technische universität
dortmund

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

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. Basentripletts 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 Tripletts für die bereits erwähnten 20 Aminosäuren. Somit codieren für die meisten Aminosäuren mehrere verschiedene Tripletts, 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 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 das 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 das 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 Synthese-Richtung 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 war.

Eine ihrer wichtigsten Aufgaben ist die Übertragung der genetische 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 Basentripletts 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 spezifischen 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 Replikationsgabe 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

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. Punktmutationen in Form von Insertionen bzw. Deletionen bewirken durch die zusätzliche bzw. fehlende Base eine Verschiebung des Leserasters, so dass sich die Triplettstruktur für alle nachfolgenden Basen verschiebt. Dies wird als Frame-Shift bezeichnet und verursacht meist eine gravierende Veränderung des resultierenden Proteins, da viele der nachfolgenden Tripletts 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. Auch bei Insertionen und Deletionen von mehreren Basen kann es zu einem Frame-Shift kommen, sind allerdings drei Basen oder ein Vielfaches hiervon betroffen, so bleibt das Leseraster erhalten.

Bei Substitutionen bleibt dagegen das Leseraster erhalten. Aufgrund der Degeneration des genetischen Codes können verschiedene Basentripletts 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 (readtrough-Mutation).

Hinsichtlich der Eigenschaften des resultierenden Proteins unterscheidet 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 nichtcodierenden 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-Sequenzing-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 SNPs. 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 modi-

fiziert.

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 beigefügt. 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

1.4 Sequenzierungsdaten und -formate

1.4.1 Reads

Identifikationsbezeichnungen, der Basensequenz, Basenqualität

1.4.2 Sequenzalignments

Häufig werden für die Analyse von Sequenzierungsdaten sogenannte Alignments erstellt, bei denen eine Readsequenz (Query) gegen eine andere Sequenz (Reference) verglichen wird. In Abhängigkeit von ihren Übereinstimmungen (Matches) und Unterschieden (Mismatches) werden die Sequenzen einander zugeordnet. Ein solches Alignment im SAM/BAM-Format wird auch in der vorliegenden Arbeit zur weiterführenden Analyse verwendet.

Das SAM/BAM-Format [27] enthält unter anderem die ID's der Query- und Reference-Sequenzen, Informationen zur Basenqualität, Readlänge und -sequenz, den sogenannten CIGAR-String sowie verschiedene optionale Tags, wie beispielsweise den NM-Tag, der die Edit-Distanz angibt (vgl. Kap. 2.3.1). Die Edit-Distanz ist die mindestens notwendige Anzahl von Ersetzungs-, Einfügungs- und Löschungsoperationen, um die Sequenz des Source-Knotens in die Sequenz des Target-Knotens zu transformieren. Auf DNA-Ebene entspricht dies den Punktmutationen im Sinne von Substitutionen, Insertionen und Deletionen (vgl. Kap. 1.1.5). Der CIGAR-String hingegen ist eine kondensierte Darstellung der Unterschiede zwischen Query- und Reference-Sequenz. In im SAM-Format werden darin Matches mit M oder $=$, Mismatches mit X oder S, Insertionen mit I und Deletionen mit D sowie jeweils mit der Anzahl der betroffenen Basen codiert (siehe auch Kap. 1.1.5). Aus ihrer Summe mit Ausnahme und abzüglich der Deletionen ergibt die Readlänge. So ergibt beispielsweise der CIGAR-String $69 = 1X24 =$ ein Matching der ersten 69 Basen des Queryreads beim Abgleich mit dem Referenzread, anschließend ist bei einer Base ein Mismatch aufgetreten und schließlich folgen 24 Basen die auf den Referenzread matchen. Die Readlänge beträgt somit insgesamt 94 Basen und die Edit-Distanz im NM-Tag besitzt den Wert 1.

Daneben gibt es weitere Darstellungsmöglichkeiten von Veränderungen der Querysequenz gegenüber der Referenzsequenz. So können im cs-Tag des SAM-Formats verkürzt

(short) oder vollständig mit Angabe der gesamten Readsequenz die genauen Veränderungen mit Bezeichnung der betreffenden Basen angegeben werden. Bezüglich des oben genannten Beispiels kann der dazugehörige short cs-Tag $69 * tc : 24$ lauten und zeigt damit an, dass nach Base 69 ein Basenaustausch von Thymin gegen Cytosin erfolgt ist.

1.4.3 Varianten

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 unter Berücksichtigung der Ploidie ohne weiteres Vorwissen allein auf Grundlage der Mutationsraten und Heterozygotiewahrscheinlichkeiten erfolgen. Die dabei ermittelten Likelihoods stehen zudem für anschließende Diversitätsanalysen zur Verfügung.

2.2 Formale Definition der Problemstellung

Gegeben sei eine Menge von Reads eines Individuums $D = (s_1, \ldots, s_m) \in \{A, C, G, T, \}^{k^m}$ mit der Readlänge k. In den Reads sind zudem für jede Base Informationen zur Qualität der Sequenzierung $Q = (q_1, \ldots, q_m) \in [0, 1]^{k^m}$ enthalten. Weiterhin seien für Substitutionen,

Insertionen und Deletionen Angaben zu den Mutationsraten $M = (m_{sub}, m_{ins}, m_{del})$ und Heterozygotiewahrscheinlichkeiten $\eta = (\eta_{sub}, \eta_{ins}, \eta_{del})$ gegeben. Ebenso ist die Ploidie ϕ des untersuchten Organismus bekannt.

Ziel ist die Zuordnung der Reads zu den Loci unter Berücksichtigung von M und η sowie die Ausgabe der Menge der ermittelten Loci mit den Sequenzen der in ihnen enthaltenen Allele. Dabei soll die Menge von Loci gefunden werden, welche die beobachteten Reads am besten erklären kann, das heißt welche die höchste Wahrscheinlichkeit in Zusammenschau mit M und η besitzt.

2.3 Lösungsansatz

Das Problem wird als gerichteter Graph betrachtet (siehe Kap. 2.3.1), dessen Knoten die einzelnen Reads beinhaltet und dessen Kanten die Wahrscheinlichkeiten repräsentieren, dass der Read des Knotens, von dem die Kante ausgeht (Source-Knoten), vom dem Read stammt, auf den die Kanten gerichtet ist (Target-Knoten).

Für die Zuordnung der Reads soll das Problem in Teilprobleme aufgeteilt werden, für jedes Teilproblem soll dann eine Lösung ermittelt werden. Der Graph wird daher entsprechend seiner Zusammenhangskomponenten partitioniert. Für jede Zusammenhangskomponente werden alle Reads mit einander verglichen und diejenigen Allele identifiziert, von denen die übrigen Reads der Komponente am wahrscheinlichsten durch Sequenzierfehler entstanden sind (Kap. 2.3.4). Hierzu werden Basenqualität und Mutatationsraten berücksichtigt, so dass die Sequenzierfehlerrate die Likelihoodberechnung bestimmt.

Die Allelkombination mit der höchsten Wahrscheinlichkeit soll anschließend für die Zuordnung der Loci verwendet werden (Kap. 2.3.5). Auch hierfür wird die Likelihood für alle Loci-Kombinationen bestimmt, wobei nun die Heterozygotiewahrscheinlichkeiten einbezogen werden. Die Loci-Kombination mit der größten Likelihood wird schließlich als Lösung der betreffenden Zusammenhangskomponente ausgegeben (Kap. 2.3.6).

Für jede Zusammenhangskomponente wird auf diese Weise also die wahrscheinlichste Lösung ermittelt, deren Komposition von Loci die beobachteten Reads innerhalb der Zusammenhangskomponente am besten erklären kann.

2.3.1 Graph und Zusammenhangskomponenten

Im Preprocessing wird aus den Reads zunächst ein Alignment mit Hilfe des Tools Minimap2 erzeugt [26]. Hierbei werden alle Reads mit einander verglichen und bei ausreichender Ähnlichkeit ihrer Sequenzen zu Paaren zusammengefasst. Zudem werden durch Minimap2 die Edit-Distanzen zwischen den jeweils verglichenen Readsequenzen bestimmt.

Wie bereits einleitend erwähnt, wird das Problem als gerichteter Graph $G = (V, E)$ mit der Knotenmenge V und der Kantenmenge E betrachtet. Die Knoten repräsentieren dabei die einzelnen Reads. Entsprechend dem von Minimap2 erzeugtem Sequenzalignment werden die Knoten der Reads durch Kanten verbunden. Die Kanten des Graphen repräsentieren also den Vergleich der Sequenzen der beiden angrenzenden Reads.

Initial basieren die Kantengewichtungen auf den durch Minimap2 bestimmten Edit-Distanzen zwischen den Reads. Um die Anzahl der Kanten im Graph in einem überschaubaren Rahmen zu halten, sollen nur Kanten eingefügt werden, die einen festgelegten Schwellenwert hinsichtlich der Edit-Distanz nicht überschreiten. Das Problem wird durch diesen Filtervorgang verkleinert, so dass die eigentliche Likelihoodberechnung der paarweisen Vergleiche der Reads über ein pair hidden Markov Model (Kap. 2.3.2) effizienter erfolgen kann.

Da nur Reads mit einander verbunden werden, deren Sequenzen also eine gewisse Ähnlichkeit zu einander aufweisen, ist der Graph in Abhängigkeit vom festgelegten Schwellenwert nicht zusammenhängend und besitzt mehrere Zusammenhangskomponenten. Dies ermöglicht eine Unterteilung des Gesamtproblems in kleinere Teilprobleme, indem für die einzelnen Zusammenhangskomponenten des Graphen jeweils Lösungen ermittelt werden. Da der Grenzwert ein gewählter und konfigurierbarer Wert ist, entsprechen die so entstandenen Zusammenhangskomponenten nicht unbedingt Clustern von Reads die nur einem einzelnen Locus zuzuordnen sind. Vielmehr können die Zusammenhangskomponenten auch jeweils mehrere Loci beinhalten.

2.3.2 Pair hidden Markov Model

========== draft begin ==========

-> Berücksichtigung der Basenqualität q_r => Sequenzierfehler fließen mit geringerer liklihood 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
paarweises Alignment aus Minimap2 als Teilproblem -> fast jede Beobachtung wird berücksichtigt
Bewertung/wslk ob ein read aus einem anderen read entstanden ist

========== draft end ==========

2.3.3 Approximation von PairHMM mittels Minimap2-Alignments

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 $pairHMM_{\epsilon,q_{query}}\ (s_{ref} \mid s_{query})$, 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}.

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

Jede Base b an Position i einer Readsequenz s der Länge k lässt sich also definieren als $b \in \{\, b_i \in \{A, C, G, T\}^k\ ,\ b_i \in s \mid i = 1, \cdots, k \,\}$. 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:

$$L_{i\,match} = 1 - p_{i\,query} \tag{3-3}$$

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:

$$L_{i\,mut} = m_{rate} \cdot\ (1 - p_{i\,query}) \tag{3-4}$$

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:

$$L_{i\,seqerr} = \frac{1}{3} \cdot\ p_{i\,query} \tag{3-5}$$

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

$$L_{i\,mismatch} = (1 - m_{rate}) \cdot\ L_{seqerr} \cdot\ L_{mut} \tag{3-6}$$

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

Das Sequenzalignment der Reads aus Minimap2 repräsentiert bereits den wahrscheinlichsten Pfad in der pairHMM-Matrix, so dass über diesen Pfad die Likelihoods der Allele (Kap. 2.3.4) und der Loci (Kap. 2.3.5) bestimmt werden können.

2.3.4 Allele-Fractions mit maximaler Likelihood

Um die Allele zu identifizieren, von denen die beobachteten Reads einer Zusammenhangskomponente am wahrscheinlichsten stammen, muss zunächst die Menge der Kandidatenallele $A = (a_1, \ldots, a_n) \in \{A, C, G, T, \}^{k^n}$ bestimmt werden. Bei größeren Clustern ist davon auszugehen, dass die Sequenzen solcher Kandidatenallele mehrfach in den Reads auftauchen. Um den Aufwand der Likelihoodberechnung bei größeren Cluster anpassen zu können, soll es möglich sein, selten auftretende Sequenzen herauszufiltern. Dies geschieht über zwei Grenzwerte, welche die Mindestgröße des Clusters und die Mindesthäufigkeit von Kandidatenallelsequenzen festlegen. Durch sie werden ab einer bestimmten Clustergröße nur diejenigen Sequenzen in die Auswahl der Kandidatenallele aufgenommen, die mit einer bestimmten Häufigkeit im Cluster vorkommen.

Anschließend wird die Wahrscheinlichkeit bestimmt, dass ein Read mit der Readsequenz mit der Fehlerrate ϵ tatsächlich von einem bestimmten Allel stammt. Die Fehlerrate ϵ ergibt sich dabei aus der Qualität der einzelnen Basen eines Reads. Jede Base der Sequenz des Source-Knotens wird mit der Sequenz des Target-Knotens verglichen. Bei einem sogenannten Mismatch unterscheiden sich die verglichenen Basen durch Substitutionen, Insertionen oder Deletionen. Im Falle eines Matches sind beide Basen identisch. Bei einem Match ergibt sich die Wahrscheinlichkeit L_{match}, dass es sich dabei im Source-Knoten tatsächlich um die korrekte Base handelt allein aus der Fehlerrate:

$$L_{match} = 1 - \epsilon \tag{2-1}$$

Bei einem Mismatch $L_{mismatch}$ müssen die Wahrscheinlichkeiten für Sequenzierfehler $L_{sequerr}$ (vgl. Formel (3-5)) und Mutationen L_{mut} (vgl. Formel (3-4)) sowie die Mutationsrate selbst m_{rate} mit einbezogen werden:

$$L_{mismatch} = (1 - m_{rate}) \cdot L_{sequerr} \cdot L_{mut} \tag{2-2}$$

Die genaue Berechnung der Likelihood der einzelnen Basen wird an späterer Stelle in Kap. 3.3.2 ausführlicher besprochen. Das Produkt aller Wahrscheinlichkeiten der einzelnen Basen ergibt schließlich die Likelihood für den paarweisen Vergleich zweier Reads im Sinne des pairHMM. Bei paarweisen Vergleich eines Allels $a_i \in A$ mit einem Read mit der Sequenz s_r lässt sich die Likelihoodberechnung also wie folgt formulieren:

$$Pr(T = s_r \,|\, S = a_i, \epsilon) = pairHMM_{\epsilon,q_r}(a_i, s_r) \tag{2-3}$$

========== draft ==========

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 $A = (a_1, \ldots, a_n)$ mit Anzahl n und ϵ Sequenzierfehlerrate

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

2. Info über alle vafs aller Kandidatenallele für 1 read; Zusammenhangskomponenten kann auch >1 locus enthalten -> 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 => Urnenmodell (entspricht binominal model)
Urnenmodell: bsp. 2 farben in unterschiedl. Menge -> 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 => Mulitnomialverteilung Θ_i + individuellen Verteilung Pr: $\Theta_i * Pr$ es gilt $\sum \Theta_i = 1$ => 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
$\Theta = \theta_1, \ldots, \theta_n) \in [0, 1]^n$ sind Allel-Fraktionen, n ist Anzahl der Allele

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

3. Für alle Reads:
1 read s_r mit n candidaten allelen -> n mögliche Allele von denen s_r stammen kann -> 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 => zu 25% allel i und zu 75% allel i+1 -> über max lh wird bestimmt, wie die seq der allele zusammenpassen -> modellierung ähnlicher, unsicherheit bei der zuordung der reads, wenn sie weder der dem eine noch dem anderen allel entstammen, z.b. bei 0.5/0.5 => Unsicherheit wird an schritt 3 weitergegeben
Produkt über alle reads für jede mögliche Kombination an fractions
max wählen innerhalb der conn. comp
insgesamt:
=> Θ berechnen -> max lh aus allel-fractions
=> alles was an Unterschieden vorkommt, kann nur ein Sequenzierfehler sein
m ist Anzahl der Reads, $L \in \{l_i \in \mathbb{N}^{\phi}_{\leq n} \,|\, i = 1, \ldots, g\}$ ist Menge der Loci, $D = (s_1, \ldots, s_m) \in \{A, C, G, T\}^{k^m}$ ist Menge der Readsequenzen/der Reads,

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

für jede vaf wurde also liklihood über alle reads bestimmt -> Maximum

2.3.5 Locuszuordnung mit maximaler Likelihood

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 einfachsen möglichen lösung
-> ploidy als eingabeparam -> zuordnung der candidaten 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_allel_fractionen 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 $A = (a_1, \ldots, a_n)$ in einer möglichen locus-verteilung $l = (l_1, \ldots, l_{g \cdot \phi})$ jeweils genau ihrer absolu-

ten Häufigkeit aus der in Kap. 2.3.4 errechneten vaf mit maximaler Likelihood entspricht, also $\theta_i \cdot g \cdot \phi$ entsprechen, andernfalls gilt $z_l = 0$. Seien dabei $\Theta = \theta_1, \ldots, \theta_n) \in [0,1]^n$ die vaf mit maximaler Likelihood, ϕ die Ploidie und gilt für eine mögliche Locusverteilung l außerdem $L \in \{l_j \in \mathbb{N}^{\phi}_{\leq n} \,|\, j = 1, \ldots, g\}$, so lässt sich die Indikatorfunktion wie folgt definieren:

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

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

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

Kombi von loci -> lh berechnen

$$Pr(\Theta, A \,|\, L = \{l_j \,|\, j = 1, \ldots, g\}) = z_l \cdot \prod_{j=1}^{g} Pr(T = a_{l_{j,2}} \,|\, S = a_{l_{j,1}}) \tag{2-xxx6}$$

=> maximum der loci Likelihoods wählen

2.3.6 Varianten und Genotyp

mit den allelen $A = (a_1, \ldots, a_n)$ jedes locus $L \in \{l_j \in \mathbb{N}^{\phi}_{\leq n} \,|\, j = 1, \ldots, g\}$, allelsequenz, genotyp
Menge von Loci, die die Beobachtungen erklären

Kapitel 3

Implementierung

Für das in Python 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 (siehe Kap. 1.4.1), 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 Probennamen zu den Individuen und ihren Barcode-Sequenzen definiert ist. Nach dem Preprocessing, der Qualitätskontrolle der Reads und dem Sequenzalignment erfolgt die RAD-Seq-Analyse durch NodeRAD. Hierbei werden die Wahrscheinlichkeiten der Allele-Fractions und der möglichen Loci bestimmt. Die Loci mit der höchsten Wahrscheinlichkeit werden schließlich mit den Sequenzen ihrer Allele 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

überrepräsentierten Sequenzen. Durch das Tool MultiQC [25] wird aus diesen Statistiken und den Log-Dateien von Cutadapt ein Report im HTML-Format 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 (vgl. Kap. 1.4.2) mit Hilfe des Tools Minimap2 [26] erstellt. Hierbei werden alle Readsquenzen paarweise mit einander verglichen. Das Ergebnis des Mappings wird im SAM/BAM-Format [27] ausgegeben und enthält neben den Angaben zur Query- und Referenzsequenz auch den CIGAR-String sowie den NM-Tag mit den Edit-Distanzen. Der CIGAR-String und der NM-Tag definieren wichtige Kanteneigenschaften des späteren Graphen.

3.3 Konstruktion des Graphen

3.3.1 Knoten des Graphen

NodeRAD benötigt als Input zu jedem Individuum die getrimmten single-end Reads 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. Zu den Konstanten gehören die verschiedenen Mutationsraten und Heterozygotiewahrscheinlichkeiten für Substitutionen, Insertionen und Deletionen, die Ploidie des Chromosomensatzes der untersuchten Spezies. Diese werden im Script `noderad_main.py` im Graphen als Grapheigenschaften abgelegt. Als konfigurierbare Grenzwerte gibt es für NodeRAD einen Schwellenwert `edit-distance-graph` für die maximal zulässige Edit-Distanz, bei der zwei Knoten noch durch eine Kante verbunden werden sowie Schwellenwerte zum Filtern selten vorkommender Sequenzen (`treshold-seq-noise`) ab einer bestimmten Clustergröße (`treshold-cluster-size`), 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, decodiert [30]. Für jeden Knoten werden die Vektoren mit den Phred Qualitiy Scores der Basen als Knoteneigenschaft gespeichert.

3.3.2 Kanten des Graphen

Die Kanten des Graphen definieren sich durch die mittels Minimap2 erzeugten Sequenzalignments (vgl. Kap. 3.2). Jedes Alignment zweier 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/BAM-Formats des Alignmentfiles erfolgt mit Hilfe der Python-Library pysam [33]. Dabei werden die Edit-Distanzen aus dem NM-Tag genutzt, um nur Kanten in den Graphen aufzunehmen, die unterhalb des durch die Konfigurationsdatei festgelegten Grenzwertes `threshold_max_edit_distance` liegen. Neben den Edit-Distanzen wird als weitere Kanteneigenschaft der zu Tupeln decodierte CIGAR-String hinzugefügt. Zusätzlich kann zur Kontrolle oder für eine spätere Verwendung auch der cs-Tag (Kap. 1.4.2) 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 (Kap. 1.4.2) 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 an. 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. Die CIGAR-Tupel werden für die Berechnung der Likelihood zwischen zwei Knoten benötigt (Kap. 2.3.3). Dies erfolgt zum einen zur Bestimmung der Likelihood der Allele-Fractions (Kap. 3.4.2) und zum anderen, um die Likelihood der Loci-Kombinationen zu ermitteln.

Nach Abschluss der Graphkonstruktion werden für jedes Individuum Anzahl der Knoten und Kanten des Graphen in den Log-Dateien festgehalten. 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.

3.3.3 Bestimmung der Zusammenhangskomponenten

Die Bestimmung der Zusammenhangskomponenten erfolgt durch graph-tool [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.

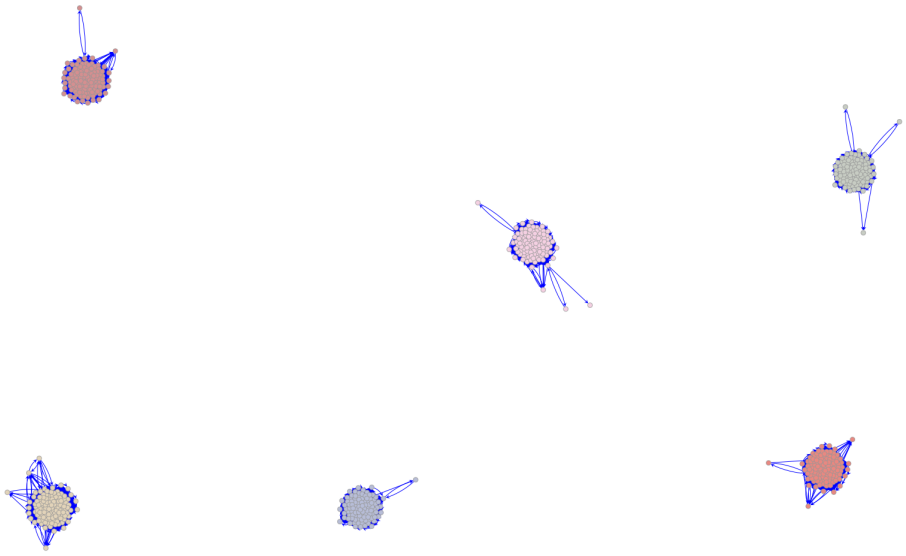

Abbildung 3.1: Ausschnitt einiger Zusammenhangskomponenten des Gesamtgraphen

Da alle weiteren Schritte des Algorithmus jeweils auf den einzelnen Komponenten durchgeführt werden, kann durch die Verwendung einer Liste von Graphen im Folgenden eine einfachen Iteration über die Komponenten ausgeführt werden, ohne dass der Filtervorgang 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.

In der Log-Datei wird die Anzahl der Knoten aller Zusammenhangskomponenten als Histogramm festgehalten. Ebenso wird dort für alle Komponenten mit mehr als einem Element die Anzahl ihrer Knoten, Kanten und Eigenschaften aufgelistet.

Über die Konfigurationsdatei und die Snakemake-Regel `noderad` können optional auch für die Zusammenhangskomponenten jeweils Visualisierungen (siehe Abbildung 3.2) und detaillierte Graphinformationen in den oben genannten Formaten des Graphen (Kap. 3.3.2) ausgegeben werden.

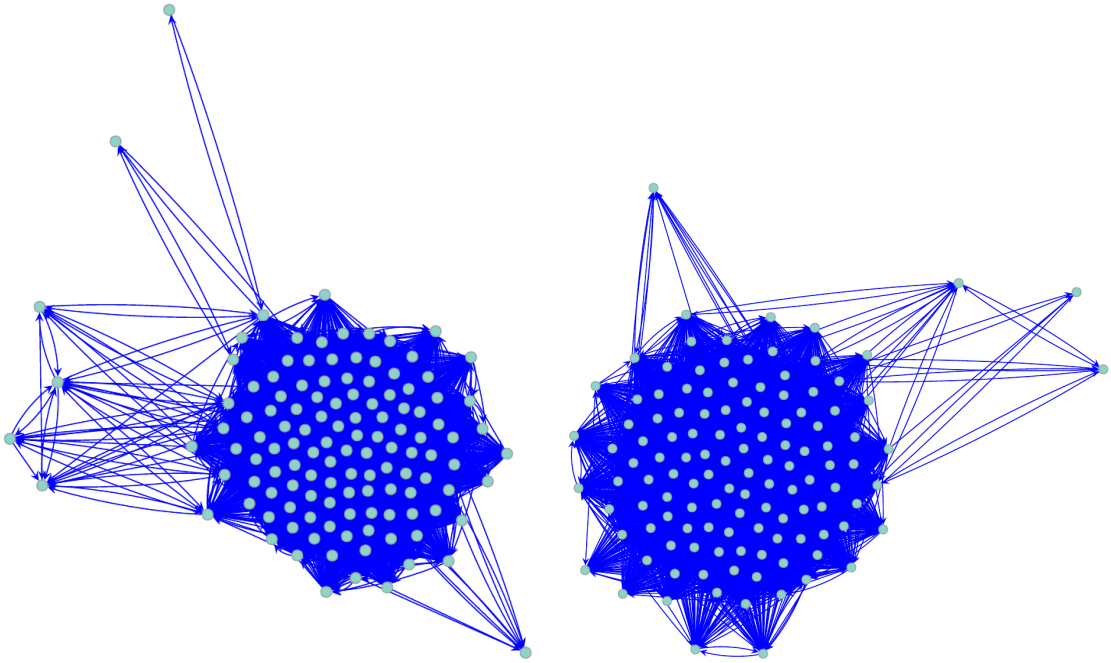

Abbildung 3.2: Beispiele einzelner Zusammenhangskomponenten

Zudem kann optional auch der gesamte Graph mit den Komponentenindizes als Knoteneigenschaften gespeichert werden. In der visuellen Darstellung werden dort seine Knoten entsprechend der zugehörigen Zusammenhangskomponente eingefärbt (siehe Abbildung 3.1).

3.4 Bestimmung der Allele-Fractions mit maximaler Likelihood

3.4.1 Bestimmung der Kandidatenallele und ihrer möglichen Häufigkeitsverteilung

Die einzelnen Zusammenhangskomponenten repräsentieren einen oder mehrere Loci. Alle weiteren Berechnungen aus diesem und den folgenden Kapiteln 3.5 und 3.6 werden für jede Komponente einzeln durchgeführt, dabei sollen die wahrscheinlichsten Allelsequenzen und die dazugehörigen Loci identifiziert werden. Die hierfür implementierten Funktionen

finden sich im Modul `likelihood_operations.py`.

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 als lexikographisch sortierte Liste erzeugt. Übersteigt die Größe eines Clusters, d.h. die Knotenanzahl einer Zusammenhangskomponenten, den in der Konfigurationsdatei als `treshold-cluster-size` festgelegten Grenzwert, so wird von allen vorkommenden Sequenzen des Clusters zunächst die absolute Häufigkeit bestimmt. Anschließend werden nur diejenigen Sequenzen lexikographisch sortiert zurückgegeben, die den `treshold-seq-noise` Schwellenwert aus der Konfigurationsdatei 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 einstellbaren Clustergröße durchgeführt werden.

Aus der so erzeugten Liste lexikographisch sortierter Kandidatenallele A_{cand} der Länge n_{cand} 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. Dies 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 Kandidatenallele, 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_{cand}, so können alle beobachteten Allele auch tatsächlich vorkommen, da die Zusammenhangskomponente auch mehrere Loci enthalten kann. Es gilt dann also $n_{alleles} = n_{cand}$. 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 und es gilt $n_{alleles} = n_{cand} + \phi - (n_{cand} \mod \phi)$.

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`). Diese Häufigkeitsverteilungen werden auch als Allele-Fractions bezeichnet. Hierfür werden zunächst alle möglichen Allelkombinationen ermittelt. Dies erfolgt nach dem Urnenmodell unter Auswahl von $n_{alleles}$ Elementen mit Zurücklegen aus insgesamt n_{cand} 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].

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

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

Beispiele möglicher Allelkombinationen:

$ploidy = 2, n_{cand} = 2, n_{alleles} = 2$:
[(0, 0), (0, 1), (1, 1)]

$ploidy = 2, n_{cand} = 3, n_{alleles} = 4$:
[(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 a_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.

$$h_{a_i} = \frac{H_{a_i}}{n_{alleles}} \tag{3-13}$$

Beispiele möglicher Allele-Fractions:

$ploidy = 2, n_{cand} = 2, n_{alleles} = 2$:
[1.0, 0.0], [0.5, 0.5], [0.0, 1.0]

$ploidy = 2, n_{cand} = 3, n_{alleles} = 4$:
[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]

3.4.2 Berechnung der Likelihoods der Allele anhand der Allele-Fractions

In diesem Kapitel soll diejenige Allel-Fraction bestimmt werden, welche die beobachteten Reads am besten erklärt (vgl. Kap. 2.3.4). Für jeden Read wird dafür zunächst die Wahrscheinlichkeit bestimmt, dass der Read durch Sequenzierfehler aus einem der Kandidatenallele entstanden ist. Hiernach wird für jeden Read die Wahrscheinlichkeit errechnet, diesen anhand einer gegebenen Allele-Fraction zu beobachten. Diese Berechnungen werden für alle Allele-Fractions durchgeführt. Aus der so gewonnenen Lösungsmenge wird die Allel-Fraction mit der höchsten Likelihood für die spätere Zuordnung der Loci (Kap. 3.5) ausgewählt.

Zunächst wird in der Funktion `get_allele_likelihood_read()` eine Kante zwischen dem betrachteten Read und dem Kandidatenallel gesucht. Während der Ausführung werden die Sequenzen und Likelihoods der bereits gefundenen Read-Kandidatenallel-Paare in einem Dictionary abgelegt. Als Key wird dabei das Tupel aus Read- und Allelsequenz verwendet, die berechnete Likelihood wird als Value eingetragen. Dies ermöglicht eine schnellere und effizientere Suche einer passenden Kante.

Existiert ein Read-Kandidatenallel-Paar noch kein Eintrag im Dictionary, so werden zuerst alle ausgehenden Nachbarn des Reads hinsichtlich ihrer in den Knoteneigenschaften abgelegten Sequenz geprüft. Besitzt einer der ausgehenden Nachbarknoten die Sequenz des Kandidatenallels, so wird die Likelihood durch `get_alignment_likelihood()` berechnet (Algorithmus 3.2), ein entsprechender Eintrag im Dictionary angelegt und die Likelihood zurückgegeben.

Gibt es keine solche Kante, die vom Read zu einem Knoten mit Kandidatenallelsequenz führt, so wird geprüft, ob es eine Kante in umgekehrter Richtung gibt. Hierfür werden nun alle eingehenden Nachbarn des Reads betrachtet. Findet sich unter ihnen ein Knoten mit der gesuchten Kandidatenallelsequenz, so wird mit `get_alignment_likelihood()` die Likelihood berechnet, wobei die Verwendung der Gegenrichtung durch das Argument `reverse=True` markiert wird. Existiert auch keine entgegengesetzt gerichteten Kante, so wird der Wert 0 zurückgegeben.

Zur Veranschaulichung ist der beschriebene Algorithmus zudem in Pseudocode dargestellt (Algorithmus 3.1). Dabei sind $C_k \in \{C_1, \ldots, C_p\}$ der Subgraph einer Zusammenhangskomponente, $r_i \in C_k$ der Knoten des Reads mit der Sequenz s_{r_i} und $a_j \in A_{cand}$ ein Kandidatenallel mit der Sequenz s_{a_j}, wobei $k, i, j \in \mathbb{N}$. Das oben beschriebene Dictionary *dict* besitzt als Keys die Tupel der Sequenzen (s_{r_i}, s_{a_i}) und als Values die Likelihoods L_{r_i,a_i}, so dass gilt $((s_{r_i}, s_{a_i}), L_{r_i,a_i}) \in dict$. Die Verwendung fester Knoten- oder Kantenei-

genschaften ist durch eckige Klammern gekennzeichnet, so ruft beispielsweise $r_i[q\_values]$ die q-Werte des Phred Quality Scores von r_i auf.

Algorithmus 3.1 Suche eines Alignments zwischen Read und Kandidatenallel

```
1: function GET_ALLELE_LIKELIHOOD_READ(C_k, r_i, s_{a_j}, dict)
2:     s_{r_i} ← r_i[sequence]
3:     m_rates ← C_k[m_sub] ∪ C_k[m_ins] ∪ C_k[m_del]
4:     qual ← r_i[q_values]
5:     if ∃(s_{r_i}, s_{a_i}) : ((s_{r_i}, s_{a_i}), L_{r_i,a_i}) ∈ dict then
6:         return L_{r_i,a_i}
7:     end if
8:     out_neighbors ← get_out_neighbors(r_i)
9:     for out_neighbor ∈ out_neighbors do
10:        if s_{a_j} = out_neighbor[sequence] then
11:            cig ← edge(r_i, out_neighbor)[cigar_tuples]
12:            rev ← False
13:            L_{r_i,a_i} ← get_alignment_likelihood(m_rates, cig, qual, rev)   ▷Alg. 3.2
14:            dict ← dict ∪ ((r_i, out_neighbor[sequence]), L_{r_i,a_i})
15:            return L_{r_i,a_i}
16:        end if
17:    end for
18:    in_neighbors ← get_in_neighbors(r_i)
19:    for in_neighbor ∈ in_neighbors do
20:        if s_{a_j} = in_neighbor[sequence] then
21:            cig ← edge(in_neighbor, r_i)[cigar_tuples]
22:            rev ← True
23:            L_{r_i,a_i} ← get_alignment_likelihood(m_rates, cig, qual, rev)   ▷Alg. 3.2
24:            dict ← dict ∪ ((r_i, in_neighbor[sequence]), L_{r_i,a_i})
25:            return L_{r_i,a_i}
26:        end if
27:    end for
28:    return 0
29: end function
```

Die eigentliche Likelihoodberechnung erfolgt in der Methode `get_alignment_likelihood()` (Algorithmus 3.2) nach Formel (2-3) des Models. Wie in Kap. 2.3.3 beschrieben wird hierfür die Approximation des PairHMM aus den Sequenzalignments von Minimap2 verwendet.

Von der in `get_allele_likelihood_read()` ermittelten Kante zwischen Read und Kandidatenallel werden der Funktion die CIGAR-Tupel aus den Kanteneigenschaften, der Vektor mit dem Phred Quality Scores $Q = (q_1, \ldots, q_m)$ aus den Knoteneigenschaften (siehe Kap. 3.3.1) sowie die Kantenrichtung als boolescher Wert übergeben. Aus den Phred Quality Scores des Reads wird zunächst die geschätzte Fehlerwahrscheinlichkeit $P = (p_1, \ldots, p_m)$ für jede Base nach Formel (3-1) errechnet [31].

$$p_i = 10^{\frac{-q_i}{10}} \tag{3-1}$$

Im Anschluss wird aus den p-Werten der Basenqualität für jede Base des Reads die Wahrscheinlichkeit errechnet, dass es sich im Falle eines Matches um die korrekte Base handelt (3-3) bzw. im Falle eines Mismatches, dass es sich um einen Sequenzierfehler (3-5) oder um eine Mutation handelt (3-4). Entsprechend den in den CIGAR-Tupeln angegebenen Operationen wird für die betreffenden Basen die Likelihood nach (3-6) berechnet. Das Produkt der Likelihoods der einzelnen Basen ergibt schließlich die Likelihood zwischen Read und Allel (3-2).

Bei Bedarf kann die Methode `get_alignment_likelihood()` auch die Likelihood der entgegengesetzt gerichteten Kante bestimmen. Dabei wird die Querysequenz als Referenzsequenz betrachtet und umgekehrt. Dies ist über das boolesche Argument `reverse` steuerbar. Gilt `reverse = True`, so werden für die übergebenen CIGAR-Tupel Insertionen zu Deletionen und Deletionen zu Insertionen umgewandelt.

Zur Veranschaulichung ist die Methode in Algorithmus 3.2 zusätzlich als Pseudocode verfasst. Die Mutationsraten für Substitutionen, Insertionen bzw. Deletionen aus den Grapheigenschaften werden dort mit m_{sub}, m_{ins} bzw. m_{del} bezeichnet, der Vektor der Phred Quality Scores der Querysequenz mit q_query

Algorithmus 3.2 Berechnung der Likelihood zwischen Read und Kandidatenallel

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

Durch `get_allele_likelihood_read()` und `get_alignment_likelihood()` wurden also die Wahrscheinlichkeiten jedes Reads bestimmt, dass er von den verschiedenen Allelen der Zusammenhangskomponente stammt (siehe oben). Die Wahrscheinlichkeiten im Bezug auf jedes Allel sollen nun in die zuvor bestimmten Allele-Fractions (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 Allel-Fraction zu beobachten. Dazu wird über die relativen Häufigkeiten jedes Allels der Allele-Fraction iteriert und ihr Produkt mit der Readlikelihood aufsummiert.

Aus dem Produkt über alle Reads wird anschließend für die Allele-Fraction nach Formel (2-xxx3) in `calc_vafs_likelihood()` die Gesamtwahrscheinlichkeit errechnet.

Auf diese Weise wird in `noderad_main.py` für jede Allel-Fraction die Gesamtwahrscheinlichkeit über alle Reads und alle relativen Häufigkeiten der Allele berechnet und in einer Liste abgelegt.

Aus dieser Liste wird die Allele-Fraction 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.

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

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 Allelfraktion 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_{cand} 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 jeweiligen 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 Permuationen werden über jede Allelkombination in ihrer vollständigen Länge gebildet, so dass gilt $n = k$, folglich werden insgesamt $n!$ Elemente erzeugt (3-28).

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

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 *phi*-Tupel der Anzahl möglicher Loci entspricht. Mehrere Loci in einer Zusammenhangskomponenten sind nach `get_max_parsimony_n_alleles()` genau dann möglich, wenn gilt $n_{cand} > \phi$.

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:

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

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))

3.5.2 Zuordnung der Allele-Fractions 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_likelihoods()` 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_constrait()` 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 Grapheigenschaften zur Ermittlung der Likelihood nach Formel (2-xxx5) genutzt.

Hierzu werden zunächst in `get_allele_likelihood_alleles()` (Algorithmus 3.3) 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 Funktion `get_cigar_tuples()` ermittelt und für die Likelihoodberechnung in `get_heterozygosity()` verwendet (siehe Algorithmus 3.4). Die Gesamtwahrscheinlichkeit errechnet sich aus dem Produkt der Likelihood aller Allelpaare nach 2-xxx6.

Algorithmus 3.3 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. 3.5*
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. 3.4*
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 3.2 wird auch hier die Likelihood über aller Matches und Mismatches berechnet.

Algorithmus 3.4 Bestimmung der Likelihood zwischen zwei Allelen hinsichtlich der Heterozygotiewahrscheinlichkeiten

```
1: function GET_HETEROZYGOSITY(η_sub, η_ins, η_del, cigar_tuples, reverse)
2:     likelihood ← 1.0
3:     if reverse then
4:         swap values of η_ins and η_del
5:     end if
6:     η_all ← η_sub + η_ins + η_del
7:     for all (operation, length) ∈ cigar_tuples do
8:         if operation ∈ match then
9:             likelihood ← likelihood · (1 − η_all)^length
10:        end if
11:        if operation ∈ mismatch then
12:            η_factor ← 1.0
13:            if operation ∈ substitution then
14:                η_factor ← η_sub
15:            end if
16:            if operation ∈ insertion then
17:                η_factor ← η_ins
18:            end if
19:            if operation ∈ deletion then
20:                η_factor ← η_del
21:            end if
22:            likelihood ← likelihood · (η_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).

$$L_{i\,mismatch} = \eta_{rate} \tag{3-30}$$

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)).

$$L_{i\,match} = 1 - (\eta_{sub} + \eta_{ins} + \eta_{del}) \tag{3-31}$$

Die Likelihood $Pr(T = a_{l_{j,2}} \mid S = a_{l_{j,1}}, \eta)$ 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:

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

Wie bereits erwähnt, 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 $R_{source} = (v_1, \ldots, v_i)$, welche die Query-Sequenz s_{source} besitzen sowie eine Liste aller Knoten $R_{target} = (w_1, \ldots, w_j)$, 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 boolsche Wert entsprechend der Kantenrichtung zurückgegeben.

Zur Veranschaulichung ist der Algorithmus in 3.5 zudem als Pseudocode dargestellt. Dabei sind $C_k \in \{C_1, \ldots, C_p\}$ 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 3.5 CIGAR-Tupel bestimmen

```
1: function GET_CIGAR_TUPLES(C_k, s_query, s_ref)
2:     R_source ← {v_i ∈ C_k ∧ i, k ∈ ℕ | v_i[sequence] = s_query}
3:     R_target ← {w_j ∈ C_k ∧ j, k ∈ ℕ | w_j[sequence] = s_ref}
4:     for v_i ∈ R_source do
5:         for w_j ∈ 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
```

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.

3.6 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 Bennennung 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 sortiere 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 Eintag 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 Probenspalte 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 Locus aus Loc_{max} die entsprechenden Sequenzen aus der Liste der Kandidatenallele zugeordnet. 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

Laufzeitanalyse

4.1 Graphkonstruktion und Bestimmung der Zusammenhangskomponenten

4.1.1 Laufzeit für das Hinzufügen der Knoten des Graphen

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 $O(|V|^2)$.

4.1.2 Laufzeit für das Hinzufügen der Kanten des Graphen

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 $k << |V|$ 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 $O(|E| \cdot |V|)$.

$$O(|E| \cdot (k + |V|)) = O(|E| \cdot |V|) \tag{3-7}$$

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 $O(|E| \cdot |V|)$.

$$O(|E| \cdot (k + |V|)) = O(|E| \cdot (|V| + |V|)) = O(|E| \cdot 2 \cdot |V|) = O(|E| \cdot |V|) \tag{3-8}$$

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:

$$O(|E| \cdot (k + |V|)) = O(|E| \cdot (k + k)) = O(|E| \cdot 2 \cdot k) = O(|E| \cdot k) \tag{3-9}$$

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 $\overline{k} = const$, so dass für die Berechnung der Likelihoods zwischen den Reads eine Laufzeit von $O(\overline{k}) = O(1)$ veranschlagt wird. Für das Hinzufügen der Kanten des Graphen wird somit die Gesamtlaufzeit auf $O(|E| \cdot |V|)$ geschätzt.

4.1.3 Laufzeit zur Bestimmung der Zusammenhangskomponenten

Die Bestimmung und Indexierung der Zusammenhangskomponenten durch graph-tool kann in $O(|V|+|E|)$ durchgeführt werden [32]. Zusammenhangskomponenten mit mehr als einem Knoten werden zunächst als gefilterte View des Gesamtgraphen erzeugt und anschließend als neuer eigenständiger Graph initialisiert. Da der Filtervorgang für alle Knoten jeder Zusammenhangskomponente durchgeführt werden muss, beträgt die Laufzeit $O(|C| \cdot |V|)$.

Die Subgraphen der Zusammenhangskomponenten werden als Elemente einer Liste zusammengefasst. Durch Erstellen einer Liste von Subgraphen kann später eine einfache Iteration über die Komponenten in $O(|C|)$ ausgeführt werden, ohne dass der Filtervorgang über alle Knoten jeder Komponente wiederholt werden muss. Das anschließende Löschen des ursprünglichen Graphen 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.

4.1.4 Gesamtlaufzeit der Konstruktion des Graphen und der Zusammenhangskomponenten

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.

$$O(|V|^2) + O(|E| \cdot |V|) = O(|V| \cdot (|V| + |E|)) \tag{3-10}$$

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 $O(|V| \cdot (|V| + |E|))$.

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

4.2 Laufzeit zur Bestimmung der Allele-Fractions mit maximaler Likelihood

4.3 Laufzeit zur Ermittlung der Allele-Fractions

Funktion `get_candidate_alleles()` 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.

Funktion `get_max_parsimony_n_alleles()` Die Anpassung der Anzahl der Allele kann für jede Komponente aus der Menge der Zusammenhangskomponenten 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.

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_{cand} 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].

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

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:

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

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

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

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

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

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:

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

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 `treshold-cluster-size` nur Allelkandidaten ab einer bestimmten Häufigkeit ihrere Sequenz in den Reads `treshold-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 $O(e^{n_{alleles}}) \cdot O(n_{alleles}) \cdot O(n_{cand})$. Den höchsten Wert nimmt $n_{alleles}$ in dem Fall an, dass die Restdivision $n_{cand} \mod \phi = 1$, denn dann gilt $n_{alleles} = n_{cand} + \phi - 1$. Da ϕ für alle Zusammenhangskomponenten den gleichen Wert besitzt, gilt $\phi = const.$ Daraus ergibt sich nach (3-21) eine Laufzeit von $O(e^{n_{cand}} \cdot n_{cand}^2)$ für die Funktion `get_candidate_vafs()`.

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

4.4 Laufzeit zur Likelihoodberechnung der Allele anhand der Allele-Fractions

Funktion `get_allele_likelihood_read()`

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

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

Im Hinblick auf `get_allele_likelihood_read()` müssen im Worst Case alle Schritte (**??**) bis (**??**) ausgeführt werden, so dass sich ihre Laufzeiten summieren. In der unter (**??**) 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 $O(V_c^2)$.

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

Die Suche bei (**??**) 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 $\overline{k} = const$ (siehe Kap. 4.1.4).

Die Laufzeit für das Auffinden eine geeigneten CIGAR-Tupels (**??**) 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 (**??**) 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 $O(V_c^3)$.

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

Funktion `get_cigar_tuples()`

Die hierfür erforderliche Laufzeit setzt sich zusammen aus den Suchvorgengen 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 Kantenaufruf 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 $O(V_c^3)$.

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

Funktion `calc_vafs_likelihood_read()` Formel (2-xxx2)

Es über die relativen Häufigkeiten jedes Allels einer Allele-Fraction iteriert und ihr Produkt mit der Readlikelihood aufsummiert. Da die Läge der Allelfraktion genau der Anzahl der Allele einer Zusammenhangskomponenten n_{cand} entspricht, erhöht sich hierdurch die Laufzeit um den Faktor $O(n_{cand})$.

Funktion `calc_vafs_likelihood()` Formel (2-xxx3) Dadurch wird dir Laufzeit zusätzlich um den Faktor der Anzahl der Reads der Zusammenhangskomponente $O(V_c)$ erhöht.

in `noderad_main.py` für alle Allele-Fractions Dazu muss über alle VAFs iteriert werden. Die Anzahl der VAFs entspricht dabei genau der Anzahl der Allelkombinationen. Wie bereits in Kap. 4.3 besprochen, kann diese hinsichtlich der Laufzeit mit $O(e^n_{alleles})$ abgeschätzt werden (Formel (3-20)). Da im Worst Case $n_{alleles} = n_{cand} + \phi - 1$, ergibt sich eine Laufzeit von $O(e^{n_{cand}+\phi-1}) \in O(e^{n_{cand}})$. Um diesen Faktor erhöht sich also die Gesamtlaufzeit bei der Berechnung der Liste der Gesamtwahrscheinlichkeiten.

Maximumsbestimmung: 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 $O(e^{n_{cand}})$ geschätzt werden.

4.4.1 Gesamtlaufzeit zur Bestimmung der Allele-Fractions 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_{cand}. 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 Allelfraktionen $O(e^n)$, 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 Allelfraktion 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 $n = n_{cand} = V_c$, 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 `treshold-cluster-size` und `treshold-seq-noise` ermöglichen eine Anpassung solcher großen Cluster, um die Anzahl der Kombinationen und damit den Rechenaufwand flexibel reduzieren zu können.

4.5 Laufzeit zur Bestimmung der Loci mit maximaler Likelihood

4.5.1 Laufzeit zur Bestimmung der möglicher Loci-Kombinationen

Wie bereits in Kap. 4.3 besprochen, benötigt die Generierung aller Allelkombinationen eine Laufzeit von $O(e^n)$, wobei $n = n_{cand}$ 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 $n_{alleles} = n_{cand} + \phi - 1$ betragen. Wegen $\phi = const$ beträgt die Laufzeit hier also $O(n_{cand}) = O(n)$. 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 $\phi = const$ 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. 4.3). Eine sinnvolle Anpassung der Grenzwerte `treshold-cluster-size` und `treshold-seq-noise` ermöglicht diesbezüglich eine Verbesserung. Allerdings ist an dieser Stelle ein effizienterer Ansatz zur Bestimmung der Loci-Kombinationen möglich, der in diesem Protoypen zwar keine Anwendung fand, aber für die spätere Implementierung in Rust von Bedeutung sein könnte. Dieser Ansatz wird im Ausblick in Kap. 6 kurz beschrieben.

4.5.2 Laufzeit 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 $O(\overline{k})$ veranschlagt werden, wobei $\overline{k} = const$ gilt. Da für alle Permuationen 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 $O(e^n \cdot n! \cdot V_c^3)$.

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

Wie bereits in Kap. 4.4.1 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 $n = n_{cand} = V_c$. 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. 4.4.1. Dies ist für große Cluster sehr problematisch. Die Anpassung der Schwellenwerte `treshold-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. 6 vorgestellt.

4.6 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 Kon-

struktion des Graphen und die Extraktion seiner Zusammenhangskomponenten C eine Laufzeit von $O(V \cdot (V + E))$ (Kap. 4.1.4). Für jede Zusammenhangskomponente erfolgte die Berechnung der wahrscheinlichsten Allelkombination in $O(e_c^V \cdot V_c^5)$ (Kap. 4.4.1) und die Berechnung der wahrscheinlichsten Loci-Zuordnung in $(e_c^V \cdot V_c!)$ (Kap. 4.5.2). 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.

Kapitel 5

Evaluation an simulierten Datensätzen

prototyp, zunächst für ploidie 2 -> bei höherer ploidie

5.1

5.1.1

Kapitel 6

Zusammenfassung und Ausblick

var callen
verschiedene Individuen mit einander vergleichen bei gleichem Anzatz, 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

6.1

6.1.1

Abbildungsverzeichnis

Algorithmenverzeichnis

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 Hufsky, Kerstin Scheubert, Jana Schleicher, and Stefan Schuster (editors): *German Conference on Bioinformatics 2012*, volume 26 of *OpenAccess Series in Informatics (OASIcs)*, 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: *Multiqc: 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 Antao, Jeffrey T. Chang, Brad A. Chapman, Cymon J. Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, and Michiel 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 — timecomplexity 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 8. Januar 2021

Muster Mustermann