Next-Generation Sequencing Algorithms: From Read Mapping to Variant Detection

Dissertation zur Erlangung des Grades

Doktor der Naturwissenschaften (Dr. rer. nat.)

am Fachbereich

Mathematik und Informatik

der

Freien Universität Berlin

vorgelegt von

Anne-Katrin Emde

Berlin, 15. November, 2012

Betreuer:

Prof. Dr. Knut Reinert, Freie Universität Berlin

Prof. Dr. Martin Vingron, Max-Planck-Institut für Molekulare Genetik, Berlin

Gutachter:

Prof. Dr. Knut Reinert, Freie Universität Berlin

Prof. Dr. Martin Vingron, Max-Planck-Institut für Molekulare Genetik, Berlin

Prof. Dr. Peter Nürnberg, Universität Köln

Datum der Disputation: 17. April, 2013

Abstract

Next-Generation-Sequencing (NGS) has brought on a revolution in sequence analysis with its broad spectrum of applications ranging from genome resequencing to transcriptomics or metagenomics, and from fundamental research to diagnostics. The tremendous amounts of data necessitate highly efficient computational analysis tools for the wide variety of NGS applications.

This thesis addresses a broad range of key computational aspects of resequencing applications, where a reference genome sequence is known and heavily used for interpretation of the newly sequenced sample. It presents tools for read mapping and benchmarking, for partial read mapping of small RNA reads and for structural variant/indel detection, and finally tools for detecting and genotyping SNVs and short indels. Our tools efficiently scale to large NGS data sets and are well-suited for advances in sequencing technology, since their generic algorithm design allows handling of arbitrary read lengths and variable error rates. Furthermore, they are implemented within the robust C++ library SeqAn, making them open-source, easily available, and potentially adaptable for the bioinformatics community. Among other applications, our tools have been integrated into a large-scale analysis pipeline and have been applied to large datasets, leading to interesting discoveries of human retrocopy variants and insights into the genetic causes of X-linked intellectual disabilities.

Zusammenfassung

Neuste DNA-Sequenzieungstechnologien (kurz genannt NGS Technologien) ermöglichen revolutionäre neue Anwendungen, die sowohl von Genomresequenzierung über Transkriptomsequenzierung zu Metagenomik als auch von Grundlagenforschung zu Diagnostik reichen. Problematisch ist dabei die Flut an Daten, die eine grosse Herausforderung für die Bionformatik darstellt. Hocheffiziente Analysesoftware ist von enormer Wichtigkeit für das breite Spektrum von NGS Anwendungen.

Diese Arbeit adressiert mehrere Schlüsselaspekte der Analyse von Resequenzierungsdaten, bei der ein bereits sequenziertes Referenzgenom als Grundlage für die Interpretation eines neu sequenzierten Datensatzes dient. Es werden Algorithmen und Programme präsentiert für das sogenannte Read Mapping Problem und für die Auswertung der Güte seiner Lösung, für partielles Read Mapping, welches in miRNA Studien und bei der Suche nach strukturellen Variationen Anwendung findet, sowie letztlich zum Auffinden und Genotypisieren von Basenmutationen und kurzen Insertionen/Deletionen im Genom. Die vorgestellten Algorithmen sind effizient und so gestaltet, dass sie auch bei Fortschritten in Sequenzierungstechnologien weiterhin anwendbar und skalierbar bleiben. Zudem sind sie in der robusten C++ Bibliothek SeqAn implementiert, was sie leicht zugänglich und adaptierbar macht. Unter anderem wurden unsere Tools in eine Hochdurchsatz-Analysepipeline integriert und auf grosse Datensaetze angewendet, wodurch interessante biologische Erkenntnisse (vorallem im Zusammenhang X-Chromosom gebundener geistiger Behinderung) gewonnen werden konnten.

Acknowledgements

First and foremost, I want to thank my supervisors Knut Reinert and Martin Vingron who gave me the opportunity to work on this thesis and therefore the many diverse aspects of next-generation sequencing. I very much enjoyed working in the two groups and gaining experience from a theoretical as well as a more applied perspective. I am particularly grateful to Knut who has been a true mentor for many years and whose ideas, advice and encouragement I greatly appreciate. Furthermore, I want to thank Stefan Haas from whom I learned a lot and whose supervision made my usually systematic and theoretical approach more real-world applicable. I would also like to thank Peter Nürnberg for acting as external thesis examiner.

I am very grateful to have been part of many fruitful and enjoyable collaborations. With David Weese I enjoyed working on RazerS. A lot of the work I did is based on his work and ideas. I thank Manuel Holtgrewe for the collaboration on Rabema, Marcel Grunert for the joint work on MicroRazerS and Marcel Schulz for working with me on SplazerS. Further thanks for pleasant collaborations goes to Magdalena Feldhahn at the University in Tübingen, and Peter Krawitz and Na Zhu at the Humboldt University. I furthermore enjoyed working with Hugues Richard, Ruping Sun, Tomasz Zemojtel, and Vera Kalscheuer at the MPI-MG.

Thank you to Birte Kehr, Stefan Haas, and René Rahn for proof-reading and to Patrick Schultz for helping me with figures. Special thanks to Birte for extensive proof-reading, moral support and many good conversations, both scientific and non-scientific. Also thanks to my other office mates during the years: Tobias Rausch, Sandro Andreotti, Sean O'Keefe, Kathrin Trappe. Very important: the many good coffee breaks, both at the Max-Planck-Institute as well as the university. I am grateful to all members of the two groups for creating a great working (and coffee-drinking) atmosphere. I am indebted to the International Max Planck Research School and particularly our great coordinators Kirsten Kelleher and Hannes Luz, who is missed dearly.

A huge thank you to my friends, family, roommates, and all the special people in my life.

Contents

Chapter 1

Introduction

When the structure of DNA, the carrier of genetic information, was discovered by Watson and Crick (1953), and Franklin and Gosling (1953), it was surely hard to imagine that only about 50 years later in 2001 the whole DNA sequence of a human genome would have been deciphered. Another ten years later, with revolutionary advancements in DNA sequencing technology (Mardis, 2008b), we today know the sequence of hundreds of human individuals and have sequenced more than thousands of species (NCBI, 2012). Nowadays, DNA sequencing is routinely used to answer fundamental research as well as medical/diagnostic questions. For example, the 1000 Genomes Project (Durbin *et al.*, 2010), a large-scale sequencing endeavor launched in 2008 by a consortium of researchers from more than 75 universities and companies around the world, has compiled the to date most complete catalogue of human variation. This catalogue of normal differences occurring between individual human genomes serves as a solid basis for investigations into disease. Gradually paving the way towards personalized medicine, DNA sequencing has already been successfully used to identify the genetic causes of many diseases (Johnston *et al.*, 2010; Ng *et al.*, 2010a,b); predominantly advancing diagnostics of Mendelian disorders where pathogenic variations affecting a single gene provoke a strong phenotype.

Developed by Sanger *et al.* (1977), the first method for DNA sequencing through chain termination was quite laborious and slow, but was soon automated to achieve higher throughput (Smith *et al.*, 1986). In fact, automated Sanger sequencing was the technology used to produce the first human genome draft sequences in 2001 (Venter *et al.*, 2001; Lander *et al.*, 2001) – still at comparably low throughput and high cost. Altogether it took more than ten years to finish this draft sequence, at an estimated total cost of $3 billion (Collins *et al.*, 2003). In the following years, enormous efforts were put into the development of superior sequencing technologies. By the middle of the decade, three new and revolutionary sequencing technologies were in sight: Roche (454 Life Sciences) pyrosequencing, ABI SOLiD (Life Technologies) sequencing, and Illumina (Solexa) sequencing, collectively called *next-generation sequencing* (NGS) methods (Mardis, 2008b). Compared to Sanger sequencing, these NGS technologies offered extraordinary throughput and thereby became affordable for exciting new applications. Not only has sequencing since been used for inspecting genomes of different species and individuals, but also it has found application,

for example, in sequencing and characterizing transcriptomes (Wang *et al.*, 2009b) (the entirety of DNA transcribed into RNA in a cell). Other prominent NGS applications include sequencing of DNA bound by certain proteins (Park, 2009), e.g. modified histones. Altogether, these applications have already lead to fundamental insights in gene regulation, e.g. that histone modification levels are quantitatively predictive for gene expression (Karlić *et al.*, 2010), and ultimately our understanding of molecular cell biology is profiting immensely.

Over the course of the last years, sequencing throughput has increased continuously, with Illumina (formerly Solexa) being the mostly widely adopted sequencing instrument provider. In one Illumina HiSeq instrument run, a total of 600 Gb is generated within ten days, theoretically enough to reconstruct an entire human genome (Illumina Inc., 2012). For comparison, the most efficient Sanger sequencing machines (Applied Biosystems, 2012) produce four orders of magnitude less, requiring more than 500 years for the equivalent amount of data.

With this immense data flood, bottleneck has shifted from laboratory work to computational analysis (Pop and Salzberg, 2008). Challenges are not only posed by the sheer amounts of data – with sequencing power rising more quickly than storage capacity, even data storage has become a challenge (Shumway *et al.*, 2010) – but also the nature and characteristics of the data complicate analysis. Compared to Sanger sequencing, the length of DNA fragment that can be read in one piece is significantly shorter. These short sequencing *reads*, in early stages of NGS development as short as 30 base pairs, are harder to make sense of than the kilobase long Sanger reads. Especially, longer reads facilitate analysis of the many highly repetitive DNA stretches within a genome. For illustration one can imagine assembling a puzzle of a world map: if puzzle pieces are large, and contain some distinguishable part (perhaps part of a continent), they are easily placed. If pieces are small, possibly from somewhere within an ocean, we are faced with more uncertainty and potentially ambiguity. Larger pieces thus give us more information which contributes significantly to ease of placement. Analogously, the longer the read sequence is, the higher its power to bridge repetitive genome regions and reach into an informative, unique part of the genome.

As a consequence of this data flood, efficient bioinformatics methods are in great demand. While many tools from the Sanger sequencing era exist, such as for genome assembly and mapping of newly sequenced reads onto an existing reference genome sequence (Huang and Madan, 1999; Ning *et al.*, 2001), most of them scale poorly for NGS data. In addition, the emergence of novel NGS-based applications necessitates the development of entirely new computational methods. Sophisticated tools built on clever algorithms addressing all steps of computational NGS data analysis are quintessential for continuous progress and for exhausting NGS capacities to the fullest.

This work deals with the development and implementation of several novel methods for key steps of computational NGS data analysis. In particular, our focus is on methods for identifying genomic variants in re-sequencing data, i.e. characterizing a newly sequenced sample genome with respect to a reference genome. Adhering to the analogy above, in re-sequencing we have a guiding – albeit slightly different and imperfect – picture of the puzzle we aim to reconstruct. This stands in contrast to de-novo genome assembly, where we are essentially clueless about the underlying picture, or at least equipped with much less information. In genome re-sequencing, our ability to efficiently place reads to their correct location is of major importance and a prerequisite for

accurate identification of genomic variation. We therefore devised strategies for highly sensitive read mapping and for evaluating accuracy of read mapping tools (Chapter 3). Especially when reads span larger genomic variants, or when certain sequencing protocols for specific types of data are in use, specialized read mapping algorithms are required (Chapter 4). We developed two strategies for so-called partial read mapping, with applications in small regulatory RNA sequencing and in detection of structural variants that comprise more than a few base pairs. Major difficulties in variant detection after (supposedly correct) read placement are in distinguishing real variants from sequencing errors (Chapter 5), as sequencing machines unfortunately do not necessarily read out DNA fragments in an error-free fashion. Also, DNA sample preparation before sequencing may introduce biases that need to be taken into consideration when making a variant call.

Our tools have lead to the discovery of causal mutations in X-linked intellectual disability patients (Kalscheuer *et al.*, submitted) and have identified a surprisingly high number of a certain type of genomic variant, retroposed genes (Emde *et al.*, 2012). On comparison with other state-of-the-art tools we especially excel in sensitivity (Weese *et al.*, 2009; Emde *et al.*, 2010, 2012), an important factor particularly in disease studies. Especially noteworthy is that all of our methods are implemented in the well-established, high-quality C++ library SeqAn (Döring *et al.*, 2008) and are thus well-documented, well-maintained and integrated into a thoroughly tested framework. Several SeqAn tools, including the ones introduced in this work, have been developed in the last years, and are applicable to large-scale real world data sets while relying on solid theoretical foundations. In times where fast and efficient algorithm development and testing is of particular interest, the SeqAn framework offers a fruitful method development environment where data structures and algorithmic components are geared towards high performance and generic (re-)usability. Efficient computational methods as well as systematically designed biological studies will continuously contribute to progress in fundamental biological research and especially in diagnostics. With the expected continuous advances in sequencing technology – third generation sequencing technologies already at the doorstep – personalized healthcare is a near future in which computational aspects will play a key role.

Thesis Structure and Relevant Publications. In the following, we start by laying out the biological and methodological foundations in Chapter 2. Chapter 3 to 5 will present own methods in read mapping, specialized read mapping and variant calling, together with their evaluation. In chapter 6, we discuss our findings, conclude and give an outlook on current and future developments. Our results are mainly published in the following articles:

- *RazerS –fast read mapping with sensitivity control.* David Weese, Anne-Katrin Emde, Tobias Rausch, Andreas Döring, and Knut Reinert. *Genome Research (2009).*

- *MicroRazerS: rapid alignment of small RNA reads.* Anne-Katrin Emde, Marcel Grunert, David Weese, Knut Reinert, and Silke R. Sperling. *Bioinformatics (2010).*

- *A novel and well-defined benchmarking method for second generation read mapping.* Manuel Holtgrewe, Anne-Katrin Emde, David Weese, and Knut Reinert. *BMC Bioinformatics (2011).*

- *Detecting genomic indel variants with exact breakpoints in single- and paired-end sequencing data using SplazerS.* Anne-Katrin Emde, Marcel H. Schulz, David Weese, Ruping Sun, Martin Vingron, Vera M. Kalscheuer, Stefan A. Haas, and Knut Reinert. *Bioinformatics (2012).*

- *Draining the pond: 14 novel candidate genes for X-linked intellectual disability.* Vera M. Kalscheuer, ... Anne-Katrin Emde... Hans-Hilger Ropers. *Manuscript submitted.*

Chapter 2

Background

This background chapter lays the foundation for the following chapters where own contributions to next-generation sequencing (NGS) data analysis, in particular variant detection, will be addressed. In section 2.1, we briefly introduce some essential biological background, with focus on genomic variants in humans. We assume the reader is familiar with the terms DNA, RNA, protein, gene and gene structure, and has basic knowledge in genome biology including DNA replication, recombination, transcription and splicing. Next, we dive into the methodological background in section 2.2, introducing NGS data characteristics and the fundamental computational analysis methods.

