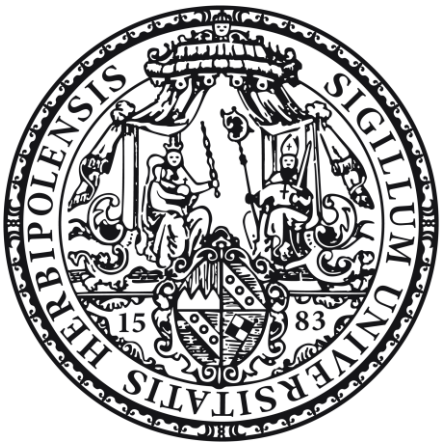
Bioinformatische Methoden zur Identifizierung und Klassifizierung somatischer Mutationen in hämatologischen Erkrankungen



Bioinformatics approaches for the detection and classification of somatic mutations in hematological malignancies

Dissertation zur Erlangung des

naturwissenschaftlichen Doktorgrades

der Julius-Maximilians-Universität Würzburg

Vorgelegt von

Jordan Ivanov Pischimarov

aus

Sofia (Bulgarien)

Würzburg 2016

Eingereicht am: _____

<u>Mitglieder der Promotionskommission</u>:

Vorsitzender:

Gutachter: Prof. Dr. Thomas Dandekar

Gutachter: Prof. Dr. Andreas Rosenwald

Tag des Promotionskolloquiums: _____

Doktorurkunde ausgehändigt am: _____

Erklärungen nach §4 Abs. 3 Satz 3, 5, 8 der Promotionsordnung der Fakultät für Biologie

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, die Dissertation: „**Bioinformatische Methoden zur Identifizierung und Klassifizierung somatischer Mutationen in hämatologischen Erkrankungen**“, eigenständig, d. h. insbesondere selbständig und ohne Hilfe eines kommerziellen Promotionsberaters, angefertigt und keine anderen, als die von mir angegebenen Quellen und Hilfsmittel verwendet zu haben.

Ich erkläre außerdem, dass die Dissertation weder in gleicher noch in ähnlicher Form bereits in einem anderen Prüfungsverfahren vorgelegen hat.

Würzburg, den_____ _____

Jordan Pischimarov

Affidavit

I hereby declare that my thesis entitled: „**Bioinformatics approaches for the detection and classification of somatic mutations in hematological malignancies**” is the result of my own work.

I did not receive any help or support from commercial consultants. All sources and / or materials applied are listed and specified in the thesis.

Furthermore I verify that the thesis has not been submitted as part of another examination process neither in identical nor in similar form.

Würzburg, den_____ _____

Jordan Pischimarov

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Arbeit begleitet, unterstützt und motiviert haben.

Zuerst gebührt mein Dank Prof. Dr. Andreas Rosenwald, der mir die Möglichkeit zur Erstellung meiner Doktorarbeit gab und mit hilfreichen Anregungen und konstruktiver Kritik sowie der Begutachtung die Anfertigung der Arbeit unterstützte.

Ebenfalls möchte ich mich bei Prof. Dr. Thomas Dandekar, der immer ein offenes Ohr für mich hatte, für die Anregungen, konstruktive Kritik und die hilfreiche Unterstützung sowie der Begutachtung der Arbeit bedanken.

Frau Dr. Ellen Leich gebührt besonderer Dank für die Projekt nahe Betreuung sowie Geduld, Interesse, Hilfsbereitschaft und zahlreiche interessante Debatten und Ideen, die maßgeblich dazu beigetragen haben, dass diese Doktorarbeit in dieser Form vorliegt.

Prof. Dr. Franz Cemič und Prof. Dr. Jürgen Hemberger möchte ich herzlich für die Unterstützung und den regen Austausch zu statistischen und bioinformatischen Fragestellungen danken.

Bedanken möchte ich mich auch bei meinen stetigen, ehemaligen und momentanen Arbeitskollegen Dr. Susann Weißbach, Dr. Hilka Rauert-Wunderlich, Dr. Shuji Momose, Sarah Keppler, Tina Grieb, Theodora Nedeva für die tolle Zusammenarbeit und ein unvegessliches Arbeitsklima.

Meinen Freunden und Fußball-Kameraden die es stets verstanden haben mich und zu motivieren möchte ich ebenfalls danken.

Ebenso möchte ich mich bei meiner Familie bedanken, die immer in allen Lebenssituationen für mich da ist und auch mir einen großen Rückhalt gibt.

Abschließend möchte ich mich bei einer ganz besonderen Person bedanken bei meiner Freundin die sowohl meine Ehefrau als auch meine Seelenverwandte ist, bedanken.

Inhaltsverzeichnis

1 Zusammenfassung

Die Sequenzierungstechnologien entwickeln sich stetig weiter, dies ermöglicht eine zuvor nicht erreichte Ausbeute an experimentellen Daten und auch an Neuentwicklungen von zuvor nicht realisierbaren Experimenten. Zugleich werden spezifische Datenbanken, Algorithmen und Softwareprogramme entwickelt, um die neu entstandenen Daten zu analysieren. Während der Untersuchung bioinformatischer Methoden für die Identifizierung und Klassifizierung somatischer Mutationen in hämatologischen Erkrankungen, zeigte sich eine hohe Vielfalt an alternativen Softwaretools die für die jeweiligen Analyseschritte genutzt werden können. Derzeit existiert noch kein Standard zur effizienten Analyse von Mutationen aus *Next-Generation-Sequencing* (NGS)-Daten. Die unterschiedlichen Methoden und Pipelines generieren Kandidaten, die zum größten Anteil in allen Ansätzen identifiziert werden können, jedoch werden Software spezifische Kandidaten nicht einheitlich detektiert.

Um eine einheitliche und effiziente Analyse von NGS-Daten durchzuführen war im Rahmen dieser Arbeit die Entwicklung einer benutzerfreundlichen und einheitlichen Pipeline vorgesehen. Hierfür wurden zunächst die essentiellen Analysen wie die Identifizierung der Basen, die Alignierung und die Identifizierung der Mutationen untersucht. Des Weiteren wurden unter Berücksichtigung von Effizienz und Performance diverse verfügbare Softwaretools getestet, ausgewertet und sowohl mögliche Verbesserungen als auch Erleichterungen der bisherigen Analysen vorgestellt und diskutiert. Durch Mitwirken in Konsortien wie der klinischen Forschergruppe 216 (KFO 216) und *International Cancer Genome Consortium* (ICGC) oder auch bei Haus-internen Projekten wurden Datensätze zu den Entitäten Multiples Myelom (MM), Burkitt Lymphom (BL) und Follikuläres Lymphom (FL) erstellt und analysiert. Die Selektion geeigneter Softwaretools und die Generierung der Pipeline basieren auf komparativen Analysen dieser Daten, sowie auf geteilte Ergebnisse und Erfahrungen in der Literatur und auch in Foren. Durch die gezielte Entwicklung von Skripten konnten biologische und klinische Fragestellungen bearbeitet werden. Hierzu zählten eine einheitliche Annotation der Gennamen, sowie die Erstellung von Genmutations-*Heatmaps* mit nicht *Variant-Calling-File* (VCF)-Syntax konformen Dateien. Des Weiteren konnten nicht abgedeckte Regionen des Genoms in den NGS-Daten identifiziert und analysiert werden. Neue Projekte zur detaillierten Untersuchung der Verteilung von wiederkehrender Mutationen und Funktionsassays zu einzelnen Mutationskandidaten konnten basierend auf den Ergebnissen initiiert werden.

Durch eigens erstellte Python-Skripte konnte somit die Funktionalität der Pipeline erweitert

werden und zu wichtigen Erkenntnissen bei der biologischen Interpretation der Sequenzierungsdaten führen, wie beispielsweise zu der Detektion von drei neuen molekularen Subgruppen im MM. Die Erweiterungen, der in dieser Arbeit entwickelten Pipeline verbesserte somit die Effizienz der Analyse und die Vergleichbarkeit unserer Daten. Des Weiteren konnte durch die Erstellung eines eigenen Skripts die Analyse von unbeachteten Regionen in den NGS-Daten erfolgen.

2 Summary

The sequencing technologies, while still being under further development, render it possible to develop novel experiments and allow the generation of larger amounts of utilizable data. At the same time novel software tools, databases and algorithms are developed to analyze these larger amounts of data. The analysis of somatic mutations in hematological malignancies showed that a high variety of alternative software tools can be used for different analysis steps. Furthermore there is currently no standardized procedure for the efficient identification and analysis of mutations in NGS data. The different pipeline and methods are, for the most part, able to identify the same mutation candidates, however there are software specific candidates which are not called by all pipelines.

The scope of this dissertation was therefore to develop a user-friendly pipeline which is able to call candidate mutations uniformly and efficiently. For this purpose necessary analysis steps including base calling, alignment generation and variant calling were investigated. Furthermore available software tools were tested and evaluated regarding their efficiency and performance. Possible improvements of these software tools and previously performed analysis are explained and discussed in this work. NGS data sets of the different cancer entities multiple myeloma (MM), Burkitt lymphoma (BL) and follicular lymphoma (FL) were generated and analyzed within the framework of cooperate projects like the International Cancer Genome Consortium (ICGC) and the Clinical Research Group 216 (KFO) as well as for internal projects. The development of the pipeline and selection of suitable software tools is based on the comparative analysis of the generated data sets, as well as previously described results and experiences in literature and forums. The selective development of certain python scripts enabled the evaluation of novel biological and clinical questions by standardizing gene names in the annotation step, generating heat- maps of non-standardized VCF-files as well as the identification and analysis of uncovered regions in NGS data sets. This work and the obtained results thereby provide the groundwork for further projects e.g. the analysis of the distribution of recurrent mutations or the functional analysis of specific mutation candidates. This extensions of the developed pipeline with python scripts helped to improve the efficiency and comparability of the NGS data. The interpretation of the NGS data with the extended script for example led to the discovery of three distinct molecular subgroups in MM. Furthermore the generation of the novel python scripts helped to analyze uncovered regions in the NGS data sets.

3 Einleitung

3.1 Mutationen

Zur genauen Definition von Mutationen, Variationen, oder auch als Alterationen bezeichnete Veränderungen im Genom, müssen Faktoren wie Ursprung, Größe, Art, Frequenz und Lokalisation berücksichtigt werden. Mutationen, welche von den Vorfahren vererbt und ab dem Beginn des Lebenszyklus vorhanden sind, werden im Allgemeinem als harmlos bewertet und als Keimbahnmutationen bzw. als *germline* Mutationen bezeichnet. Jedoch darf die Signifikanz der Prädisposition dieser Mutationen bei der Kanzerogenese nicht außer Acht gelassen werden. Im Gegensatz zu den Keimbahnmutationen entstehen somatische Mutationen während der Entwicklung des Individuums und können durch äußere Einflüsse wie beispielsweise durch Umweltfaktoren, Strahlung, Infektionen oder Lebensgewohnheiten begünstigt oder sogar initiiert werden [1]. Hierbei sind nicht alle somatischen Mutationen gleichbedeutend mit krankheitsassoziierten Veränderungen des Genoms, da die natürliche Weiterentwicklung, und somit die Veränderung des Genoms, Teil der Evolution ist. Die beobachtete Mutationsrate liegt bei den Keimbahn-Punktmutationen zwischen 50 – 216 Substitutionen pro Generation eines diploiden menschlichen Genoms [2-5]. Bei den somatischen Punktmutationen ist zur Bestimmung der Mutationsrate das Alter, die Replikationszeit der Zellen und auch das spezifische Gewebe zu beachten. So ist beispielsweise bei einem 60-jährigen Individuum eine somatische Mutationsrate von 4000 – 40000 Punktmutationen in den kodierenden Regionen von Darmepithel- oder Epidermis-Zellen zu erwarten. Die Mutationsrate von B- oder T-Lymphozyten liegt in vitro bei $1{,}47 \times 10^{-9}$ Mutationen pro Nukleotid und pro Zellteilung [3].
Den ersten Katalog in dem der Zusammenhang von Genen des humanen Genoms mit genetischen Alterationen in Verbindung gebracht wird, veröffentlichte 1966 McKusick in „*Mendelian Inheritance in Man*“ (MIM) [6]. Die online Version von MIM (OMIM) wurde 1987 zur Verfügung gestellt und die Administration der Daten im Jahre 2000 vom *National Center of Biotechnologie Information* (NCBI) übernommen [7].
Ein weiteres entscheidendes Merkmal bei den Mutationen ist die Größe, beziehungsweise die Anzahl an betroffenen Basen. Einzelne im Genom ausgetauschte, eingefügte oder entfernte Basen werden als Punktmutationen klassifiziert und dementsprechend in Substitutionen, Insertionen oder Deletionen unterteilt (Abbildung 1). Wobei die Insertionen und Deletionen (Indel) als Indels zusammengefasst werden. Der Austausch einer Base in der genkodierenden Desoxyribonukleinsäure (DNA)-Region impliziert eine Basenänderung im Codon. Führt die Mutation nicht zur Änderung der translatierten Proteinsequenz, wird diese als *synonymous*

bezeichnet. Ist jedoch ein Aminosäureaustausch die Folge, wird die Mutation als *non-synonymous* definiert. Dabei wird ein induzierter Aminosäureaustausch als *missense* bezeichnet, wobei die Änderung zu einem ungewollten Ende der Proteinsequenz führen kann und als *nonsense* Mutation eingestuft wird. Die Indels von einzelnen Basen in genkodierenden Regionen bewirken eine Verschiebung in der zu translatierenden DNA-Sequenz und werden als *frameshift* definiert. Diese verändern die Reihenfolge der DNA-Sequenz und somit die Kodierung der zu transkribierenden Aminosäuresequenz. Hieraus resultiert ein Funktionsverlust oder sogar eine Funktionsänderung des Proteins.

Ähnlich wie zu den OMIM Katalog existieren zu den Punktmutationen viele spezifische Datenbanken, wie *dbSNP* [8], *COSMIC* [9], *APC gene* [10], *GENT* [11] und *1000Genomes* [12]. Diese beinhalten Informationen zu den bisher identifizierten Punktmutationen, wie beispielsweise deren mögliche Auswirkungen auf die Funktion betroffener Gene, dieser Datensatz wird stetig aktualisiert. Die Punktmutationen, oder auch als *single nucleotide variant* (SNV) bezeichnet, können anhand der Frequenz ihres Auftretens in einer Population, die bei über 1% liegen muss, als *single nucleotide polymorphism* (SNP) definiert werden. Die SNPs werden als harmlosere Veränderung des Genoms betrachtet, da ein gewisser Anteil der Bevölkerung diese im eigenen Genom etabliert hat. Somit können Punktmutationen, die auf Grund ihrer Frequenz in einer gesunden Population, SNPs, vorkommen, von bisher undefinierten oder potentiell krankheitsassoziierten Mutationen, SNVs, unterschieden werden. Zudem ist die Lokalisation der SNV entscheidend. Ist die Mutation in einem Gen, dessen Exon, Intron oder in der nicht-translatierten Region zu finden, können unterschiedliche Auswirkungen postuliert werden. Mutationen im Exom ändern oft die Aminosäuresequenz des betroffenen Genprodukts. Somit bergen diese ein viel höheres Potential zur Änderung der Zellphysiologie, da die Mutationen beginnend vom Genom über das Transkriptom bis hin zum Proteom vorkommen können.

Mutationen die mehrere Basen betreffen werden in der Regel als Insertionen, Deletionen, Inversionen, Amplifikationen oder Translokationen klassifiziert und können Auswirkungen auf Gensegmente, oder auch einzelne Chromosomenarme bis hin zu gesamten Chromosomen, haben (Abbildung 1). Dabei werden Amplifikationen oder Deletionen gesamter Chromosomen unter dem Begriff der Aneuploidie zusammengefasst. Zur Gruppe der strukturellen Variationen (SV) gehören Translokationen, Inversionen, Amplifikationen und Deletionen von DNA-Segmenten im Chromosom. Diese können Umlagerungen von Gensegmenten implizieren, welche zu Kombinationen von Genen sogenannten Fusionsgenen führen können. Diese können beispielsweise zu unkontrollierter Expression der Gene führen, da die jeweiligen

Kontrollmechanismen durch die Fusion durchbrochen werden. Zudem werden Zugewinne oder Verluste von DNA-Segmenten, die Regionen von Genen oder gesamte Gene betreffen, als Veränderung der Kopienanzahl oder auch als *copy number variation* (CNV) definiert (Abbildung 1). Hierbei ist ein zuvor nicht erwartetes hohes Aufkommen von CNV im menschlichen Genom zu beobachten. Es konnte beispielsweise gezeigt werden, dass in einem Individuum bis zu 1,2% des Genoms durch CNVs und nur 0,1% durch SNPs, im Vergleich zum Referenzgenom, verändert wurde [13]. Zudem konnten Redon und Kollegen bei einer Gruppe von 270 sequenzierter Individuen zeigen, dass bis zu 12% des gesamten menschlichen Genoms von CNV betroffen sind, das entspricht etwa 360 Mega-Basenpaare (Mbp) [14]. Die Auswirkungen der CNV können eine quantitative Änderung der Genexpression bewirken, vor allem wenn Gensegmente amplifiziert oder gelöscht werden. Die Deletion von DNA-Segmenten führt oft zu einem weiteren Effekt, diese Veränderung des Genoms beinhaltet den Verlust der Heterozygotie, auch als *Loss Of Heterozygosity* (LOH) bezeichnet (Abbildung 1). Dabei ist der Verlust des Allels der betreffenden Region auf der DNA-Ebene gemeint. Eine LOH kann auch ohne Verlust des Allels einhergehen, dabei kann eine Amplifikation vor der Deletion des betroffenen DNA-Segments erfolgen, oder durch Rekombinationen können zwei gleiche Allele im Abschnitt existieren. Dies wird entsprechend als eine LOH ohne Verlust der Allelanzahl oder auch als *copy neutral* LOH (cnLOH) definiert (Abbildung 1).

Bewirkt eine Mutation den Funktionsverlust eines Genprodukts, so kann die Produktion eines funktionsfähigen Genprodukts im diploiden Chromosomensatz gewährleistet werden. Jedoch können Mutationen zur Kanzerogenese führen, hierzu wurde die Knudson Hypothese entwickelt und untersucht. Dabei ist die Inaktivierung der ersten Variante eines Gens als erster Treffer (*first hit*) beschrieben, welcher durch eine Mutation, aber auch durch eine vererbte Keimbahnmutation verursacht werden kann. Zunächst ist die Funktion des betroffenen Gens durch die zweite noch aktive Variante gewährleistet. Kommt es nun zur Inaktivierung der zweiten noch funktionsfähigen Variante, beispielsweise durch eine Mutation, dann wird dies als zweiter Treffer (*second hit*) definiert und die Funktion des Gens ist verloren. Diese Hypothese auch unter den Namen *two hit hypothesis* bekannt, wurde von Knudson und Kollegen in Retinoblastomen verifiziert [15].

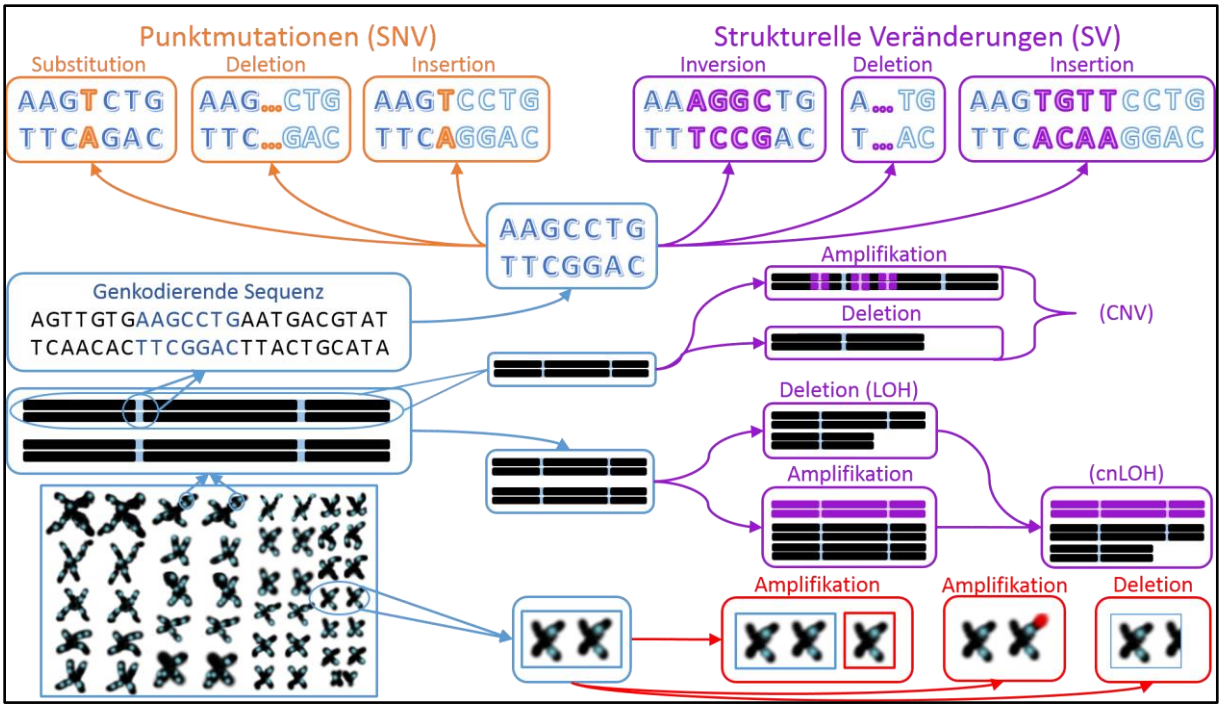


Abbildung 1: Schematische Darstellung der Mutationsklassen
Schematische Darstellung der Mutationsklassen wie die Amplifikation und Deletion gesamter oder Teile der Chromosomen (Rot). Amplifikationen, Deletionen, Insertionen und Inversionen einzelner Bereiche im Chromosom, wodurch CNV, LOH oder cnLOH entstehen können (Lila). Punktmutationen wie Substitutionen, Insertionen und Deletionen (Orange).

3.2 B-Zell Reifung

Die Entwicklung von hämatopoetischen Stammzellen über unreife B-Zellen, transitionalen und reifen B-Zellen, bis hin zu den Antikörper produzierenden Plasmazellen, findet in den lymphatischen Organen, wie Knochenmark, Lymphknoten und Milz statt (Abbildung 2). Die B-Zell Reifung durchläuft verschiedene Phasen wie die schnelle Zellteilung, die somatische Rekombination und die Hypermutationen in definierten Standorten unterschiedlicher Organe. Dies ist ein wesentliches Unterscheidungsmerkmal der B-Zellen zu anderen Zelltypen wie beispielsweise den T-Zellen [16]. Die erste Entwicklungsstufe im Knochenmark ist Antigen unabhängig und inkludiert die Umlagerung von Gensegmenten des Immunglobulins M (IgM), wobei Adhäsionsmoleküle und Integrine die Interaktion zwischen Stroma-, hämatopoetischen Stamm- und B-Zellen intensivieren [16; 17]. Speziell für die schwere Kette des Antikörpers können die Segmente in drei unterschiedliche Gruppen unterteilt werden, dazu gehören die als V- , D- und J-Segmente bezeichnet werden [18]. Im ersten somatischen Rekombinationsschritt werden zunächst die circa 27 D- und 6 J-Segmente der schweren Kette kombiniert. Bevor die etwas über 50 V-Segmente mit den entstandenen DJ-Segmenten kombiniert werden und zu einer Vielfalt an möglichen Ergebnissen führen (Abbildung 2). Der zweite somatische Rekombinationsschritt betrifft die leichte Kette des Antikörpers. Diese enthält V- und J-

Segmente, aber keine D-Segmente. Die Kombination aller möglichen Anordnungen der schweren und leichten Kette führen zu einem großen Repertoire an unterschiedlichen Antikörpervariationen [17]. Dieses Repertoire kann noch in Abhängigkeit der DNA-Doppelstrangbrüche und Umlagerungen, sowie deren induzierten Reparaturen mit unspezifischem Einbau von Basen, an den Schnittpunkten erweitert werden [18].

Sobald die somatische VDJ-Rekombination zu funktionellen membranständigen Antikörpern IgM und IgD, auch B-Zell-Rezeptoren (BZR) genannt, auf der Zelloberfläche führt, erreicht die pro-B-Zelle das sogenannte Prästadium. Durch positive Selektion der proliferativen Zellklone werden entstandene autoreaktive Varianten der Zellen inaktiviert oder eliminiert (Abbildung 2). Da die Häufigkeit dieser Zellen sehr hoch ist, können vor der induzierten Apoptose der autoreaktiven Varianten erneute Rekombinationen der leichten Kette durchgeführt werden. Danach verlassen die unreifen B-Zellen das Knochenmark und die transitionalen B-Zellen zirkulieren im Blut, Milz oder Lymphknoten. Im Blut oder Milz kann der Kontakt mit pathogenen Bakterien zur Entwicklung von kurzlebigen Plasmazellen mit niedriger Affinität führen (Abbildung 2). Dabei spielen bestimmte Komponenten der Bakterienmembranen eine wichtige Rolle bei der Erkennung durch die unreifen B-Zellen. Diese T-Zellen unabhängige Differenzierung von B-Zellen zu IgM sezernierenden Plasmazellen dient als erste Maßnahme gegen Pathogene im peripheren System [16].

Der Kontakt zwischen unreifen B-Zellen mit spezifischen Antigenen und aktivierten CD4-T-Zellen führt zur weiteren Entwicklung der unreifen B-Zellen. Dabei verarbeiten Tyrosinkinasen intrazellulär die Signale von BZR [19]. Die aktivierten B-Zellen zentralisieren und proliferieren im Lymphknoten zu einem Keimzentrum, welches in eine dunkle und eine helle Zone unterteilt werden kann (Abbildung 2). Die Zellteilung der B-Zellen in der dunklen Zone konnte auf etwa 5 bis 12 Stunden bestimmt werden [20]. Die akkumulierten Zellen, auch Zentroblasten genannt, unterlaufen einer weiteren Veränderung der Antikörper, dieser Vorgang wird als somatische Hypermutation (SHM) bezeichnet (Abbildung 2). Dabei werden einzelne Punktmutationen, Deletionen und Duplikationen in den variablen Regionen der Antikörper durchgeführt. Dies und die anschließende multiklonale Proliferation der veränderten Zellen dient zur Affinitätssteigerung der BZR, dazu wurde als essentieller Schlüsselpartner das Enzym *Activation Induced Cytidine Deaminase* (AID) identifiziert [21]. Zentroblasten können nach ihrer Transformation als Zentrozyten in die helle Zone des Keimzentrums wandern. Daraufhin können die Zellen zur dunklen Zone zurück wandern und sich erneut der SHM unterziehen oder durch den Kontakt mit spezifischen follikulären dendritischen Zellen (FDZ) sich den nächsten Selektionsschritt unterziehen. Dabei entscheidet die Affinität der einzelnen Antikörper, ob die

notwendigen T-Zellen zu dem Komplex gebunden werden können. Reicht die Affinität nicht aus, können die B-Zellen erneut in die dunkle Zone wandern oder die Apoptose der Zentrozyten wird eingeleitet [16]. Sobald der Komplex aus FDZ, Zentrozyt und T-Zellen eine ausreichende Affinität der Antikörper erreicht, ist der nächste Schritt die Differenzierung zur Plasmazelle oder B-Gedächtnis-Zelle. Während der Differenzierung zur Plasmazelle können weitere Isotope der Immunglobuline IgA, IgG und IgE durch die *Class Switch Recombination* (CSR) entstehen. Dabei werden Abschnitte der konstanten Region der schweren Kette des bestehenden Antikörpers neu angeordnet oder entfernt, hierbei spielt erneut AID eine wichtige Rolle [21]. Durch die CSR werden nicht die Antigen bindenden Regionen beeinflusst, sondern die konstanten Regionen umgelagert (Abbildung 2). Somit besitzen die Isotope zwar die gleiche Spezifität, haben aber unterschiedliche Aufgaben wie beispielsweise IgA. Dieser Antikörper-Typ ist in den Schleimhäuten vertreten und verhindert das Eindringen von Fremdkörpern. IgM und IgG unterscheiden sich in ihrer Affinität zum Antigen, dienen aber beide als Immunantwort gegen Bakterien, Viren oder Impfstoffe im Körper. IgE ist hauptsächlich bei der Bekämpfung von Parasiten und bei Allergien beteiligt. Die Aufgabe von IgD ist noch nicht eindeutig geklärt, jedoch ist dieses Paraprotein in allen Vertebraten, außer Vögeln, vertreten [22].

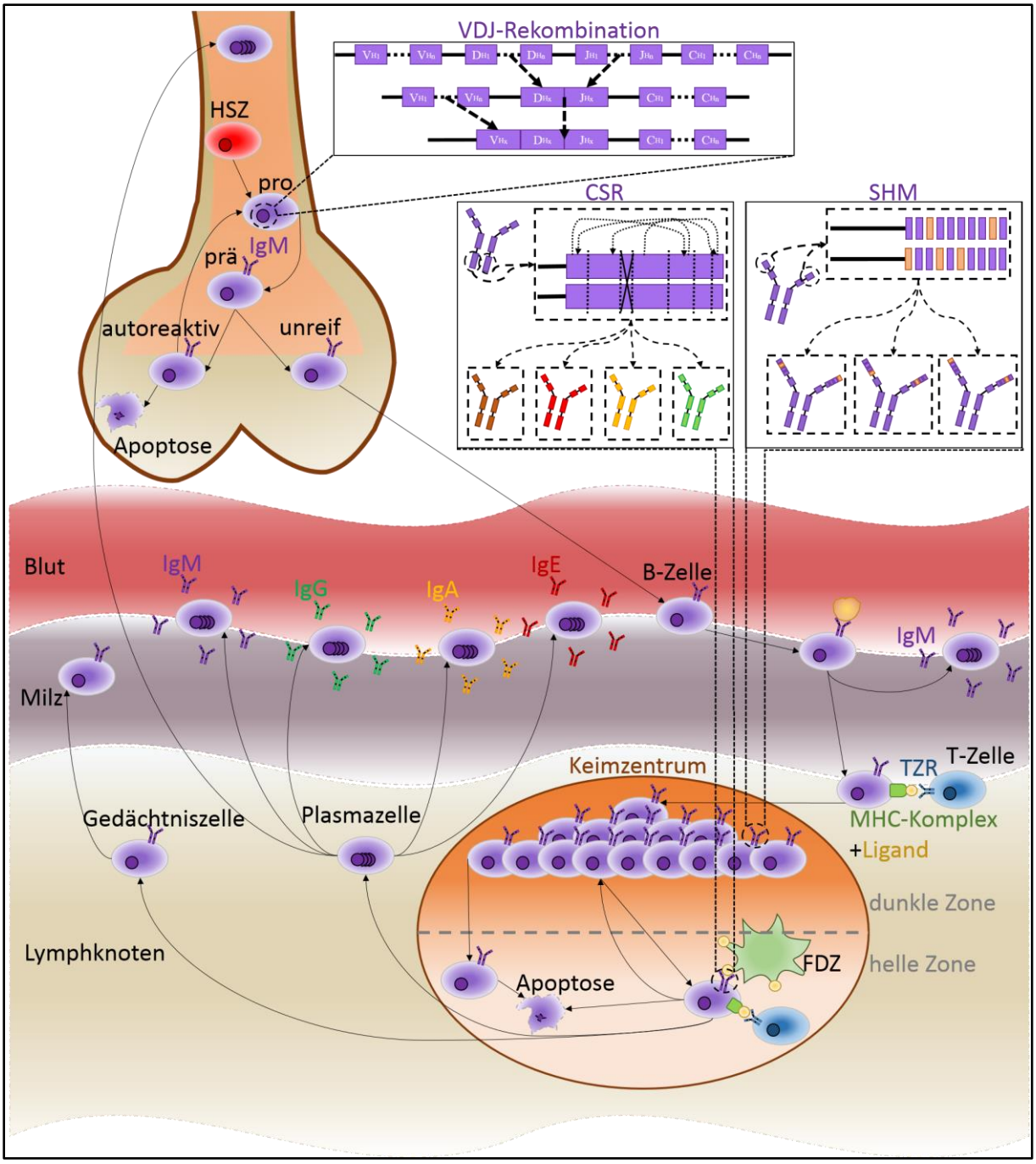


Abbildung 2: Schematische Darstellung der B-Zell Reifung
Schematisch Darstellung der Entwicklung von hämatopoetischen Stammzellen über unreife B-Zellen, transitionalen und reifen B-Zellen bis hin zu den Antikörper produzierenden Plasmazellen. Detailierte Beschreibung im Text (3.2).

3.3 Entartung der B-Zellen

An bestimmten Zeitpunkten der B-Zell-Entwicklung können durch noch unbekannte Ereignisse entartete Zellen proliferieren und nicht mehr von dem eigenen Immunsystem reguliert werden. Wie schon im Kapitel Mutationen (3.1) erwähnt begünstigen beispielsweise Umweltfaktoren, UV-Strahlen, Radioaktivität und Lebensgewohnheiten die Entstehung von Mutationen. Diese können dann zu einer Entartung von Zellen führen [1]. Bei der Diagnose und zur Planung der

Therapie werden die entarteten Zellen hinsichtlich ihrer Morphologie, Histologie, ihrem Immunophänotyp und genetischen Eigenschaften untersucht. Hierbei ist die Häufigkeit von chromosomalen Translokationen, die in Immunglobulin betreffenden Regionen auftreten, bei B-Zell Lymphomen hervorzuheben [23]. Dabei werden Gene in die Expressionsmaschinerie der Immunglobuline transferiert und unterliegen somit den gleichen Kontrollmechanismen wie die Antikörper produzierenden Gene. Diese Gene werden als Onkogene bezeichnet und auf der Grundlage ihrer Lokalisation können Rückschlüsse auf den Entstehungs-Zeitpunkt und -Ort gezogen werden. So ist zum Beispiel das Vorkommen von Onkogenen in der konstanten Region der Antikörper ein Hinweis darauf, dass der Zeitpunkt der Entstehung dieser Translokation während der CSR in der hellen Zone des Keimzentrums festzulegen ist. Des Weiteren können Onkogene in den variablen Regionen bei der VDJ-Rekombination im Knochenmark eingebaut oder bei zusätzlich inkludierten somatischen Punktmutationen durch die SHM in der dunklen Zone des Keimzentrums verändert werden [19]. Ein weiterer Hinweis, dass die CSR und SHM eine sehr wichtige Rolle bei der Entartung von B-Zellen spielen ist, dass die T-Zellen bei ihrer Entwicklung diese Schritte nicht durchlaufen und die beobachtete Häufigkeit von B-Zell-Tumoren zu T-Zell-Tumoren in der westlichen Welt um ein vielfaches höher ist (95% zu 5%) [19]. Im Gegensatz zu den konstitutiv exprimierten Onkogenen können Gene, die zum Kontrollmechanismus der Zelle gehören durch Mutationen in ihrer Funktion gestört werden. Diese Gene werden als Tumorsuppressorgene bezeichnet und ihre vollständige Inaktivierung impliziert das beide Varianten der unterschiedlichen Allele betroffen sind. Dabei kann eine Variante des Gens sowohl eine somatische als auch eine Keimbahnmutation besitzen, ohne dass die Zelle entartet. Die Funktion des geschädigten Gens kann von der zweiten Variante des Tumorsuppressors übernommen werden und erst wenn eine zweite somatische Mutation die entsprechende zweite Variante des Tumorsuppressors inaktiviert, kann eine Entartung der Zelle initiiert werden. Zu den ersten Ereignissen der Two-Hit-Hypothese von Knudson können Veränderungen des Genoms, wie die genetischen Prädispositionen in Form von SNPs, Deletionen und auch epigenetische Veränderungen gezählt werden.

Weitere Studien die auf den Ergebnissen von Hochdurchsatztechnologien basieren, zeigen dass die Phasen der Kanzerogenese nicht nur auf einen linearen Verlauf hinweisen. So können intraklonale Populationen der entarteten Zellen mit unterschiedlichen Mutationsmustern in unterschiedlichen Populationsgrößen am Entstehungsort identifiziert werden. Auch bei Metastasen der jeweiligen Krebsentität können sowohl intraklonale als auch interklonale differenzierbare Klone identifiziert werden [24; 25]. Dies wird als Hauptgrund für einen Rezidiv bei Patienten erachtet, da sich zum Diagnosezeitpunkt meist nur ein Hauptklon (*major*

clone) durchsetzt. Die folgende Therapie ist dann meist auf den Hauptklon abgestimmt während sich die Nebenklone (*minor clone*) der Therapie entziehen und ungehindert vermehren können. Ein Erfolg der Therapie ist in solch einem Fall gering und in der Regel mit einem Rezidiv verbunden.

Die Mehrheit der entarteten B-Zellen können auf Grund ihrer Oberflächenproteine als auch ihrer Lokalisation einem Stadium der B-Zell Reifung und somit einer entsprechenden B-Zell Erkrankung zugeordnet werden. Für das Multiple Myelom ist die Entartung von Plasmazellen repräsentativ, während entartete Zentroblasten für das Follikuläre Lymphom und das Burkitt Lymphom beobachtet werden [19].

3.4 Hämatologische Neoplasien

Die hämatologische Neoplasie beschreibt eine autonome Vermehrung von Erythrozyten, Leukozyten oder Thrombozyten des blutbildenden Systems. In der Onkologie ist besonders die pathologische Vermehrung der Leukozyten von Bedeutung. Nach der Klassifizierung der *World Health Organisation* (WHO) von 2008 werden hämatologische Neoplasien entsprechend ihrer Entstehungszelle in myeloische oder lymphatische Neoplasien unterteilt. Des Weiteren wird bei den lymphatischen Neoplasien die Zugehörigkeit zu den T-Zellen und B-Zellen unterschieden. Die in dieser Thesis relevanten Krebsentitäten wie das Multiple Myelom, Follikuläre Lymphom und Burkitt Lymphom gehören zu den aus lymphatischen B-Zellen entstandenen hämatologischen Neoplasien. Das Follikuläre Lymphom und das Burkitt Lymphom zählen zu den Non-Hodgkin Lymphomen, sowie das das Multiple Myelom, welches auch zur Gruppe der Plasmazell Neoplasien zugeordnet wird [26]. Zur Klassifizierung von Tumoren werden morphologische, zytogenetische und histologische Eigenschaften bewertet, um die dazu entsprechende Therapie zu spezifizieren und Prognosen erstellen zu können. Zudem werden Ergebnisse der molekularen Genetik, der Gene Expression Arrays oder der Genom Sequenzierung zu Rate gezogen.

3.4.1 Multiples Myelom

Das Multiple Myelom, auch als medulläres Plasmazytom bezeichnet, hat ein Auftreten von etwa 10 - 15% bei den hämatologischen Neoplasien. Das Durchschnittsalter der Patienten bei Erst-Diagnose liegt bei circa 70 Jahren. Ein Erkennungsmerkmal ist die monoklonale Vermehrung von Immunglobulinen im peripheren System und die steigende Konzentration an Plasmazellen im Knochenmark. Somit werden entartete Plasmazellen als zellulärer Ausgangspunkt für die Entstehung des MM postuliert (Abbildung 3). Das Auftreten klonaler Akkumulation der sehr spezifischen Immunglobulinklassen, wie IgG und IgA, liegt beim MM

jeweils bei 50% und bei 20% [26]. Entwicklungsphasen des MM, wie das extramedulläre MM und die Plasmazell Leukämie (PZL), induzieren Organschäden, außerdem werden zur Diagnose weitere Faktoren wie Hyperkalzämie, Niereninsuffizienz, Anämie und Knochenläsionen beobachtet. Zusammengefasst ist dies unter dem Begriff *hypercalcemia, renal insufficincy, anemia, bone lesions* (CRAB) bekannt [26].

Die Transformationen bei den unterschiedlichen Phasenübergängen von MGUS über SMM (auch als asymptomatisches Myelom bezeichnet), MM, extramedulläres MM bis hin zur PZL scheinen von bestimmten Ereignissen abhängig zu sein und können als Model zur Tumor Progression dienen. Die Hypothese, dass Tumore aus einer entarteten Zelle eine lineare Entwicklung beschreiten, konnte mit Hilfe der durch Hochdurchsatzsequenzierung gewonnenen Erkenntnisse nicht beobachtet werden. Vielmehr zeigt sich, dass es sich um eine verzweigte Entwicklung der einzelnen entarteten Klone handelt. Hierbei wird die darwinistische Evolutionstheorie der sich entwickelnden Klone zu Grunde gelegt [24]. Die Initiation der Entartung von Plasmazellen wird durch primäre Ereignisse definiert, dabei können diese in zwei Gruppen unterteilt werden, die sich gegenseitig ausschließen. Die nicht-hyperdiploiden Myelome inkludieren chromosomale Translokationen, wie t(4;14), t(6;14)/t(11;14) und t(14;16)/t(14;20), in den Regionen der schweren Ketten der Immunglobuline. Diese entstehen während der CSR und können in circa 50% aller MM lokalisiert werden. Die jeweiligen betreffenden Onkogene sind FGFR3/MMSET, CCND3/CCND1 und MAF/MAFB. Die CSR unabhängigen chromosomalen Translokationen, wie beispielsweise t(8;14) entstehen im späteren Verlauf und werden zu den sekundären Ereignissen gezählt [27; 28]. Die Gruppe der hyperdiploiden Myelome besitzen Trisomien in den Chromosomen 3, 5, 7, 9, 11, 15, 19 oder 21. Die Entstehung ist noch nicht geklärt, jedoch existiert die Hypothese, welche eine abrupte Unterbrechung bei der Mitose als Auslöser der Entstehung definiert. Weitere sekundäre Ereignisse sind Veränderungen der vorhandenen Anzahl an bestimmten Abschnitte des Chromosomes. Dabei werden Kopien dieser Abschnitte in den Chromosomen integriert, diese können eine Deregulation der Zellen bewirken [27; 28]. Zudem konnten Sequenzierungsstudien mit MM Patienten zeigen, dass somatische Mutationen in wichtigen Signalwegen, wie beispielsweise der Translation, Histonmethylierung, Blutgerinnung, in NF-κB, den Rezeptor-Tyrosin-Kinasen (RTK) und Adhäsionsmoleküle akkumuliert vorkommen [29; 30]. Diese große Vielfalt an unterschiedlichen und mehrstufigen Transformationen während der Tumorprogression spiegelt die Heterogenität des MM wieder. Hinsichtlich der tumorspezifischen Mutationsrate konnte beim MM im Durchschnitt 2,9 bis 4,52 Punktmutationen pro 1 Million Nukleotide mehr als bei den korrespondierenden

Normalproben festgestellt werden [30; 31]. Dies entspricht etwa 13200 tumorspezifischer Punktmutationen verteilt im gesamten Genom.

3.4.2 Follikuläres Lymphom

Das Follikuläre Lymphom wird durch die namensgebende Entstehung von untypischen follikulären Zusammenlagerungen von B-Zellen im Keimzentrum charakterisiert. Dabei ist keine klare Differenzierung der dunklen und hellen Zone erkennbar, die Zentroblasten und Zentrozyten verteilen sich zufällig. Somit werden entartete B-Zellen im Keimzentrum, vor allem die Zentroblasten, als zellulärer Ausgangspunkt für die Entstehung des FL postuliert (Abbildung 3) [32]. Die durch das Mikroskop beobachtete Quantität der Zentroblasten in den entarteten Follikeln dient zur Einteilung in folgende Grade wie 1, 2, 3A und 3B. Dabei wird die jeweilige Anzahl von erkennbaren Zentrozyten, wie 0 - 5, 6 - 15 oder > 15, als Mittelwert mehrerer beobachteter Regionen, die einer bestimmten Größe entsprechen (0,159mm^2) quantifiziert. Zur Einteilung zwischen den Graden 3A und 3B ist entscheidend, ob vereinzelt noch Zentrozyten oder eine solide Fläche von Zentroblasten mikroskopisch erkennbar sind [26]. Das Durchschnittsalter der Patienten bei der Diagnose beträgt etwa 60 Jahre und das FL ist mit einem Auftreten von über 20% das zweithäufigste Lymphom. Wesentliches molekulares Erkennungsmerkmal ist die Translokation t(14;18) in circa 85% der systemischen FL. Diese bewirkt die Deregulation der Expression des *BCL2* Gens, wodurch die Apoptose der Zellen im Keimzentrum verhindert wird. Dies ermöglicht eine weitere Reifung von B-Zellen, die eigentlich abgebaut werden sollten und hat zur Folge, dass weitere chromosomale Translokationen induziert werden können [26; 32; 33]. Die Entstehung der t(14;18) Translokation, während der VDJ-Rekombination im Knochenmark, ist wahrscheinlich ein primäres Ereignis in der FL Entwicklung. Als Indiz dient hierzu das beobachtete Auftreten dieser Translokation in 50 – 70% der zirkulierenden B-Zellen in gesunden Individuen [32; 34]. Als sekundäre Ereignisse werden eine Deletion im Chromosom 6, Trisomie des Chromosoms 7 oder 12, Duplikate des Chromosom X sowie somatische Mutationen in Histon modifizierenden Genen wie *EZH2*, *MLL2*, *CREBBP* und *EP300* gezählt [35; 36].

Beim indolenten FL können durch diverse Ereignisse bis zu 40% der FL-Entitäten zu aggressiveren Non-Hodgkin-Lymphomen transformieren, wie beispielsweise zum diffus großzelliges B-Zell Lymphom (DLBZL). Dabei ist zu beobachten dass nicht die Transformation im Hauptklon des FL stattfindet, sondern meist ein Subklon zu einem agressiven DLBZL transformiert [33; 37].

Die Pathogenese der t(14;18)-negativen FL ist zum größten Teil noch unklar. Alternative

genetische Veränderung die eine anti-apoptotische Auswirkung wie bei den t(14;18)-positiven FL hervorrufen, konnten noch nicht identifiziert werden. Komparative Studien zeigten jedoch zur Pathogenese relevante potentielle Kandidaten in t(14;18)-negativen FL, wie beispielsweise *BCL6* Umlagerungen, erhöhte Expression von BCL-X_L sowie durch micro Ribonukleinsäure (RNA)-Analysen identifizierte microRNAs (miRs) (miR-101, miR-138, miR-16, miR-29c und miR-26a) [33; 38]. Die Tatsache, dass Zelllinien von FL nicht kultiviert werden können, erschwert die funktionelle Charakterisierung von genetischen Alterationen im FL und ist zeitgleich ein weiteres Indiz dafür, dass das Mikromilieu eine wichtige Rolle in der Pathogenese der entarteten B-Zellen im FL spielt [39].

3.4.3 Burkitt Lymphom

Das Burkitt Lymphom mit einer Inzidenz von 2,4% besitzt eine sehr hohe Proliferationsrate der entarteten B-Zellen und wurde als erster Tumor mit den Infektionen von Viren wie dem Epstein-Barr-Virus (EBV) und dem Human-Immundefizienz-Virus (HIV) assoziiert [26; 40]. Die drei Hauptgruppen sind das endemische, sporadische und immundefiziente BL. Das endemische BL wird anhand des gleichzeitigen geografischen Auftretens mit Malaria assoziiert und in allen Fällen kann zudem das EBV identifiziert werden. Das sporadische BL ist in der ganzen Welt vertreten und weder eine ortsgebundene noch eine EBV Assoziation konnten bisher beobachtet werden. Das immundefiziente BL wird hauptsächlich in HIV-positiven Patienten identifiziert, und in weniger als 40% dieser Patienten ist das EBV vertreten [26; 40]. Translokationen von cMYC werden in 90% aller BL Patienten beobachtet und können die schwere oder die leichte Kette des Immunglobulins betreffen. Hierbei kann das Auftreten von Translokationen wie t(8;14) zu 80% und t(8;2) Ig-kappa oder t(8;22) Ig-lambda zu 20% in dieser Gruppe der Patienten beobachtet werden. Die durch diese Translokationen inkludierte Deregulation der B-Zellen ist das wesentliche Erkennungsmerkmal des BLs, neben einer kurzen Verdopplungszeit der entarteten Zellen von 24 - 48 Stunden [40; 41]. In den bis zu 10% der gesamten BL Patienten sind keine Myc-Translokationen beobachtet worden.

Einen Standard zur Diagnose hinsichtlich der Morphologie, Immunphänotypisierung und Zytogenetik ist im Einzelnen nicht möglich, somit gestaltet sich die Planung zur Therapiebehandlung schwierig. Wichtig ist hierbei die Differenzierung zwischen BL und DLBZL, da sehr ähnliche Diagnosekriterien, jedoch unterschiedliche Behandlungen, der jeweiligen Entitäten nötig sind. Auf Grundlage von Genexpressionsanalysen konnten Studien zeigen, dass BL und DLBZL spezifische Expressionsmuster besitzen und auf molekularer Ebene differenziert werden können [42; 43]. Des Weiteren wurde beobachtet, dass die cMyc-

Translokationen ein primäres Ereignis in der Pathologie des BL ist. Mit Hilfe der Hochdurchsatzsequenzierung konnte weiterhin beobachtet werden, dass die wiederkehrenden Mutationen der Gene ID3, MYC, TCF3, TP53, GNA13, SMARCA4, DDX3X, TPST2 und RET sekundäre Ereignisse sind [44-46]. Große Sequenzierungsstudien wie beispielsweise *The Cancer Genome Atlas* (TCGA) und ICGC konnten eine tumorspezifische somatische Mutationsrate von etwa 0,3 – 12 Punktmutationen pro 1 Million Nukleotide bei den B-Zell Lymphomen beobachten [47]. Für das BL ist im Durchschnitt eine Mutationsrate von 4,2 pro 1 Million Nukleotide zu erwarten [48].

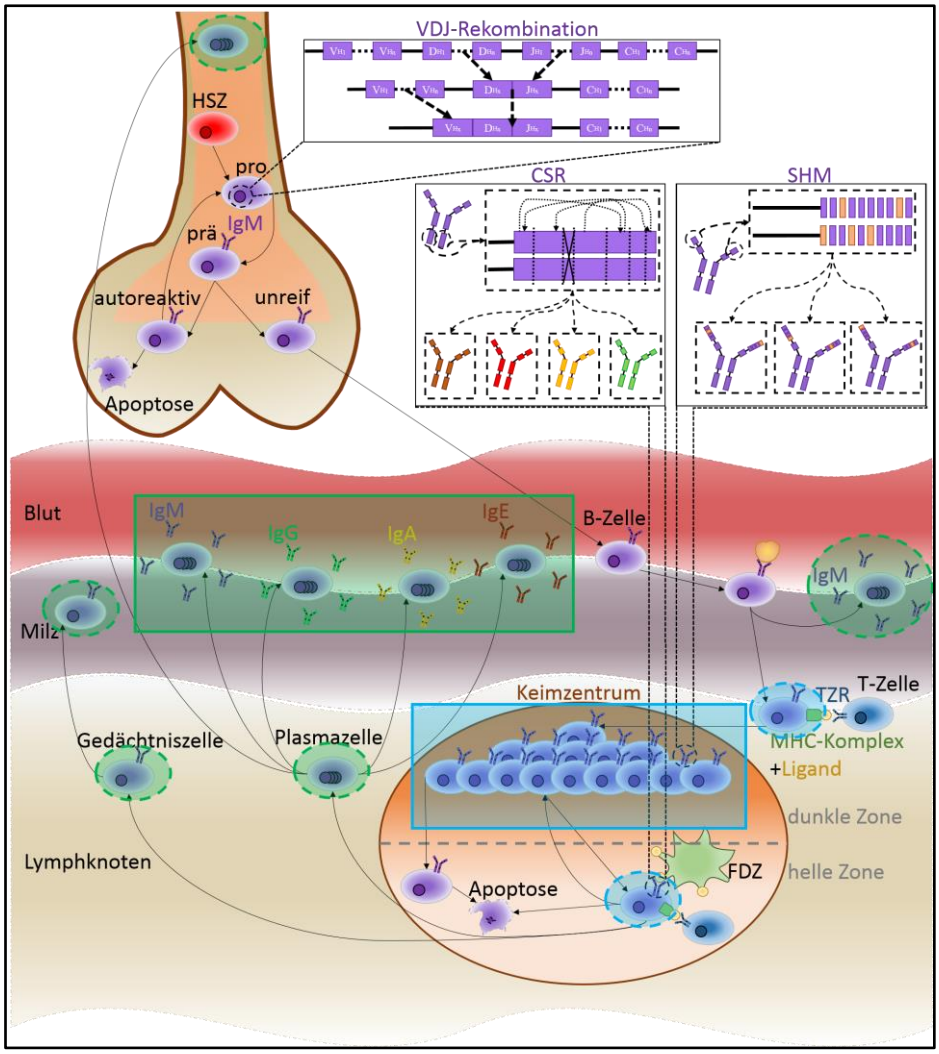


Abbildung 3: Schematisch Darstellung der zellulären Ausgangspunkte bei der Entstehung der jeweiligen Neoplasien
Die postulierten Ausgangspunkte der entarteten B-Zellen sind für das MM mit grün und für das FL und BL mit blau hervorgehoben. Die die gestrichelten Linien repräsentieren alternative Entstehungsorte der entsprechenden Entitäten.

3.5 Genom Sequenzierung

Die Didesoxymethode nach Sanger wurde 1977 entwickelt und dient seit über 35 Jahren als Standardmethode zur DNA-Sequenzierung [49]. Das im Jahre 1990 gestartete Human Genome

Projekt nutzte eine weiterentwickelte Form der Sanger-Sequenzierung, um das erste menschliche Genom zu sequenzieren. Der erste Entwurf dieses Genoms wurde im Jahre 2001 veröffentlicht. Zeitgleich konnte ein privates Unternehmen eine eigene Version des menschlichen Genoms mit Hilfe einer automatisierten Form der Sanger-Sequenzierung erstellen und publizieren. Nachfolgend konnte im Jahre 2004 die nahezu komplette Version NCBI35/hg17 veröffentlicht werden [50-52]. Aktuell ist die neuste Version *Genome Reference Consortium Human Build* (GRCh)38/hg38 verfügbar. In Abhängigkeit der verfügbaren Exom Extraktion-Kits wurden und werden momentan noch häufig die vorherigen Genomversionen NCBI36/hg18 und GRCh37/hg19 verwendet, wobei die GRCh37/hg19 Version noch bis zu 357 Lücken enthält (http://hgdownload.cse.ucsc.edu/goldenPath/hg19/database/gap.txt.gz) [53]. Speziell die Lücken zwischen den chromatischen Regionen entsprechen etwa 30 Mbp. Dazu kommen etwa 200 Mbp, die in den Zentromeren, Telomeren oder akrozentrischen Chromosomen lokalisiert werden können [54]. Limitierungen der momentan verfügbaren Sequenzierungstechnologien führen zur Entstehung dieser Lücken. Hierbei sind nicht zugeordnete Regionen bei der Assemblierung oder nicht sequenzierte Regionen die einen zu geringen oder erhöhten GC-Gehalt in der Sequenzabfolge besitzen verantwortlich [55].

Den ersten Sequenzierer der nächsten Generation konnte 454 Life Sciences im Jahr 2005 herausgeben und somit die Ära der *High-Throughput-Sequencing* (HTS) oder NGS Technologien einleiten. Der Sequenzierer von 454 Life Sciences konnte Sequenzlängen von bis zu 100 Basenpaare (bp) produzieren, 2007 wurde diese Technologie von Roche aufgekauft [56]. Die Firma Solexa konnte ihrerseits 2006 einen NGS-Sequenzierer vorweisen, welcher jedoch nur eine Sequenzlänge von 30bp erreichte. Kurz darauf wurde Solexa von Illumina übernommen [56]. Bei Applied Biosystems begann der Verkauf eines NGS-Sequenzierers im Jahre 2007, dieser produzierte Sequenzlängen von bis zu 35bp und momentan hat Life Technologies die Rechte dieser Technologie erworben [56]. Bei dem Applied Biosystems und Illumina Sequenzierungsmaschinen ist die Ausgabemenge der Ergebnisdaten sehr hoch, wodurch mehr Informationen zu den sequenzierten Regionen generiert werden. Somit ist die hauptsächliche Nutzung dieser Sequenzierer die Resequenzierung von Genomen. Die Roche 454 Technologie generiert zwar längere Sequenzen und ist für die *de novo* Sequenzierung von Genomen prädestiniert, jedoch ist die Gesamtausgabemenge pro Durchlauf verhältnismäßig klein und die Kosten pro Base liegen hierdurch im Vergleich zu den anderen Technologien weitaus höher [57]. IonTorrent entwickelte 2010 einen Sequenzierer, der auf einer neuen Identifizierungsmethode über Messung des Proteomstroms während der Kettensynthese mit einfachen Halbleiternbauelementen basiert. Hierbei entfällt die optische Identifizierung, was

mit enormen Vorteilen hinsichtlich Geschwindigkeit und Kosten verknüpft ist [56].
Im Vergleich der Sanger Sequenzierung waren die neuen Technologien zunächst hinsichtlich der produzierten Sequenzlänge unterlegen, jedoch war und ist die Geschwindigkeit und Ausgabemenge der neuen Sequenzierer um ein vielfaches höher. Weitere Vorteile der NGS gegenüber der Sanger-Sequenzierung sind beispielsweise die nicht Nutzung von Bakteriengenome zur Aufbereitung der Proben und der damit verbundenen Störfaktoren. Zudem erfolgt die Detektion der Basenabfolge ohne Elektrophorese und sowohl die benötigten Reaktionen als auch deren Identifikation kann parallel durchgeführt werden. Dies führt zu einer enormen Verbesserung der Performance. Die benötigte Zeit zur Sequenzierung eines einzelnen gesamten humanen Genoms konnte nun innerhalb einiger Wochen anstatt über eine Dekade durch geführt werden [56]. Die immer währende Entwicklung der Sequenzierungstechnologien konnte sogar die für die Sequenzierung benötigte Zeit eines gesamten Genoms auf unter einen Tag reduzieren. Jedoch obliegt die Validierung von detektierten Mutationen immer noch der Sanger-Sequenzierung, die auch rückwirkend als *First-Generation-Sequencing* bezeichnet wird. Als Nachteile der NGS-Technologie können die höheren Fehlerraten und die kurzen produzierten Sequenzlängen gezählt werden. Diese kurzen Sequenzen können ebenso beim *Alignment* zu höheren Fehlerraten führen, die die Zuordnung der Position im Genom oder die Assemblierung erschweren [56; 58].
Im weiteren Verlauf der Entwicklung neuer Sequenzierungstechnologien entstanden Alternativen zu den bereits gängigen Plattformen. Vereinzelte Erfolge konnten sich auf dem Markt etablieren und leiteten die *3rd-Generation-Sequencing* Ära ein. Das Hauptaugenmerk der neuen Technologien ist die Schwächen der NGS zu verringern, dabei sollen die Sequenzlängen erweitert, die Fehlerraten und die Kosten weiter reduziert werden.

3.5.1 Sequenzierungsplattformen

Die drei am meisten genutzten NGS-Plattformen der letzten Jahre waren jene die zuerst auf dem Markt einen verfügbaren Sequenzierer anbieten konnten, wie Solexa/Illumina, 454 Life Sciences/Roche und Applied Biosystems/Life Technologies. Im Wesentlichen kann bei allen Plattformen der Ablauf der Sequenzierung in drei Schritte unterteilt werden. Zunächst wird die DNA-Probe vorbereitet, danach werden die zu sequenzierenden Sequenzen immobilisiert und amplifiziert, bevor die eigentliche Sequenzierung durchgeführt wird.
Die Vorbereitung der DNA-Probe umfasst die Fragmentierung der genomischen DNA und die Ligation von definierten Adaptern sowie Barcodes an den Fragmenten. Dabei sind die Adapter Plattform spezifisch und können zur Festlegung der Leserichtung bei der Sequenzierung

dienen. Die Barcodes ermöglichen die Differenzierung von Proben, somit können mehrere unterschiedliche Patienten in einem Sequenzierungslauf simultan prozessiert werden. Das Resultat der Aufbereitung, welches insgesamt eine Sequenzierungsbibliothek repräsentiert, ist eine Anhäufung von DNA-Fragmenten mit ligierten Adaptern und Barcodes (Abbildung 4).

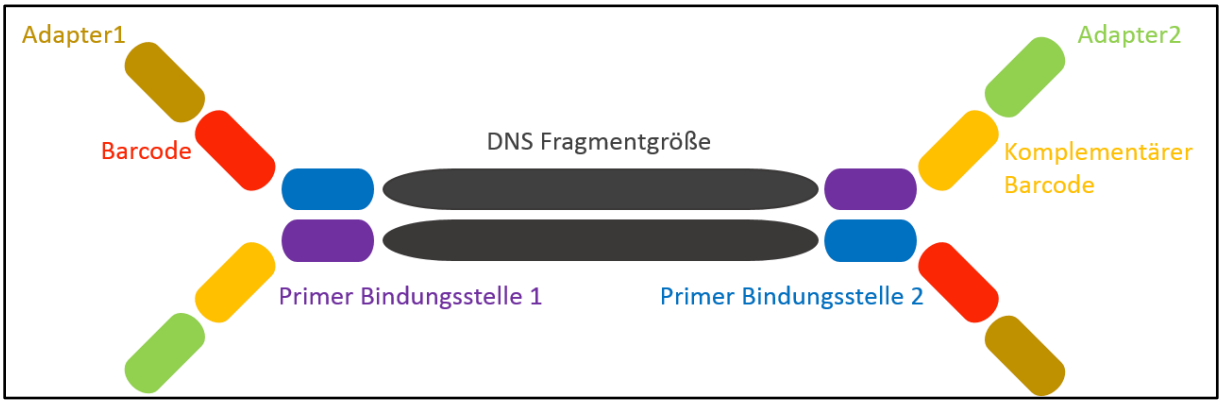


Abbildung 4: Schematisch Darstellung eines aufbereiteten DNA Fragments
Die DNA Fragmente werden mit Adaptern für die Immobilisierung, Barcodes für die Erkennung und den Primer Bindungsstellen ausgestattet. Alle bearbeiteten Fragmente der Sequenzierungsbibliothek entsprechen dieser Darstellung.

Die Immobilisierung der Fragmente ist plattformabhängig. Bei Illumina werden die Fragmente mit Hilfe von fixierten zu den ligierten Adaptern korrespondierenden Sequenzen, direkt an die Oberfläche einer Durchflusszelle, gebunden. Roche und Life Technologies nutzen Beads auf denen die zu den Adaptern korrespondierenden Sequenzen präpariert werden. Die Beads werden dann bei Life Technologies nach der Amplifizierung auf einer Glasfolie fixiert. Roche nutzt eine Picotiter Platte mit Wells in denen nur ein einzelner Bead Platz findet. Zuvor werden die isolierten und immobilisierten Fragmente per *Polymerase Chain Reaction* (PCR) amplifiziert, um das Signal zur Detektion zu verstärken. Die *emulsion* PCR (emPCR) wird für die Sequenzierungstechnologien von Roche und Life Technologies genutzt, wobei für Illumina erst nach der Fixierung auf der Durchflusszelle eine *bridge* PCR zur Amplifizierung durchgeführt wird (Abbildung 5).

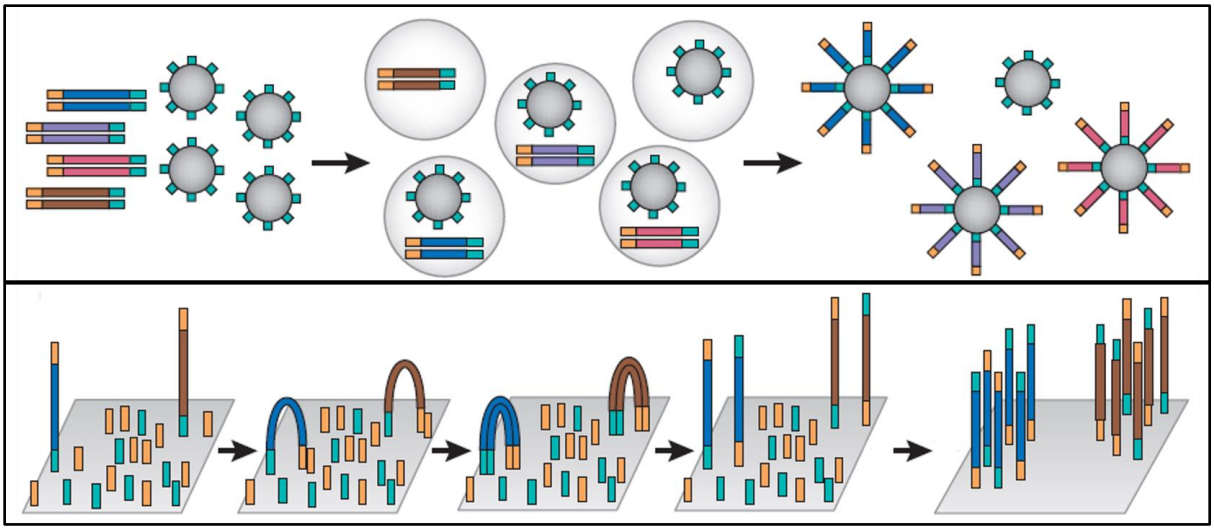


Abbildung 5: Schematisch Darstellung der Immobilisierungs- und Amplifizierungsarten[59]
Im oberen Abschnitt (454 und Solid) werden die DNA Fragmente mit Hilfe der emPCR immobilisiert. Dabei werden die Fragmente an den Beads ligiert und anschließend amplifiziert. Somit repräsentiert ein Bead mit mehreren gleichen Fragmenten jeweils einen Read. Im unteren Abschnitt (Illumina) werden die Fragmente an einer Durchflusszelle ligiert und anschließend mit Hilfe der bridgePCR amplifiziert. Dabei werden etwa 1000 gleiche Fragmente generiert, diese repräsentieren ein Cluster. Die Amplifikationen implizieren eine Signalverstärkung während der Sequenzierung und erleichtern die Basenerkennung.

