwendig, wenn ein Seed gefunden wurde.

In den folgenden Abschnitten werden folgende Ideen dargelegt:

- Wichtig in den Vorüberlegungen war, dass die Positionsangaben von Seeds eines menschlichen Genoms sehr gut in den Adressraum einer Standardarchitektur passen.

- Festzustellen ist, dass sich bei dem Alphabet des Genoms aus vier Buchstaben eine einfache Hash-Tabelle anbietet, um darin eine direkte Adressierung umzusetzen.
- Kollisionen im Index durch doppelte Sequenzen müssen sehr selten vorkommen.

Adressraum

Das menschliche Genom hat eine Länge von $3 * 10^9$ Basenpaaren. Hiermit ist es möglich, Positionen in dem Genom mit 32bit zu speichern: $log_2(3 * 10^9) \approx 32$. Dies entspricht gerade der Anzahl an Bits, die sehr gut mit aktuellen Prozessoren verarbeitet werden können (bzw. vielfache davon).

Hash-Tabelle

In den Grundlagen unter 2.4.6 sind die Grundlagen zu Hashtabellen zu finden.

Das Genom ist eine Sequenz über dem Alphabet {`A`,`C`,`G`,`T`} und kann damit mit zwei Bit pro Buchstabe dargestellt werden.

Weiterhin gibt es gerade 4^k verschiedene Zeichenketten bei einer Sequenz der Länge k über einem Alphabet der Mächtigkeit vier. Betrachtet man nun aktuelle Rechnerarchitekturen, sind Arbeitsspeicher bis zu 32GB üblich, darüber hinaus möglicherweise aber mit hohen Investitionen verbunden. Hieraus folgt, dass eine Hash-Tabelle, die die Einträge direkt adressiert, nur eine gewisse Sequenzlänge beinhalten kann; in der Hash-Tabelle ist also ein Eintrag in binärer Repräsentation gerade die Adresse in der Hash-Tabelle.

Beispiel: Sequenzen der Länge 5 sollen mit Positionen gespeichert werden. Gegeben sei eine Sequenz $GAACT$ an Position 250 im Genom. Abbildungsvorschrift des Alphabets `A` $\rightarrow$ 00, `C` $\rightarrow$ 01, `G` $\rightarrow$ 10, `T` $\rightarrow$ 11. Die Sequenz ist dann in Binärrepräsentation 1000000111. In der Hash-Tabelle würde nun an Position 1000000111, also Position 519, der Wert 250 stehen. Die Hash-Tabelle müsste 4^5 Einträge haben, die jeweils 10 Bit benötigen.

Entsprechend lässt sich ableiten, dass in die maximale Größe eines Arbeitsspeichers von 32GB eine vollständige Hash-Tabelle mit vier Byte Adressen (32bit = Positionsmaximum Humangenom) eine Sequenzlänge von 15 eines vierelementigen Alphabets ablegbar ist:

$log_4(\frac{2^{35}}{4}) = 15$

Diversität

Ist die Größe der Hash-Tabelle gegeben, ist eine wichtige Frage, wie viel Kollisionen es gibt, also wie oft identische Sequenzen der Länge k im Genom vorkommen und an der selben Stelle in der Hash-Tabelle gespeichert werden müssten. Diese Fragestellung sei im Folgenden mit Diversität bezeichnet.

Betrachtet man hierzu das Genom der Länge G als zufällige Sequenz, kann man das Eintragen in die Hash-Tabelle als *Ziehen* von $G - l + 1$ zufälligen Sequenzen betrachten und dann ermitteln, wie viele doppelte Sequenzen in diesem Zufallsprozess entstanden sind. Da $G >> l$, wird im Folgenden nur von G Sequenzen gesprochen.

Die Wahrscheinlichkeit, dass eine Sequenz x mal gezogen wird, ein Eintrag in der Hash-Tabelle also x mal gefüllt werden soll, ist

$P_s(x) = B(x|\frac{1}{4^k}, G),$

wobei die Hash-Tabelle Sequenzen der Länge k über dem Alphabet der Mächtigkeit Vier adressiert. Die Wahrscheinlichkeit, dass nun eine Adresse in der Hash-Tabelle mehr als einmal gezogen wird, ist dann entsprechend

$P_{>1} = 1 - (P_s(0) + P_s(1)).$

Hieraus folgt, dass $4^k >> N$ sein sollte, wobei N der Anzahl der zu speichernden Einträge entspricht, damit jeder Eintrag möglichst nur einmal im Genom vorhanden ist.

Anmerkung: Wichtig ist allerdings die Länge der Sequenzen für die spätere Suche, da zu lange Sequenzen absolut gesehen mehr Fehler enthalten und damit nicht in der Hash-Tabelle zu finden wären. Eine Hash-Tabelle mit beispielsweise

Einträgen der Länge 50 würde nicht funktionieren, da die Sequenzen sehr viele Fehler enthalten würden und nicht auffindbar wären.

Da k durch die maximale Hash-Tabellen-Größe beschränkt ist, kann nur die Anzahl der Einträge reduziert werden, um $P_{>1}$ zu reduzieren. Dies ist wie folgt möglich:

Ein k-Raster wird über das Genom gelegt und nur jede k-te Position ergibt einen Seed, der in die Hash-Tabelle eingefügt wird. Dies führt dazu, dass im schlimmsten Fall $k-1$ Anfragen notwendig sind, bis ein Seed bei einem fehlerfreien Read gefunden wird (siehe Abbildung 4.4). Im Mittel sind es $\frac{k-1}{2}$ Anfragen, die an die Hash-Tabelle gestellt werden müssen, bis eine Sequenz gefunden wurde.

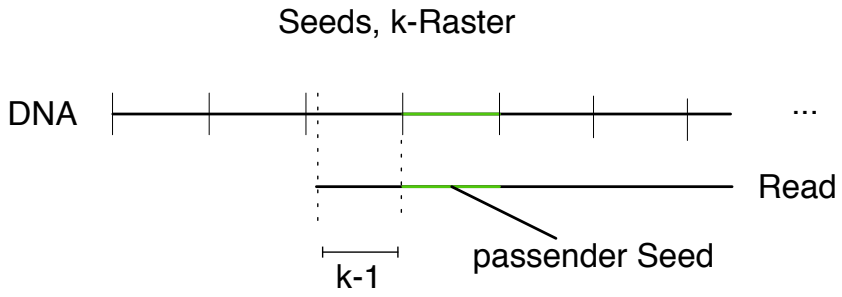

Abb. 4.4: Schlechtester Fall einer Seedposition
Im schlechtesten Fall sind $k-1$ Seedanfragen notwendig, bis ein Seed gefunden wird.

In Abbildung 4.5 ist zu sehen, wie sich die Wahrscheinlichkeit mehr als einmal vorkommender Sequenzen in Abhängigkeit zu der Sequenzlänge k beim menschlichen Genom entwickelt, wenn jede Sequenz in die Hash-Tabelle eingefügt würde. Im Vergleich dazu ist zu sehen, wie sich diese Wahrscheinlichkeit bei einem k-Raster entwickelt.

Distanzfunktion

Die Distanzfunktion ist nach der Seed-Suchen Funktion die am Häufigsten genutzte Funktion im Algorithmus. Sie bestimmt, wie ähnlich sich zwei Sequenzen der Länge l sind. Wie bereits unter Grundlagen 2.4.3 erläutert, bietet sich im

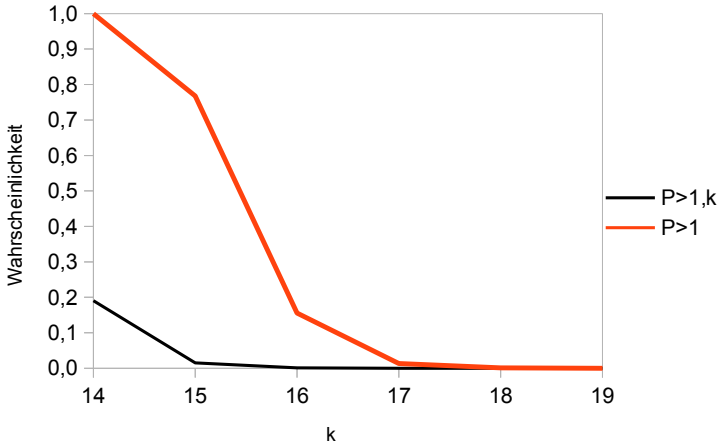

Abb. 4.5: Entwicklung der Diversität
Sequenzanzahl $3 * 10^9$, Variable Hash-Tabellen-Größe k. Graph zeigt Entwicklung mit und ohne k-Raster.

Wesentlichen der Smith-Waterman-Algorithmus an, der Lücken in den Sequenzen zulässt, oder die Hammingdistanz, die keine Lücken zulässt, aber sehr effizient implementiert werden kann. Der eigentliche Smith-Waterman-Algorithmus benötigt $O(l^2)$ Operationen, wohingegen die Hammingdistanz in $O(l)$ einen Abstand bestimmt.

Verwendet werden im Algorithmus beide Funktionen, wobei die Hammingdistanz für alle Alignments verwendet wird. Wird nach einer bestimmten Anzahl von Versuchen kein passender Treffer gefunden, wird der Smith-Waterman-Algorithmus eingesetzt. Details hierzu werden bei der Beschreibung des Algorithmus erläutert.

Fehlerrate

Ein weiterer kritischer Punkt bei dem Design eines Alignmentalgorithmus ist die Fehlerrate.

Wie weiter oben beschrieben, steigt die Anzahl der potentiellen Fehler pro Seed, wenn man dessen Länge erhöht. So enthält ein Seed der Länge k folgende Fehleranzahl: $P_F(e;k) = 1 - B(0|e,k)$

Bei üblichen Werten von $k = 17, e = 0,01$ ergibt sich $P_E(0,01; 17) = 0,1571$. Ein möglicher Seed hat somit zu knapp 16% einen Fehler und kann damit nicht in der Hash-Tabelle gefunden werden. In diesem Fall ist es aber möglich, den nächsten Seed zu nehmen, der k Positionen versetzt liegt. Dieser Seed enthält ebenfalls mit $P_F(e; k)$ einen Fehler, aber die Wahrscheinlichkeit, dass beide Seeds einen Fehler enthalten, fällt auf $P_F(e; k)^2$. Bei den obigen Beispielwerten sind dies nur noch 2,47% nicht auffindbare Seedtupel und bei Tripel wären es nur noch 0,39%.

Die Entwicklung dieser Wahrscheinlichkeiten mit höheren Potenzen und verschiedenen Fehlerraten ist in Abbildung 4.6 zu sehen. Es lässt sich daraus ableiten, dass eine gewisse Anzahl von Reads mit einem beliebigen Seed-basierten Algorithmus nicht alignt werden kann, da jeder Read nur im Mittel $\lfloor \frac{l-\frac{k}{2}}{k} \rfloor$ Seeds enthält.

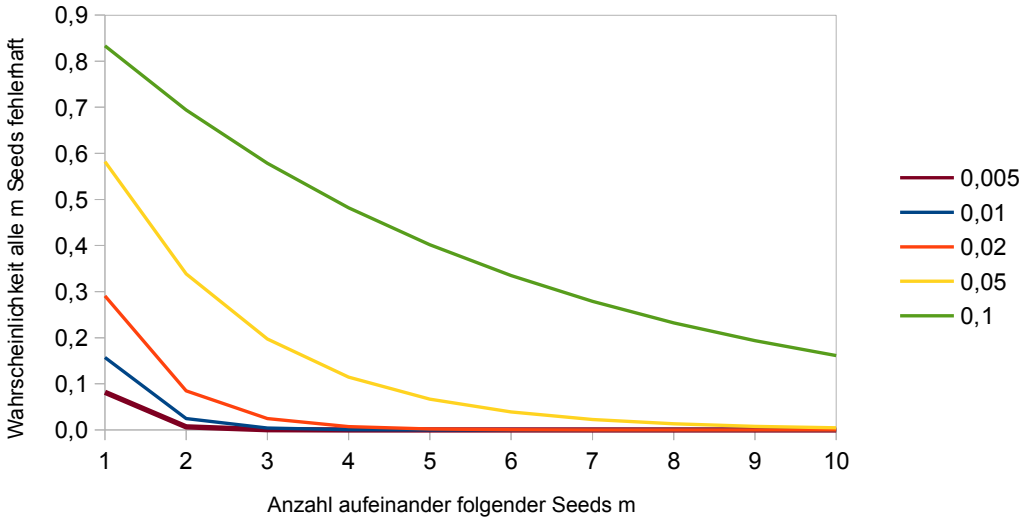

Abb. 4.6: Verlauf Seed-Fehlerhaftigkeit
Die x-Achse gibt die Anzahl der Seeds m an, die maximal verwendet werden. Die Kurven entsprechen verschiedenen Fehlerraten der Readsets und geben an, mit welcher Wahrscheinlichkeit alle m Seeds fehlerhaft sind und kein Seed erfolgreich in der Hash-Tabelle gefunden werden kann. Für eine logarithmische Darstellung siehe Abbildung 4.7

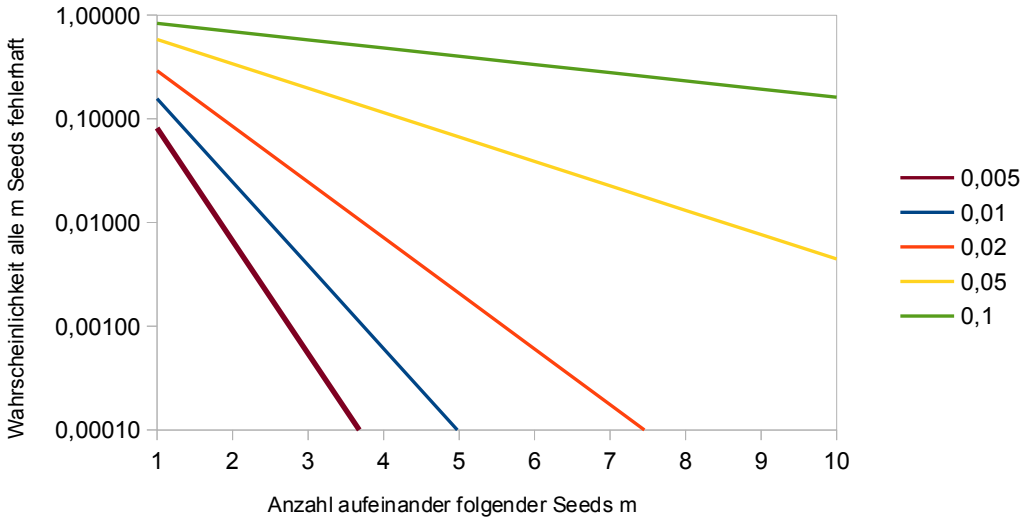

Abb. 4.7: Verlauf Seed-Fehlerhaftigkeit (logarithmisch)
Wie Abbildung 4.6, nur mit logarithmischer Darstellung und beschnitten.

Einstellung der Distanzfunktion

Wenn ein Alignmentalgorithmus ein gültiges Alignment finden soll, muss definiert werden, welchen maximalen Abstand v die Distanzfunktion zulassen darf, damit ein Alignment als gültig anerkannt wird. Ist v zu klein, werden mögliche gültige Positionen, die durch Fehler etwas unterschiedlich sind, verworfen. Ist v hingegen zu groß, werden Alignments gefunden, die nur ähnlich sind und gar nicht zusammen passen. Das Festlegen von v erfolgt bei den aktuell verbreiteten Algorithmen über einen nicht intuitiven Parameter, den der Anwender eingeben muss. Die Standardeinstellungen sind zwar für Standard-Anwendungsfälle verwendbar, liefern aber nicht immer das Optimum.

Aus diesem Grund wurde betrachtet, wie der maximale Distanzwert v für die Distanzfunktion (hier die Hammingdistanz) automatisch bestimmt werden kann, ohne dass der Benutzer komplexere Rechnungen selber durchführen muss. Hierbei ergab sich, dass die Distanzfunktion im Wesentlichen von der Fehlerrate e des Readsets abhängt. Die Distanzfunktion hängt auch von dem Unterschied des sequenzierten Genoms zum Referenzgenom ab, da aber die Wahrscheinlichkeit eine unterschiedliche Base zu haben mindestens eine Potenz kleiner als e ist, sei dies nicht weiter betrachtet.

Mit Hilfe von e ergibt sich nun für v, dass die Distanzfunktion mit Wahrscheinlichkeit P_H ein gültiges Alignment zulässt:

$P_H = \sum_{i=0}^{v} B(i|l, e)$

Dies entspricht der Wahrscheinlichkeit, dass ein Read der Länge l maximal v Fehler enthält.

In Abbildung 4.8 ist zu sehen, wie sich bei einer Readlänge $l = 150$ die Wahrscheinlichkeit der Alignmentmöglichkeit (y-Achse) in Abhängigkeit der Maximalen Distanz v (x-Achse) für verschiedene Fehlerraten (Kurven 0,01-0,25) verhält.

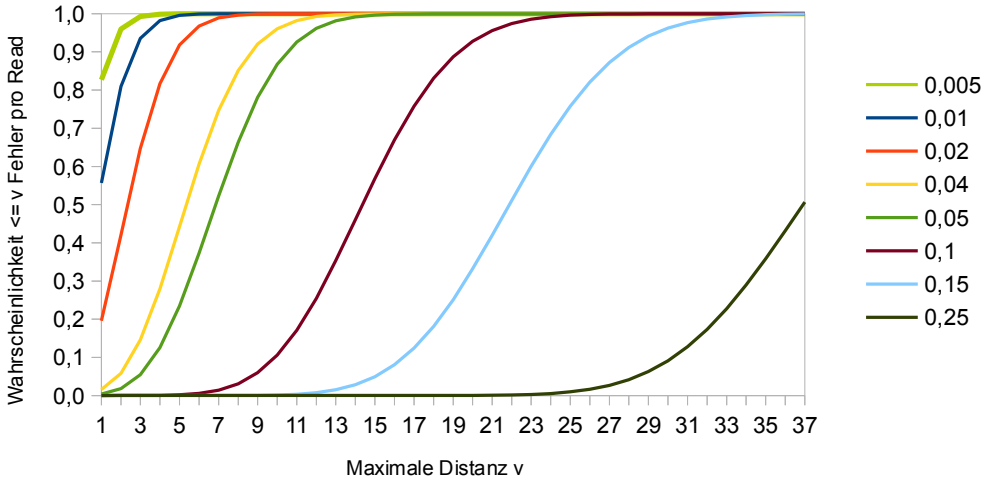

Abb. 4.8: Entwicklung der maximalen Distanz v
Readlänge 150, die Kurven 0,01 bis 0,25 entsprechen verschiedenen Readsetfehlerraten. Die Grafik gibt an, wie sich die Wahrscheinlichkeit einen Read mit maximaler Distanz v alignen zu können verhält.

Da die Fehlerraten der verschiedenen Sequenziertechnologien bekannt sind, kann der Anwender dem Aligner die Fehlerrate als Parameter übergeben. Alternativ kann die Fehlerrate automatisch ermittelt werden, indem eine genügend große Stichprobe, z.B. 1000 Reads, genommen wird und mit einem relativ großen maximalen Distanzwert alignt wird. Aus den Alignments kann man nun die Fehlerraten ermitteln, indem man die Basenunterschiede bestimmt. Der Mittelwert der Fehlerraten ist nun ein guter Wert für das weitere Alignment. Die im Mittelwert enthaltenen Ausreißer führen dazu, dass etwas *großzügiger* alignt wird. Bei dem

Alignment fallen dann aber die Ausreisser wiederum heraus, da sie deutlich über dem Mittelwert liegen.

Anzumerken ist, dass der Algorithmus durch diesen Automatismus selbständig mit wachsenden Readlängen zurecht kommt, da er die Anzahl der zugelassenen Fehler pro Read in Abhängigkeit zur Readlänge selber anpasst. Es ist nicht notwendig, dass Benutzer sich bei verlängerten Reads selber über die neuen Parameter Gedanken machen.

4.3.3 Der Algorithmus

Im folgenden Abschnitt wird beschrieben, wie mit den Vorüberlegungen des vorherigen Kapitels der Smarti Algorithmus entstanden ist. Hierbei wird in diesem Kapitel vor allem auf die technischen Aspekte eingegangen.

Aufbau der Hash-Tabelle

Die im obigen Abschnitt theoretisch entwickelte Hash-Tabelle kann mit ein paar Erweiterungen direkt umgesetzt werden. Da Kollisionen von Einträgen in der Praxis vorkommen, wurde eine Unterlisten-Struktur geschaffen, die Seeds verwaltet, die mehr als einmal vorkommen. In Abbildung 4.9 ist diese Struktur zu sehen.

Für Seeds, die einmal vorkommen, wird direkt die Position in der Hash-Tabelle gespeichert. Für Seeds, die mehrfach vorkommen, wird ein Zeiger auf eine Unterlistenstruktur gespeichert, die wie in Abbildung 4.10 aufgebaut ist.

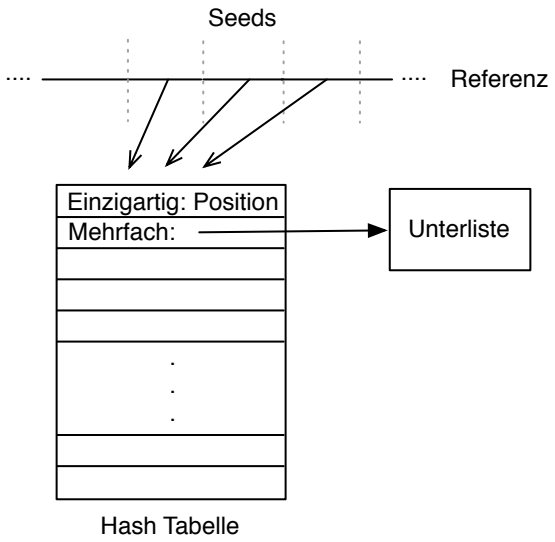

Abb. 4.9: Aufbau der Hash-Tabelle
Einzigartige Seeds halten in der Hash-Tabelle direkt die Adresse im Genom vor. Mehrfach vorkommende Seeds halten die Positionen in Unterlisten.

Um effizient auf die Unterliste zugreifen zu können, wird zuerst die Anzahl der Einträge der Unterliste gespeichert, dann folgt eine Liste der Sequenzen, die darin enthalten sind. Diese Liste der enthaltenen Einträge wird wiederum verkleinert, indem das für alle Sequenzen identische Präfix entfernt wird und nur ein Suffix übrig bleibt.

Zu den Suffixen folgen Adressen in das darauf folgende Array. Auf diese Art und Weise kann sehr schnell in die entsprechende Unterstruktur gesprungen werden und die Positionen können iteriert werden. Über die Adressen im Array der darauffolgenden Suffixe kann die Anzahl der Einträge ebenfalls ermittelt werden.

Es hat sich in der Praxis gezeigt, dass die direkt adressierende Hash-Tabelle weniger Einträge als erwartet hat, dafür die Unterlisten aber mehr Einträge besitzen. In Abbildung 4.11 ist dies exemplarisch für das Humangenom zu sehen.

Dies wurde genutzt, indem die direkt adressierbare Hash-Tabelle um einzigartige Seeds aus der Referenz, die nicht auf dem k-Raster liegen, erweitert wurde. Hierdurch erhöht sich die Anzahl von möglichen Seeds und es reduziert sich die Anzahl der durchschnittlich notwenigen $\frac{k-1}{2}$ Anfragen an die Hash-Tabelle. Da

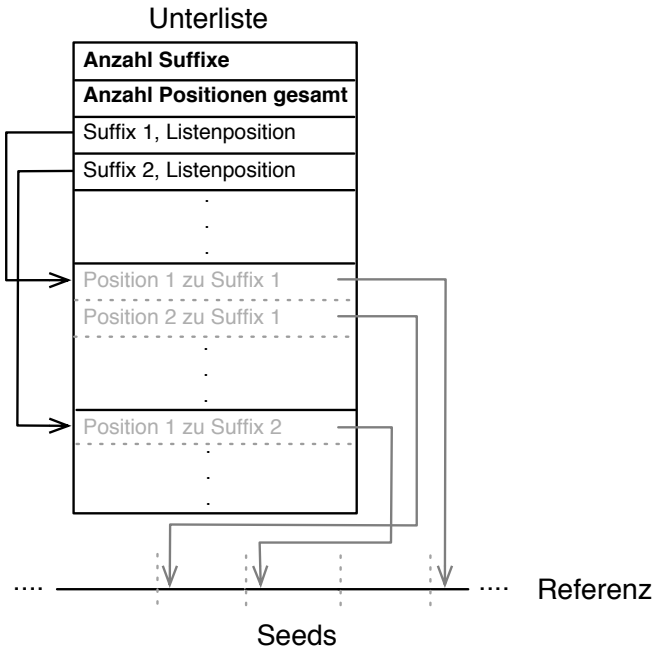

Abb. 4.10: Aufbau der Hash-Tabellen Subliste
Nach einem Header folgt eine Liste mit Suffixen. Diese Suffixe haben Zeiger auf sortierte Listen, in denen die zugehörigen Positionen innerhalb des Referenzgenoms stehen.

keine Unterlisten hierbei betroffen sind, da die Präfixe sonst nicht einzigartig wären, ist die Umsetzung nicht weiter zu erläutern.

Obwohl die Erstellung des Index sehr rechenaufwändig ist, muss sie in der Praxis nur ein mal erstellt werden und kann immer wieder verwendet werden. Da der Index sich im Gegensatz zu verschiedenen zu sequenzierenden Individuen nicht ändert, könnte dieser sogar anderen Benutzern zur Verfügung gestellt werden, so dass diese die Generierung nicht durchführen müssen.

Reduzierung der Tabellen-Größe

Nachdem die Hash-Tabelle gefüllt wurde, muss sie noch einmal bearbeitet werden. Da die Daten des Humangenoms sehr viele sich wiederholende Regionen haben, haben einige Unterlisten sehr viele Einträge. Dies führt dazu, dass die spätere Suche nach möglichen Positionen unnötig viele mögliche Positionen für bestimmte

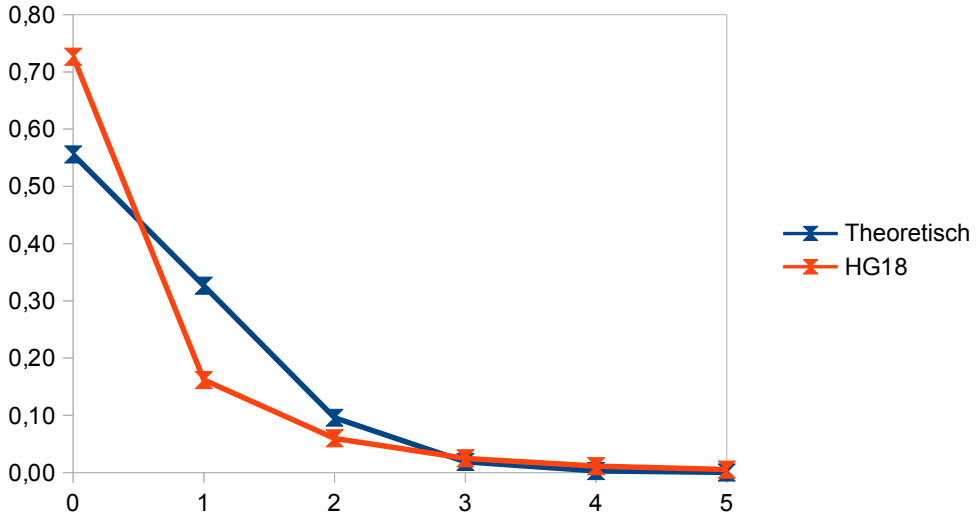

Abb. 4.11: Füllung der Hashtabelle
Es sind die theoretische Füllung im Vergleich zur tatsächlichen Füllung zu sehen. Verwendet wurde das HG18 und eine Hashtabelle mit Präfix der Länge 14 und Sequenzen der Länge 17. Abbildung nach [49].

