

Bachelorarbeit

**Analyse von RAD-Seq-Daten als Optimierungsproblem unter Berücksichtigung von
Sequenzierfehler- und Mutationsraten**

Antonie Vietor

Gutachter:

Dr. rer. nat. Johannes Köster

Prof. Dr. rer. nat. Sven Rahmann

Technische Universität Dortmund

Fakultät für Informatik

Lehrstuhl 11

Bioinformatics for High-Throughput Technologies

<http://ls11-www.cs.tu-dortmund.de/>

In Kooperation mit:

Universität Duisburg-Essen

Genome Informatics

<http://genomeinformatics.uni-due.de/>

Inhaltsverzeichnis

1	Einleitung	1
1.1	Biologischer Hintergrund	1
1.1.1	Aufbau und Struktur der DNA	1
1.1.2	Bindungen innerhalb und zwischen DNA-Molekülen	2
1.1.3	RNA und die Proteinbiosynthese	3
1.1.4	DNA-Replikation	4
1.1.5	Mutationen und SNPs	5
1.2	Molekulargenetische Verfahren und Techniken	7
1.2.1	Sanger-Sequenzierung	7
1.2.2	PCR	8
1.2.3	Next Generation Sequencing	8
1.3	RAD-Sequencing	8
1.3.1	Verfahren	8
2	Analyse von RAD-Seq-Daten	9
2.1	Problemstellung	9
2.2	Formale Definition	9
2.3	Lösungsansatz	10
2.3.1	Graph und Zusammenhangskomponenten	10
2.3.2	Pair Hidden Markov Model	10
2.3.3	Die maximale Likelihood der Allele	10
2.3.4	Die maximale Likelihood der Loci	12
2.3.5	Varianten und Genotyp	13
3	Algorithmus	15
3.1	Preprocessing	15
3.2	Edit-Distanzen	16
3.3	Konstruktion des Graphen	16
3.3.1	Knoten des Graphen	16
3.3.2	Kanten des Graphen	17

3.3.3	Bestimmung der Zusammenhangskomponenten	22
3.3.4	Laufzeitanalyse zur Konstruktion des Graphen	23
3.4	Bestimmung der maximalen Likelihood der Allele	23
3.4.1	Bestimmung der Allele und ihrer möglichen Häufigkeitsverteilung . .	23
3.4.2	Berechnung der Likelihoods der Allele anhand der möglichen Häufigkeitsverteilungen	28
3.4.3	Laufzeitanalyse für die Bestimmung der Allelfraction mit maximaler Likelihood	35
3.5	Bestimmung der maximalen Likelihood der Loci	35
3.5.1	Bestimmung der möglichen Loci	36
3.5.2	Zuordnung der Allelverteilung mit maximaler Likelihood zur wahrscheinlichsten Loci-Kombination	38
3.5.3	Laufzeitanalyse für die Bestimmung der Loci-Kombination mit maximaler Likelihood	41
3.6	Abschließende Laufzeitanalyse des gesamten Algorithmus	42
3.7	Ausgabe der wahrscheinlichsten Loci als VCF-Datei	43
4	Evaluation an simulierten Datensätzen	45
4.1	45
4.1.1	45
5	Zusammenfassung und Ausblick	47
5.1	47
5.1.1	47
	Algorithmenverzeichnis	49
	Literaturverzeichnis	55
	Erklärung	55

Kapitel 1

Einleitung

1.1 Biologischer Hintergrund

1.1.1 Aufbau und Struktur der DNA

In den vergangenen Jahren wurden durch die molekulargenetische Methoden in Medizin und Biologie enorme Fortschritte erzielt. Heute sind sie nicht nur ein wesentliches Instrument bei der Erforschung, Diagnostik und Therapie verschiedenster Erkrankungen sondern sind auch bei der Entdeckung und Klassifikation von Organismen oder ganzen Ökosystemen von entscheidender Bedeutung.

Einer der ersten und wichtigsten Meilensteine auf dem noch eher jungen Gebiet der Molekulargenetik wurde 1953 durch die Entdeckung der Doppelhelixstruktur der DNA und die Beschreibung ihres Aufbaus erreicht [1]. Die DNA (desoxyribonucleic acid) ist ein langkettiges, aus zwei gegenläufigen, komplementären Strängen bestehendes und zu einer Helix gewundenes Molekül, welches die Erbinformation der meisten Zellen codiert. Die Komplementarität und Gegenläufigkeit werden in Kap. 1.1.2 gesondert beschrieben. Jeder Strang besteht aus vielen aneinandergereihten Nukleotiden, die sich jeweils aus einem Zuckermolekül (Desoxyribose), einem Phosphatrest und einer von vier möglichen Basen zusammensetzen. Als Basen kommen in der DNA Adenin (A), Thymin (T), Guanin (G) und Cytosin (C) vor. Die Kombinationen dieser Basen codieren über ihre Sequenz die genetische Information.

Hierbei codieren im Rahmen der Proteinbiosynthese (Kap. 1.1.3) Kombinationen aus jeweils drei Basen, sog. Basentriplets bzw. Codons, entweder für eine von i.d.R. 20 Aminosäure [2, 3] oder für Start- bzw. Stop-Sequenzen, welche den Anfang bzw. das Ende von Genen signalisieren (genetischer Code). Hiervon codieren 61 Triplets für die bereits erwähnten 20 Aminosäuren. Somit codieren für die meisten Aminosäuren mehrere verschiedene Triplets, diese Eigenschaft des genetischen Codes wird auch als Degeneration

bezeichnet.

Die Basensequenz der DNA entspricht somit einer Aminosäuresequenz, welche die Primärstruktur von Proteinen (Eiweißmolekülen) darstellt. Proteine erfüllen in lebenden Organismen umfangreiche Funktionen, sie können als Hormone, Enzyme, Strukturproteine fungieren und sind an den meisten Stoffwechselprozessen und Signalwegen von Zellen beteiligt. Die verschiedenen Proteine werden jeweils von für sie spezifischen Abschnitten auf der DNA codiert. Solche DNA-Abschnitte werden als Gene bezeichnet. Ein DNA-Strang besteht also aus vielen verschiedenen Genen. In komplexeren Zellen befinden sich im Zellkern in der Regel mehrere DNA-Stränge, die jeweils ein Chromosom repräsentieren, auf dem sich jeweils mehrere Gene befinden. Die Anzahl der Chromosomen innerhalb der Zellen ist speziesabhängig.

Zwischen den einzelnen Genen eines DNA-Stranges befinden sich nicht-codierende und oft repetitive Sequenzen. Ebenso gibt es auch innerhalb der Gene codierende Abschnitte (Exons) und nicht-codierende Sequenzen (Introns). Insgesamt machen die für Proteine codierenden Bereiche der DNA nur einen geringen Anteil des Genoms, also der gesamten Erbinformation einer Zelle aus. Beim Menschen wird dieser Anteil auf etwa 2 % geschätzt, d.h. ca. 98 % des menschlichen Genoms besteht aus DNA, die nicht für Proteine codiert. Über diese nicht-codierenden Abschnitte ist bislang nur wenig bekannt, teilweise werden ihnen regulatorische Funktionen zugeschrieben [4, 5].

1.1.2 Bindungen innerhalb und zwischen DNA-Molekülen

DNA liegt meist in Form eines Doppelstrangs vor. Die Basen beider Stränge sind dabei intermolekular über schwache chemische Bindungen, sogenannte Wasserstoffbrücken, mit einander verbunden. Dabei kann Adenin nur an Thymin unter Ausbildung von zwei Wasserstoffbrückenbindungen binden. Ebenso kann Cytosin nur mit Guanin über insgesamt drei Wasserstoffbrücken eine Bindung eingehen. Diese selektive Basenpaarung wird auch als Komplementarität bezeichnet. Es sind also Adenin und Thymin ebenso wie Cytosin und Guanin jeweils komplementär zu einander. Bezogen auf einen DNA-Doppelstrang sind auch seine beiden Einzelstränge komplementär, so dass sich für jede Base des einen Stranges auf dem anderen Strang jeweils die komplementäre Base an der entsprechenden Position befindet. Es genügt also so die Sequenz von einem der beiden Stränge zu kennen, um die Sequenz des jeweils anderen rekonstruieren zu können. Dies wird sowohl zum Auslesen der genetischen Informationen bei der Transkription (Kap. ??) als auch zum Kopieren von DNA im Rahmen der DNA-Replikation (Kap. 1.1.4) genutzt.

Wie bereits in Kap. 1.1.1 erwähnt, ist doppelsträngige DNA gegenläufig orientiert. Die Einzelstränge besitzen eine Polarität, welche durch die intramolekulare Bindung zwischen den einzelnen Nukleotiden über sogenannte Phosphodiesterbindungen zustande kommt. Dabei bindet der Phosphatrest, der sich jeweils am 5. Kohlenstoffatom (C5) des Zuckermoleküls der Nukleotide befindet, an das 3. Kohlenstoffatom (C3) des Zuckermoleküls des nachfolgenden Nukleotids. An den Enden eines DNA-Strangs fehlt jedoch diese Phosphodiesterbindung, so dass an einem Ende das C3 ungebunden bleibt (3'-Ende) während am anderen Ende der Phosphatrest nur an C5 gebunden ist (5'-Ende) und somit die zweite Esterbindung am Phosphatrestes fehlt. Aufgrund der Gegenläufigkeit befindet sich also an beiden Enden eines Doppelstrangs jeweils ein 3'-Ende des einen und ein 5'-Ende des anderen Einzelstrangs. Diese Polarität spielt eine wichtige Rolle bei der Lese- und Syntheserichtung im Rahmen der Transkription und der DNA-Replikation.

1.1.3 RNA und die Proteinbiosynthese

Ebenso wie DNA gehört auch RNA (ribonucleic acid) zu den Nukleinsäuren. Sie unterscheidet sich in ihrem Aufbau von der DNA durch die Base Uracil (U) statt Thymin und den Zucker Ribose statt Desoxyribose. Meist liegt RNA einzelsträngig oder nur über kürzere Abschnitte doppelsträngig vor. Während DNA insbesondere der Speicherung der Erbinformation dient, hat RNA eher die Funktion der Informationsübertragung. RNA nimmt daher zahlreiche regulatorische Funktionen wahr.

Eine ihrer wichtigsten Aufgaben ist die Übertragung der genetischen Information von der DNA in die Aminosäuresequenz der Proteine bei der Proteinbiosynthese. Dabei werden von dem für das herzustellende Protein codierenden DNA-Abschnitt zunächst Arbeitskopien in Form von mRNA (messenger RNA) hergestellt. Ein solches Umschreiben von DNA in mRNA wird auch als Transkription bezeichnet. Nach der Transkription erhalten die mRNA-Fragmente noch einige Modifikationen und werden aus dem Zellkern hinaus in das Cytoplasma transportiert. Im Cytoplasma erfolgt schließlich mit Hilfe sogenannter tRNAs (transfer RNA) die Übersetzung der Basentriplets in eine Aminosäuresequenz (Translation, siehe auch Kap. 1.1.1). tRNAs besitzen eine Bindungsstelle bestehend aus jeweils drei Nukleotiden, mit der sie komplementär an ein passendes Basentriplett der mRNA binden. In Abhängigkeit vom Basentriplett an ihrer mRNA-Bindungsstelle trägt jede tRNA entsprechend dem genetischen Code eine spezifische Aminosäure. Entlang der mRNA wird nun ab der Startsequenz für jedes Basentriplett die passende tRNA nacheinander angelagert. Sobald eine tRNA bindet, wird die an sie gebundene Aminosäure gelöst und an die Aminosäure der nachfolgenden tRNA gebunden. Dadurch entsteht eine Kette von aneinander gebundene Aminosäuren, die sich jeweils entsprechend der mRNA Sequenz an die Aminosäure der nächsten passenden tRNA anlagert und um deren Aminosäure verlängert.

Beim Erreichen einer Stop-Sequenz kann diese keine tRNA angelagern, die Synthese wird abgebrochen und die Aminosäurekette löst sich von der zuletzt gebundenen tRNA und wird weiteren Modifikationen unterzogen.

1.1.4 DNA-Replikation

Die DNA-Replikation dient der Verdopplung der DNA im Rahmen der Zellteilung, so dass jede der beiden resultierenden Tochterzellen das gleiche genetische Material erhält. Die DNA-Replikation ist also ein natürlicher Vorgang zur Erzeugung von DNA-Kopien. Ihr grundlegendes Prinzip findet bei der PCR (Polymerase-Ketten-Reaktion, siehe Kap. 1.2.2) Anwendung und wird für verschiedene molekulargenetische Verfahren genutzt, um DNA-Kopien synthetisch herzustellen. In diesem Zusammenhang sei hier auf die verschiedenen Sequenzierungstechniken insbesondere im Hinblick auf das RAD-Sequencing verwiesen (Kap. 1.2.1, Kap. 1.2.3 und Kap. 1.3). Daher soll der Vorgang der DNA-Replikation im Folgenden kurz umrissen werden [6, 7, 8].

Bei eukaryotischen Zellen, die im Gegensatz zu Bakterien (Prokaryoten) einen Zellkern besitzen, liegt die DNA im Zellkern häufig gewunden und stark kondensiert vor. Um ihre Sequenz Base für Base kopieren zu können, muss sie zunächst entwunden werden. Dies geschieht durch Enzyme aus der Gruppe der Topoisomerasen. Diese erzeugen gezielt am Replikationsursprung temporäre Strangbrüche in der DNA und entspannen so den Doppelstrang. Anschließend setzt am Replikationsursprung ein weiteres Enzym, die Helikase, an und trennt die beiden komplementären Stränge auf. Es entsteht die sogenannte Replikationsgabel. Während der Replikation schiebt sich die Helikase unter fortlaufender Auftrennung der beiden Stränge auf der DNA entlang. Auch bei diesem Prozess sorgen Topoisomerasen immer wieder für eine Entspannung des DNA-Fadens hinsichtlich seiner Windung.

Nun kann der eigentliche Kopiervorgang an den beiden von einander getrennten Einzelsträngen mit Hilfe von DNA-Polymerasen erfolgen. Dabei fahren die DNA-Polymerasen an den Strängen entlang und fügen an jeder Position des Elternstranges Nukleotide mit der jeweils komplementären Base an. Im Ergebnis entsteht also an jedem der beiden Elternstränge ein neuer komplementärer DNA-Strang, der also die gleiche Basensequenz wie der jeweils andere Elternstrang besitzt. Bei den DNA-Polymerasen handelt es sich um Enzyme, die für die Initiation des Kopiervorgangs eine Startsequenz benötigen. Diese Startsequenzen sind kleine RNA-Fragmente mit spezifischer Basensequenz, die sich an die jeweils passende, komplementäre Stelle auf dem Elternstrang anlagern. An diese, auch als RNA-Primer bezeichneten Fragmente kann nun die DNA-Polymerase binden und mit der Replikation

beginnen.

Die DNA-Polymerase kann den Elternstrang nur in eine Richtung lesen, nämlich vom 3' Ende zum 5' Ende (3'-5'-Richtung). Aufgrund der gegenläufigen, antiparallelen Ausrichtung zweier komplementärer Stränge zu einander, kann folglich die Synthese des Tochterstranges nur in 5'-3'-Richtung erfolgen. Für die beiden antiparallel ausgerichteten Elternstränge bedeutet dies, dass nur bei einem Strang die Richtung der sich öffnenden Replikationsgabel der 5'-3'-Richtung des Stranges entspricht. Dieser Strang kann kontinuierlich repliziert werden, da sich die DNA-Polymerase auf ihm in Richtung der voranschreitenden Aufspaltung des Doppelstranges bewegt. Der auf diese Weise kontinuierlich synthetisierte Strang wird als Leitstrang bezeichnet.

Der andere Strang ist jedoch in Gegenrichtung orientiert, so dass seine Syntheserichtung, also die 5'-3'-Richtung, entgegengesetzt zur Bewegungsrichtung der Replikationsgabel orientiert ist. Dadurch können jeweils nur kleinere Fragmente synthetisiert werden die von der Replikationsgabel bis zum bereits replizierten Teil des Strangs reichen. Schreitet die Öffnung der Replikationsgabel weiter fort, muss der nun frei gewordene Strangabschnitt ebenfalls synthetisiert werden. Es muss also erneut ein RNA-Primer angelagert werden und dann mit Hilfe der DNA-Polymerase der Bereich zwischen Replikationsgabel und bereits replizierten Strang synthetisiert werden. Die Replikation erfolgt somit diskontinuierlich. Der so synthetisierte Strang wird als Folgestrang bezeichnet und besteht zunächst aus multiplen Fragmenten (Okazaki-Fragmente). Nach dem Replikationsvorgang werden mit Hilfe der DNA-Polymerase die RNA-Primer durch DNA ersetzt. Im Anschluss werden die multiplen Fragmente des Folgestrangs durch Ligasen zu einem kontinuierlichen Strang verbunden. Im Ergebnis sind also nach Abschluss der DNA-Replikation zwei identische Kopien der beiden Elternstränge entstanden.

1.1.5 Mutationen und SNPs

ToDo: ggf. Literatur zu den verschiedenen Mutationsarten ergänzen (DD: als Grundlagenwissen voraussetzen?)

Veränderungen in der DNA-Sequenz werden als Mutationen bezeichnet. Sie können durch zellinterne Faktoren verursacht werden, wie beispielsweise Fehler beim Kopiervorgang der DNA (DNA-Replikation) während der Zellteilung. Ebenso können sie durch zahlreiche Umwelteinflüsse entstehen.

Mutationen können unterschiedlich große Abschnitte der DNA betreffen, von ganzen Chromosomen oder großen Chromosomenabschnitten über Veränderungen von mehreren Basen bis hin zu sogenannten Punktmutationen, bei denen nur eine einzige Base verändert ist. Auf DNA-Ebene können Punktmutationen in Form Substitutionen, Insertionen und

Deletionen auftreten. Bei der Substitution wird eine Base durch eine andere ausgetauscht, bei der Insertion wird eine zusätzliche Base in den DNA-Strang eingefügt und bei der Deletion kommt es zum Verlust einer Base.

Liegt eine solche Punktmutation in den codierenden DNA-Abschnitten, so können sich auf Proteinebene verschiedene Konsequenzen daraus ergeben. Insertionen und Deletionen bewirken durch die zusätzliche bzw. fehlende Base eine Verschiebung des Leserasters, so dass sich die Tripletstruktur für alle nachfolgenden Basen verschiebt. Dies wird als Frame-Shift bezeichnet und verursacht meist eine gravierende Veränderung des resultierenden Proteins, da viele der nachfolgenden Triplets nun für andere Aminosäuren codieren. Dies führt häufig zu einem deutlich veränderten Protein, welches seine reguläre Funktion nicht mehr oder nur noch unvollständig wahrnehmen kann.

Bei Substitutionen bleibt dagegen das Leseraster erhalten. Aufgrund der Degeneration des genetischen Codes können verschiedene Basentriplets für die gleiche Aminosäure codieren. Dadurch kann ein Basentriplett mit einer Punktmutation trotz des Basenaustauschs noch für die ursprüngliche Aminosäure codieren, so dass das resultierende Protein unverändert bleibt. In diesem Fall spricht man von einer silent-Mutation. Codiert das Basentriplett aber aufgrund der Mutation für eine andere Aminosäure, so handelt es sich um eine missense-Mutation. Die Proteinsequenz wird dadurch in einer Aminosäure geändert, so dass es je nach Position der betreffenden Aminosäure zu verschieden starken Effekten hinsichtlich der Proteinfunktion kommen kann.

Zudem können Substitutionen und Frame-Shifts dazu führen dass ein für eine Aminosäure codierendes Basentriplett zu einem Stop-Codon umgewandelt wird (nonsense-Mutation) oder ein Stop-Codon durch die Mutation für eine Aminosäure codiert (readthrough-Mutation).

Hinsichtlich der Eigenschaften des resultierenden Proteins unterscheiden man im Zusammenhang mit Mutationen zudem zwischen sogenannten loss-of-function- und gain-of-function-Mutationen. Loss-of-function-Mutationen führen zu einer verringerten Funktionalität oder dem vollständigen Funktionsverlust des Proteins. Gain-of-function-Mutationen bewirken dagegen eine verstärkte oder veränderte Aktivität bzw. Funktionalität des Proteins.

Wie bereits erwähnt führen aber nicht alle Veränderungen der DNA-Sequenz zu Störungen der Genfunktion. Veränderungen ohne unmittelbaren Krankheitswert werden als genetische Varianten bezeichnet, wenn sie innerhalb einer Spezies vermehrt auftreten [9, 10]. Am häufigsten finden sich dabei Varianten einzelner Basenpaare, sogenannte SNP's (single nucleotide polymorphisms). SNP's kommen sowohl in codierenden als auch nicht-

codierenden DNA-Abschnitten vor und treten regionsabhängig in unterschiedliche Häufigkeit auf. SNP's können als genetische Marker benutzt werden [11, 12], ihr Auftreten und ihre Verteilung spielen vor allem in der Populationsgenetik eine wichtige Rolle. Sie können Aufschluss hinsichtlich der Diversität, Selektion und Demographie einer Population geben [13, 14, 15].

Während große strukturelle Chromosomenaberrationen unter Umständen bereits lichtmikroskopisch erkennbar sind, ist bei Punktmutationen oder SNP's lediglich eine einzige Base verändert. Solche Veränderungen lassen durch verschiedene molekulargenetische Verfahren detektieren [12, 16]. Insbesondere die direkte Analyse der DNA-Sequenz mittels Sequenzierung (siehe Kap. 1.2.1) ist durch die Entwicklung der sogenannten Next-Generation-Sequencing-Verfahren (NGS) im Hochdurchsatzverfahren und mit hoher Parallelisierung durchführbar (siehe Kap. 1.2.3). Diese Techniken ermöglichen inzwischen umfangreiche, genomweite Analysen hinsichtlich einer großen Vielfalt molekulargenetischer Fragestellungen. Die vorliegende Arbeit befasst sich mit der Analyse und Auswertung von RAD-Sequencing-Daten. Auch die RAD-Sequenzierung gehört zu den NGS-Verfahren und dient insbesondere der Detektion von SNP's. Daher wird dieses Verfahren in Kap. 1.3 detaillierter vorgestellt.

1.2 Molekulargenetische Verfahren und Techniken

1.2.1 Sanger-Sequenzierung

Nach der Erforschung der DNA-Struktur war schließlich die Entwicklung der Sanger-Sequenzierung im Jahr 1975 ein entscheidender Meilenstein der molekulargenetischen Forschung [17]. Durch sie war es erstmals möglich die genaue Basensequenz eines DNA-Strangs zu bestimmen.

Hierbei wird die zu sequenzierende DNA-Probe in vier Teile aufgeteilt, denen jeweils eine der vier DNA-Basen in Form von radioaktiv markierten synthetischen Nukleotiden, sowie anteilig einige modifizierte Nukleotiden dieser Base hinzugefügt werden. Die jeweils anderen drei Basen werden als unmarkierte und unmodifizierte Nukleotide hinzugegeben. In jeder der Probengemische ist also eine andere Base markiert und zum Teil auch modifiziert.

Wie bei der natürlichen DNA-Replikation (siehe Kap. 1.1.4) während der Zellteilung kann die Proben-DNA durch Hinzugabe der DNA-Polymerase I kopiert werden. Dabei werden auch die radioaktiv markierten Nukleotide in den kopierten Strang eingebaut. Die Kopiervorgänge starten jeweils an einem kleinen Fragment mit bekannter DNA-Sequenz, dem sog. Primer. Auch die Primer werden vorab den Probengemischen beigelegt. Der Primer

bindet komplementär an die Ausgangs-DNA der Probe und ermöglicht dadurch schließlich die Bindung der DNA-Polymerase. Diese fährt vom Primer aus am Ausgangsstrang entlang und fügt dabei zu jeder Base des Ausgangsstrangs ein Nukleotid mit komplementärer Base an die Kopie an. Wird dabei eines der modifizierten Nukleotide eingefügt, so kann im nächsten Schritt kein weiteres Nukleotid mehr an den kopierten Strang angefügt werden und der Synthesevorgang wird abgebrochen. Dadurch entstehen multiple, radioaktiv markierte DNA-Fragmente unterschiedlicher Länge. In jedem der Probenansätze enden diese Fragmente mit der selben Base, da nur eine der vier Basen in modifizierter Form hinzugegeben wurde.

Die vier Proben werden nun nebeneinander auf ein Gel aufgetragen. Da DNA negativ geladen ist bewegt sie sich im elektrischen Feld zur Anode. Wird also an das Gel ein elektrisches Feld angelegt, so werden die DNA-Fragmente durch das Gel bewegt. Kleinere Fragmente werden dabei schneller bewegt als größeren. Dadurch ist es möglich die DNA-Fragmente der Proben entsprechend ihrer Länge aufzutrennen. Es entstehen im Gel Anhäufungen von Fragmenten gleicher Länge, die auch als Banden bezeichnet werden. Die radioaktive Markierung der Banden kann auf Röntgenfolie sichtbar gemacht werden. Bei moderneren Verfahren ist die Markierung mit radioaktiven Isotopen durch Fluoreszenzfarbstoffe abgelöst worden. Da bekannt ist in welchen Proben welche der Basen markiert ist, kann die DNA-Sequenz direkt aus der aufsteigenden Länge der DNA-Fragmente an den Banden abgelesen werden.

1.2.2 PCR

Zunächst waren für die Sanger-Sequenzierung große Mengen an Zellmaterial notwendig, um daraus ausreichend DNA für sichtbare Banden auf dem Gel extrahieren zu können. Die Sequenzierung mit nur geringen DNA-Mengen war nicht möglich. Durch die Entwicklung des Verfahrens der Polymerasekettenreaktion (PCR, polymerase chain reaction) [18] wurden die Möglichkeiten der Sequenzierung revolutioniert. Die PCR ermöglicht es aus kleinsten DNA-Proben multiple Kopien herzustellen. Und so ist es inzwischen möglich, selbst aus der DNA einer einzigen Zelle umfangreiche Analysen durchzuführen [19].

1.2.3 Next Generation Sequencing

1.3 RAD-Sequencing

1.3.1 Verfahren

Kapitel 2

Analyse von RAD-Seq-Daten

2.1 Problemstellung

Ausgangsmaterial für RAD-Seq-Analysen sind in der Regel gepoolte Probengemische bestehend aus multiplen DNA-Fragmenten mehrerer Individuen. Diese DNA-Fragmentsequenzen stammen aus der RAD-Sequenzierung und werden auch als Reads bezeichnet. Durch Barcodemarkierungen können die Reads den zugehörigen Individuen zugeordnet werden. Neben der Kosteneffizienz durch die Verwendung gepoolter Probengemische besteht ein großer Vorteil des Verfahrens darin, die Reads ohne Kenntnis eines Referenzgenoms möglichen Loci zuordnen zu können. Auch die Loci und ihre Sequenz sind somit anfangs unbekannt und werden erst durch die Analyse ermittelt.

Das RAD-Sequencing-Verfahren bedingt, dass die DNA nur in den für die verwendeten Restriktionsenzyme spezifischen Sequenzen geschnitten wird und dass durch PCR und Sequenzierung mehrere Reads erzeugt werden, die jeweils vom gleichen Locus stammen. Das hier implementierte Tool NodeRAD stellt einen Prototypen dar, der im Gegensatz zu derzeit etablierten RADSeq-Tools, wie beispielsweise das Tool Stacks [20], mit Hilfe von nur zwei Parametern diese Zuordnung realisiert. Das Clustering der Reads und die Identifizierung und Zuordnung der Loci soll dabei allein auf Grundlage der Mutations- und der Sequenzierfehlerrate erfolgen. Die dabei ermittelte Likelihood steht zudem für anschließende Diversitätsanalysen zur Verfügung.

2.2 Formale Definition

===== draft =====

gegeben: reads einer sample

Ziel: Zuordnung der Reads zu loci -> Ausgabe: Sequenz der Loci

es soll die Menge von Loci gefunden werden, die die Beobachtungen erklären können
 Ergebnis: loci aus den beobachtungen der reads im Gegensatz zu Stacks ohne Vorwissen abschätzbar, Stacks berücksichtigt keine Ploidie und Heterozygotie

Problem wird aufgeteilt in Teilprobleme -> Zusammenhangskomponenten
 für jede Komponente: Lösung finden für eine Komposition von Loci, die am besten die beobachteten Reads erklären können -> Berücksichtigung der Basenqualität q_r => Sequenzierfehler fließen mit geringerer likelihood in die weitere Berechnung ein -> werden weniger berücksichtigt und kommen seltener vor, z.T. als noise entfernt -> loci likelihood wird mit den sequenzen mit seq-fehlern geringer sein -> seq-fehler gefiltert, da es nicht möglich ist eine bessere lh für z.B. 3 Allele zu bekommen (von denen eines aber auf einem seq-fehler beruht), wenn es nur 2 tatsächliche Allele in einer komp gibt, d.h. lh für 2 Allele ist in diesem fall höher als für 3 -> loci likelihood enthält mutationen

2.3 Lösungsansatz

Defs von Sequ, Mutationsrate, ploidie, heterozyg

2.3.1 Graph und Zusammenhangskomponenten

Graph als Gesamtproblem -> Verkleinern durch Zerlegung in Teilprobleme -> Menge der conn comp
 unklar wie viele loci jede conn comp beinhaltet, jede conn comp kann auch mehr als einen locus enthalten

2.3.2 Pair Hidden Markov Model

paarweises Alignment aus Minimap2 als Teilproblem -> fast jede Beobachtung wird berücksichtigt
 Bewertung/wslk ob ein read aus einem anderen read entstanden ist

2.3.3 Die maximale Likelihood der Allele

0. noise nicht als Kandidaten-Allele -> liste lexicogra. sortierter Allele

1. lh zwischen 1 read und 1 allel; likelihood im pair: Wslk, dasss von 1 bestimmten Allel a_i mit Fehlerrate ϵ tatsächlich die readsequenz s_r stammt

s_r Sequenz von Read r , q_r Basenqualität aller Basen von Read r , a_i Allel i aus der Menge der Kandidatenallele $A = (a_1, \dots, a_n)$ mit Anzahl n und ϵ Sequenzierfehlerrate

$$Pr(T = s_r | S = a_i, \epsilon) = pairHMM_{\epsilon, q_r}(a_i, s_r) \quad (2-xxx1)$$

2. Info über alle vafs aller Kandidatenallele für 1 read; Zusammenhangskomponenten kann auch >1 locus enthalten \rightarrow Fractions finden in der möglichen Menge von loci

Berechnung der Wslk, einen Read zu beobachten anhand der gegebenen Allele-Fraktion die Wslk der Fraktion Θ_i muss sich aus den tatsächlichen Beobachtungen ergeben \Rightarrow Urnenmodell (entspricht binominal model)

Urnenmodell: bsp. 2 farben in unterschiedl. Menge \rightarrow Wslk, wenn n Kugeln gezogen werden, dass dabei jeweils genau 50% von jeder Farbe gezogen wurden (Binomial Formel), Anzahl der roten Kugeln P (Erfolgswslk) Θ_i und > 2 Farben \Rightarrow Multinomialverteilung Θ_i + individuellen Verteilung $Pr: \Theta_i * Pr$ es gilt $\sum \Theta_i = 1 \Rightarrow$ die mögliche Zuordnung entspricht der Häufigkeit die gezogen wurde Θ_i

Wslk. s_r zu beobachten unter der gegebenen Verteilung aus den Fraktionen; Wslk read zu sehen gegeben das Allel, nur ausgehend von Sequenzierfehlern

$\Theta = \theta_1, \dots, \theta_n \in [0, 1]^n$ sind Allel-Fractionen, n ist Anzahl der Allele

$$Pr(s_r | \Theta = \theta_1, \dots, \theta_n) = \sum_{i=1}^n \theta_i \cdot Pr(T = s_r | S = a_i, \epsilon) \quad (2-xxx2)$$

3. Für alle Reads:

1 read s_r mit n kandidaten allelen $\rightarrow n$ mögliche Allele von denen s_r stammen kann \rightarrow es wird für jedes der allele wslk berechnet, dass die allele aus s_r entstanden sind

bsp. allel i hat frac 0.25, also 1/4 für i, allel i+1 mit 0.75 \Rightarrow zu 25% allel i und zu 75% allel i+1 \rightarrow über max lh wird bestimmt, wie die seq der allele zusammenpassen \rightarrow modellierung ähnlicher, unsicherheit bei der zuordnung der reads, wenn sie weder der dem eine noch dem anderen allel entstammen, z.b. bei 0.5/0.5 \Rightarrow Unsicherheit wird an schritt 3 weitergegeben

Produkt über alle reads für jede mögliche Kombination an fractions

max wählen innerhalb der conn. comp

insgesamt:

$\Rightarrow \Theta$ berechnen \rightarrow max lh aus allel-fractions

\Rightarrow alles was an Unterschieden vorkommt, kann nur ein Sequenzierfehler sein

m ist Anzahl der Reads, $L \in \{l_i \in \mathbb{N}_{\leq n}^\phi \mid i = 1, \dots, g\}$ ist Menge der Loci, $D = (s_1, \dots, s_m) \in \{A, C, G, T\}^{k^m}$ ist Menge der Readsequenzen/der Reads,

$$L(\Theta = \theta_1, \dots, \theta_n | D) = Pr(D | \Theta = \theta_1, \dots, \theta_n) = \prod_{r=1}^m Pr(s_r | \Theta = \theta_1, \dots, \theta_n) \quad (2-xxx3)$$

für jede vaf wurde also liklihood über alle reads bestimmt \rightarrow Maximum

2.3.4 Die maximale Likelihood der Loci

connected comp in loci aufsplitten

Zuordnung zu loci, also welche genomischen loci stecken dahinter, wie werden die allele den loci zugeordnet -> max parximony, prinzip der einfachsten möglichen lösung

-> ploidy als eingabeparam -> zuordnung der kandidaten allele zu loci

lh ausrechnen aus anzahl der allele und loci -> gerade die loci, die die seq. der reads am besten unter der geg. ploidy und heterozygotie erklären haben max lh max_lh_vafs nur noch 1 Vektor

Zuordnung von cand-Allelen zu den loci => die beobachteten max_lh_allele_fractions müssen sie durch die gewählten loci erklären, z.B. 2 loci => 1.locus: allel 0, 0 homozyg, 2. locus allel 0, 2 heterozygot

-> bei z.B 3 Allelen sind min 2 Loci notwendig => min # der loci, so dass alle ausgewählten allele passen -> diese menge muss gefunden werden und muss wieder max lh haben unter der annahme der loci-verteilung muss bestimmt werden ob sinnvoll, d.h allel-vafs müssen sich zu 1 summieren -> indikator-funktion: wenn 1, dann gilt $\Pr(T|S)$, d.h. Wslk, dass sich aus S die seq T gebildet hat

indicator-function $z_l \in [0, 1]$: wird 1, wenn anzahl des auftretens aller allele $A = (a_1, \dots, a_n)$ in einer möglichen locus-verteilung $l = (l_1, \dots, l_{g \cdot \phi})$ jeweils genau ihrer absoluten Häufigkeit aus der in Kap. 2.3.3 errechneten vaf mit maximaler Likelihood entspricht, also $\theta_i \cdot g \cdot \phi$ entsprechen, andernfalls gilt $z_l = 0$. Seien dabei $\Theta = \theta_1, \dots, \theta_n \in [0, 1]^n$ die vaf mit maximaler Likelihood, ϕ die Ploidie und gilt für eine mögliche Locusverteilung l außerdem $L \in \{l_j \in \mathbb{N}_{\leq n}^\phi \mid j = 1, \dots, g\}$, so lässt sich die Indikatorfunktion wie folgt definieren:

$$z_l = \prod_{i=1}^n 1_{\sum_{j=1}^g \sum_{k=1}^{\phi} 1_{l_{j,k}=i} = \theta_i \cdot g \cdot \phi} \quad (2\text{-xxx4})$$

pHMM: Wslk, dass allel 2 $a_{l_{j,2}}$ aus allel 1 $a_{l_{j,1}}$ entstanden ist -> heterozygotie η (in conf konfigurierbar) => $match = 1 - (het_{sub} + het_{del} + het_{ins})$

$$\Pr(T = a_{l_{j,2}} \mid S = a_{l_{j,1}}, \eta) = pairHMM_{\eta}(a_{l_{j,1}}, a_{l_{j,2}}) \quad (2\text{-xxx5})$$

Kombi von loci -> lh berechnen

$$\Pr(\Theta, A \mid L = \{l_j \mid j = 1, \dots, g\}) = z_l \cdot \prod_{j=1}^g \Pr(T = a_{l_{j,2}} \mid S = a_{l_{j,1}}) \quad (2\text{-xxx6})$$

=> maximum der loci Likelihoods wählen

2.3.5 Varianten und Genotyp

mit den allelen $A = (a_1, \dots, a_n)$ jedes locus $L \in \{l_j \in \mathbb{N}_{\leq n}^\phi \mid j = 1, \dots, g\}$, allelsequenz,
genotyp

Menge von Loci, die die Beobachtungen erklären

Kapitel 3

Algorithmus

Für das hier implementierte RAD-Sequencing-Tool, NodeRAD, wurde zur Workflowintegration das Workflow Management System Snakemake verwendet [21, 22]. Die einzelnen Analyseschritte werden dabei über Regeln abgebildet. Für jede Regel können neben dem zu verwendenden Script oder Shell-Kommando sowie den Pfadangaben für In- und Output auch zusätzliche Optionen festgelegt werden. Dazu gehören beispielsweise Angaben zu Parametern bzw. Argumenten für die verwendeten Tools, Pfadangaben für Log-Dateien oder die Anzahl der zu verwendenden Threads.

Als Input benötigt der Workflow eine Datei im FASTQ-Format, welche die single-end Reads der verschiedenen Individuen mit ihren Identifikationsbezeichnungen, der Basensequenz und Angaben zur Basenqualität enthält. Des Weiteren wird eine Tabelle im tsv-Format benötigt, in der die Zuordnung der Probenamen zu den Individuen und ihren Barcode-Sequenzen definiert ist. Nach dem Preprocessing, der Qualitätskontrolle der Reads und dem Sequence-Alignment erfolgt die RAD-Seq-Analyse durch NodeRAD. Hierbei werden die Wahrscheinlichkeiten der Allelsequenzen und der möglichen Loci bestimmt. Die Loci mit der höchsten Wahrscheinlichkeit werden schließlich mit den Sequenzen ihrer Allele und den möglichen Varianten entsprechend dem ermittelten Genotyp in einer Datei im Variant Call Format (VCF) ausgegeben.

3.1 Preprocessing

Im Preprocessing werden durch das Tool Cutadapt [23] die Reads jedes Individuums anhand ihrer Barcodesequenzen identifiziert und extrahiert (Demultiplexing). Hiernach werden die Barcodesequenzen entfernt (Trimming) und die Reads jedes Individuums in separaten Dateien im FASTQ Format abgelegt.

Im Anschluss an das Trimming erfolgt eine Qualitätskontrolle durch das Tool FastQC [24]. Dabei werden einige allgemeine Statistiken zu den Rohdaten der Reads generiert, wie beispielsweise zur Basenqualität, zum GC-Gehalt, dem Anteil an Duplikaten oder überre-

präsentierten Sequenzen. Durch das Tool MultiQC [25] wird aus diesen Statistiken und den Log-Dateien von Cutadapt ein html-Report mit diversen Plots zur Veranschaulichung erstellt.

3.2 Edit-Distanzen

Für die spätere Konstruktion eines Graphen basierend auf den Edit-Distanzen zwischen den Readsequenzen wird für jedes Individuum zunächst ein Sequenzalignment mit Hilfe des Tools Minimap2 [26] erstellt. Hierbei werden alle Readssequenzen paarweise verglichen und in Abhängigkeit von ihren Übereinstimmungen (Matches) und Unterschieden (Mismatches) einander zugeordnet. Das Ergebnis des Mappings wird im sam-Format [27] ausgegeben und enthält Angaben zur betrachteten Sequenz (Query), die gegen einen anderen Read (Reference) verglichen wurde. Neben den ID's der Query- und Reference-Sequenzen, werden dort unter anderem auch der CIGAR-String, Informationen zur Basenqualität der Query-Sequenz, sowie optional verschiedene Tags angegeben. Ein für die späteren Berechnungen wichtiges Maß sind die Edit-Distanzen, die durch den NM-Tag repräsentiert werden. Die Edit-Distanz gibt hierbei die minimale Anzahl von Editieroperationen an, um die Query-Sequenz in die Referenzsequenz zu transformieren. Als Editieroperationen sind hierbei ersetzen, einfügen und löschen von Basen möglich. Auf DNA-Ebene entspricht dies den Punktmutationen im Sinne von Substitutionen, Insertionen und Deletionen (vgl. Kap. 1.1.5). Der CIGAR-String ist eine kondensierte Darstellung der Unterschiede zwischen Query- und Reference-Sequenz. In ihm werden Matches und Mismatches wie Insertionen, Substitutionen und Deletion jeweils mit der Anzahl der betroffenen Basen angegeben. Sowohl der CIGAR-String als auch der NM-Tag definieren wichtige Kanteneigenschaften des späteren Graphen.

3.3 Konstruktion des Graphen

3.3.1 Knoten des Graphen

Das hier in Python implementierte Tool, NodeRAD, benötigt als Input zu jedem Individuum die getrimmten single-end Read-Daten sowie das Sequenzalignment. Zunächst wird daraus für jedes Individuum ein eigener, gerichteter Graph G mit $G = (V, E)$ erstellt. Seine Knoten, V , werden durch die einzelnen Reads repräsentiert. Entsprechend ergeben sich die Knoteneigenschaften aus den Daten der Reads, diese werden den FASTQ-Dateien nach Ausführung von Cutadapt (siehe 3.1) entnommen. Die Kanten, E , zwischen den Knoten ergeben sich aus dem Vergleich ihrer Sequenzen im Rahmen des Sequenzalignments mittels

Minimap2 (siehe 3.2).

Zusätzlich entnimmt NodeRAD der Konfigurationsdatei des Workflows einige Konstanten und Grenzwerte für die späteren Berechnungen. Dazu gehören die Mutationsraten und Heterozygotiewahrscheinlichkeiten für Substitutionen, Insertionen und Deletionen, die Ploidie des Chromosomensatzes der untersuchten Spezies und Grenzwerte. Die Konstanten werden als Grapheigenschaften im Graphen abgelegt. Als konfigurierbare Grenzwerte gibt es für NodeRAD einen Schwellenwert für die maximal zulässige Editierdistanz, bei dem zwei Knoten noch durch eine Kante verbunden werden sowie Schwellenwerte zum Filtern selten vorkommender Sequenzen ab einer bestimmten Clustergröße, die als Hintergrundrauschen nicht in der Berechnung Berücksichtigung finden sollen.

Zur Konstruktion des Graphen wird die Python-Library graph-tool [28] genutzt. Die Knoten werden aus den FASTQ-Daten der getrimmten Reads mittels SeqIO aus der Library Biopython [29] ausgelesen und im Graphen mit den Knoteneigenschaften ihrer Basensequenz, einer internen ID sowie Angaben zur Basenqualität abgelegt. Die Codierung des Qualitystrings der Reads variiert je nach verwendeter Plattform. Daher wird er durch SeqIO ausgelesen und für jede Base in ein einheitliches Maß, den Phred Quality Score Q , decodiert [30]. Zusätzlich wird aus den Phred Quality Scores die geschätzte Fehlerwahrscheinlichkeit P für jede Base nach Formel (3-1) bestimmt [31].

$$P = 10^{\frac{-Q}{10}} \quad (3-1)$$

Für jeden Knoten werden die Vektoren mit den Phred Quality Scores und den geschätzten Fehlerwahrscheinlichkeiten der Basen des Reads als Knoteneigenschaften gespeichert.

Die Laufzeit für das Hinzufügen eines Knotens beträgt nach der Dokumentation von graph-tool $O(V)$, da es sich hierbei um eine Einfügeoperation in die bereits bestehende Knotenmenge handelt und ein neuer Iterator über alle Knoten erzeugt und zurückgegeben wird [32]. Die Zuweisung der Knoteneigenschaften erfolgt in $O(1)$. Über alle Reads, also über die resultierende Anzahl der Knoten V ergibt sich daraus eine Gesamtlaufzeit von $O(V^2)$.

3.3.2 Kanten des Graphen

Die Kanten des Graphen definieren sich durch das mittels Minimap2 erzeugten Sequenzalignments (vgl. Kap. 3.2). Jedes Alignment zwischen zwei Reads entspricht im Graphen einer gerichteten Kante $e = (source, target)$, die den Vergleich der Query- zur Referenzsequenz repräsentiert. Sie verbindet somit zwei der zuvor aus der FASTQ-Daten erzeugten

Knoten. Das Auslesen des sam-Formats des Alignmentfiles erfolgt mit Hilfe der Python-Library `pysam` [33]. Dabei wird die Edit-Distanz aus dem NM-Tag zunächst genutzt, um nur Kanten in den Graphen aufzunehmen, die bereits einen optimierten Minimap2-Path darstellen. Liegen diese unterhalb des durch die Konfigurationsdatei festgelegten Grenzwertes, so wird die Kante dem Graphen hinzugefügt. Dabei werden als Kanteneigenschaften die Edit-Distanz, die CIGAR-Tupel sowie die aus der Basenqualität und Mutationsrate bestimmte Likelihood hinzugefügt. Zusätzlich kann zur Kontrolle oder für eine spätere Verwendung auch der CIGAR-String selbst als Kanteneigenschaft gespeichert werden, falls bei Minimap2 die Option zur Erzeugung des cs-Tags aktiviert wurde. Die CIGAR-Tupel werden durch `pysam` aus dem CIGAR-String geparsed, hierbei handelt es sich um eine Liste von Tupeln, die jeweils aus Integer-Wertepaaren bestehen. Der erste Wert jedes Tupels gibt die spezifische Operation des Matches oder Mismatches. So entspricht beispielsweise ein Wert von 7 oder 0 einem Match und ein Wert von 2 einer Deletion. Der zweite Werte jedes Tupels gibt die Anzahl der Basen an, die von der entsprechenden Operation betroffen sind.

Diese CIGAR-Tupel werden für die Berechnung der Likelihood zwischen zwei Knoten benötigt, dies erfolgt in der Methode `get_alignment_likelihood()` (Algorithmus 3.1) aus dem Modul `likelihood_operations.py`. Dabei wird aus den p-Werten der Basenqualität für jede Base der Query-Sequenz die Wahrscheinlichkeit errechnet, dass es sich im Falle eines Matches um die korrekte Base handelt (3-2) bzw. im Falle eines Mismatches, dass es sich um einen Sequenzierfehler (3-3) oder um eine Mutation handelt (3-4).

Die Berechnung der Likelihoods für die paarweisen Vergleiche der Reads basiert auf dem in Kap. 2.3.2 beschriebenen pair Hidden Markov Model. Hierbei repräsentiert das durch Minimap2 bestimmte Sequenzalignment bereits den wahrscheinlichsten Pfad durch die pairHMM-Matrix. Da dieser Pfad ohnehin die Wahrscheinlichkeit des pairHMM dominieren würde, wird zugunsten der Laufzeit direkt auf das Alignment von Minimap2 zurückgegriffen, um die Likelihoods zwischen den Readsequenzen zu bestimmen. Dabei werden die Sequenzierfehlerrate ϵ und Basenqualität q_{query} durch die bereits zuvor ermittelte geschätzte Fehlerrate p_{query} berücksichtigt. Die Likelihood $pairHMM_{\epsilon, q_{query}}(s_{ref} | s_{query})$, dass der Queryread aus dem Referenzread allein durch Sequenzierfehler und Mutationen entstanden ist, errechnet sich schließlich aus dem Produkt der Likelihoods L_i für jede Base b an jeder Position i innerhalb der Sequenz s des Queryreads s_{query} im Vergleich zum Referenzread s_{ref} .

$$pairHMM_{\epsilon, q_{query}}(s_{query} | s_{ref}) = \prod_{i=1}^k L_i \quad (3-2)$$

Jede Base b an Position i einer Readsequenz s der Länge k lässt sich also definieren als $b \in \{b_i \in \{A, C, G, T\}^k, b_i \in s \mid i = 1, \dots, k\}$. Seien $b_{i_{ref}}$ und $b_{i_{query}}$ die Basen der

Query- und der Referenzsequenzen an Position i einer Sequenz und p_{i_query} die geschätzte Fehlerrate von b_{i_query} , die sich aus dem Phred Quality Score Q nach (3-1) ergibt. Seien zudem m_{sub} , m_{ins} und m_{del} die über die Konfigurationsdatei festgelegten Mutationsraten für Substitutionen, Insertionen und Deletionen. Dann errechnet sich die Likelihood $L_i = Pr(b_{i_ref} \mid b_{i_query})$ an der Position i im Falle eine Matches unter Berücksichtigung der geschätzten Fehlerrate durch:

$$L_{i_match} = 1 - p_{i_query} \quad (3-3)$$

Bei einem Mismatch dagegen müssen die Wahrscheinlichkeiten von Mutationen und Sequenzierfehlern berücksichtigt werden. Im Falle einer Mutation muss in die Wahrscheinlichkeit eines Matches auch die Mutationsrate des aufgetretenen Mismatches $m_{rate} \in \{m_{sub}, m_{ins}, m_{del}\}$ einbezogen werden:

$$L_{i_mut} = m_{rate} \cdot (1 - p_{i_query}) \quad (3-4)$$

Die Wahrscheinlichkeit eines Sequenzierfehlers, also dass anstelle der sequenzierten Base tatsächlich eine der drei anderen Basen vorliegt, entspricht $1/3$ der geschätzten Fehlerrate des Phred Quality Scores:

$$L_{i_seqerr} = \frac{1}{3} \cdot p_{i_query} \quad (3-5)$$

Aus (3-4) und (3-5) errechnet sich also die Likelihood bei einem Mismatch durch:

$$L_{i_mismatch} = (1 - m_{rate}) \cdot L_{seqerr} \cdot L_{mut} \quad (3-6)$$

Aus den Likelihoods von Matches (3-3) und Mismatches (3-4) kann somit schließlich nach (3-2) die Likelihood zwischen den Reads paarweise bestimmt werden.

Für eine existierende Kante, von der die CIGAR-Tupel bekannt sind, kann die Methode `get_alignment_likelihood()` zudem die Likelihood in entgegengesetzter Richtung bestimmen. Dabei wird der Queryread als Referenzread betrachtet und umgekehrt. Dies ist über das boolsche Argument `reverse` steuerbar. Gilt `reverse = True`, so werden für die übergebenen CIGAR-Tupel Insertionen zu Deletionen und Deletionen zu Insertionen umbewandelt, anschließend wird die Likelihood nach (3-6) berechnet.

Zur zusätzlichen Veranschaulichung ist die Methode `get_alignment_likelihood()` in Algorithmus 3.1 in Pseudocode dargestellt.

Algorithmus 3.1 Berechnung der Likelihood zwischen zwei Reads

```

1: function GET_ALIGNMENT_LIKELIHOOD( $m_{sub}$ ,  $m_{ins}$ ,  $m_{del}$ ,  $CIGAR-Tuples$ ,  $p_{query}$ ,
   reverse)
2:    $likelihood \leftarrow 1.0$ ,  $index \leftarrow 0$ 
3:   if  $reverse$  then
4:     swap values of  $m_{ins}$  and  $m_{del}$ 
5:   end if
6:   for all ( $operation, length$ )  $\in CIGAR - Tuples$  do
7:     if  $operation \in match$  then
8:       while  $index < length$  do
9:          $likelihood \leftarrow likelihood \cdot (1 - p_{query}[index])$ 
10:         $index \leftarrow index + 1$ 
11:      end while
12:    end if
13:    if  $operation \in mismatch$  then
14:       $m_{rate} \leftarrow 0$ 
15:      if  $operation \in substitution$  then
16:         $m_{rate} \leftarrow m_{sub}$ 
17:      end if
18:      if  $operation \in insertion$  then
19:         $m_{rate} \leftarrow m_{ins}$ 
20:      end if
21:      if  $operation \in deletion$  then
22:         $m_{rate} \leftarrow m_{del}$ 
23:      end if
24:      while  $index < length$  do
25:         $likelihood \leftarrow likelihood \cdot (1 - m_{rate}) \cdot \frac{1}{3} \cdot p_{query}[index] + m_{rate} \cdot$ 
26:           $(1 - p_{query}[index])$ 
27:         $index \leftarrow index + 1$ 
28:      end while
29:    end if
30:  end for
31:  return  $likelihood$ 
32: end function

```

Hinsichtlich der Laufzeit benötigt das Hinzufügen einer Kante nach Angaben der graph-tool Dokumentation [32] eine Laufzeit von $O(1)$. Da aber die Query- und die Referenzreads den bereits zuvor angelegten Knoten zugeordnet werden müssen, erfordert dies eine Suche der betreffenden Knoten. Dabei durchsucht graph-tool mit seiner Funktion `find_vertex()`

allein die Knoten und prüft auf die gesuchte Read-ID aus den FASTQ-Daten. Die ein- und ausgehenden Kanten der Knoten werden nicht beachtet, so dass eine Tiefen- oder Breitensuche des Graphen nicht notwendig ist und die Suche in $O(V)$ durchgeführt werden kann [34]. Die Zuweisung der Kanteneigenschaften erfolgt jeweils in $O(1)$, da diese direkt bei der Erzeugung der Kante hinzugefügt werden und keine vorherige Suche der Kante erforderlich ist. Für die Berechnung der Likelihood wird die geschätzte Fehlerrate p_{query} jeder Base verwendet, so dass die Anzahl der Berechnungen für jede Kante der Länge der Readsequenz k entspricht. Die Laufzeit für das Hinzufügen einer Kante beträgt somit $O(k)$. Für alle Kanten ergibt sich daraus eine Gesamtlaufzeit von $O(E \cdot (k + V))$. Bei realen Datensätzen gilt in der Regel $k \ll V$ und die Länge der Reads variiert nur in einem engen Bereich, so dass k als vernachlässigbar klein und als nahezu konstant betrachtet werden kann. Dann ergibt sich aus Formel (3-7) eine Laufzeit von $O(E \cdot V)$.

$$O(E \cdot (k + V)) = O(E \cdot V) \quad (3-7)$$

Sind bei kleinen Datensätzen nur wenige Reads vorhanden, so dass $k \leq V$, dann ergibt sich unter zusätzlicher Berücksichtigung von k im Worst Case mit $k = V$ nach Formel (3-8) ebenfalls eine Laufzeit von $O(E \cdot V)$.

$$O(E \cdot (k + V)) = O(E \cdot (V + V)) = O(E \cdot 2 \cdot V) = O(E \cdot V) \quad (3-8)$$

Unter der Annahme, dass in seltenen Fällen die Readlänge, die meist nur wenige hundert Basenpaare zählt, tatsächlich die Anzahl der Reads übersteigt und somit $k > V$ gilt, dann dominiert k die Laufzeit. In diesem Fall wäre im Worst Case k eine obere Schranke für V , so dass gilt:

$$O(E \cdot (k + V)) = O(E \cdot (k + k)) = O(E \cdot 2 \cdot k) = O(E \cdot k) \quad (3-9)$$

Da die Länge der Reads durch die gewählten Restriktionsenzyme sowie durch das Sequenzierverfahren selbst beschränkt ist, müsste ein solcher Datensatz relativ klein sein, so dass die damit verbundene Laufzeiterhöhung nur geringfügige Auswirkungen hätte.

Zusammenfassend soll daher für die folgenden Berechnungen vereinfachend der Mittelwert der Readlänge als konstant betrachtet werden, also $\bar{k} = const$, so dass für die Berechnung der Likelihoods zwischen den Reads eine Laufzeit von $O(\bar{k}) = O(1)$ veranschlagt wird. Für das Hinzufügen der Kanten des Graphen wird somit die Gesamtlaufzeit auf $O(E \cdot V)$ geschätzt.

Nach Abschluss der Graphkonstruktion werden für jedes Individuum noch einige Statistiken in die Log-Dateien geschrieben. Hier werden neben der Anzahl der Knoten und Kanten des Graphen auch die Anzahl der Substitutionen bzw. SNPs, Insertionen und Deletionen festgehalten, die beim Auslesen der CIGAR-Tupel registriert wurden. Zudem findet

sich hier auch die maximal vorkommenden Edit-Distanz über alle Knoten, sofern sich diese unterhalb des festgelegten Schwellenwertes liegt. Ansonsten entspricht sie dem in der Konfigurationsdatei angegebenen Schwellenwert.

Als optionaler Output können über die Konfigurationsdatei und die Snakemake-Regel `rule noderad` auch die detaillierten Graphinformationen sowie eine Visualisierung des Graphen ausgegeben werden. Die Graphinformationen wie Knoten, Kanten und ihre Eigenschaften können dabei im GraphML-, DOT-, GML- oder im binären gt-Format gespeichert werden. Die graphische Darstellung wird als pdf-Datei ausgegeben, dabei entspricht die Kantenfärbung der berechneten Likelihood zwischen den Reads.

3.3.3 Bestimmung der Zusammenhangskomponenten

Die Bestimmung und Indexierung der Zusammenhangskomponenten erfolgt durch graph-tool selbst und kann in $O(V + E)$ durchgeführt werden [32]. Die Indexnummer jeder Zusammenhangskomponente wird den in ihr enthaltenen Knoten als Knoteneigenschaft hinzugefügt. Zusammenhangskomponenten mit mehr als einem Knoten werden als neuer eigenständiger Graph initialisiert und in einer Liste abgelegt. Hierfür wird aus dem Graphen für jede Komponente eine gefilterte Sicht erzeugt, die als neues Graph-Object gespeichert wird. Der Filtrervorgang jeder Zusammenhangskomponente C muss für alle Knoten des Graphen durchgeführt werden, daher beträgt die Laufzeit hierfür $O(C \cdot V)$. Da alle weiteren Schritte des Algorithmus jeweils auf den einzelnen Komponenten durchgeführt werden, kann durch die Verwendung einer Liste von Graphen im Folgenden eine einfache Iteration über die Komponenten in $O(C)$ ausgeführt werden, ohne dass der Filtrervorgang über alle Knoten jeder Komponente wiederholt werden muss. Zudem ermöglicht diese Datenstruktur eine effizientere Traversierung und Suche innerhalb der Zusammenhangskomponente, ohne dass für jede Komponente der gesamte Graph betrachtet werden muss. Der ursprüngliche Graph wird anschließend entfernt, um Arbeitsspeicher freizugeben. Auch dies erfolgt in konstanter Zeit. Die Laufzeit für die Extraktion der Zusammenhangskomponenten wird also bestimmt durch Identifikation, Indexierung und Filterung der Komponenten mit $O(C \cdot V) + O(V + E) = O(V \cdot (C + 1) + E)$. Bei realen Daten gibt es in der Regel deutlich mehr Knoten als Cluster bzw. Zusammenhangskomponenten, so dass gilt $C < V$. Würde im Worst Case aber jede Zusammenhangskomponente aus nur einem Knoten bestehen, also $C = V$, so kann die maximale Laufzeit auf $O(V \cdot (C + 1) + E) = O(V \cdot (V + 1) + E) = O(V^2 + E)$ geschätzt werden kann.

In der Log-Datei wird die Anzahl der Knoten aller Zusammenhangskomponenten als Histogramm festgehalten. Ebenso wird dort für alle Komponenten mit mehr als einem Ele-

ment die Anzahl ihrer Knoten, Kanten und Eigenschaften aufgelistet.

Über die Konfigurationsdatei und die Snakemake-Regel `rule noderad` können optional auch für die Zusammenhangskomponenten jeweils Visualisierungen und detaillierte Graphinformationen in den oben genannten Formaten (Kap. 3.3.2) ausgegeben werden. Zudem kann optional auch der gesamte Graph mit den Komponentenindizes als Knoteneigenschaften gespeichert werden. In der visuellen Darstellung werden seine Knoten entsprechend der zugehörigen Zusammenhangskomponente eingefärbt, seine Kantenfärbung richtet sich weiterhin nach der aus (3-3) resultierenden Likelihood.

3.3.4 Laufzeitanalyse zur Konstruktion des Graphen

Wie an entsprechender Stelle bereits beschrieben, ist für die Erzeugung der Knoten eine Laufzeit von $O(V^2)$ (Kap. 3.3.1) erforderlich, das Hinzufügen der Kanten kann in $O(E \cdot V)$ erfolgen. Somit wird für den vollständigen Aufbau des Graphen nach (3-10) eine Laufzeit von $O(V \cdot (V + E))$ benötigt.

$$O(V^2) + O(E \cdot V) = O(V \cdot (V + E)) \quad (3-10)$$

In Zusammenschau mit der für die Extraktion der Zusammenhangskomponenten erforderliche Laufzeit von $O(V^2 + E)$ (Kap. 3.3.3) ergibt sich daraus nach (3-11) eine Laufzeit von $O(V \cdot (V + E))$.

$$\begin{aligned} &O(V \cdot (V + E)) + O(V^2 + E) \\ &= O(V^2 + E \cdot V + V^2 + E) \\ &= O(2 \cdot V^2 + E \cdot (V + 1)) \\ &= O(V^2 + E \cdot V) \\ &= O(V \cdot (V + E)) \end{aligned} \quad (3-11)$$

3.4 Bestimmung der maximalen Likelihood der Allele

3.4.1 Bestimmung der Allele und ihrer möglichen Häufigkeitsverteilung

Die einzelnen Zusammenhangskomponenten repräsentieren einen oder mehrere Loci. Alle weiteren Berechnungen aus diesem und den folgenden Kapiteln 3.5 und 3.7 werden für jede Komponente einzeln durchgeführt, dabei sollen die wahrscheinlichsten Allelsequenzen und die dazugehörigen Loci zu identifiziert werden.

Zunächst werden mit der Funktion `get_candidate_alleles()` aus dem Modul `likelihood_operations.py` die Allelsequenzen ermittelt, die in der Zusammenhangskomponenten vorkommen. Für deterministischere Ergebnisse werden diese lexikographisch sor-

tiert in Form einer Liste durch die Funktion zurückgegeben. Übersteigt die Größe des Clusters, d.h. die Knotenanzahl einer Zusammenhangskomponenten einen in der Konfigurationsdatei als `threshold-cluster-size` festgelegten Wert, so wird von den im Cluster vorkommenden Sequenzen zunächst die absolute Häufigkeit bestimmt. Es werden dann nur diejenigen Sequenzen lexikographisch sortiert zurückgegeben, die einen weiteren, ebenfalls in der Konfigurationsdatei festgelegten Schwellenwert, `threshold-seq-noise`, überschreiten. Dies dient dazu, bei großen Clustern Komplexität und Rechenaufwand zu reduzieren, da davon auszugehen ist, dass innerhalb eines großen Clusters echte Varianten einer Sequenz mehrfach vorkommen, wohingegen Artefakte und Sequenzierfehler eher vereinzelt auftreten. Dieses sog. Rauschen kann durch Anpassung der Schwellwerte in der Konfigurationsdatei herausgefiltert werden. Da bei kleineren Clustern auch einmalig registrierte Varianten einer Sequenz von Bedeutung sein können, soll der Filtervorgang erst ab einer festlegbaren Clustergröße durchgeführt werden. Für das Erstellen der Liste muss jeder Knoten in jeder Komponente betrachtet werden, so dass der Vorgang für alle Knoten in $O(V)$ durchführbar ist.

Aus der so erzeugten Liste lexikographisch sortierter Kandidatenallele $A_{observed}$ der Länge $n_{observed}$ sollen nun die möglichen Häufigkeitsverteilungen in Abhängigkeit von der Ploidie bestimmt werden. Hierzu muss zunächst die aufgrund der Ploidie tatsächlich zu erwartende Anzahl von Allelen $n_{alleles}$ bestimmt werden. Die geschieht durch die Funktion `get_max_parsimony_n_alleles()` in `likelihood_operations.py`. Dadurch werden unnötige bzw. nicht mögliche Allelkombinationen eingespart und in der weiteren Berechnung nicht berücksichtigt. Ist die Ploidie ϕ höher als die Anzahl der beobachteten Allele, so muss es mindestens genauso viele und bei Homozygotie auch mehrfach vorkommende Allele geben, damit die Ploidie erfüllt werden kann. Es muss also gelten $n_{alleles} = \phi$. Wurden dagegen mehr Allele beobachtet als aufgrund der Ploidie möglich sind und die Ploidie ist Teiler von $n_{observed}$, so können alle beobachteten Allele auch tatsächlich vorkommen, da die Zusammenhangskomponente auch mehrere Loci enthalten kann. Es gilt dann also $n_{alleles} = n_{observed}$. Ist dagegen die Anzahl der beobachteten Allele höher als die Ploidie, aber nicht ganzzahlig durch die Ploidie teilbar, so muss die Anzahl der Allele entsprechend erhöht werden. Das heißt, es müssen tatsächlich so viele Allele vorkommen, dass eine korrekte Ploidie erreicht wird, dass also die Ploidie eine Teiler von $n_{alleles}$ wird. Die Anzahl der Allele muss also um die Ploidie abzüglich des Restes aus der Restdivision erhöht werden: $n_{alleles} = n_{observed} + \phi - (n_{observed} \bmod \phi)$. Diese Anpassung der Anzahl der Allele kann für jede Komponente C in $O(1)$ erfolgen, so dass die Laufzeit für alle Komponenten $O(C)$ beträgt. Im Worst Case, bei dem jede Komponente nur einen Knoten enthält und somit $C = V$ gilt, würde die Laufzeit maximal $O(V)$ betragen.

Mit Hilfe der tatsächlich zu erwarteten Anzahl von Allelen $n_{alleles}$ können nun alle Kombinationen möglicher Häufigkeitsverteilungen der Allele bestimmt werden (Funktion `get_candidate_vafs()` im Modul `likelihood_operations.py`). Hierfür werden zunächst alle möglichen Allelkombinationen ermittelt. Dies erfolgt nach dem Urnenmodell unter Auswahl von $n_{alleles}$ Elementen mit Zurücklegen aus insgesamt $n_{observed}$ verschiedenen Elementen und ohne Berücksichtigung der Reihenfolge. Dadurch berechnet sich die Anzahl möglicher Kombination nach (3-12) aus dem Binomialkoeffizienten der k -ten Ordnung aus n Elementen mit Zurücklegen [35, 36].

$$\binom{n+k-1}{k} = \frac{(n+k-1)!}{(n-1)! \cdot k!} = \frac{(n_{alleles} + n_{observed} - 1)!}{(n_{alleles} - 1)! \cdot n_{observed}!} \quad (3-12)$$

Diese Allelkombinationen werden mit Hilfe der Funktion `combinations_with_replacement` aus der Python-Library `itertools` erzeugt [37].

Beispiele möglicher Allelkombinationen:

$ploidy = 2, n_{observed} = 2, n_{alleles} = 2$:

[(0, 0), (0, 1), (1, 1)]

$ploidy = 2, n_{observed} = 3, n_{alleles} = 4$:

[(0, 0, 0, 0), (0, 0, 0, 1), (0, 0, 0, 2), (0, 0, 1, 1), (0, 0, 1, 2), (0, 0, 2, 2), (0, 1, 1, 1), (0, 1, 1, 2), (0, 1, 2, 2), (0, 2, 2, 2), (1, 1, 1, 1), (1, 1, 1, 2), (1, 1, 2, 2), (1, 2, 2, 2), (2, 2, 2, 2)]

Für jedes Allel i können im Anschluss aus seinen absoluten Häufigkeiten H_{a_i} innerhalb jeder Allelkombination die relativen Häufigkeiten h_{a_i} nach (3-13) bestimmt werden.

$$h_{a_i} = \frac{H_{a_i}}{n_{alleles}} \quad (3-13)$$

Beispiele möglicher Häufigkeitsverteilungen der Allele:

$ploidy = 2, n_{observed} = 2, n_{alleles} = 2$:

[1.0, 0.0], [0.5, 0.5], [0.0, 1.0]

$ploidy = 2, n_{observed} = 3, n_{alleles} = 4$:

[1.0, 0.0, 0.0], [0.75, 0.25, 0.0], [0.75, 0.0, 0.25], [0.5, 0.5, 0.0], [0.5, 0.25, 0.25], [0.5, 0.0, 0.5], [0.25, 0.75, 0.0], [0.25, 0.5, 0.25], [0.25, 0.25, 0.5], [0.25, 0.0, 0.75], [0.0, 1.0, 0.0], [0.0, 0.75, 0.25], [0.0, 0.5, 0.5], [0.0, 0.25, 0.75], [0.0, 0.0, 1.0]

Abschätzung der oberen Laufzeitschranke bei Kombinationen mit Wiederholung und Laufzeit der Funktion `get_candidate_vafs()`

Kombinationen mit Wiederholung werden sowohl für die Bestimmung der Allelkombinationen aus diesem Kapitel als auch später in Kap. 3.5.1 für die Ermittlung möglicher Kombinationen von Loci genutzt. Hinsichtlich der Laufzeit und des Speicherplatzbedarfs kann diese Berechnung bei sehr großen Clustern zum dominierenden Faktor werden. Daher soll die Laufzeit an dieser Stelle genauer abgeschätzt werden. Seien dabei für eine bessere Übersichtlichkeit $n_{alleles}$ als n und $n_{observed}$ als k bezeichnet und es gilt $n, k \in \mathbb{N}$. Nach Formel (3-12) werden also $\binom{n+k-1}{k}$ Kombinationen jeweils in $O(1)$ erzeugt, d.h. in $O(\binom{n+k-1}{k})$ für jede Zusammenhangskomponente.

Die Formel (3-12) ergibt sich aus der allgemeinen Formel des Binomialkoeffizienten (3-13), durch den die Anzahl aller Kombinationen ohne Zurücklegen berechnet wird [35].

$$\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!} \quad (3-13)$$

Mit anderen Worten, die Formel (3-12) kann auf zwei Arten interpretiert werden: es werden aus n Elementen k Elemente mit Zurücklegen gezogen oder es werden aus $n+k-1$ Elementen k Elemente ohne Zurücklegen gezogen. Sei also $m = n+k-1$, dann gilt:

$$\binom{m}{k} = \frac{m!}{(m-k)! \cdot k!} \leq m! \quad (3-14)$$

Daraus ergibt sich eine Laufzeit zunächst eine obere Schranke der Laufzeit von $O(m!) = O((n+k-1)!)$. Hierfür soll nun eine kleinere obere Schranke gefunden werden. Um die Fakultät näherungsweise zu berechnen kann die Stirlingsche Formel [36] verwendet werden:

$$n! \approx \left(\frac{n}{e}\right)^n \cdot \sqrt{2 \cdot \pi \cdot n} \quad (3-15)$$

Somit gilt auch (3-16) und ist eine untere Schranke der Fakultät [38].

$$n! \geq \left(\frac{n}{e}\right)^n \quad (3-16)$$

Durch (3-12) und (3-16) lässt sich durch einige Umformungen die obere Schranke der Laufzeit von ursprünglich $O(n!)$ auf $O\left(\left(\frac{e \cdot (n+k-1)}{k}\right)^k\right)$ eingrenzen:

$$\begin{aligned}
\binom{n+k-1}{k} &\stackrel{(3-12)}{=} \frac{(n+k-1)!}{(n-1)! \cdot k!} = \frac{\prod_{i=1}^{n+k-1} i}{\prod_{i=1}^{n-1} i \cdot \prod_{i=1}^k i} \\
&= \frac{\prod_{i=n}^{n+k-1} i \cdot \prod_{i=1}^{n-1} i}{\prod_{i=1}^{n-1} i \cdot \prod_{i=1}^k i} = \frac{\prod_{i=n}^{n+k-1} i}{\prod_{i=1}^k i} \\
&= \frac{(n+k-1) \cdot (n+k-2) \cdot \dots \cdot n}{k!} \\
&\leq \frac{(n+k-1)^k}{k!} \\
&\stackrel{(3-16)}{\leq} \frac{(n+k-1)^k}{\left(\frac{k}{e}\right)^k} \\
&= \left(\frac{e \cdot (n+k-1)}{k}\right)^k
\end{aligned} \tag{3-17}$$

Der Binomialkoeffizient verfügt über bestimmte Eigenschaften, insbesondere gelten der Symmetrie- (3-18) und der Additionssatz (3-19), die sich auch in der Struktur des Pascalschen Dreiecks widerspiegeln [36].

$$\binom{n}{k} = \binom{n}{n-k} \tag{3-18}$$

$$\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1} \tag{3-19}$$

Aufgrund der Symmetrie sind die Binomialkoeffizienten jeder Zeile des Dreiecks $\binom{n}{k_i}$ mit $k_i \in 1, \dots, n$ und $i \in \mathbb{N}$ symmetrisch im Bezug auf die Zeilenmitte. Und da nach (3-19) jeder Koeffizient aus der Summe der beiden darüber liegenden Koeffizienten entsteht, sind die Werte der Koeffizienten zu den Zeilenrändern hin abnehmend und weisen in der Zeilenmitte ihr Maximum auf. Ist n gerade, so befindet sich die Zeilenmitte bei $k = \frac{n}{2}$. Ist n ungerade so befinden sich die Maxima bei $k = \lceil \frac{n}{2} \rceil$ und $k = \lfloor \frac{n}{2} \rfloor$ und besitzen aufgrund der Symmetrie den gleichen Wert. Vereinfacht gilt also für ein gegebenes n , dass der Binomialkoeffizient $\binom{n}{k_i}$ bei $k = \frac{n}{2}$ maximal ist.

Daher soll $k = \frac{n}{2}$ als Worst Case angenommen werden, dann ergibt sich aus (3-17) eine Laufzeit von $O(3e^{\frac{n}{2}}) \in O(e^n)$.

$$\begin{aligned}
 \left(\frac{e \cdot (n + k - 1)}{k} \right)^k &= \left(\frac{e \cdot (n + \frac{n}{2} - 1)}{\frac{n}{2}} \right)^{\frac{n}{2}} \\
 &= \left(\frac{\frac{3en - 2e}{2}}{\frac{n}{2}} \right)^{\frac{n}{2}} \\
 &= \left(\frac{3en - 2e}{n} \right)^{\frac{n}{2}} \\
 &\leq \left(\frac{3en}{n} \right)^{\frac{n}{2}} \\
 &= 3e^{\frac{n}{2}}
 \end{aligned} \tag{3-20}$$

Hinsichtlich der Allelkombinationen entspricht dies einer Zeit von $O(e^{n_{alleles}})$ für jede Zusammenhangskomponente. Für die Laufzeit über alle Komponenten C_i gilt dann $O\left(\sum_{i=1}^{|C|} e^{n_{alleles}}\right)$, es dominiert also die Komponente, welche die meisten Knoten enthält.

Für sehr große Cluster ist daher die Laufzeit problematisch. Ebenso kommt es zu einem hohen Bedarf an Arbeitsspeicher, aufgrund der großen Anzahl von Kombinationen die dabei erzeugt werden. Um dies bedingt kompensieren zu können, wurden die oben beschriebenen Schwellenwerte eingeführt, die ab einer festgelegten Clustergröße **threshold-cluster-size** nur Allelkandidaten ab einer bestimmten Häufigkeit ihre Sequenz in den Reads **threshold-seq-noise** berücksichtigen.

Die Berechnung der Häufigkeitsverteilungen erfordert weitere Iterationen über die ermittelten Allelkombinationen, dabei werden sämtliche Allele für jede Kombination betrachtet. Dadurch erhöht sich die Laufzeit auf $O(e^{n_{alleles}}) \cdot O(n_{alleles}) \cdot O(n_{observed})$. Den höchsten Wert nimmt $n_{alleles}$ in dem Fall an, dass die Restdivision $n_{observed} \bmod \phi = 1$, denn dann gilt $n_{alleles} = n_{observed} + \phi - 1$. Da ϕ für alle Zusammenhangskomponenten den gleichen Wert besitzt, gilt $\phi = const$. Daraus ergibt sich nach (3-21) eine Laufzeit von $O(e^{n_{observed}} \cdot n_{observed}^2)$ für die Funktion **get_candidate_vafs()**.

$$\begin{aligned}
 O(e^{n_{alleles}} \cdot n_{alleles} \cdot n_{observed}) &= O(e^{n_{observed} + \phi - 1} \cdot (n_{observed} + \phi - 1) \cdot n_{observed}) \\
 &= O(e^{n_{observed} + \phi} \cdot n_{observed}^2) \cdot \phi \\
 &= O(e^{n_{observed}} \cdot n_{observed}^2)
 \end{aligned} \tag{3-21}$$

3.4.2 Berechnung der Likelihoods der Allele anhand der möglichen Häufigkeitsverteilungen

Nun soll nach (2-xxx1) die Wahrscheinlichkeit ermittelt werden, einen Read anhand der gegebenen Allelfraction zu beobachten (vgl. Kap. 2.3.3). Dies geschieht in der Funktion

`get_allele_likelihood_read()` aus dem Modul `likelihood_operations.py`). Für jeden Read wird dabei zunächst die Wahrscheinlichkeit bestimmt, dass der Read durch Sequenzierfehler aus einem der Kandidatenallele entstanden ist. Hierzu wird für jeden Read r_i jeweils eine ausgehende Kante zu jedem Kandidatenallel a_j gesucht und die nach (3-2) berechnete Likelihood aus den Eigenschaften dieser Kante zurückgegeben.

Geordnet nach dem Laufzeitaufwand werden für das Auffinden einer solchen Kante verschiedene Möglichkeiten geprüft:

1. Der Read besitzt eine ausgehende Kante zu einem Knoten, der mit der Sequenz des Kandidatenallels übereinstimmt. Hierfür werden alle ausgehenden Nachbarn des Knotens hinsichtlich ihrer in den Knoteneigenschaften abgelegten Sequenz geprüft. Wurde eine solche Kante gefunden, so wird die Likelihood der Kante direkt zurückgegeben. Andernfalls erfolgt die Suche wie unter beschrieben.
2. Existiert keine Kante vom Knoten des Reads zu einem Knoten mit der Kandidatenallelsequenz, so wird nach einer entgegengesetzt gerichteten Kante gesucht. Dazu werden alle eingehenden Nachbarn auf die Sequenz des Kandidatenalles überprüft. Falls es eine solche Kante gibt, dann kann die gesuchte Likelihood der Gegenrichtung, über die Funktion `get_alignment_likelihood()` mit dem Argument `reverse = True` bestimmt werden.
3. Falls auch in Gegenrichtung keine passende Kante gefunden wurde, so wird in der gesamten Zusammenhangskomponente nach einer Kante gesucht, die zwei Knoten verbindet, deren Sequenzen denen von r_i und a_j entsprechen. Existiert eine solche Kante, so werden ihre CIGAR-Tupel verwendet, um für r_i die Likelihood zu bestimmen. Da die CIGAR-Tupel allein über die Sequenz definiert werden, gelten die so ermittelten CIGAR-Tupel aufgrund identischer Sequenzen der begrenzenden Knoten auch für eine nicht existente Kante zwischen r_i und a_j . Die CIGAR-Tupel können also verwendet werden, um zusammen mit den nach (3-1) berechneten p-Werten des Phred Quality Scores von r_i die Likelihood mittels `get_alignment_likelihood()` zu bestimmen.
4. Existiert auch im gesamten Graphen keine vergleichbare Kante, die r_i und a_i mit einander verbindet, so wird der Wert Null zurückgegeben.

Zur Veranschaulichung ist der beschriebene Algorithmus zudem in Pseudocode dargestellt 3.2. Dabei sind $C_k \in \{C_1, \dots, C_p\}$ der Subgraph einer Zusammenhangskomponente, $r_i \in C_k$ der Knoten des Reads und $a_j \in A$ ein Kandidatenallel mit der Sequenz s_{a_j} , wobei $k, i, j \in \mathbb{N}$. Die Verwendung fester Knoten- oder Kanteneigenschaften ist durch eckige Klammern gekennzeichnet, so ruft beispielsweise $r_i[p_values]$ die p-Werte des Phred Qua-

lity Scores von r_i auf.

Algorithmus 3.2 Bestimmung der Likelihood zwischen einem Read und einem Kandidatenallel

```

1: function GET_ALLELE_LIKELIHOOD_READ( $C_k, r_i, s_{a_j}$ )
2:    $out\_neighbors \leftarrow get\_out\_neighbors(r_i)$ 
3:   for  $out\_neighbor \in out\_neighbors$  do
4:     if  $s_{a_j} = out\_neighbor[sequence]$  then
5:       return  $edge(r_i, out\_neighbor)[likelihood]$ 
6:     end if
7:   end for
8:    $m\_rates \leftarrow C_k[m_{sub}, m_{ins}, m_{del}]$ 
9:    $qual \leftarrow r_i[p\_values]$ 
10:   $in\_neighbors \leftarrow get\_in\_neighbors(r_i)$ 
11:  for  $in\_neighbor \in in\_neighbors$  do
12:    if  $s_{a_j} = in\_neighbor[sequence]$  then
13:       $cig \leftarrow edge(in\_neighbor, r_i)[CIGAR - Tuples]$ 
14:       $rev \leftarrow True$ 
15:      return  $get\_alignment\_likelihood(m\_rates, cig, qual, rev)$  ▷Alg. 3.1
16:    end if
17:  end for
18:   $cigar \leftarrow get\_cigar\_tuples(C_k, r_i[sequence], s_{a_j})$  ▷Alg. 3.3
19:  if  $cigar$  exists then
20:     $cig \leftarrow cigar[0]$ 
21:     $rev \leftarrow cigar[1]$ 
22:    return  $get\_alignment\_likelihood(m\_rates, cig, qual, rev)$  ▷Alg. 3.1
23:  end if
24:  return 0
25: end function

```

Im folgenden soll nun die Laufzeit der verschiedenen Suchvorgänge in `get_allele_likelihood_read()` genauer betrachtet werden. Dabei ist zu berücksichtigen, dass im Allgemeinen der Aufruf einer Kante bei bekannten Source- und Target-Knoten von ihren Knotengraden abhängt. Im Bezug auf die Knotengrade des Readknotens $d(r_i)$ und des Kandidatenallels $d(a_j)$ ist eine solche Suche durch graph-tool in $O(\min(d(r_i), d(a_j)))$ durchführbar [32]. Im Worst Case sind alle Knoten einer Zusammenhangskomponente mit den beiden Knoten r_i und a_j verbunden, so dass diese jeweils einen Knotengrad besitzen, welcher der Anzahl der Knoten der Komponente V_c mit Ausnahme des Knotens selbst ent-

spricht. In diesem Fall beträgt die Laufzeit für den Aufruf einer Kante im Graphen $O(V_c)$ (3-22).

$$O(\min(d(r_i), d(a_j))) = O(\min(d(V_c + 1), d(V_c - 1))) = O(V_c) \quad (3-22)$$

Im Hinblick auf `get_allele_likelihood_read()` müssen im Worst Case alle Schritte (1) bis (4) ausgeführt werden, so dass sich ihre Laufzeiten summieren. In der unter (1) beschriebenen Suche, werden alle ausgehenden Nachbarn des Reads betrachtet, ihre Anzahl entspricht also dem ausgehenden Knotengrad des Readknotens $d(r_{i_{out}})$. War die Suche erfolgreich, so muss die Likelihood der betreffenden Kante aus den Kanteneigenschaften ausgegeben werden. Hierfür ist ein Kantenaufruf wie oben beschrieben in $O(V_c)$ notwendig. Im ungünstigsten Fall hat jeder Knoten nur ausgehende Kanten und zwar zu allen anderen Knoten der Zusammenhangskomponente, dann beträgt die Laufzeit nach (3-23) für alle Reads der Komponente $O(V_c^2)$.

$$O(V_c) \cdot (O(d(r_{i_{out}})) + O(V_c)) = O(V_c \cdot ((V_c - 1) + V_c)) = O(V_c^2) \quad (3-23)$$

Die Suche bei (2) erfolgt analog über alle eingehenden Nachbarn in $O(V_c^2)$, allerdings mit zusätzlicher Likelihoodberechnung der rückläufigen Kante in $O(k)$. Daraus ergibt sich insgesamt eine Laufzeit von $O(V_c^2 + k) = O(V_c^2)$, da $\bar{k} = \text{const}$ (siehe Kap. 3.3.4).

Die Laufzeit für das Auffinden eines geeigneten CIGAR-Tupels (3) kann in $O(V_c^3)$ erfolgen (siehe Formel (3-25)). Auch hier führt die Likelihoodberechnung einer einzelnen Kante zu keiner relevanten Änderung der Laufzeit.

Die Rückgabe eines konstanten Wertes bei (4) benötigt lediglich eine Laufzeit von $O(1)$.

Aus den insgesamt vier Schritten, die im ungünstigsten Fall alle ausgeführt werden, ergibt sich für `get_allele_likelihood_read()` eine Gesamtlaufzeit von $O(V_c^3)$.

$$\begin{aligned} &O(V_c^2) + O(V_c^2) + O(V_c^3) + O(1) \\ &= O(2 \cdot V_c^2 + V_c^3) \\ &= V_c^3 \end{aligned} \quad (3-24)$$

Wie bereits erwähnt, kann die Bestimmung der Wahrscheinlichkeit, dass ein Read durch Sequenzierfehler aus einem bestimmten Allel entstanden ist, erfordern, die CIGAR-Tupel des Sequenzvergleichs zu identifizieren. Auch im Folgenden (Kap. 3.5) kann die Zuordnung der Kandidatenallele zu bestimmten Loci im Hinblick auf die Heterozygotiewahrscheinlichkeiten, die Ermittlung der CIGAR-Tupel zwischen den Kandidatenallelen bedingen. Dies wird in NodeRAD durch die Methode `get_cigar_tuples()` bewerkstelligt, die sich ebenfalls im Modul `likelihood_operations.py` befindet. Sie benötigt als Eingabeparameter einen Graphen oder Subgraphen sowie zwei Sequenzen: die Query-Sequenz s_{source} und die

Referenz-Sequenz s_{target} . Der Graph wird nach einer Kante durchsucht, die Knoten miteinander verbindet, welche die beiden gegebenen Sequenzen besitzen. Dabei werden nicht nur Kanten berücksichtigt, die von der Query- zur Referenzsequenz verlaufen, sondern auch entgegen gerichtete Kanten, die von der Referenz- zur Querysequenz verlaufen. Zur Markierung der Kantenrichtung gibt die Funktion neben dem CIGAR-Tupel auch einen booleschen Wert zurück. Dieser wird in den Funktionen `get_alignment_likelihood()` (Kap. 3.3.2) und `get_heterozygosity()` (Kap. 3.5.2) für die Likelihoodberechnung benötigt. Im Falle einer entgegen gerichteten Kante, also bei `reverse=True`, werden Deletionen als Insertionen gewertet und umgekehrt. Bei erfolgreicher Suche gibt die Funktion neben dem booleschen Wert auch die gefundenen CIGAR-Tupel zurück, ansonsten wird der Wert *None* zurückgegeben.

Um eine passende Kante im Graphen zu finden, erstellt `get_cigar_tuples()` zunächst eine Liste aller Knoten $R_{source} = (v_1, \dots, v_i)$, welche die Query-Sequenz s_{source} besitzen sowie eine Liste aller Knoten $R_{target} = (w_1, \dots, w_j)$, die die Referenz-Sequenz s_{target} tragen. Hierfür werden für jede Sequenz alle Knoten des Graphen mit `find_vertex()` aus graph-tool durchsucht. Nun wird nach einer Kante gesucht, die einen der Knoten aus R_{source} mit einem der Knoten aus R_{target} verbindet. Es erfolgt also ein Kantenaufruf für jeden Knoten aus R_{source} in Kombination mit jedem Knoten aus R_{target} . Der Kantenaufruf erfolgt sowohl für die angegebene Kantenrichtung und falls erfolglos auch für die entgegengesetzte Richtung. Existiert eine solche Kante, dann werden ihre CIGAR-Tupel sowie der boolesche Wert entsprechend der Kantenrichtung zurückgegeben.

Zur Veranschaulichung ist der Algorithmus in 3.3 zudem als Pseudocode dargestellt. Dabei sind $C_k \in \{C_1, \dots, C_p\}$ der Subgraph einer Zusammenhangskomponente, s_{query} die Query-Sequenz und s_{ref} die Referenz-Sequenz. Die Verwendung fester Knoten- oder Kanteneigenschaften ist durch eckige Klammern gekennzeichnet, so ruft beispielsweise $v_i[sequence]$ die Sequenz des Knotens v_i aus den Knoteneigenschaften ab.

Algorithmus 3.3 CIGAR-Tupel bestimmen

```

1: function GET_CIGAR_TUPLES( $C_k, s_{query}, s_{ref}$ )
2:    $R_{source} \leftarrow \{v_i \in C_k \wedge i, k \in \mathbb{N} \mid v_i[sequence] = s_{query}\}$ 
3:    $R_{target} \leftarrow \{w_j \in C_k \wedge j, k \in \mathbb{N} \mid w_j[sequence] = s_{ref}\}$ 
4:   for  $v_i \in R_{source}$  do
5:     for  $w_j \in R_{target}$  do
6:       if  $edge(v_i, w_j)$  exists then
7:         return (  $edge(v_i, w_j)[CIGAR - Tuples], False$  )
8:       end if
9:       if  $edge(w_j, v_i)$  exists then
10:        return (  $edge(w_j, v_i)[CIGAR - Tuples], True$  )
11:       end if
12:     end for
13:   end for
14:   return None
15: end function

```

Die hierfür erforderliche Laufzeit setzt sich zusammen aus den Suchvorgängen für die beiden Knotenmengen R_{source} und R_{target} , diese ist in graph-tool in $O(V_c + V_c) = O(V_c)$ durchführbar (Kap. 3.3.2). Seien im Worst Case die Hälfte aller Knoten der Zusammenhangskomponente in R_{source} und die andere Hälfte in R_{target} . Um die Eigenschaft der Zusammenhangskomponente zu gewährleisten muss daher eine Kante zwischen beiden Knotenmenge existieren. Sei dies zudem eine entgegen gerichtete Kante. Sind die beiden Knoten (v, w) dieser Kante allerdings jeweils an letzter Position in R_{source} und R_{target} , so erfolgen für alle übrigen Knoten der Komponente Kantenaufrufe für die Hin- und Rückrichtung. Wie bereits oben beschrieben, benötigt ein Kantenaufwurf eine Laufzeit von $O(\min(d(v), d(w)))$, mit den Knotengraden $d(v)$ von Knoten $v \in R_{source}$ und $d(w)$ von Knoten $w \in R_{target}$ [32]. Seien nun außerdem v mit allen Knoten in R_{source} und w mit allen Knoten in R_{target} verbunden, so besitzt v einen Knotengrad von $d(v) = \frac{V_c-1}{2}$. Der Knoten w besitzt einen Knotengrad von $d(w) = \frac{V_c}{2}$, da eine entgegen gerichtete Kante von w zu v verläuft. Dann ergibt sich nach (3-25) eine Gesamtlaufzeit für `get_cigar_tuples()` von $O(V_c^3)$.

$$\begin{aligned}
& O(V_c) + O(V_c^2 \cdot \min(d(v), d(w))) \\
&= O\left(V_c + V_c^2 \cdot \min\left(\frac{V_c-1}{2}, \frac{V_c}{2}\right)\right) \\
&= O(V_c + V_c^2 \cdot V_c) \\
&= O(V_c^3)
\end{aligned} \tag{3-25}$$

In `get_allele_likelihood_read()` wurden also die Wahrscheinlichkeiten jedes Reads bestimmt, dass er von den verschiedenen Allele der Zusammenhangskomponente stammt (siehe oben). Die Wahrscheinlichkeiten im Bezug auf jedes Allel sollen nun in die zuvor bestimmten Kombinationen der Häufigkeitsverteilungen der Allele (siehe Kap. 3.4.1) nach Formel (2-xxx2) einbezogen werden. In der Funktion `calc_vafs_likelihood_read()` aus dem Modul `likelihood_operations.py` wird also für jeden Read die Wahrscheinlichkeit berechnet, diesen unter der gegebenen Allelfraktion zu beobachten. Dazu wird über die relativen Häufigkeiten jedes Allels einer gegebenen VAFs-Kombination iteriert und ihr Produkt mit der Readlikelihood aufsummiert. Da die Länge der Allelfraktion genau der Anzahl der Allele einer Zusammenhangskomponenten $n_{observed}$ entspricht, erhöht sich hierdurch die Laufzeit um den Faktor $O(n_{observed})$.

Aus dem Produkt über alle Reads wird anschließend für eine gegebene Allelfraktion nach Formel (2-xxx3) in `calc_vafs_likelihood()` die Gesamtwahrscheinlichkeit errechnet. Dadurch wird die Laufzeit zusätzlich um den Faktor der Anzahl der Reads der Zusammenhangskomponente $O(V_c)$ erhöht.

Auf diese Weise wird in `noderad_main.py` für jede Allelfraktion die Gesamtwahrscheinlichkeit über alle Reads und alle relativen Häufigkeiten der Allele berechnet und in einer Liste abgelegt. Dazu muss über alle VAFs iteriert werden. Die Anzahl der VAFs entspricht dabei genau der Anzahl der Allelkombinationen. Wie bereits in Kap. 3.4.1 besprochen, kann diese hinsichtlich der Laufzeit mit $O(e^{n_{alleles}})$ abgeschätzt werden (Formel (3-20)). Da im Worst Case $n_{alleles} = n_{observed} + \phi - 1$, ergibt sich eine Laufzeit von $O(e^{n_{observed} + \phi - 1}) \in O(e^{n_{observed}})$. Um diesen Faktor erhöht sich also die Gesamtlaufzeit bei der Berechnung der Liste der Gesamtwahrscheinlichkeiten.

Aus dieser Liste wird die Allelfraktion mit maximaler Likelihood als wahrscheinlichste Lösung gewählt und in Kap. 3.5 verwendet, um die wahrscheinlichsten Loci zu ermitteln, die diese Häufigkeitsverteilung der Allele erklären können. Hinsichtlich der Laufzeit muss jede Lösung der Liste betrachtet werden. Da die Länge der Liste der Anzahl der VAFs entspricht, kann die Laufzeit hierfür nach (3-20) auf maximal $O(e^{n_{observed}})$ geschätzt werden.

Als zusätzliche Statistiken werden in den Log-Dateien für jede Komponente die Anzahl der Allele, die Ploidie sowie die Häufigkeitsverteilung Allelfraktion mit der maximaler Likelihood und den dazugehörigen Allelsequenzen festgehalten.

3.4.3 Laufzeitanalyse für die Bestimmung der Allelfraction mit maximaler Likelihood

Die geschätzten Laufzeiten der einzelnen Schritte wurden bereits an entsprechender Stelle diskutiert und sollen hier noch einmal zusammenfassend aufgeführt werden. Zur übersichtlicheren Darstellung soll die Anzahl der Allele hier nur noch mit n bezeichnet werden, dies entspricht der bisher verwendeten Variable $n_{observed}$. Für jede Zusammenhangskomponente werden vorab die Kandidatenallele bestimmt $O(V_c)$, die Anzahl der tatsächlich zu erwartenden Allele berechnet $O(V_c)$ und die Kombinationen möglicher Häufigkeitsverteilungen ermittelt $O(e^n \cdot n^2)$. Anschließend erfolgt die Likelihoodberechnung in $O(V_c^3)$ über alle Allelfractionen $O(e^n)$, für alle Reads $O(V_c)$ und im Vergleich jedes Reads zu allen Kandidatenallelen $O(n)$. Aus den Lösungen wird im Anschluss das Maximum bestimmt $O(e^n)$. Daraus ergibt sich nach 3-26 eine Gesamtlaufzeit für die Bestimmung der Allelfraction mit maximaler Likelihood von $O(e^n \cdot (n^2 + n \cdot V_c^4))$.

$$\begin{aligned}
O(V_c) + O(V_c) + O(e^n \cdot n^2) + O(e^n) \cdot O(V_c) \cdot O(n) \cdot O(V_c^3) + O(e^n) \\
= O(2V_c + e^n \cdot n^2 + e^n + e^n \cdot n \cdot V_c^4) \\
= O(2V_c + e^n \cdot (n^2 + 1 + n \cdot V_c^4)) \\
= O(e^n \cdot (n^2 + n \cdot V_c^4))
\end{aligned} \tag{3-26}$$

Im Worst Case entspricht n genau der Anzahl der Reads in der Zusammenhangskomponente und es gilt $n = n_{observed} = V_c$, so dass die Laufzeit dann auf $O(e^{V_c} \cdot V_c^5)$ geschätzt werden kann (3-27).

$$\begin{aligned}
O(e^n \cdot (n^2 + n \cdot V_c^4)) &= O(e^{V_c} \cdot (V_c^2 + V_c \cdot V_c^4)) \\
&= O(e^{V_c} \cdot (V_c^2 + V_c^5)) \\
&= O(e^{V_c} \cdot V_c^5)
\end{aligned} \tag{3-27}$$

Die Laufzeit wird also durch die Anzahl der VAFs dominiert und kann bei großen Clustern mit vielen Allelen problematisch werden. Die konfigurierbaren Schwellenwerte `threshold-cluster-size` und `threshold-seq-noise` ermöglichen eine Anpassung solcher großen Cluster, um die Anzahl der Kombinationen und damit den Rechenaufwand flexibel reduzieren zu können.

3.5 Bestimmung der maximalen Likelihood der Loci

In Abhängigkeit von der Anzahl der Kandidatenallele und der Ploidie können die Zusammenhangskomponenten auch mehr als nur einen Locus beinhalten. Daher soll die in Kap. 3.4 identifizierte Allelfraction mit maximaler Likelihood nun passenden Loci-Kombinationen zugeordnet werden. Unter Einbeziehung der Heterozygotiewahrscheinlichkeiten soll daraus die wahrscheinlichste Loci-Zuordnung ermittelt werden.

3.5.1 Bestimmung der möglichen Loci

Für die Zuordnung der Loci müssen zunächst die verschiedenen Kombinationen möglicher Allelverteilungen auf die Loci durch `get_candidate_loci()` im Modul `likelihood_operations.py` erzeugt werden. Analog zu `get_candidate_vafs()` (Kap. 3.4.1) werden aus der beobachteten $n_{observed}$ und der tatsächlich zu erwartenden Anzahl an Allelen $n_{alleles}$ alle Allelkombinationen mit Wiederholung und ohne Berücksichtigung der Reihenfolge (vgl. Formel (3-12)) durch die Funktion `combinations_with_replacement()` aus der Python-Library `itertools` gebildet. Allerdings sollen nun die Allelkombinationen zu Tupeln entsprechend ihrer Ploidie zusammengefasst werden. Jedes Tupel repräsentiert einen möglichen Locus in der Kombination. Die Allelkombination durch `combinations_with_replacement()` sind lexikographisch sortiert. Für jede der Kombinationen muss nun aber die Reihenfolge so variiert werden, dass die Allele, falls mehrere Loci möglich sind, in verschiedenen Kombinationen den Loci zugeordnet werden. Hierfür wird aus `itertools` die Funktion `permutations` verwendet. Diese bildet aus n Elementen alle Kombinationen der Länge k mit Beachtung der Reihenfolge und ohne Wiederholungen. Auch wenn sich die Allele in den Allelkombinationen durchaus wiederholen können, so werden sie doch in `permutations` als einzigartiges Element behandelt unabhängig von seinem Wert. Folglich entstehen durch die Bildung der Permutationen auch einige Duplikate. Die Anzahl der entstehenden Elemente wird für `permutations` mit $\frac{n!}{(n-k)!}$ angegeben [37]. Die Permutationen werden über jede Allelkombination in ihrer vollständigen Länge gebildet, so dass gilt $n = k$, folglich werden insgesamt $n!$ Elemente erzeugt (3-28).

$$\frac{n!}{(n-k)!} = \frac{n!}{(n-n)!} = \frac{n!}{0!} = \frac{n!}{1} = n! \quad (3-28)$$

Mit Hilfe der `itertools`-Funktion `grouper()` werden die Permutationen entsprechend der Ploidie in Tupel der Länge ϕ unterteilt. Es resultieren Tupel von ϕ -Tupeln, wobei die Anzahl der ϕ -Tupel der Anzahl möglicher Loci entspricht. Mehrere Loci in einer Zusammenhangskomponenten sind nach `get_max_parsimony_n_alleles()` genau dann möglich, wenn gilt $n_{observed} > \phi$.

Nach der Gruppierung werden die übergeordneten Tupel sowie die darin enthaltenen ϕ -Tupel jeweils sortiert, um anschließend durch die Python-Funktion `set()` die oben erwähnten Duplikate zu entfernen.

Beispiele möglicher Loci-Kombinationen:

$ploidy = 2, n_{observed} = 3, n_{alleles} = 4$:

Kombinationen der Allele:

[(0, 0, 0, 0), (0, 0, 0, 1), (0, 0, 0, 2), (0, 0, 1, 1), (0, 0, 1, 2), (0, 0, 2, 2), (0, 1, 1, 1), (0, 1, 1, 2), (0, 1, 2, 2), (0, 2, 2, 2), (1, 1, 1, 1), (1, 1, 1, 2), (1, 1, 2, 2), (1, 2, 2, 2), (2, 2, 2, 2)]

Permutationen:

(Zur anschaulicheren Darstellung wurden Duplikate entfernt, das geschieht eigentlich erst nach der Erzeugung und Sortierung der Tupel)

(1, 2, 1, 1), (2, 1, 0, 0), (2, 1, 1, 1), (0, 1, 2, 1), (0, 1, 1, 2), (0, 1, 0, 0), (2, 2, 1, 0), (0, 2, 2, 1), (2, 2, 0, 1), (1, 0, 2, 2), (0, 2, 0, 1), (2, 0, 0, 1), (1, 0, 1, 0), (0, 2, 1, 2), (0, 0, 2, 0), (2, 2, 2, 1), (1, 1, 0, 1), (2, 0, 1, 1), (2, 0, 2, 0), (0, 0, 2, 2), (1, 1, 2, 0), (1, 2, 1, 0), (2, 0, 2, 2), (2, 1, 1, 0), (2, 1, 0, 2), (1, 2, 0, 1), (0, 1, 2, 0), (1, 2, 1, 2), (1, 2, 2, 1), (0, 1, 1, 1), (1, 1, 1, 0), (0, 0, 0, 0), (2, 1, 1, 2), (2, 1, 2, 1), (1, 0, 0, 1), (0, 1, 0, 2), (2, 2, 1, 2), (0, 2, 2, 0), (1, 0, 2, 1), (2, 0, 0, 0), (0, 2, 1, 1), (1, 1, 1, 2), (0, 0, 0, 2), (0, 0, 1, 1), (1, 0, 1, 2), (2, 0, 0, 2), (0, 0, 2, 1), (1, 1, 2, 2), (2, 1, 0, 1), (1, 2, 0, 0), (0, 1, 2, 2), (1, 2, 2, 0), (0, 1, 1, 0), (2, 2, 0, 0), (0, 2, 0, 0), (2, 1, 2, 0), (1, 0, 0, 0), (1, 2, 0, 2), (0, 1, 0, 1), (2, 2, 1, 1), (2, 2, 2, 0), (1, 1, 0, 0), (1, 0, 2, 0), (0, 2, 2, 2), (1, 2, 2, 2), (2, 2, 0, 2), (0, 2, 1, 0), (0, 2, 0, 2), (2, 0, 1, 0), (0, 0, 0, 1), (1, 1, 1, 1), (0, 0, 1, 0), (2, 1, 2, 2), (1, 0, 0, 2), (1, 0, 1, 1), (2, 2, 2, 2), (1, 1, 0, 2), (2, 0, 1, 2), (2, 0, 2, 1), (0, 0, 1, 2), (1, 1, 2, 1)

Loci:

((0, 0), (0, 2)), ((1, 1), (1, 1)), ((0, 2), (0, 2)), ((1, 1), (2, 2)), ((0, 1), (0, 2)), ((1, 1), (1, 2)), ((1, 2), (2, 2)), ((1, 2), (1, 2)), ((0, 0), (1, 1)), ((0, 0), (2, 2)), ((0, 2), (1, 1)), ((0, 1), (1, 1)), ((0, 0), (0, 0)), ((0, 2), (2, 2)), ((0, 0), (1, 2)), ((0, 1), (2, 2)), ((2, 2), (2, 2)), ((0, 0), (0, 1)), ((0, 2), (1, 2)), ((0, 1), (1, 2)), ((0, 1), (0, 1))

Wie bereits in Kap. 3.4.1 besprochen, benötigt die Generierung aller Allelkombinationen eine Laufzeit von $O(e^n)$, wobei $n = n_{observed}$ die Anzahl der beobachteten Allele der Zusammenhangskomponente ist. Für diese Kombinationen werden jeweils alle $n!$ Permutationen gebildet, also mit einer Laufzeit von $O(n!)$. Für jede Permutation erfolgt über ihre jeweilige Länge die Gruppierung zu den Loci. Die Länge jeder Permutation kann wegen der Funktion `get_max_parsimony_n_alleles()` (vgl. Kap. 3.4.1) maximal $n_{alleles} = n_{observed} + \phi - 1$ betragen. Wegen $\phi = const$ beträgt die Laufzeit hier also $O(n_{observed}) = O(n)$. Die Sortierung bei Python erfordert eine Laufzeit von $O(n \cdot \log(n))$ [39]. Die beiden Sortiervorgänge erfolgen zum einen über jeden Locus mit der Länge ϕ entsprechend seiner Ploidie und

über alle Loci innerhalb der Permutation. Die Anzahl der Loci entspricht $\frac{n}{\phi}$, dadurch ergeben sich für die Sortiervorgänge Laufzeiten von $O(\phi \cdot \log(\phi))$ bzw. $O\left(\frac{n}{\phi} \cdot \log\left(\frac{n}{\phi}\right)\right)$. Unter Berücksichtigung, dass $\phi = \text{const}$ gilt, beträgt die Laufzeit für `get_candidate_loci()` insgesamt $O(e^n \cdot n! \cdot n^2 \cdot \log(n))$.

$$\begin{aligned}
& O(e^n) \cdot O(n!) \cdot O(n) \cdot O(\phi \cdot \log(\phi)) \cdot O\left(\frac{n}{\phi} \cdot \log\left(\frac{n}{\phi}\right)\right) \\
&= O\left(e^n \cdot n! \cdot n \cdot \phi \cdot \log(\phi) \cdot \frac{n}{\phi} \cdot \log\left(\frac{n}{\phi}\right)\right) \\
&= O\left(e^n \cdot n! \cdot n^2 \cdot \log(\phi) \cdot \log\left(\frac{n}{\phi}\right)\right) \\
&\in O(e^n \cdot n! \cdot n^2 \cdot \log(n))
\end{aligned} \tag{3-29}$$

Große Cluster sind hier also hinsichtlich der Laufzeit und des Speicherbedarfs noch problematischer als bei der Generierung der Allelkombinationen in der Funktion `get_candidate_vafs()` (Kap. 3.4.1). Eine sinnvolle Anpassung der Grenzwerte `threshold-cluster-size` und `threshold-seq-noise` ermöglicht diesbezüglich eine Verbesserung. Allerdings ist an dieser Stelle ein effizienterer Ansatz zur Bestimmung der Loci-Kombinationen möglich, der in diesem Prototypen zwar keine Anwendung fand, aber für die spätere Implementierung in Rust von Bedeutung sein könnte. Dieser Ansatz wird im Ausblick in Kap. 5 kurz beschrieben.

3.5.2 Zuordnung der Allelverteilung mit maximaler Likelihood zur wahrscheinlichsten Loci-Kombination

In diesem Schritt sollen schließlich für die Allelfraktion mit maximaler Likelihood (Kap.3.4.2) die ermittelten Kandidatenloci (Kap. 3.5.1) unter Berücksichtigung der Heterozygotie analysiert werden und die wahrscheinlichste Kombination der Loci ermittelt werden.

Für jede mögliche Kombination der Loci aus `get_candidate_loci()` wird aus der Funktion `calc_loci_likelihooods()` heraus zunächst geprüft, ob sich die Loci dieser Kombination der zuvor bestimmten Allelfraktion mit maximaler Likelihood zuordnen lassen. Dies geschieht mit Hilfe der Indikatorfunktion `indicator_constraint()` nach Formel (2-xxx4). Ist die Bedingung der Indikatorfunktion erfüllt, so wird die Likelihood der Loci-Kombination aus den Heterozygotiewahrscheinlichkeiten der darin enthaltenen Allele berechnet und zurückgegeben. Ist die Bedingung der Indikatorfunktion nicht erfüllt, so wird eine Likelihood von 0 zurückgegeben.

Für die Likelihoodberechnung bei erfüllter Indikatorbedingung müssen einer Loci-Kombination zunächst die passenden Allelsequenzen zugeordnet werden. Die Allelnummer in einer Loci-Kombination entspricht dabei dem Index der dazugehörigen Allelsequenz in der Liste

der Kandidatenallele aus `get_candidate_alleles()` (siehe Kap. 3.4.1). Zu jeder Loci-Kombination wird also eine Liste $L_{alleles}$ generiert, welche die Kandidatenallelsequenzen der einzelnen Loci als Sublisten beinhaltet.

Analog zur Likelihoodberechnung beim paarweisen Vergleich der Reads unter Berücksichtigung der Sequenzierfehlerwahrscheinlichkeit 3.3.2 erfolgt auch die Berechnung der Likelihood für eine bestimmte Loci-Kombination durch den paarweisen Vergleich der Allele im Sinne eines pair Hidden Markov Models $pairHMM_{\eta}(a_{l_{j,1}}, a_{l_{j,2}})$ (vgl. Kap. 2.3.2). Allerdings werden nun die Heterozygotiewahrscheinlichkeiten η_{sub} , η_{ins} und η_{del} der Grapheneigenschaften zur Ermittlung der Likelihood nach Formel (2-xxx5) genutzt.

Hierzu werden zunächst in `get_allele_likelihood_alleles()` (Algorithmus 3.4) auf jeweils einer der Listen $L_{alleles}$ für jeden Locus die darin enthaltenen Allelsequenzen paarweise mit einander kombiniert. Dies wird durch die Funktion `combinations()` aus `itertools` realisiert, welche Kombinationen ohne Wiederholung und ohne Beachtung der Reihenfolge erzeugt. Für jedes Allelpaar wird dann der CIGAR-String durch die in Kap. 3.4.2 bereits beschriebene Funktion `get_cigar_tuples()` ermittelt und für die Likelihoodberechnung in `get_heterozygosity()` verwendet (siehe Algorithmus 3.5). Die Gesamtwahrscheinlichkeit errechnet sich aus dem Produkt der Likelihood aller Allelpaare nach 2-xxx6.

Algorithmus 3.4 Bestimmung der Likelihood der Allele innerhalb einer Loci-Kombination

```

1: function GET_ALLELE_LIKELIHOOD_ALLELES( $C_k, L_{alleles}$ )
2:    $likelihood \leftarrow 1.0$ 
3:   for  $loc \in L_{alleles}$  do
4:      $pairs \leftarrow \binom{loc}{2}$ 
5:     for  $(a_i, a_j) \in pairs$  do
6:        $cigar \leftarrow get\_cigar\_tuples(C_k, a_i, a_j)$  ▷ Alg. 3.3
7:       if  $cigar$  exists then
8:          $cig \leftarrow cigar[0]$ 
9:          $rev \leftarrow cigar[1]$ 
10:         $\eta_{rates} \leftarrow C_k[\eta_{sub}, \eta_{ins}, \eta_{del}]$ 
11:         $pr \leftarrow get\_heterozygosity(C_k[\eta_{sub}], C_k[\eta_{ins}], C_k[\eta_{del}], cig, rev)$  ▷ Alg. 3.5
12:         $likelihood \leftarrow likelihood + \log(pr)$ 
13:      end if
14:    end for
15:  end for
16:  return  $e^{likelihood}$ 
17: end function

```

Wie bereits erwähnt, erfolgt in `get_heterozygosity()` die Likelihoodberechnung der paarweisen Kombinationen der Allelsequenzen eines Locus aus den CIGAR-Tupeln und den Heterozygotiewahrscheinlichkeiten. Analog zum Algorithmus 3.1 wird auch hier die Likelihood über aller Matches und Mismatches berechnet.

Algorithmus 3.5 Bestimmung der Likelihood zwischen zwei Allelen hinsichtlich der Heterozygotiewahrscheinlichkeiten

```

1: function GET_HETEROZYGOSITY( $\eta_{sub}$ ,  $\eta_{ins}$ ,  $\eta_{del}$ , CIGAR-Tuples, reverse)
2:   likelihood  $\leftarrow$  1.0
3:   if reverse then
4:     swap values of  $\eta_{ins}$  and  $\eta_{del}$ 
5:   end if
6:    $\eta_{all} \leftarrow \eta_{sub} + \eta_{ins} + \eta_{del}$ 
7:   for all (operation, length)  $\in$  CIGAR-Tuples do
8:     if operation  $\in$  match then
9:       likelihood  $\leftarrow$  likelihood  $\cdot (1 - \eta_{all})^{length}$ 
10:    end if
11:    if operation  $\in$  mismatch then
12:       $\eta_{factor} \leftarrow 1.0$ 
13:      if operation  $\in$  substitution then
14:         $\eta_{factor} \leftarrow \eta_{sub}$ 
15:      end if
16:      if operation  $\in$  insertion then
17:         $\eta_{factor} \leftarrow \eta_{ins}$ 
18:      end if
19:      if operation  $\in$  deletion then
20:         $\eta_{factor} \leftarrow \eta_{del}$ 
21:      end if
22:      likelihood  $\leftarrow$  likelihood  $\cdot (\eta_{factor})^{length}$ 
23:    end if
24:  end for
25:  return likelihood
26: end function

```

Dabei entspricht im Falle eines Mismatches die Likelihood der in der Konfigurationsdatei angegebenen Heterozygotiewahrscheinlichkeit η für die betreffende Mutationsart, also η_{sub} für Substitutionen, η_{ins} für Insertionen bzw. η_{del} Deletionen. Sei i der Index der

betreffenden Base innerhalb der betrachteten Allelsequenz und $\eta_{rate} \in \{\eta_{sub}, \eta_{ins}, \eta_{del}\}$, dann berechnet sich bei einem Mismatch die Likelihood L_i der Base nach Formel (3-30).

$$L_{i_{mismatch}} = \eta_{rate} \quad (3-30)$$

Im Falle eines Matches senkt die Möglichkeit eines Mismatches die Likelihood der betreffenden Base entsprechend um die Summe der Heterozygotiewahrscheinlichkeiten der genannten Mutationsarten (Formel (3-31)).

$$L_{i_{match}} = 1 - (\eta_{sub} + \eta_{ins} + \eta_{del}) \quad (3-31)$$

Die Likelihood $Pr(T = a_{l_{j,2}} | S = a_{l_{j,1}}, \eta)$ des paarweisen Vergleichs der Allele $a_{l_{j,1}}$ und $a_{l_{j,2}}$ hinsichtlich der Zuordnung zu den Loci-Kombinationen errechnet sich schließlich aus dem Produkt der Wahrscheinlichkeiten der einzelnen Basen:

$$Pr(T = a_{l_{j,2}} | S = a_{l_{j,1}}, \eta) = pairHMM_{\eta}(a_{l_{j,1}}, a_{l_{j,2}}) = \prod_{i=1}^k L_i \quad (3-32)$$

In `noderad_main.py` werden die beschriebenen Schritte zur Likelihoodberechnung für alle Permutationen ausgeführt und anschließend die Loci-Kombination mit maximaler Likelihood als wahrscheinlichste Loci-Verteilung ausgewählt.

In den Log-Dateien werden zusätzlich einige Statistiken zur Berechnung der wahrscheinlichsten Loci-Verteilung eingetragen. Dazu gehören die ermittelten Likelihoods aller Permutationen sowie die Loci mit maximaler Likelihood.

3.5.3 Laufzeitanalyse für die Bestimmung der Loci-Kombination mit maximaler Likelihood

Die Gesamtlaufzeit ergibt sich wiederum aus den einzelnen Laufzeiten der oben beschriebenen Funktionen. Diese Funktionen werden über allen Permutationen der Allelkombinationen in `noderad_main.py` aufgerufen, wodurch die Laufzeit um den Faktor $O(e^n \cdot n!)$ steigt (vgl. Kap. 3.5.1). Die Indikatorfunktion prüft die Bedingung über die gesamte Länge einer der Permutationen und benötigt somit eine Laufzeit von $O(n)$. Anschließend wird in `calc_loci_likelihoods()` über alle Loci einer Permutation in $O(\frac{n}{\phi})$ iteriert und die Funktion `get_allele_likelihood_allele()` aufgerufen. Diese ermittelt bei jedem Locus in $O(\phi)$ die Kombinationen ohne Wiederholung und ohne Beachtung der Reihenfolge. Nach (3-13) und (3-20) erhöhen die Kombinationen dabei die Laufzeit um den Faktor $O(\binom{n}{2}) \in O(e^n)$. Die Bestimmung der CIGAR-Tupel für jede Kombination vergrößert die Laufzeit zusätzlich um den Faktor $O(V_c^3)$ (siehe Kap. 3.4.2). Über die CIGAR-Tupel wird

schließlich für jede Base die Likelihood aus den Heterozygotiewahrscheinlichkeiten berechnet, analog zur Likelihoodberechnung zweier Reads in Kap. 3.3.2 kann dies mit $O(\bar{k})$ veranschlagt werden, wobei $\bar{k} = \text{const}$ gilt. Da für alle Permutationen der Kombinationen die Berechnung Likelihood erfolgt, wird die anschließende Maximumsbestimmung über deren Anzahl also in $O(e^n \cdot n!)$ durchgeführt.

Nach (3-33) ergibt sich damit für die Berechnung der wahrscheinlichsten Loci-Kombination insgesamt eine Laufzeit von $O(e^n \cdot n! \cdot V_c^3)$.

$$\begin{aligned}
& O(e^n \cdot n!) \cdot \left(O(n) + O\left(\frac{n}{\phi}\right) \cdot O(\phi) \cdot O(e^n) \cdot O(V_c^3) \cdot O(\bar{k}) \right) + O(e^n \cdot n!) \\
&= O(e^n \cdot n! \cdot (n + n \cdot e^n \cdot V_c^3) + e^n \cdot n!) \\
&= O(e^n \cdot n! \cdot n + e^{2n} \cdot n! \cdot n \cdot V_c^3) \\
&= O(e^{2n} \cdot n! \cdot n \cdot V_c^3) \\
&\leq O(e^{2n} \cdot n! \cdot (n + 1) \cdot V_c^3) \\
&= O(e^{2n} \cdot (n + 1)! \cdot V_c^3) \\
&\in O(e^n \cdot n! \cdot V_c^3)
\end{aligned} \tag{3-33}$$

Wie bereits in Kap. 3.4.3 durchgeführt, soll auch hier der Worst Case betrachtet werden, bei dem die Anzahl der Kandidatenallele genau der Anzahl der Reads in der Zusammenhangskomponente entspricht, also $n = n_{\text{observed}} = V_c$. Dann ergibt sich für die Berechnung der wahrscheinlichsten Loci-Zuordnung nach (3-34) eine maximale Laufzeit von $O(e^{V_c} \cdot V_c!)$.

$$\begin{aligned}
O(e^n \cdot n! \cdot V_c^3) &= O(e^{V_c} \cdot V_c! \cdot V_c^3) \\
&\leq O(e^{V_c} \cdot V_c! \cdot (V_c + 1) \cdot (V_c + 2) \cdot (V_c + 3)) \\
&= O(e^{V_c} \cdot (V_c + 3)!) \\
&\in O(e^{V_c} \cdot V_c!)
\end{aligned} \tag{3-34}$$

Die Anzahl der Permutationen dominiert durch $O(V_c!)$ die Laufzeit noch stärker als bei der Ermittlung der Allelkombinationen in Kap. 3.4.3. Dies ist für große Cluster sehr problematisch. Die Anpassung der Schwellenwerte `threshold-cluster-size` und `threshold-seq-noise` können dies zwar in einem gewissen Rahmen für den Prototyp kompensieren. Bei der späteren Implementierung in Rust sollte hier jedoch ein effizienterer Ansatz gewählt werden. Eine effizientere Vorgehensweise diesbezüglich wird in Kap. 5 vorgestellt.

3.6 Abschließende Laufzeitanalyse des gesamten Algorithmus

Für jeden Konstruktions- und Berechnungsschritt wurden die Laufzeiten bereits an entsprechender Stelle angesprochen. Zusammenfassend ergab sich dabei für die einmalige Konstruktion des Graphen und die Extraktion seiner Zusammenhangskomponenten C eine

Laufzeit von $O(V \cdot (V + E))$ (Kap. 3.3.4). Für jede Zusammenhangskomponente erfolgte die Berechnung der wahrscheinlichsten Allelkombination in $O(e_c^V \cdot V_c^5)$ (Kap. 3.4.3) und die Berechnung der wahrscheinlichsten Loci-Zuordnung in $(e_c^V \cdot V_c!)$ (Kap. 3.5.3). Die Laufzeiten können also wie folgt zusammengefasst werden:

$$\begin{aligned}
& O(V \cdot (V + E)) + O\left(\sum_{c=1}^{|C|} (e^{V_c} \cdot V_c^5 + e^{V_c} \cdot V_c!)\right) \\
&= O\left(V \cdot (V + E) + \sum_{c=1}^{|C|} (e^{V_c} \cdot V_c^5) + \sum_{c=1}^{|C|} (e^{V_c} \cdot V_c!)\right) \\
&\in \left(O\sum_{c=1}^{|C|} (e^{V_c} \cdot V_c!)\right)
\end{aligned} \tag{3-35}$$

Besitzt im Worst Case der Graph nur eine einzige Zusammenhangskomponente, welche alle Knoten des Graphen enthält, so dass gilt $|C| = 1$ und $V_c = V$ so würde sich hieraus eine maximale Laufzeit von $O(e^V \cdot V!)$ ergeben.

3.7 Ausgabe der wahrscheinlichsten Loci als VCF-Datei

Die Loci-Kombinationen mit maximaler Likelihood Loc_{max} aller Zusammenhangskomponenten werden für jedes Individuum jeweils in eine Datei im Variant Call Format (VCF, [40]) geschrieben. Wie für dieses Format üblich, gibt es acht notwendige tabulatorgetrennte Spalten: CHROM, POS, ID, REF, ALT, QUAL, FILTER, INFO. Außerdem wird hier die optionale Spalte FORMAT sowie eine Spalte zur Probe verwendet.

Die Spalte CHROM wird hier für die Benennung der Loci genutzt. Die Bezeichnung jedes Locus wird dabei aus dem Präfix "LOC" in Kombination mit einer fortlaufenden Nummer gebildet. Die Spalten ID, QUAL, FILTER und INFO werden hier nicht genutzt und mit einem Punkt als Platzhalter bei allen Loci befüllt.

In die Spalten REF und ALT sollen die Sequenzen der Allele aus Loc_{max} lexikographisch sortiert geschrieben werden. Hierfür wird in der Funktion `get_sorted_loci_alleles()` eine sortierte Liste der Allelsequenzen aus Loc_{max} gebildet. Durch die Pythonfunktion `set()` wird über dieser Liste die Menge der Allele extrahiert, so dass Duplikate herausgefiltert werden. Aus der so entstandenen Liste der Allelsequenzen A_{res} wird der erste Eintrag in die Spalte REF eingetragen, die übrigen Einträge werden in die Spalte ALT geschrieben.

Da jeweils die vollständige Sequenz der Allele für den Eintrag in die VCF-Datei verwendet wird, wird für jeden Locus in der Spalte POS der Wert 1 eingetragen.

In der FORMAT-Spalte wird durch den Eintrag "GT" festgelegt, dass in der Proben-
spalte der Genotyp spezifiziert wird. Da die Indizes der Loci der Liste der Kandidatenallele
zugeordnet sind, aber nicht alle Allele dieser Liste auch in Loc_{max} vorkommen, muss ihre
Indizierung nun auf die bereits erstellte Liste mit den zu Loc_{max} gehörigen Allelsequenzen
 A_{res} angepasst werden.

Hierfür werden in der Funktion `get_alleles_matched_to_loci()` zunächst jedem Lo-
cus aus Loc_{max} die entsprechenden Sequenzen aus der Liste der Kandidatenallele zugeord-
net. Anschließend werden diese Sequenzen in `get_gt_indices()` den Indizes der passenden
Sequenz aus A_{res} zugeordnet. Dadurch wird das lexikographisch erste Allel, also REF mit
0 im Genotyp indiziert, die Allele aus ALT erhalten höhere Indizes entsprechend ihrer
Sortierung in A_{res} . Nun lässt sich der Genotyp im Bezug auf die Sequenzen in REF und
ALT aus diesen Indizes und getrennt durch Slashes direkt angeben. Dies geschieht in der
Funktion `get_genotype()`, deren Ergebnis dann in die Probenspalte eingetragen wird.

Kapitel 4

Evaluation an simulierten Datensätzen

prototyp, zunächst für ploidie 2 -> bei höherer ploidie

4.1

4.1.1

Kapitel 5

Zusammenfassung und Ausblick

var callen

verschiedene Individuen mit einander vergleichen bei gleichem Ansatz, aber nun mit locus seq. aus den vcf's f mehrere individuen als eingabe -> vergleich der individuen

Diversität -> je diverser, desto besser geht es der Population

SE-Reads besser geeignet für Analyse: bei mutation im bereich einer der rev-schnittstelle würde bei PE nicht geschnitten werden und der read würde nicht geclustert und ausgewertet werden -> heterozygotie nicht mehr sichtbar, bei SE sind aber die reads unabhängig von einander, die Heterozygotie an dieser stelle würde erfasst werden

eventuell Vgl mit ILP:

-> ILP wird sehr groß, die Lösung wird 2 schrittig gefunden, aber dadurch kein globales Optimum, dass für jede optimale Zuordnung diese durch erwartete Sequenziertiefe erklärt -> es könnte ebenso eine nicht optimale Lösung im ILP eine optimale Lösung in zusammenschau mit der seq-tiefe werden

=> Seq-tiefe ist nur sehr bedingt als Maß geeignet, z.B. Cutting-Enzym schneidet nicht richtig -> seq-tiefe und häufigkeit ändern sich

5.1

5.1.1

Algorithmenverzeichnis

3.1	Berechnung der Likelihood zwischen zwei Reads	20
3.2	Bestimmung der Likelihood zwischen einem Read und einem Kandidatenallel	30
3.3	CIGAR-Tupel bestimmen	33
3.4	Bestimmung der Likelihood der Allele innerhalb einer Loci-Kombination . .	39
3.5	Bestimmung der Likelihood zwischen zwei Allelen hinsichtlich der Hetero- zygotiewahrscheinlichkeiten	40

Literaturverzeichnis

- [1] WATSON, J. D. and F. H. CRICK: *Molecular structure of nucleic acids; a structure for deoxyribose nucleic acid.* Nature, 171:737–8, Apr 1953.
- [2] MARTIN, R. G., J. H. MATTHAEI, O. W. JONES, and M. W. NIRENBERG: *Ribonucleotide composition of the genetic code.* Biochemical and biophysical research communications, 6:410–4, Jan 1962.
- [3] MATTHAEI, H. and M. W. NIRENBERG: *The dependence of cell-free protein synthesis in e. coli upon rna prepared from ribosomes.* Biochemical and biophysical research communications, 4:404–8, Apr 1961.
- [4] DUNHAM, IAN, ANSHUL KUNDAJE, SHELLEY F ALDRED, PATRICK J COLLINS, CARRIE A DAVIS, FRANCIS DOYLE, CHARLES B EPSTEIN, and SETH FRIETZE: *An integrated encyclopedia of dna elements in the human genome.* Nature, 489:57–74, Sep 2012.
- [5] TSAGAKIS, IOANNIS, KATERINA DOUKA, ISABEL BIRDS, and JULIE L. ASPDEN: *Long non-coding rnas in development and disease: conservation to mechanisms.* The Journal of pathology, 250:480–495, Apr 2020.
- [6] O'DONNELL, MICHAEL, LANCE LANGSTON, and BRUCE STILLMAN: *Principles and concepts of dna replication in bacteria, archaea, and eukarya.* Cold Spring Harbor perspectives in biology, 5, Jul 2013.
- [7] CHAGIN, VADIM O., JEFFREY H. STEAR, and M. CRISTINA CARDOSO: *Organization of dna replication.* Cold Spring Harbor perspectives in biology, 2:a000737, Apr 2010.
- [8] PRIOLEAU, MARIE-NOËLLE and DAVID M. MACALPINE: *Dna replication origins-where do we begin?* Genes & development, 30:1683–97, Aug 2016.
- [9] VIGNAL, ALAIN, DENIS MILAN, MAGALI SANCRISTOBAL, and ANDRÉ EGGEN: *A review on snp and other types of molecular markers and their use in animal genetics.* Genetics, selection, evolution : GSE, 34:275–305, May-Jun 2002.

- [10] SACHIDANANDAM, R., D. WEISSMAN, S. C. SCHMIDT, J. M. KAKOL, L. D. STEIN, G. MARTH, S. SHERRY, J. C. MULLIKIN, B. J. MORTIMORE, D. L. WILLEY, S. E. HUNT, C. G. COLE, P. C. COGGILL, C. M. RICE, Z. NING, J. ROGERS, D. R. BENTLEY, P. Y. KWOK, E. R. MARDIS, R. T. YEH, B. SCHULTZ, L. COOK, R. DAVENPORT, M. DANTE, L. FULTON, L. HILLIER, R. H. WATERSTON, J. D. MCPHERSON, B. GILMAN, S. SCHAFFNER, W. J. VAN ETTEN, D. REICH, J. HIGGINS, M. J. DALY, B. BLUMENSTIEL, J. BALDWIN, N. STANGE-THOMANN, M. C. ZODY, L. LINTON, E. S. LANDER, and D. ALTSHULER: *A map of human genome sequence variation containing 1.42 million single nucleotide polymorphisms*. Nature, 409:928–33, Feb 2001.
- [11] KRUGLYAK, L.: *The use of a genetic map of biallelic markers in linkage studies*. Nature genetics, 17:21–4, Sep 1997.
- [12] KWOK, PUI-YAN and XIANGNING CHEN: *Detection of single nucleotide polymorphisms*. Current issues in molecular biology, 5:43–60, Apr 2003.
- [13] NIELSEN, RASMUS: *Population genetic analysis of ascertained snp data*. Human genomics, 1:218–24, Mar 2004.
- [14] SHRIVER, MARK D., GIULIA C. KENNEDY, ESTEBAN J. PARRA, HEATHER A. LAWSON, VIBHOR SONPAR, JING HUANG, JOSHUA M. AKEY, and KEITH W. JONES: *The genomic distribution of population substructure in four populations using 8,525 autosomal snps*. Human genomics, 1:274–86, May 2004.
- [15] AKEY, JOSHUA M., GE ZHANG, KUN ZHANG, LI JIN, and MARK D. SHRIVER: *Interrogating a high-density snp map for signatures of natural selection*. Genome research, 12:1805–14, Dec 2002.
- [16] WANG, D. G., J. B. FAN, C. J. SIAO, A. BERNO, P. YOUNG, R. SAPOLSKY, G. GHANDOUR, N. PERKINS, E. WINCHESTER, J. SPENCER, L. KRUGLYAK, L. STEIN, L. HSIE, T. TOPALOGLOU, E. HUBBELL, E. ROBINSON, M. MITTMANN, M. S. MORRIS, N. SHEN, D. KILBURN, J. RIOUX, C. NUSBAUM, S. ROZEN, T. J. HUDSON, R. LIPSHUTZ, M. CHEE, and E. S. LANDER: *Large-scale identification, mapping, and genotyping of single-nucleotide polymorphisms in the human genome*. Science (New York, N.Y.), 280:1077–82, May 1998.
- [17] SANGER, F. and A. R. COULSON: *A rapid method for determining sequences in dna by primed synthesis with dna polymerase*. Journal of molecular biology, 94:441–8, May 1975.

- [18] MULLIS, K., F. FALOONA, S. SCHARF, R. SAIKI, G. HORN, and H. ERLICH: *Specific enzymatic amplification of dna in vitro: the polymerase chain reaction*. Cold Spring Harbor symposia on quantitative biology, 51 Pt 1:263–73, 1986.
- [19] GAWAD, CHARLES, WINSTON KOH, and STEPHEN R. QUAKE: *Single-cell genome sequencing: current state of the science*. Nature reviews. Genetics, 17:175–88, Mar 2016.
- [20] CATCHEN, JULIAN, PAUL A. HOHENLOHE, SUSAN BASSHAM, ANGEL AMORES, and WILLIAM A. CRESKO: *Stacks: an analysis tool set for population genomics*. Molecular ecology, 22:3124–40, Jun 2013.
- [21] KÖSTER, JOHANNES and SVEN RAHMANN: *Snakemake—a scalable bioinformatics workflow engine*. Bioinformatics, 28(19):2520–2522, 08 2012.
- [22] KÖSTER, JOHANNES and SVEN RAHMANN: *Building and Documenting Workflows with Python-Based Snakemake*. In BÖCKER, SEBASTIAN, FRANZISKA HUFISKY, KERSTIN SCHEUBERT, JANA SCHLEICHER, and STEFAN SCHUSTER (editors): *German Conference on Bioinformatics 2012*, volume 26 of *OpenAccess Series in Informatics (OASIs)*, pages 49–56, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [23] MARTIN, MARCEL: *Cutadapt removes adapter sequences from high-throughput sequencing reads*. EMBnet.journal, 17(1):10–12, 2011.
- [24] ANDREWS, SIMON, FELIX KRUEGER, ANNE SEGONDS-PICHON, LAURA BIGGINS, CHRISTEL KRUEGER, and STEVEN WINGETT: *FastQC*. Babraham Institute, January 2012. <http://www.bioinformatics.babraham.ac.uk/projects/fastqc/>.
- [25] EWELS, PHILIP, MÅNS MAGNUSSON, SVERKER LUNDIN, and MAX KÄLLER: *Multitqc: summarize analysis results for multiple tools and samples in a single report*. Bioinformatics (Oxford, England), 32:3047–8, Oct 2016.
- [26] LI, HENG: *Minimap2: pairwise alignment for nucleotide sequences*. Bioinformatics, 34(18):3094–3100, 05 2018.
- [27] LI, HENG, BOB HANDSAKER, ALEC WYSOKER, TIM FENNELL, JUE RUAN, NILS HOMER, GABOR MARTH, GONCALO ABECASIS, and RICHARD DURBIN: *The sequence alignment/map format and samtools*. Bioinformatics (Oxford, England), 25:2078–9, Aug 2009.
- [28] PEIXOTO, TIAGO P.: *The graph-tool python library*. figshare, 2014.

- [29] COCK, PETER J. A., TIAGO ANTÃO, JEFFREY T. CHANG, BRAD A. CHAPMAN, CYMON J. COX, ANDREW DALKE, IDDO FRIEDBERG, THOMAS HAMELRYCK, FRANK KAUFF, BARTEK WILCZYŃSKI, and MICHEL J. L. DE HOON: *Biopython: freely available Python tools for computational molecular biology and bioinformatics*. *Bioinformatics*, 25(11):1422–1423, 03 2009.
- [30] COCK, PETER J. A., CHRISTOPHER J. FIELDS, NAOHISA GOTO, MICHAEL L. HEUER, and PETER M. RICE: *The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants*. *Nucleic Acids Research*, 38(6):1767–1771, 12 2009.
- [31] EWING, B. and P. GREEN: *Base-calling of automated sequencer traces using phred. ii. error probabilities*. *Genome research*, 8:186–94, Mar 1998.
- [32] PEIXOTO, TIAGO DE PAULA: *graph-tool 2.35 documentation: graph_tool - efficient graph analysis and manipulation*. https://graph-tool.skewed.de/static/doc/graph_tool.html.
- [33] HEGER, ANDREAS and KEVIN JACOBS: *FastQC*. source: <https://github.com/pysam-developers/pysam>, documentation: <https://pysam.readthedocs.io/en/latest/index.html>.
- [34] PAULA PEIXOTO, TIAGO DE: *Mail conversation/graph-tool: problem building weighted graph*, September 2019. source: <https://www.mail-archive.com/graph-tool@skewed.de/msg03486.html>.
- [35] VOSS, W, SAUERBIER T: *Taschenbuch der Statistik*, chapter 8. Kombinatorik, pages 281–281. Fachbuchverlag Leipzig, 2004.
- [36] BRONSTEIN, I. N. and K.A. SEMENDJAJEW: *Taschenbuch der Mathematik, 22. Auflage*, chapter 2.2. Kombinatorik, pages 103–105. Verlag Nauka, Moskau and BSB B.G. Teubner Verlagsgesellschaft, Leipzig, 1985.
- [37] PYTHON SOFTWARE FOUNDATION, VAN ROSSUM, GUIDO: *itertools — functions creating iterators for efficient looping*. <https://docs.python.org/3/library/itertools.html>.
- [38] WITTENBURG, KIM: *Algorithmen und datenstrukturen*, 2017. https://www2.informatik.uni-hamburg.de/fachschaft/wiki/images/3/3e/AD-Skript_%28WS16-17%9.pdf, pages 156-157.
- [39] FOUNDATION, PYTHON SOFTWARE and DANIEL STUTZBACH: *Python — time complexity of various operations*. <https://wiki.python.org/moin/TimeComplexity>.

- [40] DANECEK, PETR, ADAM AUTON, GONCALO ABECASIS, CORNELIS A. ALBERS, ERIC BANKS, MARK A. DEPRISTO, ROBERT E. HANDSAKER, GERTON LUNTER, GABOR T. MARTH, STEPHEN T. SHERRY, GILEAN MCVEAN, and RICHARD DURBIN: *The variant call format and vcftools*. Bioinformatics (Oxford, England), 27:2156–8, Aug 2011.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 3. Januar 2021

Muster Mustermann