Die eigentliche Sequenzierung ist bei Roche die Pyrosequenzierung, diese basiert auf den Einbau von entsprechenden Deoxyribonukleotidtriphosphaten (dNTP) mit Hilfe der DNA Polymerase und der dadurch induzierten Abspaltung vom Pyrophosphat pro eingebauten dNTP. Die ATP-Sulfurylase verwendet die abgespalteten Pyrophosphate und Adenosinphosphosulfate zur Generierung von ATP. Die Luciferase nutzt das ATP um die Konvertierung von Luciferin zu Oxyluciferin zu begünstigen (Abbildung 6). Das Qxyluciferin wird von einer Lichtemission begleitet, dieses Signal ist proportional zu den eingebauten Nukleotiden und wird durch eine lichtempfindliche *charge-couple device* (CCD) Kamera detektiert [60]. Werden mehrere gleiche Nukleotide während des gleichen Zyklus eingebaut, so erhöht sich das Lichtsignal im entsprechendem Well. Diese Sequenzabfolgen mehrerer gleicher Basen nennt man Homopolymere, welche als eine der Hauptfehlerquellen bei der Pyrosequenzierung identifiziert werden. Die Signalintensität kann dabei ab sechs eingebauten gleichen Nukleotiden nicht mehr zu der entsprechenden Anzahl zurück gerechnet werden [61].

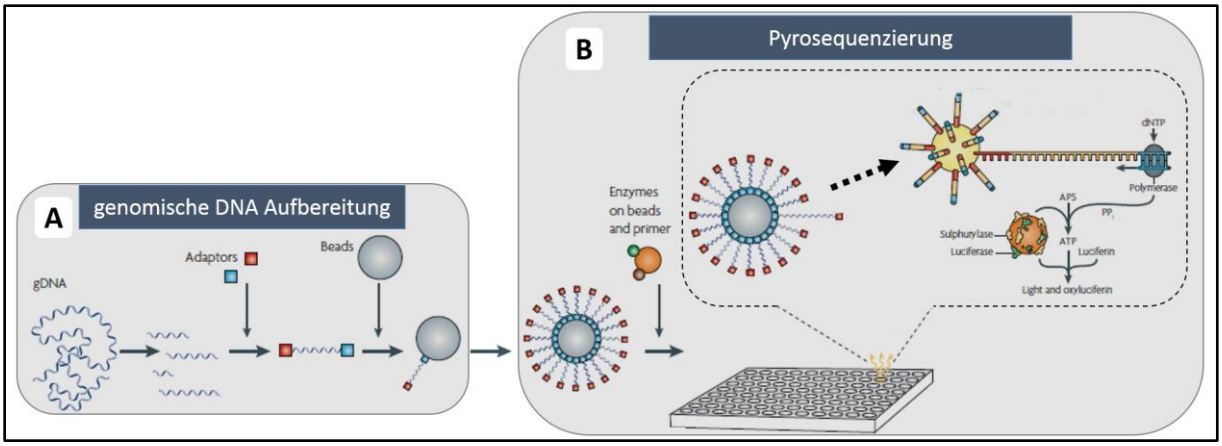


Abbildung 6: Schematische Darstellung der Pyrosequenzierung, zusammengestellt aus Abbildungen von Metzker M. et al. 2010 und Medini D. et al. 2008 [58; 62]
(A) Nach der Fragmentierung der genomischen DNA werden die Plattform-spezifischen Adapter ligiert. Bei der Pyrosequenzierung werden zunächst die Fragmente auf einem Bead immobilisiert und per emPCR vervielfältigt. (B) Anschließend werden die Beads in der Mikrotiterplatte verteilt und pro *well* entsteht beim Einbau einer Base jeweils ein Signal. Umso mehr gleiche dNTPs eingebaut werden umso stärker wird das Signal.

Die Life Technologies Sequenzierung ist auf Grundlage der *Sequencing by Oligo Ligation Detection* (SOLiD)-Methode aufgebaut. Bei dieser Methode bindet ein für die Adaptoren spezifischer Primer an die Position *n* der fixierten Sequenz auf dem *Bead*, welche als Template definiert wird. Des Weiteren können die spezifischen Sonden von SOLiD an das Template hybridisiert werden. Die Sonden enthalten 8 Nukleotide, wobei für die Ligation die ersten zwei Basen zum Template korrespondieren müssen. Die drei folgenden Nukleotide werden nicht, wie die letzten drei, bei der Fluorophor-Abspaltung vom Template entfernt. Es existieren 16 verschiedene Sonden die mit 4 Fluorophoren jeweils 16 mögliche Kombinationen der eingebauten Nukleotide repräsentieren (Abbildung 7).

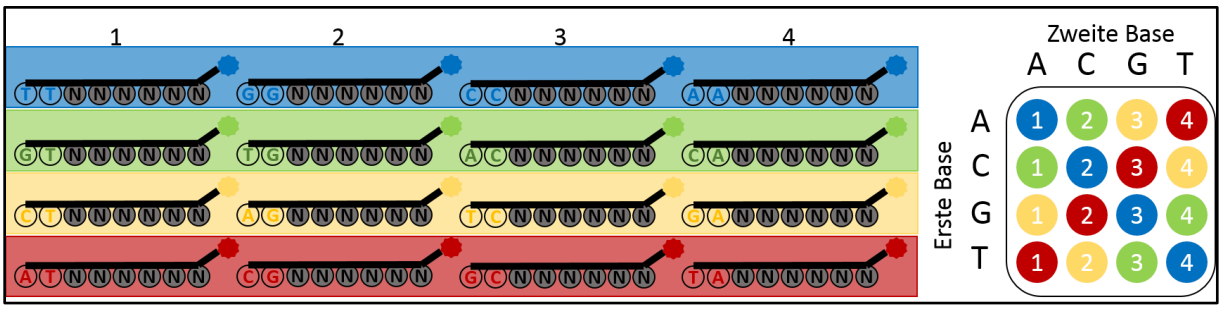


Abbildung 7: Schematisch Darstellung der spezifischen Sonden, veränderte Abbildung nach Valouev et al. 2008[63]
Schematische Darstellung der 16 Sonden mit der Dekodierungstafel, welche bei der SOLiD-Technologie verwendet werden.

Pro Zyklus wird eine Sonde hybridisiert und mit einer Ligase die korrespondierenden dualen Nukleotide an das Template ligiert. Die Fluoreszenzemission des zuvor angeregten Fluorophors wird detektiert und danach mit den drei letzten Basen der Sonde durch Waschen entfernt. Des Weiteren werden, bei einer Sequenzlänge von 50 Nukleotiden, bis zu 9 Sonden nach dem gleichen Schema eingebaut bevor der Primer und die generierte Sequenz entfernt werden. Nun

wird ein neuer Primer eingebaut, der zum vorherigen um genau einen Nukleotid verschoben (*n*-1) ist. Erneut werden, bei einer Sequenzlänge von 50 Nukleotiden, insgesamt 10 aufeinander folgende Sonden nach dem bereits beschriebenen Schema eingebaut. Diese durch den Primer induzierte Verschiebung wird fünfmal durchgeführt, somit werden alle Nukleotide des Templates zweifach detektiert (Abbildung 8) [63-65]. Dies ermöglicht eine zweifache Qualitätsangabe pro sequenzierter Base in der detektierten Sequenz, was jedoch zu einer gesonderten und meist komplexeren Verarbeitung und Speicherung der prozessierten Daten führt.

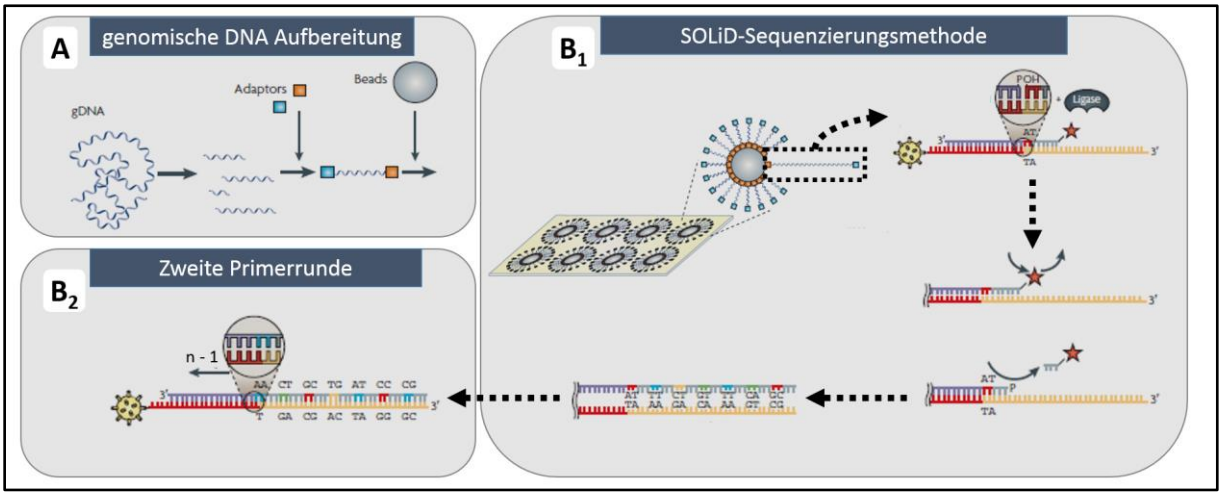


Abbildung 8: Schematische Darstellung der Sequenzierungsmethode SOLiD, zusammengestellt aus Abbildungen von Metzker M. et al. 2010 und Medini D. et al. 2008
(A) Nach der Fragmentierung der genomischen DNA werden die Plattform-spezifischen Adapter ligiert. Die Fragmente werden auf einem Bead immobilisiert und per emPCR vervielfältigt. (B_1) Anschließend werden die Beads auf einer Glasfolie fixiert. Die 1,2Probes werden an die Templates ligiert, bevor die Fluorophore und die letzten Basen der Sonde abgewaschen werden. Dies wird bei einer Readlänge von 50Bp bis zu 9 mal wiederholt. (B_2) Es folgen 4 weitere Primerrunden, wobei ein neuer Primer eingebaut wird, der zunächst um eine Position (n - 1) verschoben ist. Danach folgen wie zuvor die 10 Sonden.

Die Sequenzierung bei Illumina basiert auf der *Sequencing by Synthesis* (SBS)-Methode, hierzu werden ähnlich wie bei der Pyrosequenzierung die dNTPs mit Hilfe einer DNA-Polymerase eingebaut. Diese Polymerase wird nach jedem Einbau eines Nukleotids durch die an den dNTPs gebundenen Terminatoren unterbrochen. Diese Terminatoren sind reversibel und werden abgespalten, nachdem der spezifisch gebundene Fluorophor an den dNTPs angeregt und dessen Emission mit Hilfe der CCD Kamera und den Filtern detektiert wurde. Für die Anregung der Fluorophore werden zwei Laser genutzt. Zur darauf folgenden Messung der Fluoreszenz werden vier unterschiedliche Bilder mit der CCD Kamera und den entsprechenden vier Filtern erstellt. Die vier Bilder werden pro Zyklus zusammengeführt und alle Fluoreszenzspektren betrachtet, somit können die eingebauten Nukleotide detektiert werden. Der Einbau, die Unterbrechung und die Detektion während des Prozess werden bis zur gewünschten

Sequenzlänge wiederholt (Abbildung 9). Das exzessive und gleichzeitig belastende Bearbeiten der DNA während der SBS ist der limitierende Faktor für die zu produzierende Sequenzlängen.

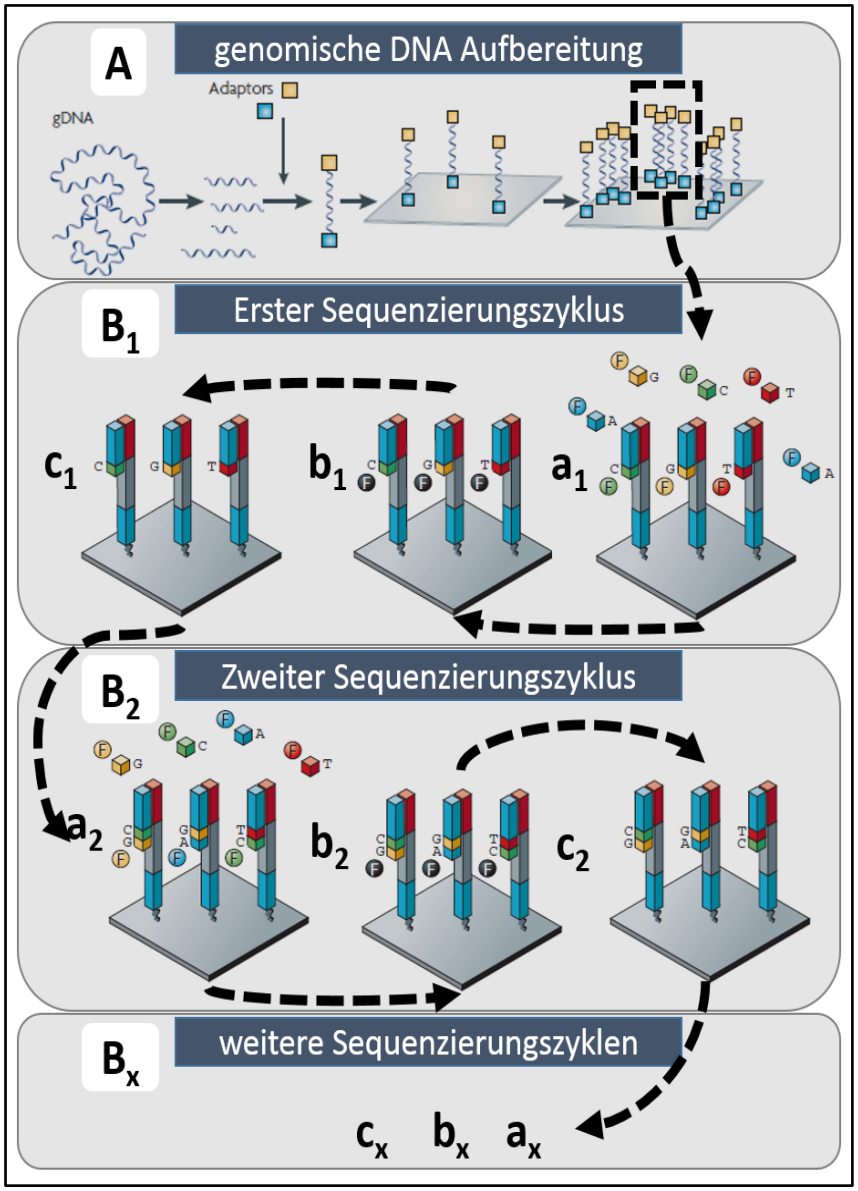


Abbildung 9: Schematische Darstellung der Sequenzierungsmethoden SBS, zusammengestellt aus Abbildungen von Metzker M. et al. 2010 und Medini D. et al. 2008 [58; 62]
(A) Nach der Fragmentierung der genomischen DNA werden die Plattform-spezifischen Adapter ligiert. Die Fragment werden auf einer Durchflusszelle immobilisiert und anschließend mittels der bridgePCR amplifiziert. (B_1) Die dNTPs besitzen einen integrierten Terminator und Fluorophore, somit können die Basen einzeln eingebaut (a_1), die Fluorophore angeregt und die jeweiligen Signale detektiert werden (b_1). Danach werden die Terminatoren abgewaschen (c_1). (B_2) Im zweiten Sequenzierungszyklus wird die neue korrespondierende Base eingebaut (a_2), die Fluorophore angeregt und die Signale detektiert (b_2), sowie die Terminatoren abgewaschen (c_2). (Bx) Dies wird bis zur gewünschten Readlänge wiederholt (a_x b_x c_x). Derzeit sind Reads mit einer Länge von 350 bp möglich.

Die beschriebenen Technologien haben sich während der letzten Jahre entwickelt und verbessert. Illumina konnte dabei hinsichtlich der Faktoren wie Sequenzlänge, Geschwindigkeit, Ausgabemenge und Kosten die größte Entwicklung vollziehen (Tabelle 1) [57; 58; 66-70]. Mit dem HiSeqX Ten konnte Illumina als erste Firma einen Sequenzierer

anbieten, der es ermöglicht ein menschliches Genom für unter 1000$ zu sequenzieren. Die Senkung der Kosten pro Base, konnte Illumina bei gleichbleibender hoher Qualität der Ergebnisse entwickeln. Dieser Technologievorsprung ist ein großer Vorteil gegenüber den anderen Plattformen und erklärt den enormen Erfolg von Illumina. Hinzu kommt der große Support durch diverse Softwaretools, die speziell für Illumina-Daten immer sehr schnell generiert und verbessert werden. Die derzeitig meist genutzte NGS-Technologie für die DNA Sequenzierung ist die von Illumina. Auch in dieser Arbeit ist der Hauptanteil der produzierten Ergebnisse mit Hilfe dieser Technologie erstellt worden. Zu den Burkitt Lymphomen wurden zunächst Sequenzierungsdaten mit der SOLiD-Methode von Life Technologies generiert. Die Validierungsrate der identifizierten Mutationen war jedoch niedriger als die Validierungsraten basierend auf den generierten Sequenzierungsdaten von Illumina. Die Analyse der erneut sequenzierten BL-Daten mit der Illumina-Technologie bestätigten dies. Deshalb basieren im Folgenden die wesentlichen Erläuterungen und bioinformatischen Analysen auf prozessierte Sequenzierungsdaten, die mit der Illumina-Technologie erstellt wurden.

Tabelle 1: Vergleich der Sequenzierungsplattformen

NGS-Plattform	NGS-Methode	Applikation	Genauigkeit	Ausgabemenge
Solexa/Ilumina	SBS	Resequenzierung	98 %	1 – 1800 GBp
Roche	Pyrosequenzierung	*De novo* Sequenzierung	99,5 %	0,02 – 0,7 GBp
Applied Biosciences/Life Technologies	SOLiD	Resequenzierung	99,94 %	3 – 320 GBp

3.5.2 Applikationen

Aus der Entwicklung der bereits beschriebenen Sanger-Sequenzierung, die zur Identifizierung einzelner Segmente des Genoms und zur Zusammenführung jener einzelnen Regionen zu einem gesamten Genom geführt hat, wurden viele Schritte automatisiert. Dies führte zu einer insgesamt schnelleren Sequenzierung und Bearbeitung [71]. Die NGS basierenden Sequenzierungsplattformen, die daraufhin folgten konnten zudem die gleichzeitige Betrachtung von weitaus mehr Regionen ermöglichen. Des Weiteren konnten sowohl die Kosten als auch die benötigte Zeit für die Sequenzierung reduziert werden. Die derzeit meist genutzte Technologie für die DNA Sequenzierung ist die NGS-Technologie, diese ermöglicht die Realisierung von neuen Ansätzen zur Erforschung des Genoms oder die Verbesserung von bereits vorhandenen *Workflows*, wie beispielsweise das Detektieren von SNPs. Die Bereiche in denen die NGS-Technologien genutzt werden sind die Genomik, Transkriptomik und Epigenomik. Die Epigenomik umfasst hierbei die Bisulfit-Sequenzierung zur Identifizierung methylierter Nukleotide und die *chromatin immunprecipitation sequencing* (ChIP-Seq) zur

Identifizierung von Proteinen, die mit bestimmten DNA Regionen interagieren. Die Sequenzierung der gesamten RNA-Konstrukte in den zu untersuchenden Proben dient zur Detektion des Transkriptoms und enthält meist einen Zwischenschritt, wie die Umwandlung der RNA zur komplementären DNA (cDNA), bevor die eigentliche Sequenzierung durchgeführt werden kann. Zur Genomik zählt die Sequenzierung von genomischen DNA-Segmenten, wobei Sequenzierungen gesamter, neuer und nicht annotierter Genome als *de novo* Sequenzierung klassifiziert werden. Existiert bereits ein meist annotiertes Referenzgenom wird die Sequenzierung von gesamten Genomen oder deren Teilabschnitte als Resequenzierung klassifiziert.

Im Vergleich von der bisher genutzten und bereits etablierten Array Technologie zu den NGS-Technologien ist der größte Vorteil der Sequenzierung, dass keine Vorkenntnisse der genauen Basenabfolge der zu detektierenden Sequenzen nötig sind. Das bedeutet, dass bei der DNA-Sequenzierung genomweit unbekannte Mutationen und SNVs detektiert, oder bei der RNA-Sequenzierung des Transkriptoms, Isoformen vorhandener Transkripte oder sogar neue Transkripte, identifiziert werden können. Neuere Algorithmen zur Analysen von NGS-Daten konnten sogar die Detektion von CNV begünstigen, einen Analysebereich der bisher hauptsächlich von den Arrays dominiert wurde. Studien hierzu zeigen dass die Detektion mittels der Sequenzierung des gesamten Genoms (WGS) eine höhere Performance als die Detektion mittels Arrays erreichen kann [72]. Des Weiteren hat auch die Identifizierung von CNVs mittels gezielter Sequenzierungsansätzen hohes Potenzial, wie beispielsweise beim *whole exom sequencing* (WES). Jedoch generiert allein die spezifische Bearbeitung der DNA-Proben für die WES Sequenzierung unüberwindbare Grenzen die eine gesamte Detektion von CNVs verhindern [73].

Es werden immer mehr neue Methoden basierend auf den NGS-Technologien entwickelt, beispielsweise wird zur Analyse bakterieller Infektionen die duale RNA Sequenzierung genutzt. Hierbei wird das gesamte Transkriptom von Zellen, die mit Bakterien infiziert wurden sequenziert [74]. Das Potenzial für die Entwicklung neuer Applikationen basierend auf den NGS-Technologien ist noch nicht erschöpft. Am wichtigsten ist momentan die Diskussion, ob die NGS-Sequenzierung zur Diagnostik zugelassen werden soll. Dies scheint nur noch eine Frage der Zeit, da die *Food and Drug Administration* (FDA) schon eine Genehmigung für den MiSeqDx (Illumina) für die medizinische Diagnostik ausgeschrieben hat. Des Weiteren konnte ein Ringversuch zur Identifizierung von charakteristischen Mutationen in den Genen BRCA1 und BRCA2 von Eierstockkrebspatienten, das Potenzial der NGS-Technologien in der Diagnostik aufzeigen [75]. Derzeit ist die Sanger-Sequenzierung auf Grund der langjährigen

Erfahrungen und sehr niedrigen Fehlerraten die Standardmethode der Diagnostik und somit die Validierungsmethode die genutzt wird, um die durch die neueren Technologien identifizierten Mutationen zu bestätigen.

3.5.3 Gezielte Resequenzierung

Eine sehr oft verwendete Applikation in der Genomik ist die gezielte Resequenzierung von spezifischen Regionen. Dadurch können Veränderungen in Teilen des Genoms identifiziert und interpretiert werden. Diese auch als *targeted resequencing* bezeichnete Applikation, beinhaltet im Gegensatz zur WGS eine Erweiterung der Bearbeitungsschritte während der Vorbereitung der DNA-Proben. Dies ist eine Selektion spezifischer Segmente nach der Fragmentierung des Genoms und vor der eigentlichen Sequenzierung. Hierzu können zur Anreicherung der selektierten DNA-Fragmente unterschiedliche Methoden genutzt werden, welche auf PCR, Zirkularisierung oder Hybridisierung basieren. Bei der Wahl der geeigneten Methode spielt die zu klärende Fragestellung eine wichtige Rolle. So sind beispielsweise für die Identifizierung von *missense* Punktmutationen die proteinkodierenden DNA-Regionen des Exoms essentiell. Diese Regionen betragen etwa 2% des 3,3 Milliarden Basen langen menschlichen Genoms (Abbildung 10). Diese Reduzierung der Sequenzierungsbibliothek auf das Exom, ermöglicht die gleichzeitige Sequenzierung einer größeren Anzahl an unterschiedlicher Individuen und eine tiefere Sequenzierung der Regionen. Dies resultiert durch die Verwendung gleicher Ressourcen, Kosten sowie einer ähnlichen Bearbeitungszeit wie bei der gesamten Genomsequenzierung, für einen viel geringeren Anteil der genomischen DNA.

Die auf Hybridisierung basierende Methode ist am besten geeignet, für die Selektion und Anreicherung eines bis zu 65 Mbp großen Exoms. Bei noch höheren Patientenzahlen und einer kleineren Gesamtregion, wie bis zu etwa 5 Mbp, eignen sich PCR basierende Methoden[76]. Diese weitere Reduzierung der zu sequenzierenden Regionen ist auch eine Applikation der gezielten Resequenzierung und wird als Amplikon-Sequenzierung zusammengefasst. Die Amplikons werden mittels Primerpaaren und der PCR aus spezifischen Regionen des Genoms generiert und ergeben zusammen ein sogenanntes g*enepanel*. Die ausgewählten Regionen können beispielsweise Tumor assoziierte Gene beinhalten, wie beispielsweise das zur Amplikon-Sequenzierung angebotene *tumor genepanel* von Illumina.

Da sich sowohl das Referenzgenom, als auch die Annotationsdatenbanken *Reference Sequence* (RefSeq) und *Consensus CDS* (CCDS) immer wieder durch neuere Forschungsergebnisse verändern, entstehen demzufolge auch neue Kits zur Extraktionsdurchführung. Die am meist genutzten Kits für die Extraktion des Exoms bei der WES sind dabei die Produkte der Firmen

Nimblegen, Illumina und Agilent, die sowohl Vor- als auch Nachteile besitzen (Tabelle 2) [77-79].

Tabelle 2: Vergleich der Exom-Extraktion-Kits, z.T zusammengestellt aus Informationen von Clark et. al 2011 [77]

Firma	Nimblegen	Illumina	Agilent
Version	*SeqCap EZ Exome Library SR v.2.0*	*Nextera Rapid Capture Exome*	*SureSelect Human All Exon*
Region [MBp]	44	37	50
Länge der Oligonukleotide [Bp]	55 - 105	95	114 - 126
Anzahl der Oligonukleotide [n]	>2100000	340427	655872
Hybridsierungszeit [h]	72	24	24

Diese Kits basieren auf zwei alternative Strategien, um ausgewählte Regionen aus dem defragmentierten Genom mittels Hybridisierung zu extrahieren. Bei der *solid*-Methode wird ein Chip mit gespotteten cDNA Sequenzen genutzt. Anschließend werden die ungebundenen Fragmente in einem Waschschritt entfernt (Abbildung 10). Die Alternative ist die *in solution*-Methode. Hierbei werden die cDNA Sequenzen auf *Beads* gespottet und zu der Probe gegeben. Danach können die entsprechenden Fragmente gebunden und anschließend extrahiert werden (Abbildung 10). Der Vorteil der *in solution*-Methode gegenüber der *solid*-Methode, ist die geringere Menge an genomischer DNA, welche zur Extraktion des Exoms benötigt wird. Die Extraktion mittels der *in solution*-Methode konnte mit etwa 1 – 3 µg an Ausgangsmaterial durchgeführt werden. In der Zwischenzeit können bereits 500 ng für die *in solution*-Methode ausreichen. Im Gegensatz dazu benötigt die Extraktion mittels der *solid*-Methode etwa 3 – 10 µg an Ausgangsmaterial. Dieser Vorteil der *in solution*-Methode ist für die Sequenzierung von Tumoren essentiell, da nicht immer genügend DNA als Ausgangsmaterial vorhanden ist.

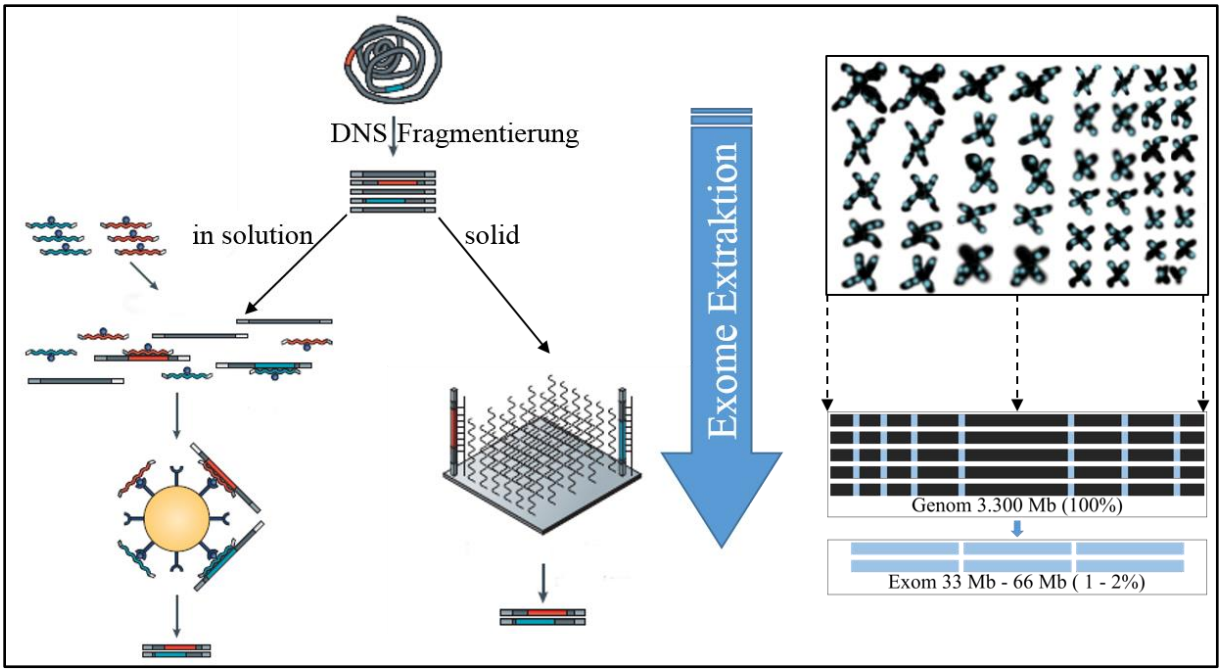


Abbildung 10: Schematische Darstellung der Extraktionsmethoden *solid* und *in solution*, veränderte Abbildung nach Metzker et al. 2010 [58]
Die fragmentierten DNA-Stücke werden bei der *in solution*-Methode (auf der linken Seite) mit Hilfe löslicher, komplementären Sequenzen hybridisiert und anschließend an einen Bead gebunden. Nachdem die nicht gebundene DNA entfernt wurde, können die spezifischen DNA Fragmente vom Bead gelöst werden und zur Generierung der Sequenzierungsbibliothek genutzt werden. Bei der *solid*-Methode (auf der rechten Seite) sind die komplementären Sequenzen auf einem Array fixiert. Die fragmentierte DNA wird über das Array gegeben und die spezifischen Sequenzen an die Templates gebunden. Ähnlich wie bei der *in solution*-Methode können anschließend die genomischen Fragmente vom Array gelöst werden und zur Generierung der Sequenzierungsbibliothek genutzt werden.

Ein großes Problem bei der Erstellung der Sequenzierungsbibliotheken für die gezielte Resequenzierung ist die einheitliche Abdeckung der Regionen. Hierzu können Störfaktoren wie eine schlechte Qualität der DNA des Ausgangsmaterials, strukturelle Veränderungen oder ein hoher sowie niedriger GC-Gehalt in den selektierten Regionen zu einer ungleichmäßigen Abdeckung führen. Deshalb ist eine hohe durchschnittliche Abdeckung bei der Planung von Resequenzierungen wichtig, um eine effiziente Analyse durchführen zu können.

3.6 Analyse von Illumina NGS-Daten

Mit dem Beginn der Digitalisierung der Ergebnisse von biologischen Experimenten, wie beispielsweise der Entschlüsselung von DNA-Sequenzen, stieg auch das Bedürfnis die Daten zunächst zu speichern und anschließend zu analysieren. Die ersten verfügbaren Programme konnten mit dem Staden Programmpaket verbreitet und genutzt werden. Dies leitete sogleich die Ära der Bioinformatik ein [80]. Nachfolgend entwickelte Softwaretools wie beispielsweise *FASTA* und *BLAST* werden bis heute genutzt und wurden kontinuierlich optimiert. Derzeit steht den Bioinformatikern eine sehr große Anzahl von Programmen und auch Datenbanken zur Beantwortung sehr unterschiedlicher Fragestellungen zur Verfügung. Sammlungen solcher

Programme können auf unterschiedlichen Servern in Form von Webclients, wie bei ENSEMBL, UCSC oder auch EBI frei genutzt werden [53; 81; 82]. Auch Daten von bereits publizierten Projekten können beispielsweise per Web-Applikation bei NCBI eingesehen werden [83]. Zudem existieren viele kleine Datenbanken mit detaillierten Informationen zu bestimmten Teilaspekten, wie z.B. Pfam mit einer Sammlung zusammengehörenden Proteinfamilien, sRNAdb mit RNA-Familien in den gram-positiven Bakterien, dbSNP und 1000Genomes mit Informationen zu Variationen und Polymorphismen einzelner Basen oder COSMIC mit Assoziationen von SNVs zu bestimmten Krebsentitäten [8; 9; 12; 84; 85]. Die Informationsflut der spezifischen Datenbanken allein erreicht eine unüberschaubare Menge, jedoch existieren weitaus mehr Softwareprogramme, um die meist durch biologische Experimente erstellten Daten und auch in den Datenbanken gespeicherten Daten zu analysieren. Die Anzahl an Programmen zur Aufbereitung, Analyse und Qualitätsprüfung von NGS-Daten ist schwierig zu bestimmen, da momentan noch neue Applikationen entstehen und weitere Softwaretools dazu entwickelt werden. Pabinger und Kollegen veröffentlichten 2013 ein Review mit einer Auflistung von etwa 205 verfügbaren Softwaretools, wobei nicht sämtliche Programme aufgelistet sind [86]. Zu den einzelnen Bereiche der Analyse von NGS-Daten wie die Identifizierung der Basen, Identifizierung der Mutationen und Generierung des Alingments sind je nach Applikation bestimmte Algorithmen besser geeignet als andere [87].

Ein weiterer wichtiger Aspekt für die Analyse von NGS-Daten ist die vorhandene Ressourcenstruktur hinsichtlich Hardware und auch im Einzelnen von Software. Zwar werden mit der Sequenzierungsmaschine für gewöhnlich auch ein Server und Softwareprogramme zur Bearbeitung der Datenflut mitgeliefert. Jedoch ist die Hardware nur zur temporären Zwischenspeicherung nutzbar. Für die generieten Bilder und Daten einer pair-end Sequenzierung mit dem Illumina *Genome Analyzer* IIx(GA IIx) benötigt die Speicherung dieser Daten bis zu circa 4 *Terabyte* (TB). Der zur Verfügung stehende Speicherplatz für die Analysen auf dem mitgelieferten Server beträgt insgesamt 10 TB. Die Verarbeitung der Bilddaten zu Signalintensitätsdaten impliziert eine Datenkompression und reduziert somit den Bedarf an Speicherplatz des erwähnten pair-end Laufs auf circa 600 *Gigabyte* (GB) (Abbildung 13). Zunächst stellte die Problematik der Bearbeitung riesiger Datenmengen eine Herausforderung an die Bioinformatik. Durch Neuentwicklung und Optimierung bereits vorhandener Algorithmen deren Parallelisierung und dem Clustern von Computern konnten Pipelines erstellt werden, die solch eine Menge an Daten problemlos bearbeiten können. Viele dieser Programme benötigen für die schnellere Bearbeitung, einen Computer mit großem Arbeitsspeicher, oft reichen 128 GB wie beispielsweise bei der kommerziellen Software von

NextGENe nicht aus. Durch die stetige Entwicklung der NGS-Technologien werden die Ausgabemengen an generierten Daten exponentiell steigen, wobei die Third Generation Technologien diesen Wachstum nicht mindern werden. Der nächste limitierende Faktor in naher Zukunft wird gewiss die effiziente Speicherung und Verfügbarkeit der Resultate darstellen. Dies zu gewährleisten bedeutet alle Informationen zu sammeln und gleichzeitig abrufbar zu speichern, um beispielsweise komparative Analysen in einer akzeptablen Zeitspanne durchzuführen.

Im Folgenden werden die einzelnen Schritte der Analyse von NGS-Daten, wie das Identifizieren von Basen, *Alignment* der Sequenzen, Identifizierung der Mutationen, Annotation und Filterung der Mutationen und deren funktionelle Vorhersage, im Detail erläutert. Dazu ist der Bezug zur gezielten Resequenzierung persistent, im Besonderen zu WES-Daten, welche mittels der Illumina-Technologie sequenziert wurden. Hierzu konnten Softwareprogramme vorselektiert werden, unter Berücksichtigung von wissenschaftlichen Veröffentlichungen und in Foren veröffentlichte Erfahrungen von Bioinformatikern.

3.6.1 Identifizierung der Basen

Die Identifizierung der Basen ist von der Nutzung der Sequenzierungsmethode, sowie der Sequenzierungsmaschine abhängig. Der Sequenzierer GA IIx von Illumina nutzt zur Sequenzierung eine Durchflusszelle, welche 8 Bahnen (*Lanes*) enthält die wiederum jeweils in 120 Kacheln (*Tiles*) aufgeteilt sind (Abbildung 13). Von diesen *Tiles* werden pro Zyklus jeweils 4 Bilder mit unterschiedlichen Filtern erstellt, welche als 16-Bit TIFF-Dateien mit einer Auflösung von 2048 x 1940 Pixel gespeichert werden [88]. Allein die Datenmenge der Bilder benötigt somit bis zu etwa 28,4 GB Speicherplatz pro Zyklus (1).

$$\sim 28{,}4GB = \frac{8 \times 120 \times 4 \times (16Bit \times (2048 \times 1940Pixel))}{(8 \times 1024 \times 1024 \times 1024)} \qquad (1)$$

Ein Sequenzierungsablauf mit 36 Basen pro Sequenz benötigt somit etwa 1 TB Speicherplatz, die konsistente Speicherung dieser Datenmenge ist durchaus realisierbar. Jedoch stieg die Länge der prozessierten Sequenzen von anfangs 30 Basen mit der rasanten Entwicklung der Technologie auf bis zu 300 Basen an. Um Speicherplatz zu sparen, werden nur noch vereinzelt Bilder unterschiedlicher Zyklen mit Speicherschonenderen JPEG-Dateien gespeichert, um mögliche Effekte wie beispielsweise Luftblasen (Abbildung 11A) oder Partikel (Abbildung 11B) bei der Sequenzierung detektieren zu können. Sind mehrere *Tiles* mit materiellen Störfaktoren übersät, reduziert dies die Ausgabemenge und auch die Qualität der benachbarten Cluster. Bei zu hohem Datenverlust oder zu geringer Qualität der Daten muss eine erneute

Sequenzierung in Betracht gezogen werden. Sind jedoch nur einzelne Kacheln betroffen, ist der Verlust meist, im Vergleich zur Ausgabemenge, irrelevant.

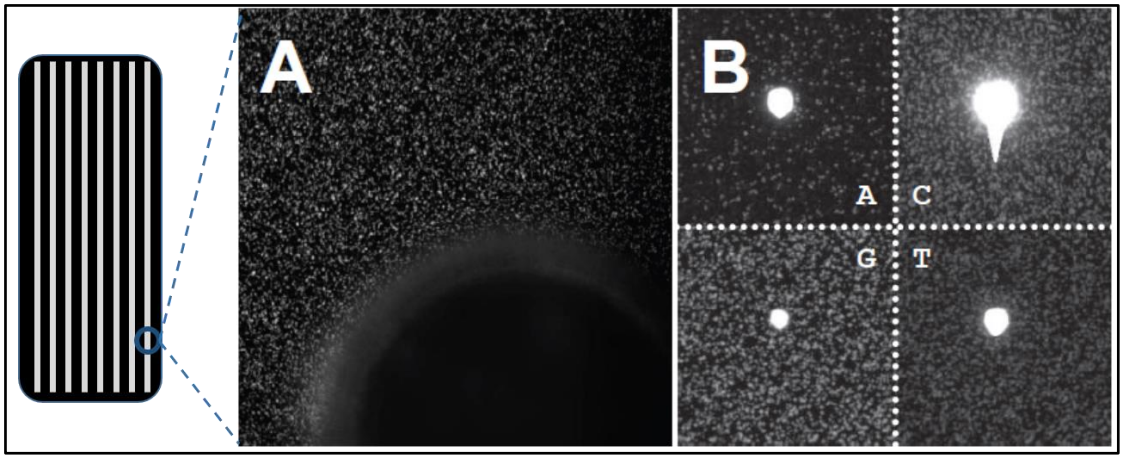


Abbildung 11: Mechanische Störeffekte während der Sequenzierung, Bild verändert nach Kircher M. et al. 2011 [89]
(A) Im unteren Abschnitt ist eindeutig eine Luftblase zu erkennen. Alle darunterliegenden Cluster können nicht detektiert werden. (B) Für jede eingebaute Base wird ein Bild erstellt, anders als bei der Luftblase kann ein Partikel hier in der Mitte der Bilder zu sehen unterschiedliche Einflüsse auf die Detektion der Cluster haben.

Während der Sequenzierung entstehen gewisse Effekte, wie beispielsweise Nebensignaleffekte (*Crosstalk*), Desynchronisation der Cluster (*Phasing*) und Intensitätsverlust der Cluster (*Fading*), welche die Identifizierung der Basen erschweren. So ist ein Hauptstörfaktor der SBS-Methode die Desynchronisation der Cluster. Die auf der Durchflusszelle generierten Cluster bestehen im Durchschnitt aus etwa 1000 Templates. An diesen Templates werden während der Sequenzierung die korrespondierenden dNTPs mit Terminator und Fluorophor gebunden (Abbildung 9). Jedoch kann ein Anteil der Terminatoren sich beim Abwaschen nicht lösen oder es ist kein eingebauter Terminator vorhanden, der die Sequenzierung unterbrechen kann. Somit entstehen desynchonisierte Templates, die dementsprechend eine Base mehr oder weniger eingebaut haben und das Signal der restlichen Templates nicht mehr verstärken. Ist der Anteil am Anfang der Sequenzierung noch sehr gering, summiert sich der Verlust an Templates exzessiv. Je länger die Reads werden desto eher kann das Signal im Verlauf der Sequenzierung nicht mehr eindeutig zugeordnet werden (Abbildung 12 A).

Zudem werden die vier unterschiedlichen Fluorophore bei der Sequenzierung mit zwei Lasern angeregt, dass bedeutet jeweils zwei Fluorophore werden mit demselben Laser bearbeitet. Dies kann zu einer Überlagerung der Emissionsspektren führen. Dabei werden unterschiedliche Fluorophore von benachbarten Clustern gleichzeitig angeregt und die erzeugten Signale somit verzerrt. Der Nebensignaleffekt ist jeweils bei den Basenpaaren A und C oder G und T zu beobachten (Abbildung 12 B).

Die sich wiederholenden Bearbeitungsschritte während der Sequenzierung belasten die DNA-

Cluster insoweit das ein Anteil an Templates ausfällt. Dies führt, neben der Desynchronisation der Cluster, zu einem weiteren Verlust der Signalstärke. Da sich auch hier der Effekt pro Zyklus verstärkt ist ein weiterer Einfluss auf die Readlänge zu erkennen. Die schwindende Signalstärke pro Zyklus ist ein limitierender Faktor für die Readlänge bei der SBS-Methode von Illumina. Jedoch kann dieser Effekt durch die Verwendung neuer Chemikalien während der Sequenzierung verringert werden. Zugleich ermöglicht diese Reduzierung des Effektes die Generierung längerer Reads (Abbildung 12 C).

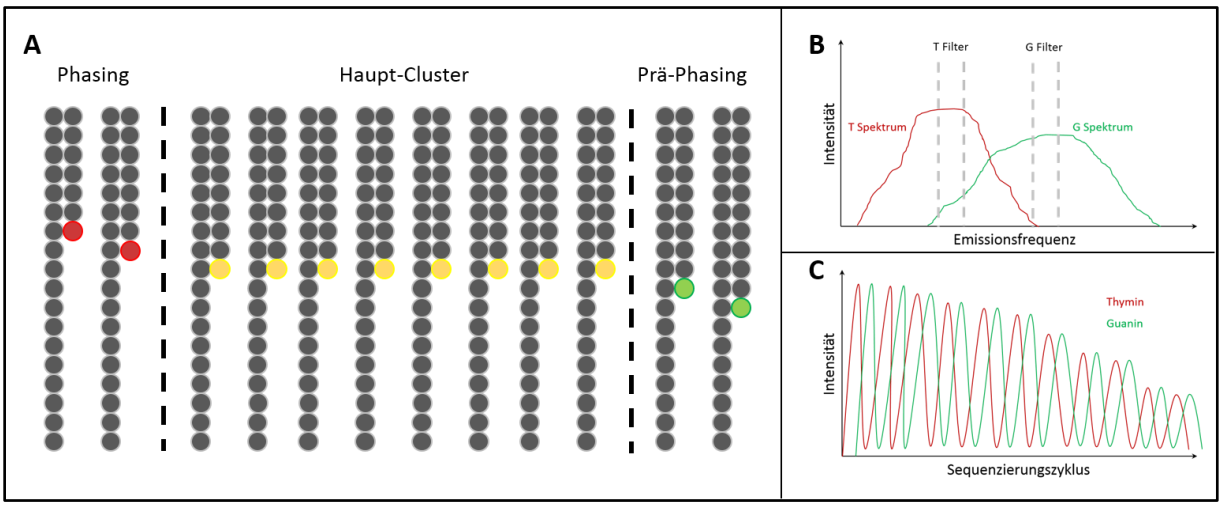


Abbildung 12: Technische und chemische Störfaktoren während der Sequenzierung [88]
(A) Die etwa 1000 Fragmente eines Clusters sind zu Beginn der Sequenzierung pro Zyklus synchron. Der als *Phasing* bezeichnete Störfaktor beschreibt Fragmente im Cluster, die nach gewissen Zyklen nicht mehr synchron sind. Hierzu können die Fragmente weniger Basen enthalten (*Phasing*) oder schon mehrere Basen enthalten (Prä-*Phasing*). (B) Beim Störfaktor *Crosstalk* können Signale benachbarter Cluster sich überlagern und sich gegenseitig verstärken. (C) Durch die gesamte Bearbeitung der genomische DNA während der Sequenzierung fallen Fragmente im Cluster aus, wodurch die Signalstärke des Clusters schwächer wird (*Fading*).

3.6.2 Bewertung der Basen

Wie bereits erwähnt werden die prozessierten Signale der verfügbaren NGS-Technologien mit unterschiedlichen Methoden identifiziert. Um die Performance der jeweiligen Plattformen zu vergleichen ist es wichtig, dass die Basen mit einem äquivalenten System bewertet werden. Dazu konnte sich die Bewertungsskala des Softwaretools *phred* letztendlich in den Analyseprogrammen etablieren. Ewing und Green entwickelten 1998 ein Programm um die Auswertung der Sanger-Sequenzierung automatisch bewerten zu können[90; 91]. Der entwickelte *Basecaller* nutzt einen Phred-Score um die Genauigkeit einer identifizierten Base basierend auf der bedingte Fehlerwahrscheinlichkeit zu bewerten. Dieses einheitliche Qualitätskriterium für die Basen erleichtert zugleich die Downstream Analysen, da die Komplexität des *Alignments* und der Identifizierung der Mutationen auf das Wesentliche beschränkt werden.

Die Qualität der identifizierten Base wird durch Plattformspezifische und vorhergesagte

Fehlerwahrscheinlichkeiten ermittelt. Mit Hilfe der von Ewing und Green erstellten Formel können die Werte zu den entsprechenden Phred-Scores umgewandelt werden (2)[91].

$$q = -10 \times \log_{10} p \qquad (2)$$

Die firmeninternen Richtwerte zur Berechnung der Wahrscheinlichkeiten basieren auf technischen und chemischen Gegebenheiten, sowie auf die Nutzung unterschiedlicher Kits bei der Sequenzierung. Somit ist auch die Beeinflussung der Qualitätswerte durch diese Gegebenheiten impliziert und zum Teil mitverantwortlich für die unterschiedlichen verwendeten *American Standard Code for Information Interchange* (ASCII) Kodierungsschemata (Tabelle 4). Zur Berechnung der Fehlerwahrscheinlichkeiten können die Phred-Scores folgendermaßen umgerechnet werden (3).

$$P = 10^{\frac{-q}{10}} \qquad (3)$$

Die erwartete Genauigkeit der identifizierten Basen liegt im Phred-Score-Bereich von 0 bis 40. Es können Werte über 40 erreicht werden, jedoch ist der verwendete Bereich von 0 - 40 für die Differenzierung von qualitativ hochwertigen und nicht nutzbaren NGS-Daten ausreichend (Tabelle 3).

Tabelle 3: Bewertungsskala des Phred-Score

Phred-Score (q)	Genauigkeit (1 – P * 100)	Fehlerwahrscheinlichkeit (p)
0	0 %	1 Fehler in 1 Base
10	90 %	1 Fehler in 10 Basen
20	99 %	1 Fehler in 100 Basen
30	99,9 %	1 Fehler in 1000 Basen
40	99,99 %	1 Fehler in 10000 Basen
...	...	...

Zu unterschiedlichen Zeitpunkten der Solexa und Illumina Ära wurden spezifische FASTQ-Formate generiert. Speziell Solexa nutzte anfänglich Qualitätswerte, welche auch eine Änderung der im FASTQ-format verwendete ASCII-Kodierung implizierten. Die ersten Illumina-Daten verwendeten ebenfalls einen Phred-Score als Qualitätskriterium, aber die Änderungen der ASCII-Kodierung wurden aus Kompatibilitätsgründen fast vollständig von Solexa übernommen. Dies setzte sich auch in der Illumina 1.5+ Version der FASTQ-Dateien fort, erst bei der Illumina 1.8+ Version änderte sich die Kodierung stärker (Tabelle 4).

Tabelle 4: ASCII-Kodierung
In FASTQ-Dateien verwendete Kodierungen des Phred-Scores. Von Sanger über Solexa bis zu den unterschiedlichen Illumina Kodierungen 1.3, 1.5 und 1.8. Die ASCII-Zeichen können je nach Kodierung mit dieser Tabelle zu dem jeweiligen Phred-Score umgewandelt werden.

ASCII	Wert	Solexa	Illumina 1.3+	Illumina 1.5+	Illumina 1.8+	Sanger
!	33				0	0
“	34				1	1
#	35				2	2
$	36				3	3
%	37				4	4
&	38				5	5
‘	39				6	6
(	40				7	7
)	41				8	8
*	42				9	9
+	43				10	10
,	44				11	11
-	45				12	12
.	46				13	13
/	47				14	14
0	48				15	15
1	49				16	16
2	50				17	17
3	51				18	18
4	52				19	19
5	53				20	20
6	54				21	21
7	55				22	22
8	56				23	23
9	57				24	24
:	58				25	25
;	59	-5			26	26
<	60	-4			27	27
=	61	-3			28	28
>	62	-2			29	29
?	63	-1			30	30
@	64	0	0		31	31
A	65	1	1		32	32
B	66	2	2	2**	33	33
C	67	3	3	3	34	34
D	68	4	4	4	35	35
E	69	5	5	5	36	36
F	70	6	6	6	37	37
G	71	7	7	7	38	38
H	72	8	8	8	39	39
I	73	9	9	9	40	40
J	74	10	10	10	41	41
K	75	11	11	11	42	42
L	76	12	12	12	43	43
M	77	13	13	13	44	44
N	78	14	14	14	45	45
O	79	15	15	15	46	46
P	80	16	16	16	47	47
Q	81	17	17	17	48	48
R	82	18	18	18	49	49
S	83	19	19	19	50	50
T	84	20	20	20	51	51
U	85	21	21	21	52	52
V	86	22	22	22	53	53
W	87	23	23	23	54	54
X	88	24	24	24	55	55
Y	89	25	25	25	56	56
Z	90	26	26	26	57	57
[	91	27	27	27	58	58
\	92	28	28	28	59	59
]	93	29	29	29	60	60
^	94	30	30	30	61	61
_	95	31	31	31	62	62
`	96	32	32	32	63	63
a	97	33	33	33	64	64
b	98	34	34	34	65	65
c	99	35	35	35	66	66
d	100	36	36	36	67	67
e	101	37	37	37	68	68
f	102	38	38	38	69	69
g	103	39	39	39	70	70
h	104	40	40	40	71	71
i	105	41	41	41	72	72
j	106	42	42	42	73	73
k	107	43	43	43	74	74
l	108	44	44	44	75	75
m	109	45	45	45	76	76
n	110	46	46	46	77	77
o	111	47	47	47	78	78
p	112	48	48	48	79	79
q	113	49	49	49	80	80
r	114	50	50	50	81	81
s	115	51	51	51	82	82
t	116	52	52	52	83	83
u	117	53	53	53	84	84
v	118	54	54	54	85	85
w	119	55	55	55	86	86
x	120	56	56	56	87	87
y	121	57	57	57	88	88
z	122	58	58	58	89	89
{	123	59	59	59	90	90
|	124	60	60	60	91	91
}	125	61	61	61	92	92
~	126	62	62	62	93	93

Um die Intensitäten aus den Bildern zu extrahieren stellt Illumina die *Firecrest* Bildverarbeitungssoftware zur Verfügung. Der Sequenzierer erstellt während des Prozesses die

Bilddateien, diese werden automatisch auf dem Computer transferiert. *Firecrest* ermittelt die jeweiligen Signalintensitäten der identifizierten Cluster pro Bild. Danach werden die temporär erstellten TIFF-Bilddateien gelöscht und die Intensitätsdateien der entsprechenden Position auf der Durchflusszelle und dem Sequenzierungszyklus zugeordnet. Für die Identifizierung der Basen werden die Intensitätsdateien mit der integrierten Illumina Pipeline *Real-Time-Analysis* (*RTA*) verarbeitet. Dabei werden Korrekturen hinsichtlich der Störfaktoren wie beispielsweise Nebensignaleffekte, Desynchronisation und Intensitätsverlust der Cluster durchgeführt, bevor die jeweilige Base mit der dazu gehörigen Qualitätsangabe bestimmt werden kann. Alternativ kann das Basecalling auch mit dem Illumina *Offline-Basecaller* (*OLB*) und den Intensitätsdateien bearbeitet oder erneut durchgeführt werden.

Als Ergebnis liefert *RTA* oder *OLB*, nach der Identifizierung der Basen, eine FASTQ-Datei [92]. Der Inhalt dieser Datei ist nach bestimmter Syntax angeordnet. Wobei die vier Zeilen pro Eintrag, den jeweiligen Namen der identifizierten Sequenzen in der ersten Zeile enthalten. Ebenso ist die Identifizierungsnummer des Sequenzierers und der entsprechenden Position auf der Durchflusszelle in der ersten Zeile notiert. Die zweite Zeile enthält die Basenabfolge der identifizierten Sequenz, diese wird als Read bezeichnet. Weitere optionale Eigenschaften, sowie die erneute Wiedergabe der ersten Zeile können in der dritten Zeile gespeichert werden. Die entsprechenden Qualitäten der jeweiligen identifizierten Basen sind in der vierten Zeile aufgelistet (Abbildung 13).

Die Identifizierung der Basen, auch als *Basecalling* bezeichnet, ist der erste Analyseschritt und beinhaltet die Aufbereitung der Daten. Dabei werden die Speicher intensiven Bilder in lesbare FASTQ-Dateien umgewandelt und zeitgleich die Speichermenge reduziert (Abbildung 13). Die resultierenden FASTQ-Dateien werden grundsätzlich als Ergebnisdateien der Sequenzierung betrachtet. Firmen die Sequenzierungen anbieten und durchführen, wie beispielsweise GATC oder CeGaT prüfen zunächst die Daten, übermitteln aber anschließend nur die FASTQ-Dateien als Ergebnis. Jedoch können somit keine weiteren Softwaretools mit alternativen Basecalling-Algorithmen oder auch Neuerungen des Illumina Softwaretools *OLB* genutzt werden.

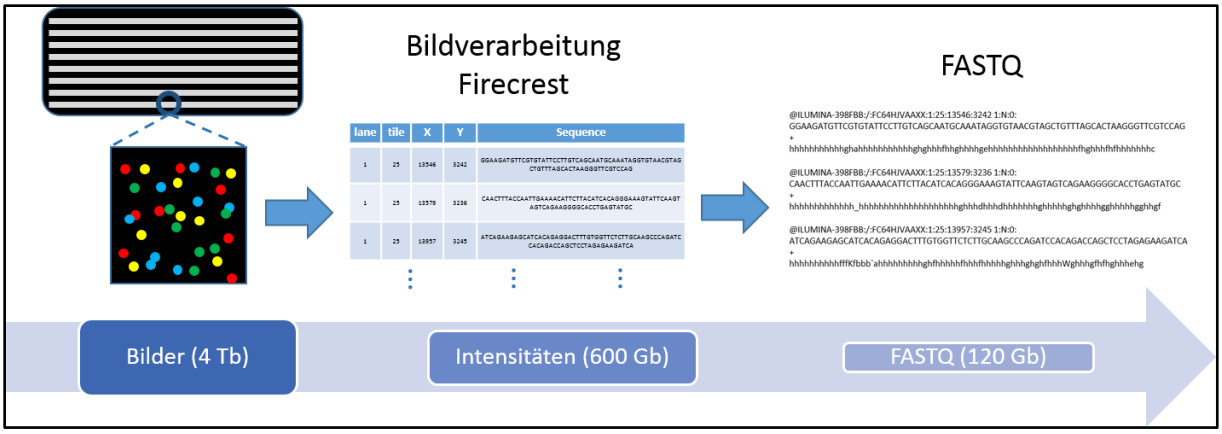


Abbildung 13: Datenkompression bei der Identifizierung der Basen
Nach der Erstellung der Bilder werden die einzelnen Basen der entsprechenden Durchflusslinie, Kachel, Cluster und Position im Bild zugeordnet. Dabei werden die Intensitäten der Cluster und auch die berechneten Wahrscheinlichkeiten in zusätzlich Dateien abgespeichert. Im nächsten Schritt werden die Sequenzen mit den berechneten Qualitäten in FASTQ-Dateien gespeichert. Zugleich entsteht bei dieser Verarbeitung eine Datenkompression, da die Informationen auf das wesentliche gebrochen und in bestimmter Syntax abgespeichert werden.

3.6.3 Zuordnung der Reads im Referenzgenom

Ein wesentlicher Analyseschritt bei der Resequenzierung ist das *Alignment*. Für den auch unter Mapping bekannten Schritt werden die generierten FASTQ-Dateien und das entsprechende Referenzgenom benötigt. Wie schon beschrieben existieren mehrere Versionen zu dem menschlichen Genom. Die während dieser Arbeit erstellten *Alignments* basieren zum größten Teil auf der Version GRCh37/hg19. Zudem wurde auch ein Teil der *Alignments* mit dem Referenzgenom NCBI36/hg18 erstellt. Da sich die Versionen in den meisten Positionen unterscheiden, ist eine konsistente Verwendung der zu Grunde liegenden Version des Referenzgenoms in allen Analysenschritten essentiell. Ausschlaggebend hierfür ist das verwendete Kit zur Extraktion des Exoms, welches bei der Vorbereitung der Sequenzierungsbibliothek zur WES genutzt wird. Dieses Kit ist für eine bestimmte Genomversion optimiert und basiert auf selektierten Regionen im Genom. Diese Regionen werden in BED-Dateien gelistet und mit Hilfe der Start- und Stopp-Position beschrieben. Es existieren Softwaretools wie *LiftOver*, NCBI *Genome Remapping Service* (http://www.ncbi.nlm.nih.gov/genome/tools/remap) und *CrossMap* um bestimmte Resultate aus dem *Alignment* oder der Identifizierung der Mutationen in verschiedene Genomversionen umzuwandeln [93; 94]. Jedoch können hierdurch Informationsverluste, durch die integrierten Erweiterungen und Reduzierungen bestimmter Regionen in den unterschiedlichen Genomversionen, entstehen.

Während des *Alignments* werden die prozessierten Sequenzen in der dazugehörigen Genomversion gesucht. Mit den Annahmen, dass das humane Genom eine Größe von 3,3

Milliarden Basen besitzt und es vier mögliche Basenpaare wie A/T, C/G, G/C und T/A gibt, kann eine benötigte Länge der Reads von etwa 16 Basen zur eindeutigen Identifizierung berechnet werden (4).

$$4^x = 3300000000 \rightarrow x \approx 15{,}81 \qquad (4)$$

Jedoch erschweren Faktoren wie die unterschiedliche Größen der DNA-Fragmente, die Länge der verwendeten Reads sowie die Länge als auch die Anzahl von repetitiven Sequenzabschnitten im Genom eine eindeutige Zuordnung. Hierzu haben Clark und Kollegen nicht nur den Einfluss von unterschiedlichen Extraktionskits zur Effizienz untersucht, sondern auch unterschiedliche Mengen an Reads zur daraus resultierenden Abdeckung gegenübergestellt [77]. Als Ergebnis konnte festgestellt werden, dass circa 80 Millionen Reads mit einer 10 fachen Abdeckung der Basen zwischen 89,6% und 96,8% der gewünschten Regionen abdecken [77]. Bei der Assemblierung werden die einzelnen Reads, die sich überlagern, als *Contigs* zusammengefasst. Die dabei verwendete Read-Länge bestimmt somit, wie lang die resultierenden *Contigs* werden und daher auch die Größe des assemblierten Genoms. Bei der Resequenzierung ist zu beobachten, dass durch längere Reads mehr Basenpaare identifiziert werden können, so ist eine prozentuale Abdeckung des gesamten Genoms zu 90% realisierbar, wenn 43 Nukleotid lange Reads verwendet werden. Bei einer Erhöhung der Read-Länge auf etwa 68 Nukleotide können bis zu 95% des gesamten menschlichen Genoms abgedeckt werden [95-97]. Zusätzlich zu den 8% der nicht assemblierten Basenpaare des hg19 Referenzgenoms, entstehen noch etwa 1% nicht abdeckbarer Regionen, durch die Verwendung von etwa 1000 Nukleotid langen Reads. Des Weiteren können dabei etwa 1% der generierten Reads nicht zu dem Genom zugeordnet werden [98].

Ein weiterer Faktor der Einfluss auf die Zuordnung der Reads nimmt, und somit auch auf die Abdeckung der sequenzierten Region, ist die Fehlertoleranz. Bei der Zuordnung der Reads muss eine gewisse Fehlertoleranz erlaubt sein, da sowohl der technische als auch der biologische Aspekt zu anderen Basen in den prozessierten Sequenzen führen kann, statt die erwarteten Basen in dem Referenzgenom zu bestätigen [99]. Die dabei zugelassene Menge an Veränderungen ist ausschlaggebend für die Geschwindigkeit der *Alignment*-Softwaretools, weshalb dieser Parameter so niedrig wie möglich gehalten werden muss.

Zur Generierung von *Alignments* wird generell das Softwaretool *BLAST* genutzt, wobei komparativen Analysen von Sequenzen unterschiedlicher Organismen durchgeführt werden. Dabei prüft *BLAST* die Sequenzen auf Gleichheit und die biologische Entwicklung der Sequenzabfolgen steht bei der Analyse im Vordergrund. Die Entwicklungswahrscheinlichkeit

der möglichen Veränderungen in den Sequenzen wird unter Berücksichtigung von Matrizen wie BLOSUM und PAM für die Berechnung der nahen oder weit entfernten Verwandtschaft genutzt. Jedoch können die erstellten FASTQ-Dateien der NGS-Technologien viele Millionen von Sequenzen beinhalten, die zu einer Referenz bestehend aus etwa 3,3 Milliarden Basen zugeordnet werden müssen. Die Laufzeit von *BLAST* würde mehrere Tage für solch eine Datei benötigen und die Reads nur auf die biologische Verwandtschaft zu der Sequenz im Referenzgenom untersuchen. Das entscheidende Ausschlusskriterium ist die Geschwindigkeit, weshalb *BLAST* nicht als *Alignment-Tool* für NGS-Daten genutzt werden kann.