2.1 Biological Background: Genomic Variation

Genomic variation is a natural consequence of evolutionary processes, introducing with certain frequency mutations during recombination and during DNA replication that escape correction by DNA repair mechanisms. Mutations happening in germ cells (resulting in *germline* variants) will be present in all cells of the developing organism and may thus be passed on to progeny and form part of the gene pool. Mutations in somatic cells (resulting in *somatic* variants) remain within the organism, possibly specific to one cell or tissue (e.g. cancer where mutations lead to uncontrolled cell growth). Typically, extinct or endangered species have a low amount of genomic variation within the population (O'Brien, 1994). This low genomic diversity decreases their ability to adapt to environmental changes and puts them at higher risk of extinction. Genomic diversity thus is a key component in a population's fitness.

Depending on the frequency of a genomic variant within a population it is classified as a *polymorphism* (allele frequency $\geq 1\%$) or as a rare *variant* (allele frequency $< 1\%$). Variants associated with or causal of a disease are expected to be rare, while polymorphisms present in many individuals are less likely to have pathogenic impact. Polymorphic variants are thus often excluded from suspicion when searching for disease-causing variants. The human variation catalogue compiled by the 1000 Genomes Project constitutes the most comprehensive population-scale variant set to date and is continuously growing towards a higher degree of completion,

improving its usefulness as a background variant set. However, in the absence of a complete catalogue, the terms polymorphism or variant are not necessarily used correctly due to lack of evidence. In this work, we will simply refer to "variant" irrespective of the population allele frequency.

Many different types of variation can be found in the (human) genome, all of which have been linked to diseases (Stenson *et al.*, 2009). In the following we first review and show examples of different types of variation. Next, we discuss their potential functional impact.

2.1.1 Types of Genomic Variation

Usually, a somewhat arbitrary size cutoff is used to divide genomic variation into two categories: small variants, for all variants up to the size cutoff, and structural variants (SVs), for all variants larger than that. The most common size cutoff in use is 50 bp (Alkan *et al.*, 2011). Figure 2.1 shows the most frequent types of variation, further subdividing SVs into balanced (no gain or loss of genetic material) and unbalanced SVs (associated with sequence gain or loss).

Small variants comprise single nucleotide variants (SNVs), where a single base pair is mutated, and short insertions/deletions (indels), where a few base pairs are inserted or deleted. SNVs are the most frequent source of genomic variation. On average two human individuals differ in 1 SNV per ~ 1 kb (International HapMap Consortium, 2003) and 1 indel per ~ 8 kb (Lunter, 2007).

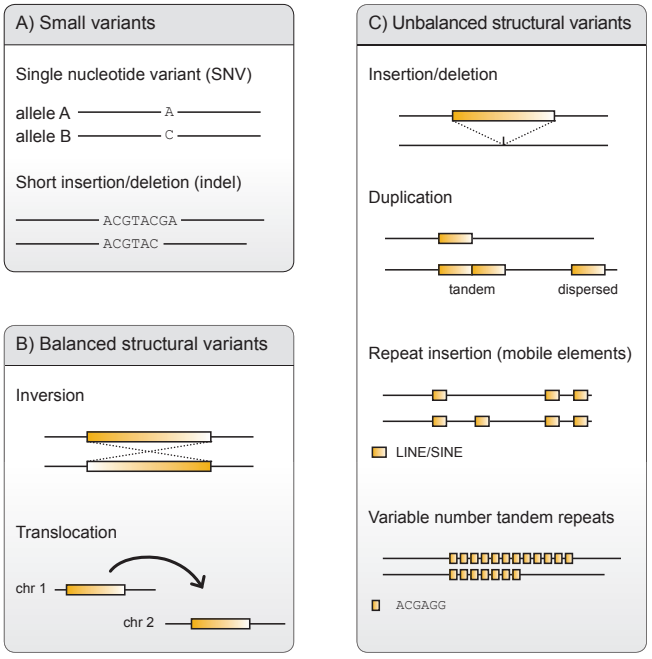

Figure 2.1: Genomic variation can be divided into small variants (A), and balanced (B) and unbalanced (C) structural variants. Irrelevant sequence here is shown as solid lines, whereas variable sequence segments are shown as yellow boxes. Small variants comprise single nucleotide variants (SNVs) and short indels smaller than 50 bp. Structural variants are variants larger than 50 bp. Unbalanced variants lead to a loss or gain of genetic material, whereas for unbalanced ones there is no such loss or gain.

SVs occur with lower frequency than small variants. However, in recent years it has been shown that human variation due to SVs is significantly more frequent than previously thought; while SVs vary strongly in size, it is estimated that on average at least $\sim$ 4-5 Mb of the individual human genome are involved in structural variation (Tuzun *et al.*, 2005; Redon *et al.*, 2006).

Balanced SVs include inversions and translocations where the inverted or translocated sequence is cut out and inserted without significant loss or gain of genetic material. Translocations can be intra- or interchromosomal, i.e. within one chromosome or between different chromosomes. Unbalanced variants (or copy number variants) are associated with significant sequence loss or gain, such as in a sequence deletion or novel sequence insertion. Further types of unbalanced SVs are duplications where sequence is copied and inserted either right after the original copy (tandem duplication) or at another location in the genome (dispersed duplication). Also, individual genomes often differ in their content of repeats. Repeats are families of highly similar sequences that can be found in large numbers in the human genome. The most common genomic repeats are long-interspersed nuclear elements (LINEs, $\sim$ 2-6 kb in size) and short-interspersed nuclear elements (SINEs, $<$ 500 bp). They are inserted into the genome through retrotransposition (DNA transcription to RNA, reverse transcription to DNA, and subsequent re-integration into the genome) and are therefore often referred to as mobile elements. Another special case of mobile elements are retroposed genes. Here, mature mRNA is reverse transcribed to DNA and re-integrated into the genome, producing a copy of the intronless transcript of the parental gene. Variable number tandem repeats (VNTRs) are expansions or contractions of short repetitive sequence, also called satellites, and are usually caused by polymerase slippage during replication. As VNTRs can be shorter than 50bp, they actually belong to both the small as well as structural variation class.

2.1.2 Functional Impact of Variants

Genomic variants can be anything from beneficial to neutral to functionally disruptive, i.e. showing a strong phenotypic variation, in the extreme case being lethal for the host cell/organism. Figure 2.2 shows how mutations can have functional impact on the protein level. We can differentiate between small variants in the coding sequence (A), variants disrupting the gene structure (B) and variants leading to gene dosage/regulatory effects (C).

Variants in the gene coding sequence can be *synonymous* or *non-synonymous*. A synonymous variant (not shown in figure) has no effect on the amino acid level, i.e. does not alter the amino acid sequence. A non-synonymous variant, however, can affect the amino acid sequence in different ways. We speak of a missense mutation when a SNV changes a codon such that a different amino acid is incorporated. A prominent example is a mutation in hemoglobin leading to incoporation of valine instead of glutamic acid, which causes sickle-cell disease (Ingram, 1957). Frameshift, nonsense and splice site mutations have been shown to be causal mutations in Duchenne muscular dystrophy (DMD) (Muntoni *et al.*, 2003), an X-chromosome-linked (X-linked) disease affecting the dystrophin gene that occurs in about one out of 3,500 male births. Also repeat expansion mutations that lead to incorporation of additional amino acids have been found to be the cause of many diseases (Orr and Zoghbi, 2007). One example is expansion of a short (CTG)-repeat in

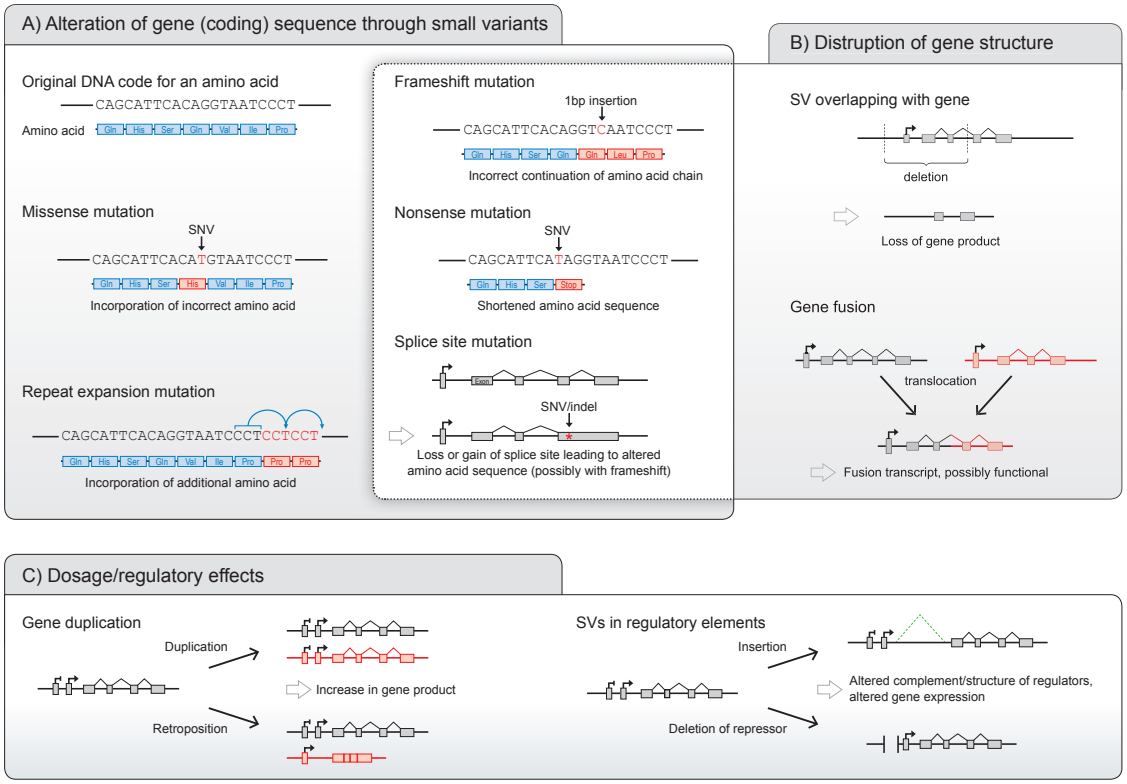

Figure 2.2: Functional impact of variants on the protein level can be divided into alteration in the coding sequence (A), disruption of the gene structure (B), and altered gene dosage (C). Small variants in the coding sequence can at the same time disrupt the gene structure (shown as overlap), for example when a splice site is mutated or a frameshift mutation disrupts the reading frame.

the DMPK gene, which leads to myotonic muscular dystrophy (Fu *et al.*, 1992), a different kind of dystrophy more common in adults.

Structural variants that overlap with a gene or several genes can have severe impact and result in loss of functional protein products. Again, we can see examples in muscular dystrophy where large deletions spanning several kilobases of the dystrophin locus are the genetic cause in more than 50% of DMD patients (Chamberlain *et al.*, 1988).

Gene fusions are formed when parts of two genes are joined by translocation[1]. Fusion transcripts are strongly associated with and often causal of cancer (Mitelman *et al.*, 2007).

Genes present in variable copy number (through duplication, deletion or retroposition) can have an influence through altered dosage. For example, high copy number of the beta-defensin gene has been shown to increase susceptibility to psoriasis (Hollox *et al.*, 2008), an autoimmune disease affecting skin cells. Differential protein dosage can also be caused by SVs or small variants affecting regulatory elements, leading to increased or decreased transcription levels.

On top of the direct influence variants can have on one allele, they can also play together in the diploid cell. Just as a functioning allele can correct for a damaged recessive one, loss of a

[1]Gene fusions can also result from *read-through* of adjacent genes, which is only visible on the RNA level

functioning allele can also unmask a damaged one (loss of heterozygosity).

In the following, we turn to the methodological background including sequencing technologies and computational analysis for genomic variant detection.

2.2 Methodological Background

The availability of high throughput DNA sequencing technologies has greatly paved the way towards studying genomic variation on a large scale (Durbin *et al.*, 2010). Different protocols for studying whole genomes (Bentley *et al.*, 2008; Wheeler *et al.*, 2008) or certain parts of a genome such as the exome (Ng *et al.*, 2009; Albert *et al.*, 2007) have emerged and have made it possible to study specific aspects in depth. We call this application of DNA sequencing where a reference sequence is available *genome re-sequencing*. Re-sequencing has been successfully applied in many studies to identify disease-associated genomic variants (Ng *et al.*, 2010a,b; Lupski *et al.*, 2010; Rios *et al.*, 2010). In general, we can differentiate between reference-guided and de-novo sequencing applications (Figure 2.3A and B). De-novo genome sequencing refers to the sequencing of an organism where no reference sequence is available yet. The main computational method here is an assembly algorithm that reconstructs the underlying sequence. Our focus however will be on reference-guided applications. In addition to genome resequencing, other reference-guided applications of DNA sequencing have arisen. For example, protocols have been developed for sequencing RNA, collectively called *RNA-Seq*, by extracting RNA from the cell, reverse transcribing and then sequencing the resulting cDNA (Wang *et al.*, 2009c). Usually, RNA-Seq refers to mRNA sequencing (where mRNA is captured by its poly-A-tails) and investigates alternative splicing and transcript expression levels within a certain tissue or cell line (Mortazavi *et al.*, 2008; Sultan *et al.*, 2008a). Another type of RNA-Seq is small RNA sequencing, where miRNA or other small regulatory RNAs are the center of interest (Morin *et al.*, 2008). *ChIP-Seq* refers to sequencing after chromatin immunoprecipitation of protein-bound DNA (Park, 2009). Just as its predecessor ChIP-Chip (Ren *et al.*, 2000) which uses array technology instead of sequencing, it has the power to identify binding of proteins such as transcription factors (Valouev *et al.*, 2008), but has the advantage of requiring less a priori knowledge.

Depending on the application, different computational analysis steps are necessary. However, most reference-guided applications fundamentally rely on read mapping with subsequent "feature" discovery. In the following sections we first introduce NGS data and it characteristics and then turn to the computational analysis protocol that all above mentioned reference-guided applications usually share. We then concentrate on genome resequencing and variant detection in more detail. For a review of other DNA sequencing applications, the interested reader is referred to (Mardis, 2008b).