Seeds liefert und somit einen sehr großen Teil der Rechenzeit beansprucht. Aus diesem Grund ist es möglich, bei der Generierung des Index einen Schwellwert anzugeben, der Unterlisten mit mehr Einträgen als des Schwellwerts entfernt (Hard cut), bzw. die Anzahl der maximalen Einträge für einen Unterliste beschränkt (Soft cut). Biologisch ist dies mit dem Ausschluss von Repeat-Regionen gleichzusetzen.

N-Basen

Bei der Sequenzierung, u.a. mit Illumina, kann es vorkommen, dass während der Sequenzierung nicht klar ist, um welches Nukleotid es sich handelt. In diesem Fall wird ein *N* in der Sequenz eingefügt. Entsprechend sind auch beim Humangenom HG19 noch einige Sequenzen nicht bestimmt. Beim Alignment muss entsprechend mit diesen Basen umgegangen werden.

Da ein weiterer Buchstabe das Alphabet auf 5 Zeichen erweitern würde, welches zur Folge hätte, dass 3 bit pro Zeichen benötigt werden, ist es keine Option mit *N*s zu arbeiten. Es werden daher alle *N*s entfernt und diese Lücken *gespeichert*.

Nach dem Alignment werden diese Lücken dann wieder entfernt. Anzumerken ist, dass Seeds, die über eine Position mit einem N verlaufen, später nicht gefunden werden.

Parallelisierung

Die Problemstellung an sich ist sehr gut parallelisierbar, da es keine Abhängigkeiten zwischen den Eingabedaten (Reads) gibt. Es ist möglich, das Readset trivial einfach in mehrere Teile aufzuteilen und verteilt zu bearbeiten. Dennoch kann das Problem schneller gelöst werden, als würde nur, wie soeben beschrieben, trivial parallelisiert werden.

Ist der Index im Arbeitsspeicher, so können mehrere Prozessoren oder Kerne (im Folgenden nur Prozessoren) auf diesen zugreifen und die Hardware kann besser ausgenutzt werden. Der Speedup ist hierbei fast linear, da kaum Überschneidungen in der Nutzung der Ressourcen entstehen.

Die Parallelisierung geschieht für den Benutzer automatisch, da aus den Systemvariablen die Anzahl der Prozessoren bestimmt wird und entsprechend viele Prozesse gestartet werden.

4.4 Ergebnisse aus Messreihen

Um die Güte eines beliebigen Aligners, also auch des entwickelten Algorithmus, zu ermitteln, ist es notwendig, dass eine Eingabe (ein Readset) für den Algorithmus vorliegt, zu der bekannt ist, wo die jeweiligen Reads ihren Ursprung haben. Verwendet man Realdaten, also Reads aus einem realen Sequenzierungsvorgang, fehlen die Ursprungspositionen der Reads. Weiterhin ist nicht bekannt, ob ein Unterschied von Basen aus dem Referenzgenom und einem Read des sequenzierten Organismus seinen Ursprung in einem SNP oder einem Fehler hat. Diese Problemstellung ist aber eine andere als das Alignment an sich und im Bereich des SNP Calling angesiedelt. Insbesondere die Falsch-Positiv Rate (s. 4.4.3) eines Alignmentalgorithmus ist schließlich interessant und nur mit synthetischen Daten möglich.

Aus diesem Grund wurden synthetische Daten aus dem Humangenom HG19 mit dem weit verbreiteten Tool mason[28] generiert. Hierbei werden die Ursprungspositionen als *Kommentar* in den Readdaten gespeichert, so dass später eine Positionsbestimmung und damit Validierung eines Alignments möglich ist. Weiterhin ist es möglich die nachzubildende Sequenzierplattform bei der Generierung mit anzugeben. Durch zusätzliche Konfigurationsmöglichkeiten von mason wurde die Fehlerverteilung der Reads möglichst nahe an reale Daten angepasst.

4.4.1 Eingabedaten

Es wurden mit mason Paired End Reads der Länge 150 generiert, wobei als Sequenziermodel Illumina vorgegeben wurde. Für die Distanz der Paired End Reads wurde 500 verwendet und Haplotype Variationen wurden ausgeschlossen. Die Fehlerraten der Readsets werden bei mason über drei Parameter angegeben, die Fehlerrate pro Base, die Fehlerrate am Anfang des Reads sowie die Fehlerrate am Ende des Reads. Die Fehlerrate am Anfang und am Ende wurden jeweils so gewählt, dass diese am Anfang wie üblich etwas niedriger und die Fehlerraten am Ende etwas höher als die durchschnittliche Fehlerrate waren. Die Parameter für Fehlerraten und Readanzahlen werden jeweils in den folgenden Ergebnissen mit angegeben.

Die Basis der Reads bildet das HG19 Genom, welches vom 1000Genomes Projekt frei verfügbar ist.

4.4.2 Alignmentrate

Um die Alignmentrate, also die Anzahl der richtig alignten Reads zu bestimmen, wurden Readsets mit mason generiert, die eine Fehlerrate zwischen 1% und 30% haben. Die Algorithmen wurden jeweils mit verschiedenen Optionen ausgeführt und die jeweils beste Option für das jeweilige Readset wurde für die Ergebnisse verwendet. Die Readsets enthielten jeweils 100.000 Reads. Abbildung 4.12 und 4.13 zeigen, wie sich die Alignmentrate der drei Aligner Bowtie2, Smarti und SOAP2 verhält. Hier ist zu sehen, dass SOAP2 sehr schlecht abschneidet und schon ab Fehlerraten von 2% nur noch weniger als 50% der Reads richtig alignt. Bowtie2 und Smarti sind bei niedrigen Fehlerraten bis 10% etwa gleich gut. Erst ab 10% ist Smarti deutlich besser. Betrachtet man den Bereich von 1% bis 5%, der als Vergrößerung in Abbildung 4.13 zu sehen ist, zeigt sich, dass Bowtie2 bei 1% und 5% etwas besser ist als Smarti. Hierbei sei allerdings auf die deutlich schlechteren Falsch-Positiv Raten von Bowtie2 vorgegriffen (s.u.).

4.4.3 Falsch-Positiv Rate

Die Falsch-Positiv Rate gibt an, wie viele Reads von einem Alignmentalgorithmus gefunden und an einer falschen Stelle angeordnet wurden. Sie ist eine Aussage über die Zuverlässigkeit eines Aligners und daher besonders wichtig für die Beurteilung, da aus der Falsch-Positiv Rate für synthetische Daten Rückschlüsse auf die Güte für reale Daten möglich sind. Alignt man reale Daten, so hat man nämlich als Ausgabe nur eine Rate von alignten Reads. Nimmt man nur diese Rate als Maß, so könnte es sein, dass der Aligner einen Teil der Reads willkürlich alignt hat und die vermeintlich hohe Rate an alignten Reads wäre irreführend.

In Abbildung 4.14 ist zu sehen, wie sich die Falsch-Positiv Rate der Aligner über verschiedene Fehlerraten hinweg entwickelt. Zu sehen ist, dass Bowtie2 mit steigender Fehlerrate eine steigende Falsch-Positiv Rate hat, wobei anzumerken ist,

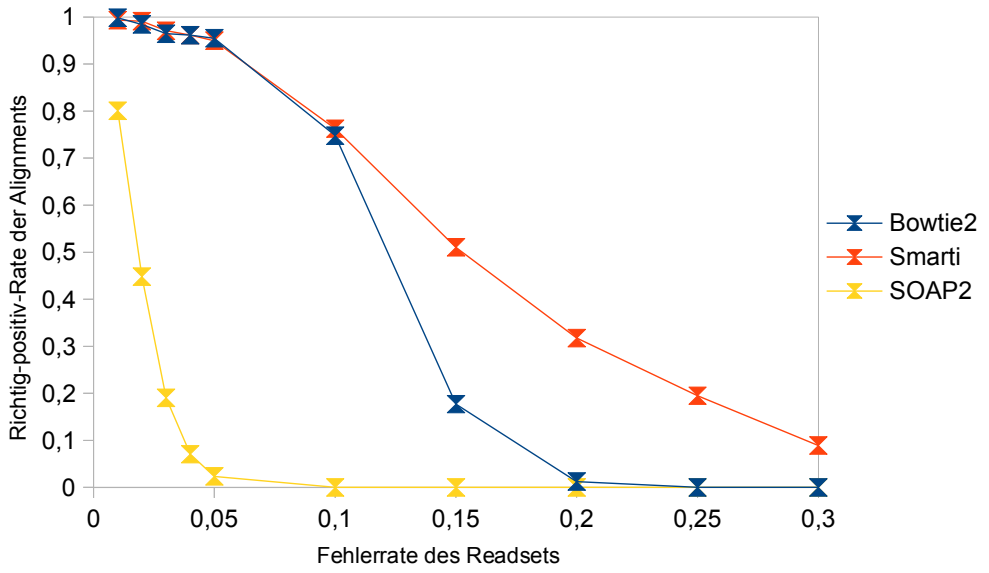

Abb. 4.12: Alignmentraten der Algorithmen
Verglichen werden die Alignmentraten der Algorithmen in Abhängigkeit zur Fehlerrate der Readsets.

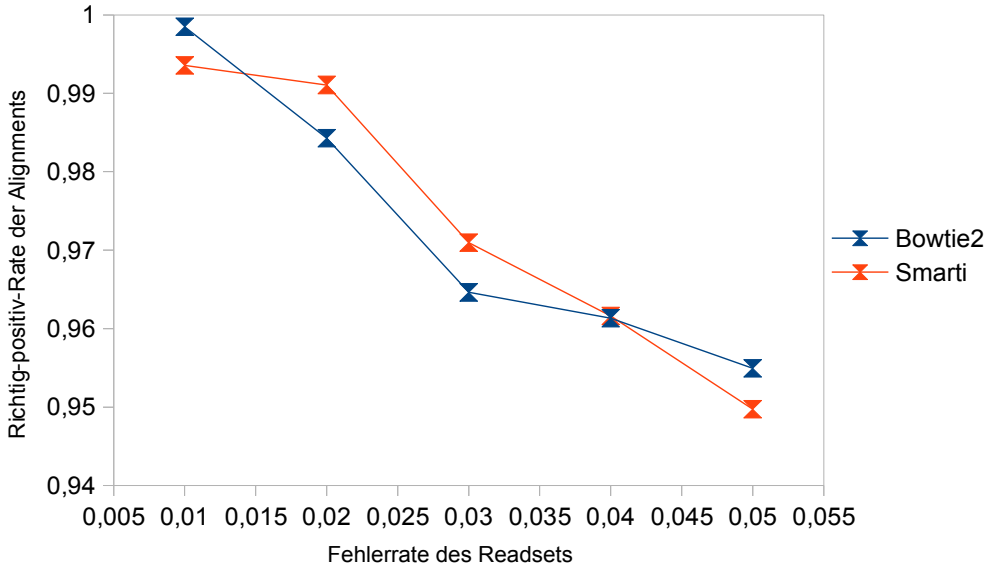

Abb. 4.13: Alignmentraten der Algorithmen (Vergrößerung)
Ausschnitt aus Abbildung 4.12

dass das starke Absinken der Falsch-Positiv Rate ab 10% direkt mit der sehr niedrigen Anzahl an Alignments einhergeht.

Bei Smarti hingegen ist die Falsch-Positiv Rate anfangs relativ stabil und sinkt mit höhere Fehlerrate.

Bei SOAP2 ist zwar die Falsch-Positiv Rate sehr gut, allerdings mit dem Nachteil, dass nur sehr wenige Reads alignt werden.

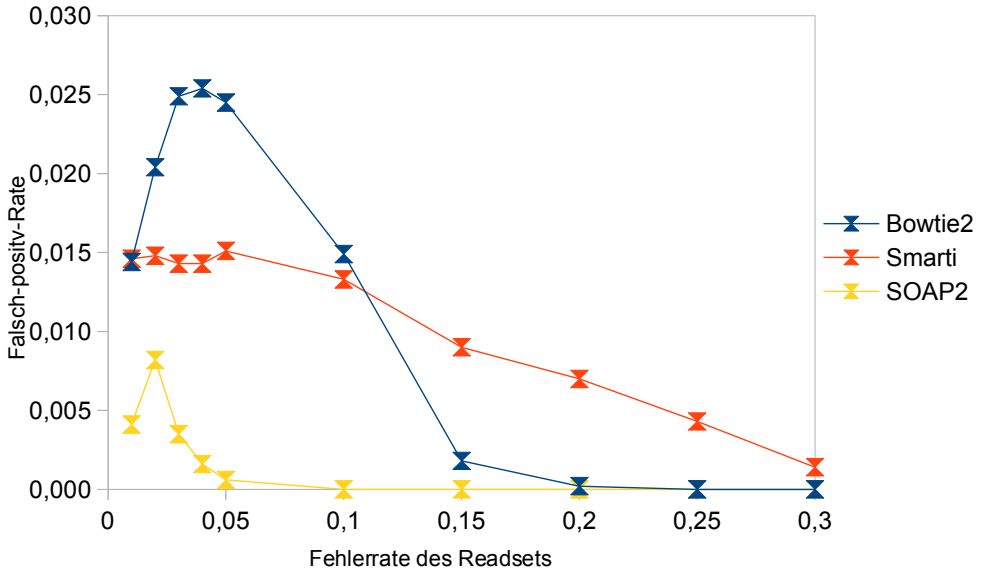

Abb. 4.14: Falsch-Positiv Rate bei verschiedenen Fehlerraten
Entwicklung der Falsch-Positiv Rate in Abhängigkeit zur Fehlerrate.

4.4.4 Laufzeit

Um die Laufzeiten zu vergleichen, wurden die Laufzeiten der Algorithmen für eine steigende Anzahl an Reads ermittelt. Hierbei wurden Ladezeiten für die jeweiligen Indizes der Aligner nicht berücksichtigt, da bei einem Sequenzierprojekt eines höheren Lebewesens in der Regel so große Readmengen und damit Laufzeiten notwendig sind, dass die Setup-Zeiten vernachlässigbar werden. Abbildung 4.15 zeigt, dass Smarti mit Abstand der schnellste Algorithmus ist. Alle drei Algorithmen zeigen einen linearen Anstieg der Laufzeiten, was die Skalierbarkeit der Algorithmen bestätigt.

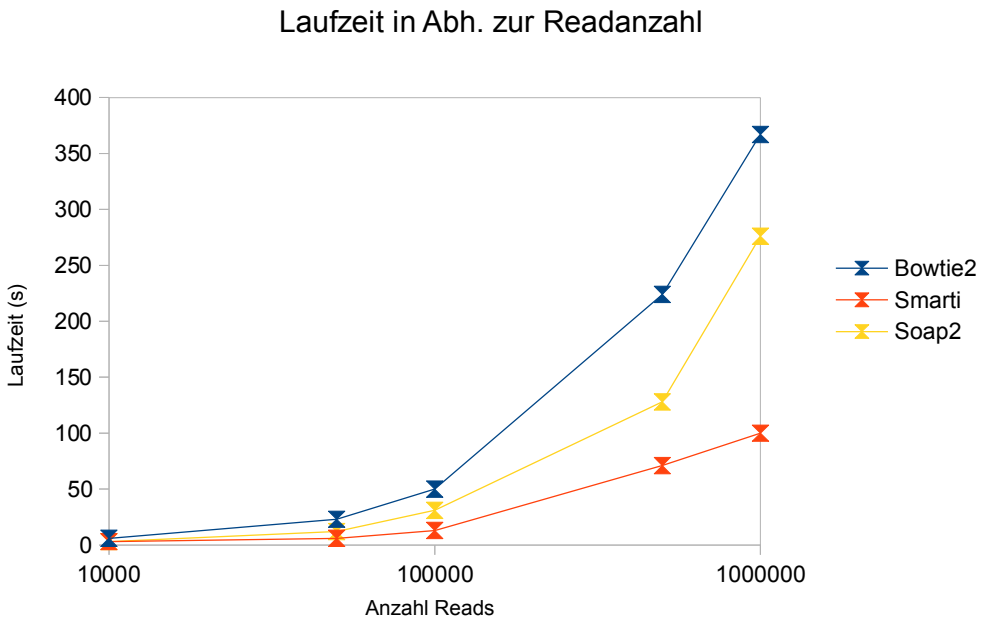

Abb. 4.15: Laufzeit in Abhängigkeit zur Readanzahl
Alignt wurden Reads mit 2% Fehlerrate auf einer EC2 Instanz des Typs m2.4xlarge. Verwendet wurden zwei Prozessoren.

Auf Basis dieser Daten lässt sich eine Hochrechnung für das vollständige Alignment der notwendigen Reads für ein Humangenom tätigen. Bowtie2 würde für ein Humangenom über neun Tage benötigen. Dies ist mehr als die dreifache Zeit, die Smarti, mit 2,5 Tagen, benötigen würde. Da die Alignmentraten von SOAP2 sehr niedrig waren, wurde dieser nicht betrachtet.

4.4.5 Parallelisierung

Um die Güte der Parallelisierung zu zeigen, wurden EC2 Instanzen des Typs hi1.4xlarge verwendet. Diese haben 16 virtuelle Prozessoren und bieten zwei SSD Festplatten. Virtuelle Prozessoren bezeichnen hierbei Recheneinheiten, es müssen allerdings nicht zwangsläufig 16 Kerne sein.

Da die Algorithmen einen sehr hohen Datendurchsatz haben, ist das Ablegen der Daten auf einem schnellen Medium hilfreich. Es wurde Bowtie2 im Vergleich zu Smarti mit fixen Parametern gestartet, wobei die Anzahl der Threads als Parameter variiert wurde.

Die zentrale Fragestellung ist nun, wie gut die Prozessoren ausgelastet werden können. Dies ist zum Einen die reine Laufzeit, weiterhin der Speedup in Relation zu einem Prozessor und schließlich der Parallelisierungsgrad, also der Quotient aus Speedup und Prozessoren.

In Abbildungen 4.16, 4.17 und 4.18 ist zu sehen, wie sich diese Werte entwickeln.

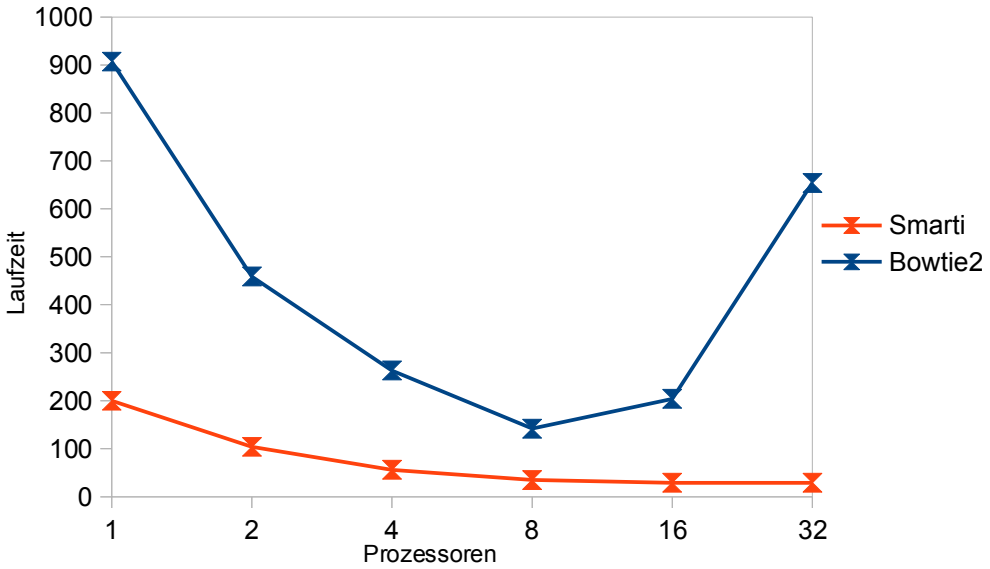

Abb. 4.16: Laufzeit bei Parallelisierung
Die Laufzeit in Abhängigkeit zu der Anzahl der verwendeten Prozessoren (Threads) ist zu sehen. Es wurden 1.000.000 Reads der Länge 150 mit 2% Fehlerrate alignt. Verwendet wurde eine EC2 Instanz des Typs hi1.4xlarge.

In Abbildung 4.16 ist zu sehen, dass Smarti durchgehend etwa vier mal schneller ist als Bowtie2. Auffallend ist, dass die Laufzeit bei Bowtie2 bei mehr als 8 Prozessoren wieder zunimmt. Dies deutet auf sehr viele Zugriffe auf begrenzte Ressourcen (z.B. Arbeitsspeicher) hin. Dies kann dazu führen, dass sehr hohe Wartezeiten entstehen, in denen die Prozessoren nicht rechnen können.

In Abbildung 4.17 ist zu sehen, wie sich der Geschwindigkeitsgewinn eines Alignments mit mehreren Prozessoren im Vergleich zu einem Prozessor verhält. Beide Aligner haben bis vier Prozessoren den selben Speedup. Danach ist Bowtie2 bei acht Prozessoren etwas besser, fällt dann aber deutlich ab, während Smarti gegen einen konstanten Speedup bei sieben konvergiert.

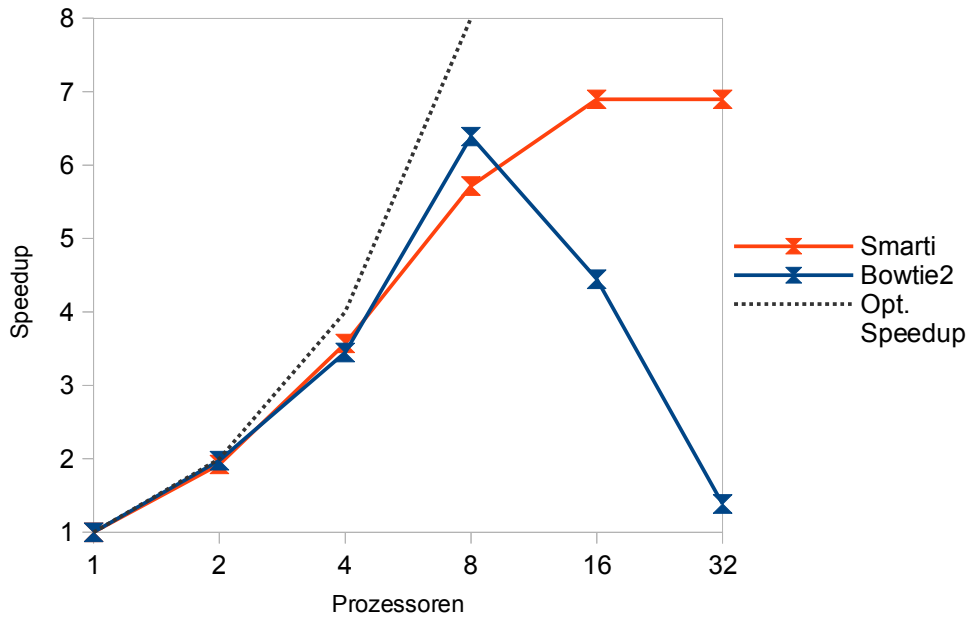

Abb. 4.17: Speedup bei Parallelisierung
Der Quotient von Abb. 4.16 aus Laufzeit mit einem Prozessor zu der Laufzeit mit mehreren Prozessoren. Die gestrichelte Linie gibt den optimalen Speedup an, wenn linear skaliert werden könnte.

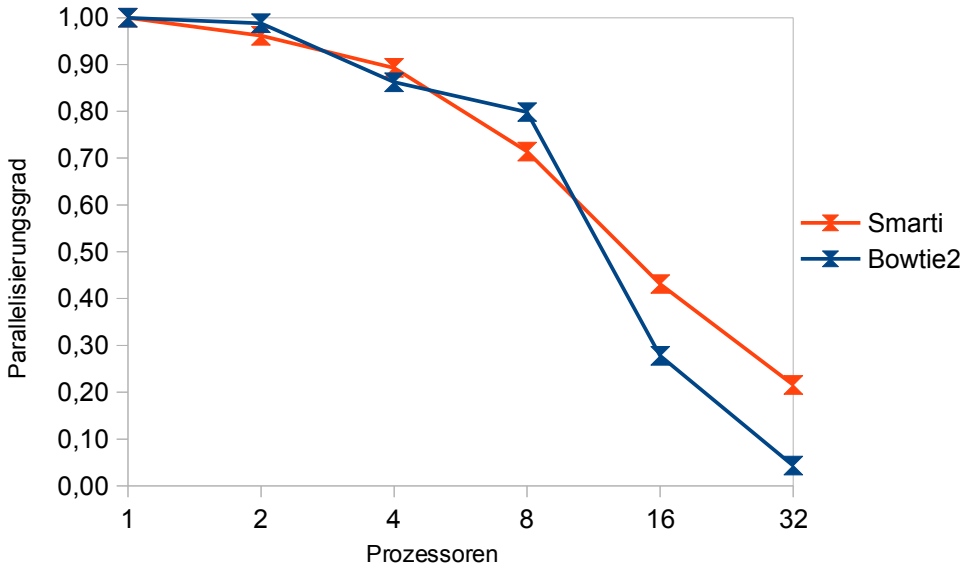

Abb. 4.18: Parallelisierungsgrad
Der Quotient von Abb. 4.17 aus Speedup und Prozessoranzahl.

4.5 Zusammenfassung und Ausblick

4.5.1 Zusammenfassung

In der Forschungsarbeit zu diesem Thema wurde festgestellt, dass aktuelle Aligner relativ langsam im Vergleich zu den theoretischen Möglichkeiten sind. Aus diesem Grund wurde nach Vorüberlegungen in dieser Arbeit ein Algorithmus zum Alignment von Reads an ein Referenzgenom entwickelt. Hierbei wurde der Fokus auf die statistischen Eigenschaften der Reads und des Referenzgenoms gelegt und besonders darauf geachtet, dass bei aktuellen Rechnerarchitekturen die Operationen durch die Zugriffszeiten von Arbeitsspeicher und Festplatten den Algorithmus nicht ausbremsen und die Prozessoren optimal genutzt werden.

Der Algorithmus verwendet hierzu einen im Arbeitsspeicher abgelegten Index in Form einer Hash-Tabelle, die mit nur einer Abfrage bestimmen kann, ob, bzw. wo, eine kurze Teilsequenz (Seed) eines Reads im Referenzgenom vorhanden ist. Ist ein Seed gefunden, wird mit einer sehr schnellen Distanzfunktion evaluiert, ob der Read in der gesamten Länge an dieser Stelle dem Referenzgenom ähnlich ist.

Die Ergebnisse der Alignments wurden mit den in der wissenschaftlichen Community verwendeten Algorithmen Bowtie2 und SOAP2 verglichen. Hierzu wurden synthetische Reads generiert, um die Anzahl korrekter Alignments und falsch-positiver Alignments bestimmen zu können.

In den Ergebnissen zeigt sich, dass der entwickelte Algorithmus sowohl mehr Reads alignt als auch eine niedrigere Falsch-Positiv Rate bei üblichen Anwendungsfällen hat. Des Weiteren ist die Laufzeit des entwickelten Algorithmus drei Mal niedriger als die des besten Aligners Bowtie2.

4.5.2 Ausblick

Ein Aligner muss, vor allem neben einer hohen Anzahl an richtigen Alignments, eine vertretbare Geschwindigkeit bieten. Diese beiden Eigenschaften wurden mit dem entwickelten Algorithmus erreicht.

Möglich ist nun, diesen schnellen *Kern* zu verwenden und für Benutzer weitere Anwendungsfälle wie SNP-Calling oder das Alignment von Readsets aus mehreren Organismen zu entwickeln.