Für die NGS-Technologien wurden neue *Alignment-Tools* entwickelt, die eine große Menge an Daten schnell verarbeiten können und bei kurzen Read-Längen eine hohe Spezifität aufweisen. Dabei bedienen sich die ersten entwickelten Algorithmen der Indexierung der Daten mittels Hashtabellen, ähnlich wie bei dem Softwareprogramm *BLAST*. Eines der ersten für diese Aufgabe verwendeten Programme ist *MAQ* [100]. Dieses Softwaretool nutzt eine auf Hash-Tabellen basierende Indexierung für die Speicherung und Verarbeitung der Reads und des Referenzgenoms. Zur Generierung der *Alignments* des 1000Genomes-Projekts wurde anfänglich *MAQ* als Standardtool verwendet. Durch die rasante Entwicklung der Technologien und der immer größeren Ausgabemengen erreichten auch die auf Hash-Tabellen basierenden Algorithmen ihr Limit um vertretbare Laufzeiten einzuhalten. Effizientere Softwareprogramme mit der Ferragina-Manzini (FM) Indexierung und integriertem *Burrows-Wheeler-Transformation* (BWT)-Algorithmus wurden daher entwickelt [101; 102]. Somit konnten die steigenden Mengen an prozessierten Reads in einer adäquaten Zeit dem Referenzgenom zugeordnet werden. Die generelle Bearbeitung beinhaltet dabei die Zerlegung des Referenzgenoms in kleinere Teilsequenzen und eine entsprechende Sortierung. Mit nur wenigen Nukleotiden des Reads können die Teilsequenzen vorselektiert und die zu durchsuchenden Regionen im Referenzgenom verringert werden. Daraus folgt eine Reduzierung der Laufzeit für die *Alignments*, ohne einen hohen Verlust der Spezifität und der bereits geringen Sensitivität zu verursachen [102]. Das Softwareprogramm *Burrows-Wheeler Alignment* (*BWA*) ist vor allem für die Bearbeitung der FASTQ-Dateien von gezielten Resequenzierung geeignet [103]. Jedoch existieren nahezu gleichwertige Softwareprogramme oder für spezifische Applikationen auch bessere Alternativen [87]. In vielen komparativen Analysen und auch in Foren konnte *BWA* stets überzeugen [87; 103-107]. Dies und die stetige Entwicklung sowie Implementierung von neuen Parametern und dem Support der *BWA* Entwickler führte dazu, dass sich *BWA* in den Pipelines großer Sequenzierungsprojekten wie 1000Genomes, TCGA und ICGC durchsetzen konnte.

Das Ergebnis der Zuordnung von Reads zu dem Referenzgenom wird standardmäßig in einer *Sequence Alignment/Map* (SAM)-Datei gespeichert. Dazu werden die gesammelten Informationen in entsprechender Reihenfolge mit jeweils einem Tabulator voneinander getrennt in einer Zeile aufgelistet [108] (Abbildung 14). Um bei dem *Alignment* auch Speicherplatz zu sparen, können die SAM-Dateien in *binary Alignment/Map* (BAM)-Dateien konvertiert und abgespeichert werden. Diese benötigen zwar weniger Speicherplatz, jedoch ist der Inhalt in Binärcode umgewandelt und nicht mehr für den Menschen lesbar.

```
1 ILLUMINA-398FBB:7:FC64HJVAAXX:1:2:6605:4188|163|chr11|73104930|
  60|76M|=|73105016|162|AAATAACTCCATACCTCTTGTCCGCAAGATCTGTTTTATTT
  CCTACTAGCATGATGATAACATCACTTCCTCTTTC|FIHIIIIDIIIIHIIIIIIGGIIIIII
  IHIIDFIIIIIIIIIIIIIIGFIIIGHIIIGGGIIHIGEIIHIHIHIIG|RG:Z:05_09_
  2011_MM1176|XT:A:U|NM:i:1|SM:i:37|AM:i:37|X0:i:1|X1:i:0|XM:i:1
  XO:i:0|XG:i:0|MD:Z:22A53
2 ILLUMINA-398FBB:7:FC64HJVAAXX:1:2:6856:4196|99|chr9|30679747|23|
  76M|=|30679877|206|TACTAAAGACAGCTATTCAAGCAGGGATTACCCAAGTTCTCGTGA
  TACTAGAGCTTATGCACCACCACCACGAGAT|GGGGG==D+FBBDE:BED@G?GGDBE:@GDGE
  GE4EGGDEG<GG?@GDGGG4=DGGGGGEGGGGGDEBGGGG@G:D|RG:Z:05_09_2011_MM
  1176|XT:A:U|NM:i:2|SM:i:23|AM:i:0|X0:i:1|X1:i:1|XM:i:2|XO:i:0|
  XG:i:0|MD:Z:24A16A34|XA:Z:chr1,-89221378,76M,3;
```

Abbildung 14: Dateiausschnitt aus einer SAM-Datei
Die einzelnen Elemente dieser zwei Zeilen aus einer SAM-Datei wurden mit einem Strich (*pipe*) anstatt mit einem Tabulator getrennt, um die Visualisierung in Textform übersichtlich zu gestalten. Die jeweiligen getrennten Eigenschaften sind in den *SAM Format Specification* des Softwareprogramms *samtools* erläutert [108].

Beim *Alignment* wird zu jedem Read die beste Position im Referenzgenom zugeordnet, dabei entstehen mehrere mögliche Zuordnungen der einzelnen Reads. Hierbei können diese Zuordnungen hinsichtlich der Bewertungen gleiche Resultate aufweisen, da aber jeder Read nur ein Fragment repräsentiert, darf dieser auch nur einer Region zugeordnet werden. Viele Programme nutzen dabei eine Zufallsselektion, um aus den mehrfach zugeordneten Reads auszuwählen. Wird der Parameter der möglichen Veränderungen erhöht, so können auch mehrere Reads die an unterschiedlichen Regionen zugeordnet werden beim *Alignment* entstehen. Neben der längeren Verarbeitung entsteht zudem ein durch das Zufallsprinzip induzierter Bias.

Durch die Amplifizierung während der Generierung der Sequenzbibliothek entstehen gleiche DNA-Fragmente, die zu Duplikaten der Reads führen. Dabei kann nicht unterschieden werden ob das Fragment zweimal entstanden ist oder die PCR dafür verantwortlich ist. Die Duplikate sowie die mehrfach zugeordneten Reads als auch qualitativ minderwertige Reads werden aus den *Alignment*-Daten entfernt. Somit verringert sich die Menge der Daten und auch die Analysezeit der folgenden Identifizierung der Mutationen.

3.6.4 Identifizierung der Mutationen

Nachdem die Nukleotide der prozessierten Sequenzen identifiziert, die daraus resultierenden Reads in den FASTQ-Dateien mit den dazugehörigen Qualitäten gespeichert und anschließend zu dem Referenzgenom aligniert wurden, können die resultierenden SAM-Dateien zur Identifizierung der Mutationen genutzt werden. Zu Beginn der Entwicklung von neuen Softwaretools zur Identifizierung von Mutationen gewann man einen Eindruck, wie sich einzelne Faktoren, wie beispielsweise Indels, Sequenzierungszyklen und Basenkompositionen auf die Anzahl falsch positiver Kandidaten auswirken. Hierfür existieren in dem *Genome Analysis Toolkit*(*GATK)*-Paket mehrere Tools um die alignierten Daten aufzubereiten. Dieser *Best Practices-Workflow* impliziert gewisse Schritte, wie beispielsweise das *Realignment* von Indels. Nicht identifizierte Indels können im *Alignment* Mutationen generieren die nicht vorhanden sind. Somit werden in den *Alignment*-Ergebnissen mögliche oder bereits aus Datenbanken bekannte Indels untersucht und zur Verbesserung des *Alignment* nachträglich aufgenommen [109]. Des Weiteren werden die Basenqualitäten angeglichen, unter Berücksichtigung des entsprechenden Sequenzierungszyklus der Base und der benachbarten Nukleotide. Die *GATK*-Entwickler beziehen sich dabei auf die bekannten Einflüsse der Illumina SBS-Methode, welche schlechtere Basenqualitäten am Ende der Reads oder bei Dinukleotiden wie beispielsweise A/C prozessiert oder sogar höhere Qualitäten bei T/G-Dinukleotiden berechnet [109]. Diese Nachbearbeitung der Daten zeigte eine hohe Reduzierung von falsch positiv Raten bei der Identifizierung von Mutationen [109]. Viele Mutationsidentifizierungstools empfehlen auch den *Best Practices-Workflow* von *GATK* vor der Nutzung des eigens entwickelten Tools anzuwenden.

Der wesentliche Schritt bei der Identifizierung von Mutationen ist die Differenzierung der reellen und der durch den Bias induzierten Veränderungen. Hierzu dienen Filterkriterien wie beispielsweise die Qualitätswerte der Basen und des *Alignments*, sowie die Abdeckung und vor allem das Verhältnis der Reads mit zu Reads ohne der Veränderung. Je nach Qualität der NGS-Daten sollten die Filtergrenzen der Kriterien angepasst werden, um die bestmöglichste Ausbeute aus den Daten zu generieren. Dazu sollte beachtet werden, dass beispielsweise die erforderte Mindestabdeckung für die Identifizierung von homozygoten oder heterozygoten Mutationen jeweils 3 Reads oder 13 Reads beträgt [110; 111]. Zur Identifizierung von kleineren Insertionen oder Deletionen werden Änderungen in den Readlängen untersucht. Werden Lücken in den alignierten Reads gefunden, sind diese als Deletionen im sequenziertem Genom anzusehen. Insertionen können mit Hilfe von Reads die zusätzliche Basen enthalten und nicht dem Referenzgenom entsprechen, identifiziert werden. Hierbei spielt erneut die erreichte

Abdeckung der Basen eine essentielle Rolle, da der technische Bias bei der Identifizierung der Basen zu falschen Nukleotiden in den Reads führen kann. Sowohl diese Fehler, als auch falsch alignierte Reads können zur Identifizierung von Mutationen führen, die nicht im sequenzierten Genom existieren. Die Wahrscheinlichkeit, dass eine identifizierte Mutation existiert, erhöht sich durch hohe Qualitäten der Basen in den zugeordneten Reads und auch durch eine höhere Anzahl dieser Reads.

Zusätzlich zu den technischen Faktoren beeinflussen biologische Faktoren wie die unterschiedliche Zellpopulation, strukturelle Veränderungen, Keimbahn-Punktmutationen und auch der Gehalt an Tumorzellen die Identifizierung von somatischen Punktmutationen. Diese biologischen Faktoren können zu gleichen Effekten führen wie die technischen Faktoren, deshalb ist eine Differenzierung in den NGS-Daten nicht eindeutig. Zum Beispiel ist die erwartete Häufigkeit der Reads von 100% bei homozygoten oder von 50% bei heterozygoten Mutationen sehr selten zu beobachten. Die genomische DNA in den NGS-Daten repräsentiert eine Zellpopulation aus unterschiedlichen Zellen, diese Zellen besitzen nicht immer die Veränderung und somit können die generierten Reads auch nicht diese Veränderung beinhalten. Diesbezüglich führt außerdem der Anteil an Tumor- und Normalzellen in einer sequenzierten Probe zu niedrigeren Prozentzahlen. Positiv oder negativ können diese Häufigkeiten auch durch strukturelle Veränderungen beeinflusst werden. Durch die Evolution entstandene Mutationen werden als relevante Mutationen identifiziert, obwohl sie keinen Beitrag zur Kanzerogenese leisten. Bis zu 216 neue individuelle Mutationen können pro Generation in einem Individuum erwartet werden. Von Individuum zu Individuum können die biologischen Faktoren in geringen Maßen variieren, während bei Zelllinien weitaus mehr Punktmutationen und strukturelle Veränderungen durch genomische Instabilität beobachtet werden.

Die identifizierten Mutationen und die dazu gehörigen berechneten Qualitäten einzelner Basen und des *Alignments* werden in VCF-Dateien gespeichert. Somit können die Mutationen mit allen bisherigen ausgewerteten Eigenschaften zusammengefasst und beschrieben werden (Abbildung 15).

#CHR	POS	ID	R	A	QUAL	INFO	Patient_2_PBMC	Patient_2_Tumor
chr1	757734	rs4951929	C	T	214.52	AC=14;AF=0.875;…	./.	1/1:0,1:1:3:34,3,0
chr1	866319	rs9988021	G	A	15038.96	AC=60;AF=1.00;…	1/1:0,12:13:15:185,15,0	1/1:1,4:5:9:108,9,0
chr1	1171377	rs7536250	A	G	55.98	AC=4;AF=1.00;…	./.	1/1:0,1:1:3:38,3,0
chr1	1171417	rs6603782	C	T	54.98	AC=4;AF=1.00;…	./.	1/1:0,1:1:3:39,3,0
chr1	1880351	.	T	C	60.75	AC=4;AF=0.222;…	./.	1/1:0,1:1:3:33,3,0
chr1	2585910	.	G	A	78.51	AC=8;AF=0.444;…	./.	1/1:0,1:1:3:33,3,0
chr1	2585919	.	A	C	71.19	AC=7;AF=0.350;…	./.	1/1:0,1:1:3:33,3,0
chr1	2615481	.	G	A	74.40	AC=6;AF=0.231;…	1/1:0,1:1:3:31,3,0	0/1:1,1:2:29:29,0,31
chr1	15845025	.	T	A	151.47	AC=8;AF=0.190;…	0/0:2,0:2:6:0,6,70	0/1:3,1:4:21:21,0,75
chr1	20827143	.	C	G	54.06	AC=2;AF=0.036;…	0/0:12,0:12:33:0,33,392	0/1:9,3:12:72:72,0,181
chr1	55251959	.	A	C	1133.26	AC=10;AF=0.179;…	0/0:8,5:13:21:0,21,242	0/1:13,3:17:13:13,0,376
chr1	55352598	.	G	A	243.13	AC=2;AF=0.034;…	0/1:6,6:12:99:148,0,144	0/1:6,6:12:99:144,0,117
chr1	66070904	.	C	T	5753.01	AC=2;AF=0.032;…	0/1:96,103:200:99:3080,0,2828	0/1:111,88:199:99:2723,0,3371

Abbildung 15: Dateiausschnitt aus einer VCF-Datei
Die einzelnen Zeilen repräsentieren eine identifizierte Mutation in dem analysierten Datensatz. Sowohl das Chromosom, als auch die Position der jeweiligen Mutation sind aufgelistet. Zudem werden die Identifikationsnummern von bekannten SNPs aus der dbSNP Datenbank hinzugefügt. Die Base des Referenzgenoms und die identifizierte veränderte Base des Individuums sind gegenübergestellt. Weitere Eigenschaften wie Anzahl der Reads, Allel-Frequenzen und eine Qualitätsangabe zu den einzelnen Mutationen sind in der jeweiligen Zeile ersichtlich. In dieser Darstellung wurden einige Informationen entfernt, um Visualisierung übersichtlicher zu gestalten. Die letzten zwei Spalten repräsentieren jeweils die sequenzierte Normalprobe und die korrespondierende Tumorprobe des Patienten_2 aus dem ersten MM-WES-Datensatzes. Farblich hervorgehoben (Grün) sind zwei Mutationen die sowohl in der Normalprobe als auch in der Tumorprobe identifiziert wurden und keine Identifikationsnummer besitzen. Diese Kandidaten sind mit hoher Wahrscheinlichkeit nicht Tumorrelevante Mutationen oder noch nicht identifizierte SNPs.

3.6.5 Filterung und Annotation

Ein wichtiger Aspekt nach der Sequenzierung von entarteten Zellen und der anschließenden Identifizierung der Mutationen ist die Einteilung der natürlich erworbenen und mit der Krankheit assoziierten SNVs. Diese Unterteilung ermöglicht die Differenzierung der Mutationen zwischen Kandidaten die zur Entartung führen können oder jenen die mit hoher Wahrscheinlichkeit bereits im Genom etabliert sind. Hierzu ist die Verwendung einer korrespondierenden Normalprobe, die idealerweise zeitgleich sequenziert wird am geeignetsten. Ist keine Normalprobe vorhanden, können Datenbanken wie dbSNP und 1000Genomes zur Identifizierung von Keimbahnmutationen verwendet werden. Alternativ können weitere Ansätze zur Filterung von Keimbahnmutationen genutzt werden, wie beispielsweise identifizierte Mutationen aus Kohorten, welche nur Normalproben beinhalten. Hierfür können die Daten aus dem *Exome Aggregation Consortium*(ExAC) ExAC65000 verwendet werden. Suzuki und Kollegen untersuchten hierzu wie effizient die Subtraktion von Mutationen aus Kohorten mit Normalproben sein kann. Als Referenz wurden Normalproben genutzt, die nicht die korrespondierenden Normalproben der Tumorproben repräsentierten, sondern aus einem Pool von unterschiedlichen Individuen bestehen. Das Ergebnis mit bis zu 64 % der Keimbahnpunktmutationen konnten in ihrem Datensatz identifiziert und entfernt werden [112]. Jedoch sind spezifische Mutationen der Normalprobe, die natürlich erworben und noch

nicht in einer Population etabliert sind, mit dieser Methode unauffindbar. Dies impliziert einen gewissen Anteil an falsch positiven Mutationen in der SNV-Liste, welche als relevante Kandidaten der Tumorprobe definiert werden.

Die Schwierigkeit bei diesem Analyseschritt ist die Bewertung und Filterung von identifizierten Kandidaten. Hierfür enthalten die VCF-Dateien neben den Qualitäten der Basen und dem *Alignment*, noch viele weitere Eigenschaften die zur Beschreibung der Kandidaten dienen. Zurzeit existiert noch keine Standard-Lösung um die Identifizierung und anschließende Bewertung der Kandidaten so durchzuführen, dass alle Mutationen eindeutig identifiziert und die relevanten Mutationen eindeutig beschrieben werden können. Es existieren aber viele Ansätze und die Identifizierungsraten bestätigen die Effizienz dieser Programme, wie beispielsweise *GATK*, *SOAPsnp*, *SVNer*, *SAMtools*, *VarScan*, *MuTect* und *SomaticSniper* [108; 109; 113-117]. Ebenso zeigen Gegenüberstellungen von generierten Ergebnisse, dass die Effizienz der Identifizierung vom Ansatz oder dem Programm der Analyse abhängen kann [118] (Abbildung 16).

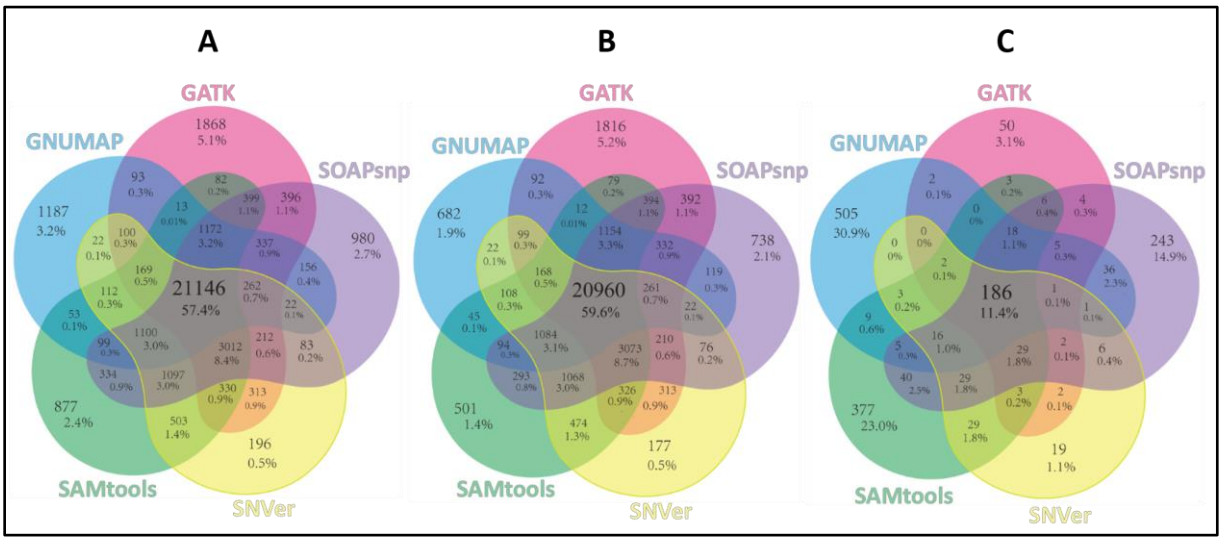


Abbildung 16: Variantcaller Vergleich aus [118]
(A) Die Softwaretools *GATK*, *SOAPsnp*, *SVNer*, *SAMtools* und *GNUMAP* konnten in einem Vergleich bis zu 58% an gleichen Mutationen in einem Datensatz identifizieren [118]. (B) Wobei die SNPs bis zu 60% von allen Softwaretools identifiziert werden. (C) Die Gegenüberstellung der Softwaretools bezüglich neuer Mutationen zeigt, dass nur etwa 12% in allen Tools identifiziert werden.

Als Alternative kann die Vereinigungsmenge der Ergebnisse von mehreren Programmen, sowohl beim *Alignment* als auch bei der Identifizierung der Basen, genutzt werden [119]. Hierbei sollen möglichst alle Mutationen identifizieren werden. Jedoch sind ebenso die Programm spezifischen, falsch positiven Kandidaten in der Vereinigungsmenge und werden mit aufgenommen. Zudem erschwert sich die einheitliche Filterung dieser Daten, da die unterschiedlichen Programme zumeist unterschiedliche Eigenschaften der identifizierten

Mutationen angeben oder sogar unterschiedliche Bewertungen durchführen.

Nach der Identifizierung der Mutationen können Fragestellungen bezüglich der generierten Daten analysiert werden. Für die Filterung ist es wichtig zuvor die Fragestellung zu spezifizieren, um die Filterkriterien dementsprechend festzulegen und priorisieren zu können. Somit ist es für unser Hauptaugenmerk, das Screening von somatischen Mutationen im Exom, essentiell exonische Regionen zu identifizieren. Die angrenzenden Regionen der einzelnen Exons werden bei der WES mit sequenziert, da zuvor bei der Extraktion des Exoms die anliegenden Regionen in Abhängigkeit der Fragmentgröße in die Sequenzierungsbibliothek integriert werden. Somit können beispielsweise bei dem Extraktion-Kit *SeqCap EZ Human Exome Library v2.0* von Nimblegen mit etwa 244000 Regionen und einer durchgängigen Größe von 200 Nukleotiden der Fragmente, eine Abdeckung von 97,6 Mbp außerhalb des Exoms erwartet werden. Dies übersteigt die eigentliche Größe der selektierten Regionen des Exoms, die bei diesem Kit etwa 44 Mbp betragen. Mit der Annotation der Daten können die Gene und somit auch Regionen des Exom identifiziert, selektiert und zugleich die Datenmenge reduziert werden. Die Informationen zur Annotation werden aus Datenbanken wie beispielsweise RefSeq und CCDS genutzt. Mit Klassifizierungen wie beispielsweise Introns oder *Untranslated-Regions* (UTRs) können die Kandidaten differenziert werden. Mutationen, die zwischen den Genen lokalisiert sind, können keinem Gen zugeordnet werden und sind durch die Selektion der exonischen Regionen automatisch gefiltert.

Mit Hilfe von Annotationssoftwaretools wie *ANNOVAR*, *SNPeff* und *SeattleSeq* können die VCF-Dateien annotiert und die SNVs klassifiziert werden. Während der Annotation können zudem komparative Analysen bezüglich bereits bekannter SNPs und der identifizierten SNVs durchgeführt werden. Somit können die SNPs, da sie mit hoher Wahrscheinlichkeit keinen Einfluss auf die Entartung haben, aus der Liste entfernt werden. Diese Unterteilung beruht auf dem Abgleich von Datenbanken wie beispielsweise dbSNP. Jedoch können die spezifischen Mutationen der einzelnen Individuen mit dieser Methode nicht identifiziert werden. Weitere komparative Analysen mit Datenbanken wie COSMIC können bereits bekannte somatische Mutationen in dem Datensatz hervorheben, die zur Kanzerogenese assoziiert werden. Jedoch ist die effektivste Methode der Differenzierung von SNPs und SNVs unter Verwendung der korrespondierenden Normalprobe zu bewerkstelligen. Dabei ist noch unklar, ob die NGS-Ergebnisse der Tumor- und Normalprobe simultan oder eher einzeln hinsichtlich der Identifizierung von Mutationen analysiert werden sollten. Bevor die vermeintlich uninteressanten SNPs anschließend entfernt werden können.

Ein weiterer Schritt während der Filterung von relevanten somatischen Mutationen ist die

Differenzierung von Kandidaten, welche außerhalb oder in den kodierenden Regionen lokalisiert sind. Für die Mutationen in den kodierenden Regionen können die vermutlichen Auswirkungen auf die translatierte Proteinsequenz bestimmt werden. Diese Auswirkungen erlauben eine Klassifizierung in *synonymous* oder *non-synonymous*. Dabei implizieren die *synonymous* Mutationen keinen Aminosäureaustausch und werden somit meistens keine direkten Auswirkungen auf die Funktion des betroffenen Gens bewirken. Im Gegensatz dazu können die *non-synonymous* Mutationen nochmals in *missense* oder *nonsense* unterteilt werden. Eine *missense* Mutation hat einen Aminosäureaustausch in der zu translatierenden Proteinsequenz zur Folge. Diese kann die Funktion des Gens mindern oder verhindern, in seltenen Fällen kann eine *missense* Mutation aber auch die Funktion des Gens verstärken. Eine *nonsense* Mutation hat auch einen Aminosäureaustausch zur Folge, jedoch impliziert diese einen vorzeitigen Abbruch der Translation des Proteins. Somit wird die Translation des gesamten Proteins verhindert. Die Unvollständigkeit der Proteinsequenz führt meist zum Abbau des Produkts, womit die Funktion des Proteins gänzlich unterbunden wird.
Die Unterteilung von SNPs und SNVs, aber auch die Klassifizierung der Kandidaten vor der funktionellen Vorhersage unterstützt die Priorisierung relevanter Mutationen, welche in der VCF-Datei gelistet sind.

3.6.6 Funktionelle Vorhersagen

Nach der Generierung der VCF-Dateien und deren Annotation und Filterung können bis zu mehrere hundert Mutationen pro untersuchtem Individuum als somatische Mutation identifiziert werden. Um die Effizienz der Sequenzierung und Filterung zu berechnen, werden zufallsmäßig Kandidaten aus der Liste ausgewählt und mit Hilfe der Sanger-Sequenzierung untersucht. Die Erfolgsrate der zu validierenden Kandidaten repräsentiert eine Validierungsrate der gesamten Liste. Eine in der Literatur akzeptierte und oft veröffentlichte Validierungsrate liegt bei etwa 90%.
Häufig wiederkehrende Genmutationen in Onkogenen könnten zur Entwicklung neuer Inhibitoren dienen. Diese Inhibitoren werden in Studien zunächst auf klinische und therapeutische Relevanz getestet, wie beispielsweise der BRAF-Inhibitor Vermurafenib. Andrulis und Kollegen untersuchten die positive Auswirkung des BRAF Inhibitors Vermurafenib bei Patienten mit MM, welche die Mutation p.V600E im BRAF aufzeigten [120]. Einschätzungen des Funktionsverlustes oder Auswirkungen der Mutationen können während der Annotation mit SeattleSeq erstellt werden (http://snp.gs.washington.edu/SeattleSeqAnnotation138/) [121]. Hierbei generieren integrierte

Programme wie *GERP* oder *PhastCons* eine Bewertung der betroffenen Region in Abhängigkeit der Konservierung jener Region und der erworbenen Mutation [121]. Dabei suggeriert eine eher konservative Region, eine Unveränderlichkeit während der Evolution. Ist eine Mutation in dieser Region lokalisiert, erhöht sich die Wahrscheinlichkeit dass diese Veränderung eine ungewollte Funktionsänderung des Gens auslöst. *PolyPhen* berechnet auf Grundlage des Aminosäureaustausches die möglichen Folgen. Wird zum Beispiel eine eher hydrophobe Aminosäure mit einer hydrophilen ausgetauscht, löst dies mit hoher Wahrscheinlichkeit eine Konformationsänderung des Proteins aus. Ähnliche Auswirkungen werden erwartet, wenn Aminosäuren aus den unterschiedlichen Gruppen wie unpolar, polar, basisch, sauer, negativ geladen oder positiv geladen untereinander ausgetauscht werden.
Zu den meisten der betroffenen Gene bestehen detaillierte Einträge in Datenbanken, wie RefSeq oder CCDS. Die dort abgelegten Informationen beruhen meist auf Ergebnissen von funktionellen Experimenten. Hierzu können schlüssige Auswirkungen der Entartung auf die Funktion des Proteins gezogen werden. In einigen Fällen existieren jedoch eher nur Beschreibungen aus Datenbanken zu den einzelnen Genen. Diese ermöglichen begrenzte Schlussfolgerungen zum Einfluss der Mutation, jedoch ist die Grundlage der Proteinfunktion nicht im Detail untersucht. Gene, die bis dato eher unbekannt oder unbeschrieben sind, liefern zum Zeitpunkt der Identifikation noch keine Informationen zum Einfluss der Entartung. Weitere funktionelle Analysen dieser Kandidaten oder der betroffenen Gene könnten jedoch neue Erkenntnisse liefern. Um aus dem identifizierten Genpool geeignete Mutationskandidaten zu priorisieren, können zusätzlich die zuvor genannten *in-silico* Methoden zur funktionellen Vorhersage genutzt werden.

4 Ziel der Arbeit

Um die immer größer werdende Menge an NGS-Daten zu analysieren, müssen effektivere Analyse-Methoden entwickelt werden. Dabei ist das Ziel eine einheitliche Identifizierung, Filterung und Bearbeitung der Daten und zudem eine qualitativ hochwertige Ausgabe zu prozessieren. Neuere oder verbesserte Algorithmen und Methoden, die eine schnellere und bessere Dateninterpretation ermöglichen, werden in den nächsten Jahren die Softwareentwicklung in der Genomsequenzierung dominieren. Um die NGS-Daten zu analysieren, sollte im Rahmen dieser Arbeit eine effiziente und benutzerfreundliche Pipeline zur Detektion von Tumor-assoziierten SNVs entwickelt werden. Im Vordergrund sollte hierbei die Identifizierung von Tumor-assoziierten SNVs stehen, wobei eine hohe Datenqualität unter Berücksichtigung der Störfaktoren vorausgesetzt wurde. Die einzelnen Schritte zur Verarbeitung der NGS-Daten wie beispielsweise die Identifizierung der Basen, das *Alignment*, die Identifizierung der Mutationen, sowie der Annotation und Filterung der Mutationen, sollten jeweils mit diversen Softwareprogrammen getestet und bewertet werden.

Des Weiteren war ein Ziel dieser Arbeit eine umfangreiche biologische Interpretation der Daten zu unterstützen. Um dies zu erreichen sollten Informationen aus bestehenden Datenbanken und den sequenzierten NGS-Daten extrahiert werden. Hierzu wurden auch neue Skripte generiert, welche die Aufnahme externer Mutationsdaten und die Analysen wiederkehrender, häufig vorkommender und fehlender Mutationen ermöglichten.

5 Material und Methoden

Bei der *whole exome* und *Amplikon* Sequenzierung entsteht eine große Menge an Daten. Das Management sowie die Generierung, Speicherung, Umwandlung und Analyse dieser Daten ist die Aufgabe der Bioinformatik. Für die entsprechenden Bearbeitungsschritte existiert eine Vielzahl an Softwaretools, die bereits in der Einleitung näher erläutert wurden. Um den Prozess der gesamten Analyse zu optimieren und die Konsistenz der Daten zu gewährleisten, bedarf es der Entwicklung einer Analyse-Pipeline. Somit können die benötigten Analyseschritte zur Bearbeitung der NGS-Daten, sowie die implementierten Umwandlungen der Daten von FASTQ- über SAM/BAM-Dateien bis hin zu der anschließenden Identifizierung von Mutationen und deren Ausgabe in VCF-Dateien in einer Software-Applikation sequentiell durchgeführt werden.

5.1 *Next Generation Sequencing*

Für die Generierung der in dieser Arbeit untersuchten NGS-Daten wurden unterschiedliche Sequenzierer, wie beispielsweise der Roche-Junior von Roche, der SOLiD von Applied Biosystems sowie der GA Ⅱx, der HiSeq und MiSeq von Illumina verwendet. Der Roche-Junior und der MiSeq wurden für die Sequenzierung der Amplikons verwendet und der SOLiD sowie der GAIIx und HiSeq für die WES. Da die mit dem SOLiD generierten NGS-Daten für die Identifizierung von Mutationen nicht zufriedenstellend waren, wurden die Sequenzierung mit der Illumina-Plattform wiederholt und für die WES generell die Sequenzierungsplattformen von Illumina genutzt. Hinzu kommt, dass sowohl der SOLiD als auch der Roche Junior aufgrund seiner erhöhten Kosten und technischen Limits keinen Absatz mehr fanden und der Vertrieb dieser Geräte eingestellt wurde, sowie der Service hierzu.

5.1.1 Exom-Extraktion-Kit

Die Extraktion des Exom der jeweiligen DNA-Proben wurde mittels der bereits beschriebenen WES-Applikation durchgeführt. Hierzu wurden diverse *whole-Exome*-Extraktions-Kits von Nimblegen, Illumina und Agilent genutzt (Tabelle 5). Die verwendeten Längen der Reads variierten bei den Illumina-Plattformen von 36 Bp bis zu 251 Bp, bei SOLiD waren es 50 Bp und bei Roche etwa 500 Bp.

Tabelle 5: Exom-Extraktion-Kits

Firma	Version	Gesamte Region	Version des Referenzgenoms
Nimblegen	*SeqCap 2.1M Human Exome Array*	36,5 MBp	GRCh36 und hg18
Nimblegen	*SeqCap EZ Human Exome Library v.2.0*	44,1 MBp	GRCh37 und hg19
Illumina	*Nextera Rapid Capture Exome*	37 MBp	GRCh37 und hg19
Agilent	*SureSelect Human All Exon V5*	50 MBp	GRCh37 und hg19
Agilent	*SureSelect Human All Exon V5+UTRs*	75 MBp	GRCh37 und hg19

Bei jeder sequenzierten Probe wurde für die WES eine durchschnittliche Abdeckung von mindestens 100 Sequenzen pro Base angestrebt, um den Anteil an vorhandener nicht-Tumor-Sequenzen zu kompensieren. Dieser Anteil beträgt bei den MM-Proben etwa 5 – 10% und bei den FL-Proben bis circa 40%. Zu den Amplikon-Sequenzierungen wurden höhere Abdeckungen angestrebt, zwischen circa 500 Reads pro Base mit dem 454 GS junior und circa 1500 Reads pro Base mit dem Illumina MiSeq. Diese hohen Abdeckungen ermöglichten auch die Berücksichtigung von Nebenklonen in der Analyse der NGS-Daten [122].

5.1.2 NGS-Daten zum Multiple Myelom

Zum MM existieren vier NGS-Datensätze. Zwei dieser Datensätze beinhalteten WES-Daten und zwei Datensätze enthielten Amplikon Sequenzierungsdaten. Der erste bereits veröffentlichte WES-Datensatz zu den MM Proben bestand aus 6 Zelllinien und 5 primäre Proben mit der jeweiligen korrespondierenden Normalprobe [29]. Für diese WES-Applikation wurden Reads mit der Länge von 36 Bp und 76 Bp, sowie die Exom-Extraktions-Kits von Nimblegen verwendet (Tabelle 6). Der gesamte Datensatz wurde mit dem GA IIx und der *paired end*-Applikation von Illumina generiert.

Tabelle 6: Erster MM-Datensatz

Probe	Typ	Länge der Reads	Exom-Extraktion-Kit
Patient 1	Primäre Probe	76 Bp	*SeqCap EZ Human Exome Library v.2.0*
Patient 2	Primäre Probe	76 Bp	*SeqCap 2.1M Human Exome Array*
Patient 3	Primäre Probe	76 Bp	*SeqCap 2.1M Human Exome Array*
Patient 4	Primäre Probe	76 Bp	*SeqCap 2.1M Human Exome Array*
Patient 5	Primäre Probe	76 Bp	*SeqCap 2.1M Human Exome Array*
MM1S	Zelllinie	36 Bp	*SeqCap 2.1M Human Exome Array*
U266	Zelllinie	36 Bp	*SeqCap 2.1M Human Exome Array*
AMO1	Zelllinie	36 Bp	*SeqCap 2.1M Human Exome Array*
OPM2	Zelllinie	36 Bp	*SeqCap 2.1M Human Exome Array*
JJN3	Zelllinie	76 Bp	*SeqCap EZ Human Exome Library v.2.0*
L363	Zelllinie	76 Bp	*SeqCap EZ Human Exome Library v.2.0*

Der zweite in dieser Arbeit untersuchte WES-MM-Datensatz umfasst 20 primäre MM Proben und die korrespondierenden Normalproben (Tabelle 7). Viele Faktoren wie der Sequenzierer (HiSeq 2500 von Illumina), die Länge der Reads (101 Bp) und auch das Exom-Extraktion-Kit (*SureSelect Human All Exon V5*) unterschieden sich wesentlich zu dem ersten WES-Daten.

Tabelle 7: Zweiter MM-Datensatz

Probe	Typ	Länge der Reads	Exom-Extraktion-Kit
P1	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P2	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P3	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P4	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P5	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P6	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P7	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P8	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P9	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P10	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P11	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P12	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P13	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P14	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P15	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P16	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P17	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P18	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P19	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*
P20	Primäre Probe	101 Bp	*SureSelect Human All Exon V5*

Des Weiteren wurden am MM zwei Amplikon Sequenzierungsstudien durchgeführt [122; 123]. Im Rahmen dieser Studien wurden die kodierenden Regionen (CDS) der Gene *EPHA2*, *ERRB3*, *IGF1R*, *NTRK1*, *NTRK2*, *DIS3*, *KRAS* und *EGFR* sequenziert. Der erste Amplikon-Datensatz bestand aus den CDS der Gene *DIS3*, *KRAS* und *EGFR* von 81 primäre Proben und 12 Zelllinien und wurde mit dem Roche 454 GS junior sequenziert. Der zweite Amplikon-Datensatz enthielt die CDS der Gene *EPHA2*, *ERBB3*, *IGF1R*, *NTRK1* und *NTRK2* von 75 primären MM-Proben und 12 MM-Zelllinien und wurde mit dem Illumina MiSeq sequenziert. Diese Datensätze wurden für komparative Analysen in dieser Arbeit verwendet.

5.1.3 NGS-Daten zum Follikuläres Lymphom

Der FL-Datensatz bestand aus 11 primäre Proben, hierzu wurden 10 als t(14;18)-negative FLs und eine Probe als t(14;18)-positives FL charakterisiert. Die Generierung der Sequenzierungsbibliothek erfolgte mit dem Exom-Extraktion-Kit *Nextera Rapid Capture Exome* von Illumina. Des Weiteren wurde der Sequenzierer GA IIx, die *paired-end*-Applikation und Reads mit einer Länge von 72 Bp verwendet (Tabelle 8).

Tabelle 8: FL-Datensatz

Probe	Typ	Länge der Reads	Exom-Extraktion-Kit
FL1269	Primäre Probe	72 Bp	*Nextera Rapid Capture Exome*
FL2162	Primäre Probe	72 Bp	*Nextera Rapid Capture Exome*
FL6128	Primäre Probe	72 Bp	*Nextera Rapid Capture Exome*
FL9782	Primäre Probe	72 Bp	*Nextera Rapid Capture Exome*
FL12333	Primäre Probe	72 Bp	*Nextera Rapid Capture Exome*
FL16865	Primäre Probe	72 Bp	*Nextera Rapid Capture Exome*
FL18053	Primäre Probe	72 Bp	*Nextera Rapid Capture Exome*
FL18701	Primäre Probe	72 Bp	*Nextera Rapid Capture Exome*
FL24332	Primäre Probe	72 Bp	*Nextera Rapid Capture Exome*
FL25221	Primäre Probe	72 Bp	*Nextera Rapid Capture Exome*
FL29758	Primäre Probe	72 Bp	*Nextera Rapid Capture Exome*

In Zusammenarbeit mit dem ICGC konnte der FL Datensatz hinsichtlich der Anzahl an Proben vergrößert werden. Die Sequenzierung als auch die Analyse der ICGC-FL Proben wurden vom DKFZ in Heidelberg organisiert und durchgeführt. Die resultierenden Mutationslisten wurden im Rahmen dieser Arbeit bearbeitet, um die Daten in die Analyse der FL t(14;18)-negativen einzubinden. Hierzu wurde der Bereich des Exom-Extraktion-Kit *Nextera Rapid Capture Exome* von Illumina als Basis verwendet. Somit wurde ein Extraktion auf der Ebene der VCF-Dateien durchgeführt und die reduzierten Dateien für die komparative Analyse genutzt.

5.1.4 NGS-Daten zum Burkitt Lymphom

Zu den BL-Proben wurden zwei WES-Datensätze aus jeweils 8 BL-Zelllinien und deren korrespondierenden Normalproben generiert. Zunächst wurden die Exome der BL-Proben mittels der SOLiD-Technologie sequenziert und ausgewertet. Hierbei wurden 50 Bp lange Reads mit der *single read*-Applikation generiert, d.h. pro Fragment wurde jeweils nur eine Seite sequenziert. Auf Grund der nicht zufriedenstellenden Ergebnisse wurden die gleichen Proben

jedoch erneut mittels der SBS-Methode unter Verwendung des HiSeq2000 und der *paired end*-Applikation sequenziert und ausgewertet (Tabelle 9).

Tabelle 9: BL-Datensatz

Probe	Typ	Länge der Reads	Exom-Extraktion-Kit
BL36	Primäre Probe	101 Bp	*SureSelect Human All Exon V5+UTRs*
BL37	Primäre Probe	101 Bp	*SureSelect Human All Exon V5+UTRs*
BL41	Primäre Probe	101 Bp	*SureSelect Human All Exon V5+UTRs*
BL64	Primäre Probe	101 Bp	*SureSelect Human All Exon V5+UTRs*
BL67	Primäre Probe	101 Bp	*SureSelect Human All Exon V5+UTRs*
BL65	Primäre Probe	101 Bp	*SureSelect Human All Exon V5+UTRs*
BL74	Primäre Probe	101 Bp	*SureSelect Human All Exon V5+UTRs*
BL18	Primäre Probe	101 Bp	*SureSelect Human All Exon V5+UTRs*

5.2 Softwareprogramme

Diverse Softwareprogramme wurden für die Verarbeitung und Analyse der NGS-Daten genutzt. Hierzu wurden komparative Analysen ausgewertet um die geeignetsten Programme für die Pipeline zu selektieren. Die Pipeline und die entwickelten Skripte wie *Genemapping*, *Heatmap* und *unconReg* wurden mit der Programmiersprache Python erstellt.

Tabelle 10: Softwareprogramme

Software	Version/Jahr	Programmtyp	Hauptfunktion
ANNOVAR	*2011 – 2014*	*Programm Paket*	*Annotation*
GenGen	*2010 - 2013*	*Programm Paket*	*Annotation*
BayesCall	*0.3*	*Programm*	*Basecalling*
Novoalign	*2.08.03*	*Programm*	*Alignment*
BEDTools	*2.1.2 – 2.25.0*	*Programm Paket*	*SAM/BAM/BED-Datei-Verarbeitung*
Picard	*1.41 – 1.139*	*Programm Paket*	*NGS-Datei-Verarbeitung und Qualität Auswertung*
samtools	*0.1.12 – 1.2*	*Programm Paket*	*NGS-Datei-Verarbeitung und Qualität Auswertung*
bowtie	*0.12.7*	*Programm Paket*	*Alignment*
BWA	*0.5.9 – 0.7.12*	*Programm Paket*	*Alignment*
MAQ	*0.7.1*	*Programm Paket*	*Alignment/Variantcalling*
eclipse	*3.6.1 – 4.3.2*	*Integrierte Entwicklungsumgebung*	*Software Entwicklung*
FastQC	*0.10.1*	*Programm*	*Qualität Auswertung*
SolexaQA	*2.2*	*Programm*	*Qualität Auswertung*
GATK	*1.0 – 3.46*	*Programm Paket*	*Mutationsidentifizierung*
RTA	*1.5.1 – 1.9*	*Pipeline*	*NGS-Datenverarbeitung und Analyse*
OLB	*1.8 – 1.9.4*	*Programm*	*Identifizierung der Basen*
VarScan	*1.0 – 2.3.2*	*Programm Paket*	*Mutationsidentifizierung*
SomaticSniper	*0.7.4.9*	*Programm Paket*	*Mutationsidentifizierung*
vcftools	*0.1.10 – 0.1.11*	*Programm Paket*	*VCF-Datei-Verarbeitung*
Echo	*1.11*	*Programm*	*Basecalling Aufwertung*
CrossMap	*0.1.6*	*Programm*	*NGS-Datei-Verarbeitung*
fastq-tools	*0.7*	*Programm Paket*	*FASTQ-Datei-Verarbeitung*
IGV	*2.3.31*	*Programm*	*Visualisierung*

5.3 Bewertung der Basenidentifizierung

Bei der Identifizierung der Basen wurden die während der SBS-Methode von Illumina produzierten Fluoreszenzsignale jedes Clusters einer entsprechenden Base zugeordnet. Anschließend wurden die Signale mit Hilfe eines plattformspezifischen Bewertungssystems ausgewertet, wobei entstandene Störfaktoren in die Berechnung des Phred-Score miteinbezogen wurden. Die Qualitäten der einzelnen sequenzierten Basen wurden mit den jeweiligen Phred-Score bewertet. Hierfür wurde eine Bewertung der sequenzierten Basen in folgende Bereiche unterteilt, von nicht verwertbaren (Phred-Score < 10), verwendbaren (Phred-

Score > 10 - 30) und sehr gut sequenzierten (Phred-Score > 30) Basen (Tabelle 11). Mittlerweile nutzt Illumina eine Klasseneinteilung bestimmter Phred-Score-Bereiche, wodurch die festgelegten Qualitätsbereiche nicht beeinflusst werden. Der Vorteil dieser Klasseneinteilung ist eine erhöhte Datenkomprimierung der FASTQ-Dateien bei einem sehr geringen Datenverlust. Der somit eingesparte Speicherplatz erleichtert die konsistente Speicherung der Daten.

Tabelle 11: Eingeteilte Bewertungsskala des Phred-Score
Der Phred-Score gibt eine schnelle Übersicht über die Qualitätswerte der identifizierten Basen. In dieser Tabelle sind die jeweiligen Werte mit der dazugehörigen prozentualen Genauigkeit und deren Bedeutung farblich hervorgehoben. So sind Phred-Scores zwischen 0 bis 20 nicht für die Donwstream Analyse geeignet. Werte ab 20 können mit einbezogen werden und Werte ab 30 repräsentieren sehr glaubwürdige Sequenzabfolgen.

Phred-Score (q)	Genauigkeit (1 – P * 100)	Fehlerwahrscheinlichkeit (p)
0	0 %	1 Fehler in 1 Base
10	90 %	1 Fehler in 10 Basen
20	99 %	1 Fehler in 100 Basen
30	99,9 %	1 Fehler in 1000 Basen
40	99,99 %	1 Fehler in 10000 Basen
...	...	...

Für die Bewertung der Basen und auch für die gesamte Analyse der NGS-Daten ist es essentiell die Kodierung der FASTQ-Dateien zu kennen. Hierfür wurde stets das Kodierungsschema in den generierten FASTQ-Dateien beachtet und vermerkt. Ergebnisse der Downstream Analysen können positiv oder negativ beeinflusst werden, wenn fälschlicherweise eine andere Kodierung genutzt wird. Dieser Effekt ist nicht immer erkennbar und könnte die Ergebnisse leicht verfälschen. Mittlerweile wurden viele *Alignment-Tools* dahingehend erweitert und können das Schema der Kodierung automatisch identifizieren. Jedoch sind die Werte der verwendeten Kodierungen von Solexa und Illumina 1.3+ – 1.5+ sehr ähnlich (Tabelle 4). Deshalb wurde stets die Kodierung der FASTQ-Dateien kontrolliert. Eine eindeutige automatische Zuordnung der verwendeten Kodierung wurde hiermit positiv bestätigt, wenn während der Zufallsbegutachtungen des Algorithmus ein Ausreißer identifiziert werden konnte. Für die Bearbeitung von FASTQ-Dateien ist stets zu beachten zu welcher Zeit und welches Format (fastq-sanger, fastq-solexa, fastq-illumina1.3+, fastq-illumina1.5+ oder fastq-illumina1.8+) zur Generierung genutzt wurde. Die gesonderte Syntax der SOLiD-Daten mit jeweils zwei Qualitätswerten zu einer sequenzierten Base, führte zu einer exakteren Identifizierung durch die zweifache Bewertung der Base. Diese Bewertung verkomplizierte jedoch unnötigerweise die Downstream Analysen. Zudem wurden nur wenige Softwaretools für die SOLiD-Daten

entwickelt.

Für die Identifizierung der Basen wurden neben den klassischen Softwaretools von Illumina, alternative *Basecaller* wie *Alta-Cyclic*, *Swift*, *Rolexa*, *Ibis* und *naiveBayesCall* entwickelt und in dieser Arbeit getestet. Ältere Ausgabeformate der Illumina Sequenzierer wie seq.txt, prb.txt, .qseq und .scarf mussten neueren Formaten wie beispielsweise .bcl weichen. Somit konnten die Softwaretools wie *Alta-Cyclic*, *Swift* und *Rolexa* nicht mehr in dem Test eingebunden werden, da diese nicht weiter entwickelt und den neuen Ausgabedateien der Illumina-Sequenzierer nicht angepasst wurden. Des Weiteren zeigte sich die Basenidentifizierung mit *Ibis* oder *naiveBayesCall* als sehr effizient und die Fehlerraten waren geringer als bei dem Illumina Softwaretool [124]. *Ibis* benötigte jedoch Trainingsdatensätze, um die Identifizierung der Basen effizient durchführen zu können. Dazu muss eine bekannte DNA-Probe oder Genom mehrfach sequenziert werden, bevor der Klassifikator der *Support Vector Machine* (SVM) spezifiziert werden kann. Zudem haben Änderungen an dem Sequenzierer oder den Chemikalien Auswirkungen auf die Bewertung der Signalintensitäten, welche erneut mit Trainingsdatensätzen in die Analyse eingebunden werden müssen.

Die verwendeten Softwaretools wie *naiveBayesCall* und *ECHO* ermöglichten eine bessere Ausbeute aus den sequenzierten Daten zu berechnen, da die zuvor beschriebenen Störfaktoren in die Berechnung des Phred-Scores berücksichtigt wurden. So war ein Vorteil dieser alternativen *Basecaller* mehr Informationen ohne Aufpreis und ohne weitere Experimente aus den NGS-Daten zu generieren. Dies führte beispielsweise bei bereits abgeschlossenen Projekten dazu, dass ein gewisser Anteil der zuvor erhaltenen Ausgabemenge erhöht und/oder die Qualitätsangaben der identifizierten Basen verbessert wurden. Jedoch bedarf die Nutzung alternativer *Basecaller* zeitgleich eine Konvertierung der Dateien oder weitere interne Berechnungen, welche sehr Zeit intensiv waren.

5.4 Bewertung des *Alignments* und der Abdeckung

Für die Resequenzierung von Genomen ist das *Alignment* ein essentieller Schritt bei der Analyse. Eine Assemblierung des gesamten Genoms ist hierbei unnötig, da die gesamte Referenzsequenz schon bekannt ist. Die generierten Sequenzen werden mit dem Referenzgenom verglichen und dabei die Positionen der Reads bestimmt.

Die Softwareprogramme *MAQ* und *BWA*, sowie die integrierten Programme der Pipelines von Illumina und NextGENe wurden für die Generierung diverser *Alignments* genutzt und für die Verwendung innerhalb der entwickelten Pipeline getestet. *MAQ* diente zunächst allgemein als Standardprogramm für das *Alignment* und wurde zuerst in der Pipeline für das *Alignment*

verwendet, sowie es auch bei dem 1000Genomes Projekt genutzt wurde. Das Softwareprogramm *BWA* verdrängte jedoch *MAQ* als Aligner in der Pipeline, bei fast gleicher Sensitivität und mit einer weitaus schnelleren Laufzeit.

Beim *Alignment* mit *BWA* wurden hauptsächlich die Standardeinstellungen genutzt, wobei bis zu 6 Basen von insgesamt 100 Nukleotiden der Reads als austauschbar erachtet werden. Das *Alignment* Programm kann dabei nicht auf die Quelle der Veränderung Rückschlüsse ziehen, sondern nur die Varianz der Reads mit einfließen lassen. Es zeigte sich wie sehr das Basecalling diese Varianz und somit den *Alignment*-Analyseschritt beeinflussen kann. Wurde durch das Basecalling der technische Bias stark reduziert, so konnte die einbezogene Anzahl an Veränderungen beim *Alignment* für die durch den biologischen Bias induzierten Veränderungen genutzt werden. Dadurch konnten mehr Reads dem Referenzgenom zugeordnet und zeitgleich die Rate der falsch zugeordneten Reads reduziert werden. Bei der Wahl des limitierenden Parameters für die Fehlertoleranz, konnte kein bestimmter Wert für das *Alignment* festgelegt werden. Wichtig waren hierbei die entstehenden Auswirkungen zu beachten, so erhöhten sich die falsch positiven Mutationen bei höheren Werten des Parameters, aber auch die Anzahl der reellen Mutationen. Im Gegensatz dazu konnten niedrigere Werte des Parameters die falsch positiven Mutationen reduzieren, aber auch die Anzahl an reellen Mutationen.

Während der Generierung des *Alignments* war stets zu beachten welche Faktoren die Qualität der NGS-Daten beeinflussen können und somit auch die folgenden Ergebnisse der Analyse. Wie in der Einleitung beschrieben sind Indels, identische Reads und Reads die mehrfach dem Referenzgenom zugeordnet werden, störende Effekte für das *Alignment*. Hierzu wurden in dieser Arbeit nach Vorgaben des *Best Practices-Workflows* von *GATK* einzelne Programme des *GATK*-Packages genutzt, um diese Effekte zu minimieren. Zudem können bei Entitäten wie dem BL auch DNA-Fragmente der assoziierten Viren bei der Generierung der Sequenzierungsbibliothek auftauchen. Diese wurden beim *Alignment* mit dem menschlichen Genom zum Referenzgenom aligniert und nicht als gesonderte Reads differenziert.

Für die Bewertung des *Alignments* und auch der zuvor durchgeführten Analysen ist die Abdeckung der sequenzierten Probe ein wichtiges Kriterium. Hierzu wurden innerhalb und außerhalb der festgelegten Regionen des Exom-Extraktions-Kits die lokalisierten Reads betrachtet. Die außerhalb liegenden Reads wurden aus den *Alignment*-Daten entfernt. Diese Reduzierung führte zur Verringerung der Analysezeit bei der Identifizierung von Mutationen. Diese Reads waren primär nicht für die Identifizierung somatischer Mutationen in den kodierenden Regionen geeignet, somit wurden keine relevanten Daten entfernt. Jedoch zeigten

Guo und Kollegen, dass diese Daten effektiv zur SNP-Detektion oder zur Genotypisierung verwendet werden können [125].

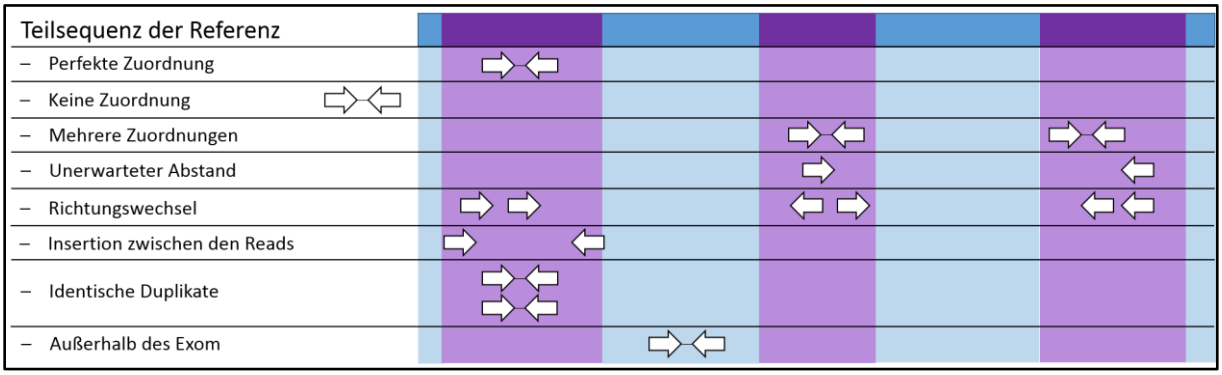


Abbildung 17: Schematische Darstellung möglicher Zuordnungen von Reads zum Referenzgenom beim *Alignment*
Die Teilsequenz des Referenzgenoms ist mit nicht kodierenden Regionen (Blau) und kodierenden Regionen (Lila) hervorgehoben. Die sequenzierten *paired-end* Reads einer WES-Applikation werden beim *Alignment* dem Referenzgenom zugeordnet und die Paare können beispielsweise perfekt, an unterschiedlichen Regionen (mehrfache Zuordnung), an gleicher Stelle (Duplikate) oder auch gar nicht zugeordnet werden. Des Weiteren können die Reads mit einem unerwarteten Abstand zueinander, mit Richtungswechseln oder außerhalb der kodierenden Regionen zugeordnet werden.

Zur Berechnung der Abdeckung können unterschiedliche Aspekte betrachtet werden. Im folgendem werden Formeln zur Berechnung der jeweiligen Abdeckung beschrieben, welche in dieser Arbeit zur Bewertung der NGS-Daten genutzt wurden.

Die durchschnittliche Abdeckung ($\emptyset Abdeckung$) ist in der Literatur häufig als Qualitätsmaß angegeben. Jedoch ist dies nur die theoretisch mögliche Abdeckung. Zur Berechnung wurde die gesamte Anzahl der Reads in den extrahierten Regionen ($Reads_{ontarget}$), sowie die Readlänge (l) und die extrahierte Region ($Target$) ausgewertet (5). Das Ergebnis repräsentierte die mögliche Häufigkeit der untersuchten, sequenzierten Basen. Dies entsprach der Anzahl wie oft eine Base in der extrahierten Region im Durchschnitt abgedeckt wurde.

$$\emptyset Abdeckung = \frac{(Reads_{ontarget}) \times (l)}{(Target)} \qquad (5)$$

Des Weiteren wurde die Abdeckung in Abhängigkeit eines bestimmten Faktors ($n\ fold$) berechnet, der zwischen den ausreichend abgedeckten ($\#Base_{covered}$) und nicht ausreichend abgedeckten Basen ($\#Base_{uncovered}$) differenziert. Dabei war entscheidend wie viele Reads die entsprechende Base repräsentieren ($Reads_{covered\ per\ Base}$). Mit Hilfe dieses Faktors konnte die Tiefe der Abdeckung und somit die Uniformität zu den jeweiligen NGS-Daten bewertet werden (6).

$$Uniformität = \begin{cases} \#Base_{uncovered} & n\,fold > Reads_{covered\ per\ Base} \\ \#Base_{covered} & n\,fold \leq Reads_{covered\ per\ Base} \end{cases} \quad (6)$$

Nachfolgend wurde eine prozentuale Abdeckung (*%Abdeckung*) der Region in Abhängigkeit des Faktors berechnet. Die Anzahl der ausreichend abgedeckten Basen ($\#Base_{covered}$) wurde dabei der Anzahl der Basen in der extrahierten Region ($\#Base_{ontarget}$) gegenübergestellt (7). Daraus konnte die prozentuale Abdeckung pro sequenziertem Individuum berechnet werden. Dieser Wert ist ein wichtiges Qualitätskriterium für die Abdeckung und der anschließenden Identifizierung der Mutationen. Hierzu ist die bereits erwähnte vertretbare minimale Grenze zur Identifizierung von homozygoten und heterozygoten Punktmutationen von jeweils 3 Reads und 13 Reads hervorzuheben [110; 111]. Die prozentuale Abdeckung repräsentiert in Abhängigkeit der limitierenden Grenzen, die als Faktor zur Berechnung der Uniformität dienen, die Gesamtmenge der extrahierten Region, welche effektiv analysiert werden.

$$\%\ Abdeckung = \frac{(\#Base_{covered},\ |\ Reads_{covered\ per\ Base} \geq n\,fold)}{(\#Base_{ontarget})} \times 100 \quad (7)$$

Untersuchungen hierzu zeigten, dass durch die Erhöhung des Faktors immer weniger Basen des sequenzierten Individuums in die Analyse der Identifizierung der Mutationen Beachtung fanden (Abbildung 18).

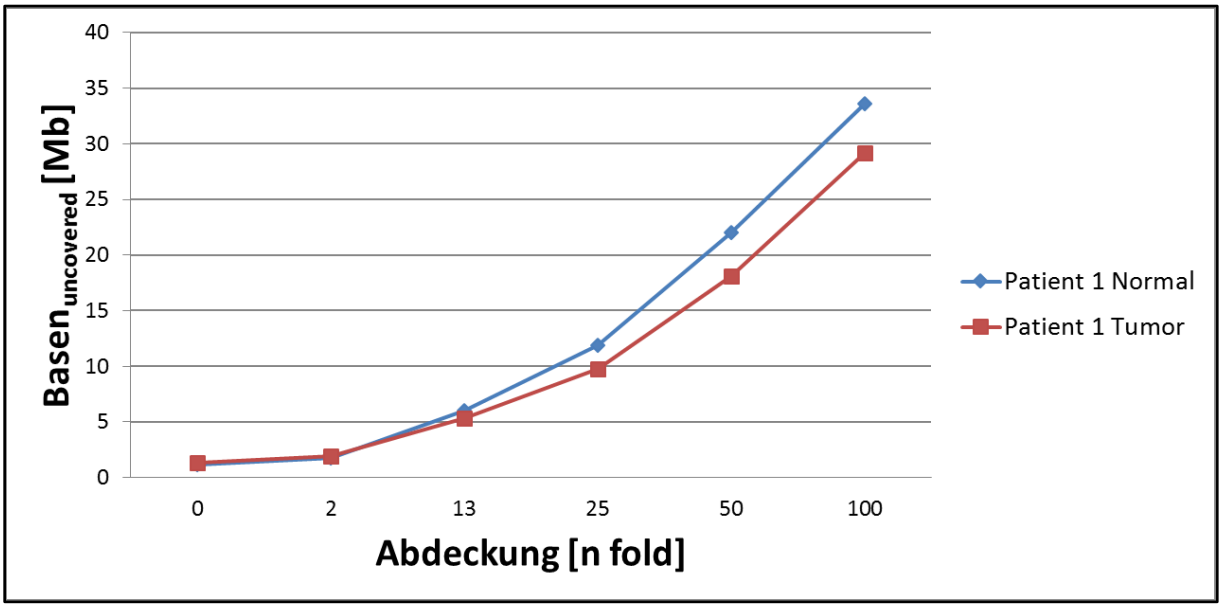


Abbildung 18: Abdeckung in Abhängigkeit der Anzahl an Reads
In diesem Plot ist zu erkennen wie viele Basen in Abhängigkeit der Anzahl an Reads abgedeckt beziehungsweise nicht abgedeckt werden. Wird die erforderte Anzahl an Reads erhöht (*n fold*), dann erhöht sich auch die Menge an Basen die dieser Anforderung nicht entsprechen.

Neben der Qualität der Basen und der zugeordneten Reads im Genom waren sowohl die Länge der Reads, als auch die Ausgabemenge der Sequenzierung ausschlaggebend für die Effizienz

des *Alignments*. Die durchschnittliche Abdeckung wurde direkt durch diese Faktoren beeinflusst, höhere Ausgabemenge und längere Reads führten zu einer höheren theoretischen Abdeckung. Die prozentuale Abdeckung betrachtete die in Wirklichkeit abgedeckten Regionen des Exoms und war vor allem von der Effizienz des Exom-Extraktion-Kits abhängig. Eine gleichmäßig oft sequenzierte Region bedeutete dass diese auch gleichmäßig oft extrahiert wurde. Der Faktor *n fold* filterte die sequenzierten Regionen entsprechend der Tiefe. Die Generierung von aussagekräftigen SAM/BAM-Dateien und auch die folgende Identifizierung der Mutationen waren von einem effizienten *Alignment* und somit von den entsprechenden durchschnittlichen und prozentualen Abdeckungen abhängig. Die Ergebnisse der Berechnungen wurden für die Bewertung der NGS-Daten genutzt.