2.2.1 Next-Generation Sequencing Technologies

NGS technologies are based on *sequencing-by-synthesis*. That is, complementary bases are added to a growing DNA sequence using a single stranded DNA fragment as template (synthesis), while

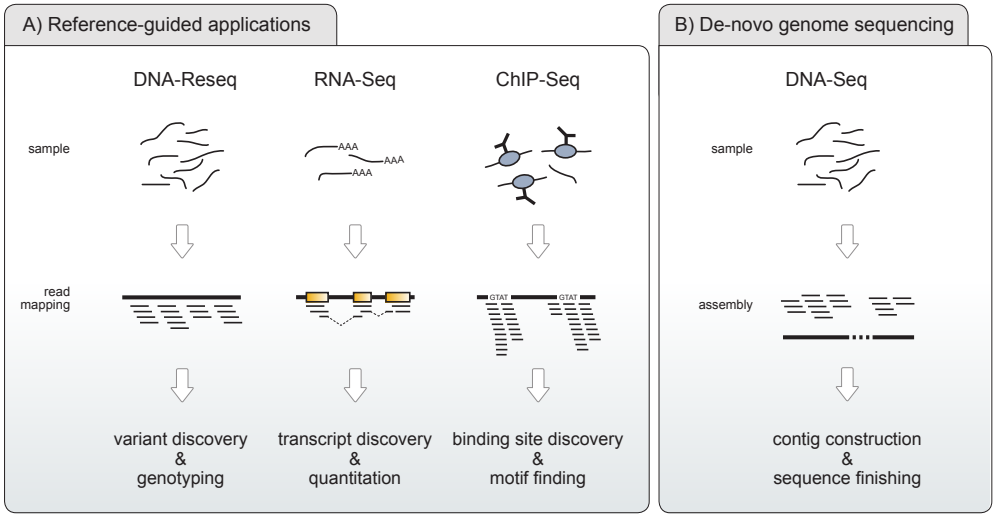

Figure 2.3: Sequencing applications can be divided into reference-guided applications (A) where a reference sequence is known and de-novo genome sequencing (B) where a new organism is sequenced. The probably most common reference-guided applications are resequencing of genomic sequence (DNA-Reseq), transcriptome sequencing (RNA-Seq), and sequencing of chromatin immunoprecipitated DNA (ChIP-Seq). The diverse reference-guided applications all employ a read mapping step, while the main computational step in de-novo sequencing is an assembly step.

incorporation of new nucleotides is recorded by a camera taking fluorescence images (sequencing). This process happens in a highly parallel fashion with hundred thousands to millions of DNA fragments sequenced simultaneously. Fluorescence images are then processed by a base calling algorithm that infers the sequence of incorporated nucleotides. In contrast to Sanger sequencing, no bacterial cloning step is necessary for fragment amplification; instead, DNA fragments are amplified directly on the technology-specific sequencing surface. While characteristics of sequencing data vary a lot depending on the technology, all NGS methods have in common that they generate orders of magnitude more data and shorter reads than Sanger sequencing. Table 2.1 gives an overview of throughput of Illumina, 454 and Solid technologies (the three most successful NGS pioneers), with the tremendous advances within the last four years shown. Especially Illumina has increased its throughput significantly and has thus become the most widely adopted sequencing technology.

Recently, so-called third-generation sequencing methods have started emerging (Schadt *et al.*, 2010; Branton *et al.*, 2008; Rothberg *et al.*, 2011). Also called single molecule sequencing methods, they do not require a fragment amplification step but work on single DNA molecules. These methods are expected to deliver longer reads and lower costs per run. As of yet, they are not widely adopted. However, the definite trend in DNA sequencing is decreasing costs with increasing throughput and/or data quality.

In the following, we briefly outline the Illumina and 454 technology, as these are most relevant for this work. For a detailed review of NGS technologies, see Mardis (2008b). As we mainly use Illumina sequencing data in this work, we will from then on assume Illumina read data unless stated otherwise.

Roche 454 sequencing: Specific adapter sequence is attached to each DNA fragment with which fragments then bind to agarose beads carrying the complementary adapter sequence. Each bead holds one DNA fragment which is then amplified by emulsion PCR. The beads, now holding about a million copies of the original DNA fragment, are then poured on a picoliter plate with hundred thousands of wells. Each well can hold a single bead and contains enzymes and reactants for *pyrosequencing*: when a nucleotide is incorporated pyrophosphate is released. Through a series of reactions this produces light that is in its intensity proportional to the number of nucleotides incorporated. The most common errors in 454 sequencing are insertion and deletion errors due to over- or undercalling the number of incorporated bases. Especially in long stretches of the same nucleotide (homopolymer runs) base calling is very unreliable due to saturation. Substitution errors however are rare for this technology.

Illumina sequencing: Also in Illumina sequencing, specific adapter sequence is attached to the DNA fragments. The fragment library is then poured onto a flow cell which consists of 8 channels (lanes). Fragments get attached to the flow cell surface through binding to complementary adapter sequence. Next, so-called bridge amplification results in clusters of about a million copies of the same DNA fragment. Sequencing-by-synthesis then proceeds in a cyclic fashion, incorporating one nucleotide per cycle in each fragment cluster. All four nucleotides are added simultaneously, and the appropriate nucleotide is added to each growing fragment. Nucleotides carry reversible terminators that ensure that only one nucleotide is incorporated per cycle. Furthermore, each nucleotide is fluorescently labeled. After fluorescence imaging, reactants are washed off, terminators are chemically removed, and another sequencing cycle can take place. In the end, all reads have the same length, as the number of sequencing cycles is the same for each cluster. Sequencing quality, i.e. certainty of incorporated nucleotide, decreases with each cycle With improvements in sequencing chemistry, more than 100 cycles are nowadays possible. The main type of error is base miscalling which stems mainly from clusters becoming desynchronized with increasing cycle number.

	Illumina		454		ABI SOLiD	
	2008	2012	2008	2012	2008	2012
Gb/run	1.3	600	0.1	0.9	3	155
Run time (days)	4	10	0.3	0.8	5	8
Read length	36	100	250	700	35	75
Mb/h	13.5	2500	14.3	45	25	807.3
US $/Mb	6	0.1	84	20	6	0.2

Table 2.1: Next-generation sequencing technologies and their throughput, 2008 compared to 2012. Data collected from (Mardis, 2008a) and sequencing company websites.

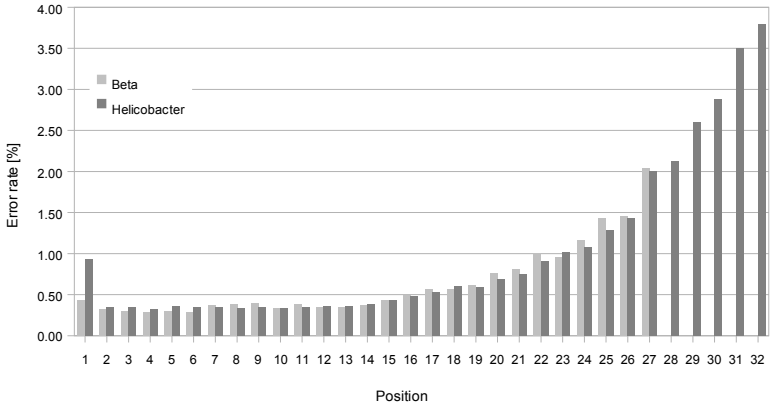

Figure 2.4: A typical Illumina error profile along read positions. Figure from (Dohm *et al.*, 2008), courtesy of Juliane Dohm.

2.2.2 Sequencing Data Characteristics

There are a number of characteristics of sequencing data that will be of relevance in this work. In the following, we introduce quality values and read pair sequencing. Next, we mention biases associated with certain sequencing protocols that need to be taken into account when analyzing sequencing data.

Sequencing Quality

Apart from identifying which nucleotides were incorporated, base calling algorithms also assign a measure of certainty, i.e. base call quality value, to each called nucleotide. This is an estimate of the probability of the base call being incorrect and is calculated from recorded signal intensities during image analysis. While some platforms initially used a different scaling method, these error probabilities can always be transformed to and are usually represented by a Phred scaled quality value (Ewing and Green, 1998):

$$Q_{\text{Phred}} = -10 \cdot log_{10} P(\text{error}) \tag{2.1}$$

Thus, a quality value of 10 means an error probability of 0.1, a quality value of 20 an error probability of 0.01 and so on. A typical, representative error profile of early Illumina reads (Illumina Genome Analyzer I) is shown in Figure 2.4 (taken from (Dohm *et al.*, 2008)), with error rate clearly increasing toward the 3'-end. Note that this profile is based on alignment errors, i.e. including real single nucleotide variants as well as sequencing errors. However, the occurrence of a variant is independent from read position, and thus the clearly visible increase in error rate towards the $3'$-end is solely due to sequencing quality degradation. Quality values are used in some downstream computational analyses and, for example, aid in distinguishing small variants from sequencing errors.

Read Pair Sequencing

Read pair sequencing produces pairs of reads with known relative layout. There are two commonly used protocols. The main principle behind *paired-end* sequencing is a size selection step where DNA fragments of a specific length (e.g. 400 bp) are isolated by use of gel electrophoresis. The size-selected fragments, which all have approximately the same size, are sequenced first from one end and then from the other end, producing read pairs. While we do not know the sequence in between the read pairs, we do know relative orientation and approximate distance of the two reads. So-called *mate-pair* sequencing involves a different sample preparation protocol, that produces read pairs with different orientation. Essentially, the ends of large size-selected DNA fragments (e.g. 3 kb) are tagged with biotin and then circularized. After random shearing, fragments containing biotin are extracted, again size-selected and sequenced from both ends as in paired-end sequencing. As we know the length of the originally circularized fragment, we know the approximate distance between the read pairs. In contrast to paired-end sequencing, sequenced read pairs face away from each other.

In both paired-end and mate-pair sequencing, the outer distance of the reads is called the insert size. On the one hand, the read pair approach makes sample preparation and sequencing more laborious and costly. On the other hand, it provides valuable information for downstream computational analysis, as we will see later.

Biases

Several steps of the sequencing protocol can introduce biases. Initial DNA fragmentation, i.e. shearing, is not entirely random, leading to biases in produced DNA fragments (Schwartz and Farman, 2010). Furthermore, DNA fragments with extreme GC content, in particular very low GC content, are harder to sequence and amplify (Aird *et al.*, 2011). This leads to underrepresentation of these sequences, or relative overrepresentation of sequences with balanced GC content.

Targeted re-sequencing refers to a special sequencing protocol where specific genomic regions of interest are targeted and captured, for example with an oligonucleotide array (Ng *et al.*, 2009). Certain sequences are harder to target as it is harder to design unique probes (this affects repetitive sequence), and some are harder to capture due to lower binding affinity (Zhang *et al.*, 2003). Again, this leads to GC bias. Moreover, preferential capture of the reference allele introduces bias towards the reference allele and may complicate identification of variant alleles (Turner *et al.*, 2009).

2.2.3 Computational Analysis Framework

After sequencing, computational analysis now deals with a set of sequenced reads including quality values and possibly read pair information. From this information, we want to reconstruct the underlying events or features. The main computational steps for reference-guided analyses can be divided into four stages:

1. Read cleaning and pre-processing: read filtering and clipping, adapter or tag removal, error correction

2. Read mapping: finding the location of origin of each read in a reference sequence, possibly including several and specialized types of read mapping depending on the application
3. Postprocessing of mapped reads: multi-read assignment, PCR artifact removal, error correction
4. Feature discovery: depending on the application e.g. genomic variant detection and genotyping or transcript detection and quantification

The details of steps 1 and 3 can vary widely and sometimes these steps, especially step 3, are even skipped. Their importance depends on the type of application and the quality and type of sequencing data. In the following, an overview of each of these steps will be given, mainly with the application of genome resequencing in mind, i.e. with the goal of detecting genomic variants, such as SNVs, indels, and structural variants.

Read Cleaning

There are different sources of noise in NGS data, some of which can be treated before reads are mapped. Apart from making data less noisy for downstream analysis, this also has the advantage of reducing the read set and thereby saving computational resources. Read cleaning approaches can be roughly divided into simple quality-based clipping or filtering strategies (Smeds and Künstner, 2011), more sophisticated sequencing-error correction methods (Salmela, 2010; Kelley *et al.*, 2010) and contaminant-removal procedures (Schmieder *et al.*, 2010; Chen *et al.*, 2007).

In quality-based clipping (Smeds and Künstner, 2011), bases with low base call quality are clipped from the 3' end or from both ends of a read. If the overall quality of a read is low, the whole read may be discarded. This removes read (sub)sequences likely to be sequencing error-prone. Error correction methods, on the other hand, aim to correct read bases likely to be sequencing errors by using information from other reads. These methods are usually based on k-mer counting or suffix array algorithms (Yang *et al.*, 2012) implemented in stand-alone tools (Salmela, 2010; Kelley *et al.*, 2010) or as built-in components of de-novo assembly tools (Zerbino and Birney, 2008). Contaminant removal can be necessary for some sample preparation protocols that lead to reads containing adaptor/tag sequence or viral contamination. Strategies basically rely on alignment methods such as Blat (Kent, 2002) for contaminant identification and masking (Schmieder *et al.*, 2010; Chen *et al.*, 2007). However, with all read cleaning approaches there is always the risk of losing information through over-cleaning. Depending on the type of sequencing data and application, this step is frequently skipped or limited to basic quality-control.

Read Mapping

The goal of read mapping is to locate for each sequenced read its originating position in a reference sequence. Read mapping is thus an alignment problem where the read is usually allowed to differ slightly from the reference sequence, i.e. some alignment errors are allowed.

There are essentially two major difficulties in read mapping: Firstly, the tremendous amount of data necessitates extremely efficient algorithms. Depending on the sequencing platform and type of

application and sequencing depth, the set of input reads can be in the range of billions of short reads ($< 100bp$). Secondly, the ambiguous nature and error-proneness of the data complicates confident read assignment. The longer the reads, the higher the probability that they can be unambiguously mapped to a reference sequence position (Figure 2.5), i.e. the higher the *uniqueness* or *mappability*. Using paired-end sequencing also has a positive effect on mappability, especially with increasing fragment insert size (Chikhi, 2012). Obviously, mappability furthermore strongly depends on the organism.

Characteristics of sequencing data and patterns of sequencing errors are specific to the different sequencing technologies. Thus, read mapping tools specifically geared toward certain types of sequencing data have been developed (Li *et al.*, 2008a; Rumble *et al.*, 2009). Also, the type of experiment needs to be taken into consideration. For example, in RNA-Seq data reads may span spliced out introns which makes mapping even harder (Trapnell *et al.*, 2009; Au *et al.*, 2010; Wang *et al.*, 2010).

Different types of read mapping algorithms will be the focus of chapters 3 and 4 of this thesis.

Postprocessing of Mapped Reads

Postprocessing of mapped reads addresses assignment of reads mapped to multiple locations (*multi-reads*), removal of noise or errors only visible after alignment, and correction of minor alignment artifacts.