Weitere qualitative Verbesserungen sind möglich, indem Qualitätswerte der Reads beim Alignment verwendet werden. Hierzu würde bei einem Alignment nicht nur die Anzahl der unterschiedlichen Basen bestimmt, sondern der Abstand mit den Qualitätswerten der jeweiligen Basen der Reads gewichtet. Dies würde ermöglichen, für einen kleinen Teil der Reads, die an mehrere Positionen passen könnten, eine zuverlässigere Aussage über die korrekte Position treffen zu können. Auch könnten die Enden der Reads, die teilweise sehr schlechte Qualitäten aufweisen, beschnitten werden, da diese fehlerreichen Bereiche das Alignment verfälschen.

Zu Erhöhung der Geschwindigkeit wäre es möglich, statt nur eines k-Rasters der Seeds, ein kleineres κ-Raster für die Abstände der Seeds zu nehmen. Da die Seeds trotzdem eine Länge von k hätten, würden sie sich also überschneiden. Diese zusätzlichen Seeds würden dazu führen, dass im Mittel $\frac{\kappa}{2} < \frac{k}{2}$ Anfragen notwendig sind, bis eine Sequenz im Referenzgenom gefunden wurde. Zu bedenken ist dabei, dass die Unterlisten stärker als bisher gefüllt würden. Dies bedeutet wiederum mehr Rechenaufwand, der aber auf lokalem Speicher im Cache des Prozessors ausgeführt würde.

Die Größe üblicher Arbeitsspeicher wird weiterhin steigen. So wird es bald möglich sein, eine größere Hash-Tabelle zu verwenden. Da der Informationsgehalt längerer Seeds mit jeder weiteren Base steigt und eine größere Hash-Tabelle *leerer* wird, da die Genome eine konstante Größe haben, könnte man in die Hash-Tabelle weitere Seeds aufnehmen. Dieses Seeds könnten normale, um Indels erweiterte, k-mere sein.

5 Fehlerkorrektur von Sequenziererdaten

5.1 Einführung und Motivation

Bei der Sequenzierung werden die Reads mit Hilfe eines komplexen Prozesses, der aus vielen manuellen, biochemischen Abläufen besteht, gewonnen. Bei diesem Prozess entstehen verschiedene Arten von Fehlern in den Reads, welche deren spätere Verarbeitung zur Rekonstruktion eines Genoms (Assembly) oder das Alignment von Reads an eine bekannte Sequenz beinträchtigen. Die Fehlerrate pro Base beträgt beispielsweise zwischen 1% und 2% für den Illumina Sequenzierer der ersten Generation [13].

Es gibt im Wesentlichen drei Ursachen, die zu Fehlern in Next Generation Sequencing Daten (siehe 2.3) führen: Systematische Fehler, Coverage Fehler und Chargenfehler, die von der Sequenzierungsplattform, dem Genominhalt und von experimentellen Schwankungen abhängen [83]. Einige dieser Fehler sind inhärent durch die Sequenzier-Bibliothek auf Grund der PCR Bedingungen, also der Anreicherung und der Aufbereitung des genetischen Grundmaterials im Labor zu Beginn des Prozesses.
In dieser Arbeit liegt der Fokus auf Sequenzierfehlern der Sequenzierplattformen, die zu Ersetzungen, Einfügungen und Löschungen in den Reads während des *Messens und Digitalisierens* führen. Weiterhin können Sequenzierer an Stelle eines der möglichen vier Nukleotide {`A,C,G,T`} ein `N` ausgeben, das angibt, dass ein Nukleotid vorhanden ist, aber die Art nicht sicher bestimmt werden kann. Während Ersetzungsfehler in einigen Plattformen, wie z.B. Illumina, dominant sind, so zeigen sich in anderen Plattformen, wie z.B. 454 und Ion Torrent, häufig homopolymer und carry-forward Fehler als multiple Einfügungen und Löschungen. Auf Grund der häufigen Verwendung der Illumina Plattformen zielen die meisten Fehlerkorrektur-Algorithmen bisher auf Ersetzungen ab. Auf Grund der neuen Technologien wie Ion Torrent existiert der Bedarf an Algorithmen, die Einfügungen und Lösungen finden und beheben. Bisher beschäftigten sich vorwiegend Chaisson et al.[7] und Salmela [64, 65] mit dieser Art von Fehlern. [94]

Die Entwicklung der Fehlerkorrektur hat ihren Ursprung in den Genom-Assemblierern, also den Algorithmen, die aus den Reads wieder ein Genom assemblieren ohne eine Referenzsequenz zu verwenden. Diese Algorithmen konnten auf Grund der NGS Daten und immer größeren zu sequenzierenden Genomen nicht mehr mit

den Fehlern zurecht kommen und integrierten daher erste Fehlerkorrekturen direkt in die Assemblierer. Erst später wurde die Fehlerkorrektur als eigenständige Disziplin durch alleinstehende Algorithmen und Programme, die vor der Assemblierung laufen, weiterentwickelt.

Hintergrund ist, dass die Qualität der Ergebnisse und teilweise auch die Laufzeit- und Speicherperformanz durch eine Vorverarbeitung der NGS Daten erreicht werden können. Im einfachsten Fall wird dies beispielsweise durch den simplen Ansatz deutlich, in dem man Reads im hinteren Bereich in Abhängigkeit zum Quality Score kürzt, da hier meist die Reads besonders fehlerhaft sind. Dies führt allerdings zum Verlust von u.U. wichtigen Informationen. [94]

Etwas genauer bedeutet dieses Vorgehen, dass oft Reads am 3' Ende gekürzt werden, da hier die Fehlerwahrscheinlichkeit stark steigt (zu 3' Ende siehe 2.2.1). Korrigiert man aber stattdessen die Fehler in dem potentiell zu kürzenden Teil zuerst und assembliert dann mit längeren Reads, führt dies zu einer Verbesserung der Alignbarkeit und damit des Assemblies [78],[75].

Grob betrachtet handelt es sich um zwei verschiedene Ansätze zur Fehlerkorrektur von Reads, die unterschieden werden müssen. Zum einen Algorithmen für das Base-Calling (z.B. [16, 63, 32]) (siehe 2.3), also die Interpretation eines Lichtblitzes, einer Fotografie oder der Lichtintensität und der Ableitung einer Base daraus. Zum anderen der Error-Correction, also der Korrektur von Basen, die bereits per Base-Calling entstanden sind. In dieser Arbeit liegt der Fokus auf der Error-Correction.

Nach einer Auflistung der bestehenden Algorithmen zur Fehlerkorrektur folgt eine Beschreibung des Algorithmus, der entwickelt wurde. Hierbei wird zunächst der Algorithmus SHREC, auf dem der entwickelte Algorithmus basiert, beschrieben. Darauf folgt eine Analyse von Verbesserungspotential des SHREC Algorithmus. Anschließend wird der entwickelte Algorithmus vorgestellt. Final folgt eine Analyse der Ergebnisse.

Einige in diesem Abschnitt der Arbeit entwickelten Konzepte und Ideen wurden unter anderem in der betreuten, studentischen Arbeit von Franziska Grohmann [23] umgesetzt, weiterentwickelt und hinsichtlich ihrer Machbarkeit evaluiert.

5.2 Bestehende Algorithmen

Es existiert eine Vielzahl von Fehlerkorrektur-Algorithmen, die sich grob in zwei Vorgehensweisen einteilen lassen. Die erste Gruppe von Algorithmen arbeitet mit einem multiplen Alignment, in dem mehrere Reads zueinander ausgerichtet werden, wobei eine Bewertungsfunktion für Unterschiede und Übereinstimmungen verwendet wird. Die Zweite ist die Gruppe der Algorithmen, die mit spektralem Alignment arbeiten. Beim spektralen Alignment werden zuerst Subsequenzen anhand ihrer Vorkommenshäufigkeit im Readset als valide oder nicht valide bewertet. Dann werden Reads anhand dieser validen Subsequenzen bewertet, bzw. so korrigiert, dass sie durch valide Subsequenzen abgebildet werden können.

Eine gute Übersicht und Evaluation der einzelnen Verfahren mit jeweils verschiedenen Eingabedaten ist in [94] zu finden.

Es folgt eine kurze Vorstellung der Algorithmen, die zum Zeitpunkt der Veröffentlichung dieser Arbeit die größte Bedeutung haben. Anzumerken ist, dass dieses Forschungsgebiet nach Beginn dieser Arbeit sehr intensiv in der Community parallel zu dieser Arbeit erforscht wurde. Es wird hierauf in der Diskussion am Ende dieses Abschnittes eingegangen.

Der SHARCGS Algorithmus [12] verwendet eine Vorfilterung der Reads, um fehlerhafte Reads auszuschließen. Es werden nur Reads zugelassen, die mindestens n mal von dem Sequenzierer generiert wurden und für die andere Reads existieren, die mit diesem Read überlappen. Diese Methode ist sehr einfach und schnell, aber nur sinnvoll, wenn sehr kleine Fehlerraten zu erwarten sind und eine sehr hohe Coverage vorhanden ist. [75]

Beispiele für Korrekturverfahren, die das einfache multiple Alignment verwenden sind MisEd [82] und der Vorverarbeitungsschritt von Arachne [6]. Beide Ansätze berechnen ein multiples Alignment der Reads und erkennen Fehler in den Spalten des Alignments. Leider sind diese Alignments sehr rechenintensiv, so dass diese Verfahren bei den Datenmengen von aktuellen Sequenzierern nicht mehr effizient sind. [75]

Der Coral Algorithmus [65] arbeitet ebenfalls mit multiplem Alignment. Im Vergleich zu den anderen Algorithmen bezieht er die Informationen der benachbarten

Basen in Form derer Qualityscores ein. Der Algorithmus kann auf verschiedene Sequenzierplattformen angepasst werden. Dies bedeutet, dass die Eigenheiten der sequenzierten Reads, die sich in der Verteilung und Wahrscheinlichkeit sowie der Art der Fehler äußern, angepasst werden können. Der Benutzer kann also das Fehlermodell selbst definieren. [65]

Der Euler-SR und Euler-USR Assemblierer sind Weiterentwicklungen des weit verbreiteten Euler Shotgun Assemblierers [59], der für sehr große Mengen von Short Reads entwickelt wurde. Sie enthalten einen Fehlerkorrekturschritt, der auf spektralem Alignment basiert [7], [75].

Reptile [95] ist ein Algorithmus, der ein Spektrum von k-meren (siehe 2.4.2) aus den Reads erzeugt. Mit Hilfe des Spektrums werden fehlerhafte (seltene) k-mere anhand der Hammingdistanz (siehe 2.4.3) überprüft und unter Umständen korrigiert, wobei benachbarte k-mere im Falle einer nötigen Korrektur in Verbindung mit Qualityscores bestätigen, ob eine Korrektur sinnvoll ist. Da nur das Spektrum gespeichert wird, ist es nicht nötig, die gesamten Daten im Speicher zu halten. [95]

Der HiTEC Algorithmus [29] ist vergleichbar mit Reptile. Er berechnet für jede Sequenz einen Wert, der für die Korrektheit der Sequenz steht, wobei dieser Korrektheitswert mit statistischen Verfahren bestimmt wird und ähnlich den Gewichten im Suffixbaum von SHREC (s.u.) ist, wobei eine andere Datenstruktur verwendet wird. Das Korrekturverfahren arbeitet wiederum mit den Wahrscheinlichkeiten der Sequenzen.[29]

Der Algorithmus SHREC [75] ist ein sehr schneller und skalierbarer Algorithmus, der weder per multiplem noch spektralem Alignment arbeitet. Er erstellt und arbeitet mit einem verallgemeinerten Suffixbaum, in dem die Read-Daten gespeichert werden. Durch eine Analyse der Knoten des Baumes werden seltene, unterrepräsentierte Knoten identifiziert und eine Fehlerkorrektur durchgeführt. Eine Erweiterung stellt Hybrid Shrec [64] dar, der es ermöglicht, auch Color Space Reads zu verarbeiten.

Der in diesem Teil der Arbeit entwickelte Algorithmus basiert auf SHREC. Entsprechend findet sich im folgenden Kapitel eine ausführlichere Beschreibung des Algorithmus.

5.3 Der SHREC Algorithmus

Im folgenden Abschnitt wird beschrieben, wie auf Basis des SHREC Algorithmus ein neuer Algorithmus entwickelt wurde. Hierzu wird vorerst der SHREC Algorithmus erklärt, anschließend folgt eine Herleitung des Verbesserungspotentials in 5.4.

5.3.1 Definitionen

Im Folgenden sei G die Länge des Genoms, l die Länge eines Reads, n die Anzahl der Reads eines Readsets und e die Fehlerrate eines Readsets.

5.3.2 Suffixbaum

Da die zentrale Datenstruktur des SHREC Algorithmus ein Suffixbaum ist, folgt eine Einführung in den grundlegenden Aufbau eines Suffixbaumes.

Um einen Suffixbaum $\tau(s)$ einer Zeichenkette s zu erhalten, nimmt man alle Suffixe dieser Zeichenkette (s. Abbildung 5.1).

Zeichenkette: `TAATA`

Suffixe:
`TAATA`
`AATA`
`ATA`
`TA`
`A`

Abb. 5.1: Suffixe einer Zeichenkette

Diese Suffixe fügt man in einen Baum ein, wobei zeichenweise durch den Baum geschritten wird und jede Kante mit der Zeichenkette beschriftet wird, die zu der Kante führt, d.h. durch Konkatenation der Zeichen. In Abbildung 5.2 ist ein Beispiel zu sehen.

Die besondere Eigenschaft eines Suffixbaumes ist, dass in ihm mit linearen Aufwand Zeichenketten (Infixe) von s gefunden werden können, indem der Suffixbaum durchgegangen wird.

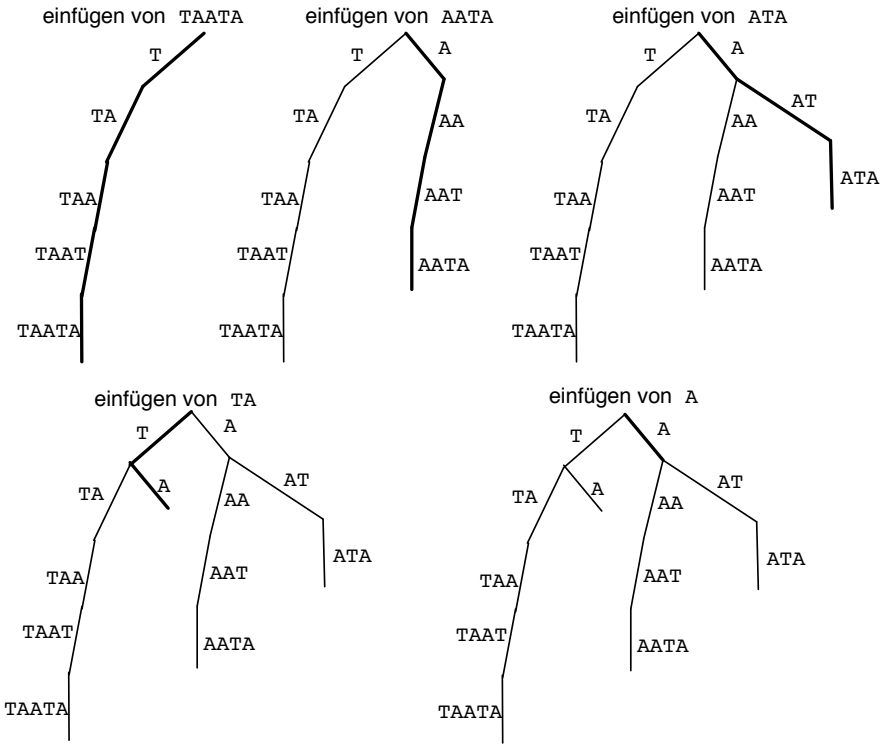

Abb. 5.2: Beispiel eines Suffixbaumes der Zeichenkette TAATA

Ein Suffixbaum kann auch mit mehreren Zeichenketten *gefüllt* werden. Soll die ursprüngliche Zeichenkette wieder identifiziert werden, muss bei jedem Einfügen eines Suffixes eine Verknüpfung (z.B. durch eine laufende Nummer) an dem Knoten, hinter dem die Zeichenkette endet, abgespeichert werden.

5.3.3 Der Algorithmus

Die folgenden Erläuterungen basieren auf Analysen des Quellcodes sowie der Beschreibungen in [75] und [23]. Eine Quellcodeanalyse war notwendig, da die Implementierung des Algorithmus von dem veröffentlichten Algorithmus abweicht.

Der Algorithmus erstellt einen Suffixbaum aus den Suffixen der Reads und deren reversen Komplementen. Sind Fehler in einem Read vorhanden, so ist die Wahrscheinlichkeit groß, dass Pfade zu diesen Suffixen unterrepräsentiert sind und erkannt sowie korrigiert werden können.

In Abbildung 5.3 ist zu sehen, wie ein Pfad unterrepräsentiert ist und korrigiert werden kann.

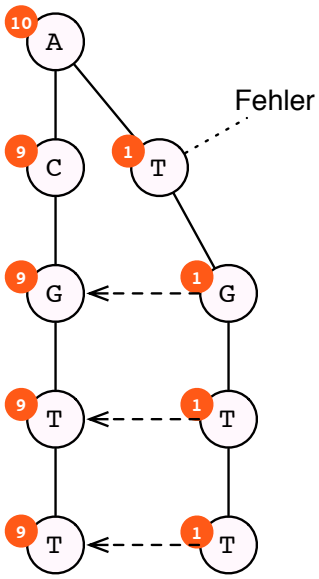

Abb. 5.3: Beispiel eines Baumes in SHREC mit einem unterrepräsentierten Pfad
Die roten Zahlen geben die Gewichte der Knoten, also die Anzahl der Suffixe an, die durch den jeweiligen Knoten gingen.

Im Detail sieht der Algorithmus wie folgt aus:

1. Suffixbaum erstellen

- Es werden alle Suffixe der Reads sowie deren reverse Komplemente in den Suffixbaum eingefügt.
- Für jedes Suffix im Baum wird ein eindeutiger Bezeichner eines Reads (z.B. eine laufende Nummer) und dessen Ausrichtung (normal oder reverses Komplement) gespeichert.
- Für jedes Blatt im Baum wird ein Gewicht gespeichert. Dieses Gewicht entspricht der Anzahl der Suffixe, die durch diesen Knoten liefen.

2. Potentielle Fehler markieren

- Alle Blätter des Suffixbaumes werden zwischen Tiefe g und h nach möglichen Fehlern untersucht. Ein Fehler ist dann potentiell vorhanden, wenn das Gewicht m' eines Knotens v_i kleiner ist der Erwartungswert des Gewichtes der Ebene $E(m)$ minus der x-fachen Standardabweichung. g, h, x sind hierbei vom Benutzer angegebene Parameter. $m' < E(m) - x\sigma(m)$, mit

  $E(m) = \frac{na}{G}$, $\sigma(m) = n(\frac{a}{G} - \frac{a^2}{G^2})$, $5 \leq x \leq 7$,

  $a = l - m + 1$,

  a entspricht der Anzahl an Suffixen unterhalb der betrachteten Ebene. Für jeden der n Reads fallen daher a Suffixe an, die sich auf G richtige Pfade aufteilen.

  Die potentiellen Fehler werden in den Knoten markiert. Abbildung 5.4 zeigt ein Beispiel.

- Hat ein Knoten v_i mindestens einen korrekten und einen fehlerhaften Kindknoten, so wird v_i zur weiteren Analyse markiert, siehe Abbildung 5.4.

3. Fehlerkorrektur durchführen

- Die zur weiteren Analyse markierten Knoten v_i werden durchgegangen. Hierbei wird analysiert, ob ein Kindknoten von v_i so korrigiert werden kann, dass dessen Teilbaum in einen anderen Teilbaum eines anderes Kindes von v_i komplett hineinpasst. Ist dies möglich, werden alle Suffixe der zu dem Teilbaum gehörenden Reads aus dem gesamten Suffixbaum entfernt, die Korrektur wird durchgeführt und die Suffixe werden wieder hinzugefügt, damit eine weitere Fehlerkorrektur ggf. wieder möglich ist. In Abbildung 5.4 ist der rechte Pfad `TGTT` zu `CGTT` korrigierbar.

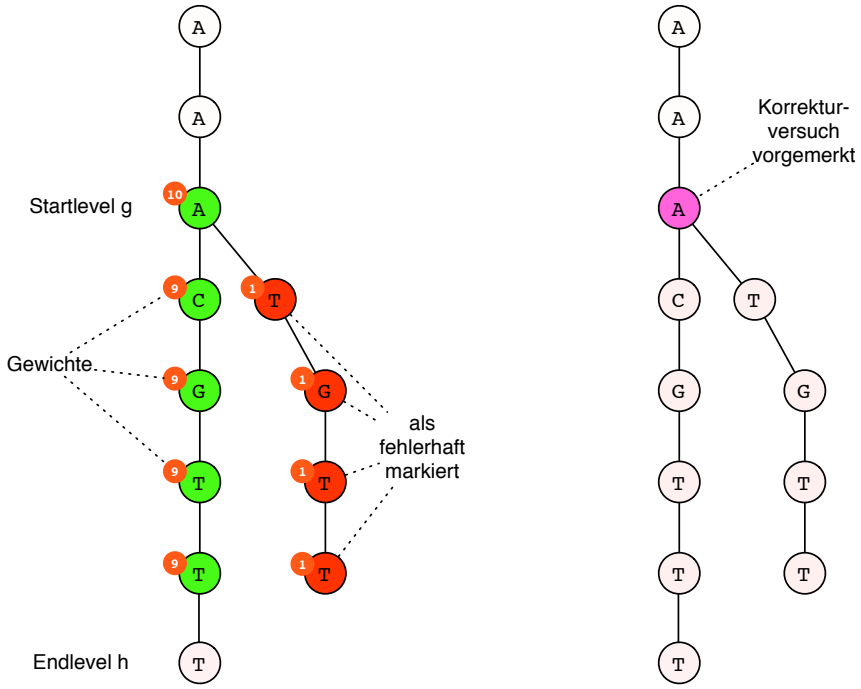

Abb. 5.4: Beispiel von Markierungen innerhalb des SHREC Algorithmus
Die kleinen Zahlen in weiss auf rotem Kreis in den Ecken der Knoten, links oben, geben die Gewichte an. Die grünen Knoten sind korrekte Knoten, die roten Knoten sind fehlerhafte (unterrepräsentierte) Knoten. Der lila Knoten ist zur Korrektur vorgemerkt.

4. Ausschreiben der Reads

- Die Ausgabe der Reads erfolgt, wobei die potenziell als fehlerhaft markierten Reads in der Ausgabe markiert werden und korrigierte Reads ausgegeben werden.

5.3.4 Eigenschaften des Baumes

Folgende Eigenschaften des Suffixbaumes lassen sich bei der Fehlerkorrektur und Umsetzung des Algorithmus zu Nutze machen. Es wird in den folgenden Abschnitten entsprechend auf diese Punkte eingegangen:

1. In den obersten t Ebenen des Suffixbaumes mit $t = min(log_4(G), log_4(n))$ ist der Baum fast vollständig, d.h., dass fast alle Knoten vier Kinder ha-

ben. Betrachtet man eine Ebene des Baumes t, dann hat diese 4^t mögliche Zeichenketten/Knoten, die in diese Ebene führen. Um zu wissen, in welcher Ebene t alle möglichen Zeichenketten mindestens einmal vorhanden sind, sucht man die Anzahl der Zeichenketten, die nötig sind, um den Baum bis zur Tiefe t komplett zu füllen. Da das Genom G Zeichen lang ist, enthält das Genom somit $O(G)$ Zeichenketten (der Länge l, $l << G$), die in den Baum eingefügt werden. Es ist dann $4^t \leq G$ bzw. $t \leq log_4(G)$. Ist aber die Anzahl der Reads n kleiner als die Anzahl der Basen des Genoms G, so müssen nicht alle Zeichenketten aus G vorkommen und die Anzahl der Reads ist maßgebend für die Füllung des Baumes, also $4^t \leq n$ bzw. $t \leq log_4(n)$. Anmerkung: Diese Abschätzung ist sehr grob, da selbst bei der angenommenen Gleichverteilung aus der Binomialverteilung dieses Experiments mit $B(0|min(G,n), \frac{1}{G})$ folgt, dass ein Teil der Positionen von G nicht überdeckt wird.

2. Für $l << G$ wird angenommen, dass für eine beliebige Zeichenkette der Länge l gilt, dass diese in einer anderen zufälligen Zeichenkette der Länge G mit einer Häufigkeit von $\frac{G}{l^4}$ als Erwartungswert vorkommt. Eine zufällige Zeichenkette der Länge G enthält $O(G)$ Zeichenketten der Länge l, mit $l << G$. Es gibt bei einem Alphabet mit vier Zeichen 4^l Zeichenketten der Länge l. Geht man von einer Gleichverteilung der Zeichen und Zeichenketten aus, so *teilen* sich alle G vorkommenden Zeichenketten auf die 4^l möglichen Zeichenketten gleichmäßig auf.

3. Bei einer Fehlerrate von e und genügend großer Coverage gilt, dass unterhalb der Ebene $log_4(e * G)$ in jeder Ebene mindestens $e * G$ der Knoten mehr als ein Kind haben. Dies folgt, da es in einem Genom der Länge G maximal G verschiedene Subsequenzen gibt, die zu G Pfaden im Suffixbaum führen. Bei einer Fehlerrate von e gibt es entsprechend $e * G$ abweichende Pfade.

4. Je näher man an der Wurzel des Baumes ist, desto weniger Sicherheit besteht, dass ein Knoten zu einem fehlerhaften Read führt. Sicherheit bezeichnet hier die Aussagekraft, ob von einem Knoten v_i ein Geschwisterknoten v_j (Knoten mit selbem Elternknoten) einen korrekten Pfad beschreibt oder v_i selbst, da die Häufigkeiten, mit denen die Knoten besucht werden, zur Wurzel hin stetig steigen.

In den obigen Betrachtungen wurde nicht berücksichtigt, dass die Wahrscheinlichkeit besteht, dass eine Zeichenkette der Länge eines Reads mehrmals an verschiedenen Positionen der DNS vorkommt. Dies würde dazu führen, dass entsprechend Suffixe der Reads mehrfach in der DNS vorkommen und somit in den Suffixbaum eingefügt werden. Die Fehlerkorrektur findet allerdings ausschließlich in den Bereichen statt, die sehr weit unten im Baum sind. Das mehrfache Vorkommen ist somit möglich, aber auf Grund der Tiefe im Baum und der damit einhergehenden hohen Sequenzlänge sehr selten. Auf der anderen Seite existieren in der DNS lange Repeat-Regionen, die diese Eigenschaften haben. Da diese Regionen von DNS zu DNS unterschiedlich sind, ist es nicht möglich, diese Eigenschaften in die obigen Formeln aufzunehmen, bzw. überrepräsentierte Pfade auszuschließen.

5.3.5 Aufteilung des Suffixbaumes

Der Suffixbaum hat eine Tiefe von l (Readlänge) und enthält schon $G - l \approx G$ Einträge bei einer Fehlerrate von null.

Die Anzahl der fehlerhaften Reads f ergibt sich mit $f = n(1 - B(0|e, l))$. Die Anzahl der Knoten wächst also in $Gn * (1 - B(0|e, l))$.

Um exemplarisch eine untere Schranke für die Anzahl der Knoten bei üblichen Daten des Humangenoms zu erhalten, seien $e = 0,001$ und $l = 50$ jeweils sehr niedrig gewählt. Sei $G \approx 3 * 10^9$ (konstant) und $n \approx 10^8$ auch als untere Schranke niedrig gewählt. Der Baum würde nun $\approx 10^{16}$ Knoten haben und es wird deutlich, dass er bei 4Byte pro Knoten nicht mehr speicherbar wäre ($\approx 10^4$TB).