5.5 Bewertung der Mutationsidentifizierung

Die Identifizierung der Mutationen, insbesondere der Punktmutationen erscheint im ersten Moment als ein eher einfacher Analyseschritt, da nur die Unterschiede der Reads zu dem Referenzgenom aufgelistet werden. Für den Erfolg der Mutationsidentifizierung sind die zuvor durchgeführten Analyseschritte umso wichtiger. Die Reduzierung möglicher Störfaktoren in den erläuterten Analyse-Bereichen, kann die Rate der falsch positiven Kandidaten in den Ergebnissen der Identifizierung verringern. Hierzu stellt *GATK* weitere Softwaretools zur Verfügung, wie beispielsweise *BaseRecalibartor* und *IndelRealigner*, um die Qualität der NGS-Daten zu verbessern. Die gesamte Bearbeitung der NGS-Daten wird in dem *Best Practices-Workflow* zusammengefasst und wurde kontinuierlich in der Analyse unserer Daten genutzt. Des Weiteren empfehlen die ebenso getesteten alternativen Softwaretools zur Identifizierung von Mutationen zuerst die Durchführung des *Best Practices-Workflow* von *GATK*.

Die verfügbaren Tools zur Identifizierung von Mutationen können in zwei Gruppen unterteilt werden. Hierzu ist die zu identifizierende Mutationsart entscheidend, beispielsweise werden bei Keimbahnmutationen *GATK*, *SOAPsnp*, *SVNer* und *SAMtools* bevorzugt [108; 109; 113; 114]. Zur Identifizierung von somatischen Mutationen werden *VarScan*, *MuTect* und *SomaticSniper* bevorzugt [115-117]. Wobei die eigentliche Identifizierung von Mutationen in den NGS-Daten zunächst keinen Unterschied zwischen somatischen und Keimbahnmutationen feststellt. Erst der Vergleich von identifizierten Mutationen im Tumor und identifizierten Mutationen in der korrespondierenden Normalprobe ermöglichten eine Differenzierung. Für die komparative Analyse wurde sowohl *GATK* und *SAMtools* als auch *VarScan* und *SomaticSniper* verwendet. Alternativ konnten zusätzlich komparative Analysen mit Datenbanken Aufschluss über die

Kategorie der Mutationen geben, unabhängig vom Vorhandensein einer korrespondierenden Normalprobe.

Die grundlegenden Unterschiede zwischen den Identifizierungstools sind vor allem die jeweiligen Bearbeitungsschritte des *Alignments* und die Analyse der Mutationen. Das Softwarepaket *GATK*, welches in unserer Pipeline implementiert wurde, nutzt zur Bewertung der Mutationen das Bayes-Theorem (8). Die bedingte Wahrscheinlichkeit der Base in Abhängigkeit der beobachteten Daten (p{B|D}) wird mit Hilfe des Theorems berechnet. Hierzu werden die Wahrscheinlichkeiten der möglichen Base (p{B}), der beobachteten Daten (p{D}) und die Bedingte Wahrscheinlichkeit der Daten in Abhängigkeit der Basen (p{D|B}) benötigt. Die Wahrscheinlichkeit der möglichen Base (p{B}), auch *Priori* genannt, ist ein Erfahrungswert und kann durch vorherige Beobachtungen geschätzt werden. Die vorherigen Beobachtungen resultieren aus den bisherigen Identifizierungen der entsprechenden Base an dieser Position und dem Auftreten dieser Base in der Population. Die Wahrscheinlichkeit der beobachteten Daten ist hierbei für alle möglichen Basen gleich (p{D}). Die Wahrscheinlichkeiten aller möglichen Basen, auch der Variante eingeschlossen, werden summiert ($p\{D\} = \sum_i p\{B_i\} \times p\{D|B_i\}$). Die bedingte Wahrscheinlichkeit der Daten in Abhängigkeit der Basen (p{D|B}) wird aus den NGS-Daten berechnet, wobei die gesamten Reads mit einbezogen werden.

$$p\{B|D\} = \frac{p\{B\} \times p\{D|B\}}{p\{D\}} \qquad (8)$$

Die Bewertung der Mutationen wird auch bei dem verwendeten Softwareprogramm *MAQ* mittels des Bayes-Theorems berechnet. Allerdings führten die vorherigen Bearbeitungsschritte zu unterschiedlichen Ergebnissen, da beispielsweise die zur Berechnung zugelassener Reads auf Grundlage der Position, Qualität und Häufigkeit unterschiedlich gefiltert wurden.

Ein wesentlicher Analyseschritt nach der Identifizierung der Mutationen ist die Differenzierung von somatischen und Keimbahnmutationen. Die evolutionär erworbenen Punktmutationen eines Individuums können fälschlicherweise als relevante Mutationen auf Grundlage der bestehenden Unterschiede zwischen dem Referenzgenom und der sequenzierten humanen Probe, identifiziert werden. Um diesen Störfaktor zu reduzieren, mussten die somatischen von den Keimbahnpunktmutationen differenziert werden. Hierzu ist eine korrespondierende Normalprobe des Individuums die ideale Lösung. Die Keimbahnmutationen wurden zugleich in den NGS-Daten der Tumor- und Normalprobe identifiziert und konnten aus den Ergebnissen ausgeschlossen werden.

Während Softwaretools wie *VarScan* oder *MuTect* die simultane Bewertung der NGS-Daten von Tumor- und Normalprobe mit anschließender Generierung einer Mutationsliste durchführen, werden dabei ausschließlich Kandidaten aus der Schnittmenge der Daten selektiert. Ist die Abdeckung der Normalprobe dabei nicht ausreichend, werden viele Kandidaten aus den Tumor-Daten nicht bewertet. Alternativ können die Mutationslisten pro Probe erstellt werden, beispielsweise mit *GATK*, *SOAPsnp*, *SVNer* oder *SAMtools*. Anschließend wird durch Subtraktion der Listen die tumorspezifische Differenzmenge der Kandidaten generiert. Dabei wird der Verlust potenziell relevanter Mutationen verhindert, welcher durch die simultane Bewertung entsteht. Diese Vorgehensweise wurde für alle DNA-Proben durchgeführt, wenn sowohl eine Tumor- als auch eine korrespondierende Normalprobe vorhanden waren. Die Mutationen, welche keine Informationen aus den Daten der korrespondierenden Normalprobe besitzen, können hierbei nicht eindeutig differenziert werden. Dies führte zu falsch positiven Kandidaten. Nicht immer existieren korrespondierende Normalproben zu den Tumorproben, vor allem bei Zelllinien ist dies oft der Fall. Hierzu musste die Differenzierung mit anderen Mitteln während der Filterung und Annotation durchgeführt werden, um die somatischen Mutationen hervorzuheben bzw. die SNPs herauszufiltern.
Mit Hilfe der Sanger-Sequenzierung wurden anschließend zufällig ausgewählte Mutationskandidaten validiert. Eine hohe Validierungsrate repräsentiert hierbei die Effizienz der Identifizierung von Mutationen und zeitgleich eine erfolgreiche Bearbeitung der NGS-Daten.

5.6 Filterung und Annotation

Nach der Generierung der VCF-Dateien basieren die identifizierten Mutationen auf Unterschiede zwischen den beobachteten Sequenzierungsdaten und des Referenzgenoms. Unter Berücksichtigung des *Best Practices-Workflows* von *GATK* konnte ein Teil des technischen Bias während der Bearbeitung aus den Daten entfernt werden. Um die Anzahl falsch positiver Mutationen zu reduzieren, wurden anschließend weitere Filterschritte durchgeführt. Anhand der Eigenschaften der einzelnen Mutationen in den VCF-Dateien wurden die Kandidaten bewertet und wenn nötig entfernt. Der *Best Practices-Workflow* von *GATK* stellt hierfür eine Auswahl an optimierten Parameter zu Verfügung. Dazu gehören die Qualität der sequenzierten Mutation in Abhängigkeit der Abdeckung (QD), das Verhältnis von vorwärts und rückwärts sequenzierten Reads (FS), die allgemeine Qualität der zugeordneten Reads aller analysierten Proben (MQ) und die Position der Mutation auf allen Reads (ReadPosRankSum). Die jeweiligen Werte für die Parameter basieren auf Erfahrungen und wurden für WES prozessierte

NGS-Daten optimiert. Jedoch sind diese Werte nur Richtwerte und müssen kritisch betrachtet werden, insbesondere wenn die Validierungsraten der Kandidatenlisten nicht ausreichend hoch sind. Im Allgemeinen wurden die vorgegebenen Standardwerte von *GATK* genutzt. Eine Anpassung der Werte wurde beispielsweise bei den FL-Daten durchgeführt. Eine Abdeckung von mindestens zehn Reads (10 *fold*) wurde als Filterkriterium zusätzlich genutzt, um die niedrige Validierungsrate zu verbessern.

Der entscheidende Punkt während der Filterung ist die biologische Relevanz der Kandidaten. Hierzu ist die bereits erwähnte Differenzierung von Keimbahn oder somatischen Mutationen essentiell. Die effizienteste Methode die Keimbahnmutationen aus den Listen zu entfernen, ist die komparative Analyse der Kandidaten aus der korrespondierenden Normalprobe. Hierzu wurde in unserer Pipeline die Subtraktion der Kandidaten aus den Tumor und Normalproben verwendet. Das Ergebnis ist die tumorspezifische Differenzmenge der Kandidaten. Anschließend wurden die falsch positiven Kandidaten der tumorspezifischen Mutationen reduziert, zu denen keine Information aus den korrespondierenden Normalproben verfügbar war. Hierzu wurden bereits bekannte Keimbahnmutationen aus Datenbanken wie 1000Genomes und dbSNP genutzt. Zelllinien oder Tumorproben ohne korrespondierender Normalprobe wurden ausschließlich mit dem Datenbankabgleich verarbeitet. Hierzu muss beachtet werden, dass Datenbanken wie dbSNP Tumorrelevante Mutationen enthalten können. So ist beispielsweise die mit MM assoziierte Mutation V600E in BRAF in der Datenbank enthalten. Eine Filterung der Mutationskandidaten in Abhängigkeit von dbSNP bis zur Version 129 konnte dies verhindern.

Im nächsten Schritt wurden die WES-Daten annotiert, um die Datensätze auf weitere Aspekte hin zu filtern und zu untersuchen. Beispielsweise Eigenschaften wie Chromosom, Position, Lokalisation und Gen Namen wurden hierbei zu den Kandidaten ermittelt. Im Rahmen dieser Arbeit wurden für die Annotation die Softwareprogramme *ANNOVAR* und *SeattleSeq* verwendet. Unter Berücksichtigung des Exom-Extraxtion-Kits und der Version des Referenzgenoms wurden hierbei die Informationen aus Datenbanken wie RefSeq und CCDS extrahiert. Die Lokalisation der Mutationen ermöglichte die nicht-kodierenden Kandidaten aus dem Datensatz zu entfernen. Somit wurden Mutationen zwischen den Genen, zwischen den Exons (*Intron*) und in den UTRs aus den Datensätzen entfernt. Des Weiteren wurden die *synonymous* Mutationen entfernt, welche keine Proteinsequenzveränderung bewirken. Die restlichen Mutationen sind in den kodierenden Regionen lokalisiert und führen zu Aminosäureaustausch in der Proteinsequenz. Zu diesen Kandidaten wurden Vorhersagen zu strukturellen Veränderungen bewertet, basierend auf den Ergebnissen der Softwareprogramme

GERP, *PhastCons* und *PolyPhen*, welche in *SeattleSeq* implementiert sind. Die Bewertung der Kandidaten basiert auf die berechneten Einflüsse der zuvor genannten Softwareprogramme. Zur weiteren Priorisierung von Kandidaten wurden komparative Analysen mit bereits bekannten Mutationen durchgeführt. Hierzu wurden Mutationen aus der Datenbank COSMIC genutzt, die im Zusammenhang mit der Kanzerogenese beschrieben wurden (Abbildung 19). Mit Hilfe von Pathway-Analysen wurden weitere Auswirkungen sowie Zusammenhänge eines Mutationsmusters beurteilt.

Während der Generierung der Pipeline wurde *ANNOVAR* und *SeattleSeq* für diese Aufgabe genutzt. Dabei nutzte *ANNOVAR* zunächst nur einen Teil verfügbarer Programme und Informationen aus den Datenbanken zur Annotation. Zudem waren Konvertierungen der Eingabe-Dateien für die Nutzung erforderlich. Des Weiteren war die Funktion einer Integration von hg19 basierten dbSNP-Daten nicht verfügbar. Die Entscheidung *SeattelSeq* zur Annotation der Daten zu nutzten hatte zur Folge, dass dieser Analyseschritt wie die Identifizierung der Basen nicht direkt in die Pipeline implementiert wurde. Jedoch hat die Web-Client-Applikation den Vorteil, dass alle Annotations- und Analyseschritte parallel durchgeführt werden. Zudem ist die Verarbeitung unterschiedlicher Datei-Formate als Eingabe- und Ausgabe-Dateien gegeben. Des Weiteren wurden aktuelle Versionen der integrierten Programme und Datenbanken genutzt. *SeattleSeq* etablierte sich als Annotationstool in unserer entwickelten Strategie zur Filterung von relevanten Kandidaten und zur Generierung der dafür benötigten Informationen der Mutationen und Gene (Abbildung 19).

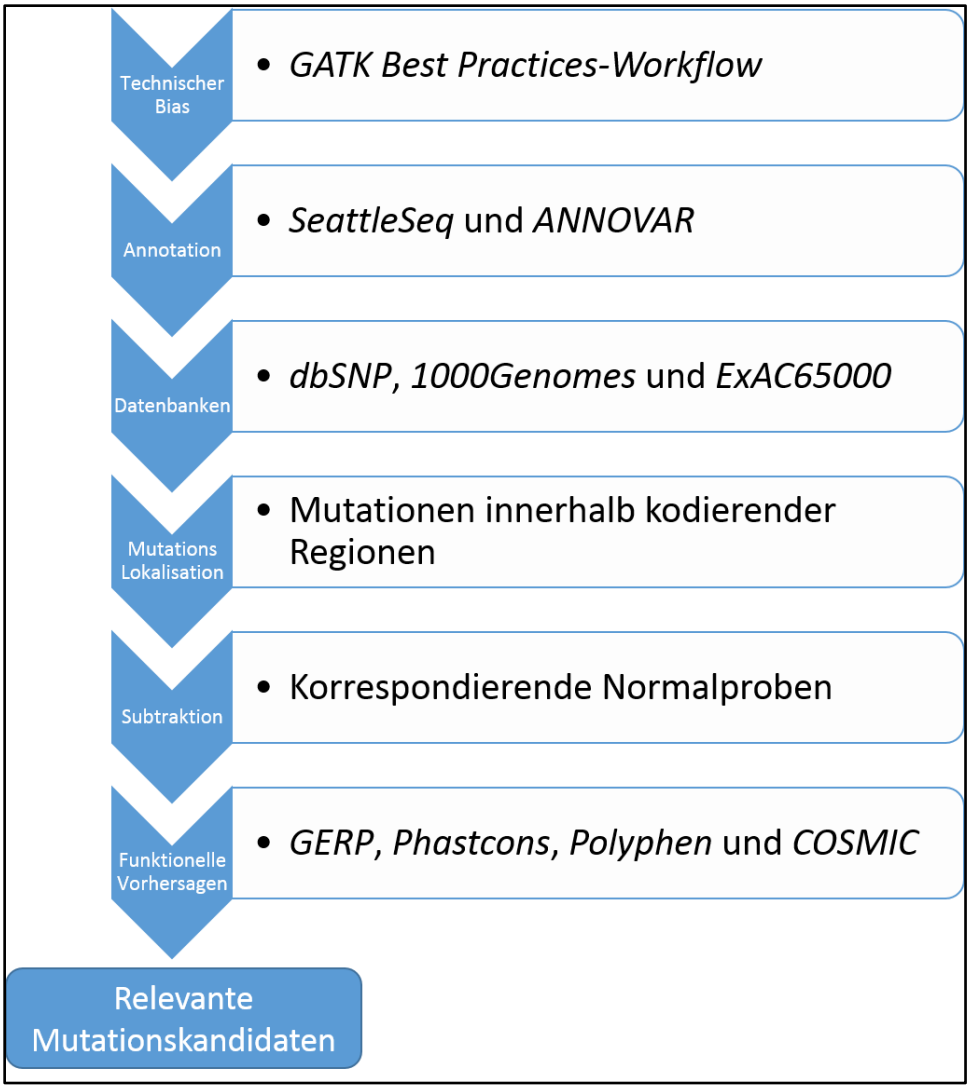


Abbildung 19: Schematische Darstellung zur Filterstrategie für die relevanten Mutationskandidaten
Diese Darstellung zeigt einen Überblick unserer Filterstrategie für die relevanten Mutationskandidaten. Beginnend mit der Annotation der Mutationen über den Abgleich mit Datenbanken wie dbSNP und 1000Genomes sowie die Lokalisation und Subtraktion der Mutationen, bis hin zur funktionellen Vorhersage.

5.7 Biologische Interpretation

Die generierten Ergebnisse der relevanten Mutationen wurden zur Interpretation von funktionellen Auswirkungen in den entarteten Zellen genutzt. Zur Selektion funktioneller Mutationen wurde nach der Annotation und Filterung zusätzlich die Berechnung der Häufigkeit des Auftretens von Genmutationen durchgeführt. Hierzu wurden zunächst alle Entitäten auf Basis der identifizierten Mutationen und ihrer zugehörigen Position untersucht. Bei weiteren Untersuchungen wiederkehrender Mutationen in unterschiedlichen Tumorarten, zeigte sich das viele eine weitaus höhere Heterogenität in der Kanzerogenese aufweisen als zuvor erwartet. Deshalb wurden die genetischen Veränderungen nicht nur positionsabhängig, sondern auch auf Basis der betroffenen Gene und der integrierten Signalwege untersucht. Die bestehende Analyse-Pipeline wurde mit drei Python-Skripten erweitert (9.1.1, 9.1.3 und 9.1.4). Diese

wurden im Rahmen dieser Arbeit neu entwickelt, um die biologische Interpretation zu ermöglichen und zu vereinfachen (6.3).

Die Verwendung von unterschiedlichen Gen Namen in den Datenbanken und Mutationslisten führt zu Unstimmigkeiten bei weiterführenden Analysen. Vor allem bei der Suche nach spezifischen Genen in den Mutationslisten konnten Übereinstimmungen partiell nicht aufgedeckt werden. Zur Lösung dieser Problematik wurde die Annotation der Mutationskandidaten erweitert. Hierzu wurde das Python-Skript *Genemapping* erstellt, welches die Vereinheitlichung der Gen Namen in den VCF-Dateien durchführt (9.1.1).

Zur Visualisierung von häufig mutierten Genen wurden *Heatmaps* erstellt. Dabei werden die sequenzierten Genome und die betroffenen Gene in einer Matrix gegenübergestellt. Ein häufig mutiertes Gen in unterschiedlichen Individuen konnte somit einfach identifiziert und übersichtlich dargestellt werden. Jedoch ist eine einheitliche Syntax der annotierten VCF-Dateien erforderlich, um bereits verfügbare Softwaretools zu nutzen. Nach der erweiterten Annotation mittels des *Genmapping* Skripts wurde das Python-Skript *Heatmap* erstellt. Dadurch konnten sowohl die vom *Genemapping* Skript veränderten VCF-Dateien als auch die annotierten VCF-Dateien der ICGC-Pipeline verarbeiten werden (9.1.3). Für die Zusammenführung unserer Daten mit denen unseres externen Kooperationspartners (ICGC) wurden die zuvor erwähnten Python-Skripte *Genmapping* und *Heatmap* verwendet. Aus rechtlichen Gründen konnten keine FASTQ- oder SAM/BAM-Dateien für die Zusammenführung verwendet werden. Die integrierten VCF-Dateien vom ICGC wurden mittels einer internen ICGC-Pipeline erstellt [45; 126].

Die WES beinhaltet eine Amplifizierung und Extraktion der ausgewählten Regionen, welche nicht immer gleichmäßig effektiv amplifiziert und extrahiert werden. Eine Uniformität von etwa 99 % bei einer Abdeckung von einem Read ist für die WES standardmäßig. Zugleich ist auch eine Uniformität von 90% bei einer Abdeckung von etwa 10 Reads die Regel (Tabelle 15, Tabelle 16, Tabelle 20 und Tabelle 21). Dieser Verlust an Daten kann sich zudem mit den integrierten Filterschritten bei der Identifikation von Mutationen erhöhen. Mit dem Skript *unconReg* wird der Verlust an NGS-Daten, welche nicht in die Analyse einbezogen werden untersucht, um unbeachtete relevante Regionen hervorzuheben (9.1.4).

6 Ergebnisse

Da sich noch keine einheitliche Standardmethode zur Identifizierung von Mutationen durchsetzen konnte und die stetige Entwicklung der Sequenzierungsplattformen und der zugehörigen Softwaretools voran schreitet, ist es essentiell eine gewisse Flexibilität in die Bearbeitungsschritte einfließen zu lassen. Jedoch erwies sich die Realisierung dieses Aspekts als schwierig und als Hauptfehlerquelle vieler unerwünschter Unterbrechungen während der Analyse. Somit waren administrative und programmiertechnische Arbeiten an der Pipeline erforderlich. Zudem konnten weitere Softwareprogramme entwickelt werden, welche die biologische Interpretation der identifizierten Mutationen verbessern.
In den gesamten Ergebnissen stehen die NGS-Daten im Vordergrund, welche mittels der WES und Amplikon Applikationen und der Illumina SBS-Methode generiert wurden.

6.1 Komparative Analyse der Softwareprogramme

Die Selektion der genutzten Softwaretools basiert auf der Berücksichtigung von Hauptfehlerquellen in den einzelnen Analyseschritten und den generierten komparativen Ergebnissen. Dabei werden Vergleiche, Erweiterungen und Verbesserungen der verwendete Softwaretools für die Aufbereitung und Analyse dieser Daten beschrieben. Des Weiteren ist noch zu erwähnen, dass die Performance der Pipeline durch die implementierten Programme und der genutzten Hardware limitiert wurde.

6.1.1 Vergleich der Software zur Basenidentifizierung

Für den Vergleich der *Basecaller* wurde die Normalprobe des Patienten 2 aus dem veröffentlichten ersten WES-MM-Datensatz genutzt [29]. Unter Verwendung der Softwaretools *OLB* Version 1.8.0 und Version 1.9.4, sowie mit *naiveBayesCall* Version 0.3 wurde die Identifizierung der Basen jeweils durchgeführt werden. Als eine weitere Alternative wurde das Softwaretool *ECHO* Version 1.11 in den Vergleich mit einbezogen (Tabelle 12). Dieses Programm berechnet mögliche Fehler bei der Bestimmung der Qualitätswerte der einzelnen Basen und korrigiert diese in den FASTQ-Dateien. Somit konnte *ECHO* erst nach dem eigentlichen Basecalling genutzt werden. Durch die Nutzung dieser alternativen Softwaretools ist eine Verbesserung der Ergebnisse sowohl in der gesamten Anzahl der Reads als auch in den *Alignments* zu erkennen (Tabelle 12). Im Gegensatz dazu konnte Illumina durch die Entwicklung ihrer Software die Ergebnisse des *Basecallings* mit dem *OLB* in der Version 1.9.4 verbessern. Hierzu konnten sowohl gleichwertige aber auch bessere Ergebnisse als mit den Alternativen erzielt werden (Tabelle 12).

Tabelle 12: Vergleich der Softwaretools zur Identifizierung der Basen

Eigenschaften der Sequenzierung und Extraktion:	Readlänge	Exom-Extraktion-Kit	Gesamt Anzahl der Nukleotide	Gesamt Anzahl der Regionen
	76 bp	SeqCapEZ SR	34.950.937 nukleotide	197.210 Regionen

Softwaretools	Gesamt Anzahl der Reads	Alignierte Reads		Eindeutig alignierte Reads		Reads in Regionen des Extraktion-Kits		Ø Abdeckung
Illumina (OLB 1.8.0)	83.807.786	74.083.798	88,40%	62.375.268	74,43%	48.591.716	57,98%	105,7
Illumina (OLB 1.9.4)	83.823.986	75.582.342	90,17%	63.511.658	75,77%	49.246.507	58,75%	107,1
BayesCall (0.3)	83.823.986	75.525.124	90,10%	63.521.860	75,78%	49.428.338	58,97%	107,5
ECHO (1.11)	83.807.786	74.258.804	88,61%	62.509.242	74,59%	48.718.430	58,13%	105,9

Die Verwendung der alternativen *Basecaller* oder Fehlerkorrektur Programme konnten zur Verbesserung der Qualitätswerte in den FASTQ-Dateien führen und infolgedessen die Downstream Analysen aufwerten (Abbildung 20). Jedoch wurde der immense Zeitaufwand in unserem Test als Nachteil angesehen, da die zusätzlichen Berechnungen der alternativen *Basecaller* bis zu mehrere Tage für eine Probe benötigten. Im Vergleich dazu war die Ausbeute an qualitativ hochwertigen Basen relativ geringfügig. Aufgrund der stetigen Entwicklung der Softwaretools von Illumina hinsichtlich Qualität und Effektivität, konnte ab einen gewissen Zeitpunkt von der Verwendung anderer Softwaretools abgesehen werden. Ein Vorteil des Basecalling mittels *RTA* war die zusätzliche Zeitersparnis, da die Berechnung während der Sequenzierung durchgeführt wurde. In unserem Vergleich ist ersichtlich, dass der *OLB* Version 1.9.4 im Gegensatz zur Version 1.8.0 das *Alignment* hinsichtlich der Anzahl an Reads in allen Analyseschritten überlegen ist und zum Teil wurden auch bessere Ergebnisse im Vergleich zu den Alternativen erzielt (Tabelle 12).

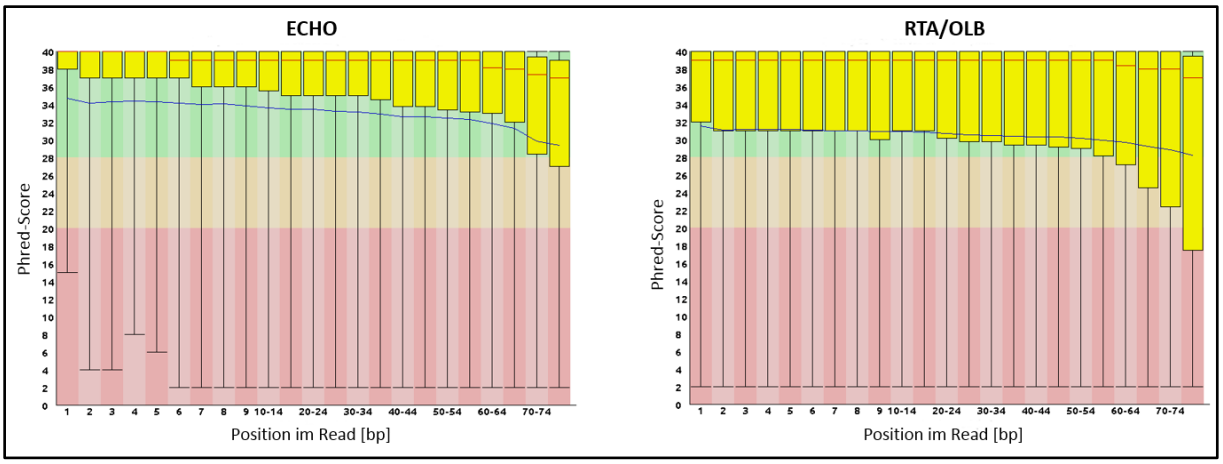


Abbildung 20: Vergleich der generierten Phred-Scores
Zur Erstellung der Plots wurde das Programm FastQC von Simon Andrews genutzt. Die Eingabedaten sind die bearbeiteten FASTQ-Dateien des ersten Reads der MM-Patient 2 Probe. Auf der rechten Seite sind die generierten Phred-Scores vom *OLB* version 1.8.0 gegen die Position im Read geplottet. Dabei wurde die FASTQ-Datei des ersten Reads der MM1176 Probe genutzt. Als Vergleich auf der linken Seite ist die mit ECHO überarbeiteten FASTQ-Datei erneut mit den Qualitätswerten und der Position der Basen geplottet.

Die Identifizierung der Basen wurde in den letzten Jahren stetig verbessert. Alternative *Basecaller*, die Verwendung neuer Chemikalien und die Entwicklung einer neuen

Durchflusszelle konnten bei Illumina nicht nur die Qualitäten der identifizierten Basen verbessern, sondern haben auch dazu beigetragen, dass die Länge der Reads verlängert und die Ausgabemengen der Sequenzierer um ein vielfaches erhöht wurden. Wodurch der Faktor Zeit noch mehr an Einfluss auf die gesamte Analyse gewinnt.

Die schnellen Illumina-Softwareprogramme unterliegen einer Lizenz und aus rechtlichen Gründen konnte die Implementierung von *RTA* und *OLB* nicht in der Pipeline realisiert werden. Für die Verwendung der entwickelten Pipeline bedeutet dies, dass das Basecalling als Aufbereitung der Daten vorausgesetzt wird. Diese Entscheidung wurde durch die nicht überzeugende Performance bzw. dem hohen Zeitaufwand bei der Nutzung der alternativen *Basecaller* bestärkt. Die Ergebnisse zeigen, dass die Verwendung der neuesten Version des *OLB* zu qualitativ hochwertigen Daten führt (Tabelle 12). Somit wurden für die intern generierten NGS-Daten jeweils die neuste Version der Software von Illumina zur Identifizierung der Basen genutzt. Des Weiteren zeigen die Ergebnisse das eine erneute Identifizierung der Basen mittels einer neueren Version von *OLB* zur Erhöhung der resultierenden NGS-Daten aus älteren Sequenzierungen führt (Tabelle 12). Die Voraussetzung hierfür ist die Speicherung der benötigten Rohdaten der Sequenzierung. Diese stehen meist nicht zur Verfügung sowie bei unseren externen Sequenzierungsdaten, welche durch unsere industriellen Kooperationspartner erstellt wurden.

6.1.2 Selektion der Software zur *Alignment* Generierung

Wie bereits beschrieben können die *Alignment* Softwaretools zwar verglichen werden, aber eine Aussage darüber zu treffen welcher Aligner der Beste ist gestaltet sich schwierig, da alle Tools ihre Vor- und Nachteile haben [87]. Um einen geeigneten Aligner für unsere Daten zu bestimmen sind die Kriterien, wie eine hohe Abdeckung, Genauigkeit, Geschwindigkeit und vor allem eine stetige Entwicklung in Bezug auf Veränderungen in der NGS-Sequenzierung für die Entscheidung ausschlaggebend.

Als ersten *Aligner* wurde *MAQ* in der entwickelten Pipeline genutzt, wobei qualitativ hochwertige *Alignments* als Ergebnis erwartet wurden. Jedoch zeigte sich während dieser Arbeit, dass die Bearbeitungszeit ein großer Nachteil für die gesamte Analyse ist. Alternativ konnte die Illumina Software genutzt werden. Die interne Pipeline generierte jedoch eine schlechte Ausbeute, vor allem bei den Ergebnissen der Identifizierung von Mutationen (Abbildung 21 **B**). Des Weiteren wurde die kommerzielle Software NextGENe von Softgenetics in den Test mit einbezogen. Der *Alignment*-Algorithmus beim NextGENe-Tool führte durch eine interne Assemblierung zu langen Bearbeitungszeiten, jedoch wurden sehr gute

Ergebnisse erzielt (Abbildung 21). Der Nachteil dieses Algorithmus war, dass der vorhandene 128 GB Arbeitsspeichers bereits mit unserem Testdatensatz, von etwa 12 WES-MM-Proben, an die Grenzen stieß.

Für die Generierung der *Alignments* kristallisierten sich die Programme *Bowtie*, *SOAP* und *BWA* als gleichwertig heraus, unter Berücksichtigung externer Ergebnisse[103; 127; 128]. Ein großer Nachteil von *Bowtie* war, dass mögliche Lücken in den Reads zunächst nicht mit einbezogen wurden. Zudem konnten sowohl *Bowtie* als auch *SOAP* bei der stetigen Entwicklung von *BWA* nicht mithalten. Dies und die bereits erworbene Erfahrung mit *MAQ* führten dazu den *BWA-Aligner* von Heng Li in die entwickelte Pipeline zu implementieren. Die stetige Entwicklung der Software in Bezug auf die Qualität der *Alignments*, die Qualität der einzelnen Basen und die Berücksichtigung der immer länger werdenden Reads unterstützten diese Entscheidung. Zudem ist die aktuell standardmäßige Benutzung des *BWA-Aligner* in großen Sequenzierungsprojekten wie 1000Genomes, TCGA und ICGC ein zusätzliches Auswahlkriterium. Des Weiteren setzt die Nutzung des *GATKs* zur Identifikation der Mutationen die Generierung der zuvor benötigten *Alignments* mittels des *BWA-Aligner* voraus. *GATK* verweist stets darauf, dass die gesamten Analyseschritte sowie Entwicklungen des Software-Pakets auf die Ausgabedateien des *BWA-Aligner* angepasst sind. Es zeigt sich auch in dem Vergleich der Punktmutationsidentifizierungstools das *BWA* und *GATK* das beste Ergebnis im Test erreichten (Abbildung 21).

Um das jeweilige *Alignment* pro Probe zu bewerten, können die Anzahl der Reads zu den einzelnen Schritten summiert werden (Tabelle 13**A**). Dazu wurden die gesamten, als Paar alignierten, eindeutigen und in den extrahierten Regionen lokalisierten Reads nacheinander gefiltert und gezählt. Ein Überblick der Uniformität wurde mit den aufsteigenden Faktoren wie 1, 5, 10, 25 und 50 generiert (Tabelle 13**B**). Hierzu wurden die Formeln zur Berechnung der *ØAbdeckung* und der *%Abdeckung* verwendet (5)(7).

Tabelle 13: *Alignment* und Uniformität

Eigenschaften der Sequenzierung und Extraktion:	Readlänge	Exom-Extraktion-Kit	Gesamt Anzahl der Nukleotide	Gesamt Anzahl der Regionen
	76 bp	SeqCap EZ Human Exome Library v2.0	44.055.647 Nukleotide	244.519 Regionen

A

Zelllinie	Gesamt Anzahl der Reads	Alignierte Reads		Eindeutig alignierte Reads		Reads in Regionen des Extraktion-Kits		Ø Abdeckung
INA6	83.167.494	82.864.343	99,64%	68.400.524	82,24%	54.081.167	65,03%	93,3
JJN3	210.512.990	178.586.244	84,83%	160.230.401	76,11%	124.229.174	59,01%	214,3

B

Zelllinie	Uniformität n ≥ 1		Uniformität n ≥ 5		Uniformität n ≥ 10		Uniformität n ≥ 25		Uniformität n ≥ 50	
INA6	42.462.731	96,38%	40.589.894	92,13%	38.608.924	87,64%	33.283.299	75,55%	26.227.302	59,53%
JJN3	42.759.980	97,06%	41.979.385	95,29%	41.404.642	93,98%	39.903.766	90,58%	36.820.684	83,58%

6.1.3 Vergleich der Software zur Mutationsidentifizierung

Die NGS-Daten von 2 MM Tumorzelllinien, MM1S und U266, wurden für den Vergleich der Mutationsidentifizierungstools genutzt. Die Generierung der Exom-Sequenzierungsbibliothek wurde mit dem Extraktions-Kit *SeqCap 2.1M Human Exome Array* von Nimblegen durchgeführt. Die *Alignments* wurden nach der Identifizierung der Basen mit dem zur Pipeline empfohlenen Aligner erstellt. Die kommerziellen Pipelines von Softgenetics und Illumina nutzen integrierte Aligner, welche jeweils intern entwickelt wurden. Hierzu verwendet Illumina das Softeareprogramm *Efficient Large-Scale Alignment of Nucleotide Database* (*Eland*), welches das *Alignment* ohne Berücksichtigung von möglichen Lücken in den Reads generiert. Softgenetics nutzt die Pipeline NextGENe, welche einen Aligner basierend auf den BWT-Algorithmus besitzt. Zu diesen Softwaretools können nur wenig detaillierte Informationen eingesehen werden, da diese Pipelines kommerziellen Lizenzen unterliegen. Wie bereits erwähnt ist *BWA* für *GATK* der empfohlene *Aligner*, im Gegensatz hierzu ist bei *MAQ* sowohl der Aligner, als auch das Mutationsidentifizierungstool im Programmpaket implementiert. Für den Vergleich wurden die Pipelines mit den jeweiligen Standardparametern genutzt und die identifizierten Kandidaten nicht gefiltert (Abbildung 21**A**). Somit wurde der Bias durch unterschiedliche Filtermethoden und Bewertungen der Softwaretools ausgeschlossen. Des Weiteren wurden 27 Mutationen mittels Sanger-Sequenzierung untersucht und als Referenz für eine weitere Gegenüberstellung verwendet (Tabelle 14). Obwohl die Illumina Pipeline mit einer guten Performance bei der Gegenüberstellung der Softwaretools überzeugte, konnte keine der 27 bereits validierten Mutationen identifiziert werden, weshalb Illumina bei dieser Gegenüberstellung der Mutationsidentifizierungstools fehlt (Abbildung 21**B**).

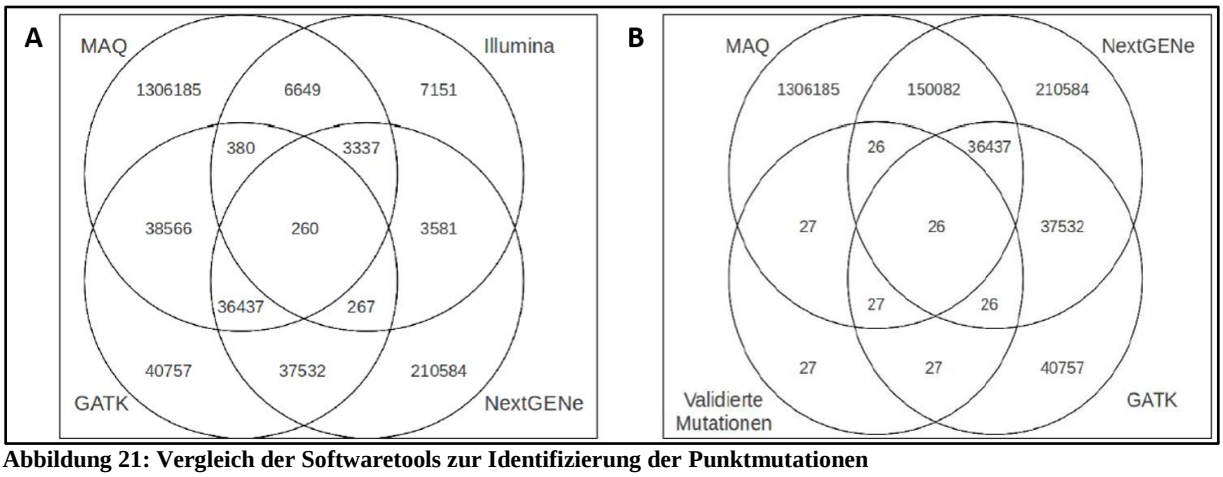


Abbildung 21: Vergleich der Softwaretools zur Identifizierung der Punktmutationen
(A) Zur Erstellung der Mutationsdaten wurden 4 Tumorzelllinien des ersten MM-WES-Datensatzes sequenziert und zum Vergleich der Softwaretools ein Überlapp der Ergebnisse erstellt. (B) Um die Identifizierungsrate zu bestimmen wurden 27 Punktmutationen, die zuvor mit der Sanger-Sequenzierung bestätigt werden konnten, als Referenz genutzt. (B) Die mit Eland identifizierten 7151 Mutationskandidaten beinhalteten keinen der 27 validierten Mutationen, weshalb Illumina in dieser Gegenüberstellung fehlt.

Im Gegensatz dazu konnten bei *MAQ* alle Mutationen identifiziert werden, jedoch war die Analyse sehr Zeit intensiv. Der Grund hierfür ist die benötigte Rechenzeit von *MAQ* für das *Alignment*, alternativ kann *BWA* das *Alignment* ohne Qualitätsverlust etwa 10 – 20 mal schneller durchführen [103]. Diese Zeitersparnis konnten wir während der komparativen Analyse feststellen. Sowohl die Performance als auch die Identifizierungsrate der Pipelines NextGENe und *GATK* konnten bei der Gegenüberstellung überzeugen. Nachfolgende Analysen mit größeren Datensätzen wurden durchgeführt, um diese zwei Pipelines direkt miteinander zu vergleichen. Die integrierte Methode zur Identifizierung der Mutationen der NextGENe Pipeline generiert eine lokale Assemblierung der einzelnen Regionen und nutzt diese Information, um die Proben untereinander zu vergleichen. Dies ist jedoch Arbeitsspeicher intensiv und konnte für eine Analyse von 12 Proben mit einem 128 GB großen Arbeitsspeicher nicht durchgeführt werden. Somit war die Nutzung dieses Tools für größere Projekte ungeeignet. Die Ergebnisse der gesamten Gegenüberstellung führten zur Entscheidung *GATK* als Softwaretool für die Identifizierung der Mutationen zu nutzen. Weitere Vorteile von *GATK* waren ein schneller Support und eine stetige Entwicklung. Während der Generierung unserer Pipeline ist die Version 2.0 von *GATK* lizensiert worden, um die kommerzielle Nutzung einzuschränken. Dies hatte zunächst keine Auswirkungen auf die Implementierung von *GATK* als Softwaretool, da die folgenden Versionen stets zur akademischen Nutzung freigegeben wurden.

Tabelle 14: Mit Sanger-Sequenzierung validierte SNV

Chromosom	Position	Referenz	MM1S	U266	Allelen	AA-Austausch
chr1	144930805	G	S	G	G/C	PRO,ALA
chr1	164815893	C	S	C	C/G	HIS,ASP
chr1	205633769	A	M	M	A/C	TRP,GLY
chr2	42522290	T	K	K	T/G	VAL,GLY
chr2	48018227	G	G	R	G/A	GLY,ASP
chr3	52588770	C	Y	C	C/T	VAL,MET
chr3	70001015	G	K	G	G/T	GLN,HIS
chr3	128204758	G	G	R	G/A	PRO,LEU
chr3	128369597	G	S	G	G/C	THR,SER
chr3	188327019	C	C	Y	C/T	PRO,LEU
chr4	1962801	G	R	G	G/A	GLU,LYS
chr4	54257205	G	R	N	G/A	GLY,ARG
chr7	55266457	G	S	G	G/C	GLY,ARG
chr7	140453132	T	T	W	T/A	LYS,ASN
chr9	20414231	C	C	M	C/A	GLU,stop
chr9	100456006	C	S	C	C/G	ASP,HIS
chr11	22646800	G	G	R	G/A	ALA,VAL
chr11	108536057	C	C	G	C/G	ASN,LYS
chr11	118376934	G	R	G	G/A	ALA,THR
chr12	25398284	C	S	C	C/G	GLY,ALA
chr13	48934228	A	A	G	A/G	LYS,ARG
chr13	48951156	G	G	T	G/T	GLU,stop
chr14	66975262	T	T	C	T/C	MET,THR
chr15	91328204	G	G	R	G/A	ASP,ASN
chr16	79632985	C	T	C	C/T	ARG,HIS
chr17	7578449	C	C	T	C/T	ALA,THR
chr22	36710303	T	Y	T	T/C	ASN,ASP

6.2 Pipeline Entwicklung zur Identifizierung von Mutationen

Zur Generierung der Pipeline wurden die Softwaretools basierend auf den Ergebnissen der komparativen Analysen ausgewählt. Hierbei wurde für das *Alignment BWA* und für diverse Zwischenschritte zur Aufbereitung der Daten sowie für die Identifizierung der Mutationen *GATK* ausgewählt. Die jeweiligen Softwaretools basieren auf unterschiedlichen Programmiersprachen wie C oder Java. Zudem ist die standardmäßige Nutzung dieser Programme für Unix-basierenden Systemen mit Ausführung per Kommandozeilen ausgelegt. Um diese Tools zu vereinigen und eine einheitliche Analyse durchführen zu können wurde ein Python-Skript entwickelt (9.1). Die einzelnen Kommandos wurden mit Hilfe des *Subprocess-Moduls* von Python nacheinander ausgeführt. Die Durchführung einzelner oder die Verschiebung von Analysenschritten ist in dem Skript vorgesehen und schnell durchführbar. Die Zwischenergebnisse der Analyse wurden zunächst mit gespeichert, um die Fortschritte der Analyse bei unvorhersehbaren Unterbrechungen nicht zu verlieren. Zudem wurden die Zwischenergebnisse für weitere Bearbeitungsschritte wie zur Berechnung der durchschnittlichen und prozentualen Abdeckung genutzt. Nach der gesamten Analyse, kann

Speicherplatz durch das Entfernen dieser Zwischenergebnisse wieder freigegeben werden.
Die Pipeline benötigt für die standardmäßige Analyse der NGS-Daten zunächst die FASTQ-Dateien. Des Weiteren werden Eingabe-Dateien wie das entsprechende Referenzgenom, die dbSNP-Daten sowie die Regionen des Exom-Extraktion-Kit für die Analyse vorausgesetzt. Die Pipeline führt die einzelnen Schritte des Alignmnets mit dem integrierten Aligner BWA durch, gefolgt von den Berechnungen zur Identifizierung der Mutationen mit *GATK*. Zudem werden Berechnungen erstellt die nach einer Aufbereitung zur Qualitätsprüfung der Daten genutzt wurden. Die Annotation und Filterung der Daten erfolgte mit *SeattleSeq*, weitere Untersuchungen, Analyseschritte und Aufbereitungen der Resultate konnten anschließend mit zusätzlichen Python-Skripten durchgeführt werden. Diese wurden je nach Problematik neu entwickelt. Nach der Validierung der Mutationen mit Sanger, können einzelne Eigenschaften aus den VCF-Dateien zu den positiven oder negative detektierten Kandidaten genutzt werden, um die zuvor verwendeten Parameter der einzelnen Analyseschritte anzupassen. Dies impliziert eine Wiederholung einzelner Analyse- oder Filterschritte um die Ergebnisse der Identifizierung zu verbessern. Dabei konnten die jeweiligen zu wiederholenden Schritte nach der Anpassung der Parameter mit Hilfe der Pipeline trivial durchgeführt werden (Abbildung 22).

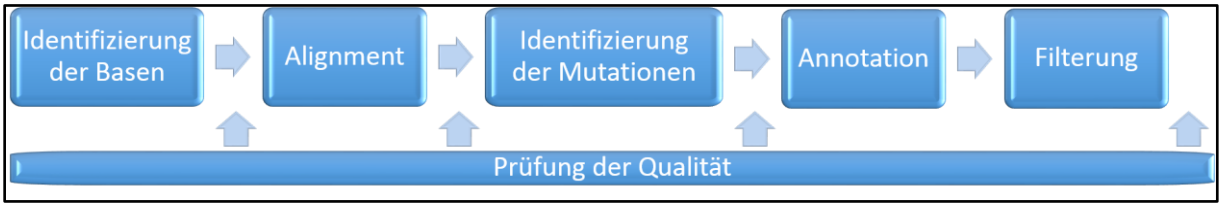


Abbildung 22: Schematische Darstellung der Pipeline
Die Darstellung zeigt einen Überblick der einzelnen Analyseschritte und deren Reihenfolge zur entwickelten Pipeline. Zudem sind die möglichen Prüfungen der Qualität zu den NGS-Daten angegeben.

6.3 Weitere Entwicklungen

Während der Implementierung der Pipeline und bei der biologischen Interpretation der Ergebnisse entstanden weitere Fragestellungen, die nicht direkt mit den verwendeten Softwaretools beantwortet werden konnten. Um die bisherigen Analysen der NGS-Daten zu verbessern und die unbeantworteten Fragestellungen zu untersuchen wurden weitere Skripte erstellt. Hierzu gehören die Erweiterung der Annotation, eine Identifizierung der Genmutationsfrequenz und die detaillierte Untersuchung von abgedeckten und nicht abgedeckten Regionen in den Illumina WES-Daten.

6.3.1 Entwicklung des Genemapping Skripts

Nach der Annotation wurden die Mutationen hinsichtlich der betroffenen Gene untersucht. Hierfür liegen Gene im Fokus, welche bereits mit der Kanzerogenese assoziiert wurden, an der

B-Zell Reifung beteiligt sind oder Gene zu denen Medikamente für die Therapie existieren. Mit der Annotation wurden verfügbare Informationen aus Datenbanken zu den Genen mit den identifizierten Mutationen verknüpft. Dabei wurde die lokalisierte Position der Mutation für die Verbindung genutzt. Das Symbol zur Beschreibung des Gens wurde durch die verwendete Datenbank bestimmt. Hierbei wird entweder das eigentliche Symbol, ein Synonym oder ein aus anderen Organismen bekanntes Symbol für dasselbe identifizierte Gen verwendet.

Eine komparative Analyse basierend auf Gen Symbole kann ohne Kenntnis der alternativen Symbole nicht korrekt durchgeführt werden. Auch der Vergleich von Mutationslisten unterschiedlicher Studien kann durch die unterschiedlich genutzten Konventionen der Gen Symbole zu falschen Ergebnissen führen. Die *human genome organisation* (HUGO) gründete ein Ausschuss, um eine vorschriftsmäßige Benennung der humanen Gene zu erstellen. Dabei wurden jeweils ein Name und ein Symbol pro Gen von dem *HUGO gene nomenclature committee* (HGNC) bestimmt. Des Weiteren wurden die alternativ genutzten Namen und Symbole in der HGNC-Datenbank verknüpft und gespeichert. Mit dieser HGNC Datenbank und dem *Genmapping* Skript konnte das Problem der Mehrdeutigkeit von annotierten Gen Symbolen gelöst werden. Hierzu durchsucht das *Genemapping* Skript mit den annotierten Gen Symbolen aus den Mutationslisten die gesamte HGNC Datenbank (siehe 9.1.1). Sobald ein Symbol in der Datenbank gefunden wird, können die verfügbaren Informationen zu diesem Gen bereitgestellt werden. Diese Informationen können anschließend in die Mutationslisten eingefügt werden und bewirken eine einheitliche Benennung der betroffenen Gene. Zudem werden alle bekannten Alternativen mit aufgelistet. Mit Hilfe der einheitlichen Gen Symbole wurden komparative Analysen ohne Verlust effizient durchgeführt. Hierzu gehören die Suche nach spezifischen Gen-Gruppen oder Vergleiche von Mutationslisten. Das *Genemapping* Skript nutzt für diese Analyseschritte die zuvor bearbeitete VCF-Datei. Eine Erweiterung der Informationen zu den Genen konnte nach diesem Prinzip simpel durchgeführt werden, beispielsweise mit der Beschreibung des Gens hinsichtlich der Funktion. Hierzu wurde der Zugriffschlüssel (*Accession number*) aus der HGNC Datenbank zur Verknüpfung mit den Informationen aus der ENSEMBL Datenbank mit dem *Genemapping* Skript genutzt. Des Weiteren wurden zu sequenzierten FL-Proben auch Daten zur Genexpression erstellt, diese Daten konnten mit Hilfe des *Genemapping* Skript mit den Mutations-Daten der FL-Proben verknüpft werden.

6.3.2 Entwicklung des *Heatmap* Skripts

Zu der biologischen Interpretation der Daten gehört auch die Identifizierung und Untersuchung von wiederkehrenden Mutationen. Diese können für das Verständnis einer Erkrankung und vor allem auch für die Herstellung von molekularen Targets für die gezielte Therapie von Krebserkankungen nützlich sein. In den großen Sequenzierungsprojekten wie ICGC und TCGA ist die Verteilung der wiederkehrenden Mutationen oder die Häufigkeit mutierter Gene zu den einzelnen Entitäten ein Kriterium zur Priorisierung der Mutationen. In unseren NGS-Daten konnten hierzu eher wiederkehrende Punktmutationen beim BL als beim FL oder MM beobachtet werden. Dies kann auf die Heterogenität der Entitäten zurückgeführt werden. Beispielsweise ist die im MM bekannte BRAF Mutation V600E mit nur etwa 4 % eine der häufiger beobachteten Punktmutation [120].

Das *Heatmap* Skript wurde für die übersichtliche Darstellung der Genmutationsfrequenz entwickelt. Die VCF-Dateien beinhalten pro Zeile eine Position die in mindestens einer Probe mutiert ist. Am Ende dieser Zeile werden die Proben mit und ohne Mutation spaltenweise aufgelistet (Abbildung 15). Somit ist eine Gegenüberstellung von häufig mutierten Genen basierend auf der veränderten Position in den VCF-Dateien gegeben. Betroffene Gene können jedoch durch unterschiedliche Punktmutationen in ihrer Funktion beeinflusst werden, weshalb die positionsbasierende Analyse zu stringent ist. Daraus folgte eine Erweiterung der Basis auf die genkodierende Region des jeweiligen Gens. Mit Softwaretools wie beispielsweise *vcftools* können diese Analysen durchgeführt werden. Als Eingabe-Dateien werden annotierten VCF-Dateien benötigt, jedoch ist die Mehrdeutigkeit der Gen Symbole in diesen Ergebnissen persistent. Deshalb wurde zunächst für eine effiziente Analyse das *Genemapping* Skript zur Vereinheitlichung der Annotation genutzt. Die daraus entstandenen veränderten VCF-Dateien können nicht mit den bereits verfügbaren Softwaretools analysiert werden. Hierfür wurde das *Heatmap* Skript erstellt (9.1.3). Eine weitere Eigenschaft wurde in dem entwickelten *Heatmap* Skript implementiert, dass sowohl unserer veränderten VCF-Dateien als auch die ICGC VCF-Dateien verarbeitet werden können. Dies ermöglichte die Analyse der häufig mutierten Gene des FL-Datensatz von unseren 11 sequenzierten Proben auf insgesamt 49 Proben.

Die generierte Matrix des *Heatmap* Skitps repräsentiert die Mutationsverteilung des analysierten Datensatzes unter Berücksichtigung der Häufigkeit mutierter Gene und der betroffenen Positionen. Um die Information der Position der einzelnen Mutationen nicht zu verlieren, wurden diese zu jeder Zeile der Matrix in einer zusätzlichen Spalte gelistet. Durch gezielte Sortierung der Proben oder Gene in dieser Matrix wurden neben der Untersuchung der Häufigkeit mutierter Gene auch Gruppen von Proben oder Genen differenziert. Beispielsweise

konnten in unserem Datensatz des FL die Proben der FL-t(14;18)-positiven und FL-t(14;18)-negative separiert werden. Aus der resultierenden Matrix konnten spezifische Gene zu den FL-Gruppen identifiziert werden. Im Rahmen des MM-Projektes konnte unter Verwendung der Pipeline drei molekulare Subgruppen im RTK- und Adhäsions-Netzwerks identifiziert werden [29]. Die Akkumulation der Mutationen in den spezifischen Gen-Gruppen können mit Hilfe des *Heatmap* Skripts visualisiert werden.

6.3.3 Entwicklung des *unconReg* Skripts

Zur Qualitätsprüfung des *Alignment* wurden die Menge an Reads und die durchschnittliche Abdeckung (*ØAbdeckung*) der extrahierten Region untersucht. Des Weiteren konnte die Effizienz der Extraktion überprüft werden, indem die Verteilung der Reads durch die prozentuale Abdeckung (*%Abdeckung*) berechnet wurde. Der limitierende Faktor (*n fold*) bestimmt dabei die Menge an ausreichend sequenzierten Basen ($\#Base_{covered}$). Diese Regionen werden bei der Identifizierung der Mutationen miteinbezogen. Alle anderen Basen ($\#Base_{uncovered}$) werden bei den Downstream Analysen nicht weiter beachtet. Deshalb ist nicht die in der Literatur häufig hervorgehobene *ØAbdeckung* essentiell für die Identifizierung der Mutationen, sondern die *%Abdeckung*, welche den Anteil repräsentiert der effizient analysiert werden kann. Der restliche Anteil ist nicht ausreichend abgedeckt und gehört somit zu den unbeachteten Regionen. Zur Entstehung solcher Regionen tragen sowohl der technische als auch der biologische Bias indirekt bei, indem beispielsweise das Ergebnis des *Alignment* beeinflusst wird.

Um die Identifizierung von unbeachteten Regionen zu veranschaulichen wurde die Abbildung der möglichen Zuordnung von Reads (Abbildung 17) modifiziert. Für dieses Beispiel wird vorausgesetzt, das die Reads allen Filterkriterien entsprechen und dem Referenzgenom zugeordnet werden können. Des Weiteren ist der Faktor für die Berechnung der *%Abdeckung* gleich 1, die $\#Base_{covered}$ und $\#Base_{uncovered}$ in der Probe können jeweils mit Weiß und Schwarz hervorgehoben werden (Abbildung 23**A**).

Diese probenspezifische Differenzierung der Regionen kann pro sequenziertem Individuum durchgeführt werden, wobei die Mengen und Lokalisierungen an $\#Base_{covered}$ und $\#Base_{uncovered}$ variieren (Abbildung 23**B**). Dieser Unterschied ist auch zwischen den Tumor und den korrespondierenden Normalprobe zu beobachten. Werden bei der Identifizierung der Mutationen die Basen simultan betrachtet, entfallen sowohl die $\#Base_{uncovered}$ der Tumorprobe als auch die $\#Base_{uncovered}$ der korrespondierenden Normalprobe jeweils in der gesamten Analyse der Tumorprobe. Hinzu kommen bei den simultan bewertenden Softwaretools die

jeweiligen Standardwerte für die minimale Abdeckung der Tumorprobe und der korrespondierenden Normalprobe, welche nicht wie in diesem Beispiel dem Faktor 1 entsprechen. Somit erhöht sich die Menge an $\#Base_{uncovered}$ und der Verlust an sequenzierten Daten bei dieser Art der Analysen zur Identifizierung von somatischen Mutationen.

Für die Betrachtung von unbeachteten Regionen in NGS-Daten basierend auf der Ebene eines Projekts oder einer Studie, müssen die Klassen für die Differenzierung erweitert werden. Hierzu wurden im folgenden Beispiel die Regionen, welche in allen Proben nicht abgedeckt sind wie zuvor als schwarze Region zusammengefasst und alle Basen, die in jeder Probe abgedeckt sind wie zuvor als weiße Region definiert (Abbildung 23**C**). Die neue Klasse fasst alle Basen, die in mindestens einer Probe nicht abgedeckt sind, zur grauen Region zusammen (Abbildung 23**C**). Die Klassifizierung der Regionen wurde mit einem zusätzlichen Faktor erweitert. Ähnlich wie bei der *%Abdeckung* limitiert dieser Faktor die ausreichend abgedeckten Regionen, wobei die Anzahl der Proben zur Differenzierung genutzt wird. Hierzu kann die Anzahl an Proben sowohl zur Einteilung von weißen und grauen Regionen als auch zur Einteilung von grauen und schwarzen Regionen verwendet werden.

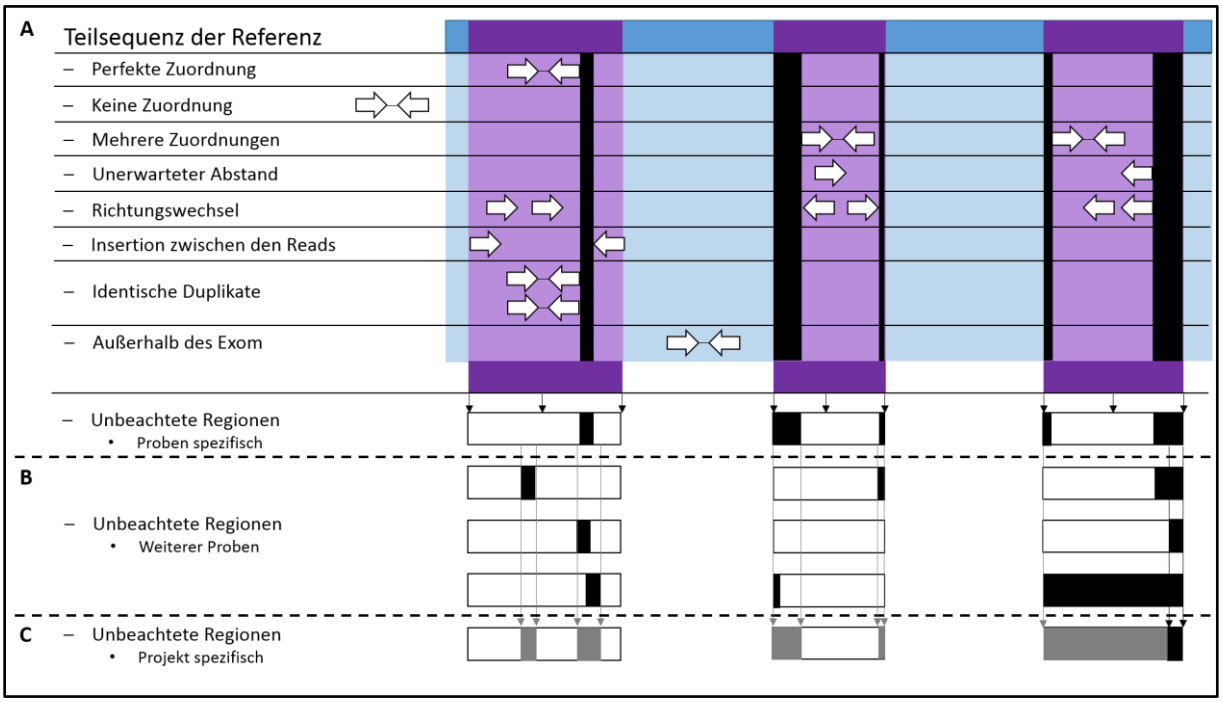


Abbildung 23: Schematische Darstellung unbeachteter Regionen
(A) Die Zuordnung der Reads zum Referenzgenom beim *Alignment* (Abbildung 17). (B) In dieser Darstellung sind die unbeachteten Regionen schwarz hervorgehoben. Diese repräsentieren die Regionen im Exom, welche nicht sequenziert wurden. Durch die Generierung der Sequenzierungsbibliothek bei der WES-Applikation, durch strukturelle Veränderungen, durch einzelne Mutationen sowie durch einen hohen GC-Gehalt entstehen diese Regionen. (C) Werden mehrere Proben zusammengefasst und analysiert, können bestimmte Regionen in der einen Probe abgedeckt werden und in einer weiteren Probe nicht. (C) Diese Projektspezifischen Regionen sind grau markiert. (C) Regionen die im zusammengefassten Datensatz nicht abgedeckt sind werden bei der gesamten Analyse nicht beachtet und sind schwarz hervorgehoben.

Für die Identifizierung und Annotation von unbeachteter Regionen wurde das Skript *unconReg* entwickelt (9.1.4). Dabei werden zunächst die Regionen eines Individuums in Abhängigkeit der Abdeckung in weiße und schwarze Bereiche unterteilt. Daraus wird eine zusätzliche SAM/BAM-Datei erstellt, welche die nicht ausreichend abgedeckten Regionen enthält. Die dazu benötigten Analyseschritte könnten in die erstellte Pipeline zur Identifizierung von somatischen Mutationen mit eingefügt werden, um die Verluste der spezifischen Proben zu ermitteln. Jedoch benötigt die Untersuchung der Verluste eines Projekts alle Daten der einzelnen Proben. Diese werden erst nach der gesamten Verarbeitung der einzelnen Proben erhoben. Somit ist die gesonderte Durchführung der Analyse zur Ermittlung der Verluste effektiver als die Implementierung dieser Teilanalyse in der bereits vorhandenen Pipeline.
Ein wichtiger Aspekt bei der Identifizierung von unbeachteten Regionen ist die Detektionsgrenze von Mutationen. Hierzu ist die in der Literatur angegebene Mindestanforderung der Abdeckung von etwa 13 Reads zur Identifizierung von heterozygoten Mutationen gleichzusetzen mit dem Faktor *n fold* für die Differenzierung von schwarzen und weißen Regionen der einzelnen Proben [110; 111]. Hierbei wird die ursprüngliche Definition der schwarzen Regionen von nicht sequenzierten zu nicht ausreichend sequenzierten Regionen erweitert. Somit ist zu den technischen und biologischen Einflüssen des *Alignments* zusätzlich die limitierende Abdeckung ein essentieller Faktor für die Berechnung der Verluste. Wird dieser Faktor *n fold* auf 13 erhöht, ist der Verlust an Daten in den einzelnen Proben auch höher (Abbildung 18). Diese Verluste betreffen in unterschiedlichen Proben nicht immer die gleichen Regionen, weshalb eine Erhöhung des Faktors *n fold* auch nachfolgend die grauen und schwarzen Regionen im gesamten Projekt erhöht. Ebenfalls ist auch die Filterstrategie zur Differenzierung von somatischen und Keimbahnmutation entscheidend. Bei der Subtraktion führen nur die $\#Base_{uncovered}$ der Tumorproben zu einem Verlust der Daten. Wobei wie zuvor erwähnt die simultane Handhabung der Tumor- und der korrespondierenden Normalprobe zur Summierung der $\#Base_{uncovered}$ beider Proben und somit zu einem höheren Verlust der Daten führt. Beispielsweise nutzen Softwaretools wie *VarScan* oder *MuTect* standardmäßig 8 Reads als Parameter für die Mindestanforderung der Abdeckung in den sequenzierten Normalproben. Der standardmäßig verwendete Parameter für die erforderliche Abdeckung zu den sequenzierten Tumorproben beträgt bei *VarScan* 6 Reads und bei *MuTect* 14 Reads. Diese Werte limitieren die Verwendung der NGS-Daten in Abhängigkeit der Abdeckung ohne die SAM/BAM-Dateien direkt zu verändern. Die generierten VCF-Dateien beinhalten die Ergebnisse der Mutationsidentifizierung, aber der Verlust an Daten in den Dateien ist nicht erkennbar. Wird sogar eine Identifizierung der Mutationen mit erhöhten Parametern für die

erforderte Abdeckung durchgeführt, so entstehen meist unbemerkt weitere unbeachtete Regionen.