The most straight-forward and common way to deal with multi-reads is to ignore them in subsequent analysis, which however results in an obvious loss of information. Another strategy is to distribute them randomly to their possible mapping locations (Li *et al.*, 2008a). However, more sophisticated methods for assigning multi-reads have been developed (Hashimoto *et al.*, 2009; Ji *et al.*, 2011) that use coverage and sequence information from reads mapped to flanking sequence. Concerning noise removal, PCR amplification artifacts can be identified after mapping as stacks of reads mapping to the same genomic coordinates. It is common to keep only a certain number

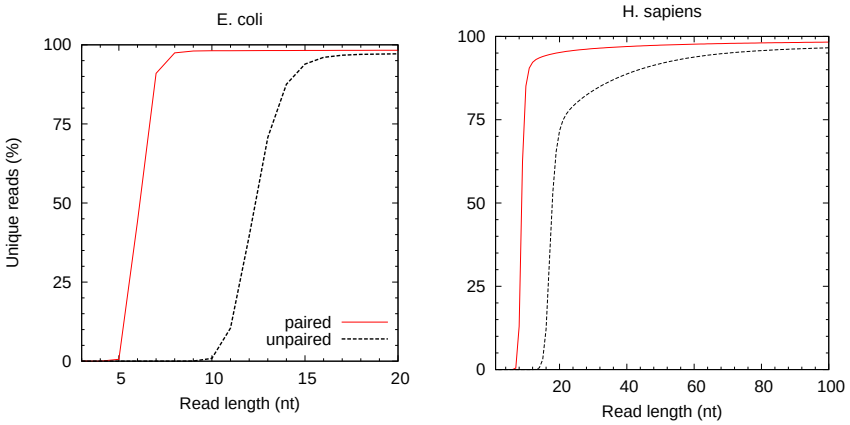

Figure 2.5: Uniqueness of single and paired-end reads for the E. coli and H. sapiens genomes plotted over read length. Paired uniqueness is shown for a fragment size of 300 bp. Figure from (Chikhi and Lavenier, 2009), courtesy of Rayan Chikhi.

of reads from each stack (no more than expected from overall sequencing coverage) to avoid propagation of errors possibly introduced during amplification (Li *et al.*, 2009a). Furthermore, alignment errors unlikely to stem from real variation can be removed with *a posteriori* read clipping, i.e. read clipping not only based on quality values but also on alignment errors (Li and Durbin, 2009). Finally, as regular read mapping maps reads in a pairwise read-to-reference fashion, there may exist small alignment inconsistencies and artifacts. By looking at all reads mapped to a location, realignment methods correct small alignment errors and make the multiple-read-to-reference-alignment more consistent (Homer and Nelson, 2010). Realignment is very important for small variant detection and will be treated in more detail in Chapter 5.

Feature Discovery

Feature discovery, obviously a very broad category, depends on the type of sequencing data and the goal of the experiment. The three probably most common types are shown in Figure 2.3: detecting transcript expression in RNA-Seq data, identifying protein-bound DNA in ChIP-Seq data, or discovering genomic variants in DNA resequencing data.

In the first case, RNA-Seq, the goal is to detect which transcripts are expressed in the sample, including prediction of alternatively spliced transcript isoforms. The number of reads mapped onto a transcript isoform is then used to quantify expression. Several tools for RNA-Seq data analysis have been developed, including pipelines that start with read mapping (usually multiple steps of mapping) and finally output transcripts and expression levels (Mortazavi *et al.*, 2008; Trapnell *et al.*, 2009, 2012). Specific tools are available for detecting gene fusions, of special interest in cancer RNA-Seq (Maher *et al.*, 2009; Iyer *et al.*, 2011).

In ChIP-Seq data, the goal is to identify which sequences have been bound by a protein, such as a transcription factor. Many tools have been developed for detecting binding peaks (Zhang *et al.*, 2008a; Zang *et al.*, 2009) where the difficulty lies in distinguishing real binding from background noise. Once bound sequences have been identified they can be further searched for binding motifs (Bailey, 2011).

The last case, genomic variant discovery, will be treated with more detail in the following section.

2.2.4 Genomic Variant Detection Methods

We differentiate between small variant and structural variant (SV) detection. Methods for detecting small variants mainly rely on inspecting the alignment of the mapped reads, looking at all reads overlapping candidate variants and trying to distinguish real variants from sequencing errors by using cutoffs or statistical calculations. Chapter 5 will deal with small variant detection in more detail. Here, we want to focus on methods for SV detection. SVs are of special interest, because they are more likely to be gene-disruptive and are at the same time harder to detect. Reference-guided SV detection methods can be divided into three categories: read pair, read depth, and split read methods, as shown in Figure 2.6.

Most SV detection methods exploit read pair information (Korbel *et al.*, 2009; Chen *et al.*, 2009; Lee *et al.*, 2009; Hormozdiari *et al.*, 2009). As approximate distance and relative orientation of read pairs are known, shifts in the mapped distance or changes in relative orientation can be used to locate SV events. The main advantage of these methods lies in the wide range of SVs they can detect (green boxes in Figure 2.6). However, read pair methods require tight insert size distributions for accurate discovery of small to medium-sized SVs and for confident localization of breakpoints (Alkan *et al.*, 2011).

Split read methods (Ye *et al.*, 2009; Karakoc *et al.*, 2012; Emde *et al.*, 2012) directly take advantage of reads crossing a breakpoint and can thereby detect SVs with base pair precision. Splitting, however, immensely increases mapping ambiguity and hence decreases mapping confidence. To increase confidence in split-read mapping, it is usually used in conjunction with read pair data for so-called anchored split read mapping. Here, the split read is anchored by a confi-

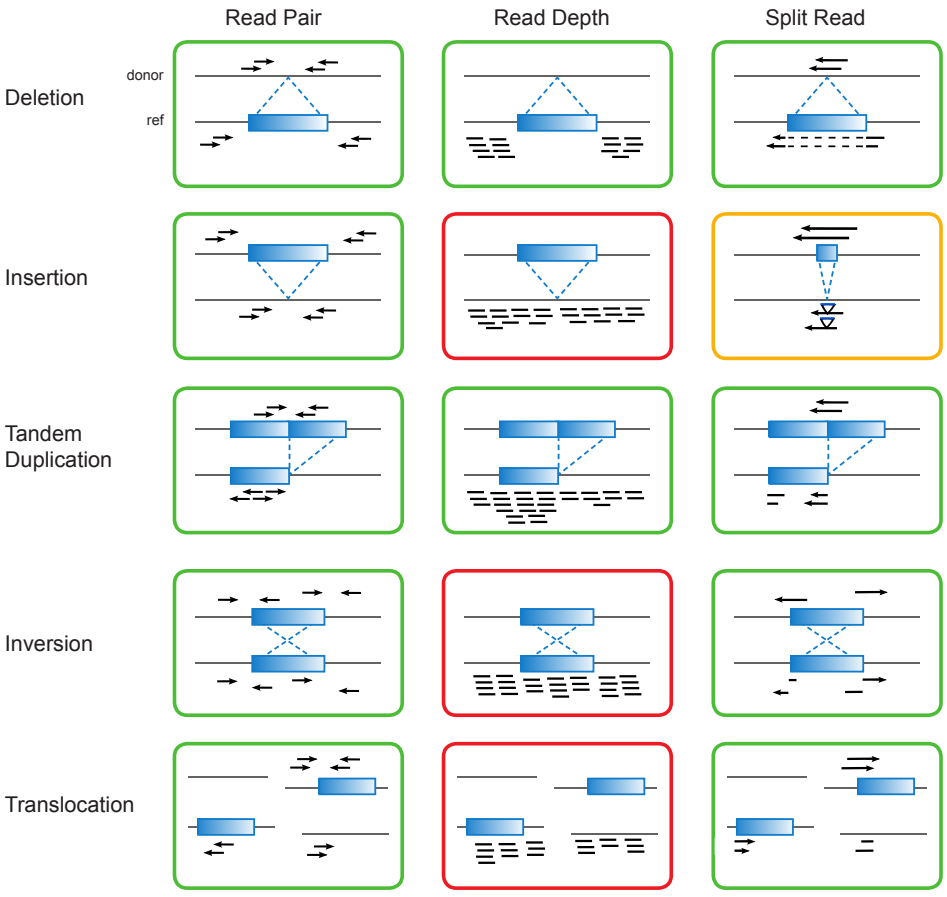

Figure 2.6: The three main read mapping-based strategies for SV detection using sequencing data. Read pair methods are able to detect all types of SVs (indicated by green boxes), read depth methods can only detect unbalanced SVs, but no novel insertions or balanced SVs (red boxes), and split read methods can theoretically identify all SV types but have limited power for novel insertions due to read length (yellow box).

dently mapped paired read which greatly decreases search space for the split read. This strategy was first developed by Ye *et al.* (2009) and is particularly useful for short reads ($< 100bp$). In chapter 4 of this thesis, we will introduce our own split read based SV detection method which can deal with paired-end as well as single-end data (Emde *et al.*, 2012).

The third strategy shown in Figure 2.6, read depth methods (Xie and Tammi, 2009; Yoon *et al.*, 2009), works on single-end as well as paired-end data and is able to detect very large deletions and duplications, i.e. changes in copy number, by comparing the observed number of mapped reads in a genomic region to the expected one. These methods can detect large unbalanced SVs, i.e. copy number variants (CNVs), but provide low resolution and hence no exact breakpoints (Alkan *et al.*, 2011). Furthermore they are vulnerable to coverage fluctuations due to mapping and sequencing biases (Medvedev *et al.*, 2009).

Apart from the three approaches shown in Figure 2.6, reference-guided assembly methods are becoming more common (Hajirasouliha *et al.*, 2010; Li *et al.*, 2011). These methods do a local de-novo assembly, often using one-end anchored reads, and therefore possess the unique ability to detect large novel sequence insertions. In addition, completely different methods for SV breakpoint detection exist, e.g. the BreakPointer (Sun *et al.*, 2012) method that exploits read mapping artifacts close to SV breakpoints (accumulation of alignment errors and sudden drop in coverage depth).

One important finding of the 1000 Genomes Project has been that the different SV detection strategies share a rather low overlap of predicted SVs (Alkan *et al.*, 2011). Due to the different strengths and weaknesses, none of the methods is comprehensive. Hence, recent developments have brought new tools that combine these strategies in order to increase SV detection sensitivity. For example, SVseq (Zhang *et al.*, 2012) combines split read and read pair information, while GenomeSTRiP combines the read pair with the read depth approach (Handsaker *et al.*, 2011). Nevertheless, these combined strategies depend on accurate and efficient basic steps, mostly using the strategies briefly introduced here as building blocks.

2.2.5 Sequencing-Based vs. Array-Based Methods

Large unbalanced SVs (CNVs) can also be detected with array-based techniques (Iafrate *et al.*, 2004; Redon *et al.*, 2006; Conrad *et al.*, 2010). In fact, array-based methods have so far been the major strategy for inferring copy number of genomic sequences. There are two major array techniques for this purpose: array comparative genomic hybridization (array CGH) (Iafrate *et al.*, 2004) and SNP microarrays (Bignell *et al.*, 2004), both based on similar principles. Most importantly, in both cases array design requires solid prior knowledge of the genome. In arrayCGH, differentially labeled test and control DNA are competitively hybridized on the probe array and copy number ratio is inferred from fluorescence signal ratios. The probes are usually long oligonucleotides ($\sim 50\,bp$) or bacterial artificial chromosome (BAC) clones. Also SNP arrays carry oligonucleotide probes, but probe sequences are shorter ($\sim 20\,bp$) and designed to be specific to single nucleotide differences. Only test DNA is hybridized to the SNP array and binding intensity measured. To generate a copy number ratio as in arrayCGH, the measured intensities are compared to signal

intensities of control DNA on a different array.

In general, arrayCGH offers a lower signal-to-noise-ratio than SNP arrays (Greshock *et al.*, 2007; Alkan *et al.*, 2011). However, SNP microarrays carry SNP allele-specific probes and can thereby detect CNV alleles with higher sensitivity (Alkan *et al.*, 2011). Both technologies have difficulty to infer high copy counts due to saturation and have limited power to detect variation in repetitive regions.

The cost-effectiveness of array techniques for CNV detection has enabled their use in diagnostics (Zhang *et al.*, 2008b). However, with the decreasing cost, sequencing is becoming the standard method for CNV detection, as it offers several advantages. Not only is sequencing in principle free from the necessity of prior knowledge of the genome, but also it does not suffer from saturation. In theory, it is able to detect all copy number ratios and in general offers a lower noise level. Furthermore, SV detection at base pair resolution is possible, and the full range of genomic variation including balanced SVs can be discovered (Alkan *et al.*, 2011). While repeat sequences are difficult to map and analyze, they are nevertheless theoretically detectable with sequencing data, and will mostly require longer reads and fragment library sizes as well as more sophisticated computational handling.

Apart from cost, another limiting factor for the use of sequencing-based SV detection in diagnostics has so far been a lack of experience in reliable computational handling; especially, with relatively short read lengths as well as sequencing biases, which can complicate analysis. But with coming advances in sequencing technology and, importantly, in computational method development, sequencing will most likely become the method of choice in the near future. The availability of efficient algorithms for clever handling of the huge amounts of sequencing data and for comprehensive variant detection will play a key role in this process.

2.3 Overview of Thesis

This thesis deals with some of the main computational steps in NGS data analysis, focusing on read mapping and variant detection. The methods developed within this work aim at being flexible to the advances in NGS technology and sufficiently efficient to handle large datasets. Ultimately, this work thus contributes to advancing computational method development towards comprehensive and efficient algorithmic tools for NGS data. In Chapter 3, we will focus on "general-purpose" read mapping which finds application in all reference-guided studies. Apart from introducing a read mapping tool with sensitivity control (joint work with David Weese), we will shed light on the non-trivial task of benchmarking read mapping tools (joint work with Manuel Holtgrewe). The subject of Chapter 4 will be two strategies for specialized, partial read mapping. We apply the first method to small RNA sequencing data (joint work with Marcel Grunert). Algorithmically an extension to this method, we then introduce a tool for split-read mapping. Employed on different data sets, we demonstrate its flexibility and high accuracy in detecting large indel variants. Chapter 5 then concentrates on small variant detection and introduces a tool for SNV and indel calling. All methods developed in this work have been implemented within the SeqAn C++ library (Döring *et al.*, 2008), a comprehensive library of C++ functionality for sequence analysis (more detail

in the following section). This makes our tools easily available to the community, reusable and adaptable for possible changes or special purposes, and, very importantly, efficient.

2.3.1 SeqAn

SeqAn (Döring *et al.*, 2008) is an open-source C++ library providing extensive functionality for biological sequence analysis. Its core components are efficient data structures and algorithms available for programmers, but also an increasing number of SeqAn applications for direct use by program users is becoming available. SeqAn uses a generic template-based design and its main goals are efficiency, integrability, generality and usability. Algorithmic components such as standard alignment algorithms are implemented in a reusable fashion such that they may be employed and reused in many applications. Since 2005, SeqAn is under active development and continuously growing. Automatic nightly builds and tests ensure high quality. Any tool and code that is part of the library is thus required to meet SeqAn's quality standards.

2.3.2 Developed SeqAn Tools

A number of tools that are available from the SeqAn project webpage (www.seqan.de) were developed in the course of this work:

1. RazerS: a general read mapping tool for gapped and ungapped semiglobal mapping (chapter 3). Joint project with David Weese, own contribution in tool development (mainly ParamChooser module/tool), testing and manuscript preparation.