Mit dem Wissen aus 5.3.4, dass die oberen Knoten jeweils vier Kinder haben und damit voll besetzt sind, wodurch eine Korrektur in dem Bereich nicht möglich ist, lassen sich Lösungen für das Speicherproblem finden. Durch eine Aufteilung des Baumes in mehrere Teilbäume mit jeweils konstantem, aber unterschiedlichem Präfix der Länge p, kann der Algorithmus an den verfügbaren Speicher angepasst werden. Ein Präfix ist hierbei die Knotenfolge von der Wurzel bis zum Knoten der Tiefe p. Führt man den Algorithmus also für alle Präfixe mit den dazugehörigen darunter hängenden Bäumen durch, ist dies äquivalent zur vorherigen Vorgehensweise.

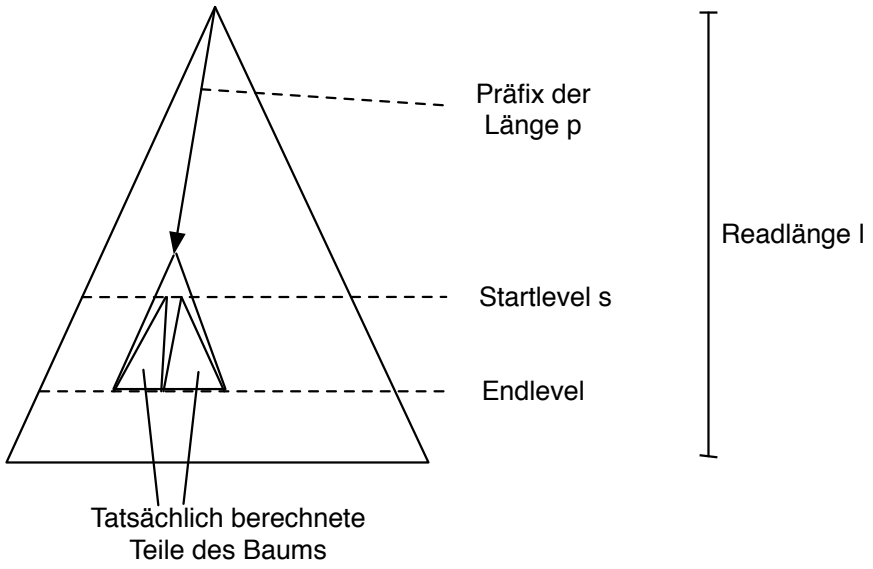

Abb. 5.5: Reduktion und Aufteilung des Suffixbaumes in Teilbäume.
Abb. nach [75]

Ein weiterer Vorteil ist, dass die Berechnung parallelisiert werden kann. So ist es möglich, dass Prozessoren oder mehrere Kerne eines Prozessors benutzt werden können.

5.3.6 Speicherreduktion des Suffixbaumes

Aus 5.3.4 folgt auch, dass der Baum weiter unten sehr dünn ist und nur noch einzelne Pfade enthält. In diesem Bereich sind keine Korrekturen auf Grund der niedrigen Gewichte möglich: Sehr lange Suffixe kommen sehr selten bei moderater Coverage vor, da ein Read sehr selten genau ein zweites Mal sequenziert wird.

Ein weiteres Problem sind Reads, die mehr als einen Fehler enthalten. Selbst wenn die Position des Reads gefunden wurde, weichen die im Suffixbaum tiefer gelegenen Unterbäume von dem Unterbaum eines Geschwisterknotens, auf Grund eines weiteren Fehlers, ab und eine Korrektur kann nicht erfolgen. Durch eine Abschwächung der Übereinstimmungsbedingung der Bäume, kann man dies umgehen: Ein Baum kann dann korrigiert werden, wenn eine definierte Tiefe im

Baum unter dem Fehler mit einem Geschwisterknoten identisch ist (und nicht der gesamte Unterbaum).

5.3.7 Technische Umsetzung

Technisch wird die Speicherung der Teilbäume über eine Hashtabelle (siehe 2.4.6) gelöst, die auf die jeweiligen Teilbäume Referenzen hält.

Genau bedeutet dies, dass der Bereich aus Abbildung 5.5 über dem Startlevel s und unter dem Ende des Präfixes einzeln im Speicher gehalten wird. Siehe hierzu Abbildungen 5.6, 5.7. Die dort abgebildeten dickeren Linien sind Zeichenketten der Länge $s-p$. Diese werden als Schlüssel in eine Hashtabelle gelegt und verweisen dann auf einen Baum, der aus Knoten und Kanten besteht. Der Vorteil dieser Vorgehensweise ist, dass beim Iterieren der Reads nach Entfernen des Präfixes schnell über die Zeichenkette auf den zu bearbeitenden Baum zugegriffen werden kann.

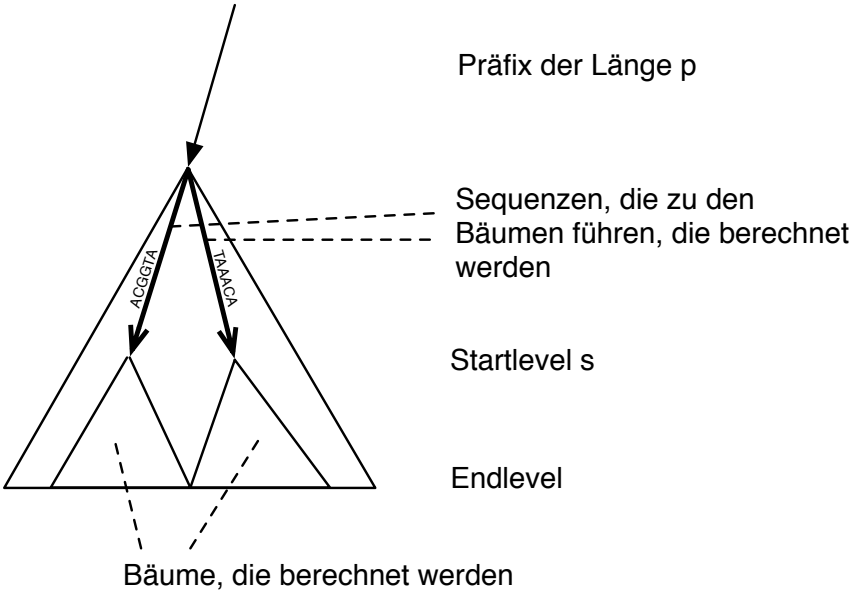

Abb. 5.6: Implementierung des Baumes, Abb. nach [75]

Des Weiteren werden Korrekturen nur direkt in der ersten Ebene des Teilbaumes vollzogen.

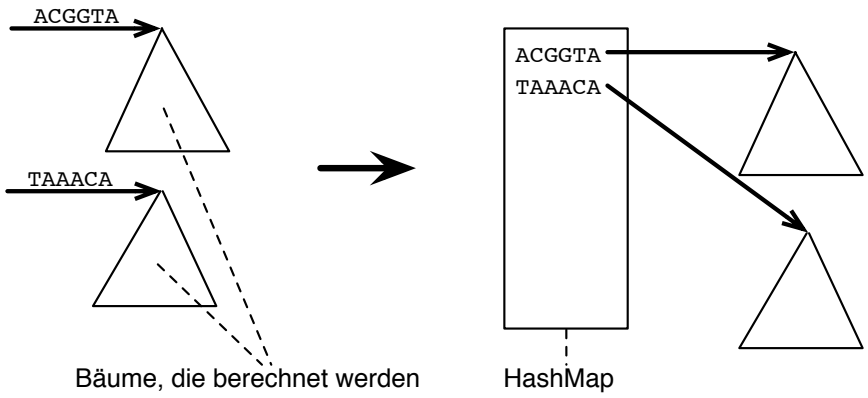

Abb. 5.7: Hashtabelle zum Zugriff auf die Bäume

Da die Genomlänge G nicht immer bekannt ist, wird die Gewichteberechnung sowie die Berechnung der Korrekturebene durch fixe Parameter ersetzt, die der Benutzer angeben muss. Dies ist eine sehr große Einschränkung, da je nach Readdaten nichtintuitive Werte erforderlich sind.

5.4 Analyse des Optimierungspotentials des SHREC Algorithmus

Es wurde analysiert, welches Verbesserungspotential der Algorithmus beinhaltet. Hierzu wurden die einzelnen Abschnitte des Algorithmus genau untersucht. Des Weiteren wurde recherchiert, inwieweit andere Veröffentlichungen Verbesserungspotential aufzeigen.

5.4.1 Eingabeformate

Ein sehr großer Nachteil von SHREC ist, dass er nur Reads gleicher Länge (z.B. von Illumina) zulässt. Dies führt dazu, dass Reads von anderen Plattformen, wie z.B. von 454, nicht verwendet werden können oder stark beschnitten werden müssen.

In [64] wurde SHREC von Salmela so erweitert, dass Colorspace Reads und Readsets aus verschiedenen Sequenziertechnologien verwendet werden können.

5.4.2 Parameterwahl

Das erwartete Gewicht der Knoten, das in der Veröffentlichung hergeleitet wird, wird praktisch in der Implementierung nicht verwendet, da die Genomlänge nicht bekannt ist. Dies führt dazu, dass der Benutzer einen nicht intuitiven, aber essentiellen, Wert manuell eingeben, bzw. viele verschiedene Werte *ausprobieren* muss. Hierbei muss er die eigentliche, meist sehr zeitintensive, Anwendung mit den Reads mehrfach ausführen, bis fest steht, welches *die besten* Parameter waren. Man könnte dies allerdings beheben, indem man wenigstens die erwartete Genomlänge oder eine Größenordnung abfragen würde, wie es in ECHO [31] vollzogen wird. Diese ist in der Regel bekannt, außer es wird mit gemischten Readsets, also Readsets bestehend aus verschieden DNS mit verschiedenen Längen, gearbeitet. Dieser Fall ist aber für diese Art von Fehlerkorrektur ohnehin nicht anwendbar, da der Suffixbaum in diesem Fall mehrere Gebiete hätte, in denen Fehler korrigiert werden müssten, was somit nicht eindeutig wäre.

Die Parameterangabe wird auch in anerkannten Veröffentlichungen kritisiert. In [34] wird beschrieben, dass mehrere verschiedene Eingabewerte ausprobiert wurden, um die Schwelle zwischen korrekten und fehlerhaften Reads zu unterscheiden, allerdings seien die Parameter dabei sehr sensitiv gewesen und dies sei aus Sicht der Autoren für Benutzer eine kritische Limitierung des Programmes.

Der in SHREC mit *strictness* bezeichnete Parameter funktionierte laut [31] mit Standardwerten für passende Eingabedaten nicht, so dass die Autoren willkürlich den kleinsten, gerade noch von SHREC akzeptierten Wert nehmen mussten. Die Autoren von ECHO schreiben, dass die Parameter auch automatisch berechenbar wären [31] (s.o.).

In einer weiteren Veröffentlichung des Autors von SHREC in [65] wird leichte Selbstkritik laut. Es wird geschrieben, dass SHREC schlechtere Ergebnisse liefere als der in [75] beschriebene Algorithmus, allerdings sei der *Parameter Raum* nicht vollständig ausgeschöpft worden, weshalb das gesamte Potential u.U. nicht erreicht worden sei.

5.4.3 Speicherbedarf

In der Veröffentlichung von Quake[34] wird auf den Speicherverbrauch von SHREC eingegangen, wobei die Autoren anmerken, dass SHREC für große Datensätze (325 Million Reads mit 124bp) mehr als 256 GB Arbeitsspeicher benötige, was eine Ausführung unmöglich mache.

Auch in [95] wird auf die Speicherproblematik in vergleichender Weise hingewiesen. Reptile benötige 8-11 mal weniger Speicher als SHREC.

Der Coral Algorithmus verwendet nach [65] 3-10 Mal weniger Speicher als SHREC.

5.4.4 Laufzeit

Viele Laufzeitvergleiche beziehen SHREC mit ein. So wird in [95] erwähnt, dass Reptile 3-10 mal weniger Zeit benötige.

Bei einer Genomsequenzierung von Hefe in [31] bemerken die Autoren, dass der SA Algorithmus 5 mal schneller sei als SHREC.

5.4.5 Analyse der Readverteilung

SHREC analysiert die Verteilung innerhalb des Baumes nicht, es wird auf einer fixen Ebene korrigiert, anstatt zu untersuchen, in welchen Teilen des Baumes welche Ebene optimal für Korrekturen ist. So wird in [31] geschrieben, dass ECHO fast jeden Read korrigiert. Er erreicht dies, indem mit Parametern gestartet wird, die eine Gleichverteilung der Reads annehmen, aber während der Laufzeit analysiert wird, ob sich der Algorithmus an einer Stelle mit deutlich höherer oder niedriger Coverage befindet.

5.5 Der Algorithmus zur Fehlerkorrektur

Der SHREC Algorithmus wurde zum Zeitpunkt der Erstellung dieses Abschnittes der Arbeit sehr viel zitiert und bekam entsprechend viel Aufmerksamkeit in der wissenschaftlichen Gemeinschaft. Er sollte weiterentwickelt und analysiert werden, da er auf Grund der hohen Parallelisierbarkeit viel Potential bietet.

In Verbindung mit der studentischen Arbeit [23] wurden einige zentrale Verbesserungsmöglichkeiten aus Kapitel 5.4 von SHREC auf Machbarkeit untersucht und implementiert. Die wichtigsten Ergebnisse hieraus werden in diesem Kapitel dargestellt, wobei Herkunftsangaben auf [23] nicht explizit markiert werden.

5.5.1 Erkennung der Eingabeformate

Um dem Benutzer die Verwendung zu vereinfachen, wurde SHREC so erweitert, dass er automatisch die Dateiformate der Reads erkennt. Dies geschieht anhand der eindeutig identifizierbaren Unterschiede im Kopf der Dateien sowie einer Datenanalyse in Bezug auf den Wertebereich bei den Qualityscores. Neben dem Komfort für den Benutzer sinkt die Fehleranfälligkeit deutlich, da der ursprüngliche SHREC Algorithmus ohne Eingabevalidierung entwickelt wurde und somit ggf. *nicht-Read-Daten* als Reads interpretiert hat.

5.5.2 Automatische Bestimmung der Programmparameter

Automatische Parallelisierung und sequenzielle Abarbeitung

Eine sehr gute Möglichkeit zur Parallelisierung des Algorithmus ergibt sich, wenn durch verschiedene Präfixe iteriert wird, für welche dann jeweils der Suffixbaum aufgestellt wird. Der gesamte Suffixbaum wird also in mehrere Bäume zerteilt, die einzeln abgearbeitet werden. Eine derartige Zerteilung ist möglich, da keine Datenabhängigkeiten zwischen den kleineren Suffixbäumen existieren. Diese kleineren Suffixbäume können nun von mehreren Kernen gleichzeitig/parallel oder sequentiell abgearbeitet werden. Abbildung 5.8 illustriert diese Parallelisierung.

Zu sehen ist, dass die Recheneinheiten (Processing Units, PUs) jeweils zu einem fixen Präfix einen Baum berechnen. Die PUs können Kerne eines CPUs, GPUs oder einzelne Rechner sein.

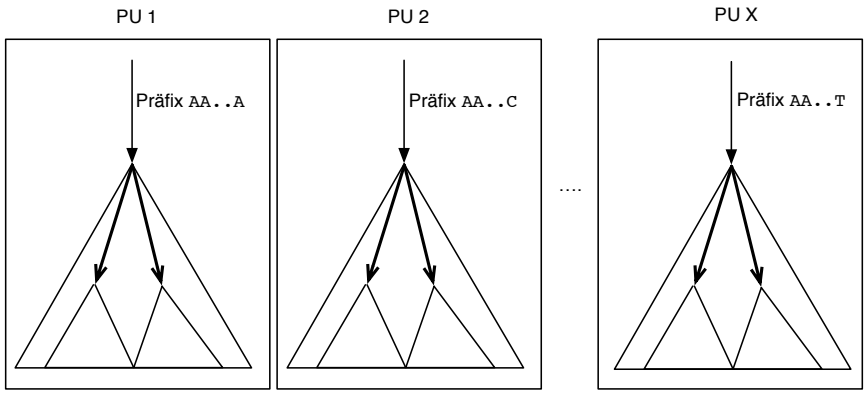

Abb. 5.8: Parallelisierung des Algorithmus

Der erste Teil der Parallelisierung ist die automatische Wahl der Anzahl der Threads, mit denen gearbeitet wird. Dies erfolgt durch ein Auslesen der Anzahl der Kerne der verwendeten CPUs.

Es wäre noch möglich, eine höhere Parallelisierung umzusetzen, indem ein vielfaches der Kernanzahl an Threads startet, da bei den meisten Rechnern das Laden von Daten aus dem Arbeitsspeicher sehr lange, im Vergleich zur eigentlichen Berechnung, dauert (Column Address Strobe Latency). Diese Ladezeiten können wiederum für Berechnungen verwendet werden, indem für jeden Kern mehrere Threads gestartet werden. Es wurde in diesem Fall nur ein Thread pro Kern gestartet, da je nach System dieses Verfahren nicht gut umsetzbar ist und je nach Betriebssystem und Scheduler auch zu längeren Wartezeiten führen kann, wenn die Ladezeiten sehr klein sind. Im Vergleich dazu ist in SHREC eine Parallelisierung vorgesehen, allerdings nur mittels eines fixen Parameters im Quellcode und einer entsprechenden Neukompilierung.

Da jeder Thread einen Teilbaum aufstellt und die Größe des Baumes bestimmend für den Speicherverbrauch ist, muss hier automatisch eine optimale Zuweisung erfolgen. Dies geschieht, indem der verfügbare Arbeitsspeicher in die Anzahl der Threads aufgeteilt wird. Aus der resultierenden Größe pro Thread könnte nun die

Präfixlänge, die die Größe des Teilbaumes bestimmt, abgeleitet werden. Aus der Gesamtanzahl der zu erwartenden Knoten G_K könnte man für ein Präfix der Länge p die durchschnittliche Anzahl an Knoten $\frac{G_K}{4^p}$ und damit den Speicherbedarf, ermitteln.

Da allerdings weder die Fehlerrate des Readsets, die Verteilung der Basen, noch die Genomlänge bekannt sind, ist dieser theoretische Ansatz nicht umsetzbar. In der Praxis wird daher die Präfixlänge bestimmt, indem für die Reads und zufällige Präfixe der Länge p jeweils ein Suffixbaum erstellt wird und daraus die durchschnittliche Verteilung der Knoten pro Ebene approximiert wird. Mit dem Wissen des Speicherverbrauches eines Knotens ist dann der Speicherbedarf für ein Präfix annäherbar und durch Anpassen von p optimal automatisch ermittelbar.

Automatische Fehlerkorrekturebenenbestimmung

Für die Fehlerkorrektur in SHREC ist es erforderlich, dass der Benutzer angibt, in welcher Ebene die Korrektur stattfinden soll und bis zu welcher Ebene die Suffixbäume bei einer Korrektur verglichen werden sollen. Hier ist nicht offensichtlich, warum es ausreicht, auf *einer* Ebene die Korrekturen vorzunehmen. Folgende Herleitungen nach [23] zeigen, warum dieses Vorgehen korrekt ist.

Lemma 1 *Für jedes $k \in \{0, ..., l-1\}$ gilt: Auf jeder Ebene $t \in \{0, ..., k\}$ im Suffixbaum $\tau(s)$, der die Suffixe von s enthält, kommt der Buchstabe s_k, k der Zeichenkette $s = s_0, ..., s_{l-1}$ vor.*

Beweis Gezeigt wird, dass die Annahme der Buchstabe s_k komme auf einer beliebigen Ebene t nicht vor, zu einem Widerspruch führt. Angenommen, die Behauptung wäre wahr, dann wäre kein Suffix von s in $\tau(s)$ eingefügt worden, das an der Stelle t den Buchstaben s_k hat. Daraus folgt, dass kein Suffix von s an der Stelle t den Buchstaben s_k hat. Sei nun $p = k - t$, dann ist $p \in \{0, ..., k\}$, da $t \in \{0, ..., k\}$. Das Suffix, das an der Stelle p startet, ist also nicht in $\tau(s)$, was ein Widerspruch wäre, da $\tau(s)$ Präfixbaum der Suffixe von s ist.

Sei s^{-1} reverses Komplement zu s, $\tau(s)^{-1}$ der Präfixbaum der Suffixe zu s^{-1} und $\tau(s, s^{-1})$ der Präfixbaum zu den Suffixen von s und s^{-1}.

Lemma 2 *Für jedes $k \in \{0, ..., l-1\}$ gilt. Auf jeder Ebene in $t^- \in \{0, ..., max(k, l-k-1)\}$ in $\tau(s, s^{-1})$ kommt s_k oder sein entsprechender Buchstabe in dem Reverse Komplement s_{l-k-1}^{-1} vor, wobei $s = s_0...s_{l-1}$ ist.*

Beweis Nach obigem Lemma gilt: s_k kommt in jeder Ebene $t \in \{0, ..., k\}$ in $\tau(s)$ vor. Entsprechend ist das reverse Komplement von s_k, also s_{k-l-1}^{-1} in s^{-1}. s_{k-l-1}^{-1} kommt also in $\tau(s^{-1})$ in jeder Ebene vor und kommt damit auch in $\tau(s, s^{-1})$ in jeder Ebene vor. s_k oder sein reverses Komplement kommt daher in den Ebenen $t^- \in \{0, ..., max(k, l-k-1)\}$ vor.

Satz 1 *Auf jeder Ebene $t^- \in \{0, ..., \lfloor \frac{l}{2} \rfloor\}$ in τ^- kommt jeder Buchstabe s_k oder sein Komplement s_{l-k-1}^{-1} vor, mit $k \in \{0, ..., l-1\}$*

Beweis Nach Lemma 2 folgt, dass ein Buchstabe s_k oder sein reverses Komplement s_{l-k-1}^{-1} auf allen Ebenen bis Ebene $t^- = max(k, l-k-1)$ vorkommt. Diese Aussage soll nun unabhängig von k getroffen werden, es muss also

$min_{k \in \{0,...,l-1\}}(max(k, l-k-1))$

bestimmt werden.

Betrachtet man diese Problemstellung als eine Suche des Minimums des Maximums zweier, sich schneidender linearer Funktionen, die in N^2 jeweils einer Gerade mit 45 Grad, respektive -45 Grad zur x-Achse, entsprechen, löst sich das Problem. Hier ist also das Minimum des Maximums erreicht, wenn sich die Geraden schneiden, also $k = l-k-1$, also $k = \frac{l}{2} - \frac{1}{2}$, da $k \in N$, folgt also, dass das Minimum des Maximums bei $\lfloor \frac{l}{2} \rfloor$ liegt.

Zusammenfassend ist es also nur möglich, bis zu Ebene $\lfloor \frac{l}{2} \rfloor$ Fehler zu korrigieren. Wie schon in Kapitel 5.3.4 erläutert, haben die Knoten in den ersten Ebenen jeweils vier Kinder und sehr hohe Gewichte. Es muss also ermittelt werden, wo die *Dichte* des Baumes gerade noch so hoch ist, dass die normalen Knoten hohe Gewichte haben, aber die Knoten im fehlerfreien Fall zum sehr großen Teil nur einen Kindknoten haben. Sinnvoll ist es also, die Ebene zu identifizieren, auf der die meisten Knoten genau zwei Kinder haben und für die Ebene gilt, dass diese kleiner gleich $\lfloor \frac{l}{2} \rfloor$ ist. Da die Verteilung der Sequenzen eines Readsets von dessen Größe, dessen Genom, der Fehlerrate, der Sequenziertechnik und vielen weiteren Faktoren abhängt, ist es nicht möglich, die Korrekturebenen zu berechnen.

Es bleibt aber die Option, aus einer repräsentativen Stichprobe auf die Gesamtheit zu schließen. Entsprechend wird im Algorithmus die Startebene empirisch, nach Bestimmung der Ebene mit den meisten Knoten mit genau zwei Knoten, ermittelt. Für die Endebene, bis zu der korrigiert wird, wird konstant $\frac{4}{3}$ Startebene gewählt. Dieser Wert wurde empirisch ermittelt.

5.5.3 Patricia Tries zur Speicherreduzierung

Die Ebenen der Suffixbäume, in denen die Korrektur stattfindet, sind so gewählt, dass der Unterschied zwischen den Gewichten der Bäume so ist, dass der Baum fast ausschließlich aus Pfaden besteht. Das bedeutet, dass der Baum sehr dünn ist, wie Abbildung 5.10 zeigt.

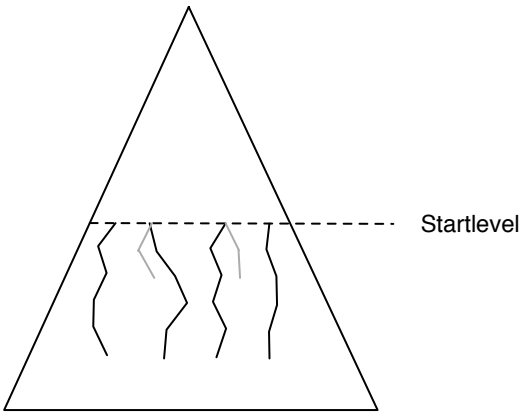

Abb. 5.9: Dünner Suffixbaum
Der Suffixbaum ist sehr dünn im Korrekturbereich und beinhaltet fast nur Pfade (schwarz). Fehler (grau) sind Pfade mit kleinem Gewicht und gehen von den Hauptpfaden ab.

Diese Pfade sind jeweils eine Folge von Knoten mit hohen Gewichten, also häufig *besuchte* Knoten, von denen nur wenige Pfade abweichen, die wiederum nur niedrig gewichtet, also sehr selten *besucht*, sind, da sie aus Fehlern entstanden sind. Da es sehr unwahrscheinlich ist ($e \approx 0,02$), dass ein Fehler im Vergleich zu einer korrekten Position häufig an der selben Stelle auftritt, sind diese abgehenden Äste sehr niedrig gewichtet. Der Suffixbaum unter dem gerade betrachteten Präfix ist also sehr dünn, aber tief. Diese Eigenschaft lässt sich ausnutzen, indem man

einen Patricia Trie zur Speicherung verwendet (s. 2.16), da Patricia Tries Knoten zusammenfassen, die nur bis zu zwei adjazente Kanten haben (siehe Abbildung 5.10). Diese Speicherersparnis geht nur mit einer rechenaufwändigeren Korrektur und Baumerstellung einher. Da allerdings der Speicherbedarf die zentrale Herausforderung ist und die Anzahl der zu korrigierenden Fehler im Vergleich zur Gesamteingabegröße der Daten sehr klein ist, wirkt sich dieser Nachteil nicht merkbar aus.

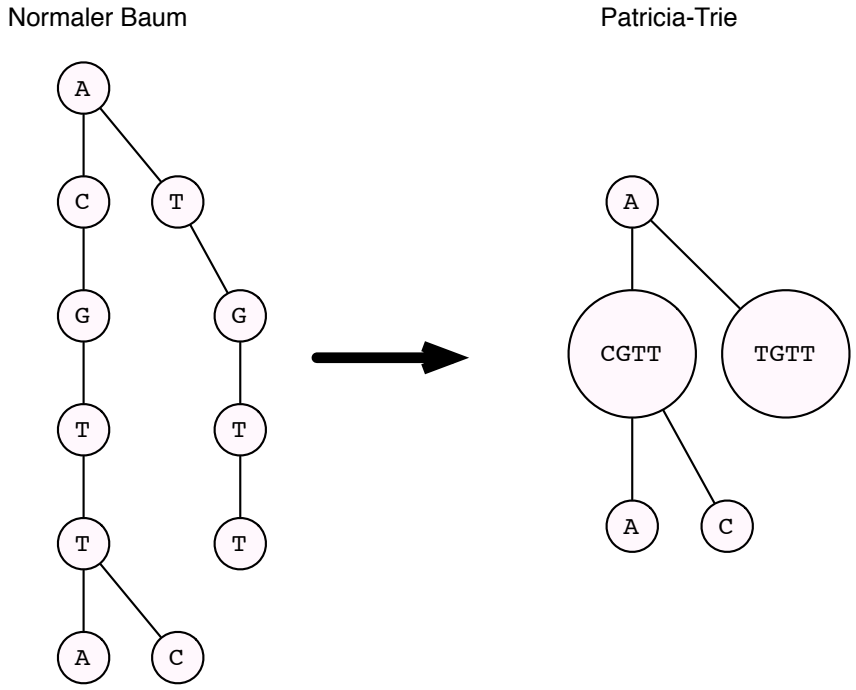

Abb. 5.10: Umwandlung eines normalen Baumes in einen Patricia Trie

5.6 Ergebnisse aus Messreihen

Es wurde die Reduzierung des Speichers durch die Patricia Tries realisiert. Um hier einen Vergleich mit SHREC zu ziehen, werden die Anzahl der Knoten und der davon linear abhängige Speicherverbrauch sowie die Knotengrößen für verschiedene Eingabedatengrößen in 5.6.1, verglichen.

Um zu zeigen, dass die Parameterwahl essentiell ist, wurde der SHREC Algorithmus mit verschiedenen Parametern evaluiert. Hierbei wurde betrachtet, wie sich die Fehlerrate nach Korrektur verhält. Kapitel 5.6.2 zeigt diese Ergebnisse.

Es ist bei einem natürlichen Readset nicht möglich, eine Aussage zu treffen, ob ein Datensatz nach einer Korrektur besser oder schlechter geworden ist, da nicht bekannt ist, wie die Fehlerrate ist und wo die Fehler genau lagen. Arbeitet man hingegen mit synthetischen Daten, kann man bei Generierung des Readsets abspeichern, wo Fehler hinzugefügt wurden. Mit Hilfe dieser Daten kann man nun evaluieren, welche Parameter zum optimalen Ergebnis führen. Hierbei ist die Annahme, dass das synthetische Readset weitgehend ähnliche Eigenschaften wie ein natürliches Readset hat, erforderlich. Da es für die Evaluation der Qualität eines Fehlerkorrekturalgorithmus nur notwendig ist, dass die Position der Fehler innerhalb eines Reads bekannt ist, wurde das Genom der Hefe (Saccharomyces cerevisiae S288c) genommen und hieraus Readsets generiert. Die Datenbasis ist also nicht rein synthetisch.

Im Rahmen dieser Arbeit wurden mit Hilfe dieses Vorgehens mehrere Readsets generiert, mit Hilfe derer empirisch die theoretischen Ergebnisse nachgewiesen werden.

Zuerst wird gezeigt, wie die Knotenanzahl durch die Verwendung von Patricia Tries reduziert werden kann, dann wird gezeigt, dass die automatische Parameterwahl deutlich bessere Ergebnisse in der Fehlerkorrektur liefert als die fixen Parameter des Basisalgorithmus SHREC.

Der im Rahmen dieser Arbeit entwickelte Algorithmus wird im Folgenden mit FIONA bezeichnet.

5.6.1 Ergebnisse zur Knotenreduzierung durch Patricia-Tries

Es wurden für die folgenden Auswertungen zehn synthetische Datensätze aus einer Sequenz des Genoms der Hefe (Saccharomyces cerevisiae S288c chromosome V) generiert. Tabelle 5.1 zeigt die Parameter der Daten.

Bezeichnung	**Readanzahl**
D1	100.000
D2	200.000
D3	300.000
D4	400.000
D5	500.000
D6	600.000
D7	700.000
D8	800.000
D9	900.000
D10	1.000.000
D11	1.500.000
D12	2.000.000

Tab. 5.1: Datensatzdefinitionen D1-D12
Daten wurden aus dem Genom Saccharomyces cerevisiae S288c chromosome V (NC_001137) mit einer Fehlerrate von 2% generiert. Die Readlänge beträgt 100bp. Das Genom hat eine Länge von 585.000bp.

Da die Anzahl der Knoten im Baum vom Readset abhängt, wurde der SHREC Algorithmus erweitert, damit die Anzahl der Knoten gezählt werden kann. Abbildung 5.11 zeigt, wie sich die Anzahl der Knoten mit den Datensätzen D1 bis D12 verhält. Es ist zu sehen, dass die Anzahl der Knoten linear in der Anzahl der Reads für beide Algorithmen wächst, allerdings ist die Steigung von FIONA deutlich niedriger.

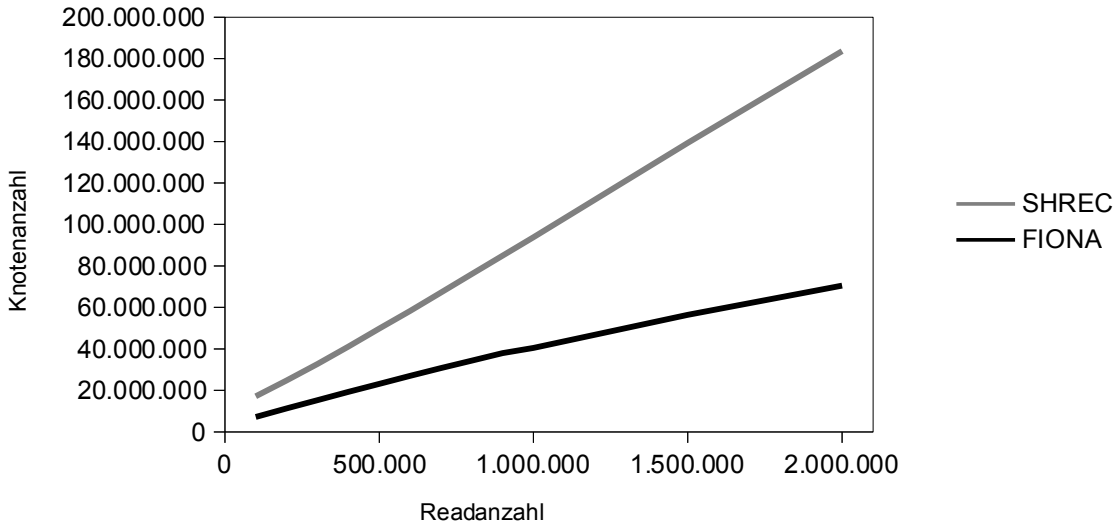

Abb. 5.11: Entwicklung der Knotenanzahl in Abhängigkeit zur Readanzahl
Entwicklung der Knotenanzahl in Abhängigkeit zur Readanzahl für die Algorithmen FIONA und SHREC. Datensätze D1 bis D12 wurden verwendet. Parameter: 2 Iterationen, Startlevel 20, Endlevel 24

5.6.2 Ergebnisse bei der automatischen Parameterwahl

Um zu zeigen, dass die theoretischen Überlegungen zur Wahl des richtigen Levels der Korrektur im Baum der vorherigen Abschnitte umsetzbar sind, werden wiederum synthetische Daten herangezogen. Um zu zeigen, dass die Parameterwahl von SHREC essentiell ist, wurden diese synthetischen Daten in Form einer Menge von Readdaten so generiert, dass sie auf den Parametern der Veröffentlichung von SHREC ([75], S.2157, Datensätze A1l, A2l, A3l, A1h, A2h, A3h) basieren. Tabelle 5.2 zeigt die Parameter der Daten.

Es wurde der SHREC Algorithmus mit verschiedenen Start- und Endleveln der Korrektur ausgeführt und die Fehlerrate nach der Korrektur ermittelt. Hierzu wurde SHREC mit Startleveln aus $\{6, 7, ..., 24, 25, 30, 35, 40, 45\}$ ausgeführt, wobei das Endlevel dem Startlevel + 4 jeweils entspricht. Dies wurde durchgeführt, um zu zeigen, dass die Wahl des Startlevels ausschlaggebend ist. In Abhängigkeit der Parameterwahl lässt sich somit die Fehlerrate nach Korrektur darstellen. Parallel dazu wurde die Parameterwahl von FIONA ermittelt. Abbildung 5.12 zeigt den Verlauf der Fehlerrate nach Korrektur und den Parameter, den FIONA

Bezeichnung	Readanzahl	Fehlerrate (konstant)
B1	200.000	1%
B2	200.000	2%
B3	200.000	3%
B4	400.000	1%
B5	400.000	2%
B6	400.000	3%

Tab. 5.2: Datensatzdefinitionen B1-B6
Daten wurden aus dem Genom Saccharomyces cerevisiae S288c chromosome V (NC_001137) generiert. Die Readlänge beträgt 100bp.

wählt. Die Abbildung zeigt in der gepunkteten Linie die Fehlerrate des Datensatzes ohne Fehlerkorrektur zum Vergleich. Es ist auffallend, dass die Fehlerrate über die Basisfehlerrate steigt, wenn ein zu kleines Startlevel gewählt wird. Die Ursache ist, dass bei kleinen Startleveln der Baum fast vollständig ist und ein abgehender Pfad von einem Pfad mit hohem Gewicht nicht bedeutet, dass der abgehende Pfad ein Fehler ist, sondern nur ein Pfad eines Suffix mit niedrigerer Coverage im Genom. Weiterhin ist zu sehen, dass die Fehlerrate hinter dem Minimum wieder ansteigt. Dies ist so zu interpretieren, dass sehr weit unten im Baum eher zufällig Knoten entstehen (vgl. Abb. 5.2) und die Fehlerkorrektur mit einem schlechten Signal-Rauschen-Verhältnis arbeiten muss.

Weiterhin ist auffallend, dass die Fehlerrate trotz doppelt so vieler Reads (B1 vs. B4) nicht besser, sondern eher schlechter wird. Grund ist, dass durch mehr Reads auch mehr Knoten entstehen, wodurch nicht mehr eindeutig entschieden werden kann.

Gut zu sehen ist, dass der von FIONA bestimmte Wert von Startlevel 15 sehr nahe am Optimum Startlevel 16 liegt, wobei in Abbildung 5.13 zu sehen ist, dass die Werte nahe des Optimums eher zufällig springen.

In Zahlen bedeutet dies, dass SHREC die Fehlerrate von 1% auf 0,0234% reduzieren kann, wobei die Parameterwahl von FIONA auf 0,0190% Fehlerrate (entspricht etwa 23% Verbesserung) nach Ausführung des SHREC Algorithmus, kommt.

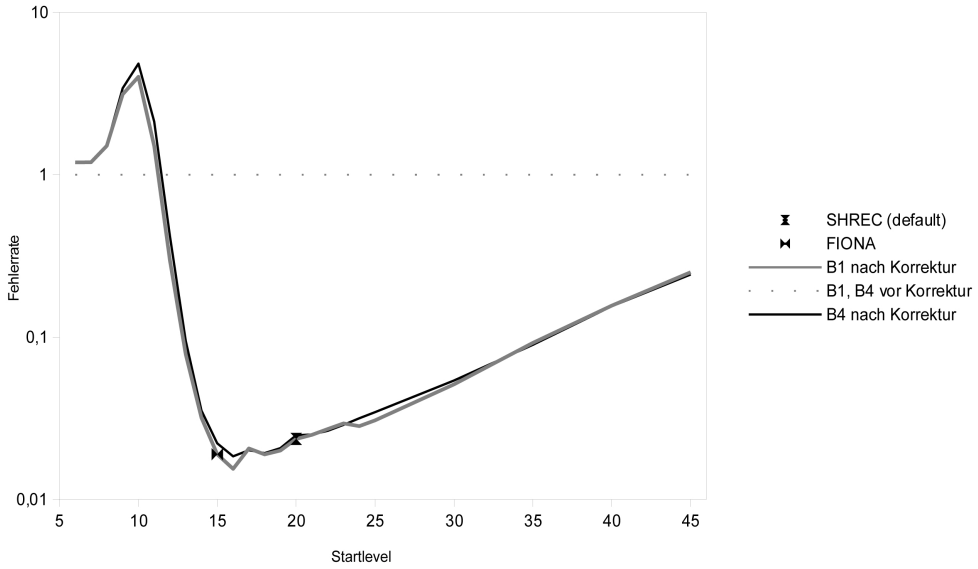

Abb. 5.12: Fehlerrate des Readsets B1,B4 nach Ausführung von SHREC
Fehlerrate des Readsets B1,B4 nach Ausführung von SHREC mit verschiedenen Startleveln. Der Defaultwert von SHREC von Startlevel 20 ist nahe des Optimums. FIONA wählt ein besseres Startlevel. Parameter: 2 Iterationen, Endlevel jeweils Startlevel + 4

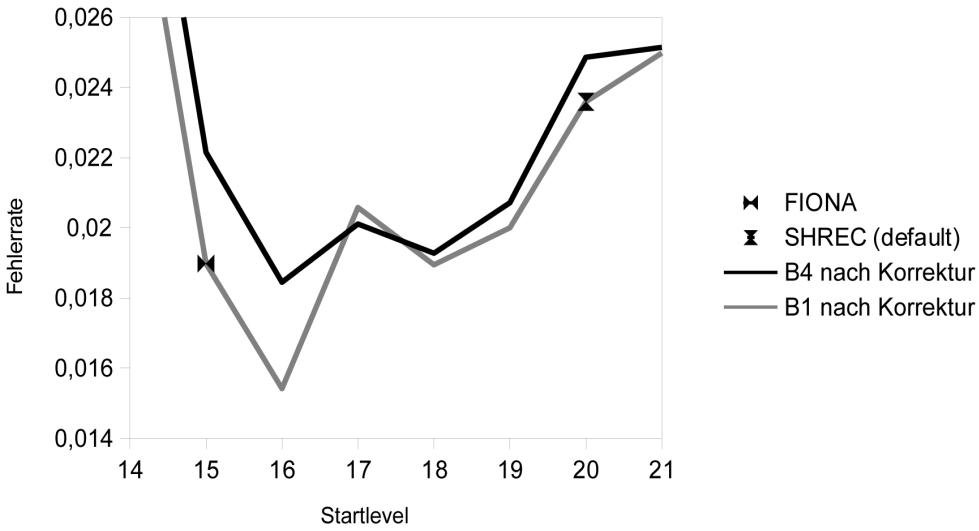

Abb. 5.13: Detailausschnitt von Abb. 5.12

5.7 Zusammenfassung und Ausblick

5.7.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde der Algorithmus SHREC auf Verbesserungspotential untersucht, wobei gezeigt wurde, dass das ermittelte Verbesserungspotential auch in einschlägigen Veröffentlichungen als notwendig angesehen wird.

Zur verbesserten Benutzbarkeit werden Eingabeformate automatisch erkannt. Eine automatische Parallelisierung erfolgt ebenfalls in Abhängigkeit zu der Architektur des Benutzers.

Um den Speicherbedarf von SHREC zu reduzieren, wurde das Prinzip der Patricia Tries angewendet. Es wurde gezeigt, dass diese Reduktion die Anzahl der Knoten und damit den hierfür nötigen Speicherplatz, sehr stark reduziert.

Es wurde die essentielle Parameterwahl von SHREC, die in der wissenschaftlichen Gemeinschaft als kritische Limitierung angesehen wird, analysiert. Hierbei wurde betrachtet, wie sich die Parameterwahl auf den Algorithmus auswirkt und einen Algorithmus entwickelt, der erst zur Laufzeit die Parameter in Abhängigkeit zu den Eingabedaten bestimmt, indem eine zufällige Teilmenge des Korrekturbereichs analysiert wird und auf dieser Basis optimale Parameter approximiert werden.

Um den Algorithmus zur Parametergenerierung mit den fixen Parametern für SHREC zu vergleichen, wurde SHREC auf verschiedenen Datensätzen ausgeführt, die den Datensätzen der Veröffentlichung zu SHREC vom Parameterumfang entsprechen.

5.7.2 Ausblick

Neben den tief gehenden Verbesserungen existieren noch einige Möglichkeiten, den SHREC Algorithmus zu verbessern, bei denen es nicht möglich war, diese im Rahmen dieser Arbeit umzusetzen. Viel Potential steckt in der Verbesserung des Kriteriums, das entscheidet, ob an einem konkreten Knoten eine Korrektur durchgeführt werden soll oder nicht. Liegt also eine Abzweigung im Baum mit

zwei Knoten vor, muss nach einem Kriterium entschieden werden, ob ein Fehler vorliegt oder nicht. Hier wäre es möglich, ähnlich wie es Hitec vollzieht, über die Einführung von Parametern noch genauere Abschätzungen zu treffen, ob ein Fehler vorhanden ist oder ob es Coverage-bedingt an der aktuellen Position kein Fehler ist. Diese Parameter könnten die Genomlänge und die Fehlerrate eines Readsets, sowie zusätzlich die Vorkommenshäufigkeit der Nukleotide im Genom sein. Konkret bedeutet dies, dass die Vorkommenshäufigkeiten der Nukleotide betrachtet werden müssen, da A,C,G,T nicht gleich häufig vorkommen. Weiterhin kann i.V.m. dem gerade angesprochenen Parameter der Genomlänge aus Hitec abgeschätzt werden, wie die theoretische Coverage ist. Aus diesen beiden Informationen könnte man ein besseres Kriterium herleiten, nach dem eine Base korrigiert wird oder nicht.

Seit Veröffentlichung von SHREC und der Arbeit an den Verbesserungen wurde das Gebiet der Error-Correction sehr weit voran getrieben. Neuere Veröffentlichungen wie Hitec erwarten vom Benutzer mehr Parameter (s.o.), die aufwändigere Wahrscheinlichkeitsmodelle ermöglichen und somit mittlerweile bessere Ergebnisse erzielen. Eine Synthese der Analysen aus den Suffixbäumen und deren Integration in diese neuen Algorithmen ist eine anschließende Forschungsmöglichkeit.

6 Kommunikations-Benchmarking in Cloud-Plattformen

6.1 Einführung und Motivation

Komplexe wissenschaftliche Anwendungen oder umfangreiche Softwarelösungen, wie z.B. beim Alignment (siehe Kapitel 4), verarbeiten häufig größere Datenmengen oder berechnen umfangreiche Problemstellungen. Diese Problemstellungen werden durch Algorithmen gelöst, die in der Regel wenige kurze Abschnitte besitzen, die den Hauptteil der Rechenressourcen benötigen. Reicht ein Rechner nicht mehr aus um diese Problemstellung im gewünschten Zeitraum zu lösen oder wird sehr viel Speicher benötigt, spricht man vom High Performance Computing (HPC). Um nun diese rechenintensiven Abschnitte zu beschleunigen, bietet es sich an, diese auf eine Spezialarchitektur, wie einen Cluster oder Spezialhardware (FPGAs, GPUs), auszulagern. Der Nachteil eines Clusters ist allerdings, dass die Investition in die Anschaffung und die Wartungskosten sehr hoch sind. Werden die Rechenressourcen nur kurzzeitig vollständig benötigt, ist die Relation von Kosten zur Auslastung nicht optimal. Für derartige Anwendungsfälle kann es sinnvoll sein, Cloud Computing zu verwenden, da von einem Cloud Computing Dienst sehr viele Rechenressourcen augenblicklich bereit gestellt werden können und nur die benutzte Rechenzeit bezahlt werden muss, wohingegen ein eigenes Rechenzentrum auch bei Nichtbenutzung Kosten verursacht. Beispielsweise ist es möglich, innerhalb von wenigen Minuten hunderte von Recheneinheiten (Instanzen) verfügbar zu haben, aber auch genauso schnell wieder abzugeben.

Aktuelle Cloud Computing Dienste verwenden eine hoch skalierbare Infrastruktur für HPC mit sehr variablen, wählbaren Parametern. So ist es möglich, Festplattengröße, Arbeitsspeicher und Anzahl der Prozessoren frei zu wählen und zu kombinieren.

Der Nachteil von Cloud Computing auf der anderen Seite ist, dass Algorithmen, die sehr kommunikationsintensiv sind, mit einer sehr hohen Varianz in der Kommunikationsgeschwindigkeit umgehen müssen. Hierdurch wird die Laufzeit eines kommunikationsintensiven Algorithmus signifikant beeinflusst.

Sehr wichtig ist ebenfalls, dass die höhere Auslastung der Rechenressourcen eines Cloud Computing Dienstes, im Vergleich zu einem eigenen Rechenzentrum, energieeffizienter ist und somit das *Green IT Paradigma* unterstützt.

Es gibt einige wissenschaftliche Anwendungen, die Cloud Computing verwenden, wobei die Felder weit gestreut sind, von Meereswissenschaften [17] bis zur Bioinformatik mit Genomsequenzanalyse [56] oder Genomalignment [40, 74].

Die Forschung im Bereich Algorithmen auf Cloud Computing Plattformen ist, genau wie die Plattformen selbst, sehr jung. Aus diesem Grund bringt diese Infrastruktur neue Herausforderungen mit sich, bietet aber auch viel Potential für Optimierungen auf Algorithmenebene.

Genauer betrachtet bedeutet die Verwendung eines Cloud Computing Dienstes Folgendes: Man arbeitet auf einer unbekannten physikalischen Netzwerkstruktur und -topologie. Hierdurch können verschiedene Kommunikationsabläufe in extrem unterschiedlicher Performanz, in Bezug auf Kommunikationsgeschwindigkeit und -latenz, resultieren. Es kann sogar vorkommen, dass ein und dieselbe Berechnung eine deutlich unterschiedliche Laufzeit hat, wenn sie mehrmals auf der selben Menge von Cloud Instanzen durchgeführt wird.

Aus dem Blickwinkel der Algorithmenentwicklung bringt die Ausführung eines Algorithmus auf einem Cloud Computing Dienst also viele Herausforderungen mit sich. Dies ist vor allem der Fall, da das Konzept eines herkömmlichen Rechenzentrums mit deterministischen, konstanten Zugriffszeiten und Latenzen nicht auf Cloud Computing Plattformen übertragbar ist. Ein weiterer Faktor ist, dass bei der mehrfachen Anforderung einer konstanten Anzahl von Cloud Instanzen nacheinander völlig verschiedene Mengen von Instanzen zur Verfügung gestellt werden können und sich damit das Kommunikationsverhalten ändern kann.

Es existiert mit dem *Message Passing Interface* Standard (MPI) zwar eine Abstraktionsschicht der Kommunikation, allerdings bietet diese Schicht keine automatische oder semi-automatische Optimierung für Kommunikationsmuster, wenn sie auf einer Cloud Computing Plattform umgesetzt wird. Dies ist allerdings aufgrund von o.a. Gründen essentiell, wenn man auf einer Cloud Computing Plattform arbeitet.

In dieser Arbeit wurde sich damit beschäftigt, ein Verfahren in Form eines Benchmarks zu schaffen, das die Nachteile in Bezug auf die Kommunikation misst und aufzeigt, um mit diesen Daten die Nachteile durch entsprechende Schritte kompensieren zu können. Konkret wurde ein Benchmark erstellt, der die minimalen,

maximalen und durchschnittlichen Kommunikationsgeschwindigkeiten sowie deren Varianz für das jeweilige Cloud Netzwerk mit N Instanzen misst und entsprechend mehrere $N \times N$ Matrizen ausgibt. Des Weiteren wird die Möglichkeit geboten, die Kommunikationsgeschwindigkeiten in Form eines Graphen visuell darzustellen.

Mit Hilfe der Messergebnisse des Benchmarks ist es möglich, einen beliebigen Algorithmus an die Gegebenheiten eines konkreten Cloud Instanzsatzes anzupassen, indem beispielsweise die Verteilung von Aufgaben des Algorithmus (Tasks), in Abhängigkeit zur Kommunikationsintensivität, vorgenommen wird.

Weiterhin wurde ein Master/Slave Auswahlalgorithmus auf Basis des beschriebenen Benchmarks entwickelt. Hierbei werden die Messergebnisse des Benchmarks für eine Optimierungsfunktion verwendet, die die Anzahl der Nachrichten zwischen Master und Slaves maximiert.

Es gibt eine Vielzahl von Cloud Computing Diensten, wobei die Unterschiede der Anbieter für diese Arbeit nicht von Bedeutung sind, da die Konzepte der Anbieter weitgehend ähnlich sind und in dieser Arbeit weder die konkrete Performanz, noch die Kostenstruktur relevant sind. Es wurde die Amazon EC2 (Amazon Elastic Computer Cloud) genutzt. Für den Kommunikationsbenchmark wurde MPI (siehe Kapitel 2.5.2) als Kommunikationsschicht verwendet, um die Kommunikationsgeschwindigkeiten zu messen.

Die Motivation im Zusammenhang mit dieser Arbeit war, dass die entwickelten Algorithmen zum Genomalignment in Kapitel 4 und zur Fehlerkorrektur in Kapitel 5 sehr rechenintensive Abschnitte haben. Die Fragestellung, ob und wie diese parallelisiert werden können, führte zur Evaluation von u.a. Cloud Computing Plattformen. Hierbei wurde festgestellt, dass die Problemstellung der Kommunikationsgeschwindigkeiten in Cloudnetzen bisher in der Forschung nicht sehr tiefgehend betrachtet wurde. Für die Evaluation der Machbarkeit einer Nutzung von Cloud Computing Plattformen für Genomalignment oder Fehlerkorretur war dies aber essentiell.

Die Ergebnisse dieser Arbeit wurden auf der Konferenz *Advances in Computing and Communications* vorgetragen und in den Proceedings von Springer [69] sowie in dem Journal Network Protocols and Algorithms in [68] veröffentlicht.

Terminologie

In diesem Abschnitt wird kurz die Terminologie von Cloud Computing Diensten erläutert. Eine weitere Einführung in die Ursprünge des Cloud Computing findet sich in 2.5.1.

Bei der Verwendung eines Cloud Computing Dienstes verwendet man sogenannte Instanzen, virtuelle Maschinen, auf denen man rechnet. Die Instanzen erscheinen für den Benutzer wie ein vollständiger Rechner und sind i.d.R. per SSH über das Internet steuerbar. Die darunter liegenden physikalischen Maschinen, auf denen die Instanzen laufen, befinden sich in verschiedenen Regionen (z.B. US-East, Europe,..) und innerhalb dieser Regionen kann es mehrere Datenzentren geben, die als sogenannte Availability Zones, im Folgenden *Zonen*, (z.b. US-East-1a, US-East-1a,..) benannt sind. Die Verwendung wird stundenweise abgerechnet.

Die Rechenleistung wird aber auch in Form von Spot Instanzen *gehandelt*. So ist es möglich, einen Preis für eine bestimmte Rechenleistung zu bieten und zu warten, bis dieser Preis erreicht wird. Ist der Preis erreicht, werden diese Instanzen zur Verfügung gestellt und man kann mit der Berechnung beginnen. Zu beachten ist, dass diese Instanzen über verschiedene Zonen verteilt sein können.

6.2 Bestehende Benchmarks

Eine Vielzahl von Forschungsgruppen hat sich mit Machbarkeitsstudien der Implementierung von wissenschaftlichen Anwendungen in der EC2 beschäftigt. Der Fokus liegt allerdings fast ausschließlich auf der Performanz der Rechenressourcen, anstatt der Performanz der Kommunikation zwischen diesen. Hazelhurst et al. untersuchten die Performanz für Bioinformatik-Anwendungen [27]. Die Performanz und Speicherkosten für Astronomie-Anwendungen wurden per Simulation von Deelman et. al. durchgeführt [10]. Das High-Energy and Nuclear Physics (HENP) STAR experiment untersuchte die Kosten und Herausforderungen, die die Verwendung von EC2 mit sich bringen würde [33]. Zusätzlich wurde betrachtet, wie sich die individuellen Amazon AWS Komponenten, wie z.B. der *Simple Storage Service* (S3), verhalten [55]. Napper et al. verwendeten die Benchmark Suite Linpack, um die Performanz von EC2 zu bestimmen [51]. [30]

In [1] wird die Performanz von physikalischen Maschinen im Vergleich zu EC2 dargestellt. Hier wird wiederum ein Schwerpunkt auf die Rechenleistung gelegt.

Einige Arbeiten betrachten aber auch die Performanz der Kommunikation, allerdings meist an einer konkreten Anwendung und nicht an abstrakten Paradigmen. So sind folgende Arbeiten in dem Bereich Kommunikationsperformanz zu nennen.

Rehr et al. zeigen, dass EC2 für Anwendungen einsetzbar ist, wenn keine besonderen Anforderungen an die Netzwerkperformanz nötig sind [61]. Hierbei wird allerdings eine Anwendung aus der Astronomie [80] als eine Art Benchmark verwendet und hieraus eine Art Bewertung abgeleitet.

In [87] wurde eine Studie auf Basis von Messungen der Netzwerkperformanz durchgeführt. Hierbei wurde der Einfluss der Virtualisierung in Cloud Computing Plattformen auf die Netzwerkperformanz in EC2 betrachtet. Unter anderem wurden dabei die Latenz, der TCP/UDP Durchsatz und der Paketverlust genauer analysiert. Zusammenfassend sagen die Autoren, dass die EC2, auf Grund der internen Virtualisierung der Plattform, das Risiko von signifikanten Durchsatzschwankungen und Verzögerungen birgt.

Eine Performanzbetrachtung, durch Analyse von verschiedenen MapReduce Implementationen auf Cloud Computing Plattformen, wird in [14] durchgeführt.

Zusammenfassend lässt sich sagen, dass der Betrachtung der reinen Performanz der Kommunikation, losgelöst von einer konkreten Anwendung, in der Forschung bisher zu wenig Aufmerksamkeit geschenkt wurde.

6.3 Der Algorithmus zum Kommunikations-Benchmarking

In diesem Abschnitt folgt nach einer Beschreibung der Algorithmen eine Darstellung von Messwerten und Ergebnissen der EC2.

6.3.1 Algorithmus zum Benchmarking

Das Arbeiten auf einer unbekannten Netzwerkstruktur kann zu vielen Komplikationen führen. Eine EC2 Spot Instanzmenge kann z.B. in verschiedene Rechenzentren verteilt sein, so dass allein die physikalische Entfernung zu einer hohen Latenz führt. Weiterhin ist die Position innerhalb eines Rechenzentrums beliebig, wobei die Instanzen aber auch auf der selben physikalischen Maschine sein können.

Weiterhin können sogenannte Noisy Neighbors die Kommunikation beeinflussen. Dies sind Instanzen innerhalb des selben Routing Levels, z.B. eine virtuelle Maschine auf dem selben physikalischen Rechner oder eine andere Instanz in der Nähe der eigenen physikalischen Position, die sehr viel kommunizieren. Da die Netzwerkressourcen geteilt werden, beeinflussen die Noisy Neighbors die eigene Kommunikation.

Schließlich steigt die physikalische Distanz bei steigender Instanzanzahl, da nur eine begrenzte Anzahl an Instanzen einen gewissen physikalischen Raum einnehmen können. Dies beeinflusst wiederum die Latenz, da die verbindende Hardware (meist Router) die Datenübertragungsrate bestimmt.

Um eine möglichst genaue Sicht auf die Verbindungen im Sinne der Kommunikationsgeschwindigkeit zu bekommen, wurde in dieser Arbeit ein Algorithmus entwickelt, der sowohl die maximale Kommunikationsgeschwindigkeit, als auch den maximalen Durchsatz für größere Datenpakete bestimmt.

Messung der Kommunikationsgeschwindigkeiten

Die Problemstellung, die Kommunikationsgeschwindigkeiten zu messen, ist sehr komplex, da das Verhalten der umgebenden Instanzen (Nachbarn) und die physikalische Entfernung (bzw. Infrastruktur) zwischen den Instanzen unbekannt sind, wobei diese aber die Kommunikationsgeschwindigkeiten stark beeinflussen können.

Weitere Faktoren sind die unterschiedlichen physikalischen Maschinen, die verschiedene Hardware haben können und daher verschiedene Verarbeitungsgeschwindigkeiten und Ein-/Ausgabegeschwindigkeiten besitzen können.

Hinzu kommt, dass Cloud Dienste fast ausschließlich virtuelle Maschinen zur Verfügung stellen, also auf einem physikalischen System mehrere virtuelle Systeme betreiben, in denen sich die virtuellen Maschinen den Speicher, Arbeitsspeicher und die CPUs teilen.

Um eine Aussage über die Geschwindigkeit zu treffen und diese schwankenden Faktoren (vor allem Nachbarn) zu berücksichtigen, stellt sich die Frage, wie lange eine Performanzanalyse dauern darf. Als Parameter ist hierbei zu betrachten, dass die Abrechnung bei EC2 im Stundentakt erfolgt. Eine Benchmarkdauer von einer Stunde ist daher als obere Schranke anzusehen. Auf der anderen Seite hat sich gezeigt, dass die Schwankungen innerhalb weniger Sekunden groß sein können, so dass ein Benchmark über mindestens eine Minute laufen sollte. Es hat sich empirisch gezeigt, dass es sinnvoll ist, für eine Dauer von zwei Minuten eine Performanzanalyse durchzuführen. Diese Dauer ist auch in der Größenordnung der Setup-Zeit einer Instanz, also der Zeit bis das System verwendbar ist, und somit für den Benutzer in einem sinnvollen Rahmen. In der Praxis ist die Benchmarkzeit anpassbar, wenn also ein Algorithmus über mehrere Tage laufen soll, könnte der Benchmark über mehrere Stunden Messungen durchführen und auf Basis dieser die Performanz-Analyse erstellen.

Nach der Performanz-Analyse können die Daten an den eigentlichen Algorithmus weitergegeben werden, so dass er auf dieser Basis die Verteilung der Tasks an die Instanzen regeln und mit der eigentlichen Berechnung beginnen kann.

Zur Performanz-Analyse wird ein $N \times N$ Kommunikationsmuster verwendet, in dem alle Instanzen mit allen anderen Instanzen jeweils paarweise kommunizieren. Die Paarung erfolgt pseudo-zufällig, wobei sichergestellt wird, dass die Paare gleichverteilt sind, also dass alle Instanzen mit allen anderen Instanzen genau gleich häufig *kommunizieren*. Eine Instanz sendet bei dieser sog. *Kommunikation* einer anderen Instanz (Nachbar) eine Nachricht von vorher festgelegter Größe s (size) und dies r (repeat) mal (Wiederholungen). Hierbei wartet sie und misst die Zeit, bis der Nachbar auf die soeben gesendete Nachricht antwortet. Weiterhin wird diese Prozedur c (cycle) mal wiederholt (Durchläufe), um eine Messung über eine bessere Basis in Form eines längeren Zeitraums zu haben.

Aus den gewonnen Messwerten werden die Werte $s_{min,i,j}$ (Minimallatenz),$s_{max,i,j}$ (Maximallatenz), $s_{avg,i,j}$ (Mittlere Latenz) und Standardabweichungen $d_{i,j}$ für alle Paare i, j von Instanzen berechnet und ausgegeben. Wenn $d_{i,j}$ einen vorher definierten Schwellwert d_m überschreitet, wird eine Warnung ausgegeben und mitgeteilt, dass die Messung gegebenenfalls nicht genau genug war, bzw. die Schwankungen sehr hoch waren.

Implementierung der Geschwindigkeitstests

Um die Tests auszuführen wurde auf EC2 ein 32bit Fedora Linux Image verwendet. Dieses Image stammt von dem Autor des EC2-Verwaltungsskriptes [77], das die Einrichtung und Abschaltung der Instanzen übernimmt. Die Benchmarking Software wurde in C implementiert. Es wurde die MPI Implementierung MPICH2, Version 1.0.5 [24] verwendet.

Als Erstes sucht das Start-Skript nach allen Instanzen, die das selbe AMI-ID (Amazon Machine Image) verwenden und erzeugt die nötige MPI-Konfiguration mit den nötigen Rechneradressen. Nachdem alle SSH Schlüssel verteilt wurden, wird das Benchmarkprogramm kompiliert und auf der ersten Instanz per *mpicc* and *mpirun* ausgeführt. Diese MPI Bibliotheken sorgen dafür, dass alle Instanzen in der MPI Umgebung das Benchmarkprogramm erhalten und ausführen.

Der Ablauf ist in Algorithmus 1 auf Seite 126 dargestellt.

Algorithm 1 Benchmark Algorithmus

```
Lies eigene Identität (Zonenname und Instanzname)
Ermittle Instanztyp (Master oder Slave)
if Instanztyp ist Master then
  Generiere Paarungen
  Sende Paarungen an Slaves
  Sende Parameter an Slaves
  (Nachrichtengröße, Anzahl der Durchläufe, Anzahl der Wiederholungen)
else if Instanztyp ist Slave then
  Empfange Paarungen
  Empfange Parameter
end if
for n Durchläufe do
  for m Wiederholungen do
    Miss die Zeit eine Nachricht zu senden mit MPI_Wtime()
  end for
end for
if Instanztyp ist Slave then
  Sende Messergebnisse an Master
else if Instanztyp ist Master then
  Empfange Messergebnisse von allen Slaves
  Generiere Zusammenfassung
end if
```

Als Erstes werden die eigene Identität und die Parameter ermittelt. Danach werden vom Master die Paarungen für die Durchläufe generiert und zusammen mit den Parametern Nachrichtengröße, Anzahl der Durchläufe und Anzahl der Wiederholungen an die Slaves gesendet. Nun führen alle Instanzen das Senden und Empfangen für die festgelegte Anzahl an Wiederholungen und Durchläufen durch, wobei *MPI_ Wtime()* verwendet wird, um die Zeit zu messen, die eine Nachricht benötigt. Nachdem alle Tests beendet sind, werden die aufgezeichneten Daten an den Master gesendet, der dann die Daten aggregiert und dem Benutzer ausgibt.

Analyse der Ergebnisse

Die Ergebnisse können dem Benutzer auf zwei verschiedene Weisen ausgegeben werden. Die Standardvariante gibt eine Tabelle mit den Durchschnittswerten aus (siehe Abbildung 6.2, S. 132), die ausführliche Variante gibt zusätzlich die minimalen und maximalen Zeiten sowie die Standardabweichung aus. Durch Einlesen dieser Ausgaben ist es möglich, die Messdaten in einem anderen Algorithmus zu verwenden, um z.B. zu entscheiden, dass einzelne Instanzen nicht für die geplante Anwendung verwendet werden sollten oder wie die Tasks auf die Instanzen verteilt werden sollen.

Graphische Darstellung

Um dem Benutzer eine visuelle Möglichkeit zu geben, die gemessenen Werte zu interpretieren, werden die gemessenen Daten auch als Grafik ausgegeben, wobei die Darstellung in Form eines Graphen bestehend aus Knoten (den Instanzen) und Kanten (den durchschnittlichen Kommunikationsgeschwindigkeiten) dargestellt werden (siehe Abbildung 6.1, S. 132). Der Graph wird hierbei im Graphviz Format [15] ausgegeben und mit Neato [53] gerendert. Neato ist ein heuristischer Algorithmus, der die Knoten des Graphen basierend auf der Kantenlänge verteilt.

Diese Darstellung ermöglicht einen ersten, sehr schnellen Überblick über die gemessene Cloudkonfiguration und lässt Cluster von Instanzen erkennen, die sehr

nahe, in Bezug auf Kommunikationsgeschwindigkeit, beieinander sind. Zu beachten ist, dass die Zweidimensionalität der Grafik es nicht ermöglicht, die Distanzen des $(N-1)$-dimensionalen Raums darzustellen. In der Praxis sind allerdings sehr oft Cluster von Instanzen vorhanden, die dazu führen, dass die Distanzen zwar nicht originalgetreu, aber sinngetreu zweidimensional dargestellt werden können.

6.3.2 Algorithmus zur Master/Slave-Selektion

Algorithmus zur Instanzauswahl

Um den Algorithmus zur Masterwahl zu beschreiben, ist es notwendig, zuerst folgende Werte zu definieren, wobei die Ergebnisse des Benchmarks für N Instanzen vier $N \times N$ Matrizen entsprechen.

$S_{avg} = (s_{avg,i,j})_{i=1...N,j=1...N}$,
$S_{min} = (s_{min,i,j})_{i=1...N,j=1...N}$,
$S_{max} = (s_{max,i,j})_{i=1...N,j=1...N}$
und $D = (d_{i,j})_{i=1...N,j=1...N}$
Diese bezeichnen die durchschnittliche Latenz S_{avg}, die minimal Latenz S_{min}, die maximale Latenz S_{max} und die Standardabweichung der Latenz D.

Weiterhin sei $s_{min,i,j}$ der kleinste gemessene Wert zwischen den Instanzen i und j. Für $i = j$ sei der Wert 0 und in den folgenden Gleichungen ausgeschlossen.

Die normierten Ergebnisse seien wie folgt definiert:

$$\bar{s}_{avg,i,j} = \frac{s_{avg,i,j}}{\max(S_{avg})}$$
$$\bar{S}_{avg} = (\bar{s}_{avg,i,j})_{i=1...N,j=1...N}$$
$$\bar{s}_{min,i,j} = \frac{s_{min,i,j}}{\max(S_{min})}$$
$$\bar{S}_{min} = (\bar{s}_{min,i,j})_{i=1...N,j=1...N}$$
$$\bar{s}_{max,i,j} = \frac{s_{max,i,j}}{\max(S_{max})}$$
$$\bar{S}_{max} = (\bar{s}_{max,i,j})_{i=1...N,j=1...N}$$
$$\bar{d}_{i,j} = \frac{d_{i,j}}{\max(D)}$$
$$\bar{D} = (\bar{d}_{i,j})$$

Die Bewertungsfunktion für eine Instanz i sei dann wie folgt definiert:

$$c_i = \sum_{j \in \{1...N\} \setminus \{i\}} [w_{avg}(\bar{s}_{avg,i,j})^{n_{avg}} + w_{min}(\bar{s}_{min,i,j})^{n_{min}} + w_{max}(\bar{s}_{max,i,j})^{n_{max}} + w_{dev}(\bar{d}_{i,j})^{n_{dev}}]$$

Für eine Instanz i wird also die Summe der normierten, gewichteten Messwerte potenziert mit einem weiteren Gewicht als Bewertung gebildet. Hierbei sind die Parameter w_{avg}, w_{min}, w_{max}, und w_{dev} Gewichte, die je nach Anforderungen des Benutzers modifiziert werden können, um den einzelnen Messwerten (z.B. der Standardabweichung) mehr oder weniger Gewichtung zu geben. n_{avg}, n_{min}, n_{max}, und n_{dev} ermöglichen eine weitere Gewichtung, um größeren Werten durch die Potenzierung stärkeren Einfluss zu geben.

Bei allen Werten ($\bar{s}_{avg,i,j}$, $\bar{s}_{min,i,j}$, $\bar{s}_{max,i,j}$, $\bar{d}_{i,j}$) entspricht ein möglichst kleiner Wert dem Optimum. Die gewichtete Summe als Bewertungsfunktion ist daher naheliegend.

Im einfachsten Fall wäre die Formel für

$w_{avg} = w_{min} = w_{max} = w_{dev} = n_{avg} = n_{min} = n_{max} = n_{dev} = 1$

$$c_i = \sum_{j \in \{1...N\} \setminus \{i\}} (\bar{s}_{avg,i,j} + \bar{s}_{min,i,j} + \bar{s}_{max,i,j} + \bar{d}_{i,j})$$

also der Summe der Messwerte.

Es wird der Master M als die Instanz i ausgewählt, für den c_i am Kleinsten ist.

$$M = i : c_i = min_{j \in \{1...N\}}(c_j)$$

6.4 Ergebnisse aus Messreihen

6.4.1 Algorithmus zum Kommunikations-Benchmarking

Es wurden verschiedene Anwendungsfälle zur Performanzevaluierung durchgeführt.

Instanzen aus einer Zone

Der wohl häufigste Anwendungsfall ist, wenn mehrere Instanzen aus einer Zone stammen und einen Task berechnen. Bei diesem Anwendungsfall würde man davon ausgehen, dass die Menge von Instanzen jeweils gleich schnell miteinander verbunden ist. Die Ergebnisse dieser Arbeit zeigen, dass dies nicht immer der Fall sein muss und die Kommunikationsgeschwindigkeiten enorm schwanken können.

Wie Abbildung 6.1 und 6.2 zeigen, ist Instanz 10 *weiter entfernt* von den anderen Instanzen als die übrigen Instanzen jeweils zueinander. Hätte man einen Algorithmus, der viel mit Instanz 10 kommunizieren müsste, würde dies die Ausführungsdauer stark beeinflussen.

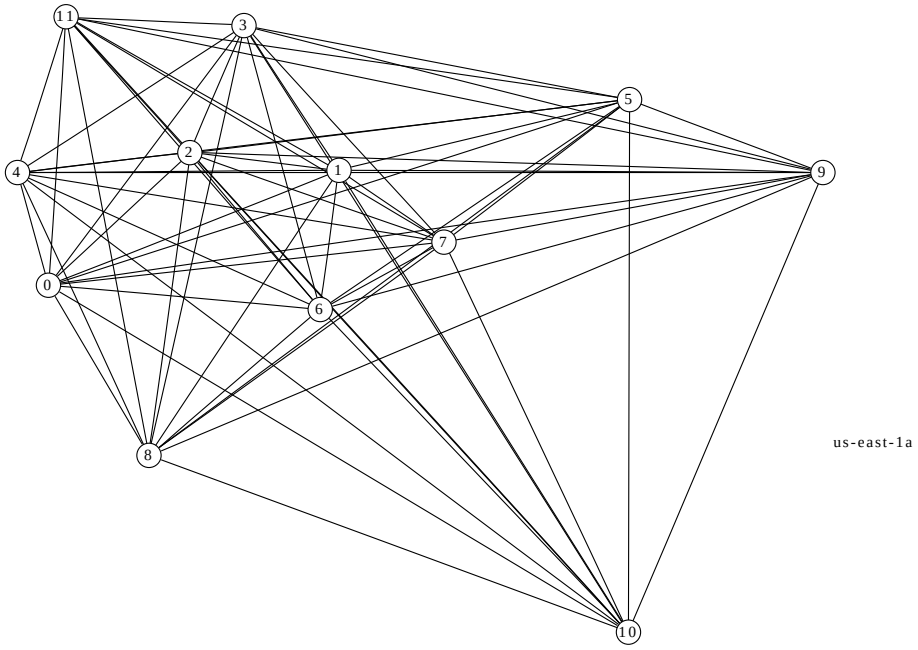

Abb. 6.1: Kommunikationsdauer aus einer Zone als Graph
Ein Knoten des Graphen entspricht einer Instanz, die Länge einer Kante zwischen zwei Knoten entspricht der Dauer einer Kommunikation. Es ist zu sehen, dass es Cluster von Instanzen gibt, die schnell miteinander kommunizieren können.

Kommunikationsdauer [s]:

	1	2	3	4	5	6	7	8	9	10	11
0	0.0209	0.0172	0.0323	0.0677	0.0243	0.0185	0.0149	0.0257	0.0364	0.0509	0.0325
1		0.0150	0.0319	0.0335	0.0205	0.0158	0.0153	0.0297	0.0242	0.0432	0.0312
2			0.0334	0.0244	0.0239	0.0197	0.0216	0.0218	0.0352	0.0405	0.0403
3				0.0344	0.0298	0.0283	0.0257	0.0309	0.0271	0.0566	0.0260
4					0.0296	0.0502	0.0273	0.0368	0.0317	0.0499	0.0350
5						0.0226	0.0266	0.0350	0.0339	0.0464	0.0347
6							0.0179	0.0294	0.0270	0.0390	0.0313
7								0.0231	0.0324	0.0494	0.0257
8									0.0388	0.0553	0.0366
9										0.0545	0.0323
10											0.0496
11											

Abb. 6.2: Kommunikationsdauer aus einer Zone als Matrix
Jeder Eintrag entspricht dem Mittelwert in Sekunden, den eine Nachricht benötigt hat, um von einem Knoten zu einem anderen und zurück gesendet zu werden. Die Nachrichtengröße betrug 102400 Byte. Es wurden fünf Wiederholungen und 25 Durchläufe durchgeführt.

Instanzen aus verschiedenen Zonen

Bei EC2 ist es möglich, Instanzen gezielt aus mehreren Zonen anzufordern (z.B. us-east-1a). Dies ist sinnvoll, wenn die Instanzen unabhängig sein sollen, um z.B. eine erhöhte Ausfallsicherheit zu haben, da die Zonen unabhängigen Rechenzentren entsprechen.

Die Ergebnisse zeigen, dass die Verbindungen zwischen Instanzen aus verschiedenen Zonen deutlich unterschiedlich zu Instanzen aus einer Zone sind. Abbildungen 6.3 und 6.4 illustrieren dies.

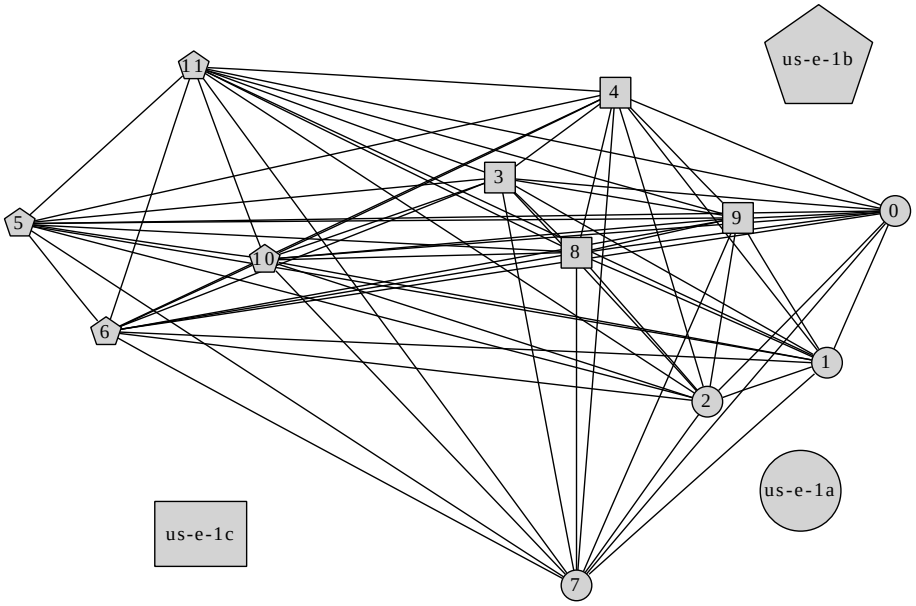

Abb. 6.3: Kommunikationsdauer aus drei Zonen als Graph
Es wurden die Zonen us-east-1a, us-east-1b und us-east-1c ausgewählt. Die Form der Knoten des Graphen repräsentiert die Zonenzugehörigkeit.

Speed [s]:

	1	2	3	4	5	6	7	8	9	10	11
0	0.0053	0.0068	0.0147	0.0146	0.0274	0.0267	0.0141	0.0153	0.0164	0.0287	0.0290
1		0.0034	0.0120	0.0134	0.0246	0.0248	0.0086	0.0160	0.0164	0.0259	0.0250
2			0.0126	0.0153	0.0293	0.0243	0.0096	0.0162	0.0138	0.0242	0.0277
3				0.0052	0.0137	0.0168	0.0197	0.0041	0.0059	0.0166	0.0134
4					0.0147	0.0150	0.0225	0.0053	0.0063	0.0151	0.0138
5						0.0073	0.0311	0.0190	0.0171	0.0074	0.0065
6							0.0311	0.0175	0.0178	0.0069	0.0071
7								0.0217	0.0210	0.0331	0.0357
8									0.0077	0.0141	0.0182
9										0.0186	0.0178
10											0.0059

Abb. 6.4: Kommunikationsdauer aus drei Zonen als Matrix
Die Einträge entsprechen den Mittelwerten der Sendedauer einer Nachricht von einer Instanz zu einer anderen und zurück. Die Nachrichtengröße betrug 100 Byte, es wurden 15 Durchläufe und 100 Wiederholungen durchgeführt.

Spot Instanzen

Spot Instanzen sind Instanzen bei EC2, die man für ein bestimmtes Gebot von EUR pro Stunde erhält. Hierbei muss man so lange auf die Bereitstellung der Instanzen warten, bis der aktuelle Gebotspreis den eigenen Gebotspreis unterschreitet. Die Instanzen können hierbei aus verschiedenen Zonen kommen.

Für diesen Anwendungsfall wurden mehrere Tests durchgeführt. Hierbei stellte sich heraus, dass in etwa 65% der Fälle die Instanzen aus einer Zone stammten, in den anderen Fällen stammten die Instanzen aus mehr als einer Zone.

Da sich die Ergebnisse für den Fall von Instanzen aus einer Zone nicht signifikant von den obigen Ergebnissen von Instanzen aus einer Zone unterschieden, werden im Folgenden nur die Ergebnisse für Instanzen aus mehreren Zonen gezeigt.

Wie in Abbildung 6.5 zu sehen ist, unterscheiden sich die Geschwindigkeiten sehr stark.

Diese Ergebnisse sind besonders wichtig, da die Verwendung von Spot Instanzen zu sehr unterschiedlichen Kommunikationsgeschwindigkeiten führen kann, da die Menge der Instanzen, im Vergleich zu normalen Instanzen, weiter über das Rechenzentrum oder die Rechenzentren gestreut sein können.

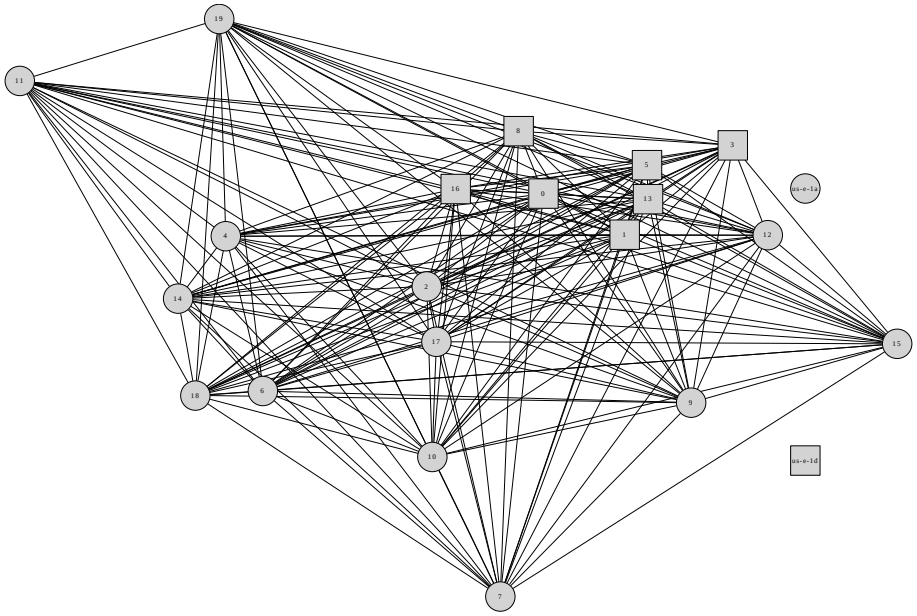

Abb. 6.5: Kommunikationsdauer von 20 Spot Instanzen aus verschiedenen Zonen als Graph
Die Instanzen stammen aus verschiedenen Zonen: us-east-1a und us-east-1d. Die Form der Knoten gibt die Zonenherkunft der Instanzen an.

Varianz der Performanzergebnisse

Nicht nur die Durchschnittsgeschwindigkeit ist aussagekräftig und ausschlaggebend für komplexe Algorithmen. Genauso wichtig oder wichtiger kann die Varianz (oder Standardabweichung) der Kommunikationsgeschwindigkeit sein, da eine Verbindung mit einer hohen Varianz unzuverlässig im Sinne von nicht planbarer Kommunikation sein kann.

Die Ergebnisse zeigen, dass langsame Verbindungen mit einer hohen Standardabweichung korrelieren. Dies bedeutet nicht, dass die Verbindungen durchgehend langsam sein müssen, sondern dass sie nicht gleichmäßige Geschwindigkeiten haben. Abbildung 6.6 zeigt ein Beispiel.

Speed Standardabweichung [s]:

	1	2	3	4	5	6	7	8	9	10	11
0	0.0964	0.0452	0.0966	0.8457	0.0178	0.0415	0.0081	0.0675	0.2410	0.1289	0.0738
1		0.0355	0.0745	0.1080	0.0088	0.0159	0.0297	0.0929	0.0758	0.0760	0.0679
2			0.0969	0.0438	0.0324	0.0627	0.0826	0.0438	0.2060	0.0450	0.0977
3				0.0814	0.0528	0.0749	0.0548	0.0515	0.0556	0.0953	0.0595
4					0.0468	0.3716	0.0683	0.0789	0.0718	0.0679	0.0800
5						0.0111	0.0939	0.0636	0.0778	0.0201	0.0636
6							0.0549	0.0703	0.0528	0.0769	0.0715
7								0.0565	0.0985	0.1028	0.0607
8									0.0860	0.1449	0.0703
9										0.0888	0.0824
10											0.0669
11											

Abb. 6.6: Standardabweichung der Kommunikationsdauer als Matrix
Die Werte gehören zu den Messungen der Durchschnittswerte von Abbildung 6.2, S. 132

Konsistenz der Ergebnisse

Da die Messungen jeweils nur über einen Zeitraum von zwei Minuten laufen, ist es notwendig, die Aussagekraft für einen längeren Zeitraum nachzuweisen. Es kann beispielsweise ein Noisy Neighbor nach 3 Minuten die Messungen beeinflussen und hypothetisch andere Ergebnisse liefern. Dies ist natürlich nicht vollständig auszuschließen und kann auch der Fall sein, allerdings ergaben die Langzeitmessungen, dass man sich auf eine Kurzzeitmessung relativ sicher verlassen kann, wie im Folgenden zu sehen ist.

Um zu zeigen, dass die Messungen für einen längeren Zeitraum repräsentativ sind, wurde der Benchmark über einen Zeitraum von 30 Stunden betrachtet, wobei der Benchmark alle fünf Minuten durchgeführt wurde. Aus diesen Messungen wurden die Latenzmatrizen genommen und wiederum die Mittelwerte und die Standardabweichung für jede der $N \times N$ Kommunikationen bestimmt (siehe Abbildung 6.7).

Die sehr niedrigen Standardabweichungen zeigen, dass die Messungen der kurzen Zeiten verallgemeinert werden können.

Weiterhin wurden Tests durchgeführt, in denen die gleichen Instanzen über einen Zeitraum von einer Stunde verwendet wurden. Hierbei wurde der Benchmark in Intervallen von 15 Minuten ausgeführt und es wurden fast gleichmäßige Latenzen gemessen und nur kleine Varianzen (z.B. durch Noisy Neighbors) festgestellt.

Standard deviation [s]:

	1	2	3	4	5	6	7	8	9	10	11
0	0.00101	0.00057	0.00064	0.00079	0.00055	0.00064	0.00059	0.00107	0.00131	0.00143	0.00135
1		0.00078	0.00082	0.00086	0.00129	0.00140	0.00099	0.00068	0.00132	0.00111	0.00109
2			0.00079	0.00070	0.00093	0.00115	0.00085	0.00079	0.00103	0.00108	0.00100
3				0.00077	0.00082	0.00098	0.00082	0.00077	0.00093	0.00085	0.00097
4					0.00076	0.00081	0.00065	0.00071	0.00064	0.00071	0.00055
5						0.00075	0.00075	0.00107	0.00088	0.00097	0.00089
6							0.00073	0.00069	0.00090	0.00169	0.00095
7								0.00059	0.00079	0.00079	0.00067
8									0.00104	0.00084	0.00077
9										0.00065	0.00060
10											0.00066
11											

Abb. 6.7: Standardabweichungen bei Langzeittest
Der Test wurde über eine Dauer von 30 Stunden ausgeführt. Dabei wurde der Benchmark alle fünf Minuten ausgeführt. Die Nachrichtengröße betrug 10 Kilobyte, es wurden 20 Durchläufe und 150 Wiederholungen durchgeführt.

Um diese Ergebnisse darzustellen, zeigt Abbildung 6.8, wie sich die Reihenfolge der schnellsten Nachbarn jedes Knotens verhält.

test 1	test 2	test 3	test 4
a: e j f k l h b g d i c	a: e g j f l k h d b i c	a: e f g j l k h b d i c	a: g j f e k l h b d i c
b: f e g j l d k c a i h	b: e g j l k f d c a h i	b: e f g j k l d c a i h	b: e f j l g k d a i c h
c: e g l f k h j b d i a	c: e g f k j h b l d i a	c: f e g j b h l k d i a	c: g f e l k h j d b a i
d: g f j e b k l a i h c	d: e h g k f a b j l i c	d: f e g b l j h i a k c	d: g f e j h k b l a i c
e: g j a f k b h l c d i	e: f b l j d g c k i a h	e: f b k g l h a j c i d	e: g f b j k l c h a d i
f: e b g j h k i d c l a	f: e h g j k b i c a l d	f: e g h b d j k c i l a	f: g e k b j h d c l a i
g: j e d k f h b i c l a	g: k e f j b l i d c a h	g: e f h j b l k c i a d	g: f i d e k a l c h j b
h: f g j e l k a c d i b	h: f k l j d e g a c i b	h: f g e j k l a d c i b	h: j k f g e l c a d i b
i: g f j e l k d b h c a	i: e j f g l k d h b c a	i: f j e g l k d h b c a	i: g k e f l h j b d a c
j: g e f h l b k d i a c	j: e g f b i h k l a c d	j: k g f h b i e a c d l	j: e a h b f g l k d c i
k: g e f j h l c b a i d	k: g e f b l h j a d c i	k: e j f l g h b a i d c	k: g f h e j c l i b a d
l: j e g h c b i k f a d	l: e b k h g j a f i d c	l: e g k f h b a d j i c	l: g b e j f c k h a i d

Abb. 6.8: Nachbarn geordnet nach Verbindungsgeschwindigkeit
Die Ordnung der Knoten in Abhängigkeit zur jeweiligen Kommunikationsgeschwindigkeit über mehrere Tests hinaus. Gruppen von Instanzen sind farblich gleich gekennzeichnet. Obwohl es Unterschiede in den Ordnungen gibt, bleiben schnelle Verbindungen relativ konstant über den Testzeitraum.

6.4.2 Algorithmus zur Master/Slave-Selektion

Es wurde der Algorithmus für verschiedene Zonen und für Spot Instanzen ausgeführt.

Die erste Evaluation wurde für 20 Instanzen in einer Zone ausgeführt. Dieser Test sollte die beste Konnektivität zwischen zwei Instanzen bieten, da alle Instanzen im selben Rechenzentrum sind.

Anschließend wurden Instanzen aus vier verschiedenen Zonen erstellt. Hierbei sind also die Instanzen aus vier räumlich entfernten Rechenzentren und haben insgesamt eine höhere Latenz.

Der letzte Test wurde mit Spot Instanzen durchgeführt.

Die Parameter für die Durchführung sind in Tabelle 6.1 zu sehen.

Parameter	Wert
n_{avg}	1.2
n_{min}	1.0
n_{max}	0.8
n_{dev}	1.3
w_{avg}	1.0
w_{min}	0.3
w_{max}	0.1
w_{dev}	0.5

Tab. 6.1: Parameter für die Messungen

Instanzen aus einer Zone

Wie Abbildung 6.9 und 6.10 zeigen, sind die Kommunikationsgeschwindigkeiten unterschiedlich. Ein Algorithmus, der beispielsweise Instanz 18 als Master auswählen würde, würde eine schlechtere Laufzeit haben. Im Vergleich zu den vorherigen Abbildungen indizieren die Farben zusätzlich die Kantenlängen (grün = schnell, orange = mittel, rot = langsam).

Es ist oft der Fall, dass ein kleiner Teil der Instanzen eine äußerst schlechte Konnektivität zu allen anderen Instanzen hat. Die Gesamtperformanz kann daher verbessert werden, indem man generell mehr Instanzen anfordert, als man tatsächlich benötigt und dann genauso viele der langsamen, überschüssigen Instanzen abschaltet. Dieses Vorgehen wurde in die Tests integriert, wobei bei der hier verwendeten Knotenanzahl die zwei langsamsten Instanzen nicht verwendet wurden.

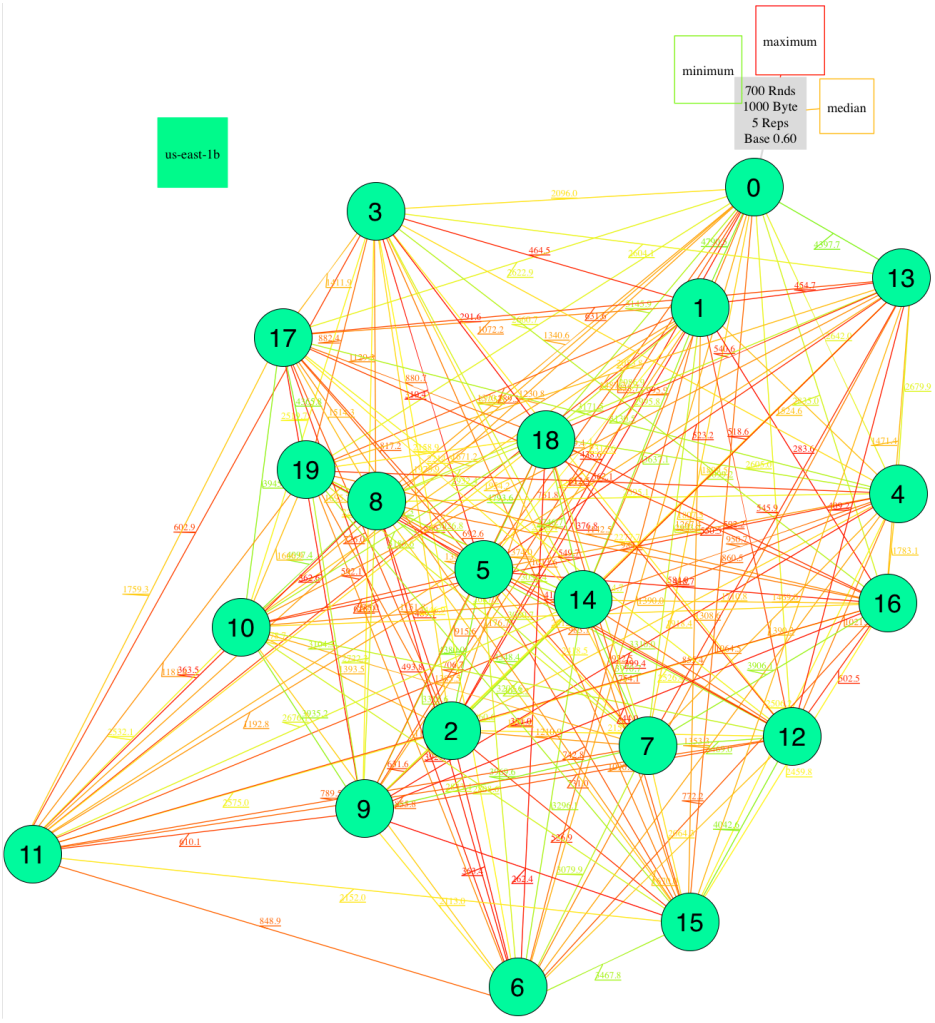

Abb. 6.9: Masterwahl aus einer Zone
Die Farben indizieren zusätzlich zur Kantenlänge die Geschwindigkeit (grün = schnell, orange = mittel, rot = langsam). Zu sehen ist, dass Knoten 18 sehr langsam mit fast allen Knoten kommuniziert.

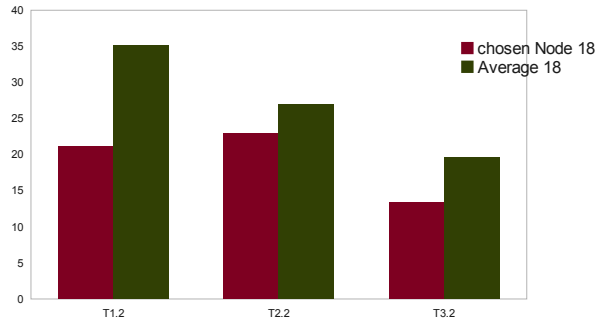

Abb. 6.10: Laufzeitvergleich bei der Masterwahl
Verglichen werden die Gesamtlaufzeiten (y-Achse) der Testläufe mit dem vom Algorithmus vorgeschlagen Master im Vergleich zur mittleren Laufzeit, wenn ein beliebiger Knoten als Master gewählt wurde. Die Nachrichtengröße betrug 1000 Byte im ersten Test (T1.2), 100 Byte im zweiten Test (T2.2) und 10 Kilobyte im dritten Test (T3.2). Die Tests wurden 40 mal wiederholt.

Abbildung 6.11 zeigt, dass hierbei ein anderer Master gewählt wird und die Gesamtperformanz steigt.

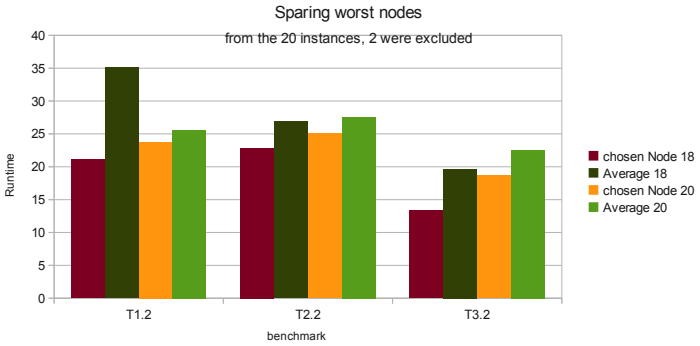

Abb. 6.11: Geschwindigkeitsvergleich bei Entfernung von langsamen Instanzen
Geschwindigkeitsvergleich mit und ohne Verwendung der zwei langsamsten Instanzen. Die Laufzeit ist in Sekunden angegeben.

Instanzen aus verschiedenen Zonen

Wie die vorherigen Tests gezeigt haben, ist die Geschwindigkeit zwischen Instanzen aus einer Zone deutlich schneller als die Verbindung zwischen Knoten aus verschiedenen Zonen. Es ist also besonders wichtig, dass ein Master verwendet wird, der eine hohe Geschwindigkeit zu möglichst allen Zonen hat.

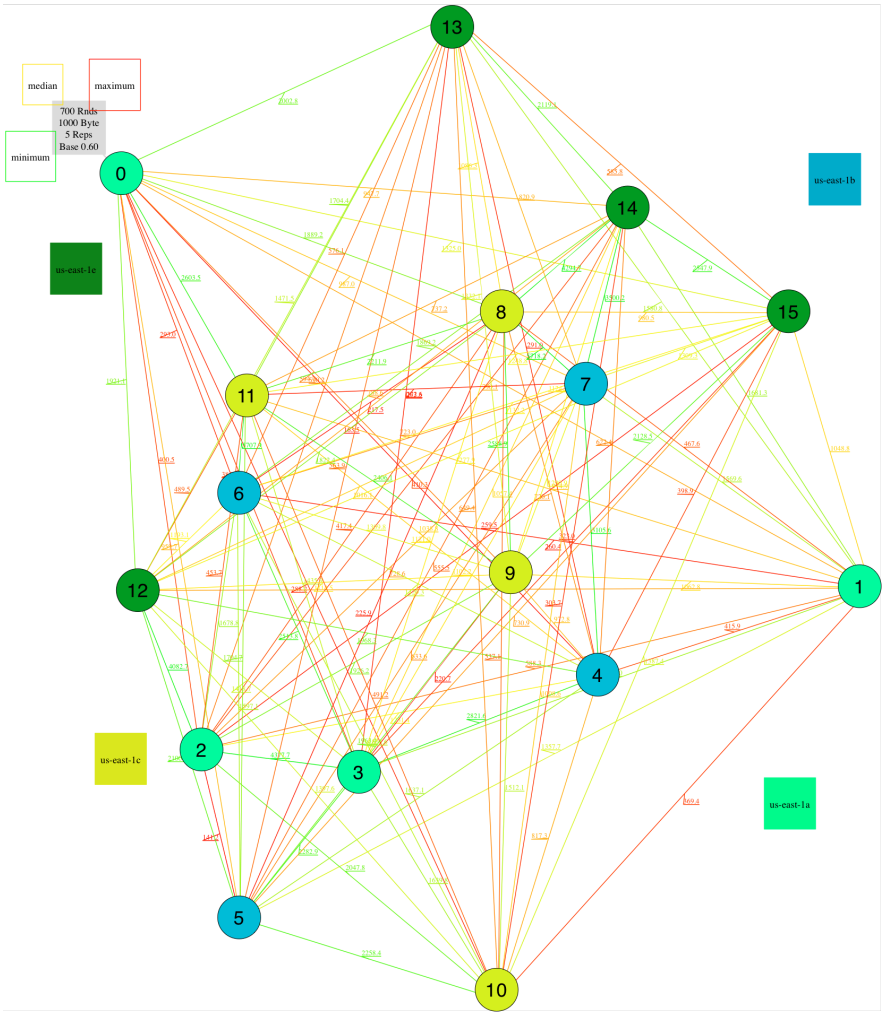

Abb. 6.12: Geschwindigkeitsgraph von 20 Knoten aus vier Zonen
Die Instanzen stammen aus den Zonen us-east-1a, us-east-1b, us-east-1c und us-east-1e. Die Farbe der Instanzen zeigt die Zonenzugehörigkeit an.

Wie Abbildung 6.13 zeigt, kann die Performanz deutlich gesteigert werden, wenn der Master verwendet wird, der durch den entwickelten Algorithmus vorgeschlagen wird.

Obwohl die Performanz zwischen den Tests stark variieren kann, bleibt die Ausführungsdauer besser als der Durchschnitt, wenn der richtige Master gewählt wird.

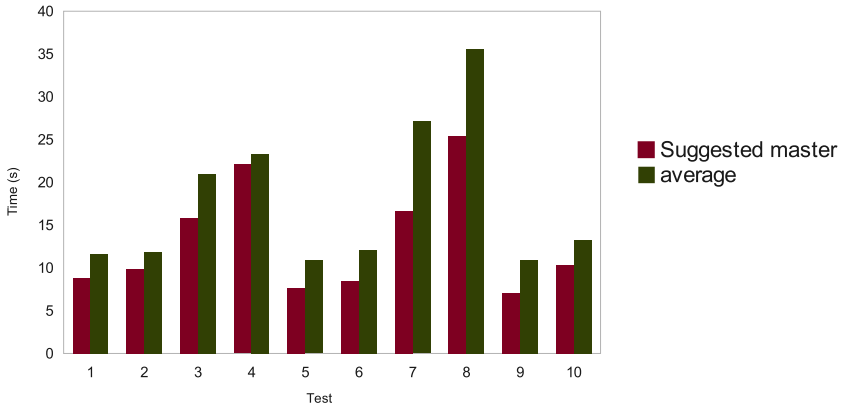

Abb. 6.13: Ausführungsdauervergleich bei Masterwahl
Verglichen werden 20 Instanzen aus vier Zonen. Die Zeiten geben die Ausführungsdauer bei Verwendung des vorgeschlagenen Masters im Vergleich zu dem Durchschnittswert aller anderen Instanzen als Master an. Die Tests wurden mit verschiedenen Nachrichtengrößen durchgeführt. Zwei aufeinanderfolgende Tests verwenden die gleiche Nachrichtengröße. Die ersten beiden Tests wurden mit 1000 Byte, die nächsten beiden mit 10000 Byte, die nächsten mit 100 Byte und schließlich mit 10000 Byte durchgeführt.

6.5 Zusammenfassung und Ausblick

6.5.1 Zusammenfassung

Es wurde in dieser Arbeit ein Benchmark entwickelt, der die Kommunikationsperformanz von EC2 Instanzen misst. Die Ergebnisse zeigen, dass die jeweilige Instanzmenge ausschlaggebend für die Performanz ist und dass die Kommunikationsgeschwindigkeiten sehr stark schwanken. Die Kommunikation zwischen Zonen ist, wie erwartet, generell langsamer als die Kommunikation innerhalb einer Zone, wobei die Kommunikation innerhalb einer Zone in Ausnahmefällen auch extrem langsam sein kann.

Der Benchmark hilft, die tatsächlich vorliegende Kommunikationsstruktur zu analysieren und vor Ausführung eines kommunikationsintensiven Algorithmus zu entscheiden, wie die Verteilung der Tasks auf die Instanzen zu erfolgen hat. Dies kann zu einer großen Performanzsteigerung führen.

Des Weiteren wurde ein Master-Slave-Auswahlalgorithmus entwickelt, der es ermöglicht, einen Master einer EC2 Instanzmenge zu bestimmen, der in Bezug auf die Kommunikation optimal ist. Messungen haben gezeigt, dass diese Auswahl, im Vergleich zur Auswahl eines beliebigen Masters, die Ausführungsdauer stark positiv beeinflussen kann. Hierbei wurden Instanzen in den Konfigurationen einer Zone und mehrerer Zonen verglichen.

6.5.2 Ausblick

Der entwickelte Benchmark verwendet MPI und ist für EC2 angepasst. Da bei der Kommunikation nur Basisfunktionalitäten, wie Senden und Empfangen, verwendet werden, wäre eine Implementierung als C Bibliothek mit Sockets eine Verbesserung und Verallgemeinerung.

Des Weiteren wäre es möglich, weitere Metriken zur Bewertung der Kommunikationsgeschwindigkeit zu entwickeln, die sich speziell auf Cloud Dienste beziehen. Denkbar wären hier Langzeitmessungen für verschiedene Durchsätze und Datenpaketgrößen.

Etwas weiter gedacht wäre eine Abstraktionsschicht denkbar, die die Rechenaufgaben autonom zur Laufzeit in Abhängigkeit zur Kommunikationsgeschwindigkeit verteilt. Die Abstraktionsschicht könnte zur Laufzeit zusätzlich das Abschalten von langsamen Instanzen sowie die Aktivierung von neuen Instanzen verwalten. Dies könnte so aussehen, dass die eigentliche Anwendung einer Abstraktionsschicht einen sog. Task (eine Rechenaufgabe) übergibt und die gewünschte Anzahl der Instanzen zu dessen Berechnung mitteilt. Die Abstraktionsschicht prüft dann, wie der aktuelle Stand der Kommunikationsgeschwindigkeiten ist. Während der Laufzeit könnte die Abstraktionsschicht dann in festgelegten Intervallen die Kommunikationsgeschwindigkeit überwachen und ggf. Tasks neu verteilen, falls eine Instanz eine langsame Kommunikationsgeschwindigkeit aufweist.

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

Das Ziel dieser Arbeit war, ein performantes und qualitativ sehr gutes Alignment von Short Read Daten zu entwickeln. Hierzu wurden die zentralen Schritte eines Sequenzierungsprozesses betrachtet. Dabei wurden von der Modellierung der Daten für das Verständnis derselben, über das Alignment von Reads aus Algorithmensicht und die Fehlerkorrektur von Reads, bis hin zur Analyse von Cloud-Netzwerken verschiedene Thematiken bearbeitet und Verbesserungen oder neue Algorithmen im Bereich Alignment geschaffen. Diese wurden in verschiedenen Veröffentlichungen der wissenschaftlichen Gemeinschaft zur Verfügung gestellt.

Die wichtigsten Ergebnisse werden im Folgenden dargelegt.

Bei der Modellierung von Short Reads wurde im dritten Kapitel ein Model entwickelt, aus dem sich Formeln ableiten ließen, die sehr gute Abschätzungen über die zu erwartenden Ergebnisse eines Sequenzierungsprozesses zuließen. Es wurden die Eigenschaften einer Sequenzierung wie die Coverage-Verteilung, Null-Coverage-Verteilung, Contig-Längen-Verteilung, N50 und Contig-Anzahl durch Formeln nachgebildet und gezeigt, dass gemessene Daten die theoretischen Ergebnisse bestätigen. An Realdaten wurde gezeigt, dass mit den entwickelten Formeln der $N50$-Wert mit einer Genauigkeit von 12 Prozent bestimmt werden konnte. Des Weiteren konnte die Anzahl der Contigs mit 10 Prozent Genauigkeit modelliert werden. Es ist dabei zu beachten, dass in Realdaten in der Regel große Null-Coverage-Regionen existieren und dass die Verteilung der Reads, die durch die jeweilige Technologie bestimmt wird, Beschränkungen an die Qualität des Alignments mit sich bringt, mit denen aber jeder Alignment-Algorithmus umgehen muss.

Im vierten Kapitel wurde das eigentliche Alignment bearbeitet und ein Algorithmus zum Alignment von Reads an ein Referenzgenom entwickelt. Der Fokus wurde auf die statistischen Eigenschaften der Eingabedaten gelegt und es wurde besonders darauf geachtet, dass bei aktuellen Rechnerarchitekturen die Operationen durch die Zugriffszeiten von Arbeitsspeicher und Festplatten nicht den Algorithmus ausbremsen und die Prozessoren optimal genutzt werden. Es wurde gezeigt, dass der entwickelte Algorithmus sowohl mehr Reads alignt als auch eine niedrigere Falsch-Positiv-Rate als der Referenzalgorithmus Bowtie2 bei üblichen

Anwendungsfällen hat. Bei Readsets mit einer Fehlerrate über 10 Prozent konnte gezeigt werden, dass der entwickelte Algorithmus noch gute Ergebnisse mit Alignmentraten von 51 bis 31 Prozent (mit steigender Fehlerrate in den Eingabedaten) erzielt, während die Referenzalgorithmen keine brauchbaren Ergebnisse mit nur 18 bis 12 Prozent lieferten. Weiterhin wurde gezeigt, dass die Falsch-Positiv-Rate bei niedrigen Fehlerraten in dem entwickelten Algorithmus bei 57 Prozent der Falsch-Positiv-Rate des Referenzalgorithmus Bowtie liegt. Des Weiteren wurde nachgewiesen, dass die Laufzeit des entwickelten Algorithmus nur ein Drittel von Bowtie2 beträgt.

Da neben der Referenzsequenz die Short Reads die wesentlichen Eingabedaten sind, wurde im fünften Kapitel eine Verbesserung des Algorithmus SHREC zur Fehlerkorrektur vorgestellt. Hierbei wurde der Speicherbedarf stark reduziert, wodurch größere Read-Sets verarbeitet werden können. Die Speicherreduktion wurde erreicht, indem Patricia Tries verwendet wurden. Hierbei konnte gezeigt werden, dass sich die Knotenanzahl um einen Faktor bis zu 2.6 bei Realdatensätzen reduzieren ließ. Weiterhin wurde die essentielle Parameterwahl von SHREC automatisiert, da sie für Mediziner oder Biologen ohne informatischen Hintergrund sehr schwierig zu bestimmen wäre. Es wurde gezeigt, dass diese Erweiterungen zu einer deutlichen Fehlerreduktion um 19 Prozent, im Vergleich zu den Standardwerten, in den Reads führt.

Um den wachsenden Datenmengen der letzten Jahre in der Sequenzierung, die zu höheren Laufzeiten führen, gerecht zu werden, wurden im sechsten Kapitel Cloud-Dienste analysiert und ein Kommunikations-Benchmark entwickelt. Dieser hilft, die tatsächlich vorliegende Kommunikationsstruktur zu analysieren und vor Ausführung eines kommunikationsintensiven Algorithmus zu entscheiden, wie die Verteilung von Aufgaben auf die verschiedenen Recheneinheiten zu erfolgen hat. Dabei wurde nachgewiesen, dass dies zu einer großen Performanzsteigerung, beispielsweise bei der Wahl des richtigen Masters, führen kann. Hierbei wurde exemplarisch gezeigt, dass sich Laufzeitverbesserungen zwischen 15 und 40 Prozent, durch höhere Kommunikationsgeschwindigkeiten, realisieren lassen.

7.2 Ausblick

Es sind während dieser Arbeit einige Ideen und Anknüpfungspunkte entstanden, die über den Rahmen dieser Arbeit hinaus für eine weitere Bearbeitung interessant sind.

Die Ideen, die den meisten Erfolg versprechen, stammen aus dem vierten Kapitel zum Thema Alignment:

- Die Qualitätswerte der Reads sind besonders beim Alignment eine weitere Möglichkeit, um die Ergebnisse weiter zu verbessern. Durch eine Verwendung der Qualitätswerte ließe sich sowohl die Alignmentrate erhöhen als auch die Falsch-Positiv-Rate senken.
- Aktuelle Aligner wie Bowtie2 bieten neben dem Alignment viele weitere Funktionen. Diese Funktionen verwenden im Wesentlichen immer denselben, schnellen Alignment-Kern. Entsprechend ist eine Erweiterung des Algorithmus um Anwendungsfälle wie dem SNP-Calling oder dem Alignment an mehrere DNS-Sequenzen ein gefragtes Thema, das den schnellen Alignment-Kern verwenden und damit eine anverwandte Problemstellung lösen kann.
- Bisher wurde beim Alignment eine Restriktion der Größe des Arbeitsspeichers vorausgesetzt. Durch stetig wachsende Arbeitsspeicher oder bei größeren finanziellen Mitteln wäre es möglich, weitere Seeds in die Hash-Tabelle aufzunehmen und dadurch eine höhere Alignmentrate zu erreichen, bzw. einen Seed schneller zu finden und damit die Laufzeit zu reduzieren.

8 Literaturverzeichnis

[1] S. Akioka und Y. Muraoka. HPC Benchmarks on Amazon EC2. In *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference*, Seiten 1029 –1034, April 2010.

[2] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers und David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, Oktober 1990.

[3] Stephen Anderson. Shotgun dna sequencing using cloned dnase i-generated fragments. *Nucl. Acids Res*, 9 (13):3015–3027, 1981.

[4] Richard Arratia, Eric S. Lander, Simon Tavare und Michael S. Waterman. Genomic mapping by anchoring random clones: A mathematical analysis. *Genomics*, 11(4):806–827, Dezember 1991.

[5] Susanne Balzer, Ketil Malde, Anders Lanzen, Animesh Sharma und Inge Jonassen. Characteristics of 454 pyrosequencing data - enabling realistic simulation with flowsim. *Bioinformatics*, 26(18):i420–i425, 2010.

[6] Serafim Batzoglou, David B. Jaffe, Ken Stanley, Jonathan Butler, Sante Gnerre, Evan Mauceli, Bonnie Berger, Jill P. Mesirov und Eric S. Lander. Arachne: A whole-genome shotgun assembler. *Genome Research*, 12(1):177–189, Januar 2002.

[7] Mark Chaisson, Pavel Pevzner und Haixu Tang. Fragment assembly with short reads. *Bioinformatics*, 20(13):2067–2074, September 2004.

[8] Louise Clarke und John Carbon. A colony bank containing synthetic coi ei hybrid plasmids representative of the entire e. coli genome. *Cell*, 9(1):91–99, September 1976.

[9] Chimpanzee Sequencing Consortium. Initial sequence of the chimpanzee genome and comparison wih the human genomes. *Nature*, 437:69–87, 2005.

[10] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman und John Good. The cost of doing science on the cloud: the montage example. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, Seiten 50:1–50:12, Piscataway, NJ, USA, 2008. IEEE Press.

[11] Prescott L. Deininger. Random subcloning of sonicated dna: Application to shotgun dna sequence analysis. *Analytical Biochemistry*, 129(1):216–223, Februar 1983.

[12] Juliane C. Dohm, Claudio Lottaz, Tatiana Borodina und Heinz Himmelbauer. Sharcgs, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. *Genome Research*, 17(11):1697–1706, November 2007.

[13] Juliane C. Dohm, Claudio Lottaz, Tatiana Borodina und Heinz Himmelbauer. Substantial biases in ultra-short read data sets from high-throughput dna sequencing. *Nucleic Acids Research*, 36(16):e105–e105, September 2008.

[14] Jaliya Ekanayake und Geoffrey Fox. High performance parallel computing with clouds and cloud technologies. In DimiterR. Avresky, Michel Diaz, Arndt Bode, Bruno Ciciani und Eliezer Dekel, editors, *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, volume 34, Seiten 20–38. Springer Berlin Heidelberg, 2010.

[15] John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North und Gordon Woodhull. Graphviz - open source graph drawing tools. *Graph Drawing*, Seiten 483–484, 2001.

[16] Yaniv Erlich, Partha P Mitra, Melissa delaBastide, W Richard McCombie und Gregory J Hannon. Alta-cyclic: a self-optimizing base caller for next-generation sequencing. *Nat Meth*, 5(8):679–682, August 2008.

[17] Constantinos Evangelinos, Pierre F. J. Lermusiaux, Jinshan Xu, Patrick J. Haley und Chris N. Hill. Many task computing for multidisciplinary ocean

sciences: real-time uncertainty prediction and data assimilation. In *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, MTAGS '09, Seiten 14:1–14:10, New York, NY, USA, 2009. ACM.

[18] W. Feller. Introduction to probability theory and its applications (3rd ed.). *Wiley*, 1968.

[19] Paolo Ferragina und Giovanni Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, Seiten 390–398. IEEE, 2000.

[20] R. A. Fisher. Tests of significance in harmonic analysis. In *Proceedings of Royal Society London*, 1929.

[21] RD Fleischmann, MD Adams, O White, RA Clayton, EF Kirkness, AR Kerlavage, CJ Bult, JF Tomb, BA Dougherty, JM Merrick und et al. Whole-genome random sequencing and assembly of haemophilus influenzae rd. *Science*, 269(5223):496–512, Juli 1995.

[22] R.A Gibbs, G.M. Weinstock, M.L. Metzker, D.M. Muzny, E.J. Sodergren und S. Scherer. Genome sequence of the brown norway rat yields insights into mammalian evolution. *Nature*, 428:493–521, 2004.

[23] Franziska Grohmann. Fehlerkorrektur von short-reads unter verwendung von suffixbaeumen. Diplomarbeit, 2011.

[24] William Gropp. Mpich2: A new start for mpi implementations. In Dieter Kranzlmueller, Jens Volkert, Peter Kacsuk und Jack Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 2474 of *Lecture Notes in Computer Science*, Seiten 37–42. Springer Berlin / Heidelberg, 2002.

[25] Genome Bioinformatics Group. genome.ucsc.edu.

[26] Elisabeth Guenther. *Lehrbuch der Genetik*. Gustav Fischer Verlag Jena, 1991.

[27] Scott Hazelhurst. Scientific computing using virtual high-performance computing: a case study using the amazon elastic computing cloud. In *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology*, SAICSIT '08, Seiten 94–103, New York, NY, USA, 2008. ACM.

[28] Manuel Holtgrewe. Mason - a read simulator for second generation sequencing data. Frei Universitaet Berlin, 2010.

[29] Lucian Ilie, Farideh Fazayeli und Silvana Ilie. Hitec: accurate error correction in high-throughput sequencing data. *Bioinformatics*, Seiten –, 2010.

[30] K.R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H.J. Wasserman und N.J. Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference*, Seiten 159 –168, 30 2010-dec. 3 2010.

[31] Wei-Chun Kao, Andrew H. Chan und Yun S. Song. Echo: A reference-free short-read error correction algorithm. *Genome Research*, 21(7):1181–1192, Juli 2011.

[32] Wei-Chun Kao, Kristian Stevens und Yun S. Song. Bayescall: A model-based base-calling algorithm for high-throughput short-read sequencing. *Genome Research*, 19(10):1884–1895, Oktober 2009.

[33] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman und M. Tsugawa. Science clouds: Early experiences in cloud computing for scientific applications. *Cloud Computing and Applications*, 2008.

[34] David R Kelley, Michael C Schatz und Steven L Salzberg. Quake: quality-aware detection and correction of sequencing errors. *Genome Biology*, 11,11, 2010.

[35] Petr Klus, Simon Lam, Dag Lyberg, Ming S Cheung, Graham Pullan, Ian McFarlane, Giles SH Yeo und Brian YH Lam. Barracuda-a fast short read sequence aligner using graphics processing units. *BMC research notes*, 5(1):27, 2012.

[36] Tak Wah Lam, Ruiqiang Li, Alan Tam, Simon Wong, Edward Wu und Siu-Ming Yiu. High throughput short read alignment via bi-directional bwt. In *Bioinformatics and Biomedicine, 2009. BIBM'09. IEEE International Conference on*, Seiten 31–36. IEEE, 2009.

[37] Eric S. Lander und Michael S. Waterman. Genomic mapping by fingerprinting random clones: A mathematical analysis. *Genomics*, Volume 2, Issue 3:231–239, 1988.

[38] Ben Langmead und Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nat Meth*, 9(4):357–359, April 2012.

[39] Ben Langmead, Cole Trapnell, Mihai Pop und Steven L. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, 10:R25, 2009.

[40] Benjamin Langmead. Highly scalable short read alignment with the burrows-wheeler transform and cloud computing. Master Thesis, University of Maryland, 2009.

[41] Heng Li und Richard Durbin. Fast and accurate short read alignment with burrows-wheeler transform. *Bioinformatics*, 2009.

[42] Heng Li und Richard Durbin. Fast and accurate long-read alignment with burrows-wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

[43] Ruiqiang Li, Chang Yu, Yingrui Li, Tak-Wah Lam, Siu-Ming Yiu, Karsten Kristianse und Jun Wang. Soap2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25 no. 15 2009:1966–1967, 2009.

[44] Ryan Lister, Mattia Pelizzola, Robert H. Dowen, R. David Hawkins, Gary Hon, Julian Tonti-Filippini, Joseph R. Nery, Leonard Lee, Zhen Ye, Que-Minh Ngo, Lee Edsall, Jessica Antosiewicz-Bourget, Ron Stewart, Victor Ruotti, A. Harvey Millar, James A. Thomson, Bing Ren und Joseph R. Ecker. Human dna methylomes at base resolution show widespread epigenomic differences. *Nature*, 462(7271):315–322, November 2009.

[45] Chi-Man Liu, Thomas Wong, Edward Wu, Ruibang Luo, Siu-Ming Yiu, Yingrui Li, Bingqiang Wang, Chang Yu, Xiaowen Chu, Kaiyong Zhao, Ruiqiang

Li und Tak-Wah Lam. Soap3: ultra-fast gpu-based parallel alignment tool for short reads. *Bioinformatics*, 28, 2012.

[46] Yongchao Liu, Bertil Schmidt und Douglas L Maskell. Cushaw: a cuda compatible short read aligner to large genomes based on the burrows-wheeler transform. *Bioinformatics*, 28(14):1830–1837, 2012.

[47] Ruibang Luo, Thomas Wong, Jianqiao Zhu, Chi-Man Liu, Xiaoqian Zhu, Edward Wu, Lap-Kei Lee, Haoxiang Lin, Wenjuan Zhu, David W. Cheung, Hing-Fung Ting, Siu-Ming Yiu, Shaoliang Peng, Chang Yu, Yingrui Li, Ruiqiang Li und Tak-Wah Lam. Soap3-dp: Fast, accurate and sensitive gpu-based short read aligner. *PLOS ONE*, 8, Issue 5, 2013.

[48] Allan M Maxam und Walter Gilbert. A new method for sequencing dna. *Proceedings of the National Academy of Sciences*, 74(2):560–564, 1977.

[49] Sascha Moeller. Short-read-alignment unter verwendung von hash-maps. 2010.

[50] Michael W Nachman und Susan L Crowell. Estimate of the mutation rate per nucleotide in humans. *Genetics*, 156(1):297–304, 2000.

[51] Jeffrey Napper und Paolo Bientinesi. Can cloud computing reach the top500? In *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, UCHPC-MAW '09, Seiten 17–20, New York, NY, USA, 2009. ACM.

[52] Saul B. Needleman und Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, März 1970.

[53] Stephen C. North. Drawing graphs with neato. *NEATO User Manual, April*, 2004.

[54] Herbert Oberacher und Florian Pitterl. Tandem mass spectrometric de novo sequencing of oligonucleotides using simulated annealing for stochastic optimization. *International Journal of Mass Spectrometry*, 304(2-3):124–129, Juli 2011.

[55] Mayur R. Palankar, Adriana Iamnitchi, Matei Ripeanu und Simson Garfinkel. Amazon s3 for science grids: a viable solution? In *Proceedings of the 2008 international workshop on Data-aware distributed computing*, DADC '08, Seiten 55–64, New York, NY, USA, 2008. ACM.

[56] Sangmi Lee Pallickara, Marlon Pierce, Qunfeng Dong und Chinhua Kong. Enabling large scale scientific computations for expressed sequence tag sequencing over grid and cloud computing clusters. In *PPAM 2009 Eigth International Conference on Parallel Processing and Applied Mathematics*, Seiten 13–16, 2009.

[57] N. Paulsen, F. Schatz und M. Schimmler. Hochperformantes hashbasiertes short-read-alignment. In *Studierendentagung zu den Life Sciences in Kiel*, 2013.

[58] Niklas Paulsen. Hoch performantes short-read alignment durch weiterentwicklung des smarti-algorithmus. 2012.

[59] Pavel A. Pevzner, Haixu Tang und Michael S. Waterman. An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, August 2001.

[60] Ethan Port, Fengzhu Sun, Daniela Martin und Michael S. Waterman. Genomic mapping by end-characterized random clones: a mathematical analysis. *Genomics*, 26(1):84–100, März 1995.

[61] J Rehr, F Vila, J Gardner, L Svec und M Prange. Scientific computing in the cloud. *Computing in Science Engineering*, PP(99):1, 2011.

[62] Jared C. Roach, Cecilie Boysen, Kai Wang und Leroy Hood. Pairwise end sequencing: a unified approach to genomic mapping and sequencing. *Genomics*, 26(2):345–353, März 1995.

[63] Jacques Rougemont, Arnaud Amzallag, Christian Iseli, Laurent Farinelli, Ioannis Xenarios und Felix Naef. Probabilistic base calling of solexa sequencing data. *BMC Bioinformatics*, 9(1):431, 2008.

[64] Leena Salmela. Correction of sequencing errors in a mixed set of reads. *Bioinformatics*, 26(10):1284–1290, Mai 2010.

[65] Leena Salmela und Jan Schroeder. Correcting errors in short reads by multiple alignments. *Bioinformatics*, 27(11):1455–1461, Juni 2011.

[66] F. Sanger, A.R. Coulson, B.G. Barrell, A.J.H. Smith und B.A. Roe. Cloning in single-stranded bacteriophage as an aid to rapid dna sequencing. *Journal of Molecular Biology*, 143(2):161–178, Oktober 1980.

[67] Frederick Sanger, Steven Nicklen und Alan R Coulson. Dna sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*, 74(12):5463–5467, 1977.

[68] Florian Schatz, Sven Koschnicke, Niklas Paulsen und Manfred Schimmler. Master/slave assignment optimization for high performance computing in an ec2 cloud using mpi. *Network Protocols and Algorithms*, 4:22–33, 2012.

[69] Florian Schatz, Sven Koschnicke, Niklas Paulsen, Christoph Starke und Manfred Schimmler. Mpi performance analysis of amazon ec2 cloud services for high performance computing. In Ajith Abraham, Jaime Lloret Mauri, John F. Buford, Junichi Suzuki und Sabu M. Thampi, editors, *Advances in Computing and Communications*, volume 190 of *Communications in Computer and Information Science*, Seiten 371–381. Springer Berlin Heidelberg, 2011.

[70] Florian Schatz, Sascha Moeller und Manfred Schimmler. Smarti - a fast short read alignment algorithm (eingereicht). In *RECOMB Satellite Workshop on Massively Parallel Sequencing*, 2011.

[71] Florian Schatz und Manfred Schimmler. Boundaries for short-read, low coverage denovo assembly and resequencing. 9th European Conference on Computational Biology, ECCB10, Gent, Belgium, Poster presentation, September 2010.

[72] Florian Schatz und Manfred Schimmler. A priori estimation of contig length distribution of short-read sequencing. Research in Computational Molecular Biology, 14th Annual International Conference, RECOMB 2010, Poster presentation, August 2010.

[73] Florian Schatz, Lars Wienbrandt und Manfred Schimmler. Probability model for boundaries of short-read sequencing. In *Advances in Computing and*

Communications (ICACC), 2012 International Conference, Seiten 223–228, 2012.

[74] Michael C. Schatz. Cloudburst: highly sensitive read mapping with mapreduce. *Bioinformatics*, 29 no. 11:1363–1369, 2009.

[75] Jan Schroeder, Heiko Schroeder, Simon J. Puglisi, Ranjan Sinha und Bertil Schmidt. Shrec: a short-read error correction method. *Bioinformatics*, 25(17):2157–2163, September 2009.

[76] A.F. Siegel. Random arcs on the circle. *Journal of Applied Probability*, 15:774–789, 1978.

[77] Peter Skomoroch. Mpi cluster with python and amazon ec2. http://www.datawrangling.com/mpi-cluster-with-python-and-amazon-ec2-part-2-of-3, 2009.

[78] Andrew Smith, Zhenyu Xuan und Michael Zhang. Using quality scores and longer reads improves accuracy of solexa read mapping. *BMC Bioinformatics*, 9(1):128, 2008.

[79] T. F. Smith und M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[80] J.G. Stadel. *Cosmological N–body simulations and their analysis*. PhD thesis, University of Washington, 2001.

[81] W.L. Stevens. Solution to a geometrical problem in probability. *Ann. Eugenics*, 9:315–320, 1939.

[82] Martti T. Tammi, Erik Arner, Ellen Kindlund und Bjoern Andersson. Correcting errors in shotgun sequences. *Nucleic Acids Research*, 31(15):4663–4672, August 2003.

[83] Margaret Taub, Hector Corrada Bravo und Rafael Irizarry. Overcoming bias and systematic errors in next generation sequencing data. *Genome Medicine*, 2(12):87, 2010.

[84] Silvana van Koningsbruggen, Marek Gierlinski, Piet Schofield, David Martin, Geoffey J Barton, Yavuz Ariyurek, Johan T den Dunnen und Angus I Lamond. High-resolution whole-genome sequencing reveals that specific chromatin domains from most human chromosomes associate with nucleoli. *Molecular biology of the cell*, 21(21):3735–3748, 2010.

[85] J Craig Venter, Mark D Adams, Eugene W Myers, Peter W Li, Richard J Mural, Granger G Sutton, Hamilton O Smith, Mark Yandell, Cheryl A Evans und Robert A Holt. The sequence of the human genome. *science*, 291(5507):1304–1351, 2001.

[86] J. Craig Venter, Mark D. Adams, Eugene W. Myers, Peter W. Li, Richard J. Mural, Granger G. Sutton, Hamilton O. Smith, Mark Yandell, Cheryl A. Evans, Robert A. Holt, Jeannine D. Gocayne, Peter Amanatides, Richard M. Ballew, Daniel H. Huson, Jennifer Russo Wortman, Qing Zhang, Chinnappa D. Kodira, Xiangqun H. Zheng, Lin Chen, Marian Skupski, Gangadharan Subramanian, Paul D. Thomas, Jinghui Zhang, George L. Gabor Miklos, Catherine Nelson, Samuel Broder, Andrew G. Clark, Joe Nadeau, Victor A. McKusick, Norton Zinder, Arnold J. Levine, Richard J. Roberts, Mel Simon, Carolyn Slayman, Michael Hunkapiller, Randall Bolanos, Arthur Delcher, Ian Dew, Daniel Fasulo, Michael Flanigan, Liliana Florea, Aaron Halpern, Sridhar Hannenhalli, Saul Kravitz, Samuel Levy, Clark Mobarry, Knut Reinert, Karin Remington, Jane Abu-Threideh, Ellen Beasley, Kendra Biddick, Vivien Bonazzi, Rhonda Brandon, Michele Cargill, Ishwar Chandramouliswaran, Rosane Charlab, Kabir Chaturvedi, Zuoming Deng, Valentina Di Francesco, Patrick Dunn, Karen Eilbeck, Carlos Evangelista, Andrei E. Gabrielian, Weiniu Gan, Wangmao Ge, Fangcheng Gong, Zhiping Gu, Ping Guan, Thomas J. Heiman, Maureen E. Higgins, Rui-Ru Ji, Zhaoxi Ke, Karen A. Ketchum, Zhongwu Lai, Yiding Lei, Zhenya Li, Jiayin Li, Yong Liang, Xiaoying Lin, Fu Lu, Gennady V. Merkulov, Natalia Milshina, Helen M. Moore, Ashwinikumar K Naik, Vaibhav A. Narayan, Beena Neelam, Deborah Nusskern, Douglas B. Rusch, Steven Salzberg, Wei Shao, Bixiong Shue, Jingtao Sun, Zhen Yuan Wang, Aihui Wang, Xin Wang, Jian Wang, Ming-Hui Wei, Ron Wides, Chunlin Xiao, Chunhua Yan, Alison Yao, Jane Ye, Ming Zhan, Weiqing Zhang, Hongyu Zhang, Qi Zhao, Liansheng Zheng, Fei

Zhong, Wenyan Zhong, Shiaoping C. Zhu, Shaying Zhao, Dennis Gilbert, Suzanna Baumhueter, Gene Spier, Christine Carter, Anibal Cravchik, Trevor Woodage, Feroze Ali, Huijin An, Aderonke Awe, Danita Baldwin, Holly Baden, Mary Barnstead, Ian Barrow, Karen Beeson, Dana Busam, Amy Carver, Angela Center, Ming Lai Cheng, Liz Curry, Steve Danaher, Lionel Davenport, Raymond Desilets, Susanne Dietz, Kristina Dodson, Lisa Doup, Steven Ferriera, Neha Garg, Andres Gluecksmann, Brit Hart, Jason Haynes, Charles Haynes, Cheryl Heiner, Suzanne Hladun, Damon Hostin, Jarrett Houck, Timothy Howland, Chinyere Ibegwam, Jeffery Johnson, Francis Kalush, Lesley Kline, Shashi Koduru, Amy Love, Felecia Mann, David May, Steven McCawley, Tina McIntosh, Ivy McMullen, Mee Moy, Linda Moy, Brian Murphy, Keith Nelson, Cynthia Pfannkoch, Eric Pratts, Vinita Puri, Hina Qureshi, Matthew Reardon, Robert Rodriguez, Yu-Hui Rogers, Deanna Romblad, Bob Ruhfel, Richard Scott, Cynthia Sitter, Michelle Smallwood, Erin Stewart, Renee Strong, Ellen Suh, Reginald Thomas, Ni Ni Tint, Sukyee Tse, Claire Vech, Gary Wang, Jeremy Wetter, Sherita Williams, Monica Williams, Sandra Windsor, Emily Winn-Deen, Keriellen Wolfe, Jayshree Zaveri, Karena Zaveri, Josep F. Abril, Roderic Guigo, Michael J. Campbell, Kimmen V. Sjolander, Brian Karlak, Anish Kejariwal, Huaiyu Mi, Betty Lazareva, Thomas Hatton, Apurva Narechania, Karen Diemer, Anushya Muruganujan, Nan Guo, Shinji Sato, Vineet Bafna, Sorin Istrail, Ross Lippert, Russell Schwartz, Brian Walenz, Shibu Yooseph, David Allen, Anand Basu, James Baxendale, Louis Blick, Marcelo Caminha, John Carnes-Stine, Parris Caulk, Yen-Hui Chiang, My Coyne, Carl Dahlke, Anne Deslattes Mays, Maria Dombroski, Michael Donnelly, Dale Ely, Shiva Esparham, Carl Fosler, Harold Gire, Stephen Glanowski, Kenneth Glasser, Anna Glodek, Mark Gorokhov, Ken Graham, Barry Gropman, Michael Harris, Jeremy Heil, Scott Henderson, Jeffrey Hoover, Donald Jennings, Catherine Jordan, James Jordan, John Kasha, Leonid Kagan, Cheryl Kraft, Alexander Levitsky, Mark Lewis, Xiangjun Liu, John Lopez, Daniel Ma, William Majoros, Joe McDaniel, Sean Murphy, Matthew Newman, Trung Nguyen, Ngoc Nguyen, Marc Nodell, Sue Pan, Jim Peck, Marshall Peterson, William Rowe, Robert Sanders, John Scott, Michael Simpson, Thomas Smith, Arlan Sprague, Timothy Stockwell, Russell Turner, Eli Venter, Mei Wang, Meiyuan Wen, David Wu,

Mitchell Wu, Ashley Xia, Ali Zandieh und Xiaohong Zhu. The sequence of the human genome. *Science*, 291(5507):1304–1351, Februar 2001.

[87] Guohui Wang und T.S.E. Ng. The impact of virtualization on network performance of amazon ec2 data center. In *INFOCOM, 2010 Proceedings IEEE*, Seiten 1–9, März 2010.

[88] Robert Waterston, Asif T. Chinwalla, Lisa L. Cook, Kimberly D. Delehaunty und et al. Initial sequencing and comparative analysis of the mouse genome. *Nature*, 420(6915):520–562, Dezember 2002.

[89] Roche Website. http://454.com/products/gs-flx-system/index.asp.

[90] Michael Wendl. Occupancy modeling of coverage distribution for whole genome shotgun dna sequencing. *Bulletin of Mathematical Biology*, 68(1):179–196, Januar 2006.

[91] Michael C. Wendl und Robert H. Waterston. Generalized gap model for bacterial artificial chromosome clone fingerprint mapping and shotgun sequencing. *Genome Res.*, 12:1943–1949, 2002.

[92] Lars Wienbrandt, Stefan Baumgart, Jost Bissel, Florian Schatz und Manfred Schimmler. Massively parallel fpga-based implementation of blastp with the two-hit method. *Procedia Computer Science*, 4:1967–1976, 2011.

[93] L.V. Yakushevich. *Nonlinear Physics of DNA*. Johns Wiley & Sons, 1998.

[94] Xiao Yang, Sriram P. Chockalingam und Srinivas Aluru. A survey of error-correction methods for next-generation sequencing. *Briefings in Bioinformatics*, April 2012.

[95] Xiao Yang, Karin S. Dorman und Srinivas Aluru. Reptile: representative tiling for short read error correction. *Bioinformatics*, 26(20):2526–2533, Oktober 2010.

[96] M Q Zhang und T G Marr. Genome mapping by nonrandom anchoring: a discrete theoretical analysis. *Proceedings of the National Academy of Sciences*, 90(2):600–604, Januar 1993.

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass diese Arbeit, abgesehen von der Beratung durch die Betreuerin oder den Betreuer, nach Inhalt und Form die eigene Arbeit ist, dass die Arbeit ganz oder zum Teil keiner anderen Stelle im Rahmen eines Prüfungsverfahrens vorgelegen hat, veröffentlicht worden ist oder zur Veröffentlichung eingereicht wurde, dass die Arbeit unter Einhaltung der Regeln guter wissenschaftlicher Praxis der Deutschen Forschungsgemeinschaft entstanden ist.

Kiel, den

Florian Schatz