Um das Ausmaß und den möglichen Einfluss der unbeachteten Regionen zu bewerten wurde das *unconReg*-Skript erweitert. Hierbei nutzt das *unconReg*-Skript das Softwarepaket *BEDTools* um die Regionen zu identifizieren und die veränderten SAM/BAM-Dateien zu generieren. Hierfür wurde zunächst der Faktor 1 als Parameter für die Differenzierung von weißen und schwarzen Regionen genutzt. Für einen gesamten Datensatz entstehen aus der Schnittmenge aller einzelnen schwarzen Regionen der Proben die schwarzen, unbeachteten Regionen. Aus den restlichen Differenzmengen aller einzelnen schwarzen und weißen Regionen der Proben wurden die grauen Regionen extrahiert. Diese Regionen, insbesondere die schwarzen Regionen, werden in den standardmäßigen Analysen nicht weiter untersucht. Deshalb wurde eine anschließende Annotation der Regionen und ein Abgleich dieser mit der COSMIC Cancer Gene Census Datenbank durchgeführt. Hierbei wurden relevanten Genen identifiziert, welche mit der Kanzerogenese assoziiert und sowohl graue als auch schwarze Regionen in den NGS-Daten enthalten.

Die WES-Daten beinhalten zu den identifizierten schwarzen Regionen keine Informationen. Eine anschließende Sequenzierung mit Sanger oder Amplikon-Sequenzierung ermöglicht eine Analyse dieser Regionen. Dadurch können sowohl neue als auch bereits bekannte Mutationen identifiziert werden, welche zunächst nicht in den WES-Daten detektiert wurden. Die grauen Regionen wurden ebenfalls in den relevanten Genen lokalisiert und können Mutationen enthalten. Die WES-Daten enthalten zu diesen Regionen nur einen gewissen Anteil an Informationen zu den jeweiligen Proben. Um beispielsweise die Häufigkeit des Auftretens einer Mutation im gesamten Datensatz zu bestimmen, müssten die Proben mit nicht ausreichender Abdeckung erneut sequenziert werden. Ohne diese erneute Untersuchung würden komparative Analysen in den grauen Regionen nicht effizient durchgeführt werden.

Mit den technischen Neuerungen sowohl der Sequenzierungsplattformen als auch der Exom-Extraktion-Kits konnten die technischen Einflüsse reduziert werden, welche zur Entstehung von unbeachteten Regionen beitragen. Vor allem die Einflüsse der Regionen mit hohen oder niedrigen Anteil an G/C-Gehalt konnten reduziert werden, diese entstehen während der Amplifikation und der Extraktion des Exoms [77; 78; 129; 130]. Auch die Verlängerung der prozessierten Reads konnten die Ergebnisse des *Alignments* verbessern. All diese Verbesserungen haben die Anzahl der unbeachteten Regionen um ein vielfaches reduzieren, aber nicht gänzlich unterbunden. Beim derzeitigen Standardprozess der WES können immer noch unbeachtete Regionen identifiziert werden. Vor allem die simultane Differenzierung von

somatischen und Keimbahnmutationen führt zu unbemerkten Verlust von bereits sequenzierten Regionen in den Tumorproben. Dies kann auch dazu führen, dass Punktmutationen nicht identifiziert werden. Durch die Berechnung der unbeachteten Regionen und unter Berücksichtigung der tumorspezifischen Mutationsraten der jeweiligen Entitäten, kann die mögliche Anzahl beziffert werden. Dies ist ein wesentlicher Punkt der bedacht werden muss, wenn die NGS-Technologie zur Diagnostik genutzt werden soll.

6.4 Ergebnisse der analysierten NGS-Projekte

Während und nach der Entwicklung der Pipeline zur Identifizierung von somatischen Mutationen in hämatologischen Erkrankungen, wurden unter Verwendung von verfügbaren und neu entwickelten Skripten (*Genemapping*, *Heatmap* und *unconReg*) Ergebnisse zu den Krebsentitäten wie MM, FL und BL generiert. Zunächst wurden Punktmutationen selektiert, gefolgt von der Filterung, Annotation und biologischer Interpretation zur Identifizierung von Tumor-assoziierten Punktmutationen (Abbildung 22). Mit der entwickelten Filterstrategie konnten wir in unseren Daten 90% oder auch höhere Validierungsraten erreichen (Abbildung 19 und Abbildung 25). Des Weiteren konnten neue Erkenntnisse in den einzelnen Projekten bereits veröffentlicht werden [29; 45; 131].

Im Folgenden werden die Ergebnisse zu den einzelnen Entitäten detailliert hervorgehoben und mit Bezug zur Pipeline sowie den neu erstellten Skripten dargestellt.

6.4.1 Ergebnisse zum Multiplen Myelom

Der erste veröffentlichte Datensatz zum MM beinhaltet 6 Zelllinien und 5 primäre Tumorproben mit der korrespondierenden Normalprobe [29]. Hierzu sind folgend die Bewertungen des *Alignments* aufgelistet (Tabelle 15). Ein Großteil der durchgeführten komparativen Analysen für die Selektion der in unserer Pipeline implementierten Softwaretools basieren auf diesem Datensatz (6.2).

Tabelle 15: Bewertung des *Alignment* und Uniformität des ersten MM Datensatz

Eigenschaften der Sequenzierung und Extraktion:	Readlänge	Exom-Extraktion-Kit	Gesamt Anzahl der Nukleotide	Gesamt Anzahl der Regionen
	36 bp	SeqCap 2.1M Human Exome Array	34.950.937 Nukleotide	197.210 Regionen
Eigenschaften der Sequenzierung und Extraktion:	Readlänge	Exom-Extraktion-Kit	Gesamt Anzahl der Nukleotide	Gesamt Anzahl der Regionen
	76 bp	SeqCap 2.1M Human Exome Array	34.950.937 Nukleotide	197.210 Regionen
Eigenschaften der Sequenzierung und Extraktion:	Readlänge	Exom-Extraktion-Kit	Gesamt Anzahl der Nukleotide	Gesamt Anzahl der Regionen
	76 bp	SeqCap EZ Human Exome Library v2.0	44.055.647 Nukleotide	244.519 Regionen

A: Alignment

Probe	Gesamt Anzahl der Reads	Alignierte Reads		Eindeutig alignierte Reads		Reads in Regionen des Extraktion-Kits		Ø Abdeckung
AMO1	181.145.710	171.936.836	94,92%	93.741.009	51,75%	48.498.086	26,77%	50,0
U266	71.750.604	65.576.591	91,40%	49.830.083	69,45%	27.156.695	37,85%	28,0
OPM2	206.298.436	195.335.221	94,69%	106.108.739	51,43%	64.606.993	31,32%	66,5
MM1S	119.042.886	114.606.217	96,27%	87.168.802	73,22%	49.843.849	41,87%	51,3
JJN3	210.512.990	178.586.244	84,83%	160.230.401	76,11%	124.229.174	59,01%	214,3
L363	187.605.370	155.783.403	83,04%	137.976.735	73,55%	111.160.450	59,25%	191,8
Patient 1	120.820.946	99.272.046	82,16%	93.497.915	77,39%	75.632.883	62,60%	130,5
Patient 2	86.599.878	80.077.796	92,47%	67.623.691	78,09%	53.544.584	61,83%	116,4
Patient 3	159.054.206	148.169.215	93,16%	113.862.835	71,59%	78.907.000	49,61%	171,6
Patient 4	182.978.916	168.741.205	92,22%	125.586.403	68,63%	89.740.672	49,04%	195,1
Patient 5	106.666.010	94.717.872	88,80%	76.848.530	71,70%	56.355.505	52,83%	122,5

B: Uniformität

Probe	n ≥ 1		n ≥ 5		n ≥ 10		n ≥ 25		n ≥ 50	
AMO1	34.235.159	97,95%	32.961.489	94,31%	30.896.020	88,40%	22.743.046	65,07%	11.409.540	32,64%
U266	33.853.696	96,86%	31.278.302	89,49%	26.833.989	76,78%	13.790.503	39,46%	3.967.267	11,35%
OPM2	34.489.136	98,68%	33.757.779	96,59%	32.458.598	92,87%	26.589.212	76,08%	16.990.218	48,61%
MM1S	34.262.922	98,03%	33.140.982	94,82%	30.971.189	88,61%	25.239.933	72,22%	15.784.315	45,16%
JJN3	42.759.980	97,06%	41.979.385	95,29%	41.404.642	93,98%	39.903.766	90,58%	36.820.684	83,58%
L363	43.095.972	97,82%	42.452.063	96,36%	41.924.223	95,16%	40.302.467	91,48%	36.557.191	82,98%
Patient 1	42.599.017	96,69%	41.407.044	93,99%	40.314.742	91,51%	36.920.209	83,80%	30.562.765	69,37%
Patient 2	34.747.985	99,42%	31.525.791	90,20%	29.188.759	83,51%	24.053.085	68,82%	18.960.918	54,25%
Patient 3	34.205.917	97,87%	33.121.688	94,77%	31.787.278	90,95%	28.077.998	80,34%	23.422.872	67,02%
Patient 4	34.290.987	98,11%	33.371.007	95,48%	32.251.088	92,28%	29.076925	83,19%	24.887.075	71,21%
Patient 5	32.688.628	93,53%	25.525.330	73,03%	21.418.033	61,28%	17.870.389	51,13%	15.822.028	45,27%

Die Identifizierung der Mutationen sowie die anschließende Annotation und Filterung wurde wie zuvor beschrieben durchgeführt (Abbildung 25). Zudem wurde die Verteilung der Mutationen untersucht, wobei die Anzahl der identifizierten Mutationen in Zelllinien höher ist als bei primären Patienten Proben. Des Weiteren konnten in den Patientendaten mit Hilfe der korrespondierenden Normalproben die Mutationen entfernt werden, welche bereits in dem Genom der Normalprobe identifiziert wurden. Dies zeigt den immensen Vorteil der Nutzung von korrespondierenden Normalproben zur Identifizierung somatischer Mutationen.

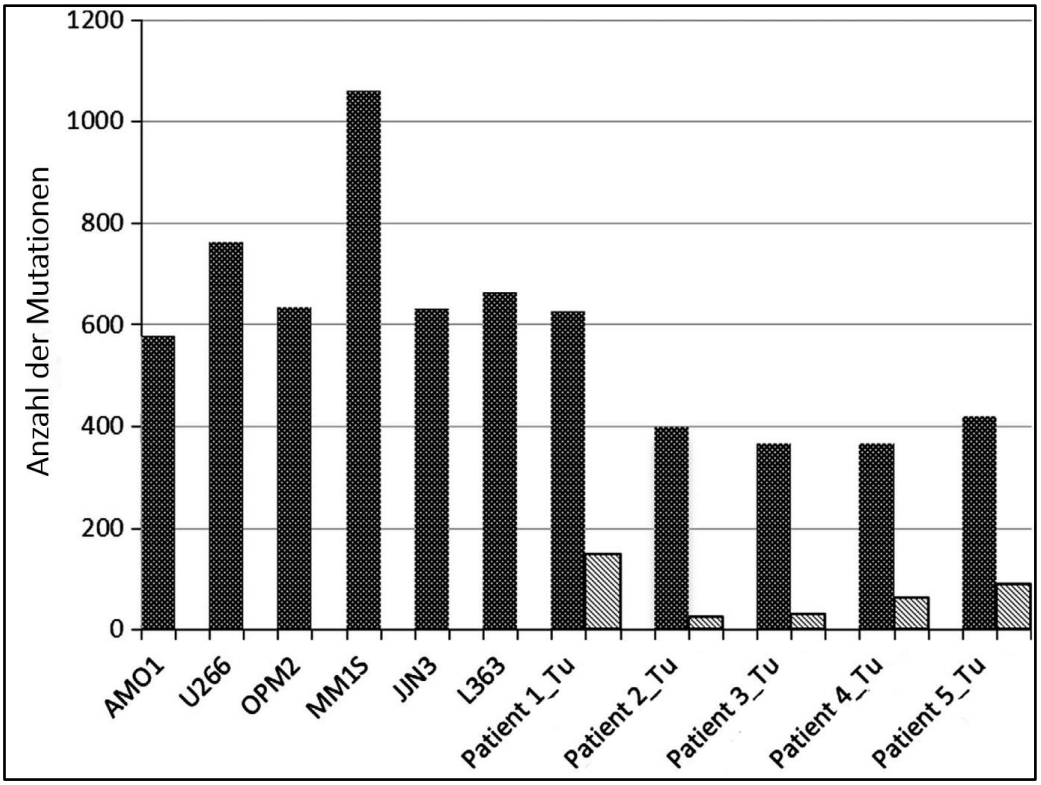


Abbildung 24: Mutationsverteilung im ersten MM Datensatz, Abbildung nach Leich et. al[29]
In diesem Plot sind die jeweilige Anzahl der identifizierten Mutationen zu den sequenzierten Proben gegenübergestellt. Hierzu sind die Zelllinien auf der linken Seite gruppiert und die primären Patientenprobe auf der rechten Seite. Die grauen Balken repräsentieren die Anzahl der Mutationen, welche nach der Subtraktion von Mutationskandidaten die sowohl in der Tumorprobe als auch in der korrespondierenden Normalprobe identifiziert wurden.

Das MM ist eine sehr heterogene Entität in der die Untersuchung wiederkehrender Mutationen basierend auf der genomischen Position nicht ausreichend ist. Aus diesem Grund wurde die Analyse auf Basis der Gen Namen erweitert. Hierzu konnten mit dem Genmapping Skript die VCF-Dateien bearbeiten und auch bereits veröffentlichte Ergebnisse zum MM integriert werden. Diese Erweiterung mit externen Ergebnissen von Chapman et. al 2011, konnte den bestehenden Datensatz von 5 auf 43 primäre Tumorproben vergrößern [30]. Hierdurch wurde die Effizienz der Filterung von relevanten somatischen Mutationen gerade im Hinblick auf Zelllinien gesteigert, welche über keine korrespondierende Normalproben verfügen [29].

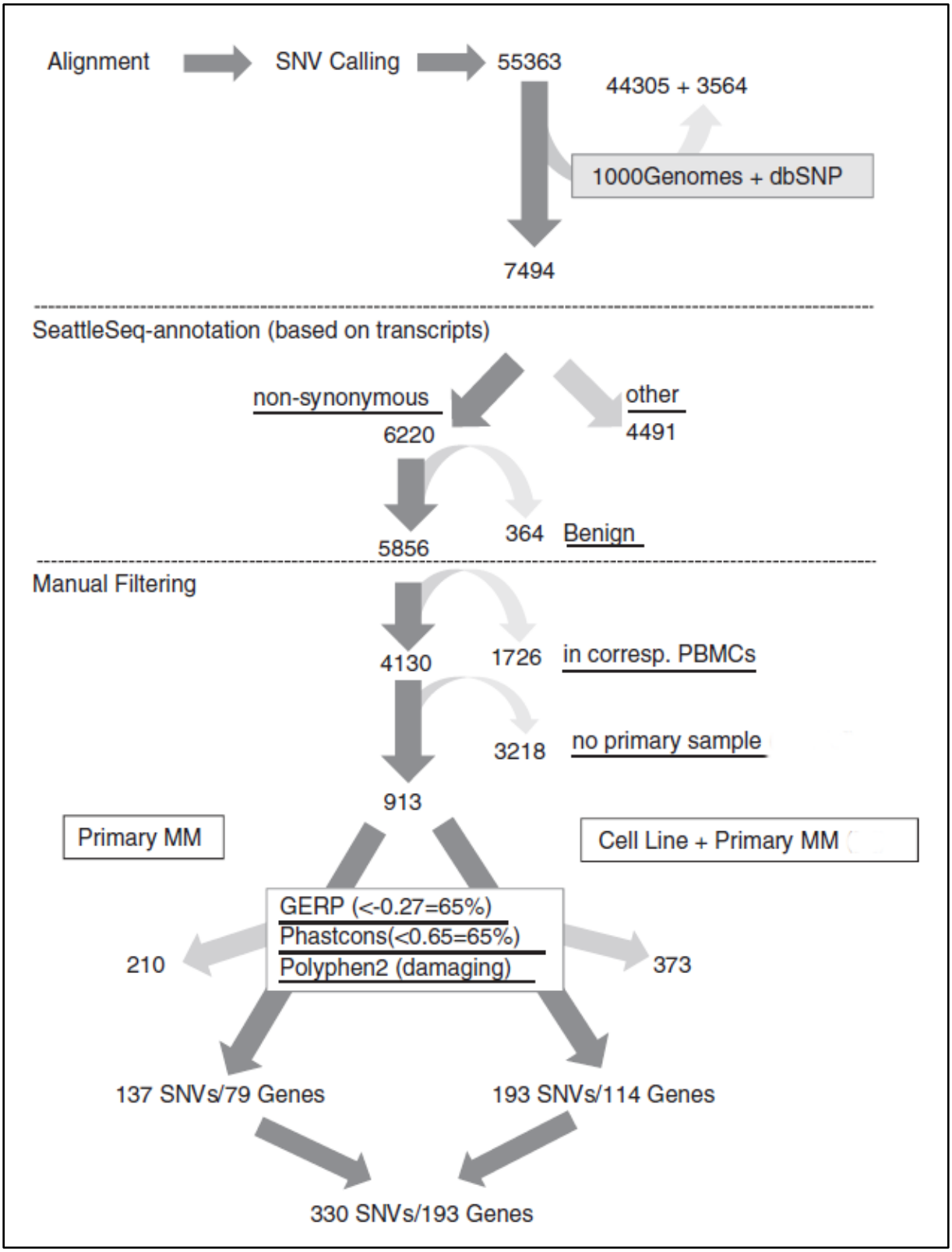


Abbildung 25: Filterstrategie der Mutationskandidaten, Abbildung nach Leich et al. [29]
In dieser Darstellung ist unsere verwendete Filterstrategie zum ersten MM-WES-Datensatzes im Detail abgebildet. Beginnend vom Abgleich der Datenbanken dbSNP und 1000Genomes über die Annotation der Mutationen und der anschließenden manuellen Filterung der relevanten Mutationskandidaten. Zusätzlich wurden mit Hilfe des Genemapping Skript externe Ergebnisse aus Chapman et al. integriert und zur Vergrößerung des Datensatzes sowie zur Selektion der relevanten somatischen Mutationen genutzt.

Die Untersuchung wiederkehrender Mutationen in diesem erweiterten MM-Datensatz führte zu dem Ergebnis, dass vermehrt somatische Mutationen in einem RTK- und Adhäsions-Netzwerk identifiziert wurden [29]. Wie bereits beschrieben spielen die Zell-Adhäsion und RTK Signalwegen sowie deren Downstream Genen eine wichtige Rolle bei der B-Zell Entwicklung. Zur Visualisierung der Mutationsverteilung in diesen Gen-Gruppen wurde das *Heatmap* Skript entwickelt.

Der zweite MM-Datensatz bestand aus 20 primäre MM Proben mit korrespondierenden Normalproben. Für die Sequenzierungsbibliothek und der Sequenzierung wurde der externe Service von GATC genutzt. Die Proben wurden mit dem Exom-Extraktions-Kit *SureSelect Human All Exon V5* von Agilent bearbeitet und Reads mit einer Länge von 101 Basen zur Sequenzierung mit dem HiSeq2500 genutzt. Zur Aufbereitung der MM-Proben wurde eine *whole genome amplification* (WGA) durchgeführt. Dieses Protokoll nutzt GATC, um aus den meist geringen Probenmaterial der MM-Proben eine ausreichende Menge an genomischer DNA zu generieren. Die Analyse der Daten und auch die folgende Bewertung des *Alignments* wurde mit der entwickelten Pipeline durchgeführt (Tabelle 16).

Tabelle 16: Bewertung des *Alignment* und Uniformität des zweiten MM Datensatz

Eigenschaften der Sequenzierung und Extraktion:	Readlänge	Exom-Extraktion-Kit	Gesamt Anzahl der Nukleotide	Gesamt Anzahl der Regionen
	101 bp	SureSelect Human V5	50.621.021 Nukleotide	230.418 Regionen

A: Alignment

Probe	Gesamt Anzahl der Reads	Alignierte Reads		Eindeutig alignierte Reads		Reads in Regionen des Extraktion-Kits		Ø Abdeckung
P1	84.743.909	82.518.077	97,37%	76.932.629	90,78%	60.354.555	78,45%	120,4
P2	93.781.734	91.943.134	98,04%	86.245.372	91,96%	67.431.020	78,19%	134,5
P3	76.903.011	74.551.509	96,94%	68.841.403	89,52%	52.665.378	76,50%	105,0
P4	84.058.609	82.075.986	97,64%	76.447.032	90,94%	57.191.871	74,81%	114,1
P5	92.216.278	89.914.208	97,50%	82.077.332	89,01%	64.338.227	78,39%	128,3
P6	105.480.213	102.796.208	97,46%	91.088.156	86,36%	70.245.142	77,12%	140,1
P7	114.554.418	111.800.547	97,60%	102.615.115	89,58%	79.799.654	77,77%	159,2
P8	71.442.471	69.316.377	97,02%	64.682.813	90,54%	49.523.280	76,56%	98,8
P9	104.056.871	101.714.400	97,75%	92.101.962	88,51%	66.497.327	72,20%	132,6
P10	70.647.170	67.847.659	96,04%	61.460.641	87,00%	43.829.764	71,31%	87,4
P11	76.883.425	73.051.676	95,02%	65.055.224	84,62%	46.715.470	71,81%	93,2
P12	67.250.029	64.861.516	96,45%	57.670.502	85,76%	41.116.271	71,30%	82,0
P13	87.901.613	82.375.667	93,71%	75.046.641	85,38%	53.351.968	71,09%	106,4
P14	108.090.522	105.366.394	97,48%	95.879.968	88,70%	72.576.128	75,69%	144,8
P15	77.271.163	75.196.946	97,32%	65.695.650	85,02%	48.141.352	73,28%	96,0
P16	100.270.582	94.096.547	93,84%	81.776.411	81,56%	58.372.511	71,38%	116,4
P17	103.574.738	99.157.946	95,74%	88.798.722	85,73%	63.155.386	71,12%	126,0
P18	98.799.868	95.448.210	96,61%	85.403.176	86,44%	61.998.432	72,59%	123,7
P19	99.493.364	96.578.037	97,07%	86.831.129	87,27%	62.667.466	72,17%	125,0
P20	179.889.446	173.619.845	96,51%	158.186.417	87,94%	70.600.816	44,63%	140,9

B: Uniformität

Probe	n ≥ 1		n ≥ 5		n ≥ 10		n ≥ 25		n ≥ 50	
P1	44.812.389	88,53%	42.173.113	83,31%	40.366.635	79,74%	36.397.607	71,90%	30.246.498	59,75%
P2	48.286.956	95,39%	47.094.451	93,03%	46.057.405	90,98%	42.821.178	84,59%	35.668.101	70,46%
P3	44.439.972	87,79%	41.530.447	82,04%	39.478.612	77,99%	34.781.431	68,71%	27.572.298	54,47%
P4	48.086.538	94,99%	46.668.017	92,19%	45.413.587	89,71%	41.270.546	81,53%	32.398.288	64,00%
P5	48.319.578	95,45%	47.086.590	93,02%	45.992.902	90,86%	42.353.928	83,67%	34.504.744	68,16%
P6	46.560.427	91,98%	44.817.940	88,54%	43.537.844	86,01%	40.182.989	79,38%	33.833.429	66,84%
P7	49.758.609	98,30%	49.117.149	97,03%	48.471.175	95,75%	45.964.542	90,80%	39.577.166	78,18%
P8	43.772.672	86,47%	40.813.216	80,63%	38.814.703	76,68%	34.075.926	67,32%	26.584.191	52,52%
P9	49.713.212	98,21%	49.036.738	96,87%	48.322.064	95,46%	45.211.472	89,31%	37.189.757	73,47%
P10	43.705.075	86,34%	40.783.223	80,57%	38.908.948	76,86%	34.181.391	67,52%	25.948.778	51,26%
P11	42.968.579	84,88%	39.615.471	78,26%	37.487.371	74,05%	32.918.720	65,03%	25.902.609	51,17%
P12	40.081.336	79,18%	36.249.127	71,61%	33.918.590	67,00%	29.110.163	57,51%	22.237.234	43,93%
P13	44.368.709	87,65%	41.362.498	81,71%	39.280.550	77,60%	34.461.449	68,08%	27.127.068	53,59%
P14	50.345.883	99,46%	50.143.720	99,06%	49.770.399	98,32%	47.454.718	93,75%	40.180.514	79,38%
P15	44.830.728	88,56%	42.195.705	83,36%	40.321.000	79,65%	35.519.930	70,17%	27.262.688	53,86%
P16	50.211.378	99,19%	49.488.567	97,76%	48.205.327	95,23%	43.179.309	85,30%	33.347.442	65,88%
P17	50.265.456	99,30%	49.566.488	97,92%	48.254.599	95,33%	43.042.514	85,03%	33.382.625	65,95%
P18	50.300.463	99,37%	49.875.027	98,53%	49.146.391	97,09%	45.338.869	89,57%	35.952.610	71,02%
P19	50.291.100	99,35%	49.743.695	98,27%	48.650.017	96,11%	43.851.673	86,63%	34.107.761	67,38%
P20	50.307.024	99,38%	50.033.640	98,84%	49.261.719	97,31%	45.484.432	89,85%	38.223.183	75,51%

Ebenso wurde das Genemapping und Heatmap Skript in die Analyse mit eingebunden. Somit konnten alle Patienten einheitlich annotiert werden, in Abhängigkeit der mutierten Gen-

Gruppen sowie der Mutationsfrequenz. Unter Berücksichtigung der Gene in dem RTK- und Adhäsions-Netzwerk wurden die Patienten unterteilt. Hieraus resultierte eine Gliederung der Patienten in drei molekulare Untergruppen (Tabelle 17). Eine Patientengruppe besitzt Mutationen nur in Adhäsionsmolekülen. Eine weitere Gruppe enthält Mutationen sowohl in Adhäsionsmolekülen als auch in Downstream Genen. Die dritte Patientengruppe kennzeichnet sich durch mindestens eine identifizierte Mutation in den RTKs oder durch zusätzliche Mutationen in Adhäsionsmolekülen und/oder Downstream Genen. Die weitere Analyse unter Berücksichtigung klinischer Daten dieser Patienten ist noch nicht abgeschlossen.

Tabelle 17: Ergebnisse der molekularen Subgruppen im MM-Datensatz

Gengruppe	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P26	P17	P18	P19	P20
RTK	1x	5x	1x	5x	1x	2x														
Adhäsion	7x	22x	2x	4x			2x	4x	7x	3x	6x	1x	6x	2x	1x	7x	12x	5x	1x	3x
Downstream	2x	9x	1x	2x	1x		3x	4x	1x	1x	3x	1x	3x							

Eine detaillierte Analyse der unbeachteten Regionen wurde zunächst mit dem ersten Datensatz der MM Proben durchgeführt, wobei weitere verfügbare Proben zu den veröffentlichten Datensatz mit aufgenommen wurden. Hierzu zählen nicht die zuvor untersuchten Patienten aus dem zweiten MM-WES-Datensatz, da diese zu einem späteren Zeitpunkt sequenziert wurden. Der erste MM-WES-Datensatz vergrößerte sich durch die Aufnahme weiterer Proben auf insgesamt 23 Proben, 7 Zelllinien und 8 primäre Fälle mit korrespondierender Normalprobe (Tabelle 15 und Tabelle 18).

Tabelle 18: Bewertung des *Alignment* und Uniformität der zusätzlichen Proben des ersten MM Datensatz

Eigenschaften der Sequenzierung und Extraktion:	Readlänge	Exom-Extraktion-Kit	Gesamt Anzahl der Nukleotide	Gesamt Anzahl der Regionen
	76 bp	SeqCap EZ Human Exome Library v2.0	44.055.647 Nukleotide	244.519 Regionen

A: Alignment

Probe	Gesamt Anzahl der Reads	Alignierte Reads		Eindeutig alignierte Reads		Reads in Regionen des Extraktion-Kits		Ø Abdeckung
INA6	83.167.494	82.864.343	99,64%	68.400.524	82,24%	54.081.167	65,03%	93,3
Patient 6	73.059.112	70.113.894	95,97%	65.545.013	90,43%	51.731.670	70,81%	89,2
Patient 7	98.006.227	97.705.090	99,69%	82.853.408	84,54%	66.202.020	67,55%	114,2
Patient 8	81.713.386	79.338.751	97,09%	73.048.899	89,40%	61.412.331	75,16%	105,9

B: Uniformität

Probe	n ≥ 1		n ≥ 5		n ≥ 10		n ≥ 25		n ≥ 50	
INA6	42.462.731	96,38%	40.589.894	92,13%	38.608.924	87,64%	33.283.299	75,55%	26.227.302	59,53%
Patient 6	42.688.604	96,90%	41.541.541	94,29%	40.283.275	91,44%	35.287.541	80,10%	25.676.777	58,28%
Patient 7	42.776.928	97,10%	41.915.211	95,14%	40.995.047	93,05%	37.545.455	85,22%	29.925.770	67,93%
Patient 8	42.915.438	97,41%	41.996.995	95,33%	41.021.339	93,11%	37.178.791	84,39%	29.020.784	65,87%

Um das Ausmaß der unbeachteten Regionen so effizient wie möglich zu bestimmen, wurde der gesamte Datensatz trotz unterschiedlicher Bearbeitungen genutzt und ausgewertet (Tabelle 19). Dabei ist das Exom zum Teil mit den *SeqCap 2.1M Human Exome Array* Kit und dem *SeqCap*

EZ Human Exome v2.0 Kit von Nimblegen extrahiert worden. Die Längen der prozessierten Reads entsprechen 36 und 76 Basen, wobei alle WES-Sequenzierungen der Proben mit dem Sequenzierer GA II x von Illumina durchgeführt wurden.

Tabelle 19: Ergebnisse der unbeachteten Regionen im MM-Datensatz

MM-Datensatz	Weiße Region	Graue Region	Schwarze Region
Anteil des Exoms [%]	60,85 (54 - 93)	38,5 (6,7 - 45)	0,65 (2,5 - 25)
Basen [MBp]	27 (24 - 41)	17 (3 -20)	0,3 (1,1 - 11)
Regionen [#]	233309	1180744	4639
Gene [#]	17154	17971	2497

Die hohe Anzahl an grauen Regionen ist auf die Vermischung der unterschiedlich bearbeiteten Proben zurück zuführen, wobei zeitgleich die Induzierung der schwarzen Regionen durch einen technischen Bias ausgeschlossen werden kann. Die anschließende Annotation und der Vergleich mit der Cancer Gene Census Datenbank dienen zur Identifizierung von relevanten Gene in den grauen und schwarzen Regionen. Die Priorisierung dieser Gene entsteht durch die Assoziation tumorrelevanter Gene in den unbeachteten Regionen, welche in der Cancer Gene Census Datenbank gelistet sind. Des Weiteren kann die Filterung relevanter Gene hinsichtlich Entitäten wie dem MM spezifiziert und somit die Anzahl der priorisierten Gene angepasst werden (Abbildung 26).

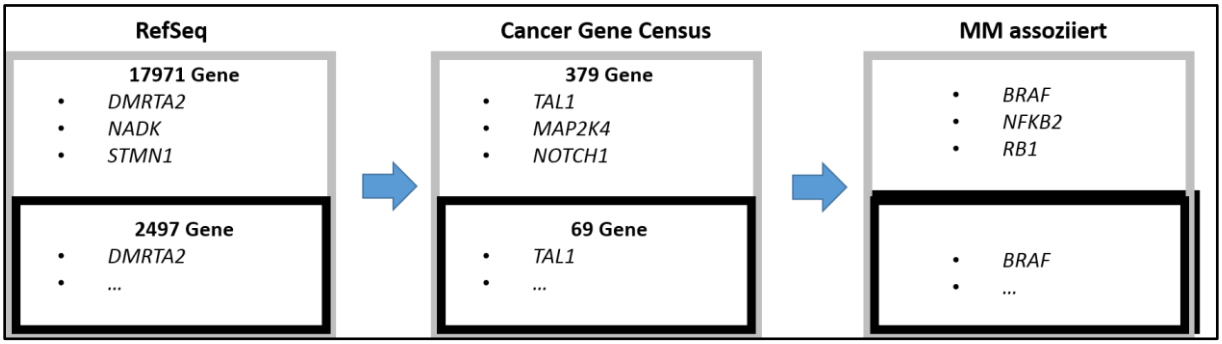


Abbildung 26: Schematische Darstellung der Annotation und Analyse von unbeachteten Regionen
Die identifizierten grauen und schwarzen Regionen werden zunächst annotiert und mit der Cancer Gene Census Datenbank abgeglichen. Somit können Gene priorisiert werden, zu denen niedrige oder keine Informationen aus den NGS-Daten gewonnen jedoch zur Kanzerogenese assoziiert werden. Des Weiteren können Entitätsspezifische Gene hervorgehoben und untersucht werden.

Für eine detaillierte Veranschaulichung der unterschiedlichen Klassen der unbeachteten Regionen, wurde das Tumorsuppressorgene *TAL1* in dem Datensatz selektiert und unter Verwendung des Softwaretools *Integrated Genomics Viewer* (*IGV*) bearbeitet [132; 133]. Die drei Klassen (weiße, graue und schwarze Regionen) sind hervorgehoben und geben Aufschluss über das Ausmaß an Informationsverlust zu *TAL1* im angegebenen Datensatz (Abbildung 27).

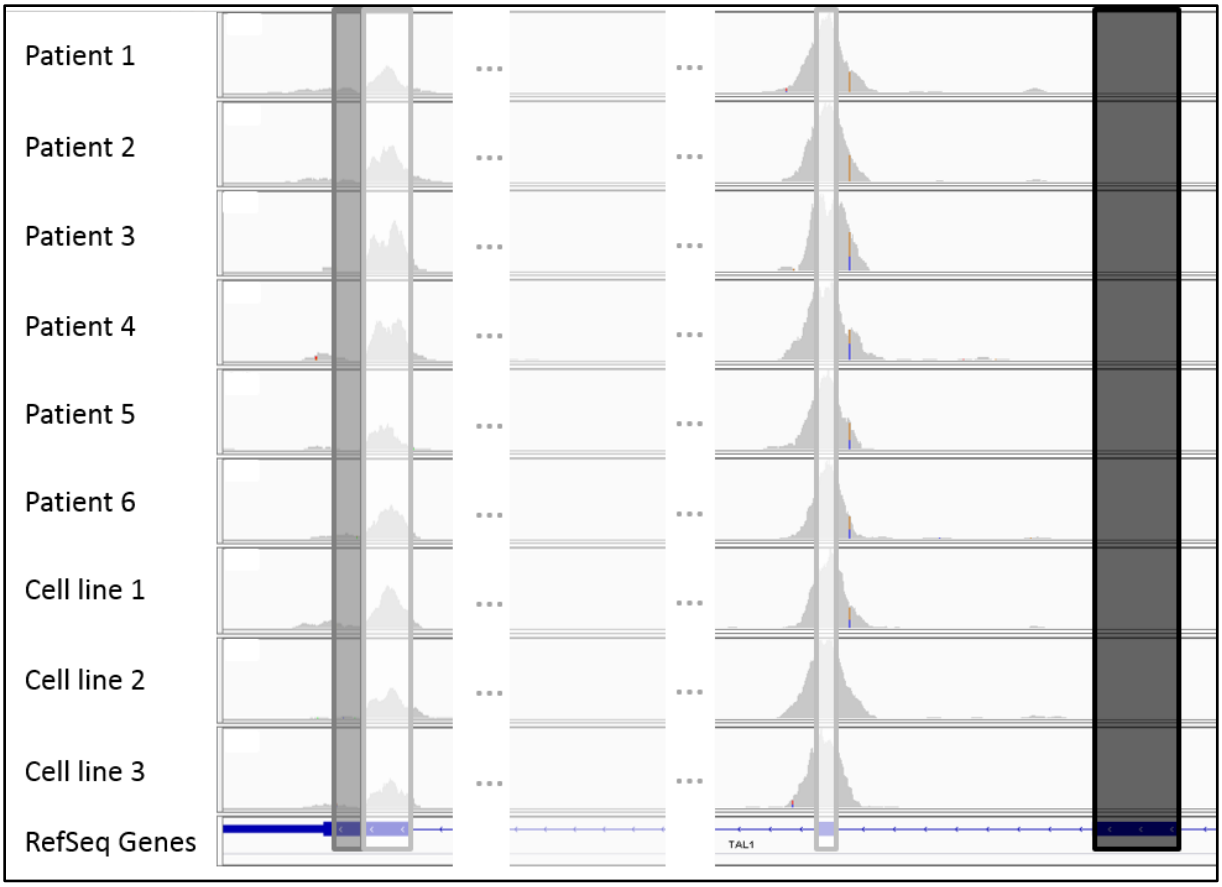


Abbildung 27: Schematische Darstellung abgedeckter Regionen in TAL1
Das Tumorsuppressorgene TAL1 enthält alle drei Arten der Projektweiten unbeachteten Regionen. Die Abbildung zeigt die weißen Regionen, welche alle der Mindestanforderung der Abdeckung in allen Proben entsprechen (n = 1). Die grauen Regionen repräsentieren die Bereiche, in denen diese Voraussetzung nicht in allen Proben gegeben sind, wie in diesem Beispiel bei Patient 5. Die schwarzen Regionen zeigen einen Bereich, der in keiner Probe ausreichend sequenziert wurde, weshalb es in dem gesamten NGS-Datensatz keine Informationen zu diesem Bereich gibt.

Mit den neuen Erkenntnissen zu den MM-Daten wurden weitere Projekte zur Analyse der RTK, Adhäsions, Downstream Gene und auch zu häufig mutierten Genen wie *DIS3* durchgeführt [122]. Hierzu kam die Amplikon Sequenzierung zum Einsatz, welche ideal war um eine tiefere Sequenzierung der Gene zu erreichen und die Probenanzahl zu erhöhen. Hierbei konnten mit dem 454 GS junior von Roche im Durchschnitt 100 – 700 Reads und mit dem MiSeq von Illumina im Durchschnitt 700 - 3000 Reads pro Base erreicht werden. Hinsichtlich der unbeachteten Regionen im WES-Datensatz, konnte das erste Exon des DIS3 Gen als nicht ausreichend sequenziert klassifiziert werden. Die erreichte *ØAbdeckung* von 57 Reads für diese Region war zwar für die Amplikon Sequenzierung eher dürftig, jedoch ausreichend um eine Analyse der Mutationen im ersten Exons des *DIS3* Gen effizient durchzuführen (Abbildung 28). Ebenso wurden weitere Regionen untersucht, die in den WES-Daten nicht genügend sequenziert und in den Amplikon-Daten ausreichend sequenziert wurden. Es konnte jedoch weder eine somatische noch eine Keimbahnmutation in diesen Regionen identifiziert werden.

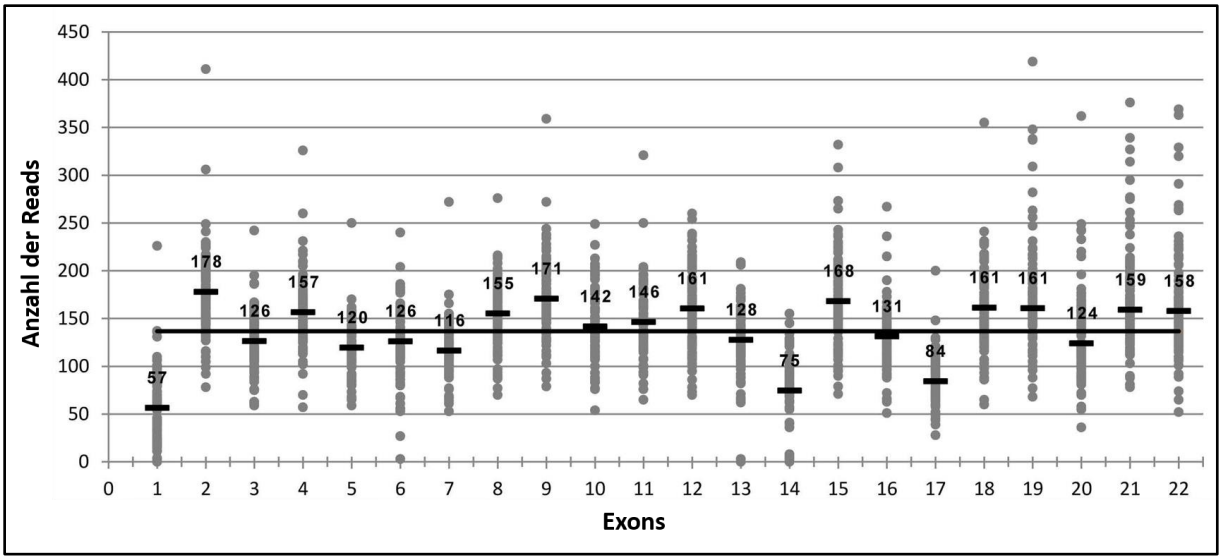


Abbildung 28: Durchschnittliche Abdeckung des Gens DIS3, Abbildung nach Weißbach et. al [122]
Die beobachteten *ØAbdeckungen* der insgesamt 22 Exons des sequenzierten DIS3-Transkripts (NM_014953.4/NP_055768.3) sind hier gegenübergestellt. Die Amplikon Sequenzierung wurde wie in Weißbach et. al beschrieben durchgeführt [122]. Die einzelnen *ØAbdeckungen* sind in der Abbildung hervorgehoben, wobei das erste Exon im Durschnitt eine Abdeckung von 57 Reads pro Base erreicht.

6.4.2 Ergebnisse zum Follikulären Lymphom

Die 11 Formalin-fixierten in Paraffin-eingebetteten FL Proben wurden einheitlich mit einer Readlänge von 72 Basen sequenziert. Die Proben wurden mit Hilfe des Exom-Extraktion-Kits *Nextera Rapid Capture Exome* für ihr Exom angereichert und danach mit dem Sequenzierer GA Ⅱx von Illumina intern von unserer Arbeitsgruppe sequenziert. Anschließend wurde der Datensatz mit der entwickelten Pipeline analysiert und dabei die Bewertung des *Alignments* erstellt (Tabelle 20). Die Ergebnisse basieren auf einem Vergleich von FL mit und ohne der Translokation t(14;18). Diese Translokation bewirkt wie bereits beschrieben in circa 85% der Fälle die Deregulation der Expression des BCL2 Gens (siehe 3.4.2), wodurch die Apoptose verhindert und die Entartung der Zellen begünstigt wird [26; 32; 33]. Die restlichen 15% der diagnostizierten FL Fälle enthalten nicht die t(14;18) Translokation, wobei keine weiteren wesentlichen Unterschiede zu den t(14;18)-positiven FL Fälle beobachtet werden. Der analysierte FL Datensatz besteht aus einem t(14;18)-positiven und zehn t(14;18)-negativen Proben. Die Identifizierung der Punktmutationen dient zur Untersuchung möglicher Alternativen, welche wie die Translokation t(14;18) zur FL-Pathogenese beitragen.

Die Bewertung des *Alignments* zeigt im Vergleich zu dem zweiten MM Datensatz eine Verschlechterung der prozessierten Daten. Dies ist eventuell auf eine schlechtere DNA-Qualität zurück zu führen, welche beispielsweise durch eine schlechte Lagerung der Proben entstanden ist. Zudem wurde die Aufbereitung der Proben mit dem Nextera-Kit von Illumina durchgeführt. Hierbei wurde eine viel geringere DNA-Menge und eine enzymatische statt einer mechanischen

Fragmentierung verwendet. Die Selektion bestimmter DNA-Fragmente in Abhängigkeit der Größe wurde durch die Nextera-Technik erschwert.

Tabelle 20: Bewertung des *Alignment* und Uniformität des FL Datensatz

Eigenschaften der Sequenzierung und Extraktion:	Readlänge	Exom-Extraktion-Kit	Gesamt Anzahl der Nukleotide	Gesamt Anzahl der Regionen
	72 bp	Nextera Rapid Capture	37.317.473 Nukleotide	212.158 Regionen

A: Alignment

Probe	Gesamt Anzahl der Reads	Alignierte Reads		Eindeutig alignierte Reads		Reads in Regionen des Extraktion-Kits		Ø Abdeckung
FL1269	136.203.464	109.121.150	80,12%	78.524.966	57,65%	45.676.783	58,17%	88,1
FL2162	137.281.006	110.637.500	80,59%	80.153.264	58,39%	47.525.131	59,29%	91,7
FL6128	123.503.540	101.388.500	82,09%	75.620.584	61,23%	43.312.576	57,28%	83,6
FL9782	124.059.380	108.474.976	87,44%	79.351.736	63,96%	49.294.783	62,12%	95,1
FL12333	131.821.964	117.638.998	89,24%	86.439.414	65,57%	51.718.539	59,83%	99,8
FL16865	153.843.426	125.591.798	81,64%	90.437.010	58,79%	52.799.369	58,38%	101,9
FL18053	141.818.152	123.540.706	87,11%	89.103.508	62,83%	52.916.055	59,39%	102,1
FL18701	136.311.146	120.401.540	88,33%	87.777.230	64,39%	52.223.435	59,50%	100,8
FL24332	135.086.714	110.774.108	82,00%	79.605.750	58,93%	48.342.872	60,73%	93,3
FL25221	126.821.704	112.464.386	88,68%	80.994.338	63,86%	51.293.899	63,33%	99,0
FL29758	169.441.518	153.438.578	90,56%	111.461.478	65,78%	64.377.161	57,76%	124,2

B: Uniformität

Probe	n ≥ 1		n ≥ 5		n ≥ 10		n ≥ 25		n ≥ 50	
FL1269	35837963	96,04%	31809403	85,24%	27749798	74,36%	20473810	54,86%	15662856	41,97%
FL2162	35832523	96,02%	31753028	85,09%	27683278	74,18%	20723989	55,53%	16216017	43,45%
FL6128	35484813	95,09%	30499849	81,73%	25889953	69,38%	18896863	50,64%	14772823	39,59%
FL9782	35009537	93,82%	29357822	78,67%	25445242	68,19%	20969179	56,19%	17433342	46,72%
FL12333	35270635	94,52%	29854324	80,00%	25691515	68,85%	20901352	56,01%	17542704	47,01%
FL16865	36027783	96,54%	32445569	86,94%	28643708	76,76%	21534070	57,71%	16700154	44,75%
FL18053	35270695	94,52%	29808966	79,88%	25556922	68,49%	20617139	55,25%	17397024	46,62%
FL18701	35321723	94,65%	29954106	80,27%	25781639	69,09%	20878012	55,95%	17524445	46,96%
FL24332	35968330	96,38%	32127389	86,09%	28102173	75,31%	20845944	55,86%	16032539	42,96%
FL25221	35261441	94,49%	29891323	80,10%	25738131	68,97%	20852706	55,88%	17447902	46,76%
FL29758	35701024	95,67%	31258777	83,76%	27112220	72,65%	21767786	58,33%	18621744	49,90%

In Zusammenarbeit mit dem ICGC konnte der FL Datensatz um 6 weitere t(14;18)-negative FL und 33 t(14;18)-positive FL vergrößert werden. Hierzu wurden die unterschiedlichen Mutationslisten mit dem *Genemapping* Skript auf Basis der Gen Namen vereinheitlicht und mit Hilfe des *Heatmap* Skript die jeweiligen Gruppen gegenübergestellt. Es war jedoch zu beachten, dass die FL Proben des ICGC-Datensatzes mittels WGS sequenziert und mit einer internen Pipeline vom ICGC verarbeitet wurden. Um die Vergleichbarkeit zu wahren, wurden diese NGS-Daten auf die Regionen reduziert, welche bei der WES in dem verwendeten Exom-Extraktion-Kit (*Nextera Rapid Capture Exome*) beschrieben werden. Hierfür wurden nur die Mutationen aus den VCF-Dateien in den vorgegebenen Regionen des *Nextera Rapid Capture Exome*-Kit in die Analyse aufgenommen.

Eine ausführliche Analyse der unbeachteten Regionen konnten nicht durchgeführt werden, da

wir keinen Zugriff auf die ICGC-Rohdaten bekamen. Es konnten nur die VCF-Dateien, jedoch nicht die benötigten SAM/BAM-Dateien vom Server transferiert werden. Durch die Erweiterung beziehungsweise Anpassung des *Heatmap* Skripts wurden die ICGC-VCF-Dateien in die Analyse aufgenommen.

6.4.3 Ergebnisse zum Burkitt Lymphom

Zu den NGS-Daten der BL Proben wurde der Service von CeGaT genutzt. Hierfür wurden zunächst die Proben bestehend aus 8 Tumorzelllinien mit korrespondierenden Normalproben, mit der SOLiD-Technologie sequenziert. Nach der erschwerten Verarbeitung der Daten bezüglich der Analyse, konnten die identifizierten Kandidaten nur mit einer Validierungsrate von etwa 40% bestätigt werden. Zudem reduzierte die Sequenzierung mittels der *single read*-Applikation die Effizienz der Alignierung und erhöhte zugleich die Rate der duplizierten Reads. Dies führte zu einem erhöhten Ausfall an NGS-Daten, da die duplizierten Reads bei der Bearbeitung standardmäßig entfernt werden. In Folge dessen wurde eine geringere Ergebnismenge an Kandidaten identifiziert, als die von der Illumina Technologie gewohnte Menge. Nach weiterführenden Analysen und stringenteren Filterstrategien zur Verbesserung der Validierungsrate, zeigte sich das nur vereinzelt Mutationen identifiziert werden konnten. Die resultierende VCF-Datei beinhaltet 9 Mutationen. Die geringe Menge an identifizierten Punktmutationen war hinsichtlich eines Screenings nicht ausreichend, weshalb CeGaT die Sequenzierung mit der Illumina Plattform wiederholte. Die Anreicherung des Exoms wurde mit dem Exom-Extraktion-Kit *SureSelect Human All Exon V5+UTRs* von Agilent durchgeführt und die Proben mit dem HiSeq2000 sequenziert. Die deutliche Verbesserung durch die Illumina Plattform, der längeren Reads und der effizienteren Exom-Extraktion sind in den Bewertungen des *Alignments* zu erkennen (Tabelle 21).

Tabelle 21: Bewertung des *Alignment* und Uniformität des BL Datensatz

Eigenschaften der Sequenzierung und Extraktion:	Readlänge	Exom-Extraktion-Kit	Gesamt Anzahl der Nukleotide	Gesamt Anzahl der Regionen
	101 bp	SureSelect Human V5+UTRs	74.569.526 Nukleotide	286.754 Regionen

A: Alignment

Probe	Gesamt Anzahl der Reads	Alignierte Reads		Eindeutig alignierte Reads		Reads in Regionen des Extraktion-Kits		Ø Abdeckung
BL36	108.099.294	106.959.189	98,95%	97.124.095	89,85%	79.245.394	74,09%	107,3
BL37	196.609.032	194.792.812	99,08%	172.068.812	87,52%	146.478.230	75,20%	198,4
BL41	255.424.296	253.210.425	99,13%	209.552.355	82,04%	193.507.495	76,42%	262,1
BL64	218.280.938	216.355.802	99,12%	184.513.204	84,53%	165.136.905	76,33%	223,7
BL67	207.935.008	206.048.864	99,09%	171.813.070	82,63%	156.124.850	75,77%	211,5
BL65	228.307.286	225.433.591	98,74%	161.701.234	70,83%	170.192.043	75,50%	230,5
BL74	230.228.388	227.311.290	98,73%	171.432.070	74,46%	170.734.899	75,11%	231,3
BL18	247.072.486	244.043.275	98,77%	174.078.627	70,46%	179.654.100	73,62%	243,3

B: Uniformität

Probe	n ≥ 1		n ≥ 5		n ≥ 10		n ≥ 25		n ≥ 50	
BL36	74.415.268	99,79%	74.123.558	99,40%	73.565.153	98,65%	69.627.702	93,37%	55.164.198	73,98%
BL37	74.533.613	99,95%	74.417.069	99,80%	74.226.554	99,54%	73.208.375	98,17%	69.110.190	92,68%
BL41	74.528.353	99,94%	74.444.606	99,83%	74.316.808	99,66%	73.662.561	98,78%	71.194.462	95,47%
BL64	74.533.214	99,95%	74.441.480	99,83%	74.292.586	99,63%	73.508.585	98,58%	70.455.082	94,48%
BL67	74.532.840	99,95%	74.418.345	99,80%	74.220.455	99,53%	73.113.297	98,05%	68.729.636	92,17%
BL65	74.455.812	99,85%	74.362.362	99,72%	74.226.829	99,54%	73.481.268	98,54%	70.539.186	94,60%
BL74	74.464.758	99,86%	74.375.416	99,74%	74.238.316	99,56%	73.511.694	98,58%	70.685.687	94,79%
BL18	74.545.816	99,97%	74.477.464	99,88%	74.365.786	99,73%	73.764.232	98,92%	71.363.168	95,70%

Eine biologische Interpretation des BL-Datensatzes ist noch nicht abgeschlossen, jedoch konnte eine erhöhte Häufigkeit wiederkehrender Mutationen beobachtet werden. Die wiederkehrenden Mutationen wurden öfters als bei den MM Datensätzen in unterschiedlichen Proben an der gleichen Position identifiziert. Dies deutet daraufhin, dass es sich beim BL im Vergleich zum MM um eine weniger heterogene Erkrankung handelt. Auch in Zusammenarbeit mit dem ICGC konnten durch Sequenzierungen von Exomen und Genomen des BL und anschließenden funktionellen Analysen neue molekulare Erkenntnisse in Bezug auf die BL-Pathogenese gewonnen werden [45].

Ein weiteres Projekt, das im Rahmen dieser Arbeit bearbeitet wurde umfasste die Differenzierung von nicht klassifizierbaren aggressiven B-Zell-Lymphomen. Es existiert eine Grauzone zwischen den Entitäten BL und DLBZL, welche intermediäre Merkmale bei der Diagnose hinsichtlich Morphologie, Immunphänotypisierung und Expressionsmuster aufweist [131]. Hierfür wurde die Verteilung der Mutationen in spezifischen Genen als alternative Klassifizierung untersucht [131]. Hierzu wurde zunächst ein Screening des WES-BL-Datensatzes erstellt und untersucht. Des Weiteren wurde für die Klassifizierung ein Datensatz bestehend aus 31 BL, 24 intermediären Lymphome und 53 DLBZL analysiert, welcher mit der Amplikon Sequenzierung generiert wurde. Die Gene und Regionen für die Amplikon Sequenzierung wurden unter Berücksichtigung bereits veröffentlichter Ergebnisse und des

zuvor analysierten BL-WES-Datensatzes selektiert. Die BL-spezifischen Gene waren hierbei *ID3*, *TCF3*, *CCDN3* und *MYC*. Die DLBZL-spezifischen Gene waren *BCL2*, *EZH2*, *CREBBP*, *EP300*, *MEF2B* und *SGK1*. Die Analyse von identifizierten Mutationen in den 10 selektierten Genen konnte nicht zu einer eindeutigen molekularen Einordnung der Proben in die spezifischen BL und DLBZL sowie der intermediären Gruppen führen[131].

6.4.4 Komparative Analyse der unbeachteten Regionen

Um das Ausmaß an unbeachteten Regionen weiter zu untersuchen wurde eine komparative Analyse zu Datensätze aus unterschiedlichen Projekten durchgeführt. Hierfür wurden die 8 BL Tumorzelllinien und die 20 MM-Tumorproben des zweiten analysierten Datensatzes genutzt. Dies verschaffte einen Überblick wie sich die Detektion von unbeachteten Regionen verschiedener NGS-Datensätze verhält. Die Bearbeitung und Bewertung des *Alignments* der einzelnen Datensätze wurde schon in den vorherigen Abschnitten beschrieben (siehe 6.4.1 und 6.4.3). Ein geringer Unterschied zwischen den Bearbeitungsschritten ist die Aufbereitung der genomischen DNA der MM Proben mittels WGA und der Aufbereitung der BL Proben mit Hilfe des Exom-Extraktion-Kit *SureSelect Human All Exon V5+UTRs* von Agilent. Im Übrigen eignen sich diese Datensätze ideal für eine Gegenüberstellung der unbeachteten Regionen, da die einzelnen Qualitäten der Daten und die Bewertungen der *Alignments* beim BL (Tabelle 21) und dem MM (Tabelle 16) vergleichbar hoch sind. Die verwendete Readlänge von 101 Bp und des HiSeq2500 wurde zur Sequenzierung der Daten genutzt.

Zunächst wurden die unbeachteten Regionen der BL Daten (Tabelle 22A) und der MM Daten (Tabelle 22B) in Abhängigkeit des Faktors 1 in den einzelnen Projekten differenziert.

Tabelle 22: Ergebnisse der unbeachteten Regionen im BL und dem zweiten MM Datensatz

A	Weiße Region	Graue Region	Schwarze Region
Basen [MBp]	74	0,235	0,0002
Regionen [#]	286645	6571	34
Gene [#]	23820	1231	22

B	Weiße Region	Graue Region	Schwarze Region
Basen [MBp]	44	6,43	0,0005
Regionen [#]	292893	173765	31
Gene [#]	22388	13541	20

C	BL ≤ 10	MM ≤ 10	BL und MM ≤ 10
Basen [MBp]	0,579	0,254	0,185
Regionen [#]	2582	1355	944
Gene [#]	1924	1076	780

Des Weiteren wurde die aus der Literatur bekannte minimale Abdeckung zur Identifizierung heterozygoter Mutationen berücksichtigt. Dies führte zur Erhöhung des Faktors zur Berechnung

der *%Abdeckung* auf den Wert 10. Die jeweiligen prozentualen Anteile der weißen Regionen der Proben können für dieses Beispiel aus den Tabellen zur Bewertung des *Alignments* abgelesen werden (Tabelle 16 und Tabelle 21). Die daraus resultierenden schwarzen Regionen entsprechen der jeweiligen restlichen Prozente. Werden diese schwarzen Regionen der Entitäten unter Berücksichtigung des Faktors 10 aus den einzelnen Projekten zusammengefasst, dann ist ersichtlich das bis zu 185000 Basen weder im BL- noch im MM Datensatz effizient sequenziert wurden (Tabelle 22C).

Für die Analyse der unbeachteten Regionen ist zunächst die erneute Sequenzierung der schwarzen Regionen relevant. Es besteht die Möglichkeit, dass Mutationen in diesen Regionen enthalten sind, die bisher nicht identifiziert werden konnten. Die Anzahl der möglichen fehlenden Mutationskandidaten kann mit Hilfe der Menge an identifizierten unbeachteten Regionen und der beobachteten tumorspezifischen Mutationsraten postuliert werden. Im MM und BL konnten bisher tumorspezifische Mutationsraten von 0,3 bis zu 12 Punktmutationen pro 1 Million Nukleotide beobachtet werden [30; 31; 47; 48]. Im Durchschnitt konnten etwa 4 Punktmutationen pro 1 Million Nukleotide in beiden Entitäten beobachtet werden. Hieraus werden unter Berücksichtigung der 0,579 Millionen Nukleotide die in allen BL-Proben nicht sequenziert wurden, bis zu zwei tumorspezifische Mutationen pro Probe postuliert. Diese Mutationen werden in der standardmäßigen Analyse nicht beachtet. In den MM-Daten konnten 0,254 Millionen Nukleotide nicht sequenziert werden. Somit kann eine tumorspezifische Mutation pro Probe erwartet werden, die in den MM-Proben nicht identifiziert werden konnte. Eine erneute WES Sequenzierung aller Proben wäre kostenintensiv und eine Garantie, dass die zuvor nicht abgedeckten Regionen diesmal sequenziert werden, ist nicht gegeben. Die Alternativen sind eine Amplikon- oder Sanger-Sequenzierung der einzelnen Regionen. Für einige der MM Proben sind Amplikon-Sequenzierungsdaten vorhanden, die zur Untersuchung von RTK, Adhäsion und *dowstream* Effektoren erstellt wurden. Ein Vergleich mit den Regionen dieser spezifischen Gene und den identifizierten schwarzen Regionen in den WES Daten, zeigte das einzelne Regionen überlappen. In den Amplikon-Sequenzdaten konnten jedoch keine Mutationen in den zuvor nicht abgedeckten Regionen identifiziert werden. Eine gesamte Amplikon Sequenzierung aller MM- und BL Proben unter Berücksichtigung der schwarzen Regionen ist aus Kostengründen nicht durchführbar. Um die Kosten für eine Sanger-Sequenzierung zu reduzieren wurden einzelne schwarze Regionen selektiert, welche zunächst bei der Integration der schwarzen Regionen unter Berücksichtigung des Faktors 10 aus beiden Datensätze entstanden (Tabelle 22C). Des Weiteren wurden diese Regionen annotiert, um relevante Gene zu priorisieren. Von den insgesamt 780 betroffenen Genen, konnten bis zu 20

mit der Kanzerogenese assoziierte Gene durch den Datenabgleich mit COSMIC identifiziert werden (Abbildung 29). Eine Assoziation der Kandidaten zu den Entitäten wurde anschließend untersucht und einige davon beispielhaft hervorgehoben (Abbildung 29).

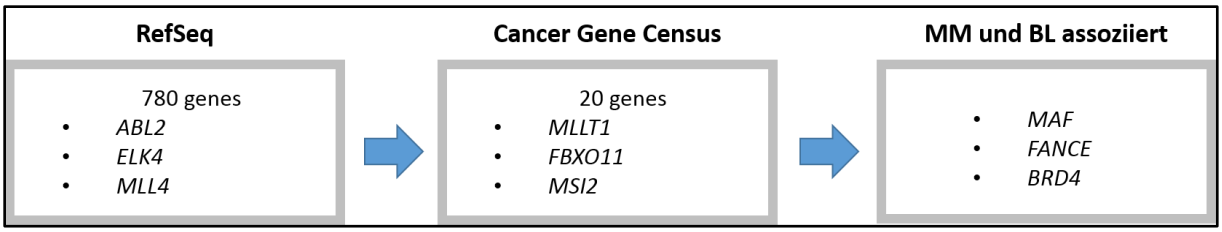


Abbildung 29: Annotation unbeachteter Regionen und Datenbankabgleich mit COSMIC
Die identifizierten grauen Regionen im zusammengefassten BL-MM-Datensatz wurden annotiert und mit der Cancer Gene Census Datenbank abgeglichen. Aus den identifizierten Kandidaten wurden die BL-relevanten Gene wie . *MAF*, *USP6*, *BRD4* und *HOXA13* für die Sanger-Validierung selektiert.

Ein Vorteil der BL-Daten ist, dass diese aus Tumorzelllinien generiert wurden. Somit ist eine Limitierung durch Probenmaterial für die Sanger-Sequenzierung nicht gegeben, wie es für die Mehrzahl der MM-Proben der Fall war. Die Gene *MAF*, *USP6*, *BRD4* und *HOXA13* wurden selektiert, um jeweils eine Sanger-Sequenzierung mit den BL Zelllinien durchzuführen. Hierzu konnten jedoch während der Aufbereitung mittels PCR keine eindeutigen Produkte der nicht abgedeckten Regionen der Kandidaten *MAF* und *HOXA13* erstellen werden. Dies ist ein Indiz dafür, dass die Eindeutigkeit der Sequenz in diesen Regionen nicht gegeben ist und somit die benötigte PCR beeinflusst. Ein weiterer Grund könnte eine fortgeschrittene genomische Instabilität in den BL-Zelllinien sein. Beides könnte die Effizienz der Exom-Extraktion während der Generierung der Sequenzbibliothek für die WES beeinflussen und zur Entstehung der schwarzen Regionen führen. Zu den Regionen der Gene *USP6* und *BRD4* konnte die PCR und die anschließende Sanger-Sequenzierung durchgeführt, jedoch keine Mutationen in den schwarzen Regionen identifiziert werden.

Zusammenfassend wurden in den NGS-Daten der Entitäten BL, MM und FL jeweils die unbeachteten Regionen analysiert. Dabei konnte beobachtet werden, dass die mit der Zeit entstandenen Neuerungen sowohl der Sequenzierungstechnologien als auch der Exom-Extraktion-Kits, die Entstehung von unbeachteten Regionen reduzieren. Die Relevanz der unbeachteten Regionen nimmt somit stetig ab. Jedoch ist besonders die Beachtung dieser Regionen bei der komparative Analyse von NGS-Daten eines gesamten Projekts essentiell, wenn unterschiedliche Exom-Extraktion-Kits, unterschiedliche Sequenzierer oder auch mehrere DNA-Proben (sowie Normal- als auch Tumorproben) des Patienten oder Zelllinie verwendet werden. Die Detektion und Analyse der abgedeckten und nicht-abgedeckten

Regionen zeigt einen Überblick wie viele Basen und wie effizient die Proben sequenziert wurden. Auch unterschiedliche Algorithmen zur Identifizierung von somatischen Mutationen haben Einfluss auf die Berücksichtigung bestimmter Regionen.