2. MicroRazerS: a prefix-based read mapping tool, specifically for application in small RNA sequencing, built on RazerS core engine. Joint project with Marcel Grunert, own contribution mainly in tool development and manuscript preparation.

3. SplazerS: a split-read mapping tool, employing a prefix and suffix alignment strategy, which can identify deletions of arbitrary size and up to medium sized insertions, built on RazerS core engine.

4. SnpStore: a variant detection tool that uses a set of mapped reads and a reference sequence as input and calls and genotypes SNVs and indels.

5. Helper tools: indelSimulator and variantComp, for simulation of a variant-containing sample genome and comparison of predicted with reference variants, respectively.

Chapter 3

Read Mapping

Given a set of sequenced reads, the goal of read mapping is to find the location of origin of each read in a reference genome. In this chapter we address the most conventional type of read mapping, *semiglobal* read mapping, where the read sequence needs to align with its entire length in order for the read to be mapped successfully. Semiglobal read mapping finds application in most reference-guided NGS data analyses, including genome resequencing and RNA-Seq (Bentley *et al.*, 2008; Mortazavi *et al.*, 2008). It is one of the most fundamental steps in NGS data analysis (as seen in the computational analysis framework introduced in section 2.2.3) and of particular importance as all subsequent steps will be based on the mapped reads.

From a computer scientist's point of view, read mapping is an approximate string search problem: we search for approximate matches of many short sequences (the reads) in a long text (the reference sequence). Difficulties lie mainly in the computational complexity due to the huge number of reads (millions to billions) and the length of the reference sequence (for example 3 billion base pairs for the human genome). Also, reads are relatively short (30 to 150 bp), making it hard to unambiguously assign them to a specific location in the genome, especially in the presence of sequencing errors and repeat sequences.

In this chapter, we will introduce the basic algorithms and concepts that play key roles in read mapping, placing focus on semiglobal read mapping. First, we will give the necessary notation. Next we turn to read mapping algorithms and point out their basic steps, which will serve as background knowledge for the remainder of this and also for the following chapter. After briefly describing some different read mapping tools, we will then focus on the q-gram counting based RazerS (Weese *et al.*, 2009) which was developed in collaboration with David Weese. We will focus on own contributions which are predominantly in RazerS' parameter choosing module (section 3.4) and in the comparison and evaluation of read mapping tools (sections 3.5). Section 3.6 presents further results for benchmarking of read mapping tools, which was done in collaboration with Manuel Holtgrewe.

Notation

Given a set of reads R where $r \in R$ and a reference sequence g, where r and g are sequences (or strings) over the alphabet $\sum = \{\texttt{A,C,G,T,N}\}$. A subsequence (or substring) of a sequence s starting in position i and ending in position j is denoted by $s_{i,j}$. Furthermore, the operator $|\cdot|$ denotes the length of a sequence or the size of a set. Given an alignment $a_r^{g_{i,j}}$ of a read sequence r and a reference subsequence $s_{i,j}$, we define a distance function $d(a_r^{g_{i,j}})$ that returns a validity measure of an alignment (see section "Distance Functions" below). Super- and subscripts will be dropped when the context is clear.

Semiglobal Read Mapping

Given a read $r \in R$, a reference sequence g, a distance function d and a maximal distance k, a *valid match* of r is an alignment $a_r^{g_{i,j}}$ with $d(a_r^{g_{i,j}}) \leq k$.

For each read, there may exist multiple valid matches. A *best match* a' of read r is one for which holds $d(a') = \min_{a \in A}(d(a))$ where A is the set of all valid matches of r. If only one valid match exists for a read, the read is considered *unique*. For short reads, it is common to call a read unique, if it has only one valid best match, i.e. suboptimal matches may exist.

Generally, the goal in read mapping is to find all valid matches, but different objectives are in use, as defined in (Holtgrewe *et al.*, 2011): apart from finding all valid matches (the *all* problem), it is common to report all best valid matches only (*all-best*), or up to c best valid matches (c-*best* or *any-best* if $c = 1$).

Distance Functions

Different distance functions are used, Hamming and Levenshtein (edit) distance being among the most popular choices in short read mapping. In the case of Hamming distance, the alignment is allowed to have matches and mismatches only, and distance function d returns the number of mismatches in the alignment. For Levenshtein distance, gaps are allowed additionally, and d returns the sum of the number of gaps and mismatches.

However, some read mapping tools (Rumble *et al.*, 2009; Li and Durbin, 2009) measure alignment similarity by using a scoring function as in Smith-Waterman alignment (Smith and Waterman, 1981). In addition, base call quality values may be incorporated into the distance function, penalizing alignment errors as a function of sequencing quality of mismatched bases (Li *et al.*, 2008a). In this work, we will concentrate on Hamming and Levenshtein distance read mapping.

3.1 Background: Filtering and Verification Methods

Figure 3.1 illustrates the two basic steps that efficient read mapping algorithms usually rely on. The first step identifies potential matches using a fast filtering technique based on an index data structure. Identified potential matches are then inspected by an exact and usually slow alignment

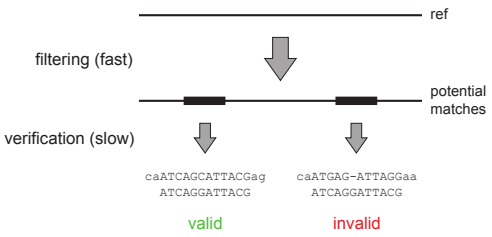

Figure 3.1: Many read mapping algorithms are based on two steps: a fast filtering step that identifies potential matches and a slow alignment verification method that evaluates the potential match and classifies it as true (valid) or false (invalid).

method. If the alignment fulfills the maximum distance criterion k it is returned as a valid match, i.e. classifies as a true positive of the filter.

In the following we briefly review different filtering techniques, including index data structures commonly in use and alignment algorithms for verification of potential matches.

Notation

We call a short sequence of length q a *q-gram*[1]. q-grams shared between two sequences are called *q-matches*. A *gapped q-gram* or *shape* is a set $Q = \{p_1, p_2, ..., p_s\}$ of positions $p_1 < p_2 < ... < p_s$. Here, we enforce that $p_1 = 1$. The *span* $s(Q)$ is the maximum entry in Q, i.e. s(Q) $= p_s$. The *weight* q or $w(Q)$ is the cardinality of the set $|Q|$. Instead of the set notation, we will sometimes use the more graphical representation of shapes, e.g. ###-##-# for $Q = \{1, 2, 3, 6, 7, 9\}$, where all "#" represent positions contained in Q and all "-"-positions are joker-positions not contained in Q. One-gapped shapes are a special case of gapped shapes that contain exactly one contiguous, arbitrary length stretch of jokers. A shape can be applied to $n - s(Q) + 1$ overlapping positions in a sequence of length n, thereby producing $n - s(Q) + 1$ q-grams.

An alignment *transcript* is a string over the edit operation alphabet {`M,R`} in the case of Hamming distance, and {`M,R,I,D`} in the case of Levenshtein distance; where `M`, `R`, `I` and `D` correspond to match, replacement (mismatch), insertion and deletion, respectively. An alignment $a_{s_2}^{s_1}$ between two sequences s_1 and s_2 is unambiguously described by exactly one transcript T. The distance $d(a_{s_2}^{s_1})$ is given by the sum of occurrences of characters `R`, `I`, and `D` in T, i.e. the sum of error operations, which we also denote as $||T||$. A Q-match produced by two matching Q-grams can then be represented by a transcript sequence of length q that consists of `M` operations only.

We will also use the term *seed* to refer to a short match.

3.1.1 Filtering Techniques

A filter is characterized by its sensitivity and selectivity. Filter sensitivity is the fraction of true positive matches returned among all valid matches. Filters that have sensitivity = 1 hence guarantee to identify all valid matches and are called lossless, while filters with sensitivity < 1 may

[1] Alternatively, the term k-mer for a short sequence of length k is often used in the literature.

miss valid matches and are called lossy. The selectivity of a filter is the fraction of true positive matches returned among all potential matches returned. Thus, the lower the selectivity of a filter, the higher is the fraction of false positive potential matches. Usually, sensitivity increases when selectivity decreases and vice versa. Therefore, filter algorithms usually try to find a good tradeoff between the two, or try to maximize selectivity while remaining lossless.

By inspecting the properties of valid match transcripts, efficient filters can be designed. Filtering approaches can be roughly divided into *seed*-based vs. *counting* approaches. While in seed-based filtering a single exact match serves to seed/initiate an alignment, counting approaches look for regions containing at least a certain number of matches and return those regions as potential matches. Furthermore, we differentiate between single shape approaches and approaches based on multiple shapes (also called seed or shape families). Usually, a high match weight q is desirable for efficiency, as we want to avoid random q-matches whose probability is inversely proportional to $|\sum|^q$. Two of the most prominent principles in filtering are the q-gram counting lemma and the pigeonhole principle, both described in the following.

Pigeonhole principle

Formalized by Myers (1994) for approximate string matching, and here adapted to fit our notation, the pigeonhole lemma states:

Lemma 3.1.1. *Given an alignment $a_s^{s'}$ of two sequences s and s' with Hamming or Levenshtein distance $k = d(a_s^{s'})$, let $s = s^1, s^2, ..., s^j$ be a concatenation of arbitrary-length non-empty subpatterns, and $m_1, m_2, ..., m_j$ non-negative integers such that $M = \sum_{i=1}^{j} m_i$. Then, for some $i \in 1, ..., j$ s' includes a substring s" such that $d(a_{s^i}^{s''}) \leq \lfloor \frac{m_i k}{M} \rfloor$.*

The basic intuition behind this principle is easily seen in an example (Figure 3.2A). Given an alignment with k errors ($k = 2$ in the example figure), if the alignment transcript is divided into k+1 non-overlapping parts, there must exist at least one part that contains no errors. The corresponding transcript T thus contains at least one $\lfloor \frac{|T|}{k+1} \rfloor$-match. This holds for Hamming as well as Levenshtein distance.

Similarly, if the alignment is divided into k+2 parts, there must be at least two parts without errors. Several early NGS read mapping tools (first used in (Cox, 2006)) exploit this special case of the pigeonhole lemma for $k = 2$. Instead of searching for two parts independently, they use a seed family of one contiguous and two one-gapped shapes in order to increase the weight of the seed (Figure 3.2C). This works for Hamming distance, as mismatches placed within the gap are tolerated. However, insertions and deletions lead to a different distance between the two matching parts in reference and read, and are thus not tolerated. Therefore, this special two-seed-pigeonhole strategy is lossless only for Hamming distance up to $k = 2$, and lossy for Levenshtein distance $k > 1$.

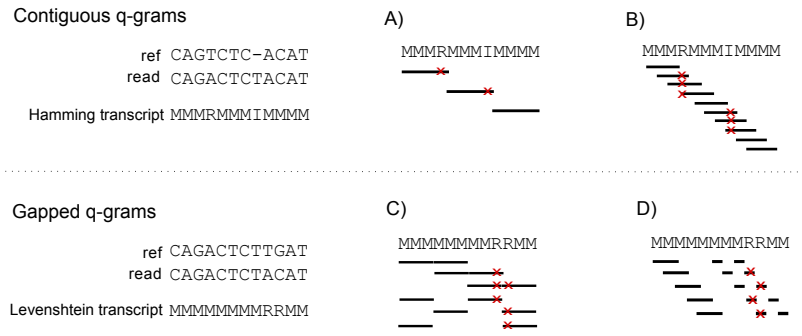

Figure 3.2: Example of filtering techniques for k = 2. The first row shows an alignment of length 12 with Levenshtein distance 2. A) Following the pigeonhole principle there must be at least one matching contiguous (ungapped) 4-gram or B) following the q-gram counting approach with overlapping q-grams there must be at least 4 matching 3-grams. The second row shows an alignment of length 12 with Hamming distance 2. C) According to the two-seed pigeonhole principle, which uses an ungapped and two gapped shapes, there must be at least one matching (gapped or ungapped) 6-gram or D) according to the gapped q-gram counting approach there is at least one gapped 4-gram of shape **##---#-#**. Note that the gapped shape would not tolerate insertion or deletions errors.

Q-gram counting

While the pigeonhole principle looks at non-overlapping seeds or seed parts, the q-gram counting approach uses overlapping seeds. The q-gram lemma (Jokinen and Ukkonen, 1991) states:

Lemma 3.1.2. *Given an alignment $a_s^{s'}$ of two sequences s and s' with Hamming or Levenshtein distance $k = d(a_s^{s'})$, then s and s' share at least $t_0 = max(|s|, |s'|) - (k+1)q+1$ common q-grams.*

This lemma holds for ungapped q-grams only. We call t_0 the optimal lossless threshold, i.e. the highest threshold value that guarantees full sensitivity. A high threshold value t is preferred to a lower one, as this increases selectivity (given the same q). A q-gram counting filter is then characterized by the tuple (q, t). An example of the worst case error scenario of a (3,4)-filter is shown in Figure 3.2B.

Placing gaps into the q-gram has been shown to dramatically increase sensitivity for Hamming distance (Burkhardt and Kärkkäinen, 2001). Or conversely, one can increase the weight of the gapped q-gram while maintaining the same sensitivity as with an ungapped q-gram, but at the same time increasing selectivity. For example, for sequence length $n = 18$ and $k = 2$, one can find a gapped 7-gram that has $t_0 > 0$ while the ungapped 7-gram cannot guarantee full sensitivity with a threshold > 0. However, no closed formula for the minimum number of q-matches, i.e. the threshold t_0, is known for gapped shapes, and optimal threshold computation has been shown to be NP-complete (Nicolas and Rivals, 2005). Algorithms for the computation of the optimal threshold, and for the calculation of filtering sensitivities have been proposed (Herms and Rahmann, 2008), but assume alignment transcripts to be generated by a Markov process, i.e. are unaware of position within the alignment. This information, however, is of importance in read mapping, as sequencing quality and therefore alignment error probability increases towards the read end. Also, there is no simple way of computing "good" shapes that lead to a high filtration efficiency (Burkhardt

and Kärkkäinen, 2001) and this problem is NP-hard (Nicolas and Rivals, 2005). We will focus on q-gram counting as implemented in RazerS in section 3.4, more specifically on sensitivity and threshold computation in section 3.4.1 and on shape computation in section 3.4.2.

3.1.2 Index Data Structures

In order to quickly identify exact q-matches (or seed matches), all read mapping algorithms build an index either of the reads or the genome or both. A popular index structure is the suffix array (Manber and Myers, 1993) which stores all suffixes of the indexed sequence in lexicographically sorted order for fast access. However, due to its high space consumption for large sequences (around 48 GB for the human genome) it is not very common in read mapping. The most recent read mapping tools to date use a compressed version of the suffix array, the FM-index (Manzini and Ferragina, 2004) (around 3 GB for human) which uses the Burrows-Wheeler transform for compression (Burrows and Wheeler, 1994). Most early read mapping tools however use a simpler, more access-efficient data structure: a hash table or a q-gram index which stores all subsequences of length q in sorted order. In the following we describe the q-gram index in more detail, as it is heavily used and of most importance for this work.

