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

1 Einleitung **1**

1.1 Biologischer Hintergrund 1

1.1.1 Aufbau und Struktur der DNA 1

1.1.2 Bindungen innerhalb und zwischen DNA-Molekülen 2

1.1.3 RNA und die Proteinbiosynthese 3

1.1.4 DNA-Replikation 4

1.1.5 Mutationen und SNPs 5

1.2 Molekulargenetische Verfahren und Techniken 7

1.2.1 Sanger-Sequenzierung 7

1.2.2 PCR 8

1.3 Next Generation Sequencing 9

1.3.1 RAD-Sequencing 9

1.4 Sequenzierungsdaten und -formate 12

1.4.1 Reads 12

1.4.2 Sequenzalignments 12

1.4.3 Varianten 13

2 Analyse von RAD-Seq-Daten **14**

2.1 Problemstellung 14

2.2 Formale Definition der Problemstellung 15

2.3 Lösungsansatz und Modell 15

2.3.1 Graph und Zusammenhangskomponenten 16

2.3.2 Approximation von PairHMM mittels Minimap2-Alignments 18

2.3.3 Allele-Fractions mit maximaler Likelihood 21

2.3.4 Locuszuordnung mit maximaler Likelihood 23

3 Implementierung **26**

3.1 Preprocessing 26

3.2 Edit-Distanzen 27

3.3 Konstruktion des Graphen 27

3.3.1 Knoten des Graphen . . . 28
3.3.2 Kanten des Graphen . . . 29
3.3.3 Bestimmung der Zusammenhangskomponenten . . . 29
3.4 Bestimmung der Allele-Fraction maximaler Likelihood . . . 31
3.4.1 Bestimmung der Kandidatenallele und ihrer möglichen Häufigkeitsverteilung . . . 31
3.4.2 Berechnung der Likelihoods der Allele anhand der Allele-Fractions . . . 34
3.5 Bestimmung der maximalen Likelihood der Loci . . . 39
3.5.1 Bestimmung der möglichen Loci . . . 39
3.5.2 Bestimmung der wahrscheinlichsten Locuszuordnung der Allele-Fraction mit maximaler Likelihood . . . 45
3.6 Ausgabe der wahrscheinlichsten Loci als VCF-Datei . . . 50

4 Laufzeitanalyse 52
4.1 Laufzeit für die Graphkonstruktion und Bestimmung der Zusammenhangskomponenten . . . 52
4.2 Laufzeitberechnung der VAFs mit maximaler Likelihood . . . 54
4.3 Laufzeit der Loci-Zuordnung . . . 60
4.4 Gesamtlaufzeit des Algorithmus . . . 64

5 Evaluation an simulierten Datensätzen 65
5.1 Simulation und Workflow-Run . . . 65
5.2 Ergebnisse . . . 66

6 Ausblick 75

A Anhang 78
A.1 Plots der BLAST-Analyse der Individuen B und C . . . 79

Abbildungsverzeichnis 88

Algorithmenverzeichnis 89

Literaturverzeichnis 97

Erklärung 97

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. 1.1.3) 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. 1.3 und Kap. 1.3.1). 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 und Deletionen (Indels) 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 Indels von mehreren Basen kann es zu einem Frame-Shift kommen, ändert sich dadurch allerdings die Länge des codierenden Abschnitts um drei Basen oder dem Vielfachen hiervon, 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. 1.3). 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 und kleinen Indels. Daher wird dieses Verfahren in Kap. 1.3.1 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 jedem Probengemisch ist also eine andere Base markiert und zum Teil auch modifiziert.

Wie bei der natürlichen DNA-Replikation (siehe Kap. 1.1.4) während der Zellteilung kann die Proben-DNA durch Hinzugabe der DNA-Polymerase I kopiert werden. Dabei werden auch die radioaktiv markierten Nukleotide in den kopierten Strang eingebaut. Die Kopiervorgänge starten jeweils an einem kleinen Fragment mit bekannter DNA-Sequenz, dem sog. Primer. Auch die Primer werden vorab den Probengemischen 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] gelang es schließlich, aus einzelnen DNA-Abschnitten multiple Kopien herzustellen, so dass auch kleinste DNA-Proben der Analyse zugänglich wurden. In modernen Verfahren ist es dadurch inzwischen möglich an der DNA einer einzigen Zelle umfangreiche Analysen durch-

zuführen [19].

Auf den bereits beschriebenen Prozessen zur Herstellung einer Kopie mit modifizierten Nukleotiden bei der Sanger-Sequenzierung basiert auch die Herstellung vielfacher, jedoch unmodifizierter Kopien bei der PCR. Temperaturabhängig wechseln hierbei mehrere Zyklen von Aufspaltung des DNA-Doppelstrangs, Primeranlagerung und DNA-Synthese ab. Entscheidend hierbei war die Verwendung einer thermostabilen DNA-Polymerase (Taq-Polymerase) [20]. Dadurch war es möglich mehrfach Zyklen wechselnder Temperaturen nacheinander durchzuführen, ohne dass die für die Synthese notwendige Polymerase bei höheren Temperaturen zerstört wurde. Die verwendeten Primer binden auf beiden Strängen am 3'-Ende der zu amplifizierenden DNA-Sequenz, so dass die DNA-Synthese auf beiden Strängen in 5'-3'-Richtung erfolgen kann. Im Gegensatz zur natürlichen Replikation entstehen somit keine Okazaki-Fragmente. Pro PCR-Zyklus c verdoppelt sich die durch die Primer eingegrenzte DNA-Sequenz, so dass die Anzahl der Kopien exponentiell mit 2^c ansteigt.

Durch die PCR war es nun möglich verstärkt Sequenzierungen durchzuführen und so die Erbinformation verschiedener Spezies, allen voran das Genom des Menschen im Rahmen des Human Genome Projects [21], zu sequenzieren und zu kartieren. Es entstanden große Gendatenbanken (z.B. Ensembl genome database [22], UCSC Genome Browser), welche die Referenzgenome vieler Spezies beinhalten.

1.3 Next Generation Sequencing

Durch die PCR wurde also der Weg bereitet, immer umfangreichere DNA-Sequenzierungen durchführen zu können und so wurden die Sequenzierungsverfahren in den darauffolgenden Jahren zunehmend optimiert und parallelisiert [23]. Mit den heutigen Next-Generation-Sequencing-Methoden [24, 25] können auf geringsten DNA-Mengen kostengünstig und im Hochdurchsatzverfahren Sequenzierungen durchgeführt werden, die auch ohne vorherige spezifische Kenntnisse über die zu sequenzierenden DNA-Bereiche möglich ist [26] . Hierzu gehört unter anderem auch die RAD-Sequenzierung.

1.3.1 RAD-Sequencing

Die RAD-Sequenzierung findet vor allem im Bereich der Populationsgenetik, Ökologie, Genotypisierung und Evolutionsforschung Anwendung. Das Prinzip der RAD-Sequenzierung basiert ursprünglich auf der Genotypisierung durch RAD-Marker [27]. Dies ermöglichte das gleichzeitige Mapping natürlicher Varianten und induzierter Mutationen bei verschiedensten Organismen. RAD-Marker oder RAD-tags sind kleine DNA-Fragmente, die durch

Verdau der DNA durch sogenannte Restriktionsenzyme entstehen.

Restriktionsenzyme sind molekulare Scheren, die in der Lage sind, die DNA an einer spezifischen DNA-Sequenz zu schneiden. Je nach verwendetem Restriktionsenzym können die Schnittstellen bezüglich des DNA-Doppelstrangs glatt oder versetzt sein [28]. Bei glatten Schnittstellen erfolgt die Aufspaltung beider DNA-Stränge an gleicher Position, so dass die entstehenden DNA-Fragmente keine Überhänge besitzen. Bei versetzten Schnittstellen erfolgt die Auftrennung auf beiden DNA-Strängen an verschiedenen Positionen, so dass bei den resultierenden DNA-Fragmenten einer der beiden Stränge gegenüber dem anderen länger hervorsteht. Diese überstehenden Sequenzen an den Fragmentenden werden häufig auch als sticky ends bezeichnet, da an ihnen besonders gut andere Sequenzen angefügt werden können. So werden beispielsweise für Sequenzierungen häufig Adaptersequenzen zur Bindung an Primer der Sequenzierplatte benötigt [29].

Neben der Nutzung der entstandenen DNA-Fragmente als Marker erkannte man auch bald die Vorteile ihrer Sequenzierung [30]. Aufgrund der bekannten Sequenz der Schnittstellen der Restriktionsenzyme können die Adaptersequenzen insbesondere bei sticky ends problemlos an die DNA-Fragmente gebunden werden. Die Adapter werden zusätzlich mit Barcodesequenzen versehen, welche die untersuchten Individuen identifizieren. Dadurch ist es möglich gepoolte Proben mit DNA-Fragmenten verschiedener Individuen gleichzeitig zu sequenzieren. Die dabei ausgelesenen kurzen DNA-Fragmente (Reads) lassen sich dann über ihre Barcodesequenzen den einzelnen Individuen wieder zuordnen [31]. Die gleichzeitige Analyse von Probengemischen verschiedener Individuen reduziert die Kosten und den Zeitaufwand der Sequenzierung erheblich. Insbesondere führt sie aber dazu, dass die Sequenzierung bei allen Individuen unter gleichen Versuchsbedingungen stattfindet, so dass die Reads der Individuen zuverlässiger mit einander vergleichbar sind. Durch die Sequenzspezifität der Restriktionsenzyme stammen die DNA-Fragmente bei evolutionär eng beieinander liegenden oder gleichen Spezies in der Regel vom gleichen genomischen Ort (Locus). Das Vorhandensein eines Referenzgenoms (siehe Kap. 1.2.2) ist hierbei keine Voraussetzung mehr, vielmehr können durch statistische Analysen für jedes Individuum mögliche Loci bestimmt [32] und hinsichtlich Varianten und Mutationen interindividuell für Diversitätsanalysen verwendet werden. Dies ermöglicht auch Untersuchungen von Spezies mit unbekanntem Genom. Auch das hier implementierte Tool NodeRAD [33] dient der Identifizierung der Loci und den Genotypen der einzelnen Individuen (Kap. 2) und kann in abgewandelter Form auch für den Vergleich verschiedener Individuen untereinander genutzt werden (Kap. 6).

Für die Sequenzierung werden nur kleine Fragmente von wenigen hundert Basenpaaren Länge verwendet, größere Fragmente werden aussortiert. Dadurch kommt es zu deutlichen

Verlusten bei den DNA-Fragmenten. Der Schritt der Größenselektion kann jedoch durch die Verwendung von zwei verschiedenen Restriktionsenzymen verbessert werden [34]. Bei diesem auch als ddRADSeq (double digest RADSeq) bezeichneten Verfahren werden die durch ein Restriktionsenzym erzeugten Fragmente an ihren Enden mit Adaptern versehen und anschließend ein weiteres mal mit einem anderen Restriktionenzym behandelt. Die neu entstandenen freien Enden werden mit anderen Adapter belegt. Dadurch werden die Fragmente weiter verkleinert, so dass mehr Fragmente nach der Größenselektion zur Verfügung stehen und sequenziert werden können. Hierbei können Restriktionsenzyme, die häufig im Genom vorkommende Sequenzen schneiden, mit Restriktionsenzymen kombiniert werden, die spezifisch für seltenere Sequenzen sind. Dadurch lässt sich eine bessere Steuerbarkeit und höhere Genauigkeit des Verfahrens erreichen.

Die DNA-Fragmente stammen aus dem gesamten Genom, ohne dieses vollständig abzudecken. Jedes der Fragmente wird in der Regel mehrfach sequenziert, so dass von jedem Abschnitt multiple, identische Reads entstehen. Dabei können in Abhängigkeit vom gewählten NGS-Verfahren die Reads von nur einem Ende (single-end) oder von beiden Enden aus (paired-end) sequenziert werden [29, 24, 25]. Durch die höhere Datendichte bei der paired-end Sequenzierung lassen sich Überlappungen der Reads und Sequenzierfehler besser erkennen.

Wie bereits erwähnt, stammen die Reads von verschiedenen Loci des Genoms. An einem Locus können innerhalb eines Individuums Unterschiede in der DNA-Sequenz der Reads vorkommen. Ursache hierfür ist die Ploidie des Chromosomensatzes. Der Mensch und die meisten Tierarten sind diploid, d.h. es liegt ein zweifacher Chromosomensatz vor. Jedes Chromosom ist in jeder Zelle doppelt vorhanden, wobei jeweils eines vom väterlichen und eines vom mütterlichen Elternteil stammt (homologe Chromosomen). Die homologen Chromosomen können jedoch Unterschiede in der DNA-Sequenz aufweisen, so dass am gleichen Locus unterschiedliche Varianten auf den homologen Chromosomen vorliegen. Solche Varianten eines Locus werden auch als Allele bezeichnet. Über die verschiedenen Allele kann der Genotyp bestimmt werden. Hierbei unterscheidet man Homo- und Heterozygotie. Bei Homozygotie bezüglich eines Locus weisen die homologen Chromosomen an dieser Stelle das gleiche Allel auf, bei Heterozygotie liegen dagegen unterschiedliche Allele vor. Die Analyse der RADSeq-Daten durch NodeRAD ist für diploide Organismen vorgesehen. Bei einigen wenigen Tierarten und häufiger auch bei Pflanzen können auch mehr als nur zwei Chromosomensätze vorkommen (Polyploidie). Um die Analyse polyploider Organismen zu ermöglichen, sind weitere Anpassungen am zugrunde liegenden Model von NodeRAD notwendig (vgl. Kap. 6)

1.4 Sequenzierungsdaten und -formate

Für die Speicherung und Verarbeitung der Daten aus genetischen Analysen haben sich eine Vielzahl verschiedener Formate für verschiedenste Anwendungsfälle etabliert. Daher soll im Folgenden kurz auf die in dieser Arbeit verwendeten Formate und ihre Besonderheiten eingegangen werden.

1.4.1 Reads

Die bei den Sequenzierung ausgelesenen kurzen Nukleotidsequenzabschnitte werden als Reads bezeichnet. Sie werden bei NGS-Verfahren in der Regel im FASTQ-Format gespeichert. Für jeden Read sind dort vier Zeilen vorgesehen, wobei die erste Zeile Angaben zur Identifikation des Reads und ggf. eine Beschreibung enthält. In der zweiten Zeile befindet sich die bei der Sequenzierung ausgelesene Nukleotidsequenz. Die dritte Zeile kann für weitere optionale Angaben und Beschreibungen verwendet werden. Und in der letzten Zeile wird der Quality-String hinterlegt, der für jede Base der Readsequenz Angaben zur Qualität der Sequenzierung beinhaltet.

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 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 [35, 36], 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.

Beim BAM-Format handelt es sich um eine komprimierte Binärdatei [35] mit der gleichen Information wie sie in einer SAM-Datei enthalten ist. Mit gängigen Tools, wie beispielsweise Samtools-view [36], können beide Formate in einander umgewandelt werden.

1.4.3 Varianten

Das Variant Call Format (VCF, [37]) ist das Standardformat für die Speicherung von Daten genetischer Varianten. Neben einem Header, in dem sich Metadaten sowie Beschreibungen, Filter und Spezifikationen einzelner Spalten befinden können, gibt es tabulatorgetrennt acht erforderliche Spalten (CHROM, POS, ID, REF, ALT, QUAL, FILTER, INFO) sowie die optionale Spalte FORMAT und beliebig viele optionale Spalten für die einzelnen Proben. Die Spalte CHROM gibt die Bezeichnung des Chromosoms an, auf dem sich die betreffende Variante befindet. In der Spalte POS wird die Position der erste Base der Variante angegeben. Die Spalte ID kann weitere Identifikatoren enthalten. In REF wird die Referenzsequenz und in ALT eine alternative Sequenz hinterlegt. Diese beiden Spalten spezifizieren somit die Sequenzunterschiede einer Variante. Angaben zur Qualität der detektieren Varianten finden sich in der Spalte QUAL. Angewendete Filter werden in der Spalte FILTER aufgeführt. Unter der Spalte INFO können zudem weitere Eigenschaften der Variante in Form von Key-Value-Paaren hinterlegt werden, die sich auf sämtlich Probenspalten beziehen. In der Spalte FORMAT können eine oder mehrere Flags gesetzt werden, die in den Probenspalten für jede Probe individuell angegeben werden. So zeigt beispielsweise die Flag “GT” an, dass in den Probenspalten Angaben zum Genotyp eingetragen werden.

Kapitel 2

Analyse von RAD-Seq-Daten

2.1 Problemstellung

Ausgangsmaterial für RAD-Seq-Analysen sind die Reads aus in der Regel gepoolten Probengemischen mehrerer Individuen. Durch entsprechende Barcodemarkierungen können die Reads den einzelnen Individuen zugeordnet werden. Die Verwendung gepoolter Probengemische macht das Verfahren kostengünstiger und zeitsparender als die meisten anderen NGS-Verfahren. Die Reads stammen aus dem gesamten Genom, wobei sie nur kleinere Sequenzabschnitte zwischen den Schnittstellen der Restriktionsenzyme repräsentieren und damit nicht das gesamte Genom abbilden. Ein deutlicher Vorteil des Verfahrens besteht 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 durch die Restriktionsenzyme die DNA nur innerhalb spezifischer Sequenzen geschnitten wird und dass durch PCR und Sequenzierung mehrere Reads erzeugt werden, die jeweils vom gleichen Locus stammen. Das hier implementierte Tool NodeRAD [33] stellt einen Prototypen dar, der im Gegensatz zu derzeit etablierten RADSeq-Tools, wie beispielsweise dem Tool Stacks [38], mit Hilfe von nur zwei Parametern diese Zuordnung realisiert. Das Clustering der Reads sowie die Identifizierung und Locizuordnung sollen dabei unter Berücksichtigung der Ploidie ohne weiteres Vorwissen allein auf Grundlage der Sequenzierfehlerrate und Heterozygotiewahrscheinlichkeit erfolgen. Die ermittelten Loci sollen für anschließende Diversitätsanalysen zur Verfügung stehen.

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 die Heterozygotiewahrscheinlichkeiten $\eta = (\eta_{sub}, \eta_{ins}, \eta_{del})$ und für Indels die Sequenzierfehlerraten $\epsilon = (\epsilon_{ins}, \epsilon_{del})$ gegeben. Ebenso ist die Ploidie ϕ des untersuchten Organismus bekannt. In der vorliegenden Arbeit wird im Model und beim implementierten Prototypen ein diploider Chromosomensatz vorausgesetzt, für höherploide Organismen ist eine Anpassung hinsichtlich des Sequenzalignments notwendig (siehe Kap. 6).

Ziel ist die Zuordnung der Reads zu den Loci unter Berücksichtigung von ϵ und η sowie die Ausgabe der Menge der ermittelten Loci mit den Sequenzen der beteiligten 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 ϵ und η besitzt.

2.3 Lösungsansatz und Modell

Das Problem wird als gerichteter Graph betrachtet (siehe Kap. 2.3.1), dessen Knoten die einzelnen Reads beinhalten und dessen Kanten auf einem Sequenzalignment der Reads basieren. Für die Zuordnung der Reads soll das Problem in Teilprobleme aufgeteilt werden, für jedes Teilproblem wird dann eine Lösung ermittelt. Der Graph wird daher entsprechend seiner Zusammenhangskomponenten partitioniert. Die in einer Zusammenhangskomponente vorkommenden Sequenzen werden in Abhängigkeit von konfigurierbaren Schwellenwerten als Kandidatenallele betrachtet und jeweils mit sämtlichen Reads der Zusammenhangskomponente verglichen. Dabei sollen diejenigen Allele identifiziert werden, von denen die übrigen Reads der Komponente am wahrscheinlichsten durch Sequenzierfehler entstanden sind (Kap. 2.3.3). Die Likelihoodberechnung hierfür wird durch die Basenqualität der Reads und Sequenzierfehlerrate bestimmt und erfolgt über alle Kombinationen möglicher Häufigkeitsverteilungen der Allele (Allele-Fractions).

Die Allele-Fraction mit der höchsten Likelihood Θ_{max_lh} soll anschließend für die Zuordnung der Loci verwendet werden (Kap. 2.3.4). Hierfür wird die Likelihood für alle Loci-Kombinationen bestimmt, welche zu der ermittelten Θ_{max_lh} passen. Für die Berechnung der Likelihoods der Locizuordnung werden die Heterozygotiewahrscheinlichkeiten verwendet. Die Loci-Kombination mit der größten Likelihood wird schließlich als Lösung der

betreffenden Zusammenhangskomponente ausgegeben.

Für jede Zusammenhangskomponente wird auf diese Weise die wahrscheinlichste Lösung ermittelt, deren Komposition von Loci die beobachteten Reads innerhalb der Zusammenhangskomponente am besten erklären kann. Die Loci mit maximaler Likelihood aus allen Zusammenhangskomponenten werden schließlich durch NodeRAD im VCF-Format ausgegeben. Eine Übersicht der einzelnen Schritte des Workflows findet sich in Abbildung 3.9. Die dort dargestellten Schritte werden im Folgenden näher erläutert.

2.3.1 Graph und Zusammenhangskomponenten

Im Preprocessing erfolgt die Zuordnung der Reads zu den einzelnen Individuen und die Entfernung der Barcodes aus den Sequenzen (Schritt 1 des Workflows). Anschließend wird in Schritt 3 aus den Reads ein paarweises Sequenzalignment mit Hilfe des Tools Minimap2 [42] erzeugt. Hierbei werden alle Reads miteinander verglichen und bei ausreichender Ähnlichkeit ihrer Sequenzen einander zugeordnet. Zusätzlich werden durch Minimap2 die Edit-Distanzen zwischen den jeweils miteinander 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 (Schritt 4 des Workflows). Die Knoten entsprechen dabei den einzelnen Reads. Auf Grundlage des von Minimap2 erzeugtem Sequenzalignment werden die Knoten der Reads durch Kanten verbunden. Die Kanten des Graphen repräsentieren somit den Vergleich der Sequenzen der beiden angrenzenden Reads. Die Anzahl der Kanten kann zusätzlich durch die von Minimap2 bestimmten Edit-Distanzen reguliert werden. Diese soll über einen konfigurierbaren Schwellenwert flexibel wählbar sein und ermöglicht ein zusätzliches Filtern der Kanten zugunsten von Laufzeit und Effizienz.

Da aufgrund des Sequenzalignments nur Reads mit einander verbunden werden, deren Sequenzen eine gewisse Ähnlichkeit zu einander aufweisen und da die RAD-Sequencing-Daten für gewöhnlich mehrere Loci beinhalten, ist der Graph in der Regel nicht zusammenhängend. Dies ermöglicht eine Unterteilung des Gesamtproblems in kleinere Teilprobleme durch Betrachtung der einzelnen Zusammenhangskomponenten des Graphen (Schritt 5). Für diese Teilprobleme wird dann nach Lösungen gesucht. Da sich die Konstellation der Zusammenhangskomponenten direkt aus dem Sequenzalignment ergibt, sind die so entstandenen Cluster nicht notwendiger Weise nur einem einzigen Locus zuzuordnen. Vielmehr können die einzelnen Zusammenhangskomponenten auch jeweils mehrere Loci beinhalten.

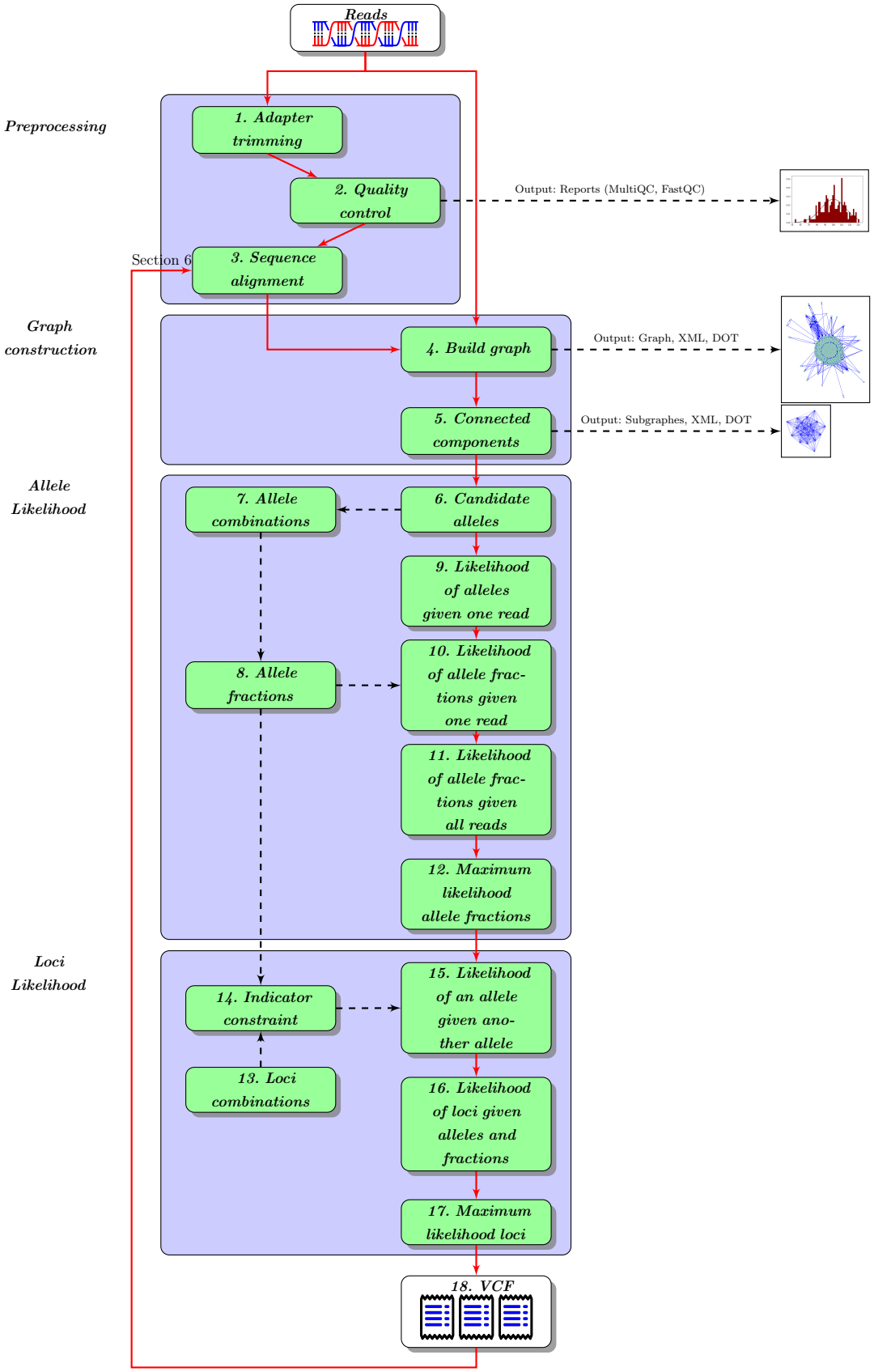

Abbildung 2.1: Übersicht der Prozesse des Workflows [39, 40, 41]

2.3.2 Approximation von PairHMM mittels Minimap2-Alignments

Um die wahrscheinlichsten Allele aus der Menge der Reads zu identifizieren und schließlich den wahrscheinlichsten Loci zuordnen zu können, muss ein paarweiser Vergleich der Readsequenzen hinsichtlich ihrer Ähnlichkeit zu einander erfolgen. Bei der Ermittlung der wahrscheinlichsten Allele werden hierfür alle Reads mit den infrage kommenden Allelen verglichen (siehe Kap. 2.3.2). Bei der Ermittlung der wahrscheinlichsten Loci erfolgt ein paarweiser Vergleich der Allele untereinander (siehe Kap. 2.3.2). Die Kanten des Graphen repräsentieren somit den Vergleich zweier Sequenzen, ihre Gewichtung soll die Ähnlichkeit zwischen den Sequenzen beschreiben. Diese Ähnlichkeit kann durch die Likelihood ausgedrückt werden, dass die jeweils betrachtete Sequenz (Querysequenz) aus der Vergleichssequenz (Referenzsequenz) entstanden ist. Die Kanten des Graphen verlaufen von der Query- zur Referenzsequenz, wobei die Likelihood der Kantengewichtung entspricht.

Die Bestimmung der Likelihood für die Kantengewichtung lässt sich durch ein pair Hidden Markov Model (pairHMM) [43] realisieren. Das beobachtete Ergebnis der Referenzsequenz wird dabei durch Abfolgen von Mismatch-, Insertions und Deletions-Operationen der Querysequenz erklärt (Evaluationsproblem). Jede dieser Operationen stellt einen Zustand dar, der Übergang zwischen den Zuständen wird über Wahrscheinlichkeiten abgebildet. Die Wahrscheinlichkeit, dass eine Base aus einem bestimmten Zustand heraus tatsächlich emittiert wird, d.h. in der Readsequenz zu beobachten ist, wird über die Emissionswahrscheinlichkeiten abgebildet. Die Zustände selbst sind allerdings nicht direkt beobachtbar. Die möglichen Kombinationen von Zustandsübergängen, die zum beobachteten Ergebnis der Referenzsequenz führen, können aber errechnet und als Matrix dargestellt werden. In der Matrix können dann die Pfade ermittelt werden, deren Zustandssequenzen geeignet sind, die Querysequenz in die Referezsequenz zu transformieren. Aus den Zustandsübergangs- und Emissionswahrscheinichkeiten dieser Pfade kann dann die Gesamtlikelihood bestimmt werden, dass die Querysequenz aus der Referenzsequenz entstanden ist. Diese Likelihood entspräche beim pairHMM der Kantengewichtung des Graphen.

Die Bestimmung der Kantenwahrscheinlichkeiten mittels pairHMM ist sehr rechenintensiv und wirkt sich entsprechend ungünstig auf die Laufzeit aus [43, 44]. Stattdessen kann das durch Minimap2 generierte Sequenzalignmment bei diploiden Organismen als Approximation des pairHMM verwendet werden. Minimap2 führt ebenfalls einen paarweisen Vergleich der Readsequenzen durch. Reads mit hoher Ähnlichkeit zueinander werden in das Alignment aufgenommen. Das Ergebnis ist ein Mapping derjenigen Reads, die eine hohe Wahrscheinlichkeit besitzen, durch Sequenzierfehler, Variationen oder Mutationen aus einem gemeinsamen Allel hervorgegangen zu sein. Dieses Mapping entspricht dem wahrscheinlichsten Pfad der pairHMM-Matrix. Da der Pfad mit maximaler Likelihood,

ohnehin das Ergebnis des pairHMM dominiert, genügt es die Likelihood dieses Pfades als Kantengewichtung zu betrachten. Somit basieren die Kanten des Graphen auf den Wahrscheinlichkeiten des Sequenzalignments.

Das durch Minimap2 generierte Sequenzalignment kann daher als Approximation des pairHMM betrachtet werden, so dass es genügt über diesem Alignment die Likelihood L_i hinsichtlich des Sequenzierfehlerrate bei der Auswahl der Allele (siehe Kap. 2.3.3) bzw. hinsichtlich der Heterozygotiewahrscheinlichkeiten bei der Zuordnung der Loci (siehe Kap. 2.3.4) zu berechnen. Diese Likelihoodberechnungen sollen zur besseren Veranschaulichung des Modells bereits an dieser Stelle besprochen werden, auch wenn dadurch einige Schritte des Workflows vorweg genommen werden.

Likelihoodberechnung der Allele bei einem gegebenen Read

Um in Kap. 2.3.3 die Wahrscheinlichkeit zu bestimmen, dass ein Read mit der Readsequenz s_r und der Fehlerrate ϵ tatsächlich aus einem bestimmten Allel hervorgegangen ist (Schritt 9 des Workflows), muss die Sequenzierfehlerrate einbezogen werden. Da der durch Minimap2 ermittelte wahrscheinlichste Pfad in der Regel die Gesamtlikelihood des pairHMM dominiert, genügt es, sich für die folgenden Likelihoodberechnungen auf diesen zu beschränken. Die wahrscheinlichste Zustandssequenz des approximierten pairHMM lässt sich daher dem CIGAR-String des Minimap2-Alignments entnehmen. Für jede Base der Querysequenz wird in Abhängigkeit von den Operationen des CIGAR-Strings die Sequenzierfehlerrate auf unterschiedliche Weise in die Likelihoodberechnung einbezogen.

Bei einem Match sind beide Basen aus Query- und Referenzsequenz identisch. Daher ergibt sich die Wahrscheinlichkeit L_{match}, dass es sich bei der Querysequenz an Position i tatsächlich um die korrekte Base handelt allein aus der Fehlerrate. Die geschätzte Fehlerrate p_i lässt sich anhand der Basenqualität des Reads nach (3-3) bestimmen (vgl. Kap. 3.4.2). Die resultierende Likelihood der Base wird um diese Fehlerrate reduziert:

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

Für Insertionen und Deletionen werden empirisch ermittelte Sequenzierfehlerraten ϵ_{ins} und ϵ_{del} verwendet [45]:

$$L_{indel} \in \{\epsilon_{ins}, \epsilon_{del}\} \tag{2-2}$$

Bei Substitutionen ergibt sich die Sequenzierfehlerrate ebenfalls aus der geschätzten Fehlerrate p_i im Bezug auf die Basenqualität des Queryreads q_{query}. Die Likelihood L_{sub}, dass anstelle der sequenzierten Base tatsächlich eine der drei anderen Basen vorliegt, entspricht dabei 1/3 der geschätzten Fehlerrate [46].

$$L_{sub} = \frac{1}{3} \cdot p_i \tag{2-3}$$

Die Likelihood $pairHMM_{\epsilon,q_{query}}\ (s_{query} \mid s_{ref})$, dass der Queryread aus dem Referenzread allein durch Sequenzierfehler entstanden ist, errechnet sich schließlich aus dem Produkt der Likelihoods $L_i \in \{L_{match}, L_{indel}, L_{sub}\}$ für jede Position i der Sequenz des Queryreads s_{query} im Vergleich zum Referenzread s_{ref}.

$$pairHMM_{\epsilon,q_{query}}\ (s_{query} \mid s_{ref}) \approx \prod_{i=1}^{k} L_i \tag{2-4}$$

Durch die Berücksichtigung der Basenqualität fließen Sequenzierfehler mit geringerer Likelihood in die weitere Berechnung ein. Folglich wird auch die Likelihood bei der späteren Loci-Zuordnung für Reads mit Sequenzierfehlern geringer ausfallen. Dadurch sinkt die Wahrscheinlichkeit, eine bessere Likelihood bei der späteren Loci-Zuordnung unter Einschluss eines Reads mit Sequenzierfehlern zu erreichen, als ohne diesen Read (siehe Kap. 2.3.4). Insgesamt wird somit der Einfluss von Reads mit Sequenzierfehlern reduziert.

Likelihoodberechnung bei der Zuordnung der Allele zu möglichen Loci

Ein weiteres Mal wird auf die Approximation des pairHMM durch Minimap2 in Kap. 2.3.4 zurückgegriffen, um die Menge der wahrscheinlichsten Allele einer Zusammenhangskomponente sinnvollen Locikombinationen zuordnen zu können (Schritt 15 des Workflows). Hierfür können bei diploiden Spezies die Allele ebenfalls paarweise unter Berücksichtigung der Hetereozygotiewahrscheinlichkeiten η_{sub}, η_{ins} und η_{del} mit einander verglichen werden. Auch hier wird die Sequenz eines der Allele (Query) gegen die Sequenz ein anderes Allels (Referenz) verglichen. Entsprechend den Operationen des CIGAR-Strings wird die Likelihood für jede Position i der Querysequenz bestimmt.

Im Falle eines Matches senken sämtliche Heterozygotiewahrscheinlichkeiten die Likelihood L_{match}, dass beide Allele vom gleichen Locus stammen (Formel (2-5)).

$$L_{match} = 1 - (\eta_{sub} + \eta_{ins} + \eta_{del}) \tag{2-5}$$

Bei einem Mismatch entspricht die Likelihood $L_{mismatch}$ der Heterozygotiewahrscheinlichkeit η der betreffenden Mutationsart, also η_{sub} für Substitutionen , η_{ins} für Insertionen bzw. η_{del} bei Deletionen.

$$L_{mismatch} \in \{\, \eta_{sub},\, \eta_{ins},\, \eta_{del} \,\} \tag{2-6}$$

Die Wahrscheinlichkeit $pairHMM_\eta(a_{l_{j,query}} \mid a_{l_{j,ref}})$ des paarweisen Vergleichs zweier Allele hinsichtlich eines Locus l_j errechnet sich wieder aus dem Produkt der Wahrscheinlichkeiten $L_i \in \{L_{match}, L_{mismatch}\}$ der einzelnen Basen:

$$pairHMM_\eta(a_{l_{j,query}} \mid a_{l_{j,ref}}) \approx \prod_{i=1}^{k} L_i \tag{2-7}$$

2.3.3 Allele-Fractions mit maximaler Likelihood

Bestimmung der Kandidatenallele

Um später 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 (Schritt 6 des Workflows). 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 Grenzwerte, welche die Mindestgröße der Cluster und die Mindesthäufigkeiten von Kandidatenallelsequenzen berücksichtigen sollen.

Über der Menge der Kandidatenallele sollen nun mögliche Allelkombinationen erzeugt werden (Schritt 7), die später den Loci zugeordnet werden. Dabei sollen nach dem Maximum-Parsimony-Prinzip unnötige und unmögliche Allelkombinationen eingespart werden. Die Länge dieser Kombinationen, also die erwartbare Anzahl von Allelen $n_{alleles}$, die auf einen oder mehrere Loci verteilt sind, ergibt sich dabei aus der Anzahl der Kandidatenallele n_{cand} und der Ploidie ϕ. 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 somit 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. Und es gilt dann $n_{alleles} = n_{cand}$. 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 somit um die Ploidie abzüglich des Restes aus der Restdivision $d = n_{cand} \mod \phi$ erhöht werden.

$$n_{alleles} = \begin{cases} \phi, & \phi \geq n_{cand} \\ n_{cand} + \phi - d, & \phi < n_{cand} \wedge d \neq 0 \\ n_{cand}, & \phi < n_{cand} \wedge d = 0 \end{cases} \tag{2-8}$$

Die Zusammenhangskomponenten können auch mehrere Loci enthalten, so dass in ihnen die einzelnen Allele mehrfach in verschiedenen Loci oder homozygot innerhalb eines Locus vorkommen können. Hinsichtlich der Allelkombinationen müssen also Wiederholungen der Allele erlaubt sein. Dies bedingt unterschiedliche Häufigkeiten der Kandidatenallele innerhalb der Allelkombinationen. Für jede Allelkombination werden in Schritt 8 des Workflows die relativen Häufigkeiten der darin enthaltenen Allele bestimmt. Diese repräsentieren die Häufigkeitsverteilung der Allele der jeweiligen Kombination und werden im Folgenden als Allele-Fractions oder VAFs bezeichnet.

Bestimmung der Likelihood eines Alleles im Bezug auf einen Read

Für jeden Read wird nun in Schritt 9 des Workflows die Wahrscheinlichkeit bestimmt, dass er aus einem bestimmten Allel allein durch Sequenzierfehler hervorgegangen ist. Die genaue Berechnung der Likelihood wurde in Kap. 2.3.2 bereits vorgestellt.

In Zusammenschau mit Formel (2-4) lässt sich beim paarweisen Vergleich eines Reads mit der Sequenz s_r und dem Kandidatenallel $a_i \in A$ die Likelihood $Pr(T = s_r \mid S = a_i, \epsilon)$ unter Berücksichtigung der Fehlerrate ϵ wie folgt errechnen:

Korollar (Likelihood eines Kandidatenallels gegeben ein Read).

$$Pr(T = s_r \mid S = a_i, \epsilon) = pairHMM_{\epsilon,q_r}(a_i, s_r) \tag{2-9}$$

Bestimmung der Likelihood einer Allele-Fraction im Bezug auf einen Read

In Schritt 10 des Workflows wird anschließend die Wahrscheinlichkeit ermittelt, einen bestimmten Read s_r anhand einer gegebenen Allele-Fraction $\Theta_i = (\theta_1, \ldots, \theta_n) \in [0, 1]^n$, mit $n = n_{cand}$, zu beobachten. Hierzu wird die in Formel (2-9) ermittelte Likelihood auf die relativen Häufigkeiten aller Kandidatenallele $(\theta_1, \ldots, \theta_n)$ der Allele-Fraction bezogen.

Da eine Zusammenhangskomponente auch mehrere Loci enthalten kann, sollen nun die Fractions in einer möglichen Menge von Loci ermittelt werden. Die Likelihood einer gegebenen Allele-Fraction ergibt sich aus einem Urnenmodell ähnlich dem einer Multinomialverteilung. Jedem Allel wird eine Kugelfarbe zugeordnet, die Häufigkeit der Kugeln einer Farbe entspricht der Fraction θ_i des betreffenden Allels. Die Wahrscheinlichkeit s_r zu ziehen ergibt sich dann entsprechend Korollar (2-10).

Korollar (Likelihood einer Allele-Fraction gegeben ein Read).

$$Pr(s_r \mid \Theta = \theta_1, \ldots, \theta_n) = \sum_{i=1}^{n} \theta_i \cdot Pr(T = s_r \mid S = a_i, \epsilon) \tag{2-10}$$

Bestimmung der Likelihood einer Allele-Fraction im Bezug auf alle Reads

Schließlich soll in Schritt 11 die resultierende Likelihood einer Allele-Fraction in Zusammenschau mit allen Reads bestimmt werden. Die Unsicherheit bei der Zuordnung der Reads wird dabei durch die relativen Häufigkeiten in der Allele-Fraction abgebildet und an die spätere Loci-Zuordnung (Kap. 2.3.4) weitergereicht. Sei L eine mögliche Loci-Verteilung, die durch die gegebene Allele-Fraction abgebildet wird und $D = (s_1, \ldots, s_m) \in \{A, C, G, T\}^{k^m}$ die Menge der Readsequenzen. Die möglichen Loci im Bezug auf die beobachteten Reads der Zusammenhangskomponente ergeben sich dann aus der Likelihood $Pr(D \mid \Theta)$ der Allel-Fraction Θ über allen Readsequenzen D. Dies lässt sich als Produkt der Likelihoods $Pr(s_r \mid \Theta)$ der Allele-Fractions für jeden Read formulieren:

Korollar (Likelihood einer Allele-Fraction gegeben alle Reads).

$$L(\Theta = \theta_1, \ldots, \theta_n \mid D) = Pr(D \mid \Theta = \theta_1, \ldots, \theta_n) = \prod_{r=1}^{m} Pr(s_r \mid \Theta = \theta_1, \ldots, \theta_n) \tag{2-11}$$

Die Schritte 9, 10 und 11 des Workflows werden für alle Allele-Fractions jeder Zusammenhangskomponente durchgeführt. Für jedes Kandidatenallel jeder Allel-Fraction wird so die Wahrscheinlichkeit berechnet, dass die Reads der Zusammenhangskomponente aus den einzelnen Kandidatenallelen entstanden sind. Da nur die Fehlerrate ϵ einbezogen wurde, können alle dabei entstandenen Abweichungen ausschließlich auf Sequenzierfehlern beruhen. Um diejenige Allel-Fraction zu wählen, deren Verteilung der Kandidatenallelsequenzen anhand der beobachteten Reads am wahrscheinlichsten ist, wird in Schritt 12 aus der Menge der errechneten Likelihoods aller Allele-Fractions das Maximum bestimmt.

2.3.4 Locuszuordnung mit maximaler Likelihood

In diesem Kapitel sollen die in der Allel-Fraction mit maximaler Likelihood $\Theta_{max\_lh} = (\theta_1, \ldots, \theta_n) \in [0, 1]^n$ vorkommenden Allele A_{max_lh} mit $\theta_i \neq 0$ den möglichen genomischen Loci zugeordnet werden.

Bestimmung Likelihood aus dem Vergleich zweier Allele

Um die Wahrscheinlichkeit zu bestimmen, inwieweit die einzelnen Allele aus A_{max_lh} hinsichtlich der Heterozygotiewahrscheinlichkeiten mit einander in Beziehung stehen, werden die Allele aus A_{max_lh} paarweise mit einander verglichen. Dies wurde bereits in Kap. 2.3.2 beschrieben.

In Zusammenschau mit Formel (2-7) ergibt sich für Schritt 15 die Wahrscheinlichkeit $Pr(T = a_{l_{j,2}}\,,\, S = a_{l_{j,1}}, \eta)$, dass zwei Allele $a_{l_{j,1}}$ und $a_{l_{j,2}}$ vom gleichen Locus l_j stammen:

Korollar (Likelihood des paarweisen Vergleichs zweier Allele).

$$Pr(T = a_{l_{j,2}}\,,\, S = a_{l_{j,1}}, \eta) = pairHMM_{\eta}(a_{l_{j,1}}, a_{l_{j,2}}) \tag{2-12}$$

Mögliche Locikombination und die Indikatorfunktion

Wie bereits erwähnt, können in Abhängigkeit von der Ploidie ϕ und der Anzahl der Kandidatenallele n_{cand} auch mehrere Loci in einer Zusammenhangskomponente vorkommen. Zunächst müssen daher in Schritt 13 des Workflows alle denkbaren Loci-Kombinationen gebildet werden. Hierfür werden die Permutationen über den zuvor bestimmten Allelkombinationen gebildet (Kap. 2.3.3). Jede Permutation wird dann entsprechend der gegebenen Ploidie zu Loci gruppiert. Dies wird in Kap. 3.5.1 näher ausgeführt. Jede Loci-Kombination kann hierdurch einen oder mehrere Loci beinhalten.

Für jede Loci-Kombination muss nun geprüft werden, ob sie die durch Θ_{max_lh} ermittelte wahrscheinlichste Häufigkeitsverteilung der Kandidatenallele erfüllen kann. Die absolute Häufigkeit jedes Kandidatenallels aus A_{max_lh} muss dabei, unter Berücksichtigung der zu erwartende Anzahl von Allelen $n_{alleles}$ und der Ploidie ϕ, zur Häufigkeit des entsprechenden Allels in der Loci-Kombination passen. Die Überprüfung wird durch eine Indikatorfunktion $z_l \in [0, 1]$ in Schritt 14 realisiert. Diese nimmt den Wert 1 an, wenn die Anzahl der einzelnen Kandidatenallele $A = (a_1, \ldots, a_n)$ in einer Loci-Kombination $L \in \{l_j \in \mathbb{N}^{\phi}_{\leq n} \,|\, j = 1, \ldots, g\}$ jeweils ihren absoluten Häufigkeiten in Θ_{max_lh} entspricht, andernfalls gilt $z_l = 0$. Jede Loci-Kombination L wird dabei als Tupel von Indizes einer lexikographisch sortieren Liste der Kandidatenallele dargestellt.

Korollar (Indikatorfunktion).

$$z_l = \prod_{i=1}^{n} 1_{\sum_{j=1}^{g} \sum_{k=1}^{\phi} 1_{l_{j,k}=i} = \theta_i \cdot g \cdot \phi} \tag{2-13}$$

Bestimmung der Likelihood der Loci-Kombinationen

Für jede Loci-Kombination L, welche die ermittelte Allele-Fraction mit maximaler Likelihood Θ erklären kann, wird schließlich in Schritt 16 die Wahrscheinlichkeit bestimmt, dass die enthaltenen Allele A bei gegebenen Heterozygotiewahrscheinlichkeiten in dieser Konstellation der Loci vorliegen. Die Likelihood errechnet sich dabei aus dem Produkt der paarweisen Allelkombinationen für jeden Locus l_j der gegebenen Loci-Kombination.

Korollar (Likelihood einer Loci-Kombination).

$$Pr(\Theta, A \,|\, L = \{l_j \,|\, j = 1, \ldots, g\}) = z_l \cdot \prod_{j=1}^{g} Pr(T = a_{l_{j,2}} \,,\, S = a_{l_{j,1}}) \tag{2-14}$$

Aus der Ergebnismenge der Likelihoods möglicher Loci-Kombinationen wird in Schritt 17 für jede Zusammenhangskomponente die Lösung mit maximaler Likelihood ausgewählt. Sie spiegelt die wahrscheinlichste Loci-Zuordnung der Komponente anhand der beobachteten Reads wider. Die Lösungen aus allen Zusammenhangskomponenten werden in Schritt 18 mit ihren Sequenzen und dem Genotyp in eine Datei im VCF-Format geschrieben.

Kapitel 3

Implementierung

Für das in Python implementierte RAD-Sequencing-Tool, NodeRAD, wurde zur Workflowintegration das Workflow-Management-System Snakemake verwendet [47, 48]. 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 IDs, der Basensequenz und Angaben zur Basenqualität enthält. Des Weiteren ist eine Tabelle im tsv-Format erforderlich, 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.

Der Source Code von NodeRAD kann unter https://github.com/AntonieV/NodeRAD eingesehen werden.

3.1 Preprocessing

Im Preprocessing werden durch das Tool Cutadapt [49] die Reads jedes Individuums anhand ihrer Barcodesequenzen identifiziert und extrahiert (Demultiplexing). Hiernach werden die Barcodesequenzen entfernt (Trimming) und die Reads jedes Individuums in

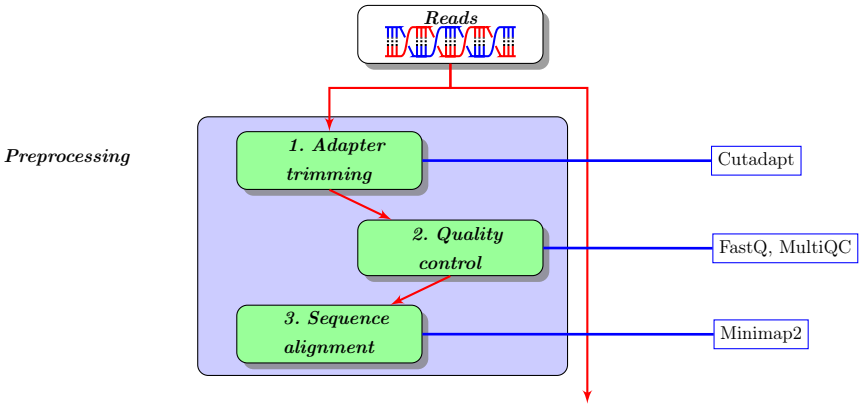

Abbildung 3.1: Prozesse des Workflows - Preprocessing [39, 40]

separaten Dateien im FASTQ Format abgelegt.

Im Anschluss an das Trimming erfolgt eine Qualitätskontrolle durch das Tool FastQC [50]. 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 [51] 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 [42] erstellt. Hierbei werden alle Readsquenzen paarweise mit einander verglichen. Das Ergebnis des Mappings wird anschließend durch Samtools-view [36] in das BAM-Format konvertiert 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

NodeRAD benötigt als Input zu jedem Individuum die getrimmten single-end Reads sowie das Sequenzalignment. Zur Konstruktion des Graphen wird die Python-Library graph-tool [52] genutzt. Zunächst wird daraus für jedes Individuum ein eigener, gerichteter Graph G mit $G = (V, E)$ erstellt. Seine Knoten, V, werden durch die einzelnen Reads repräsen-

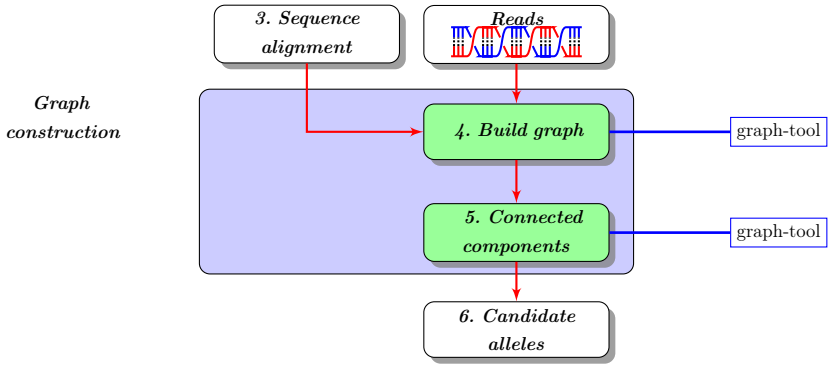

Abbildung 3.2: Prozesse des Workflows - Konstruktion des Graphen [39, 40]

tiert. Entsprechend ergeben sich die Knoteneigenschaften aus den Daten der Reads, diese werden den FASTQ-Dateien nach Ausführung von Cutadapt (siehe Kap. 3.1) entnommen. Die Kanten, E, zwischen den Knoten ergeben sich aus dem Vergleich ihrer Sequenzen im Rahmen des Sequenzalignments mittels Minimap2 (siehe Kap. 3.2).

Zusätzlich entnimmt NodeRAD der Konfigurationsdatei des Workflows einige Konstanten und Grenzwerte für die späteren Berechnungen. Zu den Konstanten gehören die Sequenzierfehlerraten für Indels, die Heterozygotiewahrscheinlichkeiten für Substitutionen, Insertionen und Deletionen sowie 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` für `small-clusters` und `large-clusters`) ober- und unterhalb einer bestimmten Clustergröße (`treshold-cluster-size`), die als Hintergrundrauschen nicht in der Berechnung Berücksichtigung finden sollen.

3.3.1 Knoten des Graphen

Die Knoten werden aus den FASTQ-Daten der getrimmten Reads mittels SeqIO aus der Library Biopython [53] 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 [54]. 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 den FASTQ-Daten erzeugten Knoten. Das Auslesen des Alignmentfiles im BAM-Format erfolgt mit Hilfe der Python-Library pysam [55]. 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. 2.3.2). 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 `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 ebenfalls durch graph-tool [56]. Die Indexnummer jeder Zusammenhangskomponente wird den in ihr enthaltenen Knoten als Knoteneigenschaft hinzugefügt. Zusammenhangskomponenten mit mehr als einem Kno-

ten 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.

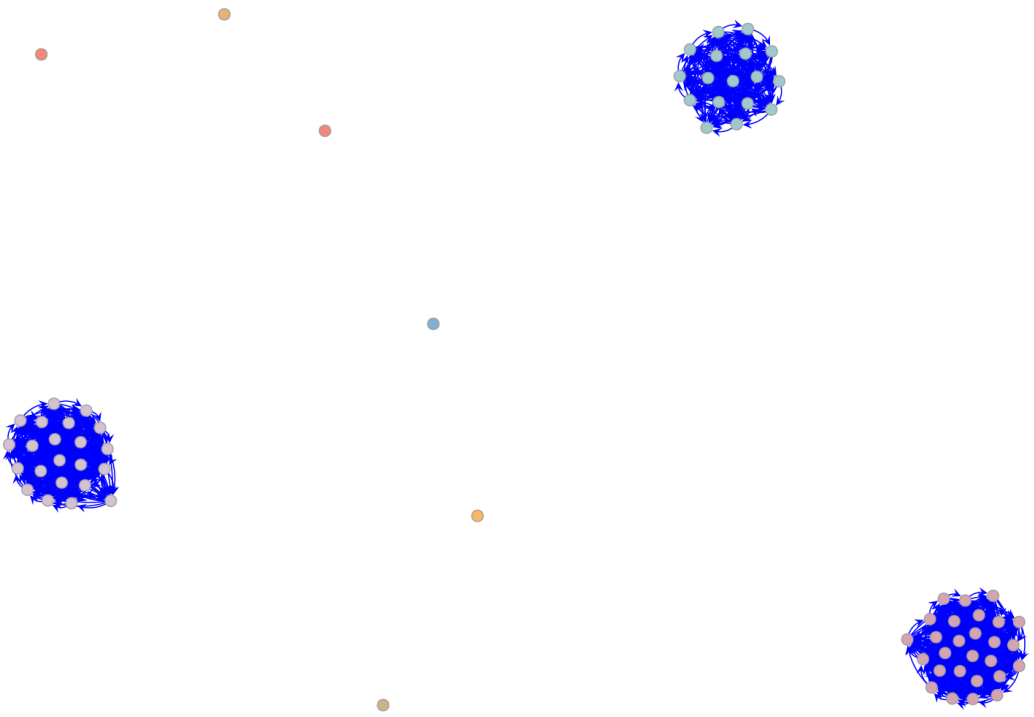

Abbildung 3.3: Ausschnitt einiger Zusammenhangskomponenten des Gesamtgraphen

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 3.4) und detaillierte Graphinformationen in den oben genannten Formaten des Graphen (Kap. 3.3.2)

ausgegeben werden.

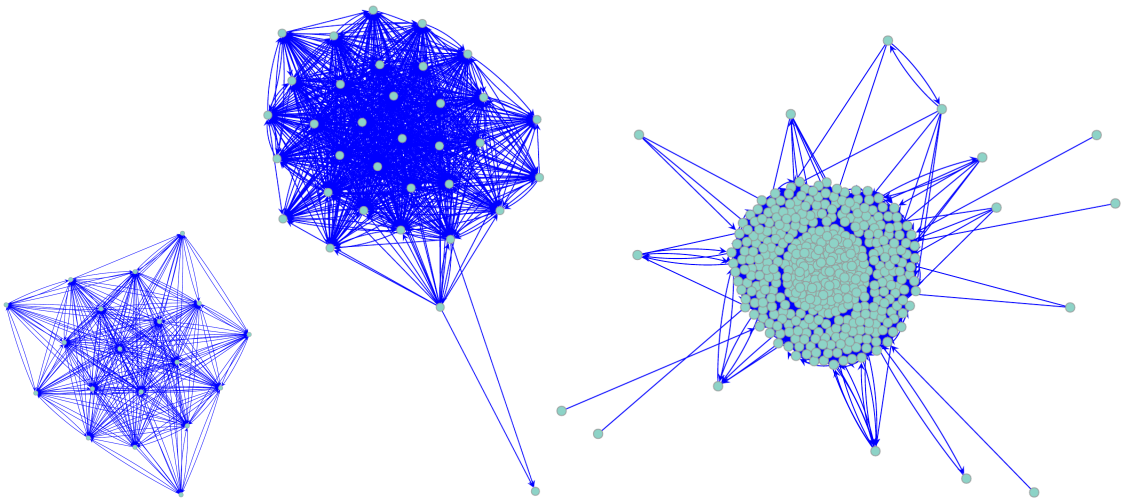

Abbildung 3.4: Beispiele von einzelnen Zusammenhangskomponenten unterschiedlicher Größe

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 3.3).

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. Alle hierfür implementierten Funktionen finden sich im Modul `likelihood_operations.py`.

Zunächst werden mit der Funktion `get_candidate_alleles()` die Allelsequenzen ermittelt, die in der Zusammenhangskomponenten vorkommen und deren Häufigkeiten bestimmt. Übersteigt die Größe eines Clusters, d.h. die Knotenanzahl einer Zusammenhangskomponenten, den in der Konfigurationsdatei unter `treshold-seq-noise` festgelegten Grenzwert `large-clusters`, so werden nur diejenigen Sequenzen als Kandidatenallele ausgewählt, deren Häufigkeit über dem `large-clusters` Schwellenwert liegt. Dies dient

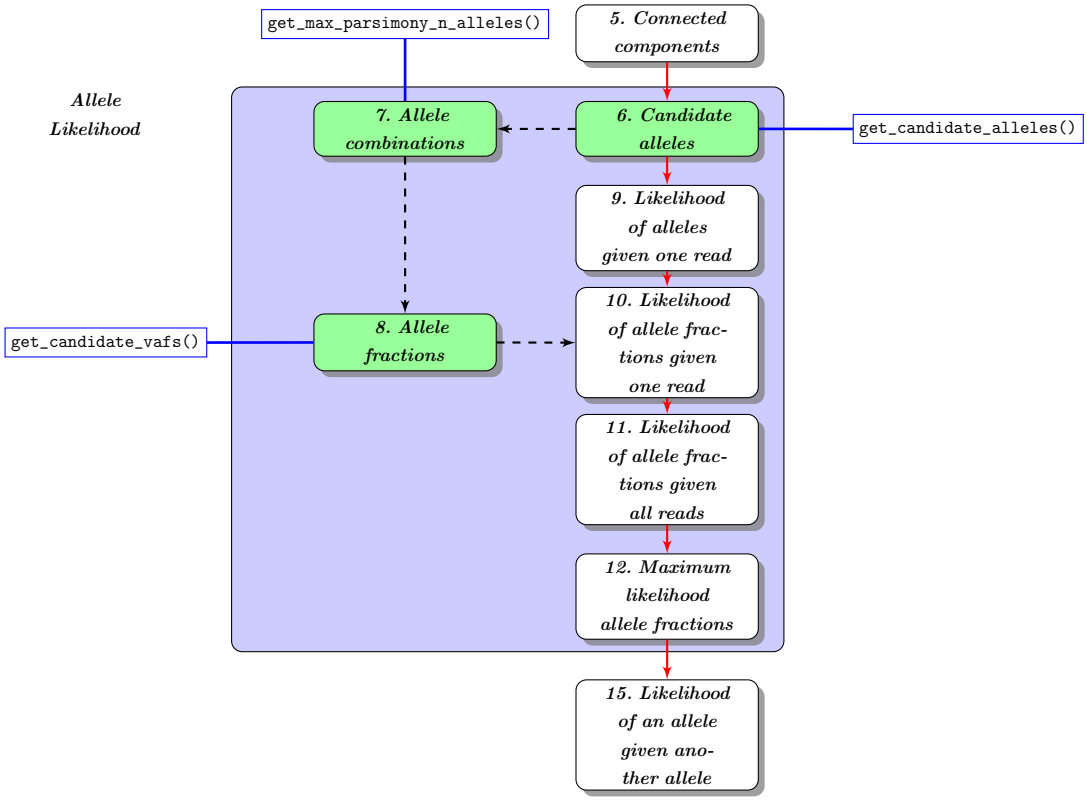

Abbildung 3.5: Prozesse des Workflows - Kandidatenallele und Allele-Fractions [39]

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. Ebenso kann es jedoch vorkommen, dass auch bei kleinen Clustern viele verschiedene Sequenzen auftreten, so dass es durch eine hohe Anzahl von Kandidatenallele zu einem erheblichen Rechenaufwand bei der späteren Analyse kommen kann. Um dem entgegenwirken, kann durch einen weiteren unter `treshold-seq-noise` festgelegten Grenzwert `small-clusters` die Mindesthäufigkeit der Sequenzen bei kleineren Clustern unterhalb von `treshold-cluster-size` angegeben werden. Da bei kleinen Clustern auch einmalig registrierte Varianten einer Sequenz von Bedeutung sein können, sollte der Filtervorgang hier möglichst gering gehalten werden. Gilt `small-clusters = 1`, so wird die Option für kleine Cluster vollständig deaktiviert.

Die Kandidatenallele werden anschließend für deterministischere Ergebnisse lexikographisch sortiert und als Liste A_{cand} der Länge n_{cand} von `get_candidate_alleles()` zurückgegeben. Aus dieser Liste 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()` nach Formel (2-8). Dadurch werden nicht mögliche Allelkombinationen eingespart und in der weiteren Berechnung nicht berücksichtigt. Dabei gibt `get_max_parsimony_n_alleles()` die Länge der Kombinationen zurück, also die erwartbare Anzahl von Allelen $n_{alleles}$, die auf einen oder mehrere Loci verteilt sein können. Wie bereits in Kap. 2.3.3 beschrieben, ergibt sich $n_{alleles}$ aus der Anzahl der Kandidatenallele n_{cand} und der Ploidie ϕ. Sie entspricht entweder direkt der Ploidie, falls diese höher ist als die Anzahl der Kandidatenallele. Andernfalls gilt $n_{alleles} \geq n_{cand}$ und $n_{alleles}$ muss ein ganzzahliges Vielfaches der Ploidie ergeben.

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()`). Diese Häufigkeitsverteilungen werden auch als Allele-Fractions bezeichnet. Hierfür werden zunächst alle möglichen Allelkombinationen ermittelt. Dies erfolgt nach einem 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 (3-1) aus dem Binomialkoeffizienten der k-ten Ordnung aus n Elementen mit Zurücklegen [57, 58].

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

Die Allelkombinationen werden mit Hilfe der Funktion `combinations_with_replacement` aus der Python-Library `itertools` erzeugt [59].

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 (3-2) bestimmt werden.

$$h_{a_i} = \frac{H_{a_i}}{n_{alleles}} \tag{3-2}$$

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

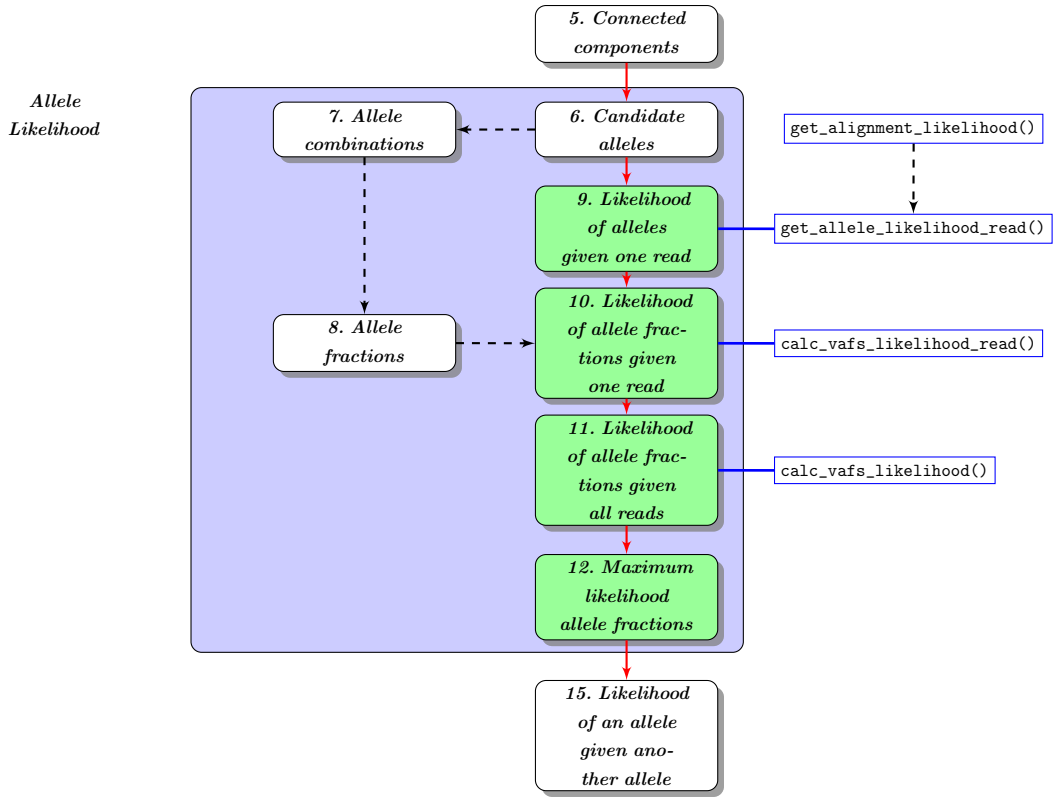

Abbildung 3.6: Prozesse des Workflows - Likelihood der Allele-Fractions [39]

In diesem Kapitel soll diejenige Allel-Fraction bestimmt werden, welche die beobachteten Reads am besten erklärt (vgl. Kap. 2.3.3). Für jeden Read wird dafür zunächst die Wahrscheinlichkeit bestimmt, dass der Read durch Sequenzierfehler aus einem der Kandi-

datenallele 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 der Read-ID und der Allelsequenz verwendet, die berechnete Likelihood wird als Value eingetragen. Da später für die Likelihoodberechnungen der verschiedenen Allel-Fraktions mehrfach die Werte der Read-Allel-Likelihoods verwendet werden, ermöglicht die Verwendung eines Dictionary eine schnellere und effizientere Suche der passenden Kante ohne wiederholte Likelihoodberechnung.

Existiert für 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 ID idx_{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 (idx_{r_i}, s_{a_i}) und als Values die Likelihoods L_{r_i,a_i}, so dass gilt $((idx_{r_i}, s_{a_i}), L_{r_i,a_i}) \in dict$. Die Verwendung fester Knoten- oder Kanteneigenschaften 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:     idx_{r_i} ← r_i[name]
3:     qual ← r_i[q_values]
4:     ϵ ← C_k[ϵ_ins] ∪ C_k[ϵ_del]
5:     if ∃(idx_{r_i}, s_{a_i}) : ((idx_{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(ϵ, 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(ϵ, 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 (2-9) des Modells. Wie in Kap. 2.3.2 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 des

Reads (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 (3-3) errechnet [60].

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

Im Anschluss wird aus P die Wahrscheinlichkeit errechnet, dass es sich im Falle eines Matches um die korrekte Base handelt (2-6) bzw. im Falle eines Mismatches, dass sich der Read durch Sequenzierfehler aus dem Allel erklären lässt (2-3). Für Indels werden die empirisch ermittelten Sequenzierfehlerraten von Illumina für Sequenzierplattformen [45] aus der Konfigurationsdatei unter dem Parameter `errors-per-base` entnommen. Entsprechend den in den CIGAR-Tupeln angegebenen Operationen wird für jede Base die Likelihood berechnet. Das Produkt der Likelihoods der einzelnen Basen ergibt schließlich die Likelihood zwischen Read und Allel (2-4).

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 Wahrscheinlichkeiten für Sequenzierfehler bei Insertionen und Deletionen aus den Grapheigenschaften werden dort mit ϵ_{ins} und ϵ_{del} bezeichnet, der Vektor der Phred Quality Scores der Querysequenz mit q_{query}

Algorithmus 3.2 Berechnung der Likelihood zwischen Read und Kandidatenallel

function GET_ALIGNMENT_LIKELIHOOD(ϵ_{ins}, ϵ_{del}, $cigar\_tuples$, q_{query}, reverse)
  $likelihood \leftarrow 1.0$, $index \leftarrow 0$, $p_{query} \leftarrow [\,]$
  for $q_i \in q_{query}$ **do**
    $p_{query} \leftarrow p_{query} \cup (10^{\frac{-q_i}{10}})$
  end for
  if $reverse$ **then**
    swap values of ϵ_{ins} and ϵ_{del}
  end if
  for all $(operation, length) \in cigar\_tuples$ **do**
    while $index < (index + length)$ **do**
      if $operation \in match$ **then**
        $likelihood \leftarrow likelihood \cdot (1 - p_{query}[index])$
      end if
      if $operation \in substitution$ **then**
        $likelihood \leftarrow likelihood \cdot \frac{1}{3} \cdot p_{query}[index]$
      end if
      if $operation \in insertion$ **then**
        $likelihood \leftarrow likelihood \cdot \epsilon_{ins}$
      end if
      if $operation \in deletion$ **then**
        $likelihood \leftarrow likelihood \cdot \epsilon_{del}$
      end if
      $index \leftarrow index + 1$
    end while
  end for
  return $likelihood$
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 (2-10) einbezogen werden. In der Funktion `calc_vafs_likelihood_read()` wird nun 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 (2-11) in `calc_vafs_likelihood()` die Gesamtlikelihood errechnet.

Auf diese Weise wird in `noderad_main.py` für jede Allel-Fraction die Gesamtlikelihood 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 maximalen Likelihood und den dazugehörigen Allelsequenzen festgehalten.

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 Allele-Fraction 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 aus der Allele-Fraction mit maximaler Likelihood die verschiedenen Kombinationen möglicher Loci durch `get_candidate_loci()` erzeugt werden. Die Funktion ist bereits in einer effizienteren Form implementiert und die zugrundeliegenden Verbindungen zum Modell sind dadurch nicht direkt erkennbar. Aus diesem Grund sollen an dieser Stelle zunächst die Überlegungen aufgezeigt werden, auf denen die effizientere Implementierung basiert, im Anschluss werden die Änderungen zur Effizienzverbesserung aufgezeigt.

Analog zu `get_candidate_vafs()` (Kap. 3.4.1) sollen aus der beobachteten n_{cand} und der tatsächlich zu erwartenden Anzahl von Allelen $n_{alleles}$ alle Allelkombinationen mit Wiederholung und ohne Berücksichtigung der Reihenfolge (vgl. Formel (3-1)) 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 Kom-

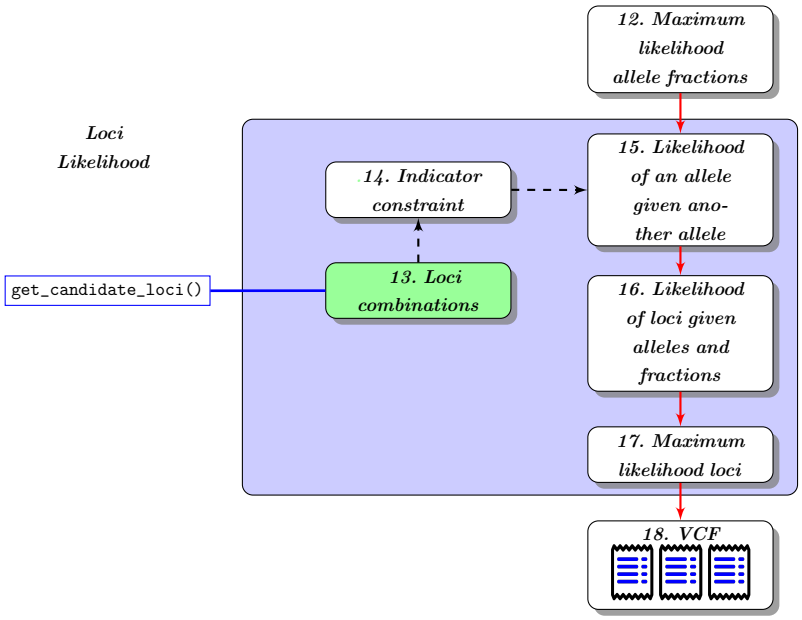

Abbildung 3.7: Prozesse des Workflows - Loci-Kombinationen [39, 41]

bination. Die Allelkombinationen 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 [59]. Die Permutationen werden über jede Allelkombination in ihrer vollständigen Länge gebildet, so dass gilt $n = k$, folglich werden insgesamt $n!$ Elemente erzeugt (3-4).

$$\frac{n!}{(n-k)!} = \frac{n!}{(n-n)!} = \frac{n!}{0!} = \frac{n!}{1} = n! \tag{3-4}$$

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 Zusam-

menhangskomponenten 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 der Allele:
(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)

Mögliche Loci-Kombinationen:
((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))

Für jede Loci-Kombination wird später in Kap. 3.5.2 durch die Indikatorfunktion nach Formel (2-13) geprüft, ob sie die Allele-Fraction mit maximaler Likelihood Θ_{max_lh} überhaupt repräsentieren kann. Das heißt die Häufigkeit jedes Allels über alle Loci einer gegebenen Loci-Kombination muss unter Berücksichtigung der Ploidie und der zu erwartenden Anzahl der Allele $n_{alleles}$ die relativen Häufigkeiten der ermittelten Allele-Fraction mit maximaler Likelihood erklären können.

Die hier zunächst vorgestellte Bildung von Permutationen über alle Allelkombinationen führt bei großen Clustern zu einer enormen Anzahl von Kombinationen. Da diese später ohnehin durch die Indikatorfunktion wieder reduziert werden, ermöglicht die direkte Erzeugung passender Loci-Kombinationen eine weitaus effizientere Implementierung. Im Hinblick auf die relativen Häufigkeiten in der Allel-Fraction mit maximaler Likelihood gibt es also eine feste Anzahl für jedes Allel, wie oft es über die Loci hinweg in einer Loci-Kombination auftreten darf. Diese Anzahl wird für jedes Allel durch die Indikatorfunktion getestet.

Da `combinations_with_replacement()` eine sortierte Ausgabe von Kombinationen ohne Beachtung der Reihenfolge erzeugt, erfüllt nur eine einzige der resultierenden Allelkombinationen tatsächlich die Bedingung der Indikatorfunktion (Satz 1) und es genügt, nur über dieser Kombination die Permutationen zu bilden. Es genügt also, aus der Allel-Fraction mit maximaler Likelihood, die absoluten Häufigkeiten der beteiligten Allele aus ihren relativen Häufigkeiten durch Umformung von Formel (3-2) zu bestimmen. Ein sortiertes Tupel der betreffenden Allele in der jeweils ermittelten Häufigkeit entspricht dann genau der einzigen gültigen Lösung für die Indikatorfunktion, die sich aus den Kombinationen bei Ausführung von `combinations_with_replacement()` ergeben hätte.

Durch die Umformung der Allel-Fraction mit maximaler Likelihood zu einer Allelkombination, müssen nun nur noch die Permutationen für diese Allelkombination bestimmt werden. Das unten stehenden Beispiel illustriert die Änderungen bei der effizienteren Implementierung. Nach anschließender Sortierung und Gruppierung zu Loci können die Duplikate entfernt werden.

Beispiele möglicher Loci-Kombinationen bei effizienter Implementierung:

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

Errechnete Allel-Fraction mit maximaler Likelihood:
[0.25, 0.5, 0.25]

Daraus resultierende sortierte Allelkombination:
(0, 1, 1, 2)

Permutationen der resultierenden Allelkombination:
(Duplikate wurden auch hier bereits entfernt)
(1, 0, 1, 2), (2, 1, 1, 0), (1, 1, 2, 0), (0, 1, 2, 1), (1, 0, 2, 1), (1, 2, 0, 1), (2, 1, 0, 1), (2, 0, 1, 1), (1, 1, 0, 2), (1, 2, 1, 0), (0, 2, 1, 1), (0, 1, 1, 2)

Mögliche Loci-Kombinationen:
((0, 1), (1, 2)), ((0, 2), (1, 1))

Indikatorfunktion:
$z_l = \prod_{i=1}^{3} 1_{\sum_{j=1}^{g} \sum_{k=1}^{\phi} 1_{l_{j,k}=i} = \theta_i \cdot g \cdot \phi}$

$z_{((0,1),(1,2))} = \prod_{i=1}^{n} 1_{\sum_{j=1}^{2} \sum_{k=1}^{2} 1_{l_{j,k}=i} = \theta_i \cdot 2 \cdot 2} = 1_{(1=0.25 \cdot 4)} \cdot 1_{(2=0.5 \cdot 4)} \cdot 1_{(1=0.25 \cdot 4)} = 1$

$z_{((0,2),(1,1))} = \prod_{i=1}^{n} 1_{\sum_{j=1}^{2} \sum_{k=1}^{2} 1_{l_{j,k}=i} = \theta_i \cdot 2 \cdot 2} = 1_{(1=0.25 \cdot 4)} \cdot 1_{(2=0.5 \cdot 4)} \cdot 1_{(1=0.25 \cdot 4)} = 1$

Das Ergebnis der Indikatorfunktion hängt allein von der absoluten Häufigkeiten der Allele ab und wird durch die Reihenfolge ihres Auftretens in einer Allelkombination nicht beeinflusst. Alle resultierenden Loci-Kombinationen der Funktion `get_candidate_loci()` erfüllen daher auch die Indikatorfunktion, da sie Permutationen einer Allelkombination sind, welche die Indikatorfunktion erfüllt. Daher müsste mit diesen Änderungen die Indikatorfunktion eigentlich nicht mehr geprüft werden. Sie wurde dennoch in der Implementierung belassen, um das Modell besser zeigen zu können und wird an entsprechender Stelle noch einmal erwähnt.

Satz (1). *Unter allen Allelkombinationen mit Wiederholung und ohne Beachtung der Reihenfolge gibt es nur eine Kombination, welche die Indikatorfunktion* (2-13) *erfüllt.*

Beweis. Für Kombinationen der Länge p mit Wiederholung aber ohne Beachtung der Reihenfolge gilt, dass sich jedes Element bis zu p Mal wiederholen kann. Da die Reihenfolge nicht beachtet werden soll, gilt außerdem dass jede Häufigkeitsverteilung der Elemente innerhalb einer Kombination in der Menge aller erzeugten Kombinationen nur genau einmal vorkommen darf. Andernfalls wäre dies eine Beachtung der Reihenfolge. Daraus kann abgeleitet werden:

1. Die absoluten Häufigkeiten einer Kombination sind in der Menge der erzeugten Kombinationen zu einer gegebenen Anzahl von Allelen und einer gegebenen Ploidie einzigartig.

2. Nach Formel 3-2 sind somit auch die relativen Häufigkeiten einer Allel-Fraction in der Menge der erzeugten Allele-Fractions zu einer gegebenen Anzahl von Allelen und einer gegebenen Ploidie einzigartig.

Es bleibt also zu beweisen, dass die Bedingung der Indikatorfunktion für genau eine Allel-Fraction gilt:

$$
\begin{aligned}
z_l &= \prod_{i=1}^{n} 1_{\sum_{j=1}^{g}\sum_{k=1}^{\phi} 1_{l_{j,k}=i}=\theta_i \cdot g \cdot \phi} \wedge \theta_i \in \Theta_{max\_lh} \\
&\Rightarrow z_l = 1 \Leftrightarrow \forall a_i \in A_{cand} : H_{a_i} = \theta_{a_i} \cdot g \cdot \phi \\
&\Rightarrow z_l = 1 \Leftrightarrow \forall a_i \in A_{cand} : H_{a_i} \overset{3-2}{=} \left(\frac{H_{a_i}}{n_{alleles}}\right)_{max\_lh} \cdot g \cdot \phi \\
&\Rightarrow z_l = 1 \Leftrightarrow \forall a_i \in A_{cand} : H_{a_i} = \left(\frac{H_{a_i}}{n_{alleles}}\right)_{max\_lh} \cdot n_{alleles} \\
&\Rightarrow z_l = 1 \Leftrightarrow \forall a_i \in A_{cand} : H_{a_i} = (H_{a_i})_{max\_lh}
\end{aligned}
$$

Für die Indikatorfunktion gilt also, dass ihre Bedingung nur dann erfüllt wird, wenn die absoluten Häufigkeiten der Allele einer Allelkombination den relativen Häufigkeiten der Allel-Fraction mit maximaler Likelihood Θ_{max_lh} entsprechen. Da absolute und relative Häufigkeitsverteilungen jeweils nur genau einmal bei den Kombinationen mit Wiederholung und ohne Beachtung der Reihenfolge auftreten, gibt es auch nur eine Allelkombination, welche über die Indikatorfunktion die Häufigkeitsverteilung der Allel-Fraction mit maximaler Likelihood erfüllen kann. □

3.5.2 Bestimmung der wahrscheinlichsten Locuszuordnung der Allele-Fraction mit maximaler Likelihood

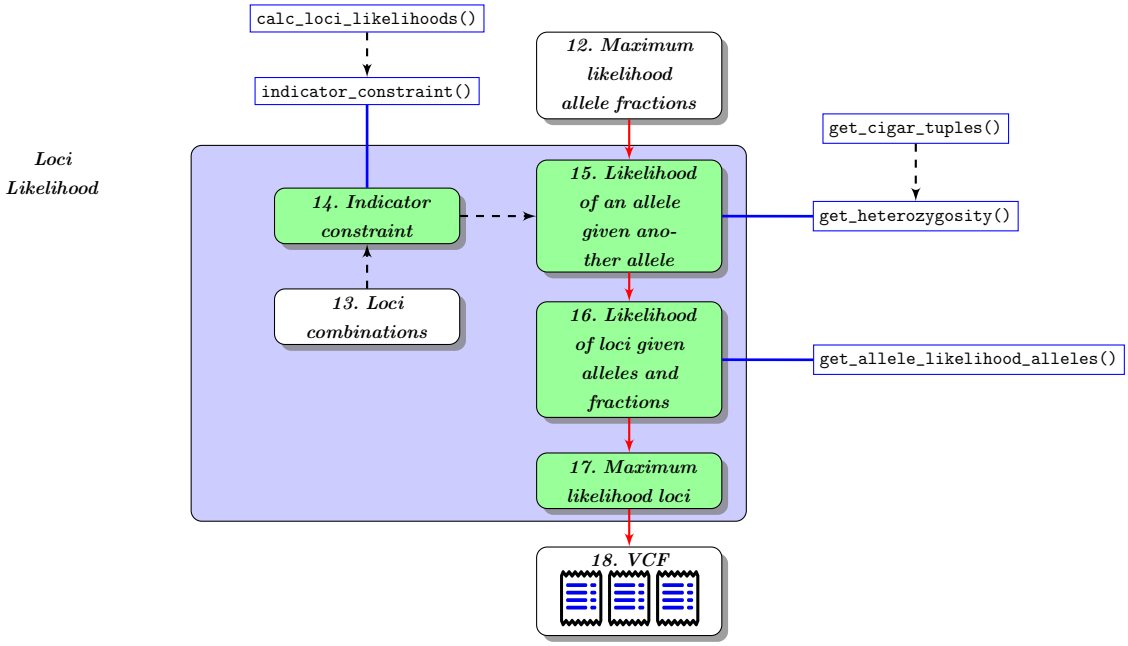

Abbildung 3.8: Prozesse des Workflows - Likelihood der Loci-Zuordnung [39, 41]

In diesem Schritt sollen schließlich für die Allel-Fraction 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 Loci-Kombination aus `get_candidate_loci()` wird aus der Funktion `calc_loci_likelihoods()` heraus zunächst geprüft, ob sich die Loci dieser Kombination der zuvor bestimmten Allele-Fraction mit maximaler Likelihood zuordnen lassen. Dies geschieht mit Hilfe der Indikatorfunktion `indicator_constraint()` nach Formel (2-13). 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. Wie bereits besprochen, ist durch die Änderungen an der Implementierung von `get_candidate_loci()` dieser Schritt nicht mehr notwendig und wurde nur

belassen, um das dahinterstehende Modell strukturierter zeigen zu können.

Für die Likelihoodberechnung müssen nun einer Loci-Kombination zunächst die passenden Allelsequenzen zugeordnet werden. Die einzelnen Allele innerhalb einer Loci-Kombination werden durch Integerwerte repräsentiert, die dem Index des dazugehörigen Kandidatenalleles entsprechen (siehe Kap. 3.4.1). Zu jeder Loci-Kombination wird also eine Liste $S_{loc,alleles}$ generiert, welche die Kandidatenallelsequenzen der einzelnen Loci als Sublisten beinhaltet.

Die Berechnung der Likelihood ist im vorliegenden Modell zunächst nur für diploide Organismen möglich. Analog zur Likelihoodberechnung beim paarweisen Vergleich der Reads unter Berücksichtigung der Sequenzierfehlerwahrscheinlichkeit (Kap. 3.3.2) erfolgt auch die Likelihoodberechnung für eine 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 (2-12) genutzt.

Hierzu erfolgt in `get_allele_likelihood_alleles()` (Algorithmus 3.3) die Likelihoodberechnung der Allelsequenzpaare a_i, a_j eines Locus entsprechend den Sublisten von $S_{loc,alleles}$. Der paarweise Vergleich der Allele ist allerdings nur für Daten diploider Organismen möglich, bei höherer Ploidie wird ein multiples Sequenzalignment benötigt (vgl. Kap 6). Für die Allelpaare jedes Locus werden dabei die CIGAR-Tupel durch die Funktion `get_cigar_tuples()` (Algorithmus 3.5) ermittelt und für die Likelihoodberechnung in `get_heterozygosity()` verwendet (Algorithmus 3.4). Die Gesamtwahrscheinlichkeit errechnet sich aus dem Produkt der Likelihood aller Allelpaare nach 2-14. Analog zur Likelihoodberechnung der Allele (vgl. Algorithmus 3.1) werden auch die ermittelten Locilikelihoods für die verschiedenen Loci-Kombinationen mehrfach benötigt. Daher wird die Effizienz auch hier durch Integration eines Dictionary mit den bereits ermittelten Likelihoods deutlich erhöht. Die Einträge des Dictionary sollen dabei die Tupel aus den beiden Allelsequenzen als Keys und die bereits errechneten Likelihoods als Values verwenden. Für alle Allelsequenzpaare wird noch vor der Suche nach geeigneten CIGAR-Tupeln das Dictionary $dict_{loc}$ auf bereits vorhandene Einträge überprüft. Falls es einen Eintrag gibt, so wird auf eine erneute Likelihoodberechnung verzichtet und stattdessen der Wert des Dictionary verwendet. Falls kein Eintrag existiert, so wird im Anschluss an die Likelihoodberechnung ein entsprechender Eintrag dem Dictionary hinzugefügt.

Algorithmus 3.3 Bestimmung der Likelihood der Allele innerhalb einer Loci-Kombination

1: **function** GET_ALLELE_LIKELIHOOD_ALLELES(C_k, $S_{loc,\,alleles}$, $dict_{loc}$)
2: $likelihood \leftarrow 1.0$
3: **for** $loc \in S_{loc,\,alleles}$ **do**
4: $pairs \leftarrow combinations(loc)$ ▷*itertools.combinations()*
5: **for** $(s_{a_i}, s_{a_j}) \in pairs$ **do**
6: **if** $\exists(s_{a_i}, s_{a_j}) : ((s_{a_i}, s_{a_j}), L_{a_i,a_j}) \in dict_{loc}$ **then**
7: $likelihood \leftarrow likelihood + L_{a_i,a_j}$
8: **else**
9: $cigar \leftarrow get\_cigar\_tuples(C_k, s_{a_i}, s_{a_j})$ ▷*Alg. 3.5*
10: **if** *cigar* exists **then**
11: $cig \leftarrow cigar\,[0]$
12: $rev \leftarrow cigar\,[1]$
13: $\eta_{rates} \leftarrow C_k[\eta_{sub}] \cup C_k[\eta_{ins}] \cup C_k[\eta_{del}]$
14: $L_{a_i,a_j} \leftarrow log(get\_heterozygosity(\eta_{rates}, cig, rev))$ ▷*Alg. 3.4*
15: $dict_{loc} \leftarrow dict_{loc} \cup ((s_{a_i}, s_{a_j}), L_{a_i,a_j})$
16: $likelihood \leftarrow likelihood + L_{a_i,a_j}$
17: **end if**
18: **end if**
19: **end for**
20: **end for**
21: **return** $e^{\,likelihood}$
22: **end function**

In `get_heterozygosity()` erfolgt die Likelihoodberechnung der Allelsequenzen eines Locus aus den CIGAR-Tupeln und den Heterozygotiewahrscheinlichkeiten. Dies beschränkt den Prototypen derzeit auf diploide Organismen (siehe auch Kap. 6). Wie in Algorithmus 3.2 wird auch hier die Likelihood über aller Matches und Mismatches aus den Informationen der CIGAR-Tupel bestimmt. Die Likelihoodberechnung der Matches erfolgt dabei nach Formel 2-5, die der Mismatches nach Formel 2-6. Das Argument `reverse` markiert zudem die Richtung der Kante, von der die CIGAR-Tupel stammen. Für Kanten in Rückrichtung, die von der Referenz- zur Query-Sequenz verlaufen, gilt `reverse=True`. In diesem Fall werden Deletionen als Insertionen behandelt und umgekehrt.

Algorithmus 3.4 Bestimmung der Likelihood zwischen zwei Kandidatenallelen 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:     for all (operation, length) ∈ cigar_tuples do
7:         if operation ∈ match then
8:             likelihood ← likelihood · (1 − (η_sub + η_ins + η_del))^length
9:         end if
10:        if operation ∈ substitution then
11:            likelihood ← likelihood · (η_sub)^length
12:        end if
13:        if operation ∈ insertion then
14:            likelihood ← likelihood · (η_ins)^length
15:        end if
16:        if operation ∈ deletion then
17:            likelihood ← likelihood · (η_del)^length
18:        end if
19:    end for
20:    return likelihood
21: end function
```

Wie bereits erwähnt, werden für den paarweisen Vergleich zweier Kandidatenallele im Rahmen der Likelihoodberechnung die dazugehörigen CIGAR-Tupel benötigt. Diese werden durch die Funktion `get_cigar_tuples()` ermittelt. Als Eingabeparameter erhält die Funktion den Subgraphen der Zusammenhangskomponente sowie die beiden Kandidatenallelsequenzen: die Query-Sequenz s_{source} und die Referenz-Sequenz s_{target}. Der Subgraph 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 Kanten in Gegenrichtung. Zur Markierung der Kantenrichtung gibt die Funktion neben den CIGAR-Tupeln auch einen booleschen Wert zurück. Dieser wird in der Funktion `get_heterozygosity()` (Kap. 3.5.2) für das Argument `reverse` verwendet.

Um eine passende Kante im Subgraphen zu finden, erstellt `get_cigar_tuples()` zunächst eine Liste aller Knoten $R_{source} = (v_1, \ldots, v_i)$, welche die Query-Sequenz s_{query} besitzen sowie eine Liste aller Knoten $R_{target} = (w_1, \ldots, w_j)$, die die Referenz-Sequenz s_{ref}

tragen. Hierfür werden für beide Sequenzen die Knoten des Subgraphen mit `find_vertex()` aus graph-tool durchsucht. Anschließend 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 wird sowohl für die angegebene Kantenrichtung und falls erfolglos auch für die entgegengesetzte Richtung durchgeführt. Existiert eine solche Kante, dann werden die CIGAR-Tupel sowie der boolsche Wert entsprechend der Kantenrichtung zurückgegeben, andernfalls wird $None$ zurückgegeben.

Zur Veranschaulichung ist der Algorithmus in 3.5 zudem als Pseudocode dargestellt. Dabei sei $C_k \in \{C_1, \ldots, C_p\}$ der Subgraph einer Zusammenhangskomponente mit seiner Kantenmenge E_k und s_{query} bzw. s_{ref} seien die Query- bzw. 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

function GET_CIGAR_TUPLES(C_k, s_{query}, s_{ref})
  $R_{source} \leftarrow \{v_i \in C_k \wedge i, k \in \mathbb{N} \mid v_i[sequence] = s_{query}\}$
  $R_{target} \leftarrow \{w_j \in C_k \wedge j, k \in \mathbb{N} \mid w_j[sequence] = s_{ref}\}$
  for $v_i \in R_{source}$ **do**
    for $w_j \in R_{target}$ **do**
      if $(v_i, w_j) \in E_k$ **then**
        return ($E_k[(v_i, w_j)][cigar\_tuples]$, $False$)
      end if
      if $(w_j, v_i) \in E_k$ **then**
        return ($E_k[(w_j, v_i)][cigar\_tuples]$, $True$)
      end if
    end for
  end for
  return $None$
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 der

Loci-Kombination sowie die Loci mit maximaler Likelihood.

3.6 Ausgabe der wahrscheinlichsten Loci als VCF-Datei

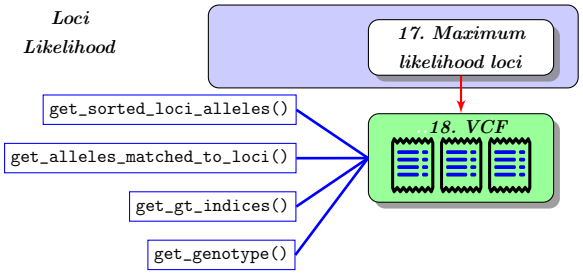

Abbildung 3.9: Prozesse des Workflows - Ausgabe als VCF-File [39, 41]

Die Loci-Kombinationen mit maximaler Likelihood Loc_{max} aller Zusammenhangskomponenten werden für jedes Individuum jeweils in eine Datei im Variant Call Format (siehe Kap. 1.4.3) geschrieben. Dabei werden zusätzlich die optionalen FORMAT- und Proben-Spalten zur Kennzeichnung des Genotyps verwendet. Die Spalte CHROM wird für die Locibezeichnungen verwendet. Diese setzen sich zusammen aus dem Präfix "LOC" und einer laufenden Nummer, beginnend bei der Zahl Null, zur Indexierung.

In die Spalten REF und ALT werden die Sequenzen der Allele aus Loc_{max} lexikographisch sortiert eingetragen. 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. In Abbildung 3.10 ist eine VCF-Datei, wie sie durch NodeRAD erzeugt wird, mit kurzen beispielhaften Sequenzen dargestellt.

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	A
LOC0	1	.	TTTGCATGTTGCGTCAGA	TTTGCATGTTGCGTCCGA	.	.	.	GT	0/1
LOC1	1	.	GGGTATCGCCTGTGGCTG	.	.	.	.	GT	0/0
LOC2	1	.	AAGGATCTTTGCCGACTT	.	.	.	.	GT	0/0
LOC3	1	.	GGCTAAGTTAACTTGAGA	.	.	.	.	GT	0/0
LOC4	1	.	GCGTCGAATCGGCACTCG	.	.	.	.	GT	0/0
LOC5	1	.	GTAAATCGATGCGGCGTG	.	.	.	.	GT	0/0
LOC6	1	.	GATGCCTGATGCGTCTTT	.	.	.	.	GT	0/0
LOC7	1	.	CGATCCCGCCATATGCAC	.	.	.	.	GT	0/0
LOC8	1	.	CCCCGACGAGTCTATCTC	.	.	.	.	GT	0/0
LOC9	1	.	TGACGCTTTGTTTATCTG	TGACGCTTTGATTATCTG, TTACGCTTTGTTTATCTG	.	.	.	GT	0/1
LOC10	1	.	TGACGCTTTGTTTATCTG	TGACGCTTTGATTATCTG, TTACGCTTTGTTTATCTG	.	.	.	GT	1/2
LOC11	1	.	TTTATACGCGGACACTCT	.	.	.	.	GT	0/0
LOC12	1	.	GTTTGGTTCACTGTCCTT	.	.	.	.	GT	0/0
LOC13	1	.	CGCCGTGTGTGTTTGCCA	CGCCGTGGGTGTTTGCCA	.	.	.	GT	0/1
LOC14	1	.	TGGTCGCCTTTTGCCAAT	.	.	.	.	GT	0/0
LOC15	1	.	AGCAATTTAAACCCGATA	.	.	.	.	GT	0/0

Abbildung 3.10: Beispiel einer durch NodeRAD erzeugten VCF-Datei (zur besseren Darstellbarkeit wurden verkürzte Sequenzen verwendet)

Kapitel 4

Laufzeitanalyse

4.1 Laufzeit für die Graphkonstruktion und Bestimmung der Zusammenhangskomponenten

Laufzeit für das Hinzufügen der Knoten des Graphen

Die Laufzeit für das Hinzufügen eines Knotens (siehe Kap. 3.3.1) im Graphen $G = (V, E)$ beträgt nach der Dokumentation von graph-tool $O(1)$ [56]. Die Zuweisung der Knoteneigenschaften erfolgt ebenfalls in $O(1)$. Über alle Reads, also über die resultierende Anzahl der Knoten $|V|$ ergibt sich daraus eine Gesamtlaufzeit von $O(|V|)$.

Laufzeit für das Hinzufügen der Kanten des Graphen

Das Hinzufügen einer Kante im Graphen (siehe Kap. 3.3.2) benötigt laut graph-tool-Dokumentation [56] eine Laufzeit von $O(1)$. Da aber die Query- und die Referenzreads aus dem Alignment den zuvor angelegten Knoten zugeordnet werden müssen, erfordert dies eine Suche der betreffenden Knoten im Graphen. Dabei durchsucht graph-tool mit seiner Funktion `find_vertex()` nur die Knoten und prüft auf die gesuchte Read-ID aus dem FASTQ-File. 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 [61]. 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 alle Kanten ergibt sich daraus eine Gesamtlaufzeit von $O(|E| \cdot |V|)$.

Laufzeit zur Bestimmung der Zusammenhangskomponenten

Die Bestimmung der Zusammenhangskomponenten $C = (C_1, \ldots, C_h)$ durch graph-tool kann in $O(|V|+|E|)$ durchgeführt werden [56], hierbei wird jedem Konten die Indexnummer seiner Zusammenhangskomponente zugewiesen (siehe Kap. 3.3.3). Aus Zusammenhangskomponenten mit mehr als einem Knoten sollen hiernach eigenständige Subgraphen erzeugt werden. Dazu wird in der Funktion `get_components()` des Moduls `graph_operations.py` zunächst ein nach den Knoten der Komponente gefilterter View des Gesamtgraphen erzeugt, welcher anschließend als neuer eigenständiger Graph initialisiert wird. Da beim Filtervorgang jeder Zusammenhangskomponente jeweils alle Knoten des Graphen betrachtet werden müssen, beträgt die Laufzeit hierfür $O(h \cdot |V|)$, wobei h die Anzahl der Zusammenhangskomponenten ist.

Die Subgraphen der Zusammenhangskomponenten werden als Elemente einer Liste zusammengefasst. Durch das Erstellen einer Liste von Subgraphen kann später eine einfache Iteration über die Komponenten in $O(h)$ ausgeführt werden, ohne dass der Filtervorgang über alle Knoten jeder Komponente wiederholt werden muss.

Die Gesamtlaufzeit für die Bestimmung der Zusammenhangskomponenten und die Generierung eigenständiger Subgraphen beträgt nach Formel (4-1) somit $O(h \cdot |V| + |E|)$.

$$\begin{aligned} O(h \cdot |V|) \; + \; O(|V| + |E|) \; &= O(|V| \cdot (h+1) + |E|) \\ &= O(h \cdot |V| + |E|) \end{aligned} \tag{4-1}$$

Bei realen Daten gibt es in der Regel deutlich mehr Knoten als Cluster bzw. Zusammenhangskomponenten, so dass gilt $h < |V|$. Würde im Worst Case aber jede Zusammenhangskomponente aus nur einem Knoten bestehen, also $h = |V|$, so kann die maximale Laufzeit auf $O(|V|^2 + |E|)$ geschätzt werden (siehe Formel (4-2)).

$$\begin{aligned} O(h \cdot |V| + |E|) \; &= O(|V| \cdot |V| + |E|) \\ &= O(|V|^2 + |E|) \end{aligned} \tag{4-2}$$

Gesamtlaufzeit der Konstruktion des Graphen und der Zusammenhangskomponenten

Aus den Laufzeiten für die Erzeugung der Knoten in $O(|V|)$, das Hinzufügen der Kanten in $O(|E| \cdot |V|)$ sowie die Bestimmung und Extraktion der Zusammenhangskomponenten in

$O(|V|^2 + |E|)$ ergibt sich nach (4-3) eine Gesamtlaufzeit von $O(|V| \cdot (|V| + |E|))$.

$$\begin{aligned} & O(|V|) + O(|E| \cdot |V|) + O(|V|^2 + |E|) \\ &= O(|V| + |V| \cdot |E| + |V|^2 + |E|) \\ &= O(|V| \cdot (1 + |E|) + |V|^2 + |E|) \\ &\in O(|V| \cdot |E| + |V|^2 + |E|) \\ &= O(|V|^2 + |E| \cdot (1 + |V|)) \\ &\in O(|V|^2 + |E| \cdot |V|) \\ &= O(|V| \cdot (|V| + |E|)) \end{aligned} \tag{4-3}$$

4.2 Laufzeit zur Bestimmung der Allele-Fractions mit maximaler Likelihood

Da alle weiteren Schritte für jede Zusammenhangskomponente $C_i \in \{C_1, \ldots, C_h\}$ durchgeführt werden, sollen hier zunächst die Laufzeiten für eine Komponente betrachten werden und erst am Ende des Kapitels die Laufzeit für alle Komponenten bestimmt werden. Die Zusammenhangskomponenten sind Subgraphen des Gesamtgraphen $G = (V, E)$, so dass gilt $C_i \subseteq G$ und $C_i = (V_{C_i}, E_{C_i})$. Sei also im Folgenden $|V_{C_i}|$ die Anzahl der Knoten innerhalb einer Komponente.

Laufzeit der Funktion `get_candidate_alleles()`

Für das Erstellen der Liste von Kandidatenallelen muss jeder Knoten einer Zusammenhangskomponente betrachtet werden, so dass der Vorgang in $O(|V_{C_i}|)$ durchführbar ist.

Laufzeit der Funktion `get_max_parsimony_n_alleles()`

Die Anpassung der Anzahl der Allele an die gegebene Ploidie kann für jede Komponente in $O(1)$ erfolgen.

Laufzeit der Funktion `get_candidate_vafs()`

Für die Bestimmung der Allelkombinationen werden durch Verwendung von `itertools.combinations_with_replacement()` alle Kombinationen der Kandidatenallele generiert, wobei Wiederholungen erlaubt sind. Da kombinatorische Problemstellungen zu hohen Laufzeiten und deutlichem Speicherplatzbedarf führen, sind sie oft der dominierende und ggf. auch limitierende Schritt eines Programms. Daher soll an dieser Stelle eine

genauere Laufzeitabschätzung erfolgen.

Seien k die Anzahl der Kandidatenallele und n die Anzahl der Allele, die bei gegebener Ploidie nach `get_max_parsimony_n_alleles()` zu erwarten sind, und es gilt $n, k \in \mathbb{N}$. Wie bereits in Kap. 3.4.1 beschrieben, lässt sich die Anzahl aller Kombinationen mit Zurücklegen aus dem Binomialkoeffizienten nach Formel (3-1) errechnen [57]. Das bedeutet, dass für jede Zusammenhangskomponente $\binom{n+k-1}{k}$ Kombinationen jeweils in O(1) gebildet werden müssen. Die Laufzeit zum Erzeugen aller Kombinationen mit Zurücklegen entspricht somit genau ihrer Anzahl, so dass je Zusammenhangskomponente eine Laufzeit von $O(\binom{n+k-1}{k})$ benötigt wird. Ebenso wird jede spätere Iteration über der Menge der aus den Allelkombinationen gebildeten Allele-Fractions die Laufzeit um den Faktor $O(\binom{n+k-1}{k})$ erhöhen. Daher soll diese Laufzeit hinsichtlich ihrer oberen Schranken im Folgenden genauer abgeschätzt werden.

Die allgemeine Formel des Binomialkoeffizienten (4-4) beschreibt die Anzahl aller Kombinationen ohne Zurücklegen.

$$\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!} \tag{4-4}$$

Aus ihr kann die hier benötigte Anzahl aller Kombinationen mit Zurücklegen, also $\binom{n+k-1}{k}$, abgeleitet werden, wenn gilt $m = n + k - 1$:

$$\begin{aligned} \binom{m}{k} &= \frac{m!}{(m-k)! \cdot k!} = \frac{(n+k-1)!}{((n+k-1)-k)! \cdot k!} \\ &= \frac{(n+k-1)!}{(n-1)! \cdot k!} = \binom{n+k-1}{k} \end{aligned} \tag{4-5}$$

Mit anderen Worten, Kombinationen mit Zurücklegen können 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 für eine erste Laufzeitabschätzung $m = n + k - 1$, dann gilt:

$$\binom{m}{k} = \frac{m!}{(m-k)! \cdot k!} \leq m! \tag{4-6}$$

Daraus ergibt sich nach (4-7) eine obere Schranke der Laufzeit von $O((n+k)!)$.

$$O(m!) = O((n+k+1)!) = O((n+k)!) \tag{4-7}$$

Hierfür soll nun eine kleinere obere Schranke gefunden werden. Allgemein lässt sich die Fakultät näherungsweise durch die Stirlingsche Formel [58] abschätzen:

$$n! \approx \left(\frac{n}{e}\right)^n \cdot \sqrt{2 \cdot \pi \cdot n} \tag{4-8}$$

Daraus folgt, dass (4-9) eine untere Schranke der Fakultät ist [62].

$$n! \geq \left(\frac{n}{e}\right)^n \tag{4-9}$$

Durch die Formeln (3-1) und (4-9) 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 [62]:

$$\begin{aligned}
\binom{n+k-1}{k} &\overset{(3\text{-}1)}{=} \frac{(n+k-1)!}{(n-1)!\cdot k!} = \frac{\prod_{i=1}^{n+k-1} i}{\prod_{i=1}^{n-1} i \cdot \prod_{i=1}^{k} i} \\
&= \frac{\prod_{i=n}^{n+k-1} i \cdot \prod_{i=1}^{n-1} i}{\prod_{i=1}^{n-1} i \cdot \prod_{i=1}^{k} i} = \frac{\prod_{i=n}^{n+k-1} i}{\prod_{i=1}^{k} i} \\
&= \frac{(n+k-1)\cdot(n+k-2)\cdot\ \ldots\ \cdot n}{k!} \\
&\leq \frac{(n+k-1)^k}{k!} \\
&\overset{(4\text{-}9)}{\leq} \frac{(n+k-1)^k}{\left(\frac{k}{e}\right)^k} \\
&= \left(\frac{e\cdot(n+k-1)}{k}\right)^k
\end{aligned} \tag{4-10}$$

Diese obere Laufzeitschranke kann nun verwendet werden, um die Laufzeit im Worst Case zu bestimmen. Dazu soll zunächst der Worst Case selbst ermittelt werden. Da die Laufzeit der Anzahl der Kombinationen und damit dem Binomialkoefizienten entspricht, muss also ein möglichst großer Binomialkoeffizient gefunden werden, d.h. der Zusammenhang von n und k zueinander, so dass $\binom{n}{k}$ maximal wird.

Der Binomialkoeffizient verfügt über bestimmte Eigenschaften, insbesondere gelten der Symmetrie- (4-11) und der Additionssatz (4-12), die sich auch in der Struktur des Pascalschen Dreiecks widerspiegeln [58].

$$\binom{n}{k} = \binom{n}{n-k} \tag{4-11}$$

$$\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1} \tag{4-12}$$

Aufgrund der Symmetrie sind die Binomialkoeffizienten jeder Zeile des Dreiecks $\binom{n}{k_i}$ mit $k_i \in 1,\ldots n$ und $i \in \mathbb{N}$ symmetrisch im Bezug auf die Zeilenmitte. Und da nach (4-12) 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 $k_i = \frac{n}{2}$ maximal ist.

Daher soll $k = \frac{n}{2}$ als Worst Case angenommen werden, dann ergibt sich aus (4-10) eine Laufzeit von $O(3e^{\frac{n}{2}}) \in O(e^n)$.

$$\begin{aligned}
\left(\frac{e \cdot (n+k-1)}{k}\right)^k &= \left(\frac{e \cdot (n+\frac{n}{2}-1)}{\frac{n}{2}}\right)^{\frac{n}{2}} \\
&= \left(\frac{\frac{3en-2e}{2}}{\frac{n}{2}}\right)^{\frac{n}{2}} \\
&= \left(\frac{3en-2e}{n}\right)^{\frac{n}{2}} \\
&\leq \left(\frac{3en}{n}\right)^{\frac{n}{2}} \\
&= 3e^{\frac{n}{2}} \\
&\in O(e^n)
\end{aligned} \tag{4-13}$$

Hinsichtlich der Allelkombinationen entspricht dies einer Zeit von $O(e^n)$ für jede Zusammenhangskomponente.

Für sehr große Cluster und Cluster mit vielen Varianten 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 bereits beschriebenen Schwellenwerte eingeführt, die ober- und unterhalb einer festgelegten Clustergröße nur Kandidatenallele ab einer bestimmten Häufigkeit ihrer Sequenz in den Reads berücksichtigen (vgl. Kap. 3.4.1).

Bei der Berechnung der Allele-Fractions wird für jede Allelkombination die Häufigkeit der Kandidatenallele im Bezug auf die Allelkombination ermittelt. Es werden also alle n Positionen jeder Kombination betrachtet, so dass die Laufzeit der Funktion `get_candidate_vafs()` insgesamt $O(n \cdot e^n)$ beträgt.

Laufzeit der Funktion `get_alignment_likelihood()`

Für die Berechnung der Likelihood eines Read-Kandidatenallel-Paares wird die geschätzte Fehlerrate p_{query} jeder Base des Reads verwendet. Durch die Verwendung der CIGAR-Tupel werden die Likelihoodberechnungen meist gebündelt für mehrere Basen mit gleicher Operation (Match, Substitution, Insertion, Deletion) durchgeführt. Dennoch müssen vorab für alle Basen des Reads die Werte von p_{query} aus den Phred Quality Scores nach (3-3)

ermittelt werden. Daher kann die Laufzeit für die Likelihoodberechnung eines Reads über die durchschnittliche Readlänge l geschätzt werden und beträgt somit $O(l)$.

Laufzeiten der Funktionen `calc_vafs_likelihood()`, `calc_vafs_likelihood_read()` und `get_allele_likelihood_read()`

Durch die Funktion `calc_vafs_likelihood()` wird über alle Reads in $O(|V_{C_i}|)$ und durch `calc_vafs_likelihood_read()` über alle Kandidatenallele iteriert. Es gilt $n \geq k$, da die Anzahl der Kandidatenallele maximal so groß sein kann wie die tatsächliche Anzahl der Allele nach Berücksichtigung der Ploidie, dadurch kann für `calc_vafs_likelihood_read()` die obere Laufzeitschranke mit $O(n)$ veranschlagt werden. Die aus den beiden Funktionen generierten paarweisen Kombinationen der Kandidatenallele mit allen Reads werden an `get_allele_likelihood_read()` weitergereicht.

Dort wird zunächst das angelegte Dictionary nach bereits vorhandenen Einträgen durchsucht. Da hier maximal ein Eintrag für jedes Kandidatenallel zu jedem Read vorliegen kann, benötigt die Suche $O(n \cdot |V_{C_i}|)$ [63].

Liegt kein Eintrag vor, so werden zunächst die ausgehenden Nachbarn durchsucht, und bei erfolgloser Suche, werden im Anschluss die eingehenden Nachbarn betrachtet. Die Anzahl der maximalen Suchvorgänge entspricht also der Summe der aus- und eingehenden Kanten des Reads. Somit entspricht die Laufzeit hierfür seinem Knotengrad d. Im Worst Case wäre die Zusammenhangskomponente ein vollständiger gerichteter Graph, so dass $d = |V_{C_i}| - 1$. Somit beträgt die Laufzeit für die Suchvorgänge maximal $O(|V_{C_i}|)$.

Die Suchvorgänge aus `get_allele_likelihood_read()` werden genau so oft ausgeführt, bis alle Einträge im Dictionary erzeugt wurden, also $n \cdot |V_{C_i}|$ Mal. Für jeden Eintrag ins Dictionary wird dann die Likelihood in $O(l)$ errechnet. Daraus ergibt sich schließlich eine Laufzeit für die Suchvorgänge und die Likelihoodberechnungen von $O(n \cdot l \cdot |V_{C_i}|^2)$ (siehe Formel (4-14)). Hiernach wird nur noch das Dictionary in $O(n \cdot |V_{C_i}|)$ durchsucht, wodurch sich die Effizienz der Funktion deutlich erhöht.

$$O(n \cdot |V_{C_i}|) \cdot O(|V_{C_i}|) \cdot O(l) = O(n \cdot l \cdot |V_{C_i}|^2) \tag{4-14}$$

Laufzeit der Bestimmung der Likelihood aller Allele-Fractions

Nach der Berechnung der Allele-Fractions in $O(n \cdot e^n)$ werden die zuvor beschriebenen Likelihoodberechnungen der Read-Allel-Paare in `noderad_main.py` für alle Allele-Fractions ausgeführt. Die Likelihoodberechnungen werden also insgesamt e^n Mal wiederholt. Dabei kann nun zwischen der deutlich günstigeren Dictionary-Suche und den Suchvorgängen über

alle Nachbarknoten mit Likelihoodberechnung in `get_allele_likelihood_read()` differenziert werden. Wie bereits beschrieben wird die aufwändige Suche über die Nachbarn nur $O(n \cdot |V_{C_i}|)$ Mal ausgeführt bis das Dictionary vollständig ist. Im Anschluss wird nur noch das Dictionary durchsucht. Über alle Allele-Fractions ergibt sich unter Berücksichtigung dieses Aspekts nach Formel (4-15) eine Laufzeit von $O(e^n \cdot n \cdot |V_{C_i}| + l \cdot n^2 \cdot |V_{C_i}|^3)$ für die Likelihoodberechnung über alle Allele-Fractions.

$$\begin{aligned} O(n \cdot e^n) + O(e^n & - n \cdot |V_{C_i}|) \cdot O(n \cdot |V_{C_i}|) + O(n \cdot |V_{C_i}|) \cdot O(n \cdot l \cdot |V_{C_i}|^2) \\ &= O(e^n \cdot n + e^n \cdot n \cdot |V_{C_i}| - n^2 \cdot |V_{C_i}|^2 + l \cdot n^2 \cdot |V_{C_i}|^3) \\ &= O(e^n \cdot n \cdot (1 + |V_{C_i}|) - n^2 \cdot |V_{C_i}|^2 + l \cdot n^2 \cdot |V_{C_i}|^3) \\ &\in O(e^n \cdot n \cdot |V_{C_i}| - n^2 \cdot |V_{C_i}|^2 + l \cdot n^2 \cdot |V_{C_i}|^3) \\ &= O(e^n \cdot n \cdot |V_{C_i}| + n^2 \cdot |V_{C_i}|^2 \cdot (l \cdot |V_{C_i}| - 1)) \\ &\in O(e^n \cdot n \cdot |V_{C_i}| + l \cdot n^2 \cdot |V_{C_i}|^3) \end{aligned} \tag{4-15}$$

Bei der anschließenden Maximumssuche addiert sich zur Laufzeit noch eine Iteration über alle Allele-Fractions. Die Laufzeit ändert sich hierdurch nicht, da bereits $e^n \cdot n \cdot |V_{C_i}|$ dominiert (siehe Formel (4-16)). Somit erfordert die Bestimmung der Allele-Fraction mit maximaler Likelihood insgesamt eine Laufzeit von $O(e^n \cdot n \cdot |V_{C_i}| + l \cdot n^2 \cdot |V_{C_i}|^3)$.

$$\begin{aligned} O(n \cdot |V_{C_i}| & \cdot (e^n + l \cdot n)) + O(e^n) \\ &= O(e^n \cdot n \cdot |V_{C_i}| + l \cdot n^2 \cdot |V_{C_i}|^3 + e^n) \\ &= O(e^n \cdot (n \cdot |V_{C_i}| + 1) + l \cdot n^2 \cdot |V_{C_i}|^3) \\ &\in O(e^n \cdot n \cdot |V_{C_i}| + l \cdot n^2 \cdot |V_{C_i}|^3) \end{aligned} \tag{4-16}$$

Im Worst Case entspricht die Anzahl der Kandidatenallele genau der Anzahl der Reads in der Zusammenhangskomponente und es gilt $n = |V_{C_i}|$, so dass die Laufzeit dann auf $O(e^{|V_{C_i}|} \cdot |V_{C_i}|^2 + l \cdot |V_{C_i}|^5)$ geschätzt werden kann (Formel (4-17)).

$$\begin{aligned} & O(e^n \cdot n \cdot |V_{C_i}| + l \cdot n^2 \cdot |V_{C_i}|^3) \\ &= O(e^{|V_{C_i}|} \cdot |V_{C_i}|^2 + l \cdot |V_{C_i}|^5) \end{aligned} \tag{4-17}$$

Die Laufzeit wird also durch die Anzahl der Allel-Fractions dominiert und kann bei großen Clustern und bei Clustern mit vielen Kandidatenallelen problematisch werden. Die konfigurierbaren Schwellenwerte `treshold-cluster-size` sowie `large-clusters` und `small-clusters` aus `treshold-seq-noise` ermöglichen eine Anpassung solcher Cluster, um die Anzahl der Kombinationen und damit den Rechenaufwand flexibel reduzieren zu können.

4.3 Laufzeit zur Bestimmung der Loci mit maximaler Likelihood

Laufzeit der Funktion `get_candidate_loci()`

Wie bereits in Kap. 3.5.1 besprochen, wird die Indikatorfunktion nur für eine Allele-Fraction, die Allele-Fraction mit maximaler Likelihood, erfüllt. Diese wird, um die Kandidaten-Loci zu ermitteln, in ihre ursprüngliche Allelkombination mit Wiederholungen zurückgerechnet. Dies erfordert eine Iteration über die Länge einer Allele-Fraction. Sei k weiterhin die Anzahl der Kandidatenallele und somit die Länge einer Allele-Fraction und sei n die Anzahl der zu erwartenden Allele und somit die Länge einer Allelkombination (vgl. Kap. 4.2). Dann gilt nach Formel (2-8), dass die Iteration über die Länge einer Allele-Fraction maximal eine Laufzeit von $O(k) \in O(n)$ erfordert, da $k \leq n$. Die anschließende Konstruktion der Allelkombination der Länge n aus der Allele-Fraction erfolgt in $O(n)$. Wie in Satz 1 bewiesen wurde, genügt es über dieser einen Allelkombination mit maximaler Likelihood die Permutationen zu bilden. Die Permutationen entsprechen dann allen Loci-Kombinationen, welche die Indikatorfunktion erfüllen.

Es werden also für eine Kombination der Länge n alle $n!$ Permutationen gebildet, die Laufzeit hierfür beträgt somit $O(n!)$, da `itertools.permutations` auch in der Kombination wiederholt vorkommende Allele als jeweils eigenständige Items betrachtet [59]. Dadurch entstehen Duplikate. Für die anschließende Gruppierung der Permutationen zu Loci-Kombinationen, muss jede Permutation entsprechend der gegebenen Ploidie ϕ in Loci aufgeteilt werden, so dass hier über die Länge der Permutation iteriert wird. Dies benötigt somit eine Laufzeit von $O(n \cdot n!)$. Hiernach werden zwei Sortiervorgänge ausgeführt. Einmal innerhalb der Loci, so dass sich die Laufzeit um den Faktor $O\left(\frac{n}{\phi} \cdot \log\left(\frac{n}{\phi}\right)\right)$ erhöht und anschließend über die Loci innerhalb der Loci-Kombination in $O(\phi \cdot \log \phi)$ [63]. Da $\phi = const.$ kann für die Bildung der Permutationen, ihre Gruppierung und Sortierung nach Formel (4-18) insgesamt eine Laufzeit von $O\left(n! \cdot \log n\right)$ veranschlagt werden.

$$
\begin{aligned}
O\left(n \cdot n! \cdot \frac{n}{\phi} \cdot \log\left(\frac{n}{\phi}\right) \cdot \phi \cdot \log \phi\right) &= O\left(n! \cdot n^2 \cdot \log\left(\frac{n}{\phi}\right) \cdot \log(\phi)\right) \\
&\overset{\phi=const.}{=} O\left(n! \cdot n^2 \cdot \log n\right) \\
&\leq O((n+2) \cdot (n+1) \cdot n! \cdot \log n) \\
&= O((n+2)! \cdot \log n) \\
&\in O(n! \cdot \log n)
\end{aligned}
\tag{4-18}
$$

Das Entfernen der Duplikate ist in Python durch die `set()` Operation effizient über Hash-Tabellen möglich [63]. Hierfür wird jede Permutation nur einmal betrachtet, so dass

die Duplikatentfernung in $O(n!)$ erfolgen kann.

Insgesamt ergibt sich also für die Generierung der Loci-Kombinationen nach Formel (4-19) eine Laufzeit von $O(n! \cdot \log n)$.

$$\begin{aligned} O(n) + O(n) \; &+ O(n!) + O(n! \cdot \log n) + O(n!) \\ &= O(2n + 2n! + n! \cdot \log n) \\ &\in O(n + n! + n! \cdot \log n) \\ &\in O(n! \cdot \log n) \end{aligned} \tag{4-19}$$

Nach dem zugrundeliegenden Modell hätten für die Loci-Kombinationen alle Permutationen über allen Allelkombinationen erzeugt werden müssen. Dadurch wären insgesamt $(e^n)!$ Permutationen entstanden. Für jede dieser Permutationen hätte die Bedingung der Indikatorfunktion geprüft werden müssen. Es hätte somit eine Laufzeit von mindestens $\Omega((e^n)!)$ dominiert. Durch die Implementierung auf Grundlage von Satz 1 konnte somit die Laufzeit $O(n! \cdot \log n)$ deutlich reduziert werden. Da hier nur noch $n!$ Permutationen erzeugt werden ist auch der Speicherplatzbedarf wesentlich geringer.

Laufzeit der Funktion `get_cigar_tuples()`

Die Suchvorgänge für die beiden Knotenmengen R_{source} und R_{target} (Kap. 3.5.2) benötigen in graph-tool jeweils $O(|V_{C_i}|)$ [61]. Anschließend wird eine Vorwärts- oder Rückwärtskante zwischen beiden Knotenmengen gesucht, d.h. im Worst Case muss für jeden Knoten $v \in R_{source}$ mit jedem Knoten aus $w \in R_{target}$ ein zweifacher Kantenaufruf erfolgen. Die Laufzeit eines Kantenaufrufs hängt in graph-tool vom Knotengrad d der beteiligten Knoten ab. Seien $d(v)$ und $d(w)$ die Knotengrade der Knoten v und w, so beträgt die Laufzeit für einen Kantenaufruf zwischen ihnen $O(\min(d(v), d(w)))$ [56].

Seien im Worst Case die Hälfte aller Knoten der Zusammenhangskomponente in R_{source} und die andere Hälfte bis auf einen Knoten u in R_{target}. Seien R_{source} und R_{target} jeweils vollständige gerichtete Subgraphen, die jeweils nur eine Kante zu u besitzen. Dadurch sind R_{source} und R_{target} allein über u mit einander verbunden und da $u \notin R_{source}$ und $u \notin R_{target}$, existiert keine direkte Kante, die R_{source} und R_{target} mit einander verbindet. In diesem Falle müssen also für alle $\frac{|V_{C_i}|}{2}$ Knoten aus R_{source} mit jeweils allen $\frac{|V_{C_i}|}{2} - 1$ Knoten aus R_{target} die Kantenaufrufe ausgeführt werden. Da R_{source} und R_{target} ohne den Knoten u zwei jeweils von einander unabhängige vollständige Graphen sind, beträgt der Knotengrad bei jedem Kantenaufruf für $d(v) = \frac{|V_{C_i}|}{2}$ und für $d(w) = \frac{|V_{C_i}|}{2} - 1$. Nach Formel (4-20) beträgt daher die Laufzeit von `get_cigar_tuples()` im Worst Case $O(|V_{C_i}|^3)$.

$$
\begin{aligned}
O(|V_{C_i}|) + O(|V_{C_i}|) \; &+ O\left(\frac{|V_{C_i}|}{2} \cdot \left(\frac{|V_{C_i}|}{2} - 1\right) \cdot \min\left(\frac{|V_{C_i}|}{2}, \frac{|V_{C_i}|}{2} - 1\right)\right) \\
&= O\left(2 \cdot |V_{C_i}| + \left(\frac{|V_{C_i}|}{2}\right)^2 \cdot \left(\frac{|V_{C_i}|}{2} - 1\right)\right) \\
&\in O\left(|V_{C_i}| + \left(\frac{|V_{C_i}|}{2}\right)^3\right) \\
&\in O(|V_{C_i}|^3)
\end{aligned}
\tag{4-20}
$$

Laufzeiten der Funktionen `indicator_constraint()`, `calc_loci_likelihoods()`, `get_allele_likelihoods_allele()` und `get_heterozygosity()` und Bestimmung der Gesamtlaufzeit der Loci-Zuordnung

Die Prüfung der Bedingung der Indikatorfunktion erfolgt über alle Positionen einer Loci-Kombination in $O(n)$. Da aber nach Satz 1 diese Prüfung nicht mehr notwendig ist, soll ihre Laufzeit in der weiteren Laufzeitanalyse keine Berücksichtigung finden.

Die eigentliche Likelihoodberechnung erfolgt schließlich über die Readlänge der Allelpaare jedes Locus aller $n!$ Loci-Kombination der Allele-Fraction mit maximaler Likelihood. Wie schon bei den Allele-Fractions werden die Ergebnisse der Likelihoodberechnungen in einem Dictionary abgelegt. Dadurch müssen von allen $n!$ Loci-Kombinationen nur für $\binom{n}{2}$ Allelkombinationen die Likelihoodberechnungen mit Ermittlung der CIGAR-Tupel über `get_cigar_tuples()` durchgeführt. Für alle übrigen Loci-Kombinationen können dann die Werte dem Dictionary entnommen werden.

Zunächst kann der Laufzeitfaktor für die Allelkombinationen, also $\binom{n}{2}$, nach Formel (4-21) durch $O(n^2)$ abgeschätzt werden.

$$
\begin{aligned}
O\left(\binom{n}{2}\right) &= O\left(\frac{n!}{(n-2)! \cdot 2!}\right) \\
&= \frac{n \cdot (n-1) \cdot (n-2)!}{(n-2)! \cdot 2!} \\
&= \frac{n \cdot (n-1)}{2} \\
&\in O(n^2)
\end{aligned}
\tag{4-21}
$$

Für die $\binom{n}{2}$ Berechnungen mit Ermittlung der CIGAR-Tupel und Likelihoodberechnung ergibt sich nach (4-22) eine Laufzeit von $O(l \cdot n^5 \cdot |V_{C_i}|^3)$.

$$\begin{aligned} & O\left(\binom{n}{2}\right) \cdot O\left(\frac{n}{\phi}\right) \cdot O(|V_{C_i}|^3) \cdot O(l) \\ & \overset{\phi=const.}{=} O\left(\binom{n}{2} \cdot n \cdot |V_{C_i}|^3 \cdot l\right) \\ & \overset{(4\text{-}21)}{=} O(n^2 \cdot n \cdot |V_{C_i}|^3 \cdot l) \\ & = O(l \cdot n^3 \cdot |V_{C_i}|^3) \end{aligned} \tag{4-22}$$

Für die übrigen $n! - \binom{n}{2}$ Berechnungen kann der Eintrag dem Dictionary entnommen werden, so dass sich nach (4-23) eine Laufzeit von $O(n^3 \cdot n!)$ ergibt.

$$\begin{aligned} & O\left(n! - \binom{n}{2}\right) \cdot O\left(\frac{n}{\phi}\right) \\ & \overset{\phi=const.}{=} O\left(\left(n! - \binom{n}{2}\right) \cdot n\right) \\ & \overset{(4\text{-}21)}{=} O((n! - n^2) \cdot n) \\ & = O(n \cdot n! - n^3) \\ & \in O(n \cdot n!) \end{aligned} \tag{4-23}$$

Die Gesamtlaufzeit für die Loci-Zuordnung ergibt sich aus der Summe der Laufzeiten beider Berechnungsvarianten sowie der Bestimmung der Loci-Kombinationen und der Maximumsbestimmung über alle $n!$ Loci-Kombinationen mit $O(l \cdot n^5 \cdot |V_{C_i}|^3 + n^3 \cdot n!)$.

$$\begin{aligned} & O(l \cdot n^3 \cdot |V_{C_i}|^3) + O(n \cdot n!) + O(n! \cdot \log n) + O(n!) \\ & \in O(l \cdot n^3 \cdot |V_{C_i}|^3 + n \cdot n!) \end{aligned} \tag{4-24}$$

Auch hier soll abschließend der Worst Case betrachtet werden, bei dem die Anzahl der Kandidatenallele genau der Anzahl der Reads in der Zusammenhangskomponente entspricht, also $n = |V_{C_i}|$. Dann ergibt sich für die Berechnung der wahrscheinlichsten Loci-Zuordnung nach (4-25) eine maximale Laufzeit von $O(l \cdot |V_{C_i}|^8 + |V_{C_i}|^3 \cdot |V_{C_i}|!)$.

$$\begin{aligned} & O(l \cdot n^3 \cdot |V_{C_i}|^3 + n \cdot n!) \\ & = O(l \cdot |V_{C_i}|^6 + |V_{C_i}| \cdot |V_{C_i}|!) \end{aligned} \tag{4-25}$$

Die Anzahl der Permutationen dominiert durch $O(|V_{C_i}|!)$ die Laufzeit noch stärker als bei der Ermittlung der Allelkombinationen in Kap. 4.2. Dies ist für große Cluster bzw. für Cluster mit vielen Kandidatenallelen sehr problematisch. Die Anpassung der Schwellenwerte für `treshold-cluster-size` und `threshold-seq-noise` können dies zwar in einem gewissen Rahmen für den Prototypen kompensieren. Bei der späteren Implementierung in Rust sollte hier die Anspassung der Schwellenwerte weiter optimiert werden (vgl. Kap. 6).

4.4 Abschließende Laufzeitanalyse des gesamten Algorithmus

Für jeden Konstruktions- und Berechnungsschritt wurden die Laufzeiten bereits an entsprechender Stelle angesprochen. Zusammenfassend ergab sich dabei für die einmalige Konstruktion des Graphen und die Extraktion seiner Zusammenhangskomponenten C eine Laufzeit von $O(|V| \cdot (|V| + |E|))$ (Kap. 4.1). Für jede Zusammenhangskomponente erfolgte die Berechnung der wahrscheinlichsten Allelkombination in $O(e^n \cdot n \cdot |V_{C_i}| + l \cdot n^2 \cdot |V_{C_i}|^3)$ (Kap. 4.2) und die Berechnung der wahrscheinlichsten Loci-Zuordnung in $O(l \cdot n^5 \cdot |V_{C_i}|^3 + n^3 \cdot n!)$ (Kap. 4.3). Die durchschnittliche Readlänge umfasst meist nur wenige hundert Basen, sie kann insbesondere bei großen Datensätzen als konstant betrachtet werden. Dadurch lässt sich die Gesamtlaufzeit von NodeRAD wie folgt zusammenfassen:

$$
\begin{aligned}
& O(V \cdot (V+E)) + O\left(\sum_{i=1}^{|C|} e^n \cdot n \cdot |V_{C_i}| + l \cdot n^2 \cdot |V_{C_i}|^3) + l \cdot n^3 \cdot |V_{C_i}|^3 + n \cdot n!\right) \\
& \in O\left(V \cdot (V+E) + \sum_{i=1}^{|C|} e^n \cdot n \cdot |V_{C_i}| + l \cdot n^3 \cdot |V_{C_i}|^3 + n \cdot n!\right) \\
& \overset{l=const.}{=} O\left(V \cdot (V+E) + \sum_{i=1}^{|C|} e^n \cdot n \cdot |V_{C_i}| + n^3 \cdot |V_{C_i}|^3 + n \cdot n!\right) \\
& = O\left(V \cdot (V+E) + \sum_{i=1}^{|C|} e^n \cdot n \cdot |V_{C_i}| + \sum_{i=1}^{|C|} n^3 \cdot |V_{C_i}|^3 + \sum_{i=1}^{|C|} n \cdot n!\right) \\
& \in O\left(\sum_{i=1}^{|C|} e^n \cdot n \cdot |V_{C_i}| + \sum_{i=1}^{|C|} n \cdot n!\right) \\
& \in O\left(\sum_{i=1}^{|C|} n \cdot n!\right)
\end{aligned}
\tag{4-26}
$$

Würden im Worst Case alle Reads in allen Zusammenhangskomponenten aufgrund unterschiedlicher Sequenzen als Kandidatenallele gewertet werden, also $n = |V_{C_i}|$ für alle Komponenten C_i, dann ergäbe sich nach Formel (4-27) eine Laufzeit von $O(|V|!)$.

$$
\begin{aligned}
O\left(\sum_{i=1}^{|C|} n \cdot n!\right) &= O\left(\sum_{i=1}^{|C|} |V_{C_i}| \cdot |V_{C_i}|!\right) \\
&= O(|V| \cdot |V|!) \in O(|V|!)
\end{aligned}
\tag{4-27}
$$

Ein solches Szenario ist jedoch sehr unwahrscheinlich. In allen übrigen Fällen wird die Laufzeit durch die Anzahl der Kandidatenallele der einzelnen Komponenten bestimmt, so dass die Optimierung der Schwellenwerte von zentraler Bedeutung für das Laufzeitverhalten ist. Zudem ist n meistens sehr klein und hängt in der Regel nicht von der Anzahl der Reads ab. Vielmehr hängt es von der Repetitivität des Genoms und von der Ploidie ab.

Kapitel 5

Evaluation an simulierten Datensätzen

5.1 Simulation und Workflow-Run

Zur Evaluation wurde durch das Tool ddRAGE [64, 65] ein simulierter Testdatensatz [66] von ddRADSeq-Daten für drei Individuen erzeugt. Der Testdatensatz umfasst 6965 single-end Reads mit 100 Basenpaaren Readlänge für 50 Loci und mit Mutationswahrscheinlichkeiten von 0.8999 für Substitutionen, 0.05 für Insertionen und 0.05 für Deletionen. Die höhere Rate bei der Mutationswahrscheinlichkeit für Substitutionen dient insbesondere dazu, auch bei einem relativ kleinen Testdatensatz SNPs zu simulieren. DdRAGE liefert neben verschiedenen zusätzlichen Informationen zur Simulation auch die Sequenzen der simulierten Loci. Zur Evaluation sollen diese mit den durch NodeRAD [33] identifizierten Loci verglichen werden.

Für die Analyse mit NodeRAD wurde eine Heterozygotiewahrscheinlichkeit von jeweils 0.01 für Substitutionen, Insertionen und Deletionen festgelegt. Die Sequenzierfehlerrate für Substitutionen L_{sub} wurde über die geschätzte Fehlerrate p_i der Basenqualität der Reads bestimmt (siehe Kap. 2.3.2). Für Indels wurden empirisch ermittelte Sequenzierfehlerraten der Illumina Sequenzierplattformen [45] verwendet. Diese in der Konfigurationsdatei festgelegten Konstanten sowie die angewendeten Schwellenwerte des Workflow-Runs sind in Tab. 5.1 aufgelistet.

Konfigurierte Schwellenwerte			
threshold max edit distance	**treshold-seq-noise**		**treshold-cluster-size**
	small-clusters	large-clusters	
9	2	4	300

Konfigurierte Konstanten			
	insertion	**deletion**	**substitution**
error-per-base	$2.8 \cdot 10^{-6}$	$5.1 \cdot 10^{-6}$	$L_i = \frac{1}{3} \cdot p_i$
heterozygosity	0.01	0.01	0.01

Tabelle 5.1: Loci-Zuordnung durch NodeRAD im Vergleich zu den simulierten Loci bei den Individuen A, B und C.

Für jedes der drei Individuen A, B und C wurden durch NodeRAD die wahrscheinlichsten Loci ermittelt und mit den Sequenzen der tatsächlich simulierten Loci aus den Zusatzinformationen von ddRAGE verglichen. Der Vergleich erfolgte durch das Tool BLAST (Basic Local Alignment Search Tool) [67]. Hierfür wurden sowohl die Loci der Simualtion als auch die Resultate des Workflows in das FASTA-Format geparsed. Für die simulierten Loci wurden dabei auch für jedes Individuum die jeweiligen Mutationen entsprechend dem Genotyp auf die Loci-Sequenzen gemapped.

Um die Daten lokal gegeneinander zu vergleichen, wurde anschließend für die Loci der Simulation eine BLAST-Datenbank erzeugt. Gegen diese Datenbank erfolgte dann die BLAST-Analyse mit den identifizierten Loci des Workflows. Aus den Ergebnissen der BLAST-Analyse wurden anschließend verschiedene Plots in der Programmiersprache R erzeugt, die Kap. 5.2 zu finden sind. Die Regeln und Scripte der Evaluation sind in NodeRAD integriert und können über die Konfigurationsdatei durch Angabe eines Pfades zu den durch ddRAGE simulierten Loci im yaml-Format aktiviert werden.

5.2 Ergebnisse

Erkennung der Allele

Die simulierten RADSeq-Daten zeigten in der FastQ-Analyse eine gute Qualität der Basen (Abbildung 5.1) und wiesen nach dem Trimming keine Adaptersequenzen mehr auf (Abbildung 5.2).

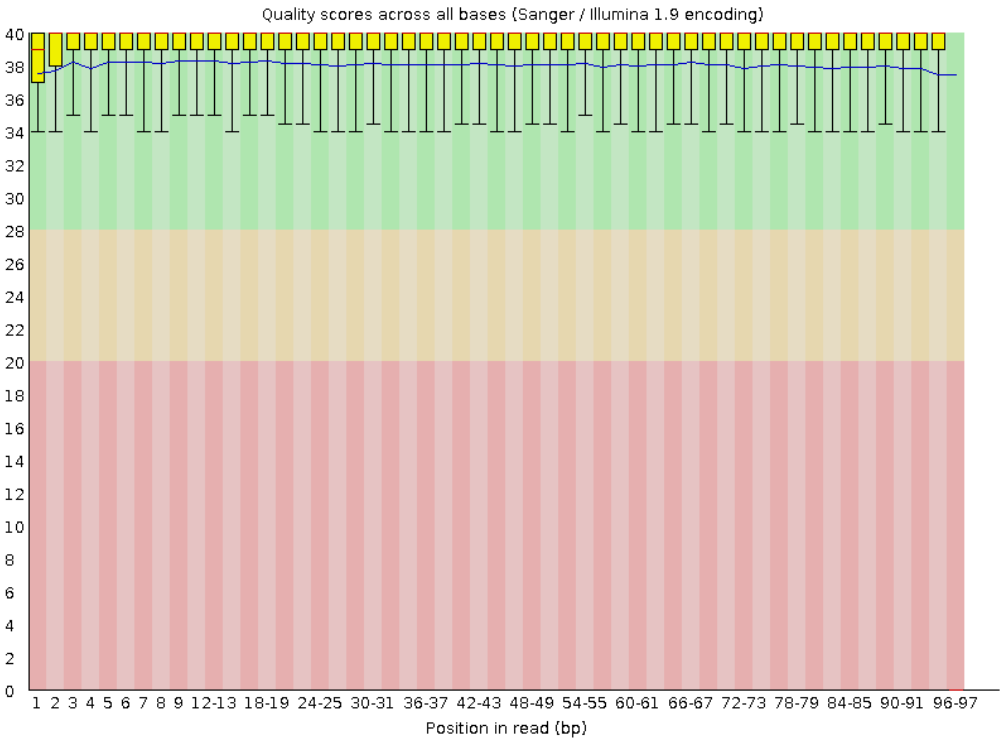

Abbildung 5.1: FastQ-Analyse: Basenqualität der Readsequenzen von Individuum A.

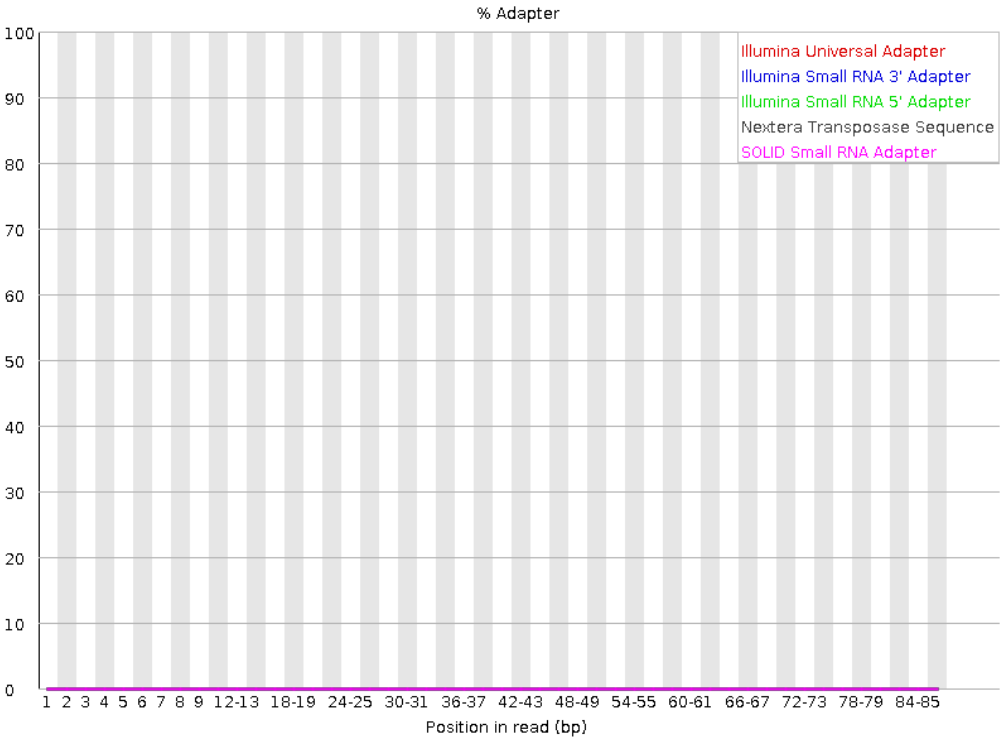

Abbildung 5.2: FastQ-Analyse: Adaptergehalt der Reads von Individuum A nach dem Trimming durch Cutadapt.

Bei den jeweils 50 Loci der ddRAGE-Simulation, kommen bei Individuum A eine homozygote und vier heterzygote Mutataionen durch Substitution vor. Individuum B weist an drei Loci homozygote und an einem Locus eine heterozygote Substitution auf. Individuum C besitzt drei homozygote Mutationen. Die BLAST-Analyse soll Aufschluss über die Genauigkeit geben, mit der NodeRAD in der Lage ist die Loci und die Varianten mit Mutationen zu finden. Für eine bessere Übersichtlichkeit werden in diesem Kapitel nur die Plots zur BLAST-Analyse von Individuum A exemplarisch dargestellt, da hier ein höherer Anteil heterozygoter Mutationen vorliegt. Sämtliche Plots für die Individuen B und C können jedoch im Anhang eingesehen werden (siehe Kap. A.1) und zeigen keine relevanten Abweichungen gegenüber den Plots von Individuum A.

Beim BLAST-Algorithmus werden die Nukleotidsequenzen der durch NodeRAD identifizierten Loci gegen die tatsächlich simulierten Loci paarweise verglichen [68]. Das resultierende Sequenzalignment gibt die besten Übereinstimmungen (Hits) an. Die Ähnlichkeit jedes Sequenzpaares wird über den prozentualen Anteil identischer Nukleotide ausgedrückt (Identität). Wie in Abbildung 5.3 erkennbar, sind die meisten Loci aus der Analyse mit NodeRAD identisch zu den tatsächlich simulierten Loci. Von insgesamt 62 durch NodeRAD identifizierten Allelen stimmen 49 Allele exakt mit den simulierten Allelen aus ddRAGE überein (siehe Abbildung 5.4). Bei denjenigen Allelen, die eine geringere Sequenzähnlichkeit aufweisen, handelt es sich um zusätzliche Allele, die NodeRAD zu dem betreffenden Locus identifiziert hat. Dort liegt ein Mismatch zur tatsächlichen und korrekt identifizierten Sequenz von maximal einer Base vor. Lediglich bei Individuum C wurde ein Locus mit einem Mismatch von einer Base falsch bestimmt ohne dass zusätzliche Allele mit der korrekten Locussequenz identifiziert wurden. Nur wenige Loci wurden durch NodeRAD gar nicht gefunden. Insgesamt wurden bei Individuum A 44 der 50 Loci gefunden, bei Individuum B konnten 48 und bei Individuum C 47 Loci identifiziert werden (siehe auch Tabelle 5.2). Über alle Individuen wurden also 92 % der Loci gefunden.

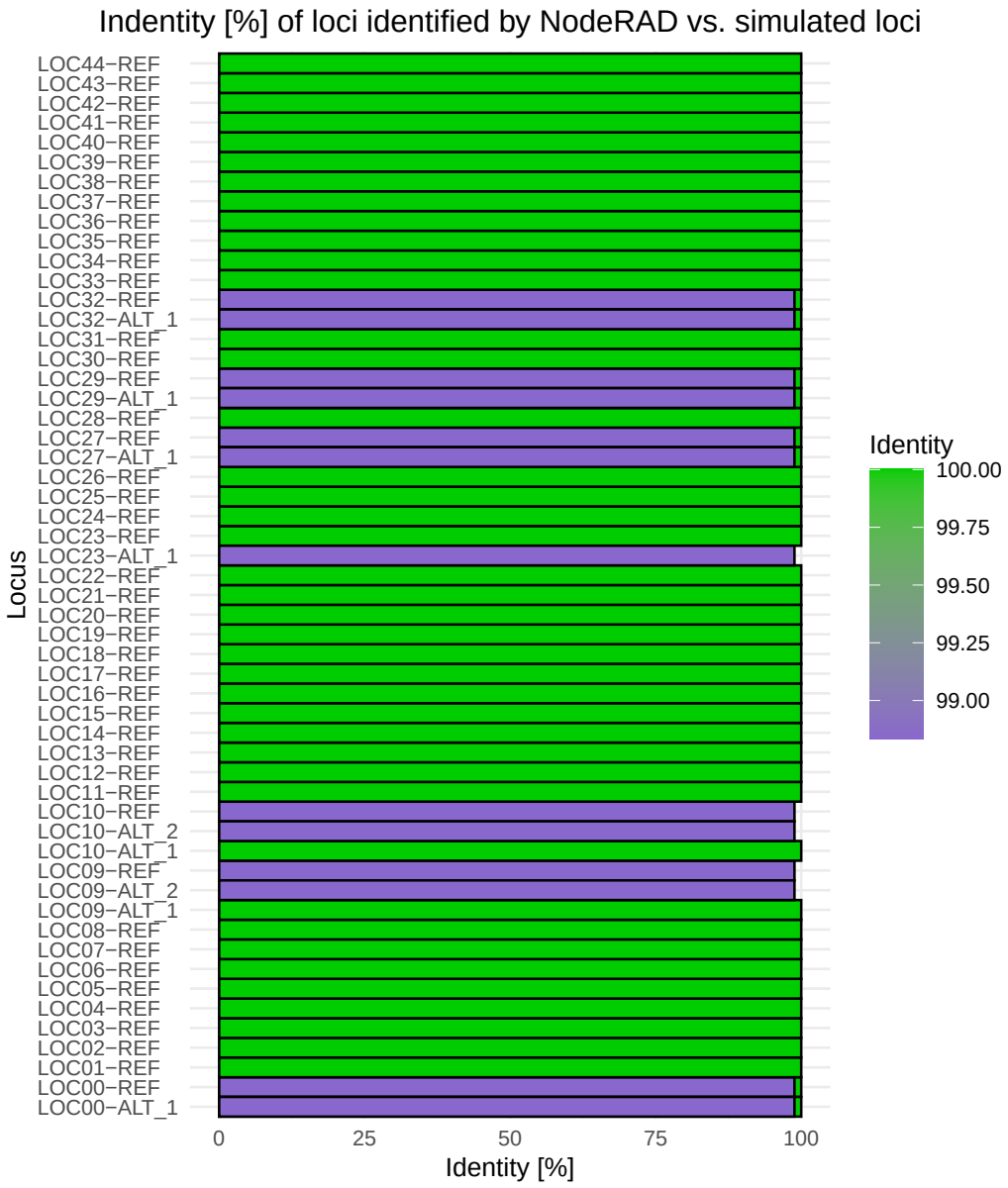

Abbildung 5.3: Individuum A: Sequenzähnlichkeit (Identität in %) der durch NodeRAD bestimmten Allele gegenüber den simulierten Allelen.

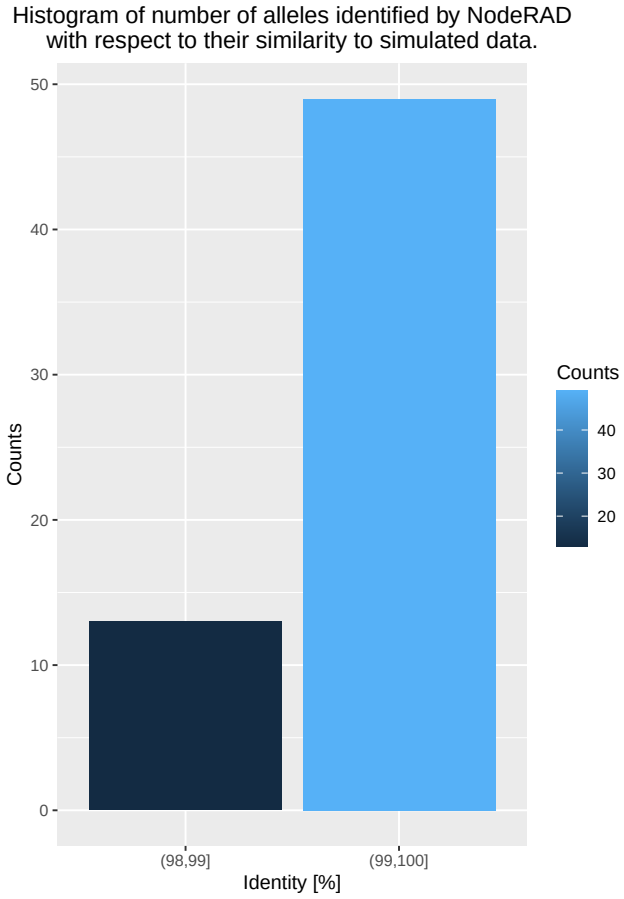

Abbildung 5.4: Individuum A: Anzahl der durch NodeRAD bestimmten Referenz- und Alternativallele (Spalten REF und ALT in der VCF-Datei) hinsichtlich der Sequenzähnlichkeit (Identität in %) zu den simulierten Allelen

Bei der BLAST-Analyse werden zudem beim Vergleich der Nukleotidsequenzen Werte für jedes Match bzw. Mismatch vergeben [68]. Die Summe dieser Werte stellt den Rohwert-Score des Sequenzaligments dar. Aus diesem Score lassen sich zwei statische Merkmale der BLAST-Analyse ableiten: der E-Value und der Bitscore.

Der E-Value (expectation value) ist ein Signifikanzmaß und beschreibt die Anzahl zufälliger Treffer mit mindestens dem betrachteten Score, die bei gegebener Größer der Referenzdatenbank zu erwarten wären [68]. Je kleiner der E-Value ist, desto geringer ist die Wahrscheinlichkeit, dass ein Hit nur zufällig gefunden wurde. Bei allen Individuen weisen die Hits eine hohe Signifikanz auf (siehe Abbildung 5.5). Lediglich bei einem Hit findet sich eine wesentlich geringere Signifikanz. Diese Abweichung betrifft bei allen Individuen den selben Locus der ddRAGE-simulierten Daten. Das korrespondierende durch NodeRAD bestimmte Allel ist deutlich verkürzt, so dass dieser Ausreißer am ehesten auf ein fehlerhaftes

Adaptertrimming zurückzuführen ist.

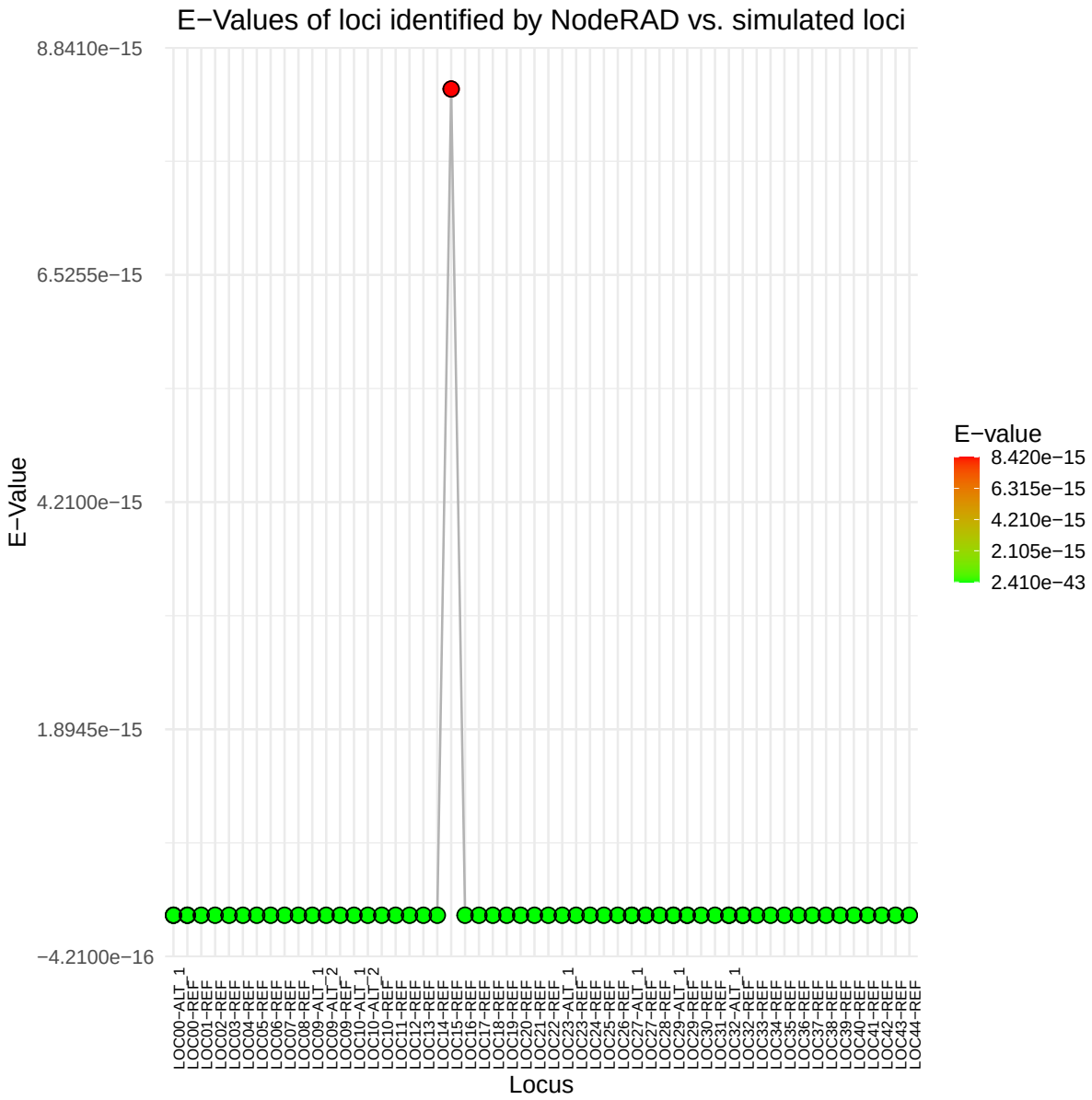

Abbildung 5.5: Individuum A: E-Values aus dem Vergleich der durch NodeRAD bestimmten Allele mit den simulierten Allelen

Der Bitscore ist der normalisierte Wert des Rohwert-Scores [68]. Durch die Normalisierung können die einzelnen Alignments mit einander hinsichtlich ihres Scores verglichen werden. Je höher Wert des Bitscores liegt, desto besser ist die Sequenzübereinstimmung. Auch hier weisen bis auf den bereits erwähnten Ausreißer mit verkürzter Sequenz sämtliche durch NodeRAD identifizierten Allele hohe Werte auf (siehe Abbildung 5.6).

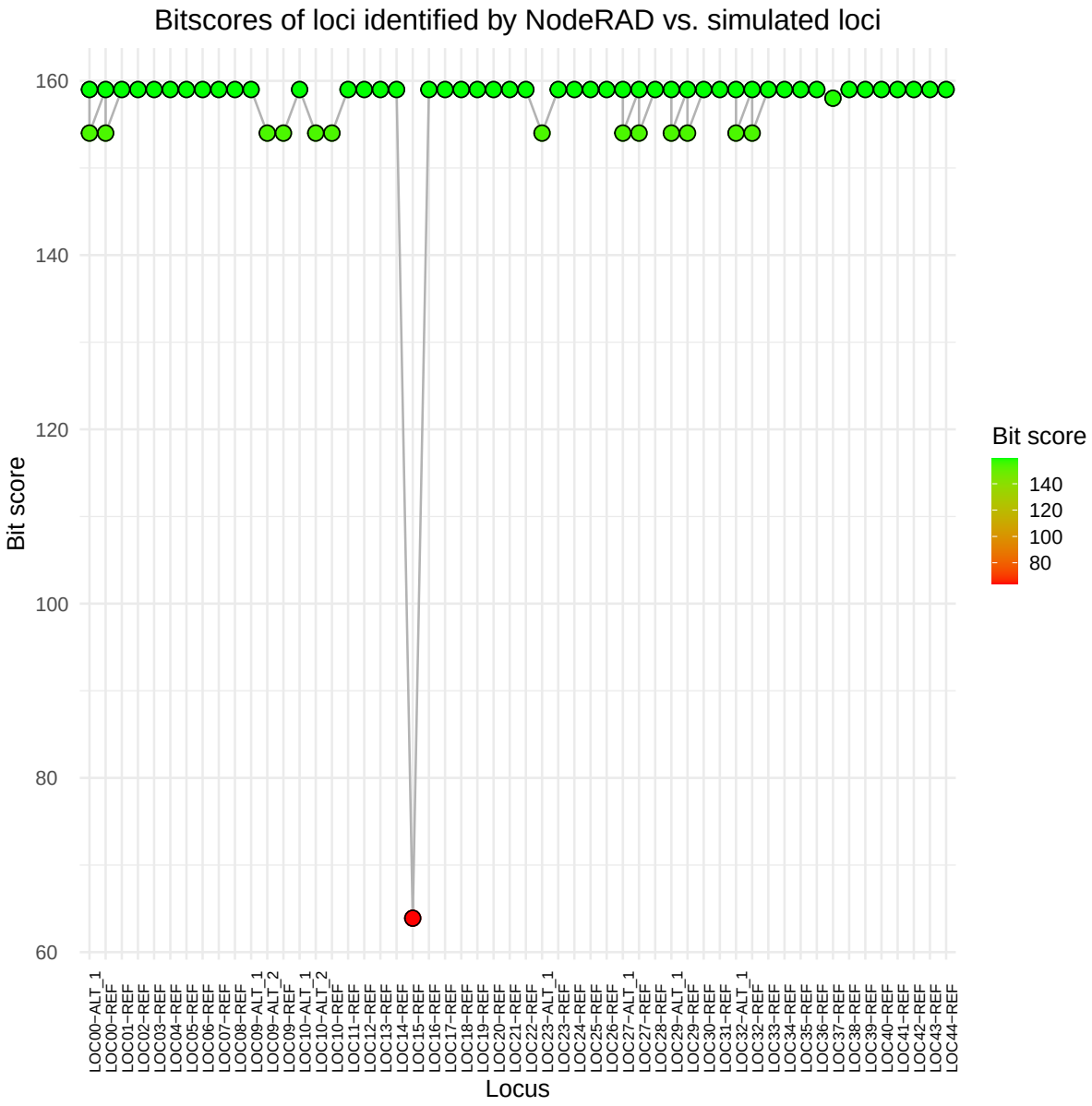

Abbildung 5.6: Individuum A: Bitscores aus dem Vergleich der durch NodeRAD bestimmten Allele mit den simulierten Allelen

Erkennung der Loci

Im Hinblick auf die Loci-Zuordnung durch NodeRAD wurden sämtliche homo- und heterozygoten Mutationen bei allen Individuen korrekt identifiziert. Bis auf einen falsch identifizierten Locus bei Individuum C und wenige fehlende Loci (siehe oben), wurden auch die meisten Allele der homozygoten Loci ohne Varianten gefunden. Hier zeigten sich allerdings auch einige Loci, bei denen NodeRAD einen heterozygoten Locus unter Beteiligung eines weiteren Allels definierte, obwohl der tatsächliche Locus homozygot ist. Bei diesen falsch-

heterozygoten Loci wiesen die zusätzlich dem Locus zugeordneten Allele ein Mismatch von jeweils einer Base auf. Insgesamt waren bei allen Individuen solche falsch-heterozygoten Loci detektiert worden. Falsch-homozygote Loci fanden sich dagegen nicht (siehe Tabelle 5.2).

Individuum A		**simulierte Loci** $n = 50$		
		homozygot	**heterozygot**	$\sum$
mit NodeRAD identifizierte Loci	**homozygot**	38	0	38
	heterozygot	2	4	6
	$\sum$	40	4	**44**

Individuum B		**simulierte Loci** $n = 50$		
		homozygot	**heterozygot**	$\sum$
mit NodeRAD identifizierte Loci	**homozygot**	45	0	45
	heterozygot	2	1	3
	$\sum$	47	1	**48**

Individuum C		**simulierte Loci** $n = 50$		
		homozygot	**heterozygot**	$\sum$
mit NodeRAD identifizierte Loci	**homozygot**	41	0	41
	heterozygot	6	0	6
	$\sum$	47	0	**47**

Tabelle 5.2: Übersicht über die Loci-Zuordnung durch NodeRAD im Vergleich zum tatsächlichen Genotyp der simulierten Daten.

Von insgesamt 139 Loci-Zuordnungen über alle Individuen wurden also 124 korrekt homozygot und 5 korrekt heterozygot identifiziert. Es fanden sich keine falsch-homozygoten, allerdings traten 10 falsch-heterozygote Zuordnungen auf. Daraus ergibt sich eine Rate falsch-homozygoter Loci-Zuordnungen von 0 %. Die Rate falsch-heterozygoter Loci-Zuordnungen beträgt $\approx 7.5\,\%$ und ergibt sich aus der Anzahl der durch NodeRAD als heterozygot gewerten Loci $het_{NodeRAD}$ bei tatsächlich homozygot simuliertem Locus hom_{ddRAGE} (siehe Formel (5-1)). Oder im Umkehrschluss erkennt NodeRAD 92.5 % der Loci korrekt homozygot.

Im Sinne eines prädiktiven Wertes nach Formel (5-2) für die Heterozygotie, beträgt die Wahrscheinlichkeit, dass bei einem durch NodeRAD als heterozygot bestimmten Locus $het_{NodeRAD}$ tatsächlich auch eine Heterozygotie het_{ddRAGE} vorliegt allerdings nur $\approx 33.\overline{3}\,\%$. Die Wahrscheinlichkeit, dass bei einem durch NodeRAD als homozygot ge-

werteten Locus auch ein homozygoter Locus in der Simulation vorliegt, beträgt 100 %, da keine falsch-homozygoten Loci-Zuordnungen auftraten.

$$P(het_{\,NodeRAD} \mid hom_{\,ddRAGE}) = \frac{10}{124 + 10} \approx 0.0746 \tag{5-1}$$

$$P(het_{\,ddRAGE} \mid het_{\,NodeRAD}) = \frac{5}{10 + 5} = 0.3\overline{3} \tag{5-2}$$

Zusammenfassend zeigt sich also eine gute Erkennungsrate der Loci, insbesondere werden heterozygote Loci korrekt erkannt. Durch NodeRAD werden jedoch mehr Loci heterozygot gewerteten Loci als tatsächlich vorhanden sind. Das tatsächlich homozygot vorliegenden Allel eines solchen Locus wird allerdings durch NodeRAD korrekt erfasst.

Kapitel 6

Ausblick

Das Tool NodeRAD [33] wurde als Prototyp für die Analyse von single-end RADSeq-Daten diploider Organismen implementiert. Das zugrundeliegenden Model erlaubt es, ohne Kenntnis des Genoms und nur mit Hilfe der Sequenzierfehlerrate und Heterozygotiewahrscheinlichkeit die wahrscheinlichsten Allele zu identifizieren, die den beobachteten Reads zugrunde liegen. Im Anschluss wird für diese Allele die jeweils wahrscheinlichste Locikomposition ermittelt. Hierbei zeigt sich an simulierten Daten eine gute Erkennungsrate der Loci (92 %), wobei homo- und heterozygote Varianten problemlos gefunden werden. Mit einer Rate von 7.5 % werden aber auch falsch-heterozygote Loci bestimmt. Dadurch sind von allen durch NodeRAD als heterozygot ausgewiesenen Loci nur $33.\overline{3}$ % auch tatsächlich heterozygot. Diese Statistiken sind zunächst nur eine grobe Abschätzung anhand einer kleineren simulierten Stichprobe. Für eine bessere Beurteilung sollte das Modell an größeren realen Datensätzen getestet werden. Es ist in Zukunft eine effizientere Programmierung des Tools in der Programmiersprache Rust geplant. Zudem kann hierbei die Bestimmung der Likelihood über den einzelnen Zusammenhangskomponenten parallelisiert werden, da diese Berechnungen unabhängig von einander erfolgen. Umfangreichere Tests auf realen Datensätzen sollten daher erst im Anschluss durch das in Rust implementieren Tool erfolgen.

Der dominierende Schritt hinsichtlich Laufzeit und Arbeitsspeicherbedarfs sind bei NodeRAD die Bestimmung der Allelkombinationen und die über ihre Allele-Fractions ausgeführten Berechnungen. Hier konnten durch optimale Wahl der Schwellenwerte für die Mindesthäufigkeiten der Allele unter- und oberhalb einer festgelegten Clustergröße deutliche Verbesserungen erreicht werden. Dieser Ansatz könnte weiter optimiert werden, so dass diese Werte beispielsweise direkt in Beziehung zur Clustergröße stehen. Denkbar wäre hier die Formulierung einer Funktionsgleichung, die in Abhängigkeit von der Clustergröße die maximale Anzahl der zu prüfenden Allele festlegt, wobei die Allele absteigend nach ihrer Häufigkeit als Kandidatenallele inkludiert werden. Eine andere Möglichkeit wäre, direkt

rechenintensive Cluster zu identifizieren und dort eine Optimierung bei der Wahl der Kandidatenallele durchzuführen. Solche rechenintensiven Cluster können zum einen sehr große Zusammenhangskomponenten sein, zum anderen aber auch eher kleine Cluster, die aber viele verschiedene Allele jeweils mit nur geringen Häufigkeiten aufweisen. Eine explizite Festlegung von Schwellenwerten nur für diese Cluster, ermöglicht eine genauere Analyse bei den übrigen, weniger problematischen Clustern. Dies könnte auch zu einem positiven Effekt hinsichtlich der Rate falsch-heterozygoter Loci führen.

Wie bereits in Kap. 1.3.1 beschrieben, ermöglicht die höhere Datendichte bei paired-end Reads eine größere Genauigkeit. Durch die damit verbundene Verringerung der Überlappungen der Reads könnte den Anteil falsch-heterozygoter Loci gesenkt werden. Allerdings führt die Verwendung von paired-end Reads dazu, dass heterozygote Varianten im Bereich der Restriktionsstelle nicht erkannt werden. Liegt bei heterozygoten Varianten die Mutation im Bereich der Restriktionsstelle, so kann das Restriktionsenzym dort nicht schneiden. In diesem Fall wäre es bei paired-end Sequenzierung nicht möglich, die Reads von beiden Seiten zu lesen. Dadurch würden die Reads des Allels mit der Variante verworfen werden und der Locus würde in der Analyse homozygot erscheinen. Bei single-end Reads würde das Allel mit der Variante von einer Seite gelesen werden. Da die Reads hier unabhängig von einander sind, würde nichts verworfen werden, so dass die Heterozygotie erkennbar bleibt. Aus diesem Grund wurde NodeRAD für single-end Reads konzipiert, so dass die Rate falsch-homozygoter Loci gering ist. In den simulierten Testdaten betrug diese Rate 0 %.

Wie bereits angesprochen, soll in diesem Kapitel auch auf die Analyse polyploider Spezies eingegangen werden. Der Prototyp NodeRAD wurde zunächst für diploide Spezies implementiert. Die Likelihoodberechnungen für die Loci-Zuordnungen kann dadurch unter Einbeziehung der Heterozygotiewahrscheinlichkeiten über ein pairHMM erfolgen. Bei polyploiden Organismen genügt der paarweise Vergleich nicht mehr. Vielmehr muss das Alignment über mehrere Kandidatenallele erfolgen, so dass hier für die Loci-Zuordnung und Bestimmung des Genotyps Algorithmen zur Erzeugung multipler Sequenzalignments Anwendung finden müssen [69, 70, 71]. Da diese im Allgemeinen eine wesentlich höhere Laufzeit aufweisen, ist die vorherige Eingrenzung der möglichen Loci-Kombinationen, wie dies bereits für diesen Prototyp implementiert wurde (vgl. Kap. 3.5.1 und Satz 1), von großem Vorteil.

Der Ansatz von NodeRAD ermöglicht es zudem im Rahmen von Diversitätsanalysen in einer zweiten Iteration durch Erweiterung des Modells auch einen interindividuellen Vergleich durchzuführen. Wie in Abbildung 3.9 bereits angedeutet, kann aus den Ergebnissen des Workflows, d.h. aus den VCF-Files der einzelnen Individuen, erneut ein Se-

quenzalignment durch Minimap2 gebildet und daraus ein Graph erzeugt werden. Aus den Zusammenhangskomponenten können dann die Kandidatenallele ermittelt und die jeweils wahrscheinlichsten Allele-Fractions über allen Individuen bestimmt werden. Diese dienen im Anschluss der Zuordnung zu einem oder mehreren gemeinsamen Loci. Je mehr Varianten die resultierenden Loci aufweisen, desto besser geht es der Population, aus der die getesteten Individuen stammen.

Anhang A

Anhang

A.1 Plots der BLAST-Analyse der Individuen B und C

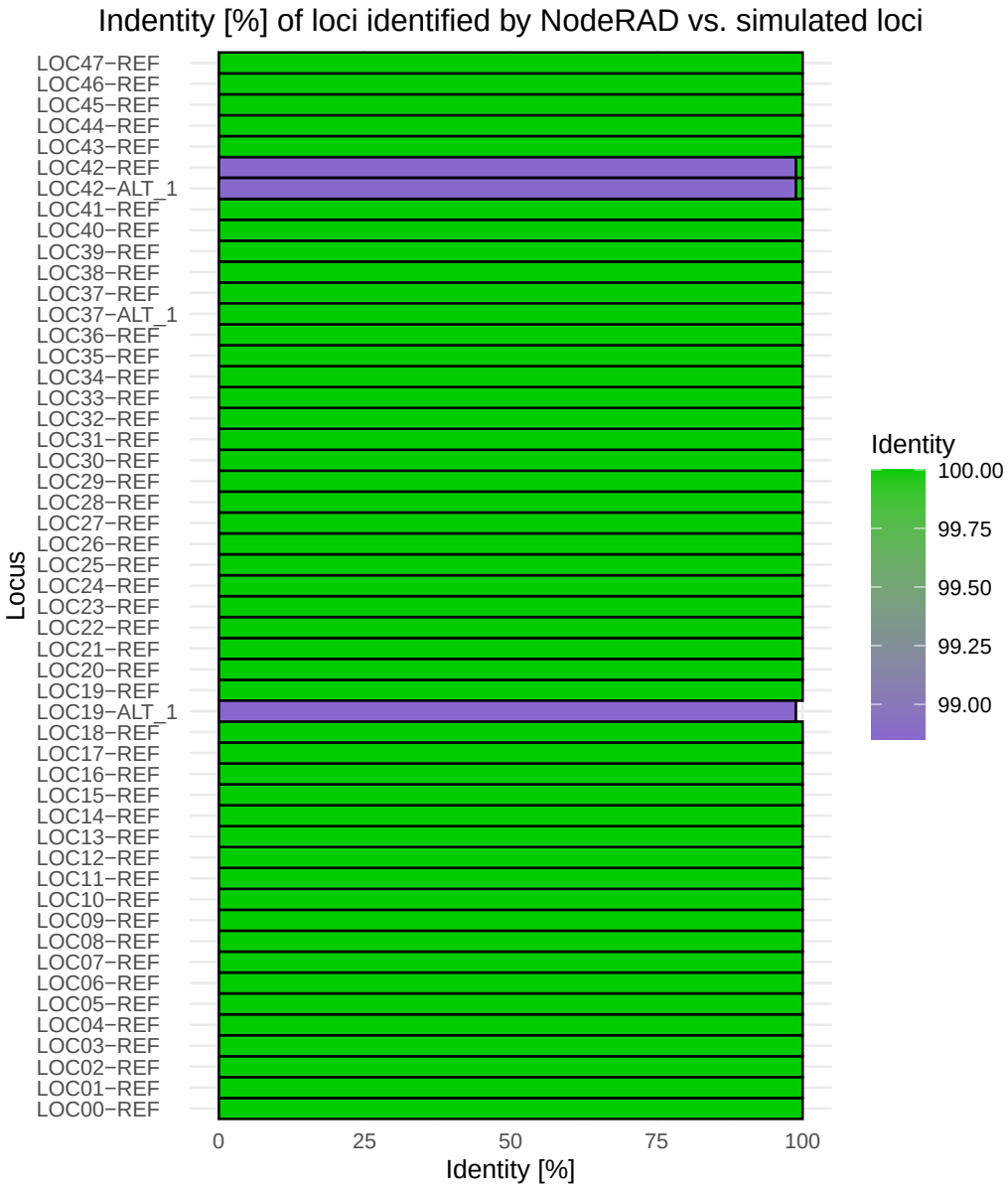

Abbildung A.1: Individuum B: Sequenzähnlichkeit (Identität in %) der durch NodeRAD bestimmten Allele gegenüber den simulierten Allelen.

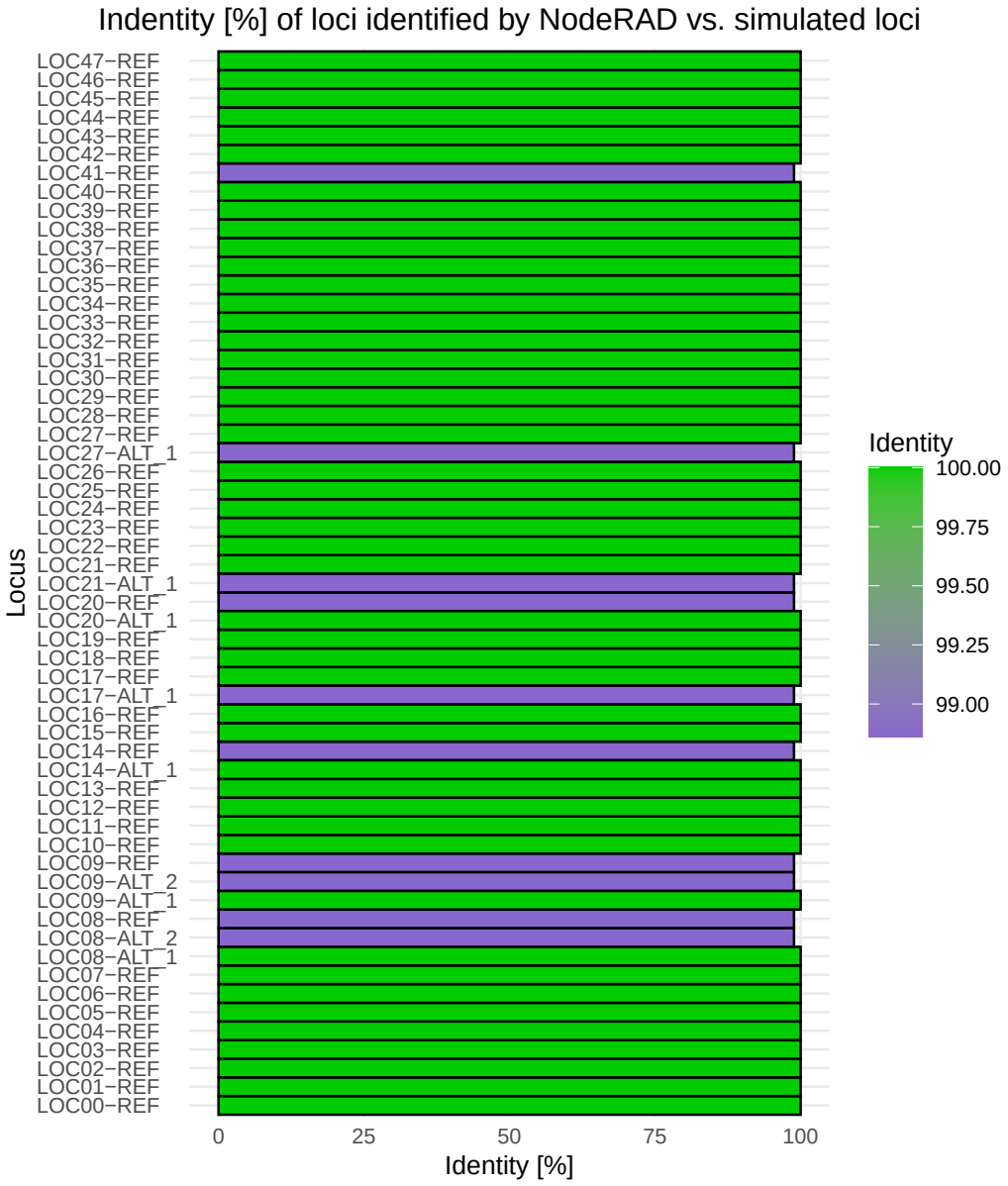

Abbildung A.2: Individuum C: Sequenzähnlichkeit (Identität in %) der durch NodeRAD bestimmten Allele gegenüber den simulierten Allelen.

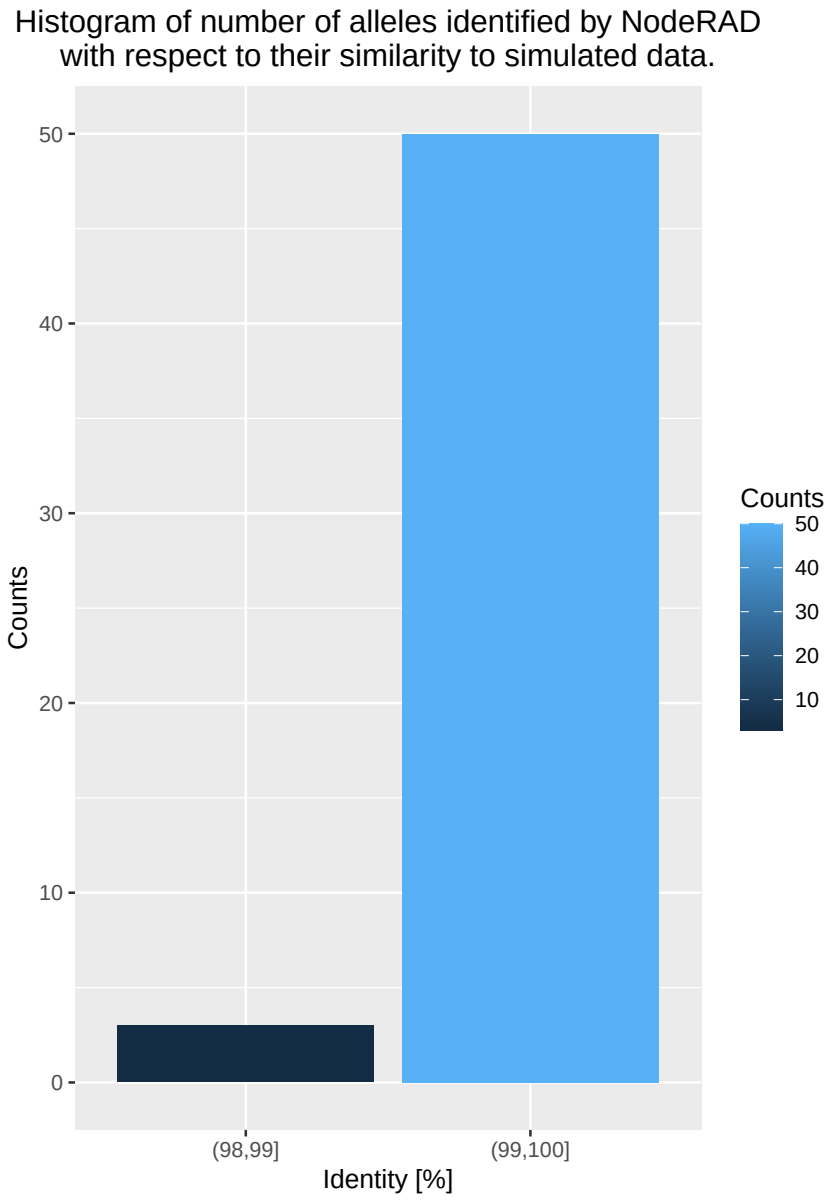

Abbildung A.3: Individuum B: Anzahl der durch NodeRAD bestimmten Referenz- und Alternativallele (Spalten REF und ALT in der VCF-Datei) hinsichtlich der Sequenzähnlichkeit (Identität in %) zu den simulierten Allelen

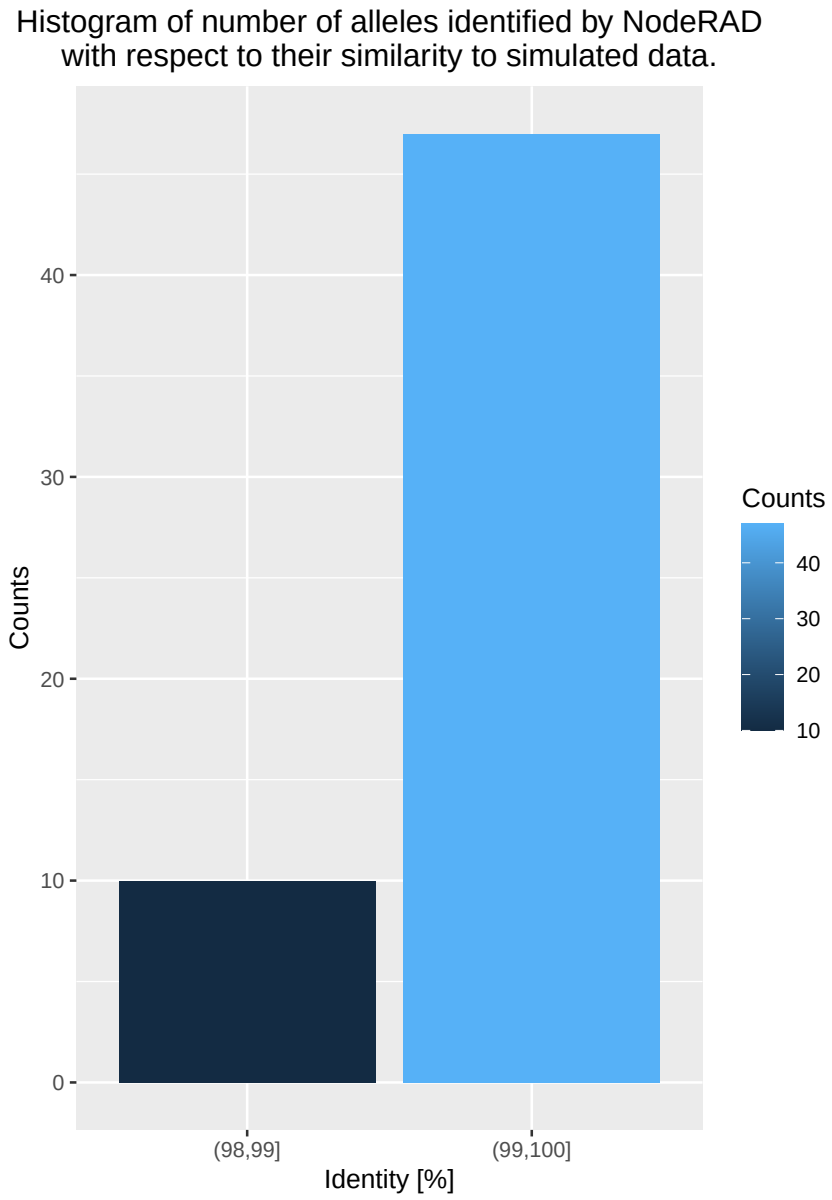

Abbildung A.4: Individuum C: Anzahl der durch NodeRAD bestimmten Referenz- und Alternativallele (Spalten REF und ALT in der VCF-Datei) hinsichtlich der Sequenzähnlichkeit (Identität in %) zu den simulierten Allelen

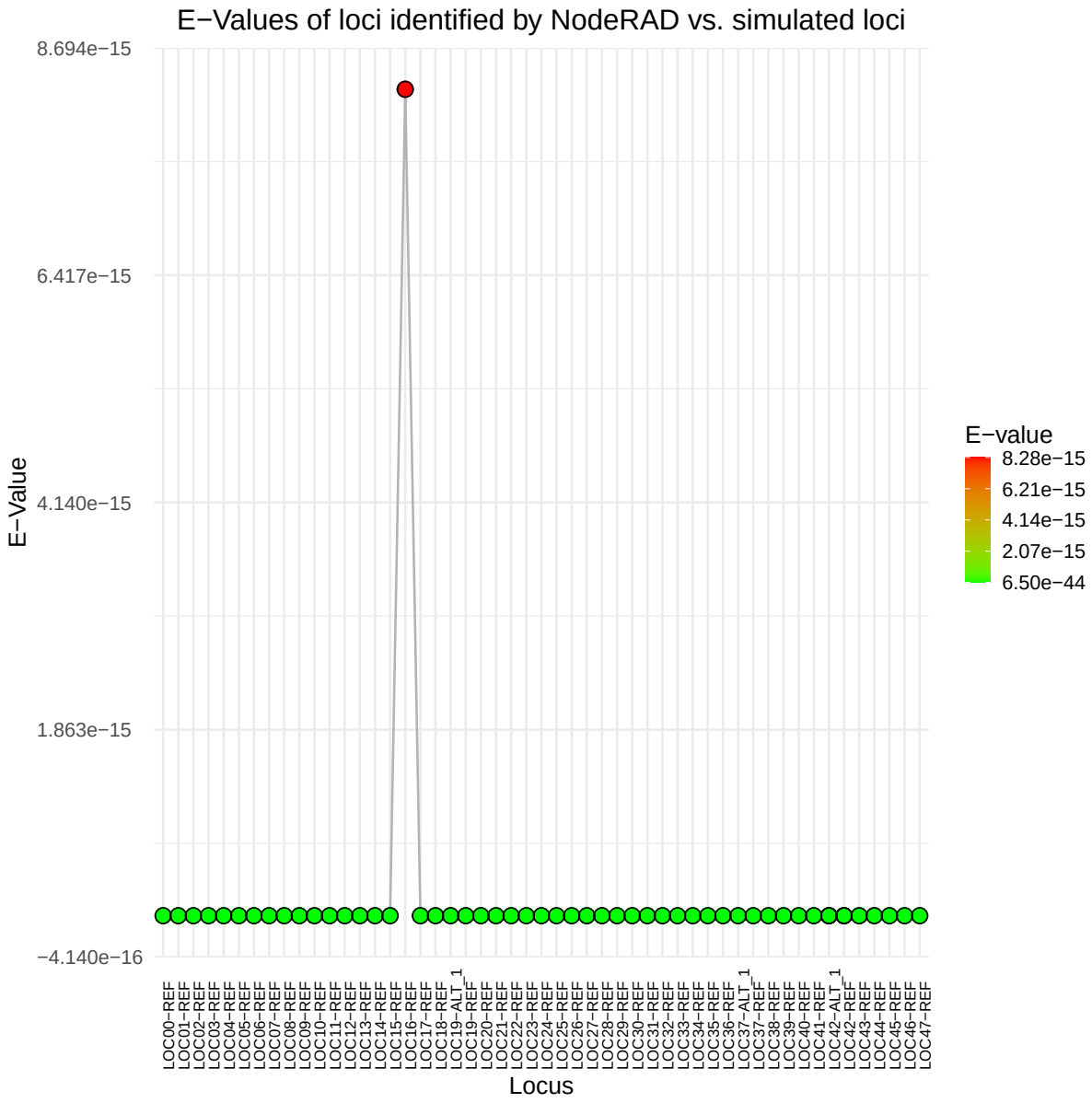

Abbildung A.5: Individuum B: E-Values aus dem Vergleich der durch NodeRAD bestimmten Loci mit den simulierten Allelen.

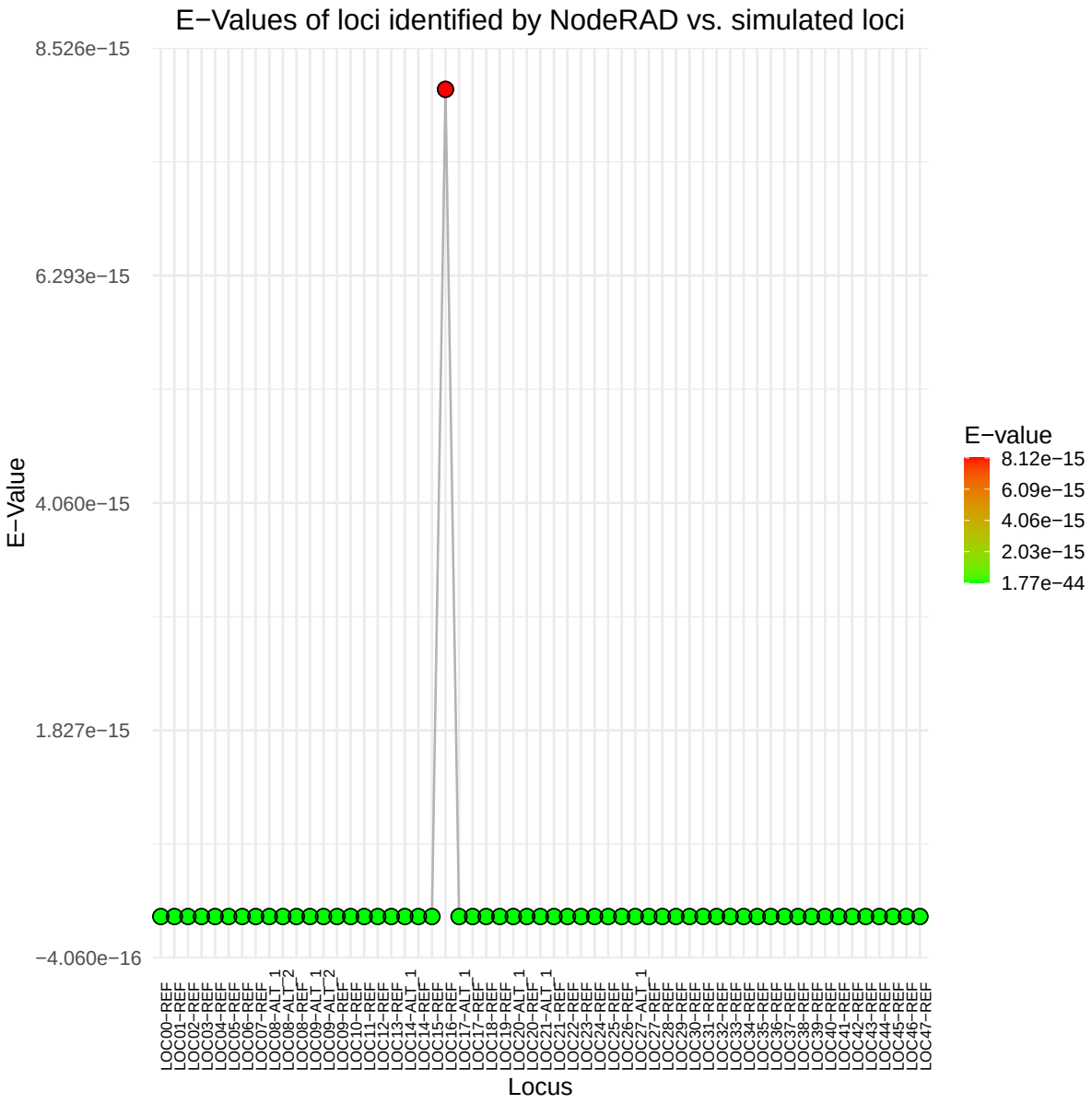

Abbildung A.6: Individuum C: E-Values aus dem Vergleich der durch NodeRAD bestimmten Allele mit den simulierten Allelen.

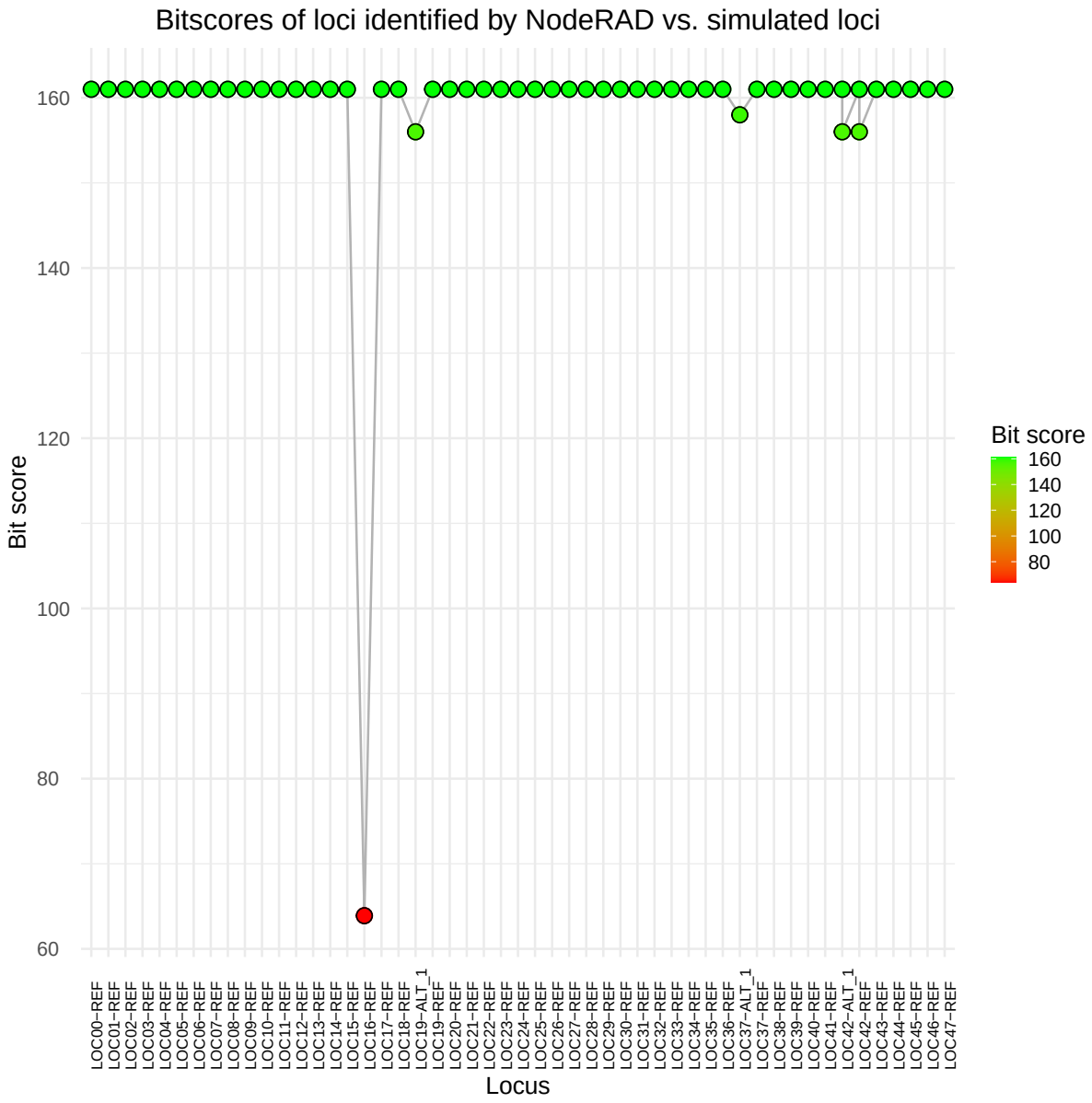

Abbildung A.7: Individuum B: Bitscores aus dem Vergleich der durch NodeRAD bestimmten Allele mit den simulierten Allelen.

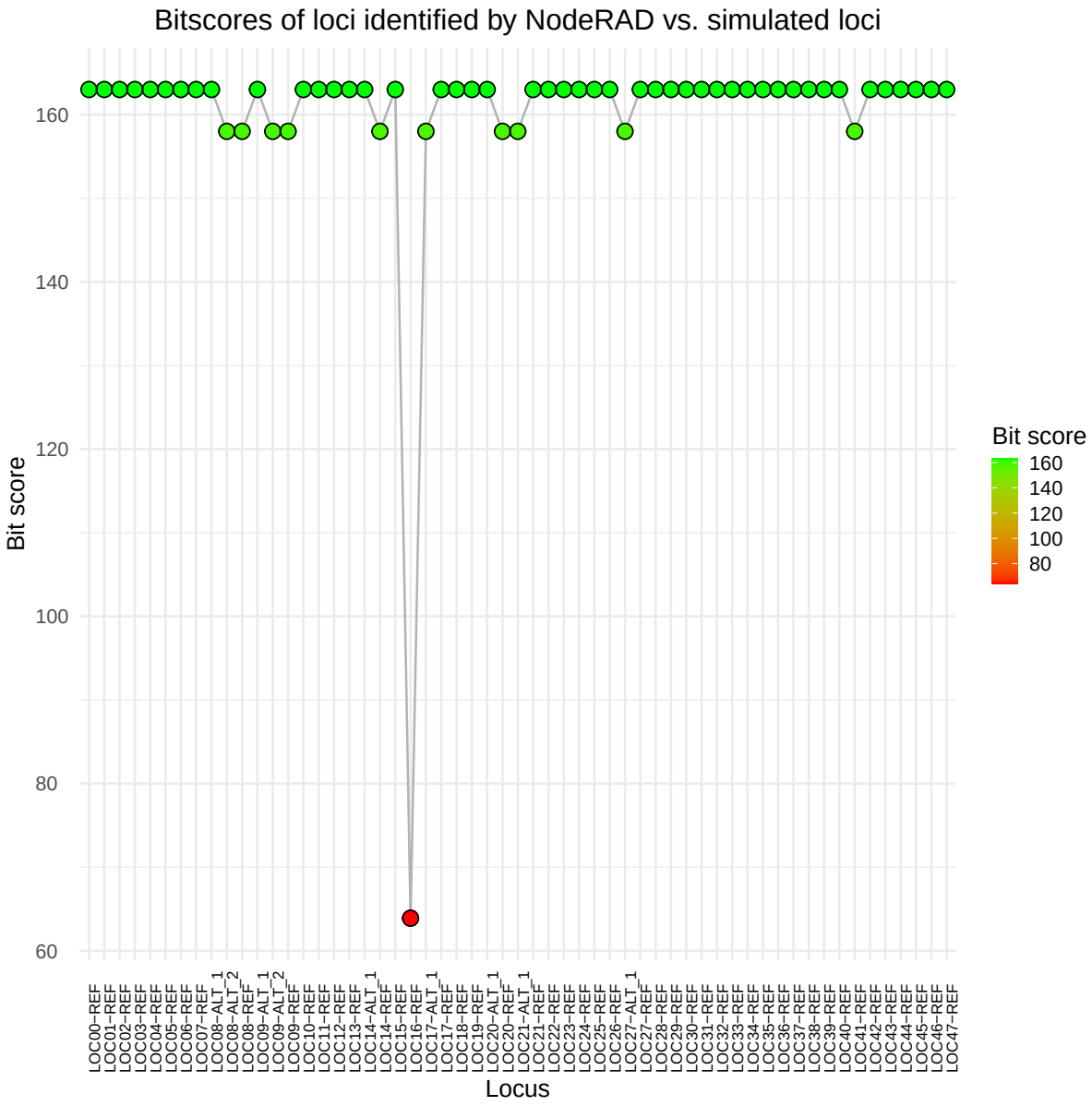

Abbildung A.8: Individuum C: Bitscores aus dem Vergleich der durch NodeRAD bestimmten Allele mit den simulierten Allelen.

Abbildungsverzeichnis

2.1 Übersicht der Prozesse des Workflows [39, 40, 41] 17

3.1 Prozesse des Workflows - Preprocessing [39, 40] 27
3.2 Prozesse des Workflows - Konstruktion des Graphen [39, 40] 28
3.3 Ausschnitt einiger Zusammenhangskomponenten des Gesamtgraphen 30
3.4 Beispiele von einzelnen Zusammenhangskomponenten unterschiedlicher Größe 31
3.5 Prozesse des Workflows - Kandidatenallele und Allele-Fractions [39] 32
3.6 Prozesse des Workflows - Likelihood der Allele-Fractions [39] 34
3.7 Prozesse des Workflows - Loci-Kombinationen [39, 41] 40
3.8 Prozesse des Workflows - Likelihood der Loci-Zuordnung [39, 41] 45
3.9 Prozesse des Workflows - Ausgabe als VCF-File [39, 41] 50
3.10 Beispiel einer durch NodeRAD erzeugten VCF-Datei (zur besseren Darstellbarkeit wurden verkürzte Sequenzen verwendet) 51

5.1 FastQ-Analyse: Basenqualität der Readsequenzen von Individuum A. 67
5.2 FastQ-Analyse: Adaptergehalt der Reads von Individuum A nach dem Trimming durch Cutadapt. 67
5.3 Individuum A: Sequenzähnlichkeit (Identität in %) der durch NodeRAD bestimmten Allele gegenüber den simulierten Allelen. 69
5.4 Individuum A: Anzahl der durch NodeRAD bestimmten Referenz- und Alternativallele (Spalten REF und ALT in der VCF-Datei) hinsichtlich der Sequenzähnlichkeit (Identität in %) zu den simulierten Allelen 70
5.5 Individuum A: E-Values aus dem Vergleich der durch NodeRAD bestimmten Allele mit den simulierten Allelen 71
5.6 Individuum A: Bitscores aus dem Vergleich der durch NodeRAD bestimmten Allele mit den simulierten Allelen 72

A.1 Individuum B: Sequenzähnlichkeit (Identität in %) der durch NodeRAD bestimmten Allele gegenüber den simulierten Allelen. 79
A.2 Individuum C: Sequenzähnlichkeit (Identität in %) der durch NodeRAD bestimmten Allele gegenüber den simulierten Allelen. 80

A.3 Individuum B: Anzahl der durch NodeRAD bestimmten Referenz- und Alternativallele (Spalten REF und ALT in der VCF-Datei) hinsichtlich der Sequenzähnlichkeit (Identität in %) zu den simulierten Allelen 81
A.4 Individuum C: Anzahl der durch NodeRAD bestimmten Referenz- und Alternativallele (Spalten REF und ALT in der VCF-Datei) hinsichtlich der Sequenzähnlichkeit (Identität in %) zu den simulierten Allelen 82
A.5 Individuum B: E-Values aus dem Vergleich der durch NodeRAD bestimmten Loci mit den simulierten Allelen. 83
A.6 Individuum C: E-Values aus dem Vergleich der durch NodeRAD bestimmten Allele mit den simulierten Allelen. 84
A.7 Individuum B: Bitscores aus dem Vergleich der durch NodeRAD bestimmten Allele mit den simulierten Allelen. 85
A.8 Individuum C: Bitscores aus dem Vergleich der durch NodeRAD bestimmten Allele mit den simulierten Allelen. 86

Algorithmenverzeichnis

3.1 Suche eines Alignments zwischen Read und Kandidatenallel 36
3.2 Berechnung der Likelihood zwischen Read und Kandidatenallel 38
3.3 Bestimmung der Likelihood der Allele innerhalb einer Loci-Kombination . . 47
3.4 Bestimmung der Likelihood zwischen zwei Kandidatenallelen hinsichtlich der Heterozygotiewahrscheinlichkeiten 48
3.5 CIGAR-Tupel bestimmen 49

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] Green, Michael R. and Joseph Sambrook: *Polymerase chain reaction.* Cold Spring Harbor protocols, 2019, Jun 2019.

[21] McPherson, J. D., M. Marra, L. Hillier, R. H. Waterston, A. Chinwalla, J. Wallis, M. Sekhon, K. Wylie, E. R. Mardis, R. K. Wilson, R. Fulton, T. A. Kucaba, C. Wagner-McPherson, W. B. Barbazuk, S. G. Gregory, S. J. Humphray, L. French, R. S. Evans, G. Bethel, A. Whittaker, J. L. Holden, O. T. McCann, A. Dunham, C. Soderlund, C. E. Scott, D. R. Bentley, G. Schuler, H. C. Chen, W. Jang, E. D. Green, J. R. Idol, V. V. Maduro, K. T. Montgomery, E. Lee, A. Miller, S. Emerling, Kucherlapati, R. Gibbs, S. Scherer, J. H. Gorrell, E. Sodergren, K. Clerc-Blankenburg, P. Tabor, S. Naylor, D. Garcia, P. J. de Jong, J. J. Catanese, N. Nowak, K. Osoegawa, S. Qin, L. Rowen, A. Madan, M. Dors, L. Hood, B. Trask, C. Friedman, H. Massa, V. G. Cheung, I. R. Kirsch, T. Reid, R. Yonescu, J. Weissenbach, T. Bruls, R. Heilig, E. Branscomb, A. Olsen, N. Doggett, J. F. Cheng, T. Hawkins, R. M. Myers, J. Shang, L. Ramirez, J. Schmutz, O. Velasquez, K. Dixon, N. E. Stone, D. R. Cox, D. Haussler, W. J. Kent, T. Furey, S. Rogic, S. Kennedy, S. Jones, A. Rosenthal, G. Wen, M. Schilhabel, G. Gloeckner, G. Nyakatura, R. Siebert, B. Schlegelberger, J. Korenberg, X. N. Chen, A. Fujiyama, M. Hattori, A. Toyoda, T. Yada, H. S. Park, Y. Sakaki, N. Shimizu, S. Asakawa, K. Kawasaki, T. Sasaki, A. Shintani, A. Shimizu, K. Shibuya, J. Kudoh, S. Minoshima, J. Ramser, P. Seranski, C. Hoff, A. Poustka, R. Reinhardt, and H. Lehrach: *A physical map of the human genome.* Nature, 409:934–41, Feb 2001.

[22] Flicek, P., B. L. Aken, K. Beal, B. Ballester, M. Caccamo, Y. Chen, L. Clarke, G. Coates, F. Cunningham, T. Cutts, T. Down, S. C. Dyer, T. Eyre, S. Fitzgerald, J. Fernandez-Banet, S. Gräf, S. Haider, M. Hammond, R. Holland, K. L. Howe, K. Howe, N. Johnson, A. Jenkinson, A. Kähäri, D. Keefe, F. Kokocinski, E. Kulesha, D. Lawson, I. Longden, K. Megy, P. Meidl, B. Overduin, A. Parker, B. Pritchard, A. Prlic, S. Rice, D. Rios, M. Schuster, I. Sealy, G. Slater, D. Smedley, G. Spu-

dich, S. Trevanion, A. J. Vilella, J. Vogel, S. White, M. Wood, E. Birney, T. Cox, V. Curwen, R. Durbin, X. M. Fernandez-Suarez, J. Herrero, T. J. P. Hubbard, A. Kasprzyk, G. Proctor, J. Smith, A. Ureta-Vidal, and S. Searle: *Ensembl 2008.* Nucleic acids research, 36:D707–14, Jan 2008.

[23] Shendure, Jay and Hanlee Ji: *Next-generation dna sequencing.* Nature biotechnology, 26:1135–45, Oct 2008.

[24] Rizzo, Jason M. and Michael J. Buck: *Key principles and clinical applications of "next-generation" dna sequencing.* Cancer prevention research (Philadelphia, Pa.), 5:887–900, Jul 2012.

[25] Ambardar, Sheetal, Rikita Gupta, Deepika Trakroo, Rup Lal, and Jyoti Vakhlu: *High throughput sequencing: An overview of sequencing chemistry.* Indian journal of microbiology, 56:394–404, Dec 2016.

[26] Dijk, Erwin L. van, Hélène Auger, Yan Jaszczyszyn, and Claude Thermes: *Ten years of next-generation sequencing technology.* Trends in genetics : TIG, 30:418–26, Sep 2014.

[27] Miller, Michael R., Joseph P. Dunham, Angel Amores, William A. Cresko, and Eric A. Johnson: *Rapid and cost-effective polymorphism identification and genotyping using restriction site associated dna (rad) markers.* Genome research, 17:240–8, Feb 2007.

[28] Roberts, Richard J., Marlene Belfort, Timothy Bestor, Ashok S. Bhagwat, Thomas A. Bickle, Jurate Bitinaite, Robert M. Blumenthal, Sergey Kh Degtyarev, David T. F. Dryden, Kevin Dybvig, Keith Firman, Elizaveta S. Gromova, Richard I. Gumport, Stephen E. Halford, Stanley Hattman, Joseph Heitman, David P. Hornby, Arvydas Janulaitis, Albert Jeltsch, Jytte Josephsen, Antal Kiss, Todd R. Klaenhammer, Ichizo Kobayashi, Huimin Kong, Detlev H. Krüger, Sanford Lacks, Martin G. Marinus, Michiko Miyahara, Richard D. Morgan, Noreen E. Murray, Valakunja Nagaraja, Andrzej Piekarowicz, Alfred Pingoud, Elisabeth Raleigh, Desirazu N. Rao, Norbert Reich, Vladimir E. Repin, Eric U. Selker, Pang-Chui Shaw, Daniel C. Stein, Barry L. Stoddard, Waclaw Szybalski, Thomas A. Trautner, James L. Van Etten, Jorge M. B. Vitor, Geoffrey G. Wilson, and Shuang-yong Xu: *A nomenclature for restriction enzymes, dna methyltransferases, homing endonucleases and their genes.* Nucleic acids research, 31:1805–12, Apr 2003.

[29] Mardis, Elaine R.: *Dna sequencing technologies: 2006-2016.* Nature protocols, 12:213–218, Feb 2017.

[30] Baird, Nathan A., Paul D. Etter, Tressa S. Atwood, Mark C. Currey, Anthony L. Shiver, Zachary A. Lewis, Eric U. Selker, William A. Cresko, and Eric A. Johnson: *Rapid snp discovery and genetic mapping using sequenced rad markers.* PloS one, 3:e3376, 2008.

[31] Davey, John W. and Mark L. Blaxter: *Radseq: next-generation population genetics.* Briefings in functional genomics, 9:416–23, Dec 2010.

[32] Leggett, Richard M. and Dan MacLean: *Reference-free snp detection: dealing with the data deluge.* BMC genomics, 15 Suppl 4:S10, 2014.

[33] Vietor, Antonie: *NodeRAD - RADSeq tool with Snakemake workflow integration for analysis of RAD sequencing data.* source: `https://github.com/AntonieV/NodeRAD`.

[34] Peterson, Brant K., Jesse N. Weber, Emily H. Kay, Heidi S. Fisher, and Hopi E. Hoekstra: *Double digest radseq: an inexpensive method for de novo snp discovery and genotyping in model and non-model species.* PloS one, 7:e37135, 2012.

[35] Group, The SAM/BAM Format Specification Working: *Sequence Alignment/Map Format Specfication.* source: `https://samtools.github.io/hts-specs/SAMv1.pdf`.

[36] 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.

[37] 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.

[38] 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.

[39] Alanis, Cristo J.: *Schema of Labs on a class.* source: `https://texample.net/tikz/examples/labs-schema/`.

[40] Veličković, Petar: *Deoxyribonucleic acid (DNA).* source: `https://github.com/PetarV-/TikZ/tree/master/DNA`.

[41] Twitter, Inc., Mark Otto, and the Bootstrap authors: *Bootstrap icons - receipt.* source: `https://github.com/twbs/icons/blob/main/icons/receipt.svg`.

[42] Li, Heng: *Minimap2: pairwise alignment for nucleotide sequences.* Bioinformatics, 34(18):3094–3100, 05 2018.

[43] Durbin, Richard, Sean R. Eddy, Anders Krogh, and Graeme Mitchison: *Markov chains and hidden markov models.* In *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, pages 47–80. Cambridge University Press, 1998.

[44] Yoon, Byung-Jun: *Hidden markov models and their applications in biological sequence analysis.* Current genomics, 10:402–15, Sep 2009.

[45] Schirmer, Melanie, Rosalinda D'Amore, Umer Z. Ijaz, Neil Hall, and Christopher Quince: *Illumina error profiles: resolving fine-scale variation in metagenomic sequencing data.* BMC Bioinformatics, 17(1):125, March 2016.

[46] Kuhner, Mary K. and James McGill: *Correcting for sequencing error in maximum likelihood phylogeny inference.* G3 (Bethesda, Md.), 4:2545–52, Nov 2014.

[47] Köster, Johannes and Sven Rahmann: *Snakemake—a scalable bioinformatics workflow engine.* Bioinformatics, 28(19):2520–2522, 08 2012.

[48] 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.

[49] Martin, Marcel: *Cutadapt removes adapter sequences from high-throughput sequencing reads.* EMBnet.journal, 17(1):10–12, 2011.

[50] Andrews, Simon, Felix Krueger, Anne Segonds-Pichon, Laura Biggins, Christel Krueger, and Steven Wingett: *FastQC.* Babraham Institute, January 2012. source: `http://www.bioinformatics.babraham.ac.uk/projects/fastqc/`.

[51] 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.

[52] Peixoto, Tiago P.: *The graph-tool python library.* figshare, 2014.

[53] 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.

[54] 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.

[55] Heger, Andreas and Kevin Jacobs: *FastQC.* source: `https://github.com/pysam-developers/pysam,documentation:https://pysam.readthedocs.io/en/latest/index.html`.

[56] Peixoto, Tiago de Paula: *graph-tool 2.35 documentation: graph_tool - efficient graph analysis and manipulation.* source: `https://graph-tool.skewed.de/static/doc/graph_tool.html`.

[57] Voss, W, Sauerbier T: *Taschenbuch der Statistik*, chapter 8. Kombinatorik, pages 281–281. Fachbuchverlag Leipzig, 2004.

[58] 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.

[59] Python Software Foundation, Van Rossum, Guido: *itertools — functions creating iterators for efficient looping.* source: `https://docs.python.org/3/library/itertools.html`.

[60] Ewing, B. and P. Green: *Base-calling of automated sequencer traces using phred. ii. error probabilities.* Genome research, 8:186–94, Mar 1998.

[61] 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`.

[62] Wittenburg, Kim: *Algorithmen und datenstrukturen*, 2017. source: `https://www2.informatik.uni-hamburg.de/fachschaft/wiki/images/3/3e/AD-Skript_(WS16-17).pdf`, pages 156-157.

[63] Foundation, Python Software and Daniel Stutzbach: *Python — time-complexity of various operations.* source: `https://wiki.python.org/moin/TimeComplexity`.

[64] Timm, Henning, Hannah Weigand, Martina Weiss, Florian Leese, and Sven Rahmann: *ddrage: A data set generator to evaluate ddradseq analysis software.* Molecular ecology resources, 18:681–690, May 2018.

[65] Timm, Henning: *ragge 1.7.1 documentation.* source: `https://ddrage.readthedocs.io/en/latest/`.

[66] *Test dataset simulated by ddRAGE.* source: `https://github.com/AntonieV/NodeRAD/tree/master/.test/data/small_dataset`.

[67] Altschul, S. F., W. Gish, W. Miller, E. W. Myers, and D. J. Lipman: *Basic local alignment search tool.* Journal of molecular biology, 215:403–10, Oct 1990.

[68] Gaedeke, Nicola: *Sequenzähnlichkeitssuche mit Hilfe des „Basic Local Alignment Search Tool“ (BLAST) in den Sequenzdatenbanken des NCBI*, pages 79–112. Birkhäuser Basel, Basel, 2007.

[69] Chowdhury, Biswanath and Gautam Garai: *A review on multiple sequence alignment from the perspective of genetic algorithm.* Genomics, 109:419–431, Oct 2017.

[70] Bawono, Punto, Maurits Dijkstra, Walter Pirovano, Anton Feenstra, Sanne Abeln, and Jaap Heringa: *Multiple sequence alignment.* Methods in molecular biology (Clifton, N.J.), 1525:167–189, 2017.

[71] Chatzou, Maria, Cedrik Magis, Jia-Ming Chang, Carsten Kemena, Giovanni Bussotti, Ionas Erb, and Cedric Notredame: *Multiple sequence alignment modeling: methods and applications.* Briefings in Bioinformatics, 17(6):1009–1023, 11 2015.

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 11. Februar 2021

Muster Mustermann