Allein die Verwendung von unterschiedlichen Exom-Extraktion-Kits oder die Integration von älteren Datensätzen die mit Hilfe anderer Exom-Extraktions-Kits generiert werden führen zu einer neuen Zusammensetzung unbeachteten Regionen. Dieser Einfluss unterschiedlicher Exom-Extraktion-Kits wurde mit Hilfe aller verfügbaren MM Daten untersucht. Hierzu wurden die Regionen der verwendeten Exom-Extraktion-Kits Nimblegen v1 (35 Mbp), Nimblegen v2 (44 Mbp) und SeqCap Agilent v5 (50 Mbp) untersucht (Abbildung 30). Das Venn-Diagramm zeigt, dass nur 26 Mbp effizient in einer komparativen Analyse Beachtung fanden, wobei eine ausreichende Abdeckung in allen Proben zu diesen 26 Mbp vorausgesetzt wurde. Dieses Ergebnis zeigt wie stark die unbeachteten Regionen vertreten waren und dies nur aufgrund der vordefinierten Regionen in den verwendeten Exom-Extraktion-Kits. Eine effiziente Identifizierung der Mutationen unter Berücksichtigung aller MM-Proben und verwendeten Exome-Extraktion-Kits kann nur für die überlappenden 26 Mbp durchgeführt werden. Weshalb eine Integration oder die Verwendung eines neuen Exom-Extraktion-Kits immer unter den Aspekt neu entstehender unbeachteter Regionen diskutiert werden muss.

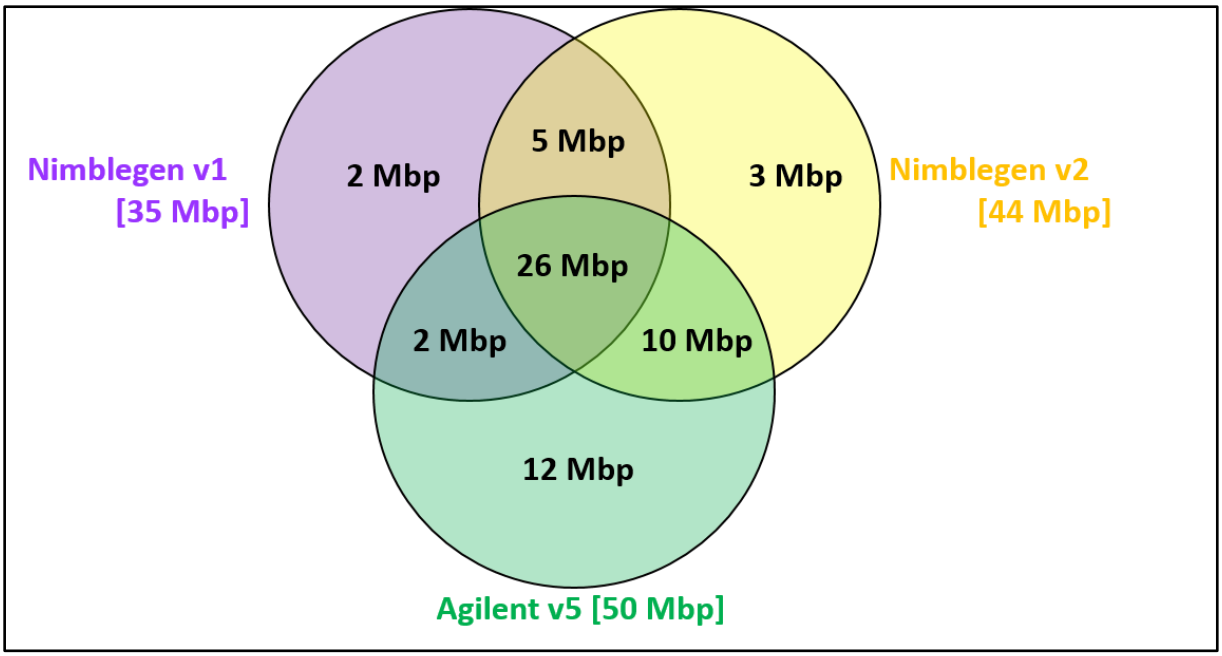


Abbildung 30: Venn-Diagramm der verwendeten Exom-Extraktion-Kits
In dieser Darstellung sind die Exom Regionen des Referenzgenoms in Abhängigkeit der Exom-Extraktion-Kits hervorgehoben. Die jeweiligen spezifischen Regionen der Kits, sowie die gesamte überlappende Region ist ersichtlich und zeigt wie viele Basen verloren, beziehungsweise sequenziert werden, sobald während eines Projektes das Kit gewechselt oder unterschiedlich prozessierte NGS-Daten verglichen werden.

7 Diskussion

Die Identifizierung und Klassifizierung von Mutationen aus NGS-Daten setzt eine Verarbeitung und mehrere Analyseschritte voraus. Hierzu können die Identifizierung von Basen, die Alignierung der Reads zum Referenzgenom und die Identifizierung von Mutationen als primäre Analysen zusammengefasst werden. Die Annotation, Filterung, funktionelle Vorhersagen und die biologische Interpretation von Mutationen können als sekundären Analysen definiert werden.

7.1 Primäre Analyse

Die primäre Analyse von NGS-Daten gibt Aufschluss über die Existenz von Mutationen in den sequenzierten Proben. Die im Rahmen dieser Arbeit entwickelte Pipeline beinhaltet die Analyseschritte wie die Generierung des *Alignments* und die Identifizierung der Mutationen. Hierzu wurden die implementierten Softwaretools unter Berücksichtigung der dargestellten Ergebnisse selektiert (Abbildung 20, Tabelle 12) [124]. Alternative Softwareprogramme zur Identifizierung von Basen sind auf die unterschiedlichen Sequenzierungsplattformen abgestimmt, da die zugrunde liegende Technik der einzelnen Plattformen jeweils spezifische Störfaktoren hervorbringt. Für Illumina Daten können wie gezeigt, *Alt-Cyclic*, *Swift*, *Rolexa*, *Ibis*, *naiveBayesCall* und die *RTA* Pipeline verwendet werden. Wobei *Alt-Cyclic*, *Swift*, *Rolexa*, *Ibis*, *naiveBayesCall* und das Fehlerkorrekturprogramm *ECHO* sich nicht als effiziente Alternativen herausstellten. Vor allem die Zeit intensiven Berechnungen von *BayesCall* und *ECHO* stehen in keinem Verhältnis zu den geringfügigen Verbesserungen der Datenqualität. Die von Illumina entwickelten Softwaretools unterlaufen genauso einer stetigen Entwicklung wie die Hardware und können neben kurzen Rechenlaufzeiten auch sehr gute Ergebnisse liefern. Eine Speicherung der prozessierten Daten und eine erneute Identifizierung der Basen mit den neueren Versionen kann die Qualität der Daten verbessern. Dieser Effekt wurde in den Ergebnissen aufgezeigt (Tabelle 12).

Jedoch unterliegen die *RTA* Pipeline und das Softwaretool *OLB*, welche zur Identifizierung von Basen genutzt werden können, kommerziellen Lizenzen von Illumina. Somit ist aus rechtlichen Gründen diese Verarbeitung der Daten nicht in der entwickelten Pipeline implementiert, sondern wird vorausgesetzt. Hierzu bleibt die Auswahl des Softwaretools zur Identifizierung der Basen dem Anwender überlassen, wobei das entscheidende Kriterium die Generierung der FASTQ-Dateien ist. Mit den hier angeführten Ergebnissen und Vorteilen ist die Generierung qualitativ hochwertiger FASTQ-Dateien mit der Illumina Software die beste Wahl. Diese Dateien werden als Eingabe für die Pipeline benötigt, um die Analyse beginnen zu können.

Ebenso werden standardmäßig nur die FASTQ-Dateien von Sequenzierungszentren als Ergebnis übermittelt, wodurch die Pipeline für die Analyse extern generierter NGS-Daten geeignet ist.

Die Qualität des anschließenden *Alignments* wird durch die Qualität der identifizierten Basen in den generierten FASTQ-Dateien beeinflusst. Bei der gezielten Resequenzierung werden die Reads dem Referenzgenom zugeordnet, wobei weniger Fehler bei der Identifizierung der Basen zu einer eindeutigeren Zuordnung der Reads führen kann. Wie bereits im Vorfeld beschrieben, kann mit der Formel (4) und unter den angegebenen Annahmen eine benötigte Länge von mindestens 16 Basen berechnet werden, um eine eindeutige Zuordnung der Reads im Referenzgenom zu ermöglichen. Eine einzelne fehlerhafte Base würde hierbei zu einer falschen Zuordnung führen. Ein weiterer entscheidender Faktor bei der Zuordnung der Reads ist die Sequenzkomposition der jeweiligen Region. Beispielsweise können Reads im Referenzgenom den repetitiven Regionen nicht eindeutig zugeordnet werden, wenn die Länge der Reads nicht die Länge der repetitiven Region übersteigt. Derzeit sind die meist genutzten Programme *Bowtie* und *BWA*, welche zur Generierung von *Alignments* für die Resequenzierungen verwendet wurden.
Eine Debatte über die Entscheidung welches Programm sich für die Zuordnung der Reads im Referenzgenom am besten eignet, ist stetig im Gange [87; 105; 107; 134]. Derzeit werden die Ergebnisse neu entwickelter Programme stets mit den Ergebnissen der meist verbreiteten Aligner wie *Bowtie* und vor allem *BWA* verglichen. Die dabei verwendeten Methoden zur Bewertung der Ergebnisse, wie die Berechnungen von Sensitivität und Spezifität, sowie den *true positive*-, *true negative*-, *false positive*- und *false negative*- Werten sind jedoch umstritten. Da sowohl die Nutzung von simulierten als auch von reellen Sequenzierungsdaten mit oder ohne induzierten Fehlern Einfluss auf die Ergebnisse haben, wird die Entscheidung zu Gunsten eines Programms erschwert. Zudem werden die generierten *Alignments* mit Hilfe der implementierten Algorithmen unterschiedlich gewertet und führen somit in der darauf folgenden Analyse wie der Identifizierung von Punktmutationen auch zu unterschiedlichen Ergebnissen. In wieweit sich die Ergebnisse der Mutationslisten in Abhängigkeit von der verwendeten Pipelines unterscheiden kann, wurde in dieser Arbeit erläutert (Kapitel 3.6.4) [118].
Zu den bereits beschriebenen Applikationen wie beispielsweise RNA-Seq, WES und WGS können bestimmte *Alignmenttools* die Zuordnung der Reads effizienter und schneller bewerkstelligen als andere [87]. Beispielsweise wäre das *Alignment* mittels *Bowtie* oder *Bowtie 2* von RNA-Seq-Daten am effizientesten. Zwar leidet die Sensitivität unter dem

Geschwindigkeitsvorteil, jedoch wird die benötigte Quantität der zugeordneten Reads, welche für die Bewertung von RNA-Seq-Daten essentiell ist, in den Vordergrund gestellt [134-136]. Eine Erweiterung von *Bowtie* ist *TopHat*, dieses Softwareprogramm inkludiert alternative Spleißvarianten bei der Generierung des *Alignments*. Somit können Reads in das *Alignment* mit aufgenommen werden, welche zu einem gewissen Anteil in unterschiedlichen Exons lokalisiert und ansonsten keine Berücksichtigung in anderen Algorithmen finden [137]. Zur Generierung des *Alignments* von WES-Daten eignet sich beispielsweise *NovoAlign*, dieser Aligner erreicht die höchste Präzision beim *Alignment* [138]. Der Nachteil dieses Hash-Tabellen basierenden Aligner ist die Geschwindigkeit. Als Alternative eignet sich *BWA*, nahezu gleichwertige Ergebnisse können bei weitaus kürzeren Rechenzeiten erreicht werden. Des Weiteren wird *BWA* stets weiter entwickelt, hinsichtlich Lücken im Referenzgenom oder in den sequenzierten Daten sowie beispielsweise die Handhabung von chimärischen Reads. Aus diesen Gründen wurde *BWA* in die entwickelte Pipeline integriert. Die neueste Version von *BWA* ist *BWA-MEM* und speziell für Reads mit einer Länge ab 70 Basen bis zu mehreren Megabasen ausgelegt. Die Implementierung der aktuellsten Version von *BWA-MEM* ermöglicht somit auch in Zukunft die Generierung hoch qualitativer Ergebnisse, da die stetige Entwicklung der NGS-Technologien mit immer länger prozessierbaren Reads keine Herausforderung stellen sollte. Entsprechend der veröffentlichten Expertenmeinungen [87; 103; 105], konnte auch in dieser Arbeit demonstriert werden, dass *BWA* ein sehr ausgewogenes und effizientes Programm im Bezug zur Geschwindigkeit, Nutzung des Arbeitsspeichers, Sensitivität und Spezifität betrifft. Zudem wird für die Nutzung zur Identifizierung der Mutationen mit Hilfe des Softwaretools *GATK*, das Programm *BWA* für die Generierung des *Alignments* empfohlen, da sämtliche darauf aufbauenden Analysen auf das Alignment mit *BWA* abgestimmt sind. Auch aus diesem Grund wurde *BWA* in die Pipeline implementiert.

Die Qualität der FASTQ-Dateien beeinflusst die Qualität des *Alignments* und die Identifizierung der Mutationen wird infolgedessen deshalb von beiden Qualitäten bestimmt. Somit haben die Identifizierung der Basen und auch die Zuordnung der Reads einen Einfluss auf die Ergebnisse der identifizierten Mutationen. Zur Verbesserung der NGS-Daten von Illumina kann zuvor der *Best Practices-Workflow* von *GATK* genutzt werden. Hiermit können falsch positive Mutationen reduziert werden, indem wie bereits beschrieben Faktoren wie Indels und überbewertete Signalintensitäten, welche zu falschen Basenqualitäten führen, in den SAM-Dateien berücksichtigt werden [109]. Wie gezeigt wurde ist die Abdeckung des Referenzgenoms ein wesentlicher Faktor um Mutationen effizient identifizieren und differenzieren zu können. Die erforderte Mindestabdeckung für die effiziente Identifizierung

von homozygoten oder heterozygoten Mutationen konnte jeweils auf 3 Reads oder 13 Reads festgelegt werden [110; 111].

Die Softwaretools für die Identifizierung von Mutationen können neben den implementierten Algorithmus in zwei Klassen unterteilt werden. Programme wie *VarScan*, *MuTect* und *SomaticSniper* bilden hierbei eine Klasse, welche eine Schnittmenge der Mutationen der sequenzierten Probenpaare erstellen [115-117]. Hierbei werden Kandidaten aus den Tumorproben selektiert, unter Berücksichtigung dass sowohl in der normalen als auch in der Tumorprobe vordefinierte Abdeckungen für die entsprechende Base erreicht werden. Diese simultane Betrachtung der Kandidaten reduziert die Ergebnisliste und wird dadurch als stringenter erachtet. Zugleich ist die Konfidenz der Existenz der Kandidaten höher, weil die Base sowohl in der normalen als auch in der Tumorprobe sequenziert wurde. Jedoch gehen bereits identifizierte Mutationen der Tumorprobe bei der simultanen Analyse auf Grund der Filterung verloren.

Die andere Klasse beinhaltet Softwaretools wie *GATK*, *SOAPsnp*, *SVNer* und *SAMtools*. Diese Programme können genutzt werden, um die Proben jeweils einzeln zu analysieren [108; 109; 113; 114]. Hierbei werden die Kandidaten in den jeweiligen Proben identifiziert und anschließend die Kandidaten der korrespondierenden Normalprobe von den Kandidaten der Tumorprobe subtrahiert. Die Analyse der jeweils einzelnen Proben mit anschließender Subtraktion generiert mehr Kandidaten, da identifizierte Mutationen in der Tumorprobe ohne Informationen aus der korrespondierenden Normalprobe zunächst nicht aus der Liste entfernt werden. Diese Vorgehensweise impliziert zwar Kandidaten, die auch in der Normalprobe vorhanden sein können, jedoch werden dadurch auch potenziell relevante Mutationen nicht aus der Liste entfernt. Zudem können Proben ohne korrespondierende Normalprobe ohne weiteren Aufwand in die Analyse aufgenommen werden.

Wie bereits beschrieben überschneiden sich die Ergebnisse unterschiedlicher Pipelines nur zu einem gewissen Grad (Abbildung 16 und Abbildung 21), wobei sich die Validierungsraten Softwarespezifischer Kandidaten nicht wesentlich unterscheiden [118]. Hieraus entstand eine weitere Analysestrategie und zwar das Zusammenführen der Ergebnisse mehrerer Pipelines. Die Sensitivität und Spezifität des gesamten Ergebnisliste kann somit zum Teil erhöht werden, jedoch entstehen hierbei extrem viele falsch positive Kandidaten [119]. Aus diesem Grund haben wir diese Strategie nicht weiter verfolgt.

Die simultane Analyse der Probenpaare ist abhängig von der Effizienz der Sequenzierung beider Proben. Hierzu konnte gezeigt werden, dass die Anzahl an unbeachteten Regionen steigt, wenn mehrere Sequenzierungen mit einander verglichen werden. Die unbeachteten Regionen

werden durch die limitierenden Parameter der simultanen Analyse definiert und addieren sich für ein Probenpaar, da eine Base nicht mehr beachtet wird sobald die erforderte Abdeckung dieser in der normalen als auch in der Tumorprobe nicht erreicht wurde. Dies ist für den Benutzer solcher Softwaretools nicht klar ersichtlich und der Eindruck entsteht alle somatischen Mutationen mit dem verwendeten Programm identifiziert zu haben. Während die Analyse basierend auf die Subtraktion der Kandidaten in erster Linie nur von der Abdeckung der Tumorprobe abhängt. Des Weiteren bestimmt die Effizienz der Sequenzierung der Normalprobe inwieweit Probenspezifische Mutationen herausgefiltert werden können und beeinflusst indirekt die Anzahl der falsch positiven Kandidaten. In der Gegenüberstellung der Pipelines von Illumina, Softgenetics, *MAQ* und *BWA+GATK* konnte die Kombination aus *BWA* und *GATK* mit den besten Ergebnissen und einer niedrigen Rechenzeit überzeugen. Die Abhängigkeit der genutzten Softwaretools zur Identifizierung der Basen und dem *Alignment* wurden zur Generierung des Vergleichs der Pipelines stets berücksichtigt. Hierzu wurden im Vergleich sowohl für die kommerziellen Pipelines von Illumina und Softgenetics als auch für die Pipeline *MAQ* und die Kombination aus *BWA* und *GATK*, die jeweiligen empfohlenen Softwaretools konsistent genutzt. Unter Berücksichtigung unserer Ergebnisse und aufgrund der bisher bekannten Erfahrungswerte ist die Wahl des *Alignment-Tools BWA* und des Mutationsidentifizierungstools *GATK* für die Identifizierung von Mutationen am effizientesten (Abbildung 21). Diese Beobachtungen werden durch unsere guten Validierungsraten gestützt [29]. Die in dieser Arbeit entwickelte Pipeline nutzt deshalb *GATK* zur Identifizierung von Mutationen.

Natürlich ist die Pipeline keine Musterlösung, da nicht alle Mutationen zu 100% detektiert und differenziert werden. Derzeit existiert nicht solch eine Musterlösung, hierfür wurde die *ICGC-TCGA-DREAM Somatic Mutation Calling Challenge* ins Leben gerufen. Die *DREAM Challenges* Organisation wurde vom IBM Wissenschaftler Gustavo Stolovitzky gegründet und befasst sich mit der Lösung wissenschaftlicher Fragestellungen die noch nicht geklärt werden konnten (http://dreamchallenges.org/). Dabei wird ein definierter NGS-Datensatz bestehend aus primären DNA-Proben als auch aus künstlich erstellten NGS-Daten analysiert. Nach den Analysen aller teilnehmenden Gruppen werden die Ergebnisse der jeweiligen Softwaretools veröffentlicht. Die Entwickler können anhand der Ergebnisse ihre Algorithmen erweitern und erneut den vorgegebenen Datensatz analysieren. Nach mehreren Runden wird ein Ranking der teilgenommenen Softwaretools zur Identifizierung von somatischen Mutationen erstellt. Für den synthetisch generierten Datensatz wurde dies auch schon komplett durchgeführt und auf der Homepage veröffentlicht (https://www.synapse.org/#!Synapse:syn312572/wiki/58893).

Jedoch steht die Entscheidung für den reellen Datensatz noch aus. Somit müssen weitere Analysen durchgeführt werden um das Beste Softwaretool zu bestimmen.
Diese Herangehensweise suggeriert fälschlicherweise, dass die primäre Analyse ausreicht um somatische Tumor-relevante Mutationen identifizieren zu können. Jedoch sind die Annotation, Filterung, biologische Interpretation und vor allem weitere Experimente für die Klassifizierung von Mutationen unumgänglich.

7.2 Sekundäre Analyse

Die Annotation von identifizierten Punktmutationen ermöglicht sowohl die Charakterisierung und Filterung als auch die Bewertung von Auswirkungen und Einflüsse der Mutationskandidaten. Wie bereits in Material und Methoden erwähnt nutzen wir hierfür die Softwaretools *ANNOVAR* und *SeattleSeq*. Hierbei werden die verfügbaren Informationen des Referenzgenoms mit den identifizierten Punktmutationen zusammengeführt. Die wichtigste Eigenschaft der Mutationen ist die Position, anhand dieser kann eine Base welche durch eine andere Base ausgetauscht wird in Bezug zum Referenzgenom gesetzt werden. Hierzu kann die Basenänderung innerhalb oder außerhalb Genkodierender Regionen lokalisiert werden. Die daraus folgende Auswirkung der Basenänderung bezüglich eines Aminosäureaustauschs in der Proteinsequenz kann mit diesen Informationen bewertet werden. Weitere Eigenschaften der Mutationen bezüglich der betroffenen Region wie beispielsweise die Konservierung und mögliche Strukturänderungen können zusätzlich jeweils berechnet werden. Diese Informationen dienen zur Differenzierung von Mutationen die eher eine funktionelle Auswirkung haben als andere. Zu diesen berechneten Prädiktionen und Eigenschaften werden auch Informationen aus Datenbanken wie dbSNP und 1000Genomes den jeweiligen Kandidaten zugeordnet.
Die anschließende Filterung nutzt die annotierten Informationen, um die Mutationskandidaten zu priorisieren, welche eine wahrscheinlichere Auswirkung auf die Entartung der Zelle haben. Basierend auf diesen Analysen, konnte eine Filterstrategie zur Identifizierung Tumor relevanter somatischer Mutationen im MM entwickelt werden (Abbildung 25) [29]. Diese Vorgehensweise wurde ebenso bei der Analyse der BL- und FL-Daten genutzt. Des Weiteren stehen wiederkehrende Mutationen bei vielen Filterstrategien im Fokus [45]. Auch statistische Methoden werden genutzt um Mutationskandidaten zu priorisieren [30]. Vor allem bei komparativen Analysen mussten wir jedoch feststellen, dass annotierte Informationen nicht immer eindeutig sind. Vor allem der annotierte Genname ist von der verwendeten Datenbank des Softwaretools abhängig, da in den Datenbanken die Identifikationsnummer eindeutig sind

aber nicht die Gennamen. Somit konnte die Analyse basierend auf mutierten Genen, anhand der unterschiedlichen genutzten Syntax der Gennamen nicht effizient durchgeführt werden. Auch wiederkehrende mutierte Gene wurden bei komparativen Analysen dadurch falsch gewertet. Deshalb wurde das *Genmapping* Skript erstellt, welches alle verfügbaren Namen eines spezifischen Gens mit einem eindeutigen vom HGNC vergebenen Namen verbindet. Somit ist eine einheitliche Syntax des eigenen Datensatzes hinsichtlich der Gennamen gegeben. Des Weiteren besteht die Möglichkeit in Abhängigkeit des Gennamens externe Mutationslisten in die Genbasierende Analyse mit einzubinden ohne den gesamten Datensatz erneut zu analysieren. Unsere initiale WES-Studie zeigte, wie effizient ein extern analysierter Datensatz (39 primäre MM-Proben) in die Analyse eingebunden werden kann [29]. Hierzu ist hervorzuheben, dass durch die Vereinheitlichung der Namen und damit die Vereinigung der MM-Daten, die Beobachtung der Akkumulation von Mutationen in den RTK, Adhäsionsmoleküle und deren *downstream* Effektoren ermöglicht wurde. Unter Verwendung der *SeattleSeq* Software wurden die Gennamen basierend auf den RefSeq und CCDS Datenbanken annotiert. Neben den vielen effizienten Berechnungen des Softwaretools war die statische Verwendung dieser Datenbanken jedoch nicht ausreichend. Beispielsweise wurden die Immunglobuline in den verwendeten Datenbanken zwar gelistet, entsprachen jedoch nicht den Vorgaben von *SeattleSeq*, um in die Datenbank aufgenommen zu werden. Somit existieren zwar NGS-Daten zu den Immunglobulinen, jedoch gehen diese Informationen durch die Verwendung von *SeattleSeq* verloren. Hierzu ist die Flexibilität von *ANNOVAR* eine Alternative, da unterschiedliche Datenbanken in die Annotation eingebunden werden können. Die Annotation der Mutationen mit *ANNOVAR* ist fast so effizient wie mit *SeattleSeq*, jedoch verlangsamen benötigte Bearbeitungen der Dateien den gesamten Prozess.

Die Analyse mutierter Gene hinsichtlich der Häufigkeit von Mutationen und Anzahl der betroffenen sequenzierten Genome kann mit Softwaretools wie *vcftools* durchgeführt werden. Hierzu dienen die Mutationslisten als Eingabe, jedoch ist die Vereinheitlichung der Gennamen nicht gegeben. Mit Hilfe des *Genmapping* Skript konnten die Eingabedateien erweitert werden. Sowohl die zuvor genutzten als auch die neu annotierten Namen wurden eingebunden. Da dies jedoch die Struktur der VCF-Dateien ändert, konnten Softwaretools wie *vcftools* nicht mehr verwendet werden, da solche Veränderungen nicht unterstützt werden. Um dies zu umgehen, wurde das *Heatmap* Skript entwickelt. Somit konnte aus den veränderten VCF-Dateien eine Heatmap erstellt werden. Diese Heatmaps ermöglichen unter Berücksichtigung spezifischer Gen Gruppen die Beobachtung von Akkumulationen der Mutationen. Sowie bei den ersten MM-Datensatz standen die RTK, Adehäsionsmoleküle und deren *downstream* Effektoren beim

zweiten MM-Datensatz im Fokus. Hierzu wurden mit Hilfe der Heatmap die Patienten hinsichtlich der Gen Gruppen untersuch. Dies ermöglichte die Identifizierung von drei neuen molekularen Subgruppen im MM, die zurzeit klinisch und funktionell näher unter sucht werden. Zudem wurde das *HeatMap* Skript erweitert, um das Einbinden veränderter VCF-Dateien zu ermöglichen. Hierbei wurden VCF-Dateien von FL Daten aus dem ICGC verwendet, die mit einer ICGC-internen Pipeline bearbeitet wurden und uns zur Verfügung standen. Im Rahmen des FL-Projektes diente das *HeatMap* Skript der Zusammenführung dieser veränderten VCF-Dateien und der Unterteilung in t(14;18)-positive und t(14;18)-negative FL.

Der Fokus bei der WES-Applikation liegt auf den Gen-kodierenden Regionen des Referenzgenoms. Deshalb liegt auch der Fokus der Analysestrategien auf diesen Regionen. Wobei auch benachbarte oder nicht im Exom lokalisierte Bereiche ebenso sequenziert werden, die aufgrund des Sonden-Designs bei der Erstellung der Exom-Sequenzierungsbibliothek mit angereichert wurden. Diese zusätzlichen NGS-Daten können zur Identifizierung von SNPs außerhalb des Exoms dienen und somit für epidemiologische Analysen genutzt werden [125]. Wichtige Eigenschaften wie die Qualitäten der Basen und der zugeordneten Reads, die Anzahl an identifizierten und validierten Mutationen, sowie die Abdeckung der exonischen Bereiche innerhalb des Referenzgenoms beschreiben die Qualität von NGS-Daten. Vor allem die durchschnittliche Abdeckung wird standardmäßig zu den NGS-Daten ausgewertet und angegeben. Die prozentuale Abdeckung sequenzierter Proben von WES-Daten beträgt meist 99 %, wobei mindestens ein Read die Base abdeckt. In den hier aufgezeigten Ergebnissen zu den MM-, FL- und BL-Daten ist dies zu beobachten. Ebenso der Verlust an abgedeckten Basen, der durch die erforderte Mindestanforderung für die effiziente Identifizierung von heterozygoten Mutationen von etwa 13 Reads entsteht [110; 111]. Dieser Verlust ist nicht sofort ersichtlich oder so gering, dass dies nicht weiter beachtet wird. In dieser Arbeit wurde der Verlust an Basen in den gesamten sequenzierten NGS-Daten im Detail analysiert und annotiert. Hierzu konnten diverse Onkogene und Tumorsupressorgene identifiziert werden, die entweder durch keine oder nur wenige Reads abgedeckt wurden. Der Verlust oder die nicht Beachtung dieser Regionen, könnte dazu führen das signifikante Mutationen bei den Analysen untergehen.

Der generelle Verlust an Informationen ist bei einzelnen Proben gering. Werden jedoch, wie bei der simultanen Analyse zur Identifizierung der Mutationen, die NGS-Daten von Tumor- und Normalproben zusammengeführt, können diese sich je nach Effizienz der Sequenzierung unter Berücksichtigung der Normalprobe erweitern, wenn diese in dem jeweiligen Bereich schlechter abgedeckt ist. Im Rahmen eines NGS-Projektes werden meist mehrere Proben sequenziert. Die gesamte Analyse aller Proben führt zur Entstehung von unbeachtete Regionen.

Die schwarzen Regionen sind meist nicht größer als bei der am besten sequenzierten Probe im Datensatz, jedoch können die grauen Regionen eine hohe Menge an Basen betreffen. Diese Regionen müssen bei der Analyse wiederkehrender Mutationen im Detail untersucht werden. Beispielsweise kann im gesamten Projekt nur eine geringe Anzahl an Proben eine Mutation aufweisen, wobei die restlichen Proben keine NGS-Daten für diese Region bereitstellen. Hierzu muss zunächst die Region in den restlichen Proben erneut sequenziert werden, um die Frequenz der identifizierten Mutation im gesamten Datensatz zu bewerten. Ebenso können beispielsweise häufig mutierte Regionen in nur einer Probe abgedeckt sein, wobei diese Probe unter Umständen jedoch keine Mutationen aufweist. Somit würden die potenziellen Mutationen in den nicht abgedeckten Proben keine Beachtung in der Analyse finden.

Letztlich konnte ein kleiner Anteil an unbeachteten Regionen in den MM-WES-Datensatz mit Hilfe der Amplikon Sequenzierung näher untersucht werden. Mutationen konnten mit diesem Ansatz jedoch nicht in diesen Regionen aufgedeckt werden. Des Weiteren wurden auch vereinzelte schwarze Regionen in den BL-Daten mit Hilfe der Sanger Sequenzierung näher untersucht. Die Hälfte der Regionen konnte nicht sequenziert werden und in den untersuchten Regionen konnten ebenfalls keine Mutationen detektiert werden. Eine Amplikon Sequenzierung speziell zu den identifizierten unbeachteten Regionen wäre eine weitere Möglichkeit, um eine umfassenden Überblick über möglicherweise nicht detektierte Mutationen zu gewinnen. Da es sich bei den nicht abgedeckten Regionen unter anderem um schwer zu amplifizierende Regionen handelt, wie beispielsweise repetitive Elemente oder ein hoher G/C-Gehalt, könnte ein solcher Ansatz jedoch auch technische Probleme bereiten und somit mit einem hohen Ausfall verbunden sein. Da die technischen Möglichkeiten in dieser Hinsicht limitiert sind, bleibt eine detaillierte Untersuchung dieser Regionen aus. Unter Berücksichtigung der beobachteten durchschnittlichen somatischen Mutationsfrequenzen von etwa 4 Mutationen pro Mbp im MM und im BL, sowie der identifizierten unbeachteten Regionen von etwa 0,254 Mbp im MM und 0,579 Mbp im BL, kann die jeweilige Anzahl an möglicherweise nicht detektierten Mutationen in den Datensätzen berechnet werden. Diese belaufen sich auf eine Mutation im MM-Datensatz pro Probe und auf zwei Mutationen im BL-Datensatz pro Probe.

Des Weiteren konnten wir beobachten, dass sich die Komposition und Anzahl der unbeachteten Regionen je nach verwendeten Exom-Extraktion-Kits ändert. Zum einen liegt das daran, das jedes Kits individuelle Bereiche abdeckt und sich jeweils nur ein Teil des zugrunde liegenden Designs überlappt (Abbildung 30). Eine konsistente Nutzung eines Exom-Extraktion-Kits in einem NGS-Projekt kann die Anzahl an unbeachteten Regionen somit minimieren, wenn die

Effizienz des Kits gleichbleibend hoch ist. Ein nicht zu unterbindender Effekt ist der Verlust von sequenzierten Regionen, die durch die Lücken des jeweiligen Referenzgenoms verursacht werden. Während des *Alignments* können die Lücken in den Referenzgenom Versionen hg18 (302 Lücken), hg19 (457 Lücken) und hg38 (819 Lücken) nicht mit den sequenzierten Daten abgeglichen werden. Zu den Lücken der Version hg38 existieren derzeit noch keine Sequenzinformationen, somit besteht dieser Verlust an Daten in allen Sequenzierungsdaten des menschlichen Genoms.

Neben der Entwicklung der Pipeline zur Identifizierung somatischer Mutationen in hämatologischen Neoplasien wurden im Rahmen dieser Arbeit sowohl die Methoden der primären Analyse als auch die Ergebnisse aus der sekundären Analyse hinterfragt und näher untersucht. Es zeigte sich, dass die entwickelte Pipeline im Vergleich zu anderen Pipelines sehr gute Ergebnisse liefert. Zudem wurde sowohl eine vereinheitlichte Analyse von einzelnen Tumorproben als auch die Analyse von Tumor- und Normalprobenpaare in die Pipeline implementiert. Schwächen von bisherigen standardisierten Softwaretools wurden durch die Entwicklung neuer Skripte kompensiert. Hierzu gehören die Vereinheitlichung der Gennamen mit Hilfe des *Genmapping* Skripts und der Generierung von *Heatmaps* mit Hilfe des Heatmap Skripts. Beide Skripte unterstützen die effiziente Integration externer Daten und erleichtern somit die biologische Interpretation der Daten. Die Identifizierung sowie Annotation von unbeachteten Regionen unter Verwendung des *unconReg* Skripts ermöglichte zudem eine erweiterte Interpretation der NGS-Daten. Wobei Fehlinterpretationen aufgrund fehlender Mutationen, die beispielsweise verursacht durch niedrige Allel Frequenzen oder geringe Rekurrenz auftreten, können durch die Analyse von unbeachteten Regionen bzw. die Identifizierung von nicht vorhandenen Sequenzierungsdaten vermieden werden.

7.3 Ausblick

Die entwickelte Pipeline und deren Erweiterung durch drei neu entwickelte Skripte führten zu wichtigen biologischen Erkenntnissen. Im Rahmen von Projekten zum MM, BL und FL wurde diese Pipeline verwendet und zu Teilen veröffentlicht [29]. Ein wichtiger Schritt war die Überprüfung der identifizierten Mutationen mit der Sanger Sequenzierung. Jedoch können Reparaturmechanismen, beispielsweise während der Transkription, die Punktmutation beseitigen. In der Zukunft könnten zusätzliche RNA-Sequenzierungen die Mutationen auf mRNA-Ebene bestätigen und anschließende funktionelle Analysen Aufschluss über mögliche Funktionsänderungen geben.

Des Weiteren sind Prädispositionen, welche mit der Kanzerogenese assoziert werden können

oft nicht in den Ergebnissen aufgelistet. Diese werden sowohl durch die simultane Analyse als auch durch die Subtraktion der Probenpaare aus den Listen entfernt. Hierzu untersuchen wir interessante Kandidaten im zweiten Amplikon-MM-Datensatz, wobei während der Analyse auf die Subtraktion verzichtet wurde [123]. Ebenso sollte der Einfluss von Mutationen die nicht in den kodierenden Regionen lokalisiert sind nicht vernachlässigt werden. Diese Mutationen können Regionen mit einer Regulierungsfunktion betreffen und somit zur Entartung der Zelle beitragen. Hierzu könnten die Kandidaten in einer gesonderten Analyse untersucht werden, da sie sonst aufgrund ihrer Lokalisation aus der standardisierten Analyse gefiltert werden.

Eine Verbesserung der Identifizierung von Mutation kann mit Hilfe der Informationen wie Ploidie und Reinheit der Probe erreicht werden [139]. Somit wäre es in Zukunft sinnvoll die identifizierten CNVs mit in die Analyse einzuschließen, diese können aus den NGS-Daten selbst oder aus Array-Daten bestimmt werden. Hierbei ist zu beachten, dass die geringere Auflösung der Array-Daten bei der Identifizierung von Punktmutationen als zusätzlicher Störfaktor fungieren könnte oder in den WES-NGS-Daten nur CNVs innerhalb der abgedeckten Regionen identifiziert werden können. Somit würde dies für eine WGS sprechen, um CNVs zu detektieren. Allerdings ist die Effizienz der Softwaretools zur Identifizierung von CNVs auch von der Tiefe der Abdeckung abhängig. Aufgrund der hohen Kosten und großen Datenmengen, die eine WGS der zweiten Generation mit sich bringt, wirkt diese nicht attraktiv. Vielversprechend erscheint daher die *Single Molecule Real Time* (SMRT)-Sequenzierung der dritten Generation. Bei dieser Technik kann eine viel längere Sequenz produziert und die Analyse auf ein einzelnes Molekül spezifiziert werden, ohne eine PCR zu nutzen [140]. Ein positives Beispiel dafür ist die Firma PacBioSystems. Die entwickelte Sequenzierer Sequel oder RS Ⅱ können bis zu 60 Kilo-Basenpaare (Kbp) lange Sequenzen produzieren und mit Hilfe eines hochempfindlichen Halbleiters die Basenabfolge detektieren. Sowie PacBioSystems haben auch Firmen wie Oxford Nanopore oder Complete Genomics neue Technologien mit viel Potenzial entwickelt. Leider weisen jedoch noch alle Sequenzierer der dritten Generation hohe Fehlerraten auf und auch die Kosten sind nicht so niedrig wie einst prophezeit. Es bedarf noch etwas Zeit bis sich die Technik ausreichend stabilisiert hat und diese Geräte tatsächlich mit den neuesten Plattformen von Illumina konkurrieren können.

8 Literaturverzeichnis

[1] LUCH, A., 2005. Nature and nurture - lessons from chemical carcinogenesis. *Nat Rev Cancer 5*, 2 (Feb), 113-125.

[2] NACHMAN, M.W. and CROWELL, S.L., 2000. Estimate of the mutation rate per nucleotide in humans. *Genetics 156*, 1 (Sep), 297-304.

[3] LYNCH, M., 2010. Rate, molecular spectrum, and consequences of human mutation. *Proc Natl Acad Sci U S A 107*, 3 (Jan 19), 961-968.

[4] CONRAD, D.F., *et al.*, 2011. Variation in genome-wide mutation rates within and between human families. *Nat Genet 43*, 7 (Jul), 712-714.

[5] SAMUELS, M.E. and FRIEDMAN, J.M., 2015. Genetic mosaics and the germ line lineage. *Genes (Basel) 6*, 2, 216-237.

[6] MCKUSICK, V.A., 1966. *Mendelian inheritance in man; catalogs of autosomal dominant, autosomal recessive, and X-linked phenotypes*. Johns Hopkins Press, Baltimore,.

[7] HAMOSH, A., *et al.*, 2000. Online Mendelian Inheritance in Man (OMIM). *Hum Mutat 15*, 1, 57-61.

[8] SHERRY, S.T., WARD, M., and SIROTKIN, K., 1999. dbSNP-database for single nucleotide polymorphisms and other classes of minor genetic variation. *Genome Res 9*, 8 (Aug), 677-679.

[9] BAMFORD, S., *et al.*, 2004. The COSMIC (Catalogue of Somatic Mutations in Cancer) database and website. *Br J Cancer 91*, 2 (Jul 19), 355-358.

[10] BEROUD, C. and SOUSSI, T., 1996. APC gene: database of germline and somatic mutations in human tumors and cell lines. *Nucleic Acids Res 24*, 1 (Jan 1), 121-124.

[11] SHIN, G., *et al.*, 2011. GENT: gene expression database of normal and tumor tissues. *Cancer Inform 10*, 149-157.

[12] GENOMES PROJECT, C., *et al.*, 2012. An integrated map of genetic variation from 1,092 human genomes. *Nature 491*, 7422 (Nov 1), 56-65.

[13] PANG, A.W., *et al.*, 2010. Towards a comprehensive structural variation map of an individual human genome. *Genome Biol 11*, 5, R52.

[14] REDON, R., *et al.*, 2006. Global variation in copy number in the human genome. *Nature 444*, 7118 (Nov 23), 444-454.

[15] KNUDSON, A.G., JR., 1971. Mutation and cancer: statistical study of retinoblastoma. *Proc Natl Acad Sci U S A 68*, 4 (Apr), 820-823.

[16] EIBEL, H., *et al.*, 2014. B cell biology: an overview. *Curr Allergy Asthma Rep 14*, 5 (May), 434.

[17] PIEPER, K., GRIMBACHER, B., and EIBEL, H., 2013. B-cell biology and development. *J Allergy Clin Immunol 131*, 4 (Apr), 959-971.

[18] TENG, G. and PAPAVASILIOU, F.N., 2007. Immunoglobulin somatic hypermutation. *Annu Rev Genet 41*, 107-120.

[19] KUPPERS, R., 2005. Mechanisms of B-cell lymphoma pathogenesis. *Nat Rev Cancer 5*, 4 (Apr), 251-262.

[20] ALLEN, C.D., OKADA, T., TANG, H.L., and CYSTER, J.G., 2007. Imaging of germinal center selection events during affinity maturation. *Science 315*, 5811 (Jan 26), 528-531.

[21] MURAMATSU, M., *et al.*, 2000. Class switch recombination and hypermutation require activation-induced cytidine deaminase (AID), a potential RNA editing enzyme. *Cell 102*, 5 (Sep 1), 553-563.

[22] EDHOLM, E.S., BENGTEN, E., and WILSON, M., 2011. Insights into the function of IgD. *Dev Comp Immunol 35*, 12 (Dec), 1309-1316.

[23] KUPPERS, R. and DALLA-FAVERA, R., 2001. Mechanisms of chromosomal translocations in B cell lymphomas. *Oncogene 20*, 40 (Sep 10), 5580-5594.
[24] BAHLIS, N.J., 2012. Darwinian evolution and tiding clones in multiple myeloma. *Blood 120*, 5 (Aug 2), 927-928.
[25] HENG, H.H., *et al.*, 2010. The evolutionary mechanism of cancer. *J Cell Biochem 109*, 6 (Apr 15), 1072-1084.
[26] SWERDLOW SH, C.E., HARRIS NL, JAFFE ES, PILERI SA, STEIN H, 2008. *WHO Classification of Tumors of Haematopoietic and Lymphoid Tissues*.
[27] MORGAN, G.J., WALKER, B.A., and DAVIES, F.E., 2012. The genetic architecture of multiple myeloma. *Nat Rev Cancer 12*, 5 (May), 335-348.
[28] PRIDEAUX, S.M., CONWAY O'BRIEN, E., and CHEVASSUT, T.J., 2014. The genetic architecture of multiple myeloma. *Adv Hematol 2014*, 864058.
[29] LEICH, E., *et al.*, 2013. Multiple myeloma is affected by multiple and heterogeneous somatic mutations in adhesion- and receptor tyrosine kinase signaling molecules. *Blood Cancer J 3*, e102.
[30] CHAPMAN, M.A., *et al.*, 2011. Initial genome sequencing and analysis of multiple myeloma. *Nature 471*, 7339 (Mar 24), 467-472.
[31] GUPTA, A., *et al.*, 2015. Single-molecule analysis reveals widespread structural variation in multiple myeloma. *Proc Natl Acad Sci U S A 112*, 25 (Jun 23), 7689-7694.
[32] LEICH, E., OTT, G., and ROSENWALD, A., 2011. Pathology, pathogenesis and molecular genetics of follicular NHL. *Best Pract Res Clin Haematol 24*, 2 (Jun), 95-109.
[33] LEICH, E., *et al.*, 2009. Follicular lymphomas with and without translocation t(14;18) differ in gene expression profiles and genetic alterations. *Blood 114*, 4 (Jul 23), 826-834.
[34] SCHULER, F., *et al.*, 2009. Prevalence and frequency of circulating t(14;18)-MBR translocation carrying cells in healthy individuals. *Int J Cancer 124*, 4 (Feb 15), 958-963.
[35] KRIDEL, R., SEHN, L.H., and GASCOYNE, R.D., 2012. Pathogenesis of follicular lymphoma. *J Clin Invest 122*, 10 (Oct), 3424-3431.
[36] KISHIMOTO, W. and NISHIKORI, M., 2014. Molecular pathogenesis of follicular lymphoma. *J Clin Exp Hematop 54*, 1, 23-30.
[37] CASULO, C., BURACK, W.R., and FRIEDBERG, J.W., 2015. Transformed follicular non-Hodgkin lymphoma. *Blood 125*, 1 (Jan 1), 40-47.
[38] LEICH, E., *et al.*, 2011. MicroRNA profiles of t(14;18)-negative follicular lymphoma support a late germinal center B-cell phenotype. *Blood 118*, 20 (Nov 17), 5550-5558.
[39] DAVE, S.S., 2010. Host factors for risk and survival in lymphoma. *Hematology Am Soc Hematol Educ Program 2010*, 255-258.
[40] MOLYNEUX, E.M., *et al.*, 2012. Burkitt's lymphoma. *Lancet 379*, 9822 (Mar 31), 1234-1244.
[41] FERRY, J.A., 2006. Burkitt's lymphoma: clinicopathologic features and differential diagnosis. *Oncologist 11*, 4 (Apr), 375-383.
[42] DAVE, S.S., *et al.*, 2006. Molecular diagnosis of Burkitt's lymphoma. *N Engl J Med 354*, 23 (Jun 8), 2431-2442.
[43] HUMMEL, M., *et al.*, 2006. A biologic definition of Burkitt's lymphoma from transcriptional and genomic profiling. *N Engl J Med 354*, 23 (Jun 8), 2419-2430.
[44] SCHMITZ, R., *et al.*, 2012. Burkitt lymphoma pathogenesis and therapeutic targets from structural and functional genomics. *Nature 490*, 7418 (Oct 4), 116-120.
[45] RICHTER, J., *et al.*, 2012. Recurrent mutation of the ID3 gene in Burkitt lymphoma identified by integrated genome, exome and transcriptome sequencing. *Nat Genet 44*, 12 (Dec), 1316-1320.

[46] LOVE, C., *et al.*, 2012. The genetic landscape of mutations in Burkitt lymphoma. *Nat Genet 44*, 12 (Dec), 1321-1325.
[47] ALEXANDROV, L.B., *et al.*, 2013. Signatures of mutational processes in human cancer. *Nature 500*, 7463 (Aug 22), 415-421.
[48] VAQUE, J.P., *et al.*, 2014. B-cell lymphoma mutations: improving diagnostics and enabling targeted therapies. *Haematologica 99*, 2 (Feb), 222-231.
[49] SANGER, F., NICKLEN, S., and COULSON, A.R., 1977. DNA sequencing with chain-terminating inhibitors. *Proc Natl Acad Sci U S A 74*, 12 (Dec), 5463-5467.
[50] LANDER, E.S., *et al.*, 2001. Initial sequencing and analysis of the human genome. *Nature 409*, 6822 (Feb 15), 860-921.
[51] INTERNATIONAL HUMAN GENOME SEQUENCING, C., 2004. Finishing the euchromatic sequence of the human genome. *Nature 431*, 7011 (Oct 21), 931-945.
[52] VENTER, J.C., *et al.*, 2001. The sequence of the human genome. *Science 291*, 5507 (Feb 16), 1304-1351.
[53] KENT, W.J., *et al.*, 2002. The human genome browser at UCSC. *Genome Res 12*, 6 (Jun), 996-1006.
[54] LANDER, E.S., 2011. Initial impact of the sequencing of the human genome. *Nature 470*, 7333 (Feb 10), 187-197.
[55] GENOVESE, G., *et al.*, 2013. Mapping the human reference genome's missing sequence by three-way admixture in Latino genomes. *Am J Hum Genet 93*, 3 (Sep 5), 411-421.
[56] VAN DIJK, E.L., AUGER, H., JASZCZYSZYN, Y., and THERMES, C., 2014. Ten years of next-generation sequencing technology. *Trends Genet 30*, 9 (Sep), 418-426.
[57] ESCALANTE, A.E., BARBOLLA, L.J., RAMIREZ-BARAHONA, S., and EGUIARTE, L.E., 2014. The study of biodiversity in the era of massive sequencing. *Revista Mexicana De Biodiversidad 85*, 4 (Dec), 1249-1264.
[58] METZKER, M.L., 2010. Sequencing technologies - the next generation. *Nat Rev Genet 11*, 1 (Jan), 31-46.
[59] SHENDURE, J. and JI, H., 2008. Next-generation DNA sequencing. *Nat Biotechnol 26*, 10 (Oct), 1135-1145.
[60] MARGULIES, M., *et al.*, 2005. Genome sequencing in microfabricated high-density picolitre reactors. *Nature 437*, 7057 (Sep 15), 376-380.
[61] LIU, L., *et al.*, 2012. Comparison of next-generation sequencing systems. *J Biomed Biotechnol 2012*, 251364.
[62] MEDINI, D., *et al.*, 2008. Microbiology in the post-genomic era. *Nat Rev Microbiol 6*, 6 (Jun), 419-430.
[63] VALOUEV, A., *et al.*, 2008. A high-resolution, nucleosome position map of C. elegans reveals a lack of universal sequence-dictated positioning. *Genome Res 18*, 7 (Jul), 1051-1063.
[64] SHENDURE, J., *et al.*, 2005. Accurate multiplex polony sequencing of an evolved bacterial genome. *Science 309*, 5741 (Sep 9), 1728-1732.
[65] MCKERNAN, K., MARBLEHEAD, MA 01945 (US), BLANCHARD, A., MIDDLETOWN, MA 01949 (US), KOTLER, L., ALLSTON, MA 02134 (US), and COSTA, G., CARLSBAD, CA 92009 (US), 2006. Reagents, methods and libraries for bead-based sequencing.
[66] GLENN, T.C., 2011. Field guide to next-generation DNA sequencers. *Mol Ecol Resour 11*, 5 (Sep), 759-769.
[67] HARISMENDY, O., *et al.*, 2009. Evaluation of next generation sequencing platforms for population targeted sequencing studies. *Genome Biol 10*, 3, R32.
[68] SHERIDAN, C., 2014. Illumina claims $1,000 genome win. *Nat Biotechnol 32*, 2 (Feb), 115.

[69] RATAN, A., *et al.*, 2013. Comparison of sequencing platforms for single nucleotide variant calls in a human sample. *PLoS One 8*, 2, e55089.
[70] MELDRUM, C., DOYLE, M.A., and TOTHILL, R.W., 2011. Next-generation sequencing for cancer diagnostics: a practical perspective. *Clin Biochem Rev 32*, 4 (Nov), 177-195.
[71] HUNKAPILLER, T., KAISER, R.J., KOOP, B.F., and HOOD, L., 1991. Large-scale and automated DNA sequence determination. *Science 254*, 5028 (Oct 4), 59-67.
[72] ZHAO, M., *et al.*, 2013. Computational tools for copy number variation (CNV) detection using next-generation sequencing data: features and perspectives. *BMC Bioinformatics 14 Suppl 11*, S1.
[73] LONIGRO, R.J., *et al.*, 2011. Detection of somatic copy number alterations in cancer using targeted exome capture sequencing. *Neoplasia 13*, 11 (Nov), 1019-1025.
[74] WESTERMANN, A.J., GORSKI, S.A., and VOGEL, J., 2012. Dual RNA-seq of pathogen and host. *Nat Rev Microbiol 10*, 9 (Sep), 618-630.
[75] ENDRIS, V., *et al.*, 2016. NGS-based BRCA1/2 mutation testing of high-grade serous ovarian cancer tissue: results and conclusions of the first international round robin trial. *Virchows Arch 468*, 6 (Jun), 697-705.
[76] MERTES, F., *et al.*, 2011. Targeted enrichment of genomic DNA regions for next-generation sequencing. *Brief Funct Genomics 10*, 6 (Nov), 374-386.
[77] CLARK, M.J., *et al.*, 2011. Performance comparison of exome DNA sequencing technologies. *Nat Biotechnol 29*, 10 (Oct), 908-914.
[78] PARLA, J.S., *et al.*, 2011. A comparative analysis of exome capture. *Genome Biol 12*, 9, R97.
[79] SULONEN, A.M., *et al.*, 2011. Comparison of solution-based exome capture methods for next generation sequencing. *Genome Biol 12*, 9, R94.
[80] STADEN, R., 1979. A strategy of DNA sequencing employing computer programs. *Nucleic Acids Res 6*, 7 (Jun 11), 2601-2610.
[81] POTTER, S.C., *et al.*, 2004. The Ensembl analysis pipeline. *Genome Res 14*, 5 (May), 934-941.
[82] BROOKSBANK, C., *et al.*, 2003. The European Bioinformatics Institute's data resources. *Nucleic Acids Res 31*, 1 (Jan 1), 43-50.
[83] WHEELER, D.L., *et al.*, 2000. Database resources of the National Center for Biotechnology Information. *Nucleic Acids Res 28*, 1 (Jan 1), 10-14.
[84] SONNHAMMER, E.L., EDDY, S.R., and DURBIN, R., 1997. Pfam: a comprehensive database of protein domain families based on seed alignments. *Proteins 28*, 3 (Jul), 405-420.
[85] PISCHIMAROV, J., *et al.*, 2012. sRNAdb: a small non-coding RNA database for gram-positive bacteria. *BMC Genomics 13*, 384.
[86] PABINGER, S., *et al.*, 2014. A survey of tools for variant analysis of next-generation genome sequencing data. *Brief Bioinform 15*, 2 (Mar), 256-278.
[87] HATEM, A., BOZDAG, D., TOLAND, A.E., and CATALYUREK, U.V., 2013. Benchmarking short sequence mapping tools. *BMC Bioinformatics 14*, 184.
[88] RODRÍGUEZ-EZPELETA, N., HACKENBERG, M., and ARANSAY, A.M., 2011. *Bioinformatics for high throughput sequencing*. Springer Science & Business Media.
[89] KIRCHER, M., HEYN, P., and KELSO, J., 2011. Addressing challenges in the production and analysis of illumina sequencing data. *BMC Genomics 12*, 382.
[90] EWING, B., HILLIER, L., WENDL, M.C., and GREEN, P., 1998. Base-calling of automated sequencer traces using phred. I. Accuracy assessment. *Genome Res 8*, 3 (Mar), 175-185.
[91] EWING, B. and GREEN, P., 1998. Base-calling of automated sequencer traces using phred. II. Error probabilities. *Genome Res 8*, 3 (Mar), 186-194.

[92] COCK, P.J., *et al.*, 2010. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Res 38*, 6 (Apr), 1767-1771.
[93] ZHAO, H., *et al.*, 2014. CrossMap: a versatile tool for coordinate conversion between genome assemblies. *Bioinformatics 30*, 7 (Apr 1), 1006-1007.
[94] HINRICHS, A.S., *et al.*, 2006. The UCSC Genome Browser Database: update 2006. *Nucleic Acids Res 34*, Database issue (Jan 1), D590-598.
[95] SCHATZ, M.C., DELCHER, A.L., and SALZBERG, S.L., 2010. Assembly of large genomes using second-generation sequencing. *Genome Res 20*, 9 (Sep), 1165-1173.
[96] CHHANGAWALA, S., RUDY, G., MASON, C.E., and ROSENFELD, J.A., 2015. The impact of read length on quantification of differentially expressed genes and splice junction detection. *Genome Biol 16*, 131.
[97] WHITEFORD, N., *et al.*, 2005. An analysis of the feasibility of short read sequencing. *Nucleic Acids Res 33*, 19, e171.
[98] LI, W. and FREUDENBERG, J., 2014. Mappability and read length. *Front Genet 5*, 381.
[99] FOX, E.J., REID-BAYLISS, K.S., EMOND, M.J., and LOEB, L.A., 2014. Accuracy of Next Generation Sequencing Platforms. *Next Gener Seq Appl 1*.
[100] LI, H., RUAN, J., and DURBIN, R., 2008. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Res 18*, 11 (Nov), 1851-1858.
[101] FERRAGINA, P. and MANZINI, G., 2000. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on* IEEE, 390-398.
[102] BURROWS, M. and WHEELER, D.J., 1994. A block-sorting lossless data compression algorithm.
[103] LI, H. and DURBIN, R., 2009. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics 25*, 14 (Jul 15), 1754-1760.
[104] YU, X., *et al.*, 2012. How do alignment programs perform on sequencing data with varying qualities and from repetitive regions? *BioData Min 5*, 1, 6.
[105] RUFFALO, M., LAFRAMBOISE, T., and KOYUTURK, M., 2011. Comparative analysis of algorithms for next-generation sequencing read alignment. *Bioinformatics 27*, 20 (Oct 15), 2790-2796.
[106] TRAPNELL, C. and SALZBERG, S.L., 2009. How to map billions of short reads onto genomes. *Nat Biotechnol 27*, 5 (May), 455-457.
[107] SHANG, J., *et al.*, 2014. Evaluation and comparison of multiple aligners for next-generation sequencing data analysis. *Biomed Res Int 2014*, 309650.
[108] LI, H., *et al.*, 2009. The Sequence Alignment/Map format and SAMtools. *Bioinformatics 25*, 16 (Aug 15), 2078-2079.
[109] DEPRISTO, M.A., *et al.*, 2011. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nat Genet 43*, 5 (May), 491-498.
[110] MEYNERT, A.M., *et al.*, 2013. Quantifying single nucleotide variant detection sensitivity in exome sequencing. *BMC Bioinformatics 14*, 195.
[111] CIRULLI, E.T., *et al.*, 2010. Screening the human exome: a comparison of whole genome and whole transcriptome sequencing. *Genome Biol 11*, 5, R57.
[112] SUZUKI, A., *et al.*, 2013. Identification and characterization of cancer mutations in Japanese lung adenocarcinoma without sequencing of normal tissue counterparts. *PLoS One 8*, 9, e73484.
[113] LI, R., *et al.*, 2009. SNP detection for massively parallel whole-genome resequencing. *Genome Res 19*, 6 (Jun), 1124-1132.
[114] WEI, Z., *et al.*, 2011. SNVer: a statistical tool for variant calling in analysis of pooled or individual next-generation sequencing data. *Nucleic Acids Res 39*, 19 (Oct), e132.

[115] KOBOLDT, D.C., *et al.*, 2009. VarScan: variant detection in massively parallel sequencing of individual and pooled samples. *Bioinformatics 25*, 17 (Sep 1), 2283-2285.
[116] CIBULSKIS, K., *et al.*, 2013. Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. *Nat Biotechnol 31*, 3 (Mar), 213-219.
[117] LARSON, D.E., *et al.*, 2012. SomaticSniper: identification of somatic point mutations in whole genome sequencing data. *Bioinformatics 28*, 3 (Feb 1), 311-317.
[118] O'RAWE, J., *et al.*, 2013. Low concordance of multiple variant-calling pipelines: practical implications for exome and genome sequencing. *Genome Med 5*, 3, 28.
[119] GOODE, D.L., *et al.*, 2013. A simple consensus approach improves somatic mutation prediction accuracy. *Genome Med 5*, 9, 90.
[120] ANDRULIS, M., *et al.*, 2013. Targeting the BRAF V600E mutation in multiple myeloma. *Cancer Discov 3*, 8 (Aug), 862-869.
[121] NG, S.B., *et al.*, 2009. Targeted capture and massively parallel sequencing of 12 human exomes. *Nature 461*, 7261, 272-276.
[122] WEISSBACH, S., *et al.*, 2015. The molecular spectrum and clinical impact of DIS3 mutations in multiple myeloma. *Br J Haematol 169*, 1 (Apr), 57-70.
[123] KEPPLER, S., *et al.*, 2016. Rare SNPs in receptor tyrosine kinases are negative outcome predictors in multiple myeloma. *Oncotarget*(May 26).
[124] LEDERGERBER, C. and DESSIMOZ, C., 2011. Base-calling for next-generation sequencing platforms. *Brief Bioinform 12*, 5 (Sep), 489-497.
[125] GUO, Y., *et al.*, 2012. Exome sequencing generates high quality data in non-target regions. *BMC Genomics 13*, 194.
[126] ALIOTO, T.S., *et al.*, 2015. A comprehensive assessment of somatic mutation detection in cancer using whole-genome sequencing. *Nat Commun 6*, 10001.
[127] LANGMEAD, B., TRAPNELL, C., POP, M., and SALZBERG, S.L., 2009. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol 10*, 3, R25.
[128] LI, R., LI, Y., KRISTIANSEN, K., and WANG, J., 2008. SOAP: short oligonucleotide alignment program. *Bioinformatics 24*, 5 (Mar 1), 713-714.
[129] QUAIL, M.A., *et al.*, 2012. Optimal enzymes for amplifying sequencing libraries. *Nat Methods 9*, 1 (Jan), 10-11.
[130] CHILAMAKURI, C.S., *et al.*, 2014. Performance comparison of four exome capture systems for deep sequencing. *BMC Genomics 15*, 449.
[131] MOMOSE, S., *et al.*, 2015. The diagnostic gray zone between Burkitt lymphoma and diffuse large B-cell lymphoma is also a gray zone of the mutational spectrum. *Leukemia 29*, 8 (Aug), 1789-1791.
[132] ROBINSON, J.T., *et al.*, 2011. Integrative genomics viewer. *Nat Biotechnol 29*, 1 (Jan), 24-26.
[133] THORVALDSDOTTIR, H., ROBINSON, J.T., and MESIROV, J.P., 2013. Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration. *Brief Bioinform 14*, 2 (Mar), 178-192.
[134] LINDNER, R. and FRIEDEL, C.C., 2012. A comprehensive evaluation of alignment algorithms in the context of RNA-seq. *PLoS One 7*, 12, e52403.
[135] LANGMEAD, B., 2010. Aligning short sequencing reads with Bowtie. *Curr Protoc Bioinformatics Chapter 11*(Dec), Unit 11 17.
[136] LANGMEAD, B. and SALZBERG, S.L., 2012. Fast gapped-read alignment with Bowtie 2. *Nat Methods 9*, 4 (Apr), 357-359.
[137] TRAPNELL, C., PACHTER, L., and SALZBERG, S.L., 2009. TopHat: discovering splice junctions with RNA-Seq. *Bioinformatics 25*, 9 (May 1), 1105-1111.
[138] LI, H., 2013. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. In *ArXiv e-prints*.

[139] CARTER, S.L., *et al.*, 2012. Absolute quantification of somatic DNA alterations in human cancer. *Nat Biotechnol 30*, 5 (May), 413-421.
[140] EID, J., *et al.*, 2009. Real-time DNA sequencing from single polymerase molecules. *Science 323*, 5910 (Jan 2), 133-138.

9 Anhang

9.1 Skripte

Die entwickelten Skripte wurden mit Hilfe der integrierten Entwicklungsumgebung (IDE) Eclipse erstellt und in der Programmiersprache Python geschrieben. Für die Funktionserweiterung der jeweiligen Skripte wurden die Module wie beispielsweise os, sys, argparse, subprocess integriert.

9.1.1 Entwickelte Pipeline zur Identifizierung somatischer Mutationen

Wie schon gezeigt (siehe 6.2), kann die entwickelte Pipeline in folgende Analyseschritte unterteilt werden (Abbildung 31).

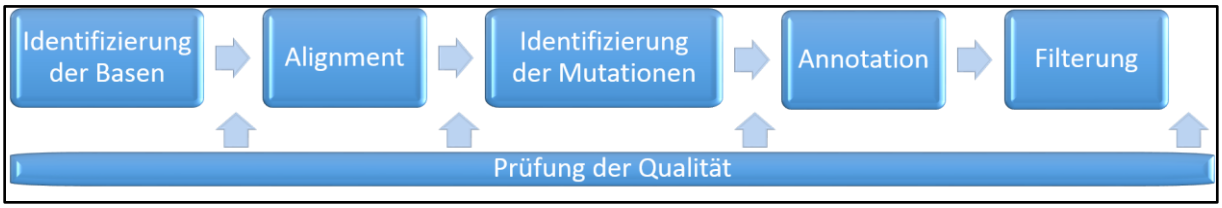


Abbildung 31: Schematische Darstellung der Pipeline
Beginnend mit der Identifizierung der Basen über die Generierung des Alignments sowie die Identifizierung der Mutationen mit anschließender Annotation und Filterung der Mutationskandidaten, wird hier der schematische Ablauf einer Analyse mit Hilfe der entwickelten Pipeline dargestellt. Zwischen den einzelnen Analyseschritten können Prüfungen der Daten durchgeführt werden, um einen Überblick zu der Qualität der NGS-Daten zu bekommen.

Im folgenedem der kommentierte Quellcode des Python-Skripts der entwickelten Pipeline, welcher in drei Dateien separiert vorliegt und den dazugehörigen Konfigurationsdateien.

Kofigurationsdatei mit benötigten Programmen für die Pipeline:

```
1.  [BWA]
2.  path = /home/jordan/Software/Aligner/bwa/bwa-0.7.10
3.  [samtools]
4.  path = /home/jordan/Software/samtools/samtools-1.1
5.  [bedtools]
6.  path = /home/jordan/Software/BEDTools/bedtools2-2.26.0/bin
7.  [Picard]
8.  path = /home/jordan/Software/Picard/picard-tools-1.118
9.  [GATK]
10. path = /home/jordan/Software/GATK/GenomeAnalysisTK-3.2-2
```

Kofigurationsdatei mit benötigten Programmen für die Pipeline:

```
1.  [BWA]
2.  #Mark shorter split hits as secondary (for Picard compatibility)
3.  param_M = -M
4.  readgroup_id = ID:MM_seq_dataset1_
5.  readgroup_cn = CN:IfPW
6.  readgroup_lb = LB:nimblegen_v2
7.  readgroup_pl = PL:Illumina_GA2
8.  readgroup_pu = PU:unknown
```

```
readgroup_sm = SM:
[samtools_view]
param = -Sb
[java]
Xmx = -Xmx12G
[bedtools_intersection]
param = -wa -abam
[bedtools_bedgraph]
param = -bga -ibam
[Picard_sortsam]
param = SORT_ORDER=coordinate VALIDATION_STRINGENCY=LENIENT
[Picard_remove_duplicates]
param = REMOVE_DUPLICATES=TRUE VALIDATION_STRINGENCY=LENIENT METRICS_FILE=
[GATK_reaglin_creator]
param = -T RealignerTargetCreator -known
[GATK_reagliner]
param = -T IndelRealigner --filter_bases_not_stored --consensusDeterminationModel KNOWNS_ONLY -
LOD 0.4 -targetIntervals
[GATK_base_recalibrator]
param = -T BaseRecalibrator -knownSites
[GATK_print_recal_base]
param = -T PrintReads -BQSR
[GATK_haplotypecalling]
param = -T HaplotypeCaller --emitRefConfidence GVCF --variant_index_type LINEAR --
variant_index_parameter 128000 -stand_call_conf 50.0 -stand_emit_conf 10.0 --dbsnp
```

Quellcode zur run-Datei:

```
#!/usr/bin/python
import Monitor_module, Command_module, argparse, os, sys, traceback, logging, tkFileDialog,
tkMessageBox, Tkinter

#Zur Ausfuehrung benoetigte Programme
required_programs=["BWA","samtools","bedtools","Picard","GATK"]
#Definition der Syntax von logging, welches fuer die Kommunikation per Kommandozeile genutzt wird
logging.basicConfig(level=logging.INFO, format='%(levelname)s:%(message)s')
#Festgelegte Speicherorte der Konfigurationsdateien
CONFIG_SOFTWARE_PARSER_PATH=os.path.join(sys.path[0], ".Pipeline_software_config_file.ini")
CONFIG_PARAMETER_PARSER_PATH=os.path.join(sys.path[0], ".Pipeline_parameter_config_file.ini")

#Initialisierung des root Fenster fuer die Eingabe und Selektion der Dateien und Pfade
root_window=Tkinter.Tk()
root_window.withdraw()
root_window.focus_force()

'''
Methoden fuer die Eingebe per tkFileDialogs
'''
def insert_sample_matrix(insert_bam_files):
    #Wenn die Eingabe nicht auf bam Dateien beschraenkt wird
    if not insert_bam_files:
        logging.info("Please select fastq files.")
        fastq_1_file="fastq_1_file"
        file_count=0
        sample_matrix={}
        #Waehrend erste Fastq Datei ausgewaehlt wird
        while fastq_1_file:
```

```
29.             file_count=file_count+1
30.             #Erste Fastq Datei per Dialog auswaehlen
31.             fastq_1_file = tkFileDialog.askopenfilename(parent=root_window, title='Select the first *.fastq file', filetypes=[("fastq file", "*.fastq")])
32.             #Wenn keine Datei ausgewaehlt wurde dann Schleife abbrechen
33.             if not fastq_1_file:
34.                 break
35.             #Wenn Datei zuvor schon ausgewaehlt wurde dann die erneute Auswahl ignorieren
36.             if fastq_1_file in sample_matrix:
37.                 logging.warn("File already exists and could not insert into the sample matrix")
38.                 continue
39.             #Auswahl der korrespondierenden zweiten Fastq Datei per Dialog
40.             fastq_2_file = tkFileDialog.askopenfilename(parent=root_window, title='Select the corresponding *.fastq file', filetypes=[("fastq file", "*.fastq")])
41.             #Wenn keine zweite Datei ausgewaehlt wurde dann Abbruch
42.             if not fastq_2_file:
43.                 logging.error("No korresponding fastq file selected. Abort.")
44.                 sys.exit(0)
45.             #Ausgabe welche Dateien ausgewaehlt wurden
46.             logging.info("Selected file pair:\nfastq_1 file: "+str(fastq_1_file)+"\ncorresponding fastq_2 file: "+str(fastq_2_file))
47.             #Ueberpruefe ob Datei bereits gespeichert wurde
48.             if fastq_1_file not in sample_matrix:
49.                 #Wenn nicht dann speichern
50.                 sample_matrix[fastq_1_file]=fastq_2_file
51.             else:
52.                 #Ansonsten ignorieren
53.                 logging.warn("tumor file already exists in the sample matrix it could not be overwrite")
54.         if sample_matrix:
55.             #Alle Dateien aus der Eingabe zurueck geben
56.             return sample_matrix
57.         else:
58.             logging.error("No fastq files are selected. Abort.")
59.             sys.exit(0)
60.     #Ansonsten die Eingabe von bam Dateien durchfuehren
61.     else:
62.         logging.info("Please select bam files.")
63.         bam_file="bam_file"
64.         file_count=0
65.         sample_matrix={}
66.         #Waehrend bam Datei ausgewaehlt wird
67.         while bam_file:
68.             file_count=file_count+1
69.             #Bam Datei per Dialog auswaehlen
70.             bam_file = tkFileDialog.askopenfilename(parent=root_window, title='Select a *.bam file', filetypes=[("bam file", "*.bam")])
71.             #Wenn keine Datei ausgewaehlt wurde dann Schleife abbrechen
72.             if not bam_file:
73.                 break
74.             #Wenn bam Datei zuvor schon ausgewaehlt wurde dann die erneute Auswahl ignorieren
75.             if bam_file in sample_matrix:
76.                 logging.warn("File already exists and could not insert into the sample matrix")
77.                 continue
78.             #Ueberpruefe ob Datei bereits gespeichert wurde
79.             if bam_file not in sample_matrix:
80.                 #Wenn nicht dann speichern
81.                 sample_matrix[bam_file]="empty"
```

```
82.             else:
83.                 #Ansonsten ignorieren
84.                 logging.warn("bam file already exists in the sample matrix it could not be overwrite")
85.         if sample_matrix:
86.             #Alle Dateien aus der Eingabe zurueck geben
87.             return sample_matrix
88.         else:
89.             logging.error("No bam file is selected. Abort.")
90.             sys.exit(0)
91.
92.     def insert_library():
93.         selected=True
94.         library_file='NONE'
95.         #Die Eingabe der Datei per Dialog abwarten
96.         while selected:
97.             #Auswahl der library Datei
98.             library_file = tkFileDialog.askopenfilename(parent=root_window, title='Select a library *.bed file',
    filetypes=[("bed file", "*.bed")])
99.             #Wenn keine Datei ausgewaehlt wird abbrechen
100.            if not library_file == 'NONE':
101.                break
102.        #Wenn eine Datei gespeichert wurde dann
103.        if library_file:
104.            #Alle Dateien aus der Eingabe zurueck geben
105.            return library_file
106.        #Ansonsten Abbruch
107.        else:
108.            logging.error("No library file is selected. Abort.")
109.            sys.exit(0)
110.
111.    '''
112.    Ueperpruefe die Kommandozeilen Parameter mit argparse
113.    '''
114.    #Die Beschreibung des Programms definieren
115.    inputparser = argparse.ArgumentParser(description='Start Pipeline to proceed WES-NGS-data to call
    SNV\'s.')
116.    #Eingabe per Fastq oder bam Datei
117.    inputparser.add_argument('-b','--insert_bam_files', action='store_true', dest='insert_bam_files',
    help='Starting from *.bam or *.fastq files. Default it is set from *.fastq files', default=False)
118.    #Prozessoranzahl festlegen
119.    inputparser.add_argument('-c','--cores', action='store', type=int , choices=xrange(1,1000), dest='cores',
    help='How many cores should be used for the analysis. Default value is 1', default=1)
120.    #Definition des Ausgabepfades
121.    inputparser.add_argument('-o','--output', action='store', dest='output_path',help='Output file
    name',required=True)
122.    #Pfad zur Referenzdatei welche mit BWA erstellt werden kann
123.    inputparser.add_argument('-r','--refmultifasta', action='store',type=argparse.FileType('r'),
    dest='refmultifasta_file', help='refmultifasta file prepared by bwa', required=True)
124.    #Pfad zur editierten Referenzdatei fuer die Benutzung der GATK Programme
125.    inputparser.add_argument('-g','--refgatkkaryotypic', action='store',type=argparse.FileType('r'),
    dest='refgatkkaryotypic_file', help='refgatkkaryotypic file prepared by BWA edit manually for karyotic
    order', required=True)
126.    #Pfad zur Indel Datei welche im GATK bundle verfuegbar ist
127.    inputparser.add_argument('-i','--knownindels', action='store',type=argparse.FileType('r'),
    dest='known_1000G_indel_file', help='known indel file downloaded from GATK bundle', required=True)
128.    #Pfad zur dbSNP Datei welche im GATK bundle verfuegbar ist
129.    inputparser.add_argument('-d','--knowndbsnp', action='store',type=argparse.FileType('r'),
```

```
dest='known_dbsnp_file', help='known snp file downloaded from GATK bundle', required=True)
#Parameter einlesen
args = inputparser.parse_args()

#Ueberpruefe ob Ausgabeordner existiert
if not os.path.isdir(args.output_path):
    #Ansonsten Abbruch
    logging.error("Directory " + args.output_path + " not available.")
    sys.exit(0)
else:
    #check Softwaretools config file
    if Monitor_module.check_for_programs(required_programs, CONFIG_SOFTWARE_PARSER_PATH):
        #Einlesen und ueberpruefen der Parameter Konfigurationsdatei
        Parameter_parser = Monitor_module.check_for_parameters(CONFIG_PARAMETER_PARSER_PATH)
        #Eingabe der Fastq oder bam Datein
        sample_matrix = insert_sample_matrix(args.insert_bam_files)
        #Auswahl der library Datei
        library = insert_library()
        #Frage ob die Analyse starten soll per Dialog
        start_analysis=tkMessageBox.askyesno("Analysis", "Do you want to start analysis?")
        root_window.destroy()
        #Die Analyse in einem try except Block durchfuehren
        if start_analysis:
            try:
                #Pro bam Datei oder pro Fastq Dateienpaar
                for sample in sample_matrix:
                    #Wenn Fastq Dateien eingegeben wurden
                    if not args.insert_bam_files:
                        #Erzeuge das Alignment
                        if not Command_module.create_alignment(sample, sample_matrix[sample], library, CONFIG_SOFTWARE_PARSER_PATH, Parameter_parser, args.cores ,args.refmultifasta_file, args.output_path):
                            logging.warn("Couldn't execute \"create_alignment\" at :"+ sample)
                            sys.exit(0)
                        #Konvertierung der sam Datei zur bam Datei
                        if not Command_module.convert_sam_to_bam(sample, library, CONFIG_SOFTWARE_PARSER_PATH, Parameter_parser, args.output_path):
                            logging.warn("Couldn't execute \"convert_sam_to_bam\" at :"+ sample)
                            sys.exit(0)
                    #Erzeuge Statistiken zu der ersten bam Datei
                    if not Command_module.create_stats(os.path.join(args.output_path, str(os.path.basename(sample).split('.')[0]) + '.bam'), args.output_path, 'all_mapped_', CONFIG_SOFTWARE_PARSER_PATH):
                        logging.warn("Couldn't execute \"create_stats_all\" at :"+ sample)
                        sys.exit(0)
                    #Sortiere die bam Datei
                    if not Command_module.sort_alignment(sample, args.output_path, Parameter_parser, CONFIG_SOFTWARE_PARSER_PATH):
                        logging.warn("Couldn't execute \"sort_alignment\" at :"+ sample)
                        sys.exit(0)
                    #Entferne die Duplikate der Reads aus der bam Datei
                    if not Command_module.remove_duplicates(sample, args.output_path, Parameter_parser, CONFIG_SOFTWARE_PARSER_PATH):
                        logging.warn("Couldn't execute \"sort_alignment\" at :"+ sample)
                        sys.exit(0)
                    #Erzeuge Statistiken zu der bearbeiteten bam Datei
                    if not Command_module.create_stats(os.path.join(args.output_path, 'remove_duplicates_' +
```

```
str(os.path.basename(sample).split(".")[0]) + '.bam'), args.output_path, 'removed_duplicates_', CONFIG_SOFTWARE_PARSER_PATH):
                    logging.warn("Couldn't execute \"create_stats_rmdup\" at :"+ sample)
                    sys.exit(0)
                #Erzeuge eine index Datei zur bam Datei
                if not Command_module.create_index(os.path.join(args.output_path, 'remove_duplicates_' + str(os.path.basename(sample).split(".")[0]) + '.bam'), args.output_path, CONFIG_SOFTWARE_PARSER_PATH):
                    logging.warn("Couldn't execute \"create_index\" at :"+ sample)
                    sys.exit(0)
                #Erzeuge eine bam Datei welche das Aligment zur angegebenen library enthaelt
                if not Command_module.create_intersection(sample, library, Parameter_parser, args.output_path, CONFIG_SOFTWARE_PARSER_PATH):
                    logging.warn("Couldn't execute \"create_index\" at :"+ sample)
                    sys.exit(0)
                #Erzeuge Statistiken zu der bam Datei welche das Aligment zur angegebenen library enthaelt
                if not Command_module.create_stats(os.path.join(args.output_path, 'intersection_ontarget_' + str(os.path.basename(sample).split(".")[0]) + '.bam'), args.output_path, 'ontarget_', CONFIG_SOFTWARE_PARSER_PATH):
                    logging.warn("Couldn't execute \"create_stats_ontarget\" at :"+ sample)
                    sys.exit(0)
                #Erzeuge die indel_target Datei fuer das Realignment
                if not Command_module.create_indel_target_realignments(sample, args.output_path, args.refgatkkaryotypic_file, args.known_1000G_indel_file, Parameter_parser, CONFIG_SOFTWARE_PARSER_PATH):
                    logging.warn("Couldn't execute \"create_indel_target_realignments\" at :"+ sample)
                    sys.exit(0)
                #Fuehre das Indel Realignment durch
                if not Command_module.create_indel_realignment(sample, args.output_path, args.refgatkkaryotypic_file, Parameter_parser, CONFIG_SOFTWARE_PARSER_PATH):
                    logging.warn("Couldn't execute \"create_indel_realignments\" at :"+ sample)
                    sys.exit(0)
                #Erzeuge die Rekalibrierungsdatei fuer die Basen
                if not Command_module.create_base_recalibration(sample, args.output_path, args.refgatkkaryotypic_file, args.known_dbsnp_file, Parameter_parser, CONFIG_SOFTWARE_PARSER_PATH):
                    logging.warn("Couldn't execute \"create_base_recalibration\" at :"+ sample)
                    sys.exit(0)
                #Fuehre die Rekalibrieung der Basen durch
                if not Command_module.create_recalibrated_bases(sample, args.output_path, args.refgatkkaryotypic_file, Parameter_parser, CONFIG_SOFTWARE_PARSER_PATH):
                    logging.warn("Couldn't execute \"create_recalibrated_bases\" at :"+ sample)
                    sys.exit(0)
                #Erzeuge eine bedgraph Datei aus der realignierten und rekalibrieten bam Datei
                if not Command_module.create_bedgraph(os.path.join(args.output_path, str(os.path.basename(sample).split(".")[0]) + '_removed_duplicates_realigned_recal.bam'), args.output_path, Parameter_parser, CONFIG_SOFTWARE_PARSER_PATH):
                    logging.warn("Couldn't execute \"create_bedgraph_file\" at :"+ sample)
                    sys.exit(0)
                #Erzeuge die vcf Datei mit dem HaplotypeCaller
                if not Command_module.create_haplotypecalling(sample, args.output_path, args.refgatkkaryotypic_file, args.known_1000G_indel_file, library, Parameter_parser, CONFIG_SOFTWARE_PARSER_PATH):
                    logging.warn("Couldn't execute \"create_calling_file_by_haplotypecaller\" at :"+ sample)
                    sys.exit(0)
            #Bei Ausnahmen die Meldung als Fehler ausgeben
            except Exception as e:
```

```
219.                logging.error(traceback.format_exc())
220.          #Ansonsten Abbruch
221.          else:
222.             sys.exit(0)
223.       #Ansonsten Abbruch
224.       else:
225.          sys.exit(0)
```

Quellcode zur Monitor_module-Datei:

```
1.    '''
2.    Created on 01.08.2015
3.
4.    @author: Jordan Pischimarov
5.
6.    @note: Sammlung von Methoden zur Ueberpruefung und Bearbeitung der Parameter und Programme
7.
8.    '''
9.    import os, sys, ConfigParser, tkFileDialog, logging, Tkinter
10.   configfile_parser = ConfigParser.SafeConfigParser()
11.
12.   '''
13.   Gib die gespeicherte option in der section der Datei zurueck
14.   '''
15.   def get_option_in_section(section, option, CONFIGFILE_PATH):
16.      configfile_parser = ConfigParser.SafeConfigParser()
17.      configfile_parser.read(CONFIGFILE_PATH)
18.      return configfile_parser.get(section, option)
19.
20.   '''
21.   Einlesen der gesamten Konfigurationsdatei und Bereitstellung dieser mittel ConfigParser Modul
22.   '''
23.   def get_configparser_file(CONFIGFILE_PATH):
24.      return configfile_parser
25.
26.   '''
27.   Ausgabe der Konfigurationsdatei
28.   '''
29.   def show_settings_in_configfile(CONFIGFILE_PATH):
30.      configfile_parser.read(CONFIGFILE_PATH)
31.      for section in configfile_parser.sections():
32.         tmp_options=''
33.         for option in configfile_parser.options(section):
34.            tmp_options=tmp_options+'\n'+option+'\t'+configfile_parser.get(section, option)
35.         logging.info("section: "+section+"\nwith options and values: "+tmp_options)
36.
37.   '''
38.   Hinzufuegen der section in die Konfigurationsdatei sowie Aktualisierung der Datei
39.   '''
40.   def update_configfile_section(section, CONFIGFILE_PATH):
41.      with open(CONFIGFILE_PATH , "r+") as configfile:
42.         logging.info("updating config file section: "+section)
43.         configfile_parser.add_section(section)
44.         configfile_parser.write(configfile)
45.      configfile.close()
46.
47.   '''
```

```
48.  Hinzufuegen der option in die Konfigurationsdatei sowie Ueberpruefung von Pfaden und Aktualisierung
     der Datei
49.  '''
50.  def update_configfile_option(section, option, CONFIGFILE_PATH, root_window):
51.     with open(CONFIGFILE_PATH , "r+") as configfile:
52.        logging.info("updating config file option: "+option+" in section: "+section)
53.        if option == "path":
54.           new_option=tkFileDialog.askdirectory(parent=root_window, title="Please select "+option+" to
     "+section+" for the analysis.", mustexist=True)
55.           if not(new_option =='NONE' or new_option ==''):
56.              configfile_parser.set(section, option, new_option)
57.           else:
58.              logging.error("Directory couldn't be selected. Abort!")
59.              sys.exit(0)
60.        else:
61.           configfile_parser.set(section, option, "{0}".format(raw_input("please enter {0}:
     ".format(option))))
62.        configfile_parser.write(configfile)
63.     configfile.close()
64.
65.  '''
66.  Solange den Pfad aendern bis ein verfuegbarer Pfad gesetzt wurde
67.  '''
68.  def change_path(section, option, path):
69.     if not os.path.isdir(path):
70.        logging.info("Path is not available, try it again: "+option+" in section: "+section)
71.        path = "{0}".format(raw_input("please enter {0}: ".format(option)))
72.        change_path(section, option, path)
73.     else:
74.        return str(path)
75.
76.  '''
77.  Schreibe oder ueberschreibe die Eigenschaften in der Konfigurationsdatei
78.  '''
79.  def write_configfile(sections, options, CONFIGFILE_PATH, root_window):
80.     with open(CONFIGFILE_PATH , "w") as configfile:
81.        logging.info("writing config file ...")
82.        for section in sections:
83.           logging.info("section: "+section+" created!")
84.           configfile_parser.add_section(section)
85.           for option in options:
86.              new_option=tkFileDialog.askdirectory(parent=root_window, title="Please select "+option+" to
     "+section+" for the analysis.", mustexist=True)
87.              if not(new_option =='NONE' or new_option ==''):
88.                 configfile_parser.set(section, option, new_option)
89.              else:
90.                 logging.error("Directory couldn't be selected. Abort!")
91.                 sys.exit(0)
92.              #configfile_parser.set(section, option, "{0}".format(raw_input("please enter "+option+" for
     "+section+": ".format(option))))
93.        configfile_parser.write(configfile)
94.     configfile.close()
95.
96.  '''
97.  Loesche die section in der Konfigurationsdatei
98.  '''
99.  def delete_configfile_section(section, CONFIGFILE_PATH):
```

```
100.     configfile_parser.remove_section(section)
101.     with open(CONFIGFILE_PATH , "w") as configfile:
102.         configfile_parser.write(configfile)
103.     configfile.close()
104. 
105. '''
106. Loesche die option in der Konfigurationsdatei
107. '''
108. def delete_configfile_option(section, option, CONFIGFILE_PATH):
109.     configfile_parser.remove_option(section, option)
110.     with open(CONFIGFILE_PATH , "w") as configfile:
111.         configfile_parser.write(configfile)
112.     configfile.close()
113. 
114. '''
115. Einlesen und ueberpruefen der Konfigurationsdatei mittels ConfigParser Modul
116. '''
117. def read_and_check_configfile(sections, options, CONFIGFILE_PATH, root_window):
118.     logging.info("reading config file {}".format(CONFIGFILE_PATH))
119.     configfile_parser.read(CONFIGFILE_PATH)
120.     #already written sections
121.     for section in configfile_parser.sections():
122.         #already written options in this default section
123.         for option in configfile_parser.options(section):
124.             if option == "path" and not os.path.isdir(configfile_parser.get(section, option)):
125.                 update_configfile_option(section, option, CONFIGFILE_PATH, root_window)
126.     #check if all default sections are available
127.     for section in sections:
128.         #if default section not included than update config file
129.         if not section in configfile_parser.sections():
130.             logging.warning("config file doesn\'t contain default section: "+section)
131.             update_configfile_section(section, CONFIGFILE_PATH)
132.             #also include all default options in this created default section
133.             for option in options:
134.                 update_configfile_option(section, option, CONFIGFILE_PATH, root_window)
135.         else:
136.             #for each default section included in the config file check for default options are available
137.             for option in options:
138.                 #if not available than include default option
139.                 if not option in configfile_parser.options(section):
140.                     update_configfile_option(section, option, CONFIGFILE_PATH, root_window)
141. 
142. 
143. '''
144. Ueberpruefe die Konfigurationsdatei welche die benoetigten Programme beinhaltet
145. '''
146. def check_for_programs(required_programs, CONFIGFILE_PATH):
147.     root_window=Tkinter.Tk()
148.     root_window.withdraw()
149.     root_window.focus_force()
150. 
151.     CONFIGFILE_PATH_SECTIONS = required_programs
152.     CONFIGFILE_PATH_OPTIONS = ["path"]
153. 
154.     logging.basicConfig(level=logging.INFO, format='%(levelname)s:%(message)s')
155. 
156. 
```

```
157.     '''
158.     Ueberpruefe die Konfigurationsdatei ob sie existiert
159.     '''
160.     if os.path.exists(CONFIGFILE_PATH):
161.         #Wenn ja ueberpruefen und einlesen
162.         logging.info("following configfile exists")
163.         show_settings_in_configfile(CONFIGFILE_PATH)
164.         read_and_check_configfile(CONFIGFILE_PATH_SECTIONS, CONFIGFILE_PATH_OPTIONS,
    CONFIGFILE_PATH, root_window)
165.         return True
166.     else:
167.         #Wenn nein erstellen ueberpruefen und einlesen
168.         logging.info("configfile not exists")
169.         write_configfile(CONFIGFILE_PATH_SECTIONS, CONFIGFILE_PATH_OPTIONS, CONFIGFILE_PATH,
    root_window)
170.         read_and_check_configfile(CONFIGFILE_PATH_SECTIONS, CONFIGFILE_PATH_OPTIONS,
    CONFIGFILE_PATH, root_window)
171.         logging.info("created following configfile")
172.         show_settings_in_configfile(CONFIGFILE_PATH)
173.         return True
174.
175. '''
176. Einlesen der Konfigurationsdatei mit den Parametern
177. '''
178. def check_for_parameters(DEFAULT_PARAMETER_CONFIGFILE):
179.     configfile_parameter_parser = ConfigParser.SafeConfigParser()
180.     configfile_parameter_parser.read(DEFAULT_PARAMETER_CONFIGFILE)
181.     return configfile_parameter_parser
```

Quellcode zur Command_module-Datei:

```
1.  '''
2.  Created on 04.08.2015
3.
4.  @author: Jordan Pischimarov
5.
6.  @note: Sammlung von Methoden zur Durchfuehrung der Analyse
7.  '''
8.  import Monitor_module, os, subprocess, logging
9.
10. '''
11. Methoden zur Durchfuehrung
12. '''
13. #Erstellung des Alignments
14. def create_alignment(sample_1, sample_2, library, CONFIG_SOFTWARE_PARSER_PATH,
    Parameter_parser, cores, multifasta, output_path):
15.     #sample namen aus Pfad extrahieren
16.     sample_name=os.path.basename(sample_1).split(".")[0]
17.     #Ausfuehrbares Kommando zusammenstellen
18.     alignment_command=os.path.join(Monitor_module.get_option_in_section('BWA', 'path',
    CONFIG_SOFTWARE_PARSER_PATH), 'bwa mem -t ') + str(cores) + ' ' +Parameter_parser.get('BWA',
    'param_M') + ' -R \'@RG\\t'+Parameter_parser.get('BWA', 'readgroup_id')+ str(sample_name) +
    '\\t'+Parameter_parser.get('BWA', 'readgroup_cn')+'\\t'+Parameter_parser.get('BWA',
    'readgroup_lb')+'\\t'+Parameter_parser.get('BWA', 'readgroup_pl')+'\\t'+Parameter_parser.get('BWA',
    'readgroup_pu')+'\\t'+Parameter_parser.get('BWA', 'readgroup_sm') + str(sample_name) + '\' ' +
    str(multifasta.name) + ' ' + str(sample_1) + ' ' + str(sample_2) + ' > ' + str(output_path) + str(sample_name)
    + '.sam'
```

```
19.     #Wenn die Durchfuehrung erfolgreich war True zurueck geben
20.     if execute_analysis_step(alignment_command) == 0:
21.       return True
22.     #Ansonsten False zurueck geben
23.     else:
24.       return False
25.   #Konvertierung von sam zu bam Dateien
26.   def convert_sam_to_bam(sample_1, library, CONFIG_SOFTWARE_PARSER_PATH, Parameter_parser,
      output_path):
27.     #sample namen aus Pfad extrahieren
28.     sample_name=os.path.basename(sample_1).split(".")[0]
29.     #Ausfuehrbares Kommando zusammenstellen
30.     convert_command=os.path.join(Monitor_module.get_option_in_section('samtools', 'path',
      CONFIG_SOFTWARE_PARSER_PATH), 'samtools view ') + Parameter_parser.get('samtools_view', 'param')
      + ' ' + str(output_path) + str(sample_name) + '.sam > ' + str(output_path) + str(sample_name) + '.bam'
31.     #Wenn die Durchfuehrung erfolgreich war True zurueck geben
32.     if execute_analysis_step(convert_command) == 0:
33.       return True
34.     #Ansonsten False zurueck geben
35.     else:
36.       return False
37.   #Statistiken des Alignments erstellen
38.   def create_stats(file_bam, output_path, file_type, CONFIG_SOFTWARE_PARSER_PATH):
39.     bam_name=os.path.basename(file_bam).split(".")[0]
40.     stats_command=os.path.join(Monitor_module.get_option_in_section('samtools', 'path',
      CONFIG_SOFTWARE_PARSER_PATH), 'samtools flagstat ') + file_bam + ' > ' + str(output_path) +
      str(file_type) + str(bam_name) + '.stats'
41.     #Wenn die Durchfuehrung erfolgreich war True zurueck geben
42.     if execute_analysis_step(stats_command) == 0:
43.       return True
44.     #Ansonsten False zurueck geben
45.     else:
46.       return False
47.   #Das Alignment nach Koordinaten sortieren
48.   def sort_alignment(sample_1, output_path, Parameter_parser, CONFIG_SOFTWARE_PARSER_PATH):
49.     #sample namen aus Pfad extrahieren
50.     sample_name=os.path.basename(sample_1).split(".")[0]
51.     #Ausfuehrbares Kommando zusammenstellen]
52.     sort_command='java ' + Parameter_parser.get('java', 'Xmx') + ' -jar ' +
      os.path.join(Monitor_module.get_option_in_section('Picard', 'path',
      CONFIG_SOFTWARE_PARSER_PATH), 'SortSam.jar ') + ' INPUT=' + str(output_path) + str(sample_name) +
      '.bam OUTPUT=' + str(output_path) + 'sorted_' + str(sample_name) + '.bam ' +
      Parameter_parser.get('Picard_sortsam', 'param')
53.     #Wenn die Durchfuehrung erfolgreich war True zurueck geben
54.     if execute_analysis_step(sort_command) == 0:
55.       return True
56.     #Ansonsten False zurueck geben
57.     else:
58.       return False
59.   #Duplikate von Reads entfernen
60.   def remove_duplicates(sample_1, output_path, Parameter_parser,
      CONFIG_SOFTWARE_PARSER_PATH):
61.     #sample namen aus Pfad extrahieren
62.     sample_name=os.path.basename(sample_1).split(".")[0]
63.     #Ausfuehrbares Kommando zusammenstellen
64.     sort_command='java ' + Parameter_parser.get('java', 'Xmx') + ' -jar ' +
      os.path.join(Monitor_module.get_option_in_section('Picard', 'path',
```

```
CONFIG_SOFTWARE_PARSER_PATH), 'MarkDuplicates.jar ') + ' INPUT=' + str(output_path) + 'sorted_' +
str(sample_name) + '.bam OUTPUT=' + str(output_path) + 'remove_duplicates_' + str(sample_name) +
'.bam ' + Parameter_parser.get('Picard_remove_duplicates', 'param') + str(output_path) +
str(sample_name) + '.metrics'
    #Wenn die Durchfuehrung erfolgreich war True zurueck geben
    if execute_analysis_step(sort_command) == 0:
        return True
    #Ansonsten False zurueck geben
    else:
        return False
#Eine index Datei zur bam Datei erstellen
def create_index(file_bam, output_path, CONFIG_SOFTWARE_PARSER_PATH):
    #Ausfuehrbares Kommando zusammenstellen
    index_command=os.path.join(Monitor_module.get_option_in_section('samtools', 'path',
CONFIG_SOFTWARE_PARSER_PATH), 'samtools index ') + file_bam
    #Wenn die Durchfuehrung erfolgreich war True zurueck geben
    if execute_analysis_step(index_command) == 0:
        return True
    #Ansonsten False zurueck geben
    else:
        return False
#Das Alignment auf die Regionen der library Datei limitieren
def create_intersection(sample_1, library, Parameter_parser, output_path,
CONFIG_SOFTWARE_PARSER_PATH):
    #sample namen aus Pfad extrahieren
    sample_name=os.path.basename(sample_1).split(".")[0]
    #Ausfuehrbares Kommando zusammenstellen
    intersection_command=str(os.path.join(Monitor_module.get_option_in_section("bedtools", "path",
CONFIG_SOFTWARE_PARSER_PATH), "bedtools") + ' intersect ' +
Parameter_parser.get('bedtools_intersection','param')+ ' ' + str(output_path) + 'remove_duplicates_' +
str(sample_name) + '.bam -b ' + library + ' > ' + str(output_path) + 'intersection_ontarget_' +
str(sample_name) + '.bam')
    #Wenn die Durchfuehrung erfolgreich war True zurueck geben
    if execute_analysis_step(intersection_command) == 0:
        return True
    #Ansonsten False zurueck geben
    else:
        return False
#Erzeuge die Indel_target Datei fuer das Realignment
def create_indel_target_realignments(sample_1, output_path, reference_gatk_karyo, known_indels,
Parameter_parser, CONFIG_SOFTWARE_PARSER_PATH):
    #sample namen aus Pfad extrahieren
    sample_name=os.path.basename(sample_1).split(".")[0]
    #Ausfuehrbares Kommando zusammenstellen
    sort_command='java ' + Parameter_parser.get('java', 'Xmx') + ' -jar ' +
os.path.join(Monitor_module.get_option_in_section('GATK', 'path',
CONFIG_SOFTWARE_PARSER_PATH), 'GenomeAnalysisTK.jar ') +
Parameter_parser.get('GATK_reaglin_creator', 'param') + ' ' + known_indels.name + ' -I ' +
str(output_path) + 'remove_duplicates_' + str(sample_name) + '.bam -o ' + str(output_path) +
str(sample_name) + '.intervals -R ' + reference_gatk_karyo.name
    #Wenn die Durchfuehrung erfolgreich war True zurueck geben
    if execute_analysis_step(sort_command) == 0:
        return True
    #Ansonsten False zurueck geben
    else:
        return False
#Durchfuehrung des Realignments
```

```
106. def create_indel_realignment(sample_1, output_path, reference_gatk_karyo, Parameter_parser, CONFIG_SOFTWARE_PARSER_PATH):
107.     #sample namen aus Pfad extrahieren
108.     sample_name=os.path.basename(sample_1).split(".")[0]
109.     #Ausfuehrbares Kommando zusammenstellen
110.     sort_command='java ' + Parameter_parser.get('java', 'Xmx') + ' -jar ' + os.path.join(Monitor_module.get_option_in_section('GATK', 'path', CONFIG_SOFTWARE_PARSER_PATH), 'GenomeAnalysisTK.jar') + Parameter_parser.get('GATK_reagliner', 'param') + ' ' + str(output_path) + str(sample_name) + '.intervals -I ' + str(output_path) + 'remove_duplicates_' + str(sample_name) + '.bam -o ' + str(output_path) + str(sample_name) + '_realigned.bam -R ' + reference_gatk_karyo.name
111.     #Wenn die Durchfuehrung erfolgreich war True zurueck geben
112.     if execute_analysis_step(sort_command) == 0:
113.         return True
114.     #Ansonsten False zurueck geben
115.     else:
116.         return False
117. #Berechnung der zu rekalibrierenden Basen
118. def create_base_recalibration(sample_1, output_path, reference_gatk_karyo, known_dbsnp, Parameter_parser, CONFIG_SOFTWARE_PARSER_PATH):
119.     #sample namen aus Pfad extrahieren
120.     sample_name=os.path.basename(sample_1).split(".")[0]
121.     #Ausfuehrbares Kommando zusammenstellen
122.     sort_command='java ' + Parameter_parser.get('java', 'Xmx') + ' -jar ' + os.path.join(Monitor_module.get_option_in_section('GATK', 'path', CONFIG_SOFTWARE_PARSER_PATH), 'GenomeAnalysisTK.jar') + Parameter_parser.get('GATK_base_recalibrator', 'param') + ' ' + known_dbsnp.name + ' -I ' + str(output_path) + str(sample_name) + '_realigned.bam -o ' + str(output_path) + str(sample_name) + '_recal_data.grp -R ' + reference_gatk_karyo.name
123.     #Wenn die Durchfuehrung erfolgreich war True zurueck geben
124.     if execute_analysis_step(sort_command) == 0:
125.         return True
126.     #Ansonsten False zurueck geben
127.     else:
128.         return False
129. #Die Rekalibrierung der Basen durchfuehren
130. def create_recalibrated_bases(sample_1, output_path, reference_gatk_karyo, Parameter_parser, CONFIG_SOFTWARE_PARSER_PATH):
131.     #sample namen aus Pfad extrahieren
132.     sample_name=os.path.basename(sample_1).split(".")[0]
133.     #Ausfuehrbares Kommando zusammenstellen
134.     sort_command='java ' + Parameter_parser.get('java', 'Xmx') + ' -jar ' + os.path.join(Monitor_module.get_option_in_section('GATK', 'path', CONFIG_SOFTWARE_PARSER_PATH), 'GenomeAnalysisTK.jar') + Parameter_parser.get('GATK_print_recal_base', 'param') + ' ' + str(output_path) + str(sample_name) + '_recal_data.grp -I ' + str(output_path) + str(sample_name) + '_realigned.bam -o ' + str(output_path) + str(sample_name) + '_removed_duplicates_realigned_recal.bam -R ' + reference_gatk_karyo.name
135.     #Wenn die Durchfuehrung erfolgreich war True zurueck geben
136.     if execute_analysis_step(sort_command) == 0:
137.         return True
138.     #Ansonsten False zurueck geben
139.     else:
140.         return False
141. #Erstelle eine bedgraph Datei zur bam Datei
142. def create_bedgraph(sample_1, output_path, Parameter_parser, CONFIG_SOFTWARE_PARSER_PATH):
143.     #sample namen aus Pfad extrahieren
144.     sample_name=os.path.basename(sample_1).split(".")[0]
```

```
145.    #Ausfuehrbares Kommando zusammenstellen
146.    bamgraph_command=str(os.path.join(Monitor_module.get_option_in_section("bedtools", "path", CONFIG_SOFTWARE_PARSER_PATH), "bedtools") + ' genomecov ' + Parameter_parser.get('bedtools_bedgraph','param')+ ' ' + sample_1 + " > " + output_path + sample_name + ".bedgraph")
147.    #Wenn die Durchfuehrung erfolgreich war True zurueck geben
148.    if execute_analysis_step(bamgraph_command) == 0:
149.      return True
150.    #Ansonsten False zurueck geben
151.    else:
152.      return False
153. #Durchfuehrung des Calling mittels des HaplotypeCallers
154. def create_haplotypecalling(sample_1, output_path, reference_gatk_karyo, known_dbsnp, library, Parameter_parser, CONFIG_SOFTWARE_PARSER_PATH):
155.    #sample namen aus Pfad extrahieren
156.    sample_name=os.path.basename(sample_1).split(".")[0]
157.    #Ausfuehrbares Kommando zusammenstellen
158.    sort_command='java ' + Parameter_parser.get('java', 'Xmx') + ' -jar ' + os.path.join(Monitor_module.get_option_in_section('GATK', 'path', CONFIG_SOFTWARE_PARSER_PATH), 'GenomeAnalysisTK.jar ') + Parameter_parser.get('GATK_haplotypecalling', 'param') + ' ' + known_dbsnp.name + ' -I ' + str(output_path) + str(sample_name) + '_removed_duplicates_realigned_recal.bam -o ' + str(output_path) + str(sample_name) + '_snp_indel.raw.vcf -L ' + library + ' -R ' + reference_gatk_karyo.name
159.    #Wenn die Durchfuehrung erfolgreich war True zurueck geben
160.    if execute_analysis_step(sort_command) == 0:
161.      return True
162.    #Ansonsten False zurueck geben
163.    else:
164.      return False
165. #Ausfuehrung des zusammengestellten Kommandos
166. def execute_analysis_step(command):
167.    #Aktuelle Durchfuehrung ausgeben
168.    logging.info("Running: \n" + command + " ...")
169.    myprocess = subprocess.Popen(command, shell=True ,stdout=subprocess.PIPE)
170.    (sout, serr) = myprocess.communicate()
171.    #Ausgabe der Ausgabe des aktuellen Kommandos
172.    for line in sout.split('\n'):
173.      if line.strip().startswith('Bytes received'):
174.        print ("This workstation received %s bytes." % line.strip().split(' ')[-1])
175.    #auf das Ende des Prozesses waren
176.    myprocess.wait()
177.    return myprocess.returncode
```

9.1.2 Genemapping

Im folgendem der kommentierte Quellcode des Python-Skripts Genmapping.

```
1.  #!/usr/bin/python
2.  import os, sys, argparse
3.
4.  ''' Syntax der Ausgabe-Datei.
5.  Beschreibung der Syntax der zu generierenden Ausgabe:
6.  Mehrfache hits des gerade zu durchsuchenden Gens werden durch '|' getrennt
7.  und mehrfache Gene in der Eingabe-Datei werden dann durch ; von deren jeweiligen Hits getrennt.
8.  Ein Beispiel waere hierzu die Gen B_1 und B_2 mit den Hits zu B_1 mit H_1_1, H_1_2
```

```
9.      und B_2 mit H_2. Die generierte Ausgabe waere dann
10.     'B_1;B_2 H_1_1|H_1_2;H_2' usw...
11.     '''
12.
13.     #Parameter ueberpruefen mit argparse.
14.     inputparser = argparse.ArgumentParser(description='Compare gene names to HGNC-database entries')
15.     inputparser.add_argument('-i','--input',action='store',dest='gene_list_path',help='Input file, gene names listed separately',required=True)
16.     inputparser.add_argument('-o','--output',action='store',dest='output_path',help='Output file name',required=True)
17.     inputparser.add_argument('-hgnc','--hgnc_database',action='store',dest='HGNC_database_release_file_path',help='HGNC download with current amount of approved gene symbols', required=True)
18.     #default default='/home/jordan/Daten/database/HGNC/edit_hgnc_download_09_03_2015.txt'
19.     inputparser.add_argument('-hgnc_c','--hgnc_columns',action='store',dest='HGNC_gene_symbol_columns',help='HGNC file columns containing gene symbols. Separated by comma', required=True)
20.     #default default='1,4,6'
21.
22.     args = inputparser.parse_args()
23.
24.     #Den HGNC Datenbank Parameter nur mit den benoetigten Spalten Parameter erlauben
25.     if args.HGNC_database_release_file_path and args.HGNC_gene_symbol_columns is None:
26.         inputparser.error('Parameter hgnc_c is required. Please set the gene symbol columns separated by comma.')
27.
28.     #Die gesetzten Pfade ueberpruefen.
29.     if os.path.isfile(args.gene_list_path) and os.path.isfile(args.HGNC_database_release_file_path):
30.       #Zuerst den Ausgabepfad ueberpruefen
31.       try:
32.         output_file=open(args.output_path, 'w')
33.         output_file.close()
34.       except:
35.         inputparser.error('Could not create file: \"'+args.output_path+'\"')
36.         sys.exit()
37.
38.
39.       #Die Information der HGNC-Datenbank einlesen
40.       HGNC_database_release_file=open(args.HGNC_database_release_file_path, 'r')
41.       HGNC_database_release_file_content=HGNC_database_release_file.readlines()
42.
43.       #Initialisieren der Counter, Strings und Listen
44.       count=0
45.       line_list_output=[]
46.       line_list_compare=[]
47.       final_list=[]
48.       found_genes_columns=[]
49.       temp_attribute_array=[]
50.       temp_attribute_hit=[]
51.       column_amount=0
52.       title_HGNC=''
53.       title_input=''
54.
55.       #Die Zeilen der HGNC Datei durchgehen
56.       for line in HGNC_database_release_file_content:
57.         count=count+1
58.         #Wenn 1000 zeilen eingelesen dann die Anzahl ausgeben
```

```
59.         if count%1000 == 0:
60.             print '%d genes read' % (count)
61.         #Die Ueberschriften speichern
62.         if count == 1:
63.             title_HGNC=line
64.         #Die Ueberschriften nicht bearbeiten
65.         if count != 1:
66.             #Die folgenden Attribute der Zeile einlesen.
67.             #Approved Symbol=0 Previous Symbols=1(comma seperated) Previous Names=2
68.             #Synonyms=3(comma seperated) Chromosome=4 Accession Numbers=5 RefSeq IDs=6 Approved
    Name=7
69.             attributes_HGNC=line.split('\t')
70.             #Spaltenanzahl speichern
71.             column_amount=len(attributes_HGNC)
72.             #initialisieren
73.             all_names=''
74.             #Alle Attribute zusammenfuehren
75.             for attribute in attributes_HGNC:
76.                 all_names=all_names+attribute+'&'
77.             all_names=all_names.rstrip('&')
78.             #Leerzeichen entfernen
79.             all_names=all_names.replace(' ', '')
80.             #Letztes new line entfernen
81.             all_names=all_names.rstrip('\n')
82.             #Daten fuer formatierte Ausgabe speichern
83.             attributes_all_names=all_names.split('&')
84.             line_list_output.append(attributes_all_names)
85.             #Daten fuer den Vergleich bearbeiten und speichern
86.             #initialisieren
87.             all_names=''
88.             #Die Attribute der jeweiligen Spalten speichern
89.             #Beachte Array Index minus 1
90.             columns=args.HGNC_gene_symbol_columns.split(',')
91.             for column in columns:
92.                 all_names=all_names+attributes_HGNC[int(column)-1]+','
93.             #Leerzeichen dazwischen loeschen
94.             all_names=all_names.replace(' ', '')
95.             #Letztes Komma entfernen
96.             all_names=all_names.rstrip(',')
97.             #Nur die Namen ohne ,
98.             only_all_names=all_names.split(',')
99.             line_list_compare.append(only_all_names)
100.
101.    #Input Gennamen einlesen
102.    input_file=open(args.gene_list_path, 'r')
103.    input_file_content=input_file.readlines()
104.
105.    #Ausgabe wie viele lines pro datei
106.    print 'input lines:\t' + str(len(input_file_content))
107.    print 'reference lines:\t' + str(len(line_list_compare))
108.
109.    #Initialisieren der Counter und Strings
110.    index=0
111.    found_count=0
112.    count_line=0
113.    tmp_founded=''
114.    tmp_more_input_genes_founded=''
```

```
115.
116.    #Fuer jede Zeile im input beziehungsweise Gen Namen
117.    for line in input_file_content:
118.       #print line
119.       count_line=count_line+1
120.       #Wenn 1000 zeilen verarbeitet dann die Anzahl ausgeben
121.       if count_line%1000 == 0:
122.          print '%d genes done' % (count_line)
123.       #Die Ueberschriften speichern
124.       if count_line == 1:
125.          title_input=line
126.       #Die Ueberschriften nicht bearbeiten
127.       if count_line != 1:
128.          #Pro Gen neu initialisieren
129.          index=0
130.          found_count=0
131.          tmp_founded=''
132.          tmp_more_input_genes_founded=''
133.          gene_names=line.strip().split(',')
134.          gene_count=0
135.
136.          if len(gene_names) == 0:
137.             final_list.append('\n')
138.
139.          #Wenn nur ein Gen in der Zeile dann folgende Bearbeitung
140.          if len(gene_names) == 1:
141.             #Alle Gen Namen durch gehen, sollte nur einer sein
142.             for genes in gene_names:
143.                #Die Anzahl der gen Namen zaehlen
144.                gene_count=gene_count+1
145.                #Wenn Zeile nicht leer dann
146.                if genes != '':
147.                   #zurueck setzen des hit_counter
148.                   found_count=0
149.                   #Alle verfuegbaren Gen Namen aus HGNC durch gehen
150.                   for counter in range(len(line_list_compare)):
151.                      #Wenn ein Name gefunden wurde dann
152.                      if genes in line_list_compare[counter]:
153.                         #counter hoch setzen, hit zaehlen
154.                         found_count=found_count+1
155.                         index=0
156.                         #Wenn hit zum erstenmal gefunden dann
157.                         if found_count == 1:
158.                            #Alle Attribute der Referenz zum hit zum ersten mal hinzufuegen
159.                            for i in line_list_output[counter]:
160.                               #Alle Attribute Spaltenweise speichern
161.                               found_genes_columns.append(str(i))
162.                            #Alle Attribute des gefundenen Gens in temporaeren string speichern
163.                            tmp_founded=tmp_founded+'&'.join(str(hits) for hits in found_genes_columns)
164.                            tmp_founded=tmp_founded.rstrip('&')
165.                            tmp_founded=tmp_founded.rstrip('\n')
166.                         #Wenn es mehrere hits zu einem Gene Namen gibt, dann mit : trennen!
167.                         else:
168.                            temp_string_array=[]
169.                            temp_string_array=found_genes_columns
170.                            temp_string_array.reverse()
171.                            found_genes_columns=[]
```

```
172.                        #Alle Attribute der Referenz zum hit erneut hinzufuegen, Attribute der
    unterschiedlichen hits mit : trennen!
173.                        for i in line_list_output[counter]:
174.                           #temporaeren String leeren
175.                           temp_string=''
176.                           temp_string=str(temp_string_array.pop())
177.                           found_genes_columns.append(str(str(temp_string)+'|'+str(i)))
178.                        tmp_founded=''
179.                        #Alle Attribute des gefundenen Gens in temporaeren string speichern
180.                        tmp_founded=tmp_founded+'&'.join(str(hits) for hits in found_genes_columns)
181.                        tmp_founded=tmp_founded.rstrip('&')
182.                        tmp_founded=tmp_founded.rstrip('\n')
183.                  #Wenn Gen Name gefunden wurde dann
184.                  if tmp_founded != '':
185.                     #Fuer die Ausgabe speichern
186.                     final_list.append(genes+'&'+tmp_founded)
187.                  else:
188.                     #Ansonsten Platzhalter mit Gen Name leeren, danach einfuegen
189.                     temp_genes=''
190.                     for temp_column in range(column_amount):
191.                        temp_genes=temp_genes+'&n_a'
192.                     final_list.append(genes+temp_genes)
193.                  tmp_founded=''
194.                  index=0
195.                  found_genes_columns=[]
196.                  temp_string_array=[]
197.            #Wenn mehrere Gene in der Zeile dann folgende Bearbeitung
198.            #Detailierte Kommentare siehe einzel Gen Bearbeitung
199.            if len(gene_names) > 1:
200.               #print 'mehrere Gene!\n'
201.               tmp_founded=''
202.               for genes in gene_names:
203.                  gene_count=gene_count+1
204.                  #tmp_founded=''
205.                  if genes != '':
206.                     found_count=0
207.                     for counter in range(len(line_list_compare)):
208.                        if genes in line_list_compare[counter]:
209.                           found_count=found_count+1
210.                           #Da mehrere Gene vorhanden sind, muessen auch die gefundenen Synonyme mit
    Semikolon getrennt werden
211.                           if found_count == 1:
212.                              #Alle Attribute der Referenz zum hit zum ersten mal hinzufuegen
213.                              for i in line_list_output[counter]:
214.                                 #Alle Attribute spaltenweise speichern
215.                                 found_genes_columns.append(str(i))
216.                              tmp_founded=tmp_founded+'&'.join(str(hits) for hits in found_genes_columns)
217.                              tmp_founded=tmp_founded.rstrip('&')
218.                              tmp_founded=tmp_founded.rstrip('\n')
219.                           #Wenn es mehrere hits zu einem Gen Namen gibt, dann mit : trennen!
220.                           else:
221.                              temp_string_array=[]
222.                              temp_string_array=found_genes_columns
223.                              temp_string_array.reverse()
224.                              found_genes_columns=[]
225.                              #Alle Attribute der Referenz zum hit erneut hinzufuegen, Attribute der
    unterschiedlichen hits mit : trennen!
```

```
226.                        #Da die Anzahl der Attribute nicht variiert kann die gleiche Schleife genutzt werden
227.                        for i in line_list_output[counter]:
228.                           temp_string=str(temp_string_array.pop())
229.                           found_genes_columns.append(str(str(temp_string)+'|'+str(i)))
230.                        tmp_founded=''
231.                        tmp_founded=tmp_founded+'&'.join(str(hits) for hits in found_genes_columns)
232.                        tmp_founded=tmp_founded.rstrip('&')
233.                        tmp_founded=tmp_founded.rstrip('\n')
234.                  #Nur ein Gen was bisher untersucht wurde
235.                  if gene_count == 1:
236.                     #print 'erster Hit!!\n'
237.                     if tmp_founded != '':
238.                        tmp_more_input_genes_founded=genes+'&'+tmp_founded
239.                     else:
240.                        temp_genes=''
241.                        for temp_column in range(column_amount):
242.                           temp_genes=temp_genes+'&n_a'
243.                        tmp_more_input_genes_founded=genes+temp_genes
244.                     index=0
245.                     temp_string_array=[]
246.                  #Weitere Gene in der Eingabezeile
247.                  else:
248.                     #print 'weitere Hits!!\n'
249.                     #Wenn hits vorhanden hinzufuegen
250.                     if tmp_founded != '':
251.                        tmp_founded=genes+'&'+tmp_founded
252.                        temp_for_one_gene_array=[]
253.                        temp_for_one_gene_array=tmp_founded.split('&')
254.                        temp_for_one_gene_array.reverse()
255.                        temp_string_array=[]
256.                        temp_string_array=tmp_more_input_genes_founded.split('&')
257.                        #temp_gene=temp_string_array.pop()
258.                        temp_string_array.reverse()
259.                        found_genes_columns=[]
260.                        #Alle Attribute der Referenz zum hit erneut hinzufuegen, Attribute der unterschiedlichen
    hits mit ; trennen!
261.                        for i in range(len(temp_string_array)):
262.                           #temporaeren String leeren
263.                           temp_string_one_gene=''
264.                           temp_string_one_gene=str(temp_for_one_gene_array.pop())
265.                           temp_string=''
266.                           temp_string=str(temp_string_array.pop())
267.                           found_genes_columns.append(str(str(temp_string)+';'+str(temp_string_one_gene)))
268.                        tmp_more_input_genes_founded=''
269.                        #Alle Attribute des gefundenen Gens in temporaeren String speichern hinzufuegen
270.                        tmp_more_input_genes_founded=tmp_more_input_genes_founded+'&'.join(str(hits)
    for hits in found_genes_columns)
271.                     #Ansonsten platzhalter hinzufuegen
272.                     else:
273.                        temp_for_one_gene_array=[]
274.                        for temp_column in range(column_amount):
275.                           temp_for_one_gene_array.append('n_a')
276.                        temp_for_one_gene_array.append(genes)
277.                        temp_string_array=[]
278.                        temp_string_array=tmp_more_input_genes_founded.split('&')
279.                        #temp_gene=temp_string_array.pop()
280.                        temp_string_array.reverse()
```

```
281.                    found_genes_columns=[]
282.                    #Alle Attribute der Referenz zum hit erneut hinzufuegen, Attribute der unterschiedlichen
    hits mit ; trennen!
283.                    for i in range(len(temp_string_array)):
284.                      #temporaeren String leeren
285.                      temp_string_one_gene=''
286.                      temp_string_one_gene=str(temp_for_one_gene_array.pop())
287.                      temp_string=''
288.                      temp_string=str(temp_string_array.pop())
289.                      found_genes_columns.append(str(str(temp_string)+';'+str(temp_string_one_gene)))
290.                    tmp_more_input_genes_founded=''
291.                    #Alle Attribute des gefundenen Gens in temporaeren String speichern hinzufuegen
292.                    tmp_more_input_genes_founded=tmp_more_input_genes_founded+'&'.join(str(hits)
    for hits in found_genes_columns)
293.                #Variablen wieder freigeben
294.                found_genes_columns=[]
295.                tmp_founded=''
296.              #Endergebnis der Gene/Gens mit den dazugehoerigen hit/hits speichern
297.              final_list.append(tmp_more_input_genes_founded)
298.              tmp_more_input_genes_founded=''
299.            found_genes_columns=[]
300.    input_file.close()
301.    HGNC_database_release_file.close()
302.
303.    #Ausgabe der gen Anzahlen
304.    print 'Amount of input elements:\t'+str(len(input_file_content))
305.    print 'Amount of founded elements:\t'+str(len(final_list))
306.    #Datei fuer die Ausgabe zum schreiben oeffnen
307.    output_file=open(args.output_path, 'a')
308.    #Ueberschriften aus input und HGNC Datei einfuegen
309.    output_file.write(title_input.rstrip('\n')+title_HGNC)
310.
311.    #Ausgabe der gefundenen Gene
312.    for index in range(len(final_list)):
313.      #die einzelenen Spalten trennen
314.      final_line=final_list[index].split('&')
315.      #Pro Spalte die Ausgabe aller Hits
316.      for columns in range(len(final_line)):
317.        output_file.write(final_line[columns]+'\t')
318.      #Neue Zeile beginnen
319.      output_file.write('\n')
320.    output_file.close()
321.  else:
322.    #Wenn Pfade nicht korrekt sind Fehlerausgabe
323.    inputparser.error('Input- or HGNC_database- file/path is incorrect!: \"'+args.gene_list_path+'\" or
    \"'+args.HGNC_database_release_file_path+'\"')
324.    sys.exit()
325.    print 'ERROR: Input- or HGNC_database- file/path is incorrect!'
```

9.1.3 Heatmap

Im folgendem der kommentierte Quellcode des Python-Skripts Heatmap.

```
1.    #!/usr/bin/python
2.    import os, sys
3.
```

```
4.  #Funktion des Skripts ist die Erstellung einer Heatmap in Abhaengigkeit des Gennamens, Mutation, Chromosom und Position.
5.  #Wichtig hierbei ist das in der Datei pro Eintrag das Chromosom und die zugehoerige Position eindeutig sein muessen.
6.  #Da die Funktion, Dateien mit unterschiedlicher Syntax in Abhaengigkeit der Parameter einzulesen,
7.  #nicht mit argparse bewerkstelligt werden konnte, wurde die Ueberpruefung der Parameter implementiert.
8.
9.  #Benoetigte Variablen setzen
10. single_parameter=0
11. multiple_parameter=0
12. output_parameter=0
13. single_parameter_columns=[]
14. multiple_parameter_columns=[]
15. single_files=[]
16. multiple_files=[]
17. outputfile=''
18. heatmap_titles=''
19. heatmap={}
20. chromosome_dict={}
21. position_dict={}
22. alteration_dict={}
23. samples=[]
24. all_samples_in_all_files=[]
25.
26. #Ueberpruefe welche parameter im Kommando verwendet werden
27. for parameter in sys.argv:
28.     if parameter == "-single":
29.         single_parameter=1
30.     if parameter == "-multiple":
31.         multiple_parameter=1
32.     if parameter == "-out":
33.         output_parameter=1
34. #Ueberpruefe ob ausreichend Argumente im Kommando vorhanden sind oder Hilfe benoetigt wird.
35. if len(sys.argv) <= 2 or sys.argv[1] == '-h':
36.     #Wenn nicht ausreichend Argumente vorhanden sind oder -h verwendet wird, dann folgende Ausgabe.
37.     print "\nmissing arguments to create heatmap!\n\n\tusage for single sample file: \n\t\tpython create_HeatMap.py -single [column_of_genenames,column_of_chromosome,column_of_position,column_of_reference_base,column_of_alterated_base] [file_1,...,file_n]] -out file\
38.     \n\n\tusage for multiple sample file: \n\t\tpython create_HeatMap.py -multiple [column_of_genenames,column_of_chromosome,column_of_position,column_of_reference_base,column_of_first_alterated_base,column_of_last_alterated_base] [file_1,...,file_n]] -out file\
39.     \n\n\tusage for both types of sample files: \n\t\tpython create_HeatMap.py [-single [column_of_genenames,column_of_chromosome,column_of_position,column_of_reference_base,column_of_alterated_base] [file_1,...,file_n]]\n [-multiple [column_of_genenames,column_of_chromosome,column_of_position,column_of_reference_base,column_of_first_alterated_base,column_of_last_alterated_base] [file_1,...,file_n]]\n [-out file]"
40.     sys.exit()
41. else:
42.     #Ueberpruefe ob die notwendigen Parameter gesetzt wurden
43.     if not ((single_parameter and output_parameter) or (multiple_parameter and output_parameter) or ( single_parameter and multiple_parameter and output_parameter)):
44.         #Wenn notwendige Parameter fehlen, dann folgende Ausgabe
45.         print "\nmissing arguments to create heatmap!\n\n\tusage for single sample file: \n\t\tpython create_HeatMap.py -single
```

```
[column_of_genenames,column_of_chromosome,column_of_position,column_of_reference_base,colu
mn_of_alterated_base] [file_1,...,file_n]] -out file\
46.     \n\n\tusage for multiple sample file: \n\t\tpython create_HeatMap.py -multiple
[column_of_genenames,column_of_chromosome,column_of_position,column_of_reference_base,colu
mn_of_first_alterated_base,column_of_last_alterated_base] [file_1,...,file_n]] -out file\
47.     \n\n\tusage for both types of sample files: \n\t\tpython create_HeatMap.py [-single
[column_of_genenames,column_of_chromosome,column_of_position,column_of_reference_base,colu
mn_of_alterated_base] [file_1,...,file_n]]\n [-multiple
[column_of_genenames,column_of_chromosome,column_of_position,column_of_reference_base,colu
mn_of_first_alterated_base,column_of_last_alterated_base] [file_1,...,file_n]]\n [-out file]"
48.     sys.exit()
49.   #Ueberpruefe ob die Parameter der notwendigen Syntax entsprechen
50.   else:
51.     for parameter in sys.argv:
52.       #Ueberpruefe Parameter -single
53.       if ((parameter == "-single") and single_parameter):
54.         #Extrahiere die gesetzten Werte zum Parameter -single
55.         single_parameter_columns=sys.argv.pop(sys.argv.index(parameter, )+1).split(',')
56.         #Ueberpruefe ob die benoetigte Anzahl an Werte vorhanden ist.
57.         if(len(single_parameter_columns)!=5):
58.           #Wenn nicht, dann folgende Ausgabe.
59.           print "\nsingle column parameters are not in following syntax:\n\n\t -single
[column_of_genenames,column_of_chromosome,column_of_position,column_of_reference_base,colu
mn_of_alterated_base] [file_1,...,file_n]]"
60.           sys.exit()
61.         #Ueberpruefe alle Dateien zum Parameter -single, ob sie existieren.
62.         single_files=sys.argv.pop(sys.argv.index(parameter, )+1).split(',')
63.         for inputfile in single_files:
64.           if os.path.isfile(inputfile) != True:
65.             #Wenn nicht, dann folgende Ausgabe.
66.             print "file \"" + str(inputfile) + "\" does not exists"
67.             sys.exit()
68.       #Ueberpruefe Parameter -multiple
69.       if ((parameter == "-multiple") and multiple_parameter):
70.         #Extrahiere die gesetzten Werte zum Parameter -multiple
71.         multiple_parameter_columns=sys.argv.pop(sys.argv.index(parameter, )+1).split(',')
72.         #Ueberpruefe ob die benoetigte Anzahl an Werte vorhanden ist.
73.         if(len(multiple_parameter_columns)!=6):
74.           #Wenn nicht, dann folgende Ausgabe.
75.           print "\multiple column parameters are not in following syntax:\n\n\t -multiple
[column_of_genenames,column_of_chromosome,column_of_position,column_of_reference_base,colu
mn_of_first_alterated_base,column_of_last_alterated_base] [file_1,...,file_n]]"
76.           sys.exit()
77.         #Ueberpruefe alle Dateien zum Parameter -multiple, ob sie existieren.
78.         multiple_files=sys.argv.pop(sys.argv.index(parameter, )+1).split(',')
79.         for inputfile in multiple_files:
80.           if os.path.isfile(inputfile) != True:
81.             #Wenn nicht, dann folgende Ausgabe.
82.             print "file \"" + str(inputfile) + "\" does not exists"
83.             sys.exit()
84.       #Ueberpruefe Parameter -out
85.       if ((parameter == "-out") and output_parameter):
86.         #Setze Pfad zur output-Datei
87.         outputfile=sys.argv.pop(sys.argv.index(parameter, )+1)
88.         try:
89.           #Versuche Datei zu erstellen
90.           output_file=open(outputfile, 'w')
```

```
91.                 output_file.close()
92.             except:
93.                 #Wenn Datei nicht erstellt werden kann dann folgende Ausgabe
94.                 print "ERROR\n\tCould not create file: \""+outputfile+"\""
95.                 sys.exit()
96.
97.     #Fuer alle Dateien des Typs -single
98.     for inputfile_path in single_files:
99.         #Parameter definieren
100.        #Datei oeffnen
101.        inputfile=open(inputfile_path, 'r')
102.        #Datei-Inhalt einlesen
103.        inputfile_content=inputfile.readlines()
104.        #Sample Namen setzen
105.        sample_name=inputfile.name
106.        #Ueberschrift fuer die Heatmap setzen
107.        heatmap_titles=heatmap_titles+sample_name+'\t'
108.        #Samples aus Datei speichern
109.        all_samples_in_all_files.append(sample_name)
110.        #Zeilenzaehler setzen
111.        count=0
112.        for line in inputfile_content:
113.            #Zaehle jede Zeile
114.            count=count+1
115.
116.    #       #wenn 1000 zeilen eingelesen dann anzahl ausgeben
117.    #       if count%100 == 0:
118.    #           print '%d genes finished' % (count)
119.
120.            #Erste Zeile igniorieren
121.            if count != 1:
122.                #Alle Strings, welche durch Tabulator getrennt sind speichern.
123.                #Zusaetzlich am Ende den Zeilenumbruch entfernen.
124.                line_elements=line.rstrip('\n').split('\t')
125.                #Gen Namen setzen
126.                gene_name=line_elements[int(single_parameter_columns[0])-1]
127.                #Chromosom setzen
128.                chromosome=line_elements[int(single_parameter_columns[1])-1]
129.                #Position setzen
130.                position=line_elements[int(single_parameter_columns[2])-1]
131.                #Reference Base setzen
132.                reference=line_elements[int(single_parameter_columns[3])-1]
133.                #Alteration setzen
134.                alteration=line_elements[int(single_parameter_columns[4])-1]
135.
136.                #Alle dictionaries und array leeren ausser heatmap
137.                chromosome_dict={}
138.                position_dict={}
139.                alteration_dict={}
140.                samples=[]
141.
142.                #Ueberpruefe ob der Gen Name schon in der Heatmap gelistet ist.
143.                if gene_name in heatmap.keys():
144.                    #Wenn Gen Name schon vorhanden, dann den Eintrag herausholen
145.                    chromosome_dict=heatmap[gene_name]
146.
147.                    #Ueberpruefe ob Chromosom schon gelistet ist.
```

```
148.            if chromosome in chromosome_dict.keys():
149.              #Wenn Chromosom schon vorhanden, dann den Eintrag herausholen
150.              position_dict=chromosome_dict[chromosome]
151.
152.              #Ueberpruefe ob Position schon gelistet ist.
153.              if position in position_dict.keys():
154.                #Wenn Position schon vorhanden, dann den Eintrag herausholen
155.                alteration_dict=position_dict[position]
156.
157.                #Ueberpruefe ob Alteration schon gelistet ist.
158.                if alteration in alteration_dict.keys():
159.                  #Wenn Alteration schon vorhanden, dann den Eintrag herausholen
160.                  samples=alteration_dict[alteration]
161.                  #Ueberpruefe ob Sample schon gelistet ist.
162.                  if str(sample_name) in samples:
163.                    #Wenn ja, dann Fehlerausgabe da dieser Eintrag schon existiert.
164.                    print "ERROR entry already exists!!!!\nIn file :\""+ str(inputfile.name) +"\"\nIn line :\""+
    line +"\"\nCombination of sample, gene, chromosome, position, alteration is doubled!\n\tPlease remove
    this line"
165.                    sys.exit()
166.                  #Wenn Sample noch nicht eingetragen ist, dann abspeichern
167.                  else:
168.                    #Speichere sample_file Namen
169.                    samples.append(sample_name)
170.                    #Speichere Alteration
171.                    alteration_dict[alteration]=samples
172.                    #Speichere Position
173.                    position_dict[position]=alteration_dict
174.                    #Speichere Chromosome
175.                    chromosome_dict[chromosome]=position_dict
176.                    #Speichere zusammengefasste Information zum Gen Namen in die Heatmap
177.                    heatmap[gene_name]=chromosome_dict
178.
179.                #Wenn Alteration nicht vorhanden ist speichere Sample und Alteration und aktualisiere
    Heatmap Eintrag
180.                else:
181.                  #Speicher sample_file Namen
182.                  samples.append(sample_name)
183.                  #Fuege Alteration hinzu und aktualisiere Heatmap
184.                  alteration_dict.update({alteration:samples})
185.                  position_dict[position]=alteration_dict
186.                  chromosome_dict[chromosome]=position_dict
187.                  heatmap[gene_name]=chromosome_dict
188.
189.              #Wenn Position nicht vorhanden ist speichere Sample, Alteration und Position und
    aktualisiere Heatmap Eintrag
190.              else:
191.                #Speichere sample_file Namen
192.                samples.append(sample_name)
193.                #Speichere Alteration
194.                alteration_dict[alteration]=samples
195.                #Fuege Position hinzu und aktualisiere Heatmap Eintrag
196.                position_dict.update({position:alteration_dict})
197.                chromosome_dict[chromosome]=position_dict
198.                heatmap[gene_name]=chromosome_dict
199.
```

```
200.            #Wenn Chromosom nicht vorhanden ist speichere Sample, Alteration, Position und Chromosom und aktualisiere Heatmap Eintrag
201.            else:
202.              #Speichere sample_file Namen
203.              samples.append(sample_name)
204.              #Speichere Alteration
205.              alteration_dict[alteration]=samples
206.              #Speichere Position
207.              position_dict[position]=alteration_dict
208.              #Fuege Chromosom hinzu und aktualisiere Heatmap Eintrag
209.              chromosome_dict.update({chromosome:position_dict})
210.              heatmap[gene_name]=chromosome_dict
211.
212.          #Wenn Gen Name nicht vorhanden ist speichere Sample, Alteration, Position, Chromosom und Gen Namen und aktualisiere Heatmap Eintrag
213.          else:
214.            #Speichere sample_file Namen
215.            samples.append(sample_name)
216.            #Speichere Alteration
217.            alteration_dict[alteration]=samples
218.            #Speichere Position
219.            position_dict[position]=alteration_dict
220.            #Speichere Chromosome
221.            chromosome_dict[chromosome]=position_dict
222.            #Speichere zusammengefasste Gen Information in der Heatmap
223.            heatmap[gene_name]=chromosome_dict
224.
225.        #Ueberpruefe die erste Zeile
226.        else:
227.          line_elements=line.split('\t')
228.          column_number_temp=[]
229.          for column_number in single_parameter_columns:
230.            #Ueberpruefe ob doppelte Eintraege in den -single parameter angegeben werden
231.            if (column_number in column_number_temp):
232.              #Wenn ja, dann folgende Ausgabe.
233.              print "\nsingle column parameters are doubled!\n\n\t This number is doubled \"" + str(column_number) + "\". Each column number could be used one time!"
234.              sys.exit()
235.            else:
236.              #Ansonsten speichere Spaltennummer
237.              column_number_temp.append(column_number)
238.            #Ueberpruefe ob die Spaltennummer in der Datei vorhanden ist
239.            if(int(column_number) > len(line_elements)):
240.              #Wenn nicht, dann folgende Ausgabe
241.              print "\nsingle column parameters are not in following file:\n\n\t \"" + str(column_number) + "\" is not available in file \"" + str(inputfile.name) + "\"\n\t each line should be in tab delimited manner!"
242.              sys.exit()
243.
244.  #Fuer alle Dateien des Typs -multiple
245.  for inputfile_path in multiple_files:
246.    #Parameter definieren
247.    #Datei oeffnen
248.    inputfile=open(inputfile_path, 'r')
249.    #Datei-Inhalt einlesen
250.    inputfile_content=inputfile.readlines()
251.    #Sample Namen setzen
252.    sample_name=inputfile.name
```

```
253.    #Ueberschrift fuer die Heatmap setzen
254.    for sample_number in range(int(multiple_parameter_columns[4]),int(multiple_parameter_columns[5])+1):
255.        corrected_sample_number=int(sample_number-int(multiple_parameter_columns[4])+1)
256.        heatmap_titles=heatmap_titles+sample_name+'_'+str(corrected_sample_number)+'\t'
257.        #Samples aus Datei speichern
258.        all_samples_in_all_files.append(sample_name+'_'+str(corrected_sample_number))
259.    #Zeilenzaehler setzen
260.    count=0
261.    for line in inputfile_content:
262.        #Zaehle jede Zeile
263.        count=count+1
264.
265.  #       #wenn 1000 zeilen eingelesen dann anzahl ausgeben
266.  #       if count%100 == 0:
267.  #           print '%d genes finished' % (count)
268.
269.        #Erste Zeile igniorieren
270.        if count != 1:
271.            #Alle Strings, welche durch Tabulator getrennt sind speichern.
272.            #Zusaetzlich am Ende den Zeilenumbruch entfernen.
273.            line_elements=line.rstrip('\n').split('\t')
274.            #Gen Namen setzen
275.            gene_name=line_elements[int(multiple_parameter_columns[0])-1]
276.            #Chromosom setzen
277.            chromosome=line_elements[int(multiple_parameter_columns[1])-1]
278.            #Position setzen
279.            position=line_elements[int(multiple_parameter_columns[2])-1]
280.            #Reference Base setzen
281.            reference=line_elements[int(multiple_parameter_columns[3])-1]
282.
283.            #Fuer -multiple Dateityp alle Samples aus den Spalten nach Alteration dursuchen
284.            for column_for_sample in range(int(multiple_parameter_columns[4])-1,int(multiple_parameter_columns[5])):
285.                #Setze Samplenummer
286.                sample_number=int(column_for_sample-int(multiple_parameter_columns[4])+2)
287.                #Extrahiere die Base des Samples
288.                alteration_per_sample=line_elements[int(column_for_sample)]
289.                #Ueberpruefe ob es eine Alteration ist
290.                if alteration_per_sample != reference and alteration_per_sample != 'N':
291.                    #Wenn ja, dann Alteration setzen
292.                    alteration=alteration_per_sample
293.
294.                    #Alle dictionaries und array leeren ausser heatmap
295.                    chromosome_dict={}
296.                    position_dict={}
297.                    alteration_dict={}
298.                    samples=[]
299.
300.                    #Ueberpruefe ob der Gen Name schon in der Heatmap gelistet ist.
301.                    if gene_name in heatmap.keys():
302.                        #Wenn Gen Name schon vorhanden, dann den Eintrag herausholen
303.                        chromosome_dict=heatmap[gene_name]
304.
305.                        #Ueberpruefe ob Chromosom schon gelistet ist.
306.                        if chromosome in chromosome_dict.keys():
307.                            #Wenn Chromosom schon vorhanden, dann den Eintrag herausholen
```

```
308.                    position_dict=chromosome_dict[chromosome]
309.
310.                    #Ueberpruefe ob Position schon gelistet ist.
311.                    if position in position_dict.keys():
312.                        #Wenn Position schon vorhanden, dann den Eintrag herausholen
313.                        alteration_dict=position_dict[position]
314.
315.                        #Ueberpruefe ob Alteration schon gelistet ist.
316.                        if alteration in alteration_dict.keys():
317.                            #Wenn Alteration schon vorhanden, dann den Eintrag herausholen
318.                            samples=alteration_dict[alteration]
319.                            #Ueberpruefe ob Sample schon gelistet ist.
320.                            for saved_sample in samples:
321.                                #Wenn ja, dann Fehlerausgabe da dieser Eintrag schon existiert.
322.                                if str(sample_name+'_'+str(sample_number)) == saved_sample:
323.                                    print "ERROR entry already exists!!!!\nIn file :\""+ str(inputfile.name) +"\"\nIn line :\""+ line +"\"\nCombination of sample, gene, chromosome, position, alteration is doubled!\n\tPlease remove this line"
324.                                    sys.exit()
325.                                    break
326.
327.                                #Wenn Sample noch nicht eingetragen ist, dann abspeichern
328.                            #Speichere sample_file Namen
329.                            samples.append(sample_name+'_'+str(sample_number))
330.                            #Speichere Alteration
331.                            alteration_dict[alteration]=samples
332.                            #Speichere Position
333.                            position_dict[position]=alteration_dict
334.                            #Speichere Chromosome
335.                            chromosome_dict[chromosome]=position_dict
336.                            #Speichere zusammengefasste Information zum Gen Namen in die Heatmap
337.                            heatmap[gene_name]=chromosome_dict
338.
339.                        #Wenn Alteration nicht vorhanden ist speichere Sample und Alteration und aktualisiere Heatmap Eintrag
340.                        else:
341.                            #Speicher sample_file Namen
342.                            samples.append(sample_name+'_'+str(sample_number))
343.                            #Fuege Alteration hinzu und aktualisiere Heatmap
344.                            alteration_dict.update({alteration:samples})
345.                            position_dict[position]=alteration_dict
346.                            chromosome_dict[chromosome]=position_dict
347.                            heatmap[gene_name]=chromosome_dict
348.
349.                    #Wenn Position nicht vorhanden ist speichere Sample, Alteration und Position und aktualisiere Heatmap Eintrag
350.                    else:
351.                        #Speichere sample_file Namen
352.                        samples.append(sample_name+'_'+str(sample_number))
353.                        #Speichere Alteration
354.                        alteration_dict[alteration]=samples
355.                        #Fuege Position hinzu und aktualisiere Heatmap Eintrag
356.                        position_dict.update({position:alteration_dict})
357.                        chromosome_dict[chromosome]=position_dict
358.                        heatmap[gene_name]=chromosome_dict
359.
```

```
360.                #Wenn Chromosom nicht vorhanden ist speichere Sample, Alteration, Position und Chromosom und aktualisiere Heatmap Eintrag
361.                else:
362.                  #Speichere sample_file Namen
363.                  samples.append(sample_name+'_'+str(sample_number))
364.                  #Speichere Alteration
365.                  alteration_dict[alteration]=samples
366.                  #Speichere Position
367.                  position_dict[position]=alteration_dict
368.                  #Fuege Chromosom hinzu und aktualisiere Heatmap Eintrag
369.                  chromosome_dict.update({chromosome:position_dict})
370.                  heatmap[gene_name]=chromosome_dict
371.
372.              else:
373.                #Wenn Gen Name nicht vorhanden ist speichere Sample, Alteration, Position, Chromosom und Gen Namen und aktualisiere Heatmap Eintrag
374.                #Speichere sample_file Namen
375.                samples.append(sample_name+'_'+str(sample_number))
376.                #Speichere Alteration
377.                alteration_dict[alteration]=samples
378.                #Speichere Position
379.                position_dict[position]=alteration_dict
380.                #Speichere Chromosome
381.                chromosome_dict[chromosome]=position_dict
382.                #Speichere zusammengefasste Gen Information in der Heatmap
383.                heatmap[gene_name]=chromosome_dict
384.            #else:
385.              #Ansonsten keine Alteration in dieser Spalte oder Sample der Datei gefunden
386.
387.      #Ueberpruefe die erste Zeile
388.      else:
389.        line_elements=line.split('\t')
390.        column_number_temp=[]
391.        for column_number in single_parameter_columns:
392.          #Ueberpruefe ob doppelte Eintraege in den -multiple parameter angegeben werden
393.          if (column_number in column_number_temp):
394.            #Wenn ja, dann folgende Ausgabe.
395.            print "\nsingle column parameters are doubled!\n\n\t This number is doubled \"" + str(column_number) + "\". Each column number could be used one time!"
396.            sys.exit()
397.          else:
398.            #Ansonsten speichere Spaltennummer
399.            column_number_temp.append(column_number)
400.          #Ueberpruefe ob die Spaltennummer in der Datei vorhanden ist
401.          if(int(column_number) > len(line_elements)):
402.            #Wenn nicht, dann folgende Ausgabe
403.            print "\nsingle column parameters are not in following file:\n\n\t \"" + str(column_number) + "\" is not available in file \"" + str(inputfile.name) + "\"\n\t each line should be in tab delimited manner!"
404.            sys.exit()
405.
406.  #Oeffne output-Datei zum Schreiben
407.  output_file=open(outputfile, 'w')
408.  #Fuege Ueberschriften hinzu
409.  output_file.write("gene\talterations per gene\t"+heatmap_titles+'alterations separated by unique chromosome_position_alteration in each sample per gene\n')
410.
411.  #Die gesamte erstellte Heatmap durchgehen
```

```
412. for gene in heatmap.keys():
413.     #Erstelle eine Zeile pro Gen
414.     heatmap_line=''
415.     #temporaere variablen um die Alterationenliste und die korrespondierenden Samples zu verarbeiten
416.     sample_consolidation={}
417.     alteration_consolidation={}
418.     consolidation_samples=[]
419.     alteration_amount=0
420.     #Alle Samples der gesamten Dateien durchgehen
421.     for name in all_samples_in_all_files:
422.         #Setze Zaehler auf null
423.         count_alteration_in_sample=0
424.         #Alle Chromosomeneintraege pro Gen durchgehen
425.         for chromosome in heatmap[gene]:
426.             #Alle Positioneneintraege pro Gen durchgehen
427.             for position in heatmap[gene][chromosome]:
428.                 #Alle Alterationeneintraege pro Gen durchgehen
429.                 for alteration in heatmap[gene][chromosome][position]:
430.                     #Alle Samples durchgehen
431.                     for sample in heatmap[gene][chromosome][position][alteration]:
432.                         #Ueberpruefe check for corresponding gene, chromosome, position and alteration at this
    sample
433.                         if name == sample:
434.                             #Finde und speichere alle Samples mit der gleichen Alteration
435.                             #Wenn gleiches Chromosom, Position und Alteration
436.                             if chromosome+"_"+position+"_"+alteration in alteration_consolidation:
437.                                 #Dann fuege Informationen der Samples zusammen
438.
    consolidation_samples=alteration_consolidation[chromosome+"_"+position+"_"+alteration]
439.                                 consolidation_samples.append(name)
440.                             else:
441.                                 #Ansonsten Informationen des Samples speichern
442.                                 consolidation_samples.append(name)
443.                             #Speichere alle Infos zum Sample
444.
    alteration_consolidation[chromosome+"_"+position+"_"+alteration]=consolidation_samples
445.
446.                             #Liste zuruecksetzen
447.                             consolidation_samples=[]
448.                             #Zaehler hochsetzen
449.                             count_alteration_in_sample=count_alteration_in_sample+1
450.                             #Zaehler hochsetzen
451.                             alteration_amount=alteration_amount+1
452.         #Informationen zur Zeile abspeichern
453.         heatmap_line=heatmap_line+str(count_alteration_in_sample)+'\t'
454.
455.     temporary_alteration_amount=''
456.     #Ausgabe der zusammengefassten Information
457.     for called_alteration in alteration_consolidation:
458.         temporary_alteration_amount=temporary_alteration_amount+called_alteration+"_samples"
459.         for sample_hitted in alteration_consolidation[called_alteration]:
460.             temporary_alteration_amount=temporary_alteration_amount+"_"+sample_hitted
461.         temporary_alteration_amount=temporary_alteration_amount+":"
462.
463.
    heatmap_line=gene+'\t'+str(alteration_amount)+'\t'+heatmap_line+str(temporary_alteration_amount)
464.
```

```
465.      #Schreibe die Zeile in die Datei und entferne das letzte Trennzeichen
466.      output_file.write(heatmap_line.rstrip(':')+"\n")
467.
468.  output_file.close()
```

9.1.4 Unbeachtete Regionen

Im folgendem der kommentierte Quellcode des Python-Skripts der unbeachteten Regionen, welcher in drei Dateien separiert vorliegt und der dazugehörigen Konfigurationsdatei.

Kofigurationsdatei mit benötigten Programmen für die Durchführung des Skriptes:

```
1.    [awk]
2.    path = /usr/bin
3.    [bedtools]
4.    path = /home/jordan/Software/BEDTools/BEDTools-Version-2.16.2/bin
5.    [gengen]
6.    path = /home/jordan/Software/Annotation/gengen
7.    [gene_matching]
8.    path = /home/jordan/workspace_neon/Merging/gene_mapping
```

Quellcode zur run-Datei:

```
1.    #!/usr/bin/python
2.    import Monitor, unconReg, argparse, os, sys, traceback, logging, tkFileDialog, tkMessageBox, Tkinter
3.
4.    #Zur Ausfuehrung benoetigte Programme
5.    required_programs=["awk","bedtools","gengen","gene_matching"]
6.    #Definition der Syntax von logging, welches fuer die Kommunikation per Kommandozeile genutzt wird
7.    logging.basicConfig(level=logging.INFO, format='%(levelname)s:%(message)s')
8.    #Initialisierung des root window fuer die Eingabe und Selektion der Dateien und Pfade
9.    root_window=Tkinter.Tk()
10.   root_window.withdraw()
11.   root_window.focus_force()
12.
13.   '''
14.   Methoden fuer die Eingabe per tkFileDialogs
15.   '''
16.   #Eingabe der Dateien der Tumor und korrespondierenden Normalproben
17.   def insert_sample_matrix(insert_bedgraph_files):
18.      #Wenn die Eingabe auf bedgraphs Dateien beschraenkt wird
19.      if insert_bedgraph_files:
20.         logging.info("Please select bedgraph files.")
21.         tumor_file="tumor_file"
22.         file_count=0
23.         sample_matrix={}
24.         #Solange wie Dateien eingegeben werden
25.         while tumor_file:
26.            file_count=file_count+1
27.            #Auswahl der Datei per Dialog
28.            tumor_file = tkFileDialog.askopenfilename(parent=root_window, title='Select a tumor *.bedgraph
      file', filetypes=[("bedgraph file", "*.bedgraph")])
29.            #Wenn keine Datei ausgewaehlt wurdedann die Schleife unterbrechen
30.            if not tumor_file:
31.               break
```

```
32.             #Wenn Datei zuvor schon ausgewaehlt wurde dann die erneute Auswahl ignorieren
33.             if tumor_file in sample_matrix:
34.                 logging.warn("File already exists and could not insert into the sample matrix")
35.                 continue
36.             #Auswahl der Datei per Dialog
37.             normal_file = tkFileDialog.askopenfilename(parent=root_window, title='Select the corresponding normal *.bedgraph file', filetypes=[("bedgraph file", "*.bedgraph")])
38.             #Wenn keine Datei ausgewaehlt wurde dann einen Platzhalter speichern
39.             if not normal_file:
40.                 normal_file="NONE"
41.             #Ausgabe welche Dateien ausgewaehlt wurden
42.             logging.info("Selected file pair:\ntumor file: "+str(tumor_file)+"\ncorresponding normal: "+str(normal_file))
43.             #Ueberpruefe ob Datei schon gespeichert wurde
44.             if tumor_file not in sample_matrix:
45.                 #Wenn nicht dann speichern
46.                 sample_matrix[tumor_file]=normal_file
47.             else:
48.                 #Ansonsten ignorieren
49.                 logging.warn("tumor file already exists in the sample matrix it could not be overwrite")
50.         if sample_matrix:
51.             #Alle Dateien aus der Eingabe zurueck geben
52.             return sample_matrix
53.         else:
54.             #Falls keine Dateien eingegeben wurden abbrechen da nichts zu tun ist
55.             logging.error("No tumor file is selected. Abort.")
56.             sys.exit(0)
57.     #Ansonsten die Standardeingabe von bam Dateien durchfuehren
58.     else:
59.         logging.info("Please select bam files.")
60.         tumor_file="tumor_file"
61.         file_count=0
62.         sample_matrix={}
63.         #Solange wie Dateien eingegeben werden
64.         while tumor_file:
65.             file_count=file_count+1
66.             #Auswahl der Datei per Dialog
67.             tumor_file = tkFileDialog.askopenfilename(parent=root_window, title='Select a tumor *.bam file', filetypes=[("bam file", "*.bam")])
68.             #Wenn keine Datei ausgewaehlt wurdedann die Schleife unterbrechen
69.             if not tumor_file:
70.                 break
71.             #Wenn Datei zuvor schon ausgewaehlt wurde dann die erneute Auswahl ignorieren
72.             if tumor_file in sample_matrix:
73.                 logging.warn("File already exists and could not insert into the sample matrix")
74.                 continue
75.             #Auswahl der Datei per Dialog
76.             normal_file = tkFileDialog.askopenfilename(parent=root_window, title='Select the corresponding normal *.bam file', filetypes=[("bam file", "*.bam")])
77.             #Wenn keine Datei ausgewaehlt wurde dann einen Platzhalter speichern
78.             if not normal_file:
79.                 normal_file="NONE"
80.             #Ausgabe welche Dateien ausgewaehlt wurden
81.             logging.info("Selected file pair:\ntumor file: "+str(tumor_file)+"\ncorresponding normal: "+str(normal_file))
82.             #Ueberpruefe ob Datei schon gespeichert wurde
83.             if tumor_file not in sample_matrix:
```

```
84.                 #Wenn nicht dann speichern
85.                 sample_matrix[tumor_file]=normal_file
86.             else:
87.                 #Ansonsten ignorieren
88.                 logging.warn("tumor file already exists in the sample matrix it could not be overwrite")
89.         if sample_matrix:
90.             #Alle Dateien aus der Eingabe zurueck geben
91.             return sample_matrix
92.         else:
93.             #Falls keine Dateien eingegeben wurden abbrechen da nichts zu tun ist
94.             logging.error("No tumor file is selected. Abort.")
95.             sys.exit(0)
96.     #Eingabe der Dateien der Exom Extraktion Kits
97.     def insert_library_list():
98.         library_file="library_file"
99.         library_list=[]
100.        #Solange wie Dateien eingegeben werden
101.        while library_file:
102.            #Auswahl der Datei per Dialog
103.            library_file = tkFileDialog.askopenfilename(parent=root_window, title='Select a library *.bed file',
    filetypes=[("bed file", "*.bed")])
104.            #Wenn keine Datei ausgewaehlt wurdedann die Schleife unterbrechen
105.            if not library_file:
106.                break
107.            #Wenn Datei zuvor schon ausgewaehlt wurde dann die erneute Auswahl ignorieren
108.            if library_file in library_list:
109.                logging.warn("File already exists and could not insert into the library list")
110.                continue
111.            #Ueberpruefe ob Datei schon gespeichert wurde
112.            if library_file not in library_list:
113.                #Wenn nicht dann speichern
114.                library_list.append(library_file)
115.            else:
116.                #Ansonsten ignorieren
117.                logging.warn("library file already exists in the library list it could not be overwrite")
118.        if library_list:
119.            #Alle Dateien aus der Eingabe zurueck geben
120.            return library_list
121.        else:
122.            #Falls keine Dateien eingegeben wurden abbrechen da nichts zu tun ist
123.            logging.error("No library file is selected. Abort.")
124.            sys.exit(0)
125.
126.    '''
127.    Ueberpruefe die Kommandozeilen Parameter mit argparse
128.    '''
129.    inputparser = argparse.ArgumentParser(description='Analyze white, gray and black regions in DNA-
    Target-NGS-datasets')
130.    #Annotationseigenschaften
131.    inputparser.add_argument('-a','--annotate_regions', action='store', choices=['black','gray','white','all'],
    dest='annotation_setting', help='Annotate identified regions. Default are black regions. Possible settings
    are black, gray, white and all', default='black')
132.    #Eingabe per bam oder bedgraph
133.    inputparser.add_argument('-b','--insert_bedgraph',
    action='store_true',dest='insert_bedgraph_files',help='Starting from *.bam or *.bedgraph files. Default it
    is set from *.bam files', default=False)
134.    #Das Limit der Abdeckung zur Differenzierung der Regionen
```

```
135. inputparser.add_argument('-c','--cutoff_coverage', action='store', type=int , choices=xrange(0,100), dest='cutoff_coverage', help='Read amount per base to define uncovered regions. Default value is 0', default=0)
136. #Definition des Ausgabepfades
137. inputparser.add_argument('-o','--output', action='store', dest='output_path',help='Output file name',required=True)
138. #Pfad zur refGene Datei fuer die Annotation heruntergeladen von UCSC hompage
139. inputparser.add_argument('-r','--refGene', action='store',type=argparse.FileType('r'), dest='refGene_file', help='refGene file downloaded from UCSC webpage', required=True)
140. #Pfad zur refLink Datei fuer die Annotation heruntergeladen von UCSC hompage
141. inputparser.add_argument('-l','--refLink', action='store',type=argparse.FileType('r'), dest='refLink_file', help='refLink file downloaded from UCSC webpage', required=True)
142. #Pfad zur COSMIC Datei fuer den Abgleich heruntergeladen von der Sanger homepage
143. inputparser.add_argument('-d','--database_COSMIC', action='store',type=argparse.FileType('r'), dest='database_COSMIC', help='COSMIC *.tsv file downloaded from COSMIC webpage', required=True)
144.
145. #Parameter einlesen
146. args = inputparser.parse_args()
147.
148. #Ueberpruefe ob Ausgabeordner existiert
149. if not os.path.isdir(args.output_path):
150.     logging.error("Directory " + args.output_path+ " not available.")
151.     sys.exit(0)
152. else:
153.     #Ueberpruefe config Datei und die benoetigten Programme
154.     configparser_file = Monitor.check_for_programs(required_programs)
155.     #Erzeuge ein Objekt fuer die Analyse
156.     unconReg_analysis = unconReg.unconReg(insert_sample_matrix(args.insert_bedgraph_files),insert_library_list(), configparser_file,str(args.output_path + "multiintersection_of_all_samples.bed"))
157.     #Frage ob die Analyse starten soll per Dialog
158.     start_analysis=tkMessageBox.askyesno("Analysis", "Do you want to start analysis?")
159.     #Das root window kann nun zerstoert werden
160.     root_window.destroy()
161.     #Die Analyse in einem try except Block durchfuehren
162.     if start_analysis:
163.         try:
164.             unconReg_analysis.run(args.output_path, args.cutoff_coverage, args.annotation_setting, args.insert_bedgraph_files, args.refGene_file, args.refLink_file, args.database_COSMIC)
165.         except Exception as e:
166.             logging.error(traceback.format_exc())
167.
168.     else:
169.         #Ansonsten Analyse abbrechen
170.         sys.exit(0)
```

Quellcode zur Monitor-Datei:

```
1. #!/usr/bin/python
2. import os, sys, ConfigParser, tkFileDialog, logging, Tkinter
3.
4. #Standard Pfad zur config Datei
5. DEFAULT_CONFIGFILE = os.path.join(sys.path[0], ".unconreg_config_file.ini")
6.
7. '''
8. Gib Pfad des Programms zurueck
9. '''
```

```
10.  def get_option_in_section(section, option):
11.      configfile_parser = ConfigParser.SafeConfigParser()
12.      configfile_parser.read(DEFAULT_CONFIGFILE)
13.      return configfile_parser.get(section, option)
14.
15.  '''
16.  Ueberpruefe die config Datei welche die benoetigten Programme beinhaltet
17.  '''
18.  def check_for_programs(required_programs):
19.
20.      root_window=Tkinter.Tk()
21.      root_window.withdraw()
22.      root_window.focus_force()
23.
24.
25.      DEFAULT_CONFIGFILE_SECTIONS = required_programs
26.      DEFAULT_CONFIGFILE_OPTIONS = ["path"]
27.
28.      logging.basicConfig(level=logging.INFO, format='%(levelname)s:%(message)s')
29.      configfile_parser = ConfigParser.SafeConfigParser()
30.
31.      '''
32.      Gib die gespeicherten Eigenschaften der aktuellen congi Datei aus
33.      '''
34.      def show_settings_in_configfile():
35.          configfile_parser.read(DEFAULT_CONFIGFILE)
36.          for section in configfile_parser.sections():
37.              tmp_options=''
38.              for option in configfile_parser.options(section):
39.                  tmp_options=tmp_options+'\n'+option+'\t'+configfile_parser.get(section, option)
40.              logging.info("section: "+section+"\nwith options and values: "+tmp_options)
41.
42.      '''
43.      Aktualisiere den Programmnamen in der config Datei
44.      '''
45.      def update_configfile_section(section):
46.          with open(DEFAULT_CONFIGFILE , "r+") as configfile:
47.              logging.info("updating config file section: "+section)
48.              configfile_parser.add_section(section)
49.              configfile_parser.write(configfile)
50.          configfile.close()
51.
52.      '''
53.      Aktualisiere und ueberpruefe den Pfad des Programms in der config Datei
54.      '''
55.      def update_configfile_option(section, option):
56.          with open(DEFAULT_CONFIGFILE , "r+") as configfile:
57.              logging.info("updating config file option: "+option+" in section: "+section)
58.              if option == "path":
59.                  new_option=tkFileDialog.askdirectory(parent=root_window, title="Please select "+option+" to
    "+section+" for the analysis.", mustexist=True)
60.                  if not(new_option =='NONE' or new_option ==''):
61.                      configfile_parser.set(section, option, new_option)
62.                  else:
63.                      logging.error("Directory couldn't be selected. Abort!")
64.                      sys.exit(0)
65.              else:
```

```
66.                 configfile_parser.set(section, option, "{0}".format(raw_input("please enter {0}: ".format(option))))
67.             configfile_parser.write(configfile)
68.         configfile.close()
69.
70.     '''
71.     Solange den Pfad aendern bis ein verfuegbarer Pfad gesetzt wurde
72.     '''
73.     def change_path(section, option, path):
74.         if not os.path.isdir(path):
75.             logging.info("Path is not available, try it again: "+option+" in section: "+section)
76.             path = "{0}".format(raw_input("please enter {0}: ".format(option)))
77.             change_path(section, option, path)
78.         else:
79.             return str(path)
80.
81.     '''
82.     Schreibe oder ueberschreibe die Eigenschaften in der config Datei
83.     '''
84.     def write_configfile(sections=DEFAULT_CONFIGFILE_SECTIONS, options=DEFAULT_CONFIGFILE_OPTIONS):
85.         with open(DEFAULT_CONFIGFILE , "w") as configfile:
86.             logging.info("writing config file ...")
87.             for section in sections:
88.                 logging.info("section: "+section+" created!")
89.                 configfile_parser.add_section(section)
90.                 for option in options:
91.                     new_option=tkFileDialog.askdirectory(parent=root_window, title="Please select "+option+" to "+section+" for the analysis.", mustexist=True)
92.                     if not(new_option =='NONE' or new_option ==''):
93.                         configfile_parser.set(section, option, new_option)
94.                     else:
95.                         logging.error("Directory couldn't be selected. Abort!")
96.                         sys.exit(0)
97.             configfile_parser.write(configfile)
98.         configfile.close()
99.
100.    '''
101.    Loesche den Programmnamen aus der config Datei
102.    '''
103.    def delete_configfile_section(section):
104.        configfile_parser.remove_section(section)
105.        with open(DEFAULT_CONFIGFILE , "w") as configfile:
106.            configfile_parser.write(configfile)
107.        configfile.close()
108.
109.    '''
110.    Loesche den Pfad in der config Datei
111.    '''
112.    def delete_configfile_option(section, option):
113.        configfile_parser.remove_option(section, option)
114.        with open(DEFAULT_CONFIGFILE , "w") as configfile:
115.            configfile_parser.write(configfile)
116.        configfile.close()
117.
118.    '''
119.    Einlesen und ueberpruefen der config Datei
```

```
120.     '''
121.     def read_and_check_configfile(required_names=required_programs, sections=DEFAULT_CONFIGFILE_SECTIONS, options=DEFAULT_CONFIGFILE_OPTIONS):
122.         logging.info("reading config file {}".format(DEFAULT_CONFIGFILE))
123.         configfile_parser.read(DEFAULT_CONFIGFILE)
124.         #Bereits gespeicherte Programmnamen durchgehen
125.         for section in configfile_parser.sections():
126.             #Wenn es kein benoetigter Programmname ist
127.             if not section in sections:
128.                 logging.info("config file contains section: "+section+" which isn't default section, deleted")
129.                 #Loeschen
130.                 delete_configfile_section(section)
131.             #Ansonsten den Pfad ueberpruefen
132.             else:
133.                 #Bereits gespeicherter Pfad durchgehen
134.                 for option in configfile_parser.options(section):
135.                     #Wenn kein Pfad angegeben
136.                     if not option in options:
137.                         logging.warning("config file section "+section+" contains option: "+option+" which isn't default option, deleted")
138.                         #Loeschen
139.                         delete_configfile_option(section, option)
140.                     #Ansonsten ueberpruefen ob Pfad existiert
141.                     else:
142.                         if option == "path" and not os.path.isdir(configfile_parser.get(section, option)):
143.                             #Wenn nicht dann aktualisieren
144.                             update_configfile_option(section, option)
145.         #Ueberpruefe ob alle benoetigten Programmnamen vorhanden sind
146.         for section in sections:
147.             #Wenn Programmnamen fehlen
148.             if not section in configfile_parser.sections():
149.                 logging.warning("config file doesn\'t contain default section: "+section)
150.                 #dann aktualisieren
151.                 update_configfile_section(section)
152.                 #Ebenso die Pfade der neu erstellten Programmnamen durchgehen
153.                 for option in options:
154.                     #und aktualisieren
155.                     update_configfile_option(section, option)
156.             #Ansonsten die Pfade durchgehen
157.             else:
158.                 for option in options:
159.                     #Wenn nicht vorhanden
160.                     if not option in configfile_parser.options(section):
161.                         #dann aktualisieren
162.                         update_configfile_option(section, option)
163.
164.     '''
165.     Ueberpruefe die config Datei ob sie existiert
166.     '''
167.     if os.path.exists(DEFAULT_CONFIGFILE):
168.         #Wenn ja dann ausgeben, uberpruefen und einlesen
169.         logging.info("following configfile exists")
170.         show_settings_in_configfile()
171.         read_and_check_configfile(required_programs)
172.     else:
173.         #Ansonsten erstellen, ueberpruefen, einlesen und ausgeben
174.         logging.info("configfile not exists")
```

```
175.          write_configfile()
176.          read_and_check_configfile()
177.          logging.info("created following configfile")
178.          show_settings_in_configfile()
179.       #Die eingelesene Datei zurueck geben
180.       return configfile_parser
```

Quellcode zur unconReg-Datei:

```
1.    #!/usr/bin/python
2.    import subprocess, os, sys, Monitor, logging, collections
3.
4.    #Definition der Syntax von logging, welches fuer die Kommunikation per Kommandozeile genutzt wird
5.    logging.basicConfig(level=logging.INFO, format='%(levelname)s:%(message)s')
6.
7.    #Klasse erstellen welche zur Durchfuehrung der Analyse genutzt wird
8.    class unconReg(object):
9.
10.      '''
11.      Konstruktor der unconReg Klasse
12.      '''
13.      def __init__(self, sample_matrix, library_list, configparser_file, MULTIINTERSECTION_FILE_PATH):
14.         self.sample_matrix = sample_matrix
15.         self.library_list = library_list
16.         self.configparser_file = configparser_file
17.         self.MULTIINTERSECTION_FILE_PATH = MULTIINTERSECTION_FILE_PATH
18.         self.annotation_regions=[]
19.      '''
20.      Zugriff auf Instanzattribut
21.      '''
22.      def get_a_r(self):
23.         return self.annotation_regions
24.      '''
25.      Veraenderung des Instanzattributs
26.      '''
27.      def set_a_r(self, a_r):
28.         self.annotation_regions = a_r
29.
30.
31.      '''
32.      Ausgabe des Inhalts der Probenmatrix
33.      '''
34.      def print_sample_matrix(self):
35.         #counter
36.         c=0
37.         for sample_1 in self.sample_matrix.keys():
38.            c=c+1
39.            print 'Round: '+str(round)+' with: '+str(sample_1)
40.            print '>pair: '+sample_1+' and '+ self.sample_matrix[sample_1]+ '<'
41.
42.      '''
43.      Analyse der der nicht abgedeckten Regionen
44.      '''
45.      def run(self, output_path, cutoff, annotate_settings, insert_bedgraph_files, refGene_file, refLink_file,
      database_COSMIC):
46.         keep_running=True
47.         sample_matrix=self.sample_matrix
```

```
48.         library_list=self.library_list
49.         #Pro Tumorprobe
50.         for tumor_sample_path in sample_matrix.keys():
51.             #Datei Pfad und Name separieren
52.             tumor_sample=os.path.basename(tumor_sample_path)
53.             tumor_sample_prefix=tumor_sample.split(".")[0]
54.             #Die bedgraph Datei erstellen
55.             if not insert_bedgraph_files:
56.                 if not create_bedgraph(tumor_sample_path, tumor_sample_prefix, output_path):
57.                     logging.warn("Couldn't execute \"create bedgraph\" at :"+ tumor_sample)
58.                     keep_running=False
59.             #Die bedgraph Datei sortieren
60.             if not sort_bedgraph(tumor_sample_prefix, output_path):
61.                 logging.warn("Couldn't execute \"sort bedgraph\" at :"+ tumor_sample)
62.                 keep_running=False
63.             #Die Basen in Abhaengigkeit des Limits aus der Datei entfernen
64.             if not remove_uncovered_regions(tumor_sample_prefix, cutoff, output_path):
65.                 logging.warn("Couldn't execute \"remove uncovered regions\" at :"+ tumor_sample)
66.                 keep_running=False
67.             #Wenn eine korrespondierende Normalprobe existiert ebenso die Schritte durchfuehren
68.             if not sample_matrix[tumor_sample_path] == "NONE":
69.                 #Datei Pfad und Name separieren
70.                 normal_sample=os.path.basename(sample_matrix[tumor_sample_path])
71.                 normal_sample_prefix=normal_sample.split(".")[0]
72.                 #Die bedgraph Datei erstellen
73.                 if not insert_bedgraph_files:
74.                     if not create_bedgraph(sample_matrix[tumor_sample_path], normal_sample_prefix,
    output_path):
75.                         logging.warn("Couldn't execute \"create bedgraph\" at :"+ normal_sample)
76.                         keep_running=False
77.                 #Die bedgraph Datei sortieren
78.                 if not sort_bedgraph(normal_sample_prefix, output_path):
79.                     logging.warn("Couldn't execute \"sort bedgraph\" at :"+ normal_sample)
80.                     keep_running=False
81.                 #Die Basen in Abhaengigkeit des Limits aus der Datei entfernen
82.                 if not remove_uncovered_regions(normal_sample_prefix, cutoff, output_path):
83.                     logging.warn("Couldn't execute \"remove uncovered regions\" at :"+ normal_sample)
84.                     keep_running=False
85.         #Die Analyse fortfuehren solange keine Fehler waehrend der bisherigen Analyseschritte gemeldet
    wurden
86.         if keep_running:
87.             input_sample_list=""
88.             sample_order={}
89.             sorted_sample_order={}
90.             #Die 6te Spalte in der multiintersection Ausgabedatei beinhaltet die erste Probe
91.             column_counter=6
92.             #Pro Tumorprobe
93.             for tumor_sample_path in sample_matrix.keys():
94.                 #Datei Name und Typ separieren
95.                 tumor_sample=os.path.basename(tumor_sample_path)
96.                 tumor_sample_prefix=tumor_sample.split(".")[0]
97.                 input_sample_list=input_sample_list + " " + os.path.join(output_path, "removed_" +
    tumor_sample_prefix) + ".bedgraph"
98.                 #Typ und Spaltennummer speichern
99.                 sample_order[column_counter]="tumor_"+tumor_sample_prefix
100.                column_counter=column_counter+1
101.                #Wenn korrespondierende Normalprobe vorhanden auch zaehlen
```

```
102.            if not sample_matrix[tumor_sample_path] == "NONE":
103.              #Datei Name und Typ separieren
104.              normal_sample=os.path.basename(sample_matrix[tumor_sample_path])
105.              normal_sample_prefix=normal_sample.split(".")[0]
106.              input_sample_list=input_sample_list + " " + os.path.join(output_path,"removed_" + normal_sample_prefix) + ".bedgraph"
107.              #Typ und Spaltennummer speichern
108.              sample_order[column_counter]="normal_"+normal_sample_prefix
109.              column_counter=column_counter+1
110.          #Pro Extraktion Kit
111.          for library_sample_path in library_list:
112.            #Datei Name und Typ separieren
113.            library_sample=os.path.basename(library_sample_path)
114.            library_sample_prefix=library_sample.split(".")[0]
115.            input_sample_list=input_sample_list + " " + library_sample_path
116.            #Typ und Spaltennummer speichern
117.            sample_order[column_counter]="library_"+library_sample_prefix
118.            column_counter=column_counter+1
119.          #Die multiintersection Matrix erzeugen
120.          if create_intersection_matrix(sample_matrix, input_sample_list, output_path, self.MULTIINTERSECTION_FILE_PATH):
121.            #Dictionary sortieren
122.            sorted_sample_order = collections.OrderedDict(sorted(sample_order.items(), key=lambda t: t[0]))
123.            #Die jeweiligen Regionen berechnen
124.            if not calculate_black_gray_white_regions(self, annotate_settings, sorted_sample_order, output_path):
125.              logging.warn("Could not execute \"calculate_black_gray_white_regions\" ")
126.              keep_running=False
127.          #Ansonsten Abbruch
128.          else:
129.            logging.error('Could not create multiintersection of bedgraph files. Abort.')
130.            sys.exit(0)
131.
132.      #Die Analyse fortfuehren solange keine Fehler waehrend der bisherigen Analyseschritte gemeldet wurden
133.      if keep_running:
134.          #Dateien der Regionen fuer gengen Eingabesyntax konvertieren
135.          if not convert_to_gengen_input(self, output_path):
136.            logging.error('Could not convert to gengen input file. Abort.')
137.            sys.exit(0)
138.          #Annotation per gengen
139.          if not annotate_gengen_input(self, output_path, refGene_file, refLink_file):
140.            logging.error('Could not annotate gengen input files. Abort.')
141.            sys.exit(0)
142.          #Abgleich mit COSMIC
143.          if not compare_to_COSMIC(self, output_path, database_COSMIC):
144.            logging.error('Could not compare to COSMIC database. Abort.')
145.            sys.exit(0)
146.      #Ansonsten Abbruch
147.      else:
148.          logging.error('Could not annotate regions and compare to COSMIC. Abort.')
149.          sys.exit(0)
150.
151.  '''
152.  Analyseschritt bedgraph Datei erzeugen
153.  '''
```

```
154.  def create_bedgraph(path, file_prefix, output_path):
155.      bamgraph_command=str(os.path.join(Monitor.get_option_in_section("bedtools", "path"),
      "bedtools") + " genomecov -bga -ibam " + path + " > " + output_path + file_prefix + ".bedgraph")
156.      if execute_analysis_step(bamgraph_command) == 0:
157.          return True
158.      else:
159.          return False
160.
161.  '''
162.  Analyseschritt bedgraph Datei sortieren
163.  '''
164.  def sort_bedgraph(file_prefix, output_path):
165.      sort_command=str(os.path.join(Monitor.get_option_in_section("bedtools", "path"), "bedtools") + "
      sort -i " + output_path + file_prefix + ".bedgraph > " +output_path + "sort_" + file_prefix + ".bedgraph")
166.      if execute_analysis_step(sort_command) == 0:
167.          return True
168.      else:
169.          return False
170.
171.  '''
172.  Basen in Abhaengigkeit des Limits aus Datei entfernen
173.  '''
174.  def remove_uncovered_regions(file_prefix, cutoff, output_path):
175.      remove_command=str(os.path.join(Monitor.get_option_in_section("awk", "path"), "awk") + "
      \'{if($4>" + str(cutoff) + ")print $0}\' " + output_path + "sort_" + file_prefix + ".bedgraph > " + output_path
      + "removed_" + file_prefix + ".bedgraph")
176.      if execute_analysis_step(remove_command) == 0:
177.          return True
178.      else:
179.          return False
180.
181.  '''
182.  Analyseschritt multiintersection Datei erzeugen
183.  '''
184.  def create_intersection_matrix(sample_matrix, input_sample_list, output_path,
      MULTIINTERSECTION_FILE_PATH):
185.      intersection_command=str(os.path.join(Monitor.get_option_in_section("bedtools", "path"),
      "bedtools") + " multiinter -header -i")+ input_sample_list + " > " + MULTIINTERSECTION_FILE_PATH
186.      if execute_analysis_step(intersection_command) == 0:
187.          return True
188.      else:
189.          return False
190.
191.  '''
192.  Analyseschritt Berechnung der Regionen
193.  '''
194.  def calculate_black_gray_white_regions(self, annotation_settings, sorted_sample_order, output_path):
195.      a_r=[]
196.      tumor=False
197.      per_library=[]
198.      #Bearbeitungskommandos der jeweiligen Regionen erstellen
199.      black_command=os.path.join(Monitor.get_option_in_section("awk", "path"), "awk") + " \'{if( $"
200.      gray_command=os.path.join(Monitor.get_option_in_section("awk", "path"), "awk") + " \'{if(( $"
201.      white_command=os.path.join(Monitor.get_option_in_section("awk", "path"), "awk") + " \'{if( $"
202.      sample_prop=[]
203.      sample_type=''
204.      sample_name=''
```

```
205.      #Alle Spalten der Proben durchgehen
206.      for column_number in sorted_sample_order:
207.        sample_prop=sorted_sample_order[column_number].split('_')
208.        sample_type=sample_prop.pop(0)
209.        sample_name='_'.join(sample_prop)
210.        #Wenn Tumorprobe
211.        if sample_type == 'tumor':
212.          #Die jeweiligen Kommandos veraendern
213.          if  black_command==str(os.path.join(Monitor.get_option_in_section("awk",  "path"),  "awk") + "
      \'{if( $"):
214.            black_command=black_command+str(column_number)+' == 0'
215.          else:
216.            black_command=black_command+' && $'+str(column_number)+' == 0'
217.          if  gray_command==str(os.path.join(Monitor.get_option_in_section("awk",  "path"),  "awk") + "
      \'{if(( $"):
218.            gray_command=gray_command+str(column_number)+' == 0'
219.          else:
220.            gray_command=gray_command+' || $'+str(column_number)+' == 0'
221.          if  white_command==str(os.path.join(Monitor.get_option_in_section("awk",  "path"),  "awk") + "
      \'{if( $"):
222.            white_command=white_command+str(column_number)+' == 1'
223.          else:
224.            white_command=white_command+' && $'+str(column_number)+' == 1'
225.          tumor=True
226.        #Wenn Normalprobe
227.        if sample_type == 'normal':
228.          #Die jeweiligen Kommandos veraendern
229.          if  black_command==str(os.path.join(Monitor.get_option_in_section("awk",  "path"),  "awk") + "
      \'{if( $"):
230.            black_command=black_command+str(column_number)+' == 0'
231.          else:
232.            black_command=black_command+' && $'+str(column_number)+' == 0'
233.          if  gray_command==str(os.path.join(Monitor.get_option_in_section("awk",  "path"),  "awk") + "
      \'{if(( $"):
234.            gray_command=gray_command+str(column_number)+' == 0'
235.          else:
236.            gray_command=gray_command+' || $'+str(column_number)+' == 0'
237.          if  white_command==str(os.path.join(Monitor.get_option_in_section("awk",  "path"),  "awk") + "
      \'{if( $"):
238.            white_command=white_command+str(column_number)+' == 1'
239.          else:
240.            white_command=white_command+' && $'+str(column_number)+' == 1'
241.          tumor=False
242.        #Wenn Extraktion Kit
243.        if sample_type == 'library':
244.          #Die jeweiligen Kommandos zusammenfassen
245.          black_command_tmp=black_command+' && $'+str(column_number)+' == 1) print $0 > "' +
      os.path.join(output_path,  'black_regions_all_samples_at_'  +  sample_name)  +  '"}\'  '  +
      self.MULTIINTERSECTION_FILE_PATH
246.          gray_command_tmp=gray_command+' ) && $'+str(column_number)+' == 1) print $0 > "' +
      os.path.join(output_path,  'gray_regions_all_samples_at_'  +  sample_name)  +  '"}\'  '  +
      self.MULTIINTERSECTION_FILE_PATH
247.          white_command_tmp=white_command+' && $'+str(column_number)+' == 1) print $0 > "' +
      os.path.join(output_path,  'white_regions_all_samples_at_'  +  sample_name)  +  '"}\'  '  +
      self.MULTIINTERSECTION_FILE_PATH
248.          #Zu dem jeweiligen Kit
249.          per_library.append(black_command_tmp)
```

```
250.            per_library.append(gray_command_tmp)
251.            per_library.append(white_command_tmp)
252.            #In Abhaengigkeit des Parameters die gewuenschten Regionen und Dateien verbinden
253.            if annotation_settings == 'black':
254.               a_r.append('black_regions_all_samples_at_' + sample_name)
255.               #self.annotation_regions.append('black_regions_all_samples_at_' + sample_name)
256.            if annotation_settings == 'gray':
257.               a_r.append('gray_regions_all_samples_at_' + sample_name)
258.            if annotation_settings == 'white':
259.               a_r.append('white_regions_all_samples_at_' + sample_name)
260.            if annotation_settings == 'all':
261.               a_r.append('black_regions_all_samples_at_' + sample_name)
262.               a_r.append('gray_regions_all_samples_at_' + sample_name)
263.               a_r.append('white_regions_all_samples_at_' + sample_name)
264.            self.set_a_r(a_r)
265.      #Boolean setzen um  einen Rueckgabeparameter nach der erfolgreichen Anlyse zurueck zu geben
266.      analysis_state=True
267.      #Zum jeweiligen Kit
268.      for library_regions in per_library:
269.         #Die Annotation durchfuehre
270.         if not execute_analysis_step(library_regions) == 0:
271.            #Bei Fehlern die Analyse abbrechen
272.            analysis_state=False
273.            break
274.      return analysis_state
275.
276.   '''
277.   Analyseschritt Konvertierung zu Eingabesyntax von gengen
278.   '''
279.   def convert_to_gengen_input(self, output_path):
280.      a_r=self.get_a_r()
281.      for region in a_r:
282.         convert_command=os.path.join(Monitor.get_option_in_section("awk", "path"), "awk") + ' \'{print $1":"$2"-"$3}\' ' + ''.join((output_path, region)) + ' > ' + ''.join((output_path, 'gengen_' + region))
283.         if execute_analysis_step(convert_command) == 0:
284.            continue
285.         else:
286.            return False
287.      return True
288.
289.   '''
290.   Analyseschritt Annotation der Regionen mit gengen
291.   '''
292.   def annotate_gengen_input(self, output_path, refGene_file, refLink_file):
293.      a_r=self.get_a_r()
294.      for region in a_r:
295.         annotated_command=os.path.join(Monitor.get_option_in_section("gengen", "path"), "scan_region.pl ") + ''.join((output_path, 'gengen_' + region)) + ' -refgene ' + str(refGene_file.name) + ' -reflink ' + str(refLink_file.name) + ' > ' + ''.join((output_path, 'annotated_gengen_' + region))
296.         if execute_analysis_step(annotated_command) == 0:
297.            continue
298.         else:
299.            return False
300.      return True
301.
302.   '''
303.   Analyseschritt Abgleich der Regionen mit COSMIC
```

```
304.  '''
305.  def compare_to_COSMIC(self, output_path, database_COSMIC):
306.     a_r=self.get_a_r()
307.     #Die Gennamen extrahieren
308.     for region in a_r:
309.        get_genes_command=os.path.join(Monitor.get_option_in_section("awk", "path"), "awk") + ' \'{print $2}\' ' + ''.join((output_path, 'annotated_gengen_' + region)) + ' > ' + ''.join((output_path, 'genes_' + region))
310.        if execute_analysis_step(get_genes_command) == 0:
311.           continue
312.        else:
313.           return False
314.     #Die extrahierten Gennamen mit COSMIC abgleichen
315.     for region in self.get_a_r():
316.        compare_command=os.path.join(Monitor.get_option_in_section("gene_matching", "path"), "genes_from_gengen_annotation_matching_to_COSMIC.py -g " + ''.join((output_path, 'genes_' + region)) + " -c " + database_COSMIC.name + " -o " + ''.join((output_path, 'COSMIC_match_' + region)))
317.        if execute_analysis_step(compare_command) == 0:
318.           continue
319.        else:
320.           return False
321.     return True
322.
323.  '''
324.  Den jeweiligen Analyseschritt durchfuehren
325.  '''
326.  def execute_analysis_step(command):
327.     logging.info("Running: \n" + command + " ...")
328.     myprocess = subprocess.Popen(command, shell=True ,stdout=subprocess.PIPE)
329.     (sout, serr) = myprocess.communicate()
330.
331.     for line in sout.split('\n'):
332.        if line.strip().startswith('Bytes received'):
333.           print ("This workstation received %s bytes." % line.strip().split(' ')[-1])
334.
335.     myprocess.wait()
336.     #print str(myprocess.returncode)
337.     return myprocess.returncode
```

9.2 Abbildungsverzeichnis

9.3 Tabellenverzeichnis

9.4 Formelverzeichnis

9.5 Abkürzungsverzeichnis

AID ..... *Activation Induced Cytidine Deaminase*
ASCII ..... *American Standard Code for Information Interchange*
BAM ..... *binary Alignment/Map*
BL ..... *Burkitt Lymphom*
bp ..... Basenpaare
BWA ..... *Burrows-Wheeler Alignment*
BWT ..... *Burrows-Wheeler-Transformation*
BZR ..... B-Zell-Rezeptor
CCDS ..... *Consensus CDS*
cDNA ..... komplementäre DNA
CDS ..... *coding sequence*
ChIP-Seq ..... *chromatin immunprecipitation sequencing*
cnLOH ..... *copy neutral Loss Of Heterozygosity*
CNV ..... *copy number variation*
CRAB ..... *hypercalcemia, renal insufficiency, anemia, bone lesions*
CSR ..... *Class Switch Recombination*
DNA ..... Desoxyribonukleinsäure
dNTP ..... Deoxyribonukleotidtriphosphat
EBV ..... Epstein-Barr-Virus
Eland ..... *Efficient Large-Scale Alignment of Nucleotide Database*
emPCR ..... *emulsion PCR*
ExAC ..... *Exome Aggregation Consortium*
FDA ..... *Food and Drug Administration*
FDZ ..... follikuläre dendritische Zelle
FL ..... *Follikuläres Lymphom*
FM ..... *Ferragina-Manzini*
GA Ⅱx ..... *Genome Analyzer* Ⅱx
GATK ..... *Genome Analysis Toolkit*
GB ..... *Gigabyte*
GRCh ..... *Genome Reference Consortium Human Build*
HGNC ..... HUGO *gene nomenclature committee*
HIV ..... Human-Immundefizienz-Virus
HTS ..... *High-Throughput-Sequencing*
HUGO ..... *human genome organisation*
ICGC ..... *International Cancer Genome Consortium*
IDE ..... integrierten Entwicklungsumgebung
IgM ..... *Immunglobulin M*
IGV ..... *Integrated Genomics Viewer*
Indel ..... Insertionen und Deletionen
Kbp ..... Kilo-Basenpaare
KFO 216 ..... klinische Forschergruppe 216
Mbp ..... Mega-Basenpaare
MIM ..... *Mendelian Inheritance in Man*
MM ..... *Multiples Myelom*
NCBI ..... *National Center of Biotechnologie Information*
NGS ..... *Next-Generation-Sequencing*

OLB ..... *Offline-Basecaller*
OMIM ..... *Online Mendelian Inheritance in Man*
PCR ..... *Polymerase Chain Reaction*
PZL ..... Plasmazell Leukämie
RefSeq ..... *Reference Sequence*
RNA ..... Ribonukleinsäure
RTA ..... *Real-Time-Analysis*
RTK ..... Rezeptor-Tyrosin-Kinasen
SAM ..... *Sequence Alignment/Map*
SHM ..... somatische Hypermutation
SMRT ..... *Single Molecule Real Time*
SNP ..... *single nucleotide polymorphism*
SNV ..... *single nucleotide variant*
SOLiD ..... *Sequencing by Oligo Ligation Detection*
SV ..... Strukturelle Variationen
SVM ..... *Support Vector Machine*
TB ..... *Terabyte*
TCGA ..... *The Cancer Genome Atlas*
UTR ..... *Untranslated-Regions*
VCF ..... *Variant-Calling-File*
WES ..... *whole exom sequencing*
WGA ..... *whole genome amplification*
WGS ..... *whole genome sequencing*
WHO ..... *World Health Organisation*

9.6 Publikationen und Poster

9.6.1 Im Rahmen dieser Arbeit veröffentlichte Publikationen

Keppler S, Weißbach S, Langer C, Knop S, **Pischimarov J**, Kull M, Stühmer T, Steinbrunn T, Bargou R, Einsele H, Rosenwald A, Leich E. Rare SNPs in receptor tyrosine kinases are negative outcome predictors in multiple myeloma. Oncotarget. 2016 Jun 21;7(25):38762-38774. doi: 10.18632/oncotarget.9607. PubMed PMID: 27246973; PubMed Central PMCID: PMC5122427.

Momose S, Weißbach S, **Pischimarov J**, Nedeva T, Bach E, Rudelius M, Geissinger E, Staiger AM, Ott G, Rosenwald A. The diagnostic gray zone between Burkitt lymphoma and diffuse large B-cell lymphoma is also a gray zone of the mutational spectrum. Leukemia. 2015 Aug;29(8):1789-91. doi: 10.1038/leu.2015.34. PubMed PMID: 25673238.

Leich E, Weißbach S, Klein HU, Grieb T, **Pischimarov J**, Stühmer T, Chatterjee M, Steinbrunn T, Langer C, Eilers M, Knop S, Einsele H, Bargou R, Rosenwald A. Multiple myeloma is affected by multiple and heterogeneous somatic mutations in adhesion- and receptor tyrosine kinase signaling molecules. Blood Cancer J. 2013 Feb 8;3:e102. doi: 10.1038/bcj.2012.47. PubMed PMID: 23396385; PubMed Central PMCID: PMC3584721.

Richter J, Schlesner M, Hoffmann S, Kreuz M, Leich E, Burkhardt B, Rosolowski M, Ammerpohl O, Wagener R, Bernhart SH, Lenze D, Szczepanowski M, Paulsen M, Lipinski S, Russell RB, Adam-Klages S, Apic G, Claviez A, Hasenclever D, Hovestadt V, Hornig N, Korbel JO, Kube D, Langenberger D, Lawerenz C, Lisfeld J, Meyer K, Picelli S, **Pischimarov J**, Radlwimmer B, Rausch T, Rohde M, Schilhabel M, Scholtysik R, Spang R, Trautmann H, Zenz T, Borkhardt A, Drexler HG, Möller P, MacLeod RA, Pott C, Schreiber S, Trümper L, Loeffler M, Stadler PF, Lichter P, Eils R, Küppers R, Hummel M, Klapper W, Rosenstiel P, Rosenwald A, Brors B, Siebert R; ICGC MMML-Seq Project. Recurrent mutation of the ID3 gene in Burkitt

lymphoma identified by integrated genome, exome and transcriptome sequencing. Nat Genet. 2012 Dec;44(12):1316-20. doi: 10.1038/ng.2469. PubMed PMID: 23143595.

9.6.2 Bereits veröffentlichte Publikationen

Pischimarov J, Kuenne C, Billion A, Hemberger J, Cemič F, Chakraborty T, Hain T. sRNAdb: a small non-coding RNA database for gram-positive bacteria. BMC Genomics. 2012 Aug 10;13:384. doi: 10.1186/1471-2164-13-384. PubMed PMID: 22883983; PubMed Central PMCID: PMC3439263.

Mraheil MA, Billion A, Mohamed W, Mukherjee K, Kuenne C, **Pischimarov J**, Krawitz C, Retey J, Hartsch T, Chakraborty T, Hain T. The intracellular sRNA transcriptome of Listeria monocytogenes during growth in macrophages. Nucleic Acids Res. 2011 May;39(10):4235-48. doi: 10.1093/nar/gkr033. PubMed PMID: 21278422; PubMed Central PMCID: PMC3105390.

Mraheil MA, Billion A, Kuenne C, **Pischimarov J**, Kreikemeyer B, Engelmann S, Hartke A, Giard JC, Rupnik M, Vorwerk S, Beier M, Retey J, Hartsch T, Jacob A, Cemič F, Hemberger J, Chakraborty T, Hain T. Comparative genome-wide analysis of small RNAs of major Gram-positive pathogens: from identification to application. Microb Biotechnol. 2010 Nov;3(6):658-76. doi: 10.1111/j.1751-7915.2010.00171.x. Review. PubMed PMID: 21255362; PubMed Central PMCID: PMC3815340.

Kuenne C, Voget S, **Pischimarov J**, Oehm S, Goesmann A, Daniel R, Hain T, Chakraborty T. Comparative analysis of plasmids in the genus Listeria. PLoS One. 2010 Sep 2;5(9). pii: e12511. doi: 10.1371/journal.pone.0012511. PubMed PMID: 20824078; PubMed Central PMCID: PMC2932693.

9.6.3 Im Rahmen dieser Arbeit veröffentlichte Poster

Jordan Pischimarov, Franz Cemič, Andreas Rosenwald, Ellen Leich.

Identification and evaluation of unconsidered regions in whole exome sequencing datasets

EMBL Heidelberg. Cancer Genomics 2013 2. – 3. November.