Q-gram index

A q-gram index consists of two arrays: one that stores the occurrence positions of q-grams in sequence s in lexicographically sorted order (*occ*), and one that stores for each q-gram a pointer to the first position in *occ* corresponding to the respective q-gram (*dict*), see Figure 3.3.

Construction time is linear in $|s|$, in the case of gapped q-grams $O(q \cdot |s|)$. In a full q-gram index, *dict* has size $|\sum|^q$. Each possible q-gram p can be quickly accessed with its hash value $h = \sum_{i=1}^{q} |\Sigma|^{q-i+1} \cdot ord(p_i)$ where the function *ord* assigns a value $0, ..., |\Sigma| - 1$ to each character of the alphabet. For $q > 14$ and the DNA alphabet, this q-gram index becomes too large in practise, as it requires at least $|\sum|^q \cdot 4$ bytes for *dict*. In this case, a q-gram index with *open addressing* may be used: by using a non-trivial hash function it requires less space for array *dict*, but then requires more time for q-gram lookups.

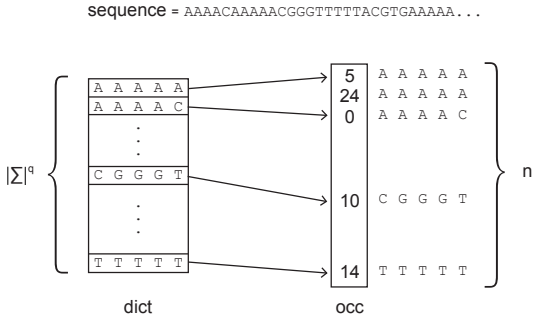

Figure 3.3: The q-gram index data structure. The dictionary *dict* provides fast lookup of q-grams in the sorted occurrence table *occ*.

3.1.3 Verification Techniques

Depending on the filter technique and index type used, a potential match is identified and verified through a backtracking procedure through the index (usually for suffix array or FM index based methods) or a potential match is returned as a potentially matching subsequence of the genome. In the latter case, a classical alignment method then verifies whether a valid read-to-reference alignment exists. We concentrate on this case, as it is more general and used in q-gram counting filters.

Let read r have a potential match in genome subsequence s. In the case of Hamming distance, it suffices to naively test each possible alignment starting position i and scan the alignment $a_r^{s_{i,i+|r|-1}}$ for mismatches going from left to right. Scanning stops once either the maximum distance k cannot be fulfilled anymore, as $k+1$ mismatches have been encountered already, or the end of the alignment has been reached, in which case a valid match is returned.

If gaps are allowed in the alignment (or in general if a scoring scheme other than Hamming distance is used), usually a dynamic programming approach is used. In the following we briefly provide the basics of dynamic programming.

Dynamic programming verification

Alignment of two sequences with dynamic programming (DP) methods has its origin in Needleman-Wunsch alignment (Needleman and Wunsch, 1970). Given two sequences r and s, a DP matrix F is computed that stores in each cell $F_{i,j}$ the score of the optimal alignment of prefixes r_i and s_j. A traceback trough the matrix gives the actual alignment. While the Needleman-Wunsch algorithm computes a global alignment, i.e. the one that ends in cell $F_{|r|,|s|}$, modifications such as the Smith-Waterman algorithm (Smith and Waterman, 1981) to compute local alignments have subsequently been proposed. Another variation is semiglobal alignment, where the optimal alignment of one sequence to a subsequence of the other sequence is computed, as is the case for alignment of a read to a potential match region. Given read r and genomic subsequence s, the DP matrix is computed with the following initialization and recursion:

1. Initialization

$$\begin{aligned} F_{i,0} &= d(r_i, '-') \cdot i \qquad && 0 \leq i \leq |r| \\ F_{0,j} &= 0 && 0 \leq j \leq |s| \end{aligned} \tag{3.1}$$

2. Recursion

$$F_{i,j} = max \begin{cases} F_{i-1,j-1} + d(r_i, s_j) \\ F_{i,j-1} + d(r_i, '-') \\ F_{i-1,j} + d('-', s_j) \end{cases} \tag{3.2}$$

where $d(x, y)$ returns a score for aligning character x with character y. If we set $d(x, y) = -1$ for $x \neq y$ and $d(x, y) = 0$ if $x = y$, the cells in F contain Levenshtein distance (with negative sign). Tracing back the alignment from the cell with the best score in the last row gives us the optimal semiglobal alignment. There may be ambiguity in the optimal alignment, as shown in Figure 3.4. Banded alignment methods reduce complexity by only computing values within a band of the

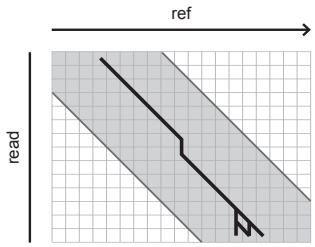

Figure 3.4: A semiglobal alignment within a band of the dynamic programming matrix. Branches of the alignment trace (black line) indicate ambiguity among ending positions.

alignment matrix (indicated by gray background) and thereby avoiding calculation of cells that cannot be part of the optimal alignment anyway.

3.2 Overview of Read Mapping Tools

Table 3.1 gives an overview of some popular read mapping tools with some of their key algorithmic properties. Many more tools have been developed (see (Fonseca *et al.*, 2012) for a recent comprehensive list), but we focus here on the ones that are among the most popular and relevant for this work.

While earlier tools were mostly based on read indices, later tools prefer the space efficient compressed FM index of the genome. **Eland** is Illumina's commercial software and was therefore one of the first read mapping tools geared toward short Illumina reads. It uses the two-seed pigeonhole strategy which was also adopted by **Maq** (Li *et al.*, 2008a), **Soap** (Li *et al.*, 2008b) and **SeqMap** (Jiang and Wong, 2008). These tools are fully sensitive for up to two errors under Hamming distance only. **Zoom** (Lin *et al.*, 2008) is also commercial; it uses multiple gapped seeds and guarantees full sensitivity for Hamming distance up to five errors, but is heuristic when gaps are allowed. **Shrimp** (Rumble *et al.*, 2009) and **RazerS** (Weese *et al.*, 2009) are the only q-gram counting based tools. Shrimp differs a bit from the other read mapping tools, as it actually performs local Smith-Waterman alignment in its verification step. Therefore it can detect local (partial) as well as semiglobal alignments, making it applicable to a wider range of problems. While Shrimp uses q-gram counting, its newer version Shrimp2 uses multiple gapped seeds. However, both versions do not provide sensitivity control, and the default settings will be lossy in many cases. The newer read mapping tools **Bowtie** (Langmead *et al.*, 2009), **BWA** (Li and Durbin, 2009), and **Soap2** (Li *et al.*, 2009b) are all based on the space-efficient FM index, and are usually very fast for reporting a best match. However, for full sensitivity they are usually restricted to low numbers of errors and use heuristics when gaps are allowed.

RazerS is the only tool that provides fine-grained sensitivity control for Hamming as well as Levenshtein distance and arbitrary read length as well as error rates up to 10%. Especially the 100 % sensitivity switch that it provides for more than 2 errors was unique at the time of development and publication. In the following, we explain the RazerS algorithm in more detail.

year of publication	**Eland** unpubl.	**Maq** 2008	**Soap** 2008	**SeqMap** 2009	**Zoom** 2009	**Shrimp** 2009	**RazerS** 2009	**Bowtie** 2009	**BWA** 2010	**Soap2** 2010	**Shrimp2** 2010
filtering technique	two-seed pigeonhole	two-seed pigeonhole	two-seed pigeonhole	two-seed pigeonhole	multiple gapped seeds	q-gram counting	q-gram counting	FM index backtracking	FM index backtracking	pigeonhole seeding (using FM index)	multiple gapped seeds
distance measure in filtering step	Hamming	Hamming	Hamming	Hamming and Levenshtein	Hamming	Hamming and Levenshtein	Hamming and Levenshtein	-	-	Hamming	both Hamming and Levenshtein possible
distance measure in mapping step	Hamming	Hamming (Smith-W. for second mate)	Hamming or one up to 3 bp gap	Hamming or edit with at most 5 errors	Hamming or edit with at most one gap	Smith-W.	either edit or Hamming	Hamming	Hamming, Levenshtein, Smith-W.	Hamming	Smith-W.
indexing strategy	q-gram index on reads	q-gram index on reads	q-gram index on reference	q-gram index on reads	q-gram index on reads	q-gram index reads	q-gram index on reads	FM index on reference	FM index on reference	FM index on reference	multiple q-gram index on reference
supported read length	≤ 32	≤ 127	≤ 60	arbitrary	≤ 63 (currently ≤ 240)	arbitrary	arbitrary	≤ 1024	arbitrary	≤ 1024	arbitrary
sensitivity	full sensitivity only for up to 2 mismatches	full sensitivity for up to 3 mismatches in 28 bp prefix	full sensitivity for up to 2 mismatches	full sensitivity	switch to guarantee full sensitivity	no help for parameter choice, default will be lossy for many settings	arbitrarily adjustable	switch for full sensitivity for best matches	switch for full sensitivity for best matches	full sensitivity only for up to 2 mismatches	no help for parameter choice, default will be lossy for most settings
can output all (suboptimal) hits	no	no	no	yes	yes	yes	yes	no	no	no	yes

Table 3.1: Short read mapping tools with their characteristics. Extends Table 1 from (Weese *et al.*, 2009).

3.3 RazerS - Overview of Read Mapping Algorithm

In RazerS, the filtering and verification algorithm is preceded by a parameter choosing step as visualized in Figure 3.5. Developed in collaboration with David Weese, own contributions are predominantly in this additional step.

RazerS makes use of a q-gram counting filter for identifying potential read matches. In its

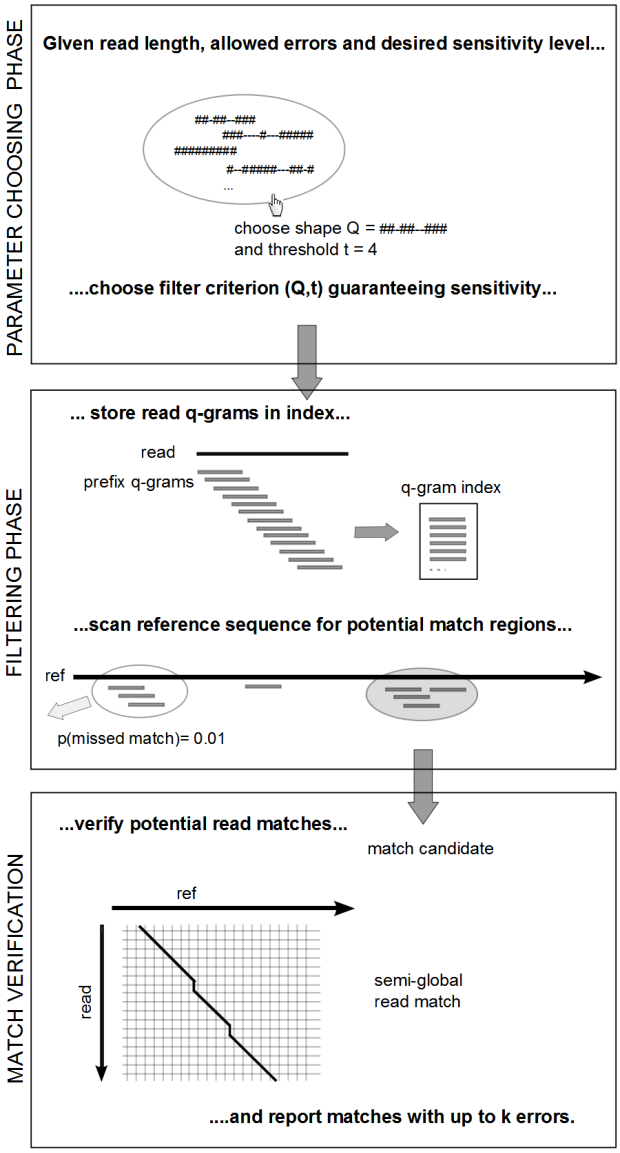

Figure 3.5: RazerS consists of three steps: a parameter choosing step followed by the two typical read mapping steps filtering and verification. During parameter choosing a shape Q and threshold t that guarantee the desired mapping sensitivity are chosen. Next, in the filtering phase, a q-gram index is built and the reference sequence is scanned for potential matches using the Swift algorithm. Potential matches are then verified by an alignment method.

first step, the parameter choosing phase, a q-gram shape and threshold guaranteeing a certain filtering sensitivity are chosen (explained in detail in section 3.4.3). Then, we build a q-gram index containing all read q-grams. Optionally, q-grams occurring more than a user-defined number of times can be masked to avoid investing computation time on counting highly repetitive/unspecific q-grams. Next, the reference sequence is scanned using the SWIFT filter algorithm (Rasmussen *et al.*, 2005), counting q-grams in parallelograms (details in section 3.3.1). Potentially matching parallelograms are passed on to a verification algorithm (details in section 3.3.2). If multiple matches for one read exist, then up to a user-defined maximum number m are recorded. Once m exact (errorless) matches are found for a read, the read is disabled, such that no more time-consuming verifications are triggered.

The user may set different parameters including allowed error rate, type of distance measure (Hamming/Levenshtein), sensitivity level, and practical options such as type of output file (e.g. SAM (Li *et al.*, 2009a) or general feature format (GFF)).

The SWIFT filtering algorithm and subsequent verification are described in the following two subsections. The parameter choosing step is described in more detail in the next section.

3.3.1 Filtering algorithm

RazerS implements the SWIFT q-gram filtering algorithm (Rasmussen *et al.*, 2005). Given a (q,t)-filtering criterion, it returns parallelograms (i.e. bands) in the read-to-reference DP matrix, that contain at least t q-matches. It is based on the observation that any match with at most k errors will lie within a band of at most $k+1$ diagonals in the DP matrix. One could now check each parallelogram with width $k+1$ separately. Instead, the SWIFT filter sets parallelogram width to w with $w > k+1$ and lets parallelograms overlap in k diagonals. This way, each $k+1$ parallelogram is fully contained in one w parallelogram (see Figure 3.6).

Each w parallelogram stores a counter for contained q-matches. Now we slide over the genome sequence, look up the current q-gram in the read q-gram index and increase the counter(s) of the corresponding parallelogram(s). For each read, only a low number of parallelograms is active at a time and therefore only few counters need to be kept in memory. Whenever a parallelogram becomes inactive (because we have scanned past it), we check whether the counter is larger or equal to the threshold and, if so, pass the parallelogram on to verification.

For Hamming distance one can use parallelograms of width 1, as any valid read match will always lie within a single diagonal (as no indels are allowed that could lead away from this diagonal).

3.3.2 Match Verification

For Hamming distance, the naive algorithm (diagonal scanning, see section 3.1.3) is implemented in RazerS. For Levenshtein distance, RazerS uses the bit-vector algorithm by (Myers, 1999) which computes the edit DP matrix exploiting bit parallelism. 64 cells of the matrix can be computed at once by a 64-bit CPU by maintaining a number of bit vectors. If a valid match of read r exists, the bit vector algorithm identifies its ending position j in the genomic subsequence s. In order to

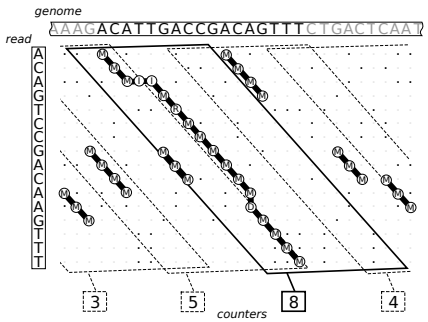

Figure 3.6: SWIFT filtering in overlapping parallelograms. Each parallelogram keeps a counter of matching 3-grams. The parallelogram with counter 8 holds a valid read alignment with Levenshtein distance k=4 which contains seven 3-matches. An additional random match contributes to the corresponding parallelogram counter. Figure from (Weese *et al.*, 2009).

identify the starting position, sequence $s_{1,j}$ and r are then reversed and the bit vector algorithm is repeated.

3.4 RazerS - Filter Parameter Computation

We now describe how to automatically choose efficient q-gram filtration parameters for RazerS. The goal is to choose a shape and a threshold that optimize the running time, while achieving a certain filter sensitivity. In other words, we want to allow the filter to be lossy in order to speed up the filtration process, but we want to be able to control *how* lossy our filter may be. More specifically, our goal here is to minimize running time of RazerS while guaranteeing the desired read mapping sensitivity given position-dependent sequencing error probabilities as are typically seen for Illumina reads (see section 2.2.1).

The two main computational issues here are 1) finding a set of "good" shapes which are considered as candidate shapes and 2) computing their sensitivity in combination with different thresholds, given a read length n, number of errors k (Hamming or Levenshtein) and position-dependent error probabilities. First, we introduce some new notation. Then we will give a DP formulation for the problem of computing filter sensitivities for a given shape (section 3.4.1). Next a heuristic approach for computing good shapes (with high weight) for a low number of errors is explained (section 3.4.2). Finally, we devise a simple strategy for choosing efficient filter parameters implemented in the SeqAn tool *ParamChooser* which is integrated into RazerS.

Notation

A position-dependent error profile is a vector $p^{\mathtt{X}} = \{p_1^{\mathtt{X}}, p_2^{\mathtt{X}}, ..., p_n^{\mathtt{X}}\}$ where $p_i^{\mathtt{X}}$ describes the probability of error type (edit operation) $\mathtt{X} \in \{\mathtt{R},\mathtt{I},\mathtt{D}\}$ at position i in a read of length n. The probability of a match is set to $p_i^{\mathtt{M}} = 1 - p_i^{\mathtt{R}} - p_i^{\mathtt{I}} - p_i^{\mathtt{D}}$.

We can then calculate the occurrence probability of a transcript T as $p(T) = \prod_{i=1}^{|T|} p_{r(T,i)}^{T_i}$ where $r(T,i)$ returns the read position corresponding to transcript position i. For insertions, we define $r(T,i)$ as the read position before the insertion.

The *loss rate* of a filter for matches with exactly k errors is defined as the probability of a transcript with k errors to be missed by the filter. Conversely, the sensitivity of a filter for matches with k errors is defined as the probability of a transcript with k errors to be identified by the filter as potential match.

3.4.1 Computing Sensitivities by Dynamic Programming

We include here the formulation of the dynamic programming approach for Hamming distance and gapped shapes as it is explained and proven correct in (Weese *et al.*, 2009). An extension to Levenshtein distance is formulated in (Weese *et al.*, 2009). The algorithm was originally developed and implemented by David Weese. Own contributions were in extending the implementation to gapped q-grams and to position-dependent error probabilities.

Given a gapped shape Q with weight q and span $s(Q)$. For a (Q,t)-filter, we are interested in the probability that a transcript T with k errors contains at least t Q-matches: $P(T \text{ contains} \geq t\, Q\text{-matches} \,\big|\, ||T|| = k)$. Note that this probability is only a lower bound for the sensitivity of a (Q,t) filter that counts in overlapping windows/parallelograms, as *cross matches*, randomly matching Q-grams that are not part of the optimal alignment of two sequences (see Figure 3.6), may contribute to the number of shared Q-grams.

We denote the sum of all occurrence probabilities of transcripts of length n with k errors containing at least t Q-matches by $S(n,k,t)$. We can then write the total sum of all occurrence probabilities of transcripts of length n with k errors as $S(n,k,0)$. Then

$$P(T \text{ contains} \geq t\, Q\text{-matches} \,\big|\, ||T|| = k) = \frac{S(n,k,t)}{S(n,k,0)} \tag{3.3}$$

By enumerating all transcripts with k errors, calculating their occurrence probabilities and counting the number of Q-matches, we could naively compute $S(n,k,t)$. However, this quickly becomes infeasible with increasing n and k, as the number of different Hamming transcripts is $\binom{n}{k}$. Instead, we use a dynamic programming approach similar to the one in (Burkhardt and Kärkkäinen, 2001).

We denote the occurrence probability of sub-transcript T starting at position j of a read with $p(T,j) = \prod_{i=1,\ldots,|T|} p_{i+j}^{T[i]}$. Let $R(i,e,t,T_2)$ be the sum of occurrence probabilities of all transcripts T_1, s.t. $|T_1| = i$ and $||T_1|| = e$ and the concatenation T_1T_2 contains at least t Q-matches. Then, it suffices to enumerate all transcripts of length $s(Q)$, i.e. $\sum_{e=0}^{k} \binom{s(Q)}{e}$ many Hamming transcripts and we can calculate

$$S(n,e,t) = \sum_{|T|=s(Q),\, \|T\| \leq e} R\left(n - s(Q), e - \|T\|, t, T\right) p(T, n - s(Q)) \tag{3.4}$$

where $p(T, n - s(Q))$ is the occurrence probability of transcript T at the end of a transcript of length n. $R\left(n - s(Q), e - \|T\|, t, T\right)$ is the sum of occurrence probabilities of all possible transcript beginnings with $e - ||T||$ errors such that the concatenation of such a beginning with T results in a transcript of length n with exactly e errors and at least t Q-matches. Summing up over all

transcript endings T we get the occurrence probability sum of all such transcripts of length n. We can calculate R recursively, for $e = 0, ..., k$, $i = 1, ..., n$, $t = 0, ..., t_{max}$ and for all $T \in \{\mathtt{M},\mathtt{R}\}^{s(Q)}$

$$R(1, e, t, T) = \begin{cases} 1 \text{ if } e = 0,\ t \leq \delta(T) \\ 0 \text{ else} \end{cases} \tag{3.5}$$

$$\begin{aligned} R(i, e, t, T) = & \qquad (3.6) \\ & p_i^{\mathtt{M}} \cdot R(i-1\ , e\ \ \ \ , t - \delta(T), shift(\mathtt{M}, T)) \\ + \quad & p_i^{\mathtt{R}} \cdot R(i-1\ , e-1, t - \delta(T), shift(\mathtt{R}, T)) \end{aligned}$$

with

$$shift(x, T) = xT_{2,|T|}\,, \quad \text{and} \quad \delta(T) = \begin{cases} 1 \text{ if } T \text{ contains a } Q\text{-match} \\ 0 \text{ else} \end{cases} .$$

Remarks

Sequencing errors in Illumina reads tend to accumulate towards the 3' ends. From a filtering efficiency standpoint, this makes lossy filtering very profitable, even more than with a uniform error profile. When taking non-uniform probabilities into account, the threshold can be set to a higher value (thereby improving filter selectivity) while maintaining the same sensitivity level as with a uniform profile. This is demonstrated with an example in Figure 3.7 where the sensitivity of an ungapped 11-gram for read length $n = 36$ and Hamming distance $k = 2$ is plotted (typical values for early Illumina reads). Sensitivities were computed using the DP algorithm above. A (11,6)-filter achieves sensitivity 95% with uniform error probabilities. In practise though, with an Illumina error profile, one can use a (11,8)-filter and achieve the same sensitivity, thus increasing the threshold by 2.

If one uses a gapped 11-gram, the effect is also seen but a bit less pronounced. In particular, sensitivity drops more rapidly (from 100% with $t = 5$ to below 95% with $t = 6$). The tradeoff

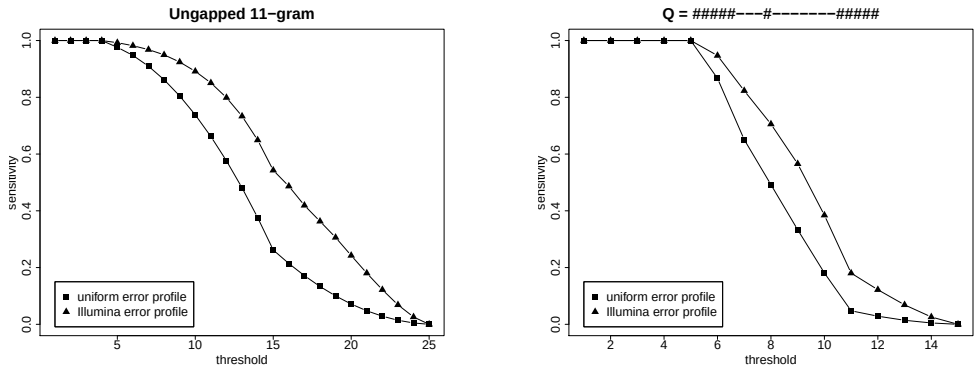

Figure 3.7: Example of sensitivity as a function of threshold, given read length $n = 36$ with $k = 2$ errors (Hamming), with and without position-dependent error probabilities, for a) the ungapped 11-gram and b) a gapped shape with weight 11.

between sensitivity and selectivity (increasing threshold) is less smooth, due to the higher span of the q-gram and therefore lower total number of q-grams that can be placed in a sequence of fixed length.

However, for 100% sensitivity we can see in this example, that the optimal lossless threshold $t_0 = 4$ for the ungapped 11-gram is smaller than $t_0 = 5$ for a gapped shape with the same weight. Gapped shapes are thus especially valuable when 100% sensitivity is required (and only mismatches are allowed).

3.4.2 Computing Heavy Lossless Shapes

An important feature of RazerS is that it can guarantee 100% mapping sensitivity if desired. For this reason, and as lossless filtering is the hardest case, we are particularly interested in having efficient filter settings for the 100% sensitivity case. For Levenshtein distance, ungapped q-grams will be the most sensitive choice, as joker positions do not tolerate indels. For Hamming distance, however, good gapped shapes are of importance (as seen in Figure 3.7).

We will later see that the weight of the shape, i.e. parameter q, turned out to have the greatest impact on running time of RazerS. Therefore, we derived a greedy approach to computationally determine heavy lossless shapes with $t_0 > 0$, given sequence length n and up to k mismatches. We show that in many cases the method returns a shape of maximum weight, in some cases even the optimal shape as determined in (Burkhardt and Kärkkäinen, 2001). As complexity is exponential in $k + 1$, we only use the method for $n \leq 50$ and $k \leq 3$.

Given a shape Q, we can naively compute the optimal threshold with the following calculations:

$$t = n - s(Q) + 1 - max(M) \tag{3.7}$$

$$\text{with } M = \sum_{i=1}^{n-s(Q)+1} I_i \tag{3.8}$$

$$\text{and } I_i(j, u, ..., v) = \begin{cases} 1 & \text{if } \exists w \in \{j, u, ..., v\} : w - i \in Q \\ 0, & \text{otherwise} \end{cases} \tag{3.9}$$

where $n - s(Q) + 1$ gives the total number of Q-grams in a sequence of length n. M is a k-dimensional array recording how many q-grams are destroyed when mismatches cooccur at the indexed positions. For example, if $k = 2$ and $M(u, v) = 3$ then 3 q-grams are destroyed if the two errors cooccur at positions u and v. Thus, we actually only need to store a part of the matrix (the upper triangle in the two-dimensional case or $\binom{n}{k}$ many entries in general). The maximum entry in M describes the worst case arrangement of k errors, thus gives the maximum number of q-matches that can be destroyed by k errors. Matrices I_i are binary indicator matrices and store the information whether a q-gram starting at position i in the alignment transcript is destroyed if mismatches cooccur at the indexed positions. As each possible combination of k mismatches is inspected, formulae 3.7 to 3.9 describe an exact method for computing t_0.

With the help of M and I_i, we can greedily compute good shapes by iteratively adding positions to a candidate shape Q with fixed span and keeping arrays M and I_i up to date in each iteration. We test positions from left to right: The leftmost position is inserted if the maximum of the updated array M remains low enough such that a minimum required threshold $minT$ is maintained using equation 3.7. Otherwise, the position will be a joker position, i.e, not inserted into Q, and we move on to the next position. The algorithm is given in pseudocode in the appendix. We run the algorithm on different spans and fix the first and the last position of Q. The output is the shape that has 1) highest weight and 2) largest span. We test spans from $\lceil n/(k+1) \rceil + 1$ to $n - k$.

With increasing number of errors, the complexity of this method quickly becomes prohibitive. We thus computed shapes with this method for read lengths up to 50 bp and up to 3 errors only. In special applications of read mapping, such as the partial read mapping tools introduced in the next chapter, the filtration criterion will be based on a seed-part of the read, i.e. independent of the entire read length. This seed length is usually relatively small (e.g. between 12 and 30 bp) and supposed to be of high-quality, such that values of $k > 3$ are uncommon, making our method usable and valuable in practise.

Observations

The method has a tendency to produce periodic shapes, which happen to be quite frequent among optimally weighted shapes (Kucherov *et al.*, 2005). As a basic evaluation, we tested whether some maximum weight shapes given in the literature could be retrieved by our method (implemented in R).

- Given in (Kucherov *et al.*, 2005): `###-#--###-#--###-#` is the optimal weight shape for $n = 25, k = 2$. Our method returns the same shape and requires < 1 second.
- Given in (Burkhardt and Kärkkäinen, 2001): `#####-##---#####-##` is the optimal weight shape for $n = 50, k = 4$. Our method returns the same shape and requires 5.5min and 4.4 GB of memory.

After these positive results, we additionally checked for all $n = 10, ..., 30$ and $k = 1, ..., 3$ whether there existed a shape with weight $w(Q_{greedy}) + 1$, where Q_{greedy} is our greedily computed shape. As we exhaustively enumerate all such $(w(Q_{greedy})+1)$-shapes, the test becomes prohibitive for larger values of n. However, for all cases tested, we found no shape with higher weight than our greedily computed shape. Still, we do know that the method is not optimal, as in some cases it provably fails to detect the heaviest shape for a fixed span. For example, for $n = 20$ and $k = 2$ there exists an 8-shape `###-#--#-#--##`, but instead it returns the 7-shape `###-#--##----#` for span 14.

3.4.3 ParamChooser

We make use of the methods for filter sensitivity and lossless shape computation (described in the previous two sections) in our tool *ParamChooser*. Integrated into RazerS but also usable as a

stand-alone tool, ParamChooser supports two basic tasks:

1. computing loss rates for a given error profile and a given set of shapes, saving results to parameter files

2. choosing a "best" filtering criterion (Q, t) for a given read length n, k errors, and a maximum allowed loss rate, by parsing precomputed parameter files

In this context, "best" means those settings that optimize the running time of RazerS while complying to the loss rate. The parameter choosing module in RazerS makes use of the second step only, i.e. parses precomputed parameter files and chooses the best filtering criterion. The two steps are explained in the following two subsections.

Precomputing Parameter Files for RazerS

We compiled a set of candidate shapes from the literature (Burkhardt and Kärkkäinen, 2001; Li *et al.*, 2003) and by using the greedy algorithm explained in the previous section. These shapes are hard-coded into the ParamChooser source code. Optionally, the user can specify additional shapes through the ParamChooser command line.

Using the DP algorithm from section 3.4.1, ParamChooser then computes loss rates for the user-specified read length n allowing an error rate of up to 10% using either Hamming or Levenshtein distance. As a default error profile, uniform error probabilities are assumed. The results are then recorded in parameter files that contain human-readable filtering criterion information (example shown in Figure 3.8). Each entry (row) in these files stores a 5-tuple (e, Q, t, L, P) where e is the number of errors, Q and t are shape and threshold, L is the computed filtering loss rate, and P is the number of potential matches as observed on a simulation run[2] of RazerS with this filter setting.

For precomputing parameter files for RazerS, we used a typical Illumina error profile (Dohm *et al.*, 2008) for 32 bp reads and interpolated for other read lengths[3]. In the case of Levenshtein distance, insertion/deletion errors received a 0.002 probability, independent of read base position. Parameter files were produced for all $n = 16, ..., 75$, once using Hamming and once using Levenshtein distance.

Choosing Filter Parameters

Given n, k and a user specified loss rate L_{user}, ParamChooser parses the corresponding parameter file and consider among all (e, Q, t, L, P) tuples only those for which $e = k$ and $L \leq L_{user}$. Then the one that is optimal with respect to the following order is chosen: 1) $w(Q)$ is maximized 2) P is minimized 3) t is maximized 4) $s(Q)$ is maximized. Only if $t = 1$, we check whether the shape with second best weight, i.e. w(Q)-1, has threshold > 2. This selection process is based on the

[2] simulating 50000 reads from a simulated 1 Mb genome and mapping them back with RazerS

[3] Standard read length at the time was 32 bp. Interpolation rather than extrapolation was chosen as advances in sequencing technology producing longer reads would be accompanied by improved quality, i.e. stretching the error profile.

results_N65_L.dat

```
errors shape      t    lossrate      PM

0  1111111111   55   0           18738
1  1111111111   46   0           41703
2  1111111111   37   0.1399      52567
2  1111111111   36   0           53846
...
```

Figure 3.8: Example of a parameter file for read length 65 bp (N65 in filename) and edit distance (L for Levenshtein distance). The first column states the number of errors, the second column the shape, column three the threshold and column four the computed loss rate. Column five is used to get an estimate of the filtration efficiency and records the number of potential matches as observed on a simulation run of RazerS with the filter settings of this row.

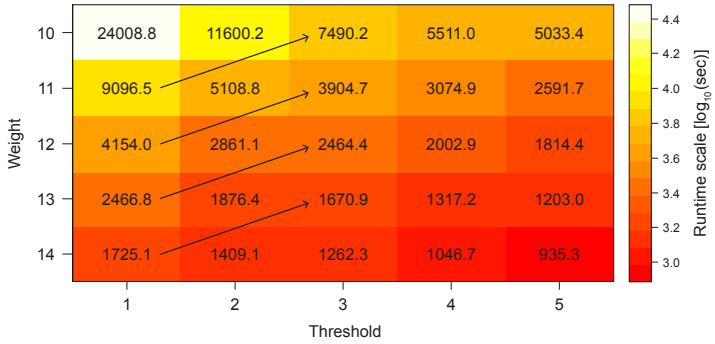

Figure 3.9: Running time of RazerS in seconds using different filtering parameters, measured on a set of 1M 50 bp reads on human chromosome X. The weight of the q-gram has the greatest influence on running time. When t=1 it pays off to use a q-gram with weight smaller by one if in that case $t \geq 3$ (indicated by arrows).

observation that the weight has the greatest influence on RazerS' running time for read lengths and error rates typical for Illumina reads at that time. Figure 3.9 clearly shows how the weight of the q-gram has the largest influence on running time. (Note that $q \leq 14$ as RazerS uses a full q-gram index which would consume too much memory for larger values of q.) Even if $t = 1$, it does not pay off to use a lower weight q-gram unless it can be used with $t > 2$ (indicated by the arrows in Figure 3.9). For thresholds higher than one, we always use the highest weight q-gram possible.

For read lengths greater than 75, we extrapolate from shorter read lengths. To this end, we choose a setting (n_e, k_e) with n_e as large as possible, such that $k_e/n_e \geq k/n$. This ensures that the loss rate for (n_e, k_e) is a lower bound for the loss rate for (n, k). We can then set the threshold value $t = \lfloor t_e \cdot n/n_e \rfloor$. This is illustrated by an example: If we extrapolate the filtering setting for (100,6) from (50,3), we can imagine concatenating two reads of length 50 bp. Therefore we can also expect to see at least twice as many q-grams. The extrapolated value for t is not necessarily optimal, as q-grams crossing the concatenation point will be not considered. However, as it constitutes a lower bound, we can still guarantee a lower bound for sensitivity.

3.5 RazerS - Results

We evaluated RazerS in terms of correctness of its mapping sensitivity control and its computational performance in comparison with five other read mapping tools popular at the time of publication: Shrimp, Maq, Soap, Zoom and SeqMap (see Table 3.1). This evaluation section is heavily based on (Weese *et al.*, 2009).

Evaluation Data

We used two real read data sets for the evaluation:

- SRR001815 from short read archive: Drosophila melanogaster whole genome resequencing,10,760,364 reads of length 36 bp, Illumina Genome Analyzer.
- SRR006387 from short read archive: Whole genome resequencing of human HapMap individual NA12005, $2 \times 7{,}894{,}743$ 76 bp paired-end reads, trimmed to the first 63 bp (in order to be able to include all tools in the comparison), Illumina Genome Analyzer.

In addition, we simulated two read data sets for the evaluation of RazerS' sensitivity estimation:

- A set of 1,000,000 36 bp reads simulated from the human X chromosome. Replacement errors (mismatches) were introduced according to the Illumina error profile (Dohm *et al.*, 2008) that we used earlier for filtering parameter computation.
- A set of 1,000,000 36 bp reads simulated from the human X chromosome. In addition to replacement errors, insertion/deletion errors were introduced with a probability of 0.002 at each read position.

As reference sequences, we use the human reference genome hg18 (NCBI build 36.3) and the drosophila melanogaster reference genome (FlyBase Release 5.9), both repeat masked (Smit *et al.*, 2004).

3.5.1 Evaluation of RazerS' Mapping Sensitivity Estimation

In order to validate our sensitivity estimation, we compared calculated and observed mapping sensitivities. We use simulated and real 36 bp read data and we test Hamming as well as Levenshtein distance. The evaluation strategy was developed in collaboration with David Weese, evaluation was scripted and run by the author, plots were generated by David Weese.

Evaluation Setup

To determine the observed, i.e. empirical, loss rate for a given number of errors k, we first categorize reads according to their true error level. On the simulated sets, we know the number of errors introduced during simulation and can therefore use the simulated error level for categorizing; on the real data, we first run RazerS with lossless filtration parameters (100% sensitivity) and then categorize according to the number of errors in their best alignment. We then define the empirical

loss rate at error level k as the number of unmapped reads with k errors divided by the total number of reads with k errors. We run this test for different sensitivity settings, i.e. different filter settings, and inspect the difference of empirical loss rate to calculated loss rate, computed by the DP algorithm with error probabilities as used during simulation in the artificial read set; for the real data set, we use the *a posteriori* error profile as observed after 100% sensitivity mapping.

Results

Results are plotted for four test cases (real/simulated 36 bp data, each with Hamming/Levenshtein distance) in Figure 3.10 (results and figures from (Weese *et al.*, 2009)). The dashed lines shows the mean relative difference between empirical and calculated loss rates ($1 - \frac{\text{empirical loss rate}}{\text{calculated loss rate}}$) as a means to measure sensitivity estimation accuracy.

For Hamming distance we see a very good correlation between empirical and calculated loss rates, on simulated as well as real data sets. We are particularly interested in low loss rates ($< 10\%$) as these will be of most practical relevance. For Hamming distance, the mean relative difference remains low (below 0.1%) for low loss rates $< 10\%$ and remains low ($< 1\%$) throughout the whole range. For Levenshtein distance on the other hand, we observe a more pronounced disagreement, which however still stays below 10% in all cases. The higher disagreement compared to Hamming distance is mainly explained by cross matches that contribute to the q-gram count, and thereby lead to more parallelogram counters reaching the threshold. In Hamming distance, parallelograms are single diagonals and thus no cross matches occur. At loss rates $< 10\%$ the mean relative difference is $< 4\%$ for simulated data and $< 2.8\%$ for real data. For simulated data, there is the additional aspect that reads may be mapped with less errors than they were initially simulated with (for example, if an insertion was simulated next to a deletion).

In total, we observe that the empirical loss rate correlates well with the calculated one and most importantly never exceeds the calculated one, i.e. serves as a good lower bound.

3.5.2 Comparison with Other Read Mappers

We evaluate RazerS' performance in terms of running time, space consumption, and number of mapped reads in comparison to five other read mapping tools: Zoom (Lin *et al.*, 2008), Shrimp (Rumble *et al.*, 2009), SeqMap (Jiang and Wong, 2008), Soap (Li *et al.*, 2008b), and Maq (Li *et al.*, 2008a). Again, we evaluate different settings, testing Hamming as well as Levenshtein distance and using the two real read data sets introduced earlier in this section. For additional comparisons on longer simulated reads and real paired-end data, see (Weese *et al.*, 2009).

Evaluation Setup

Three test settings were used: Hamming distance 2 for the 36 bp reads from Drosophila (setting A), Levenshtein distance 2 on the same data set (setting B), and Hamming distance 5 on the human 63 bp reads (setting C). We run our test on a AMD Opteron 2.8 GHz machine with 64 GB of RAM.

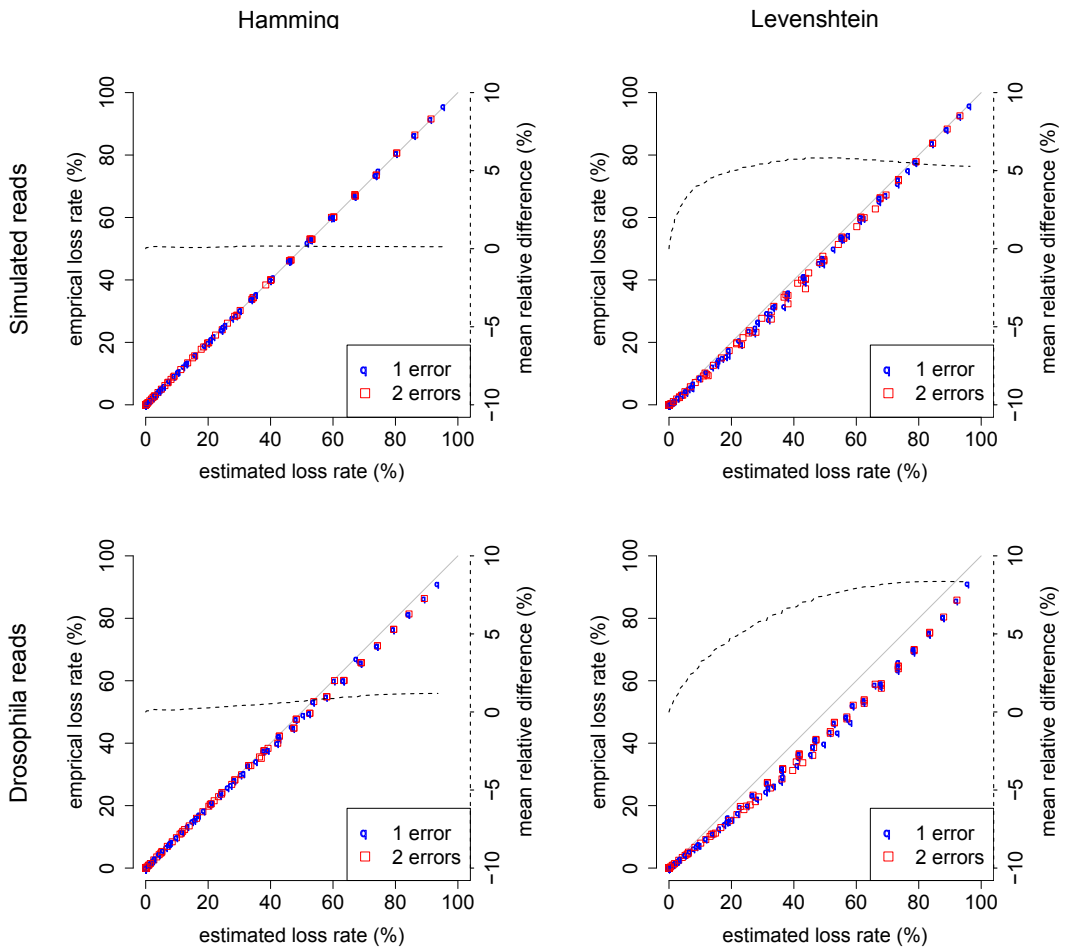

Figure 3.10: Comparison of empirical and calculated loss rates for varying parameter settings $q = 8, \ldots, 14$ and $t = 1, \ldots, 20$. The left column shows Hamming distance, the right column Levenshtein distance results. The first row is on simulated, the second row on real Drosophila reads. The dashed line reflects the mean of relative differences $1 - \frac{\text{empirical loss rate}}{\text{calculated loss rate}}$ of all calculated loss rates below the loss rate level given on the X-axis. Figure from (Weese *et al.*, 2009).

We set all tools to report only one best match. We carefully set each tools' mapping parameter such as to make the comparison as fair as possible. However, as all tools have slightly different objectives this is not in all cases straightforward. For example, Shrimp uses Smith-Waterman alignment for verification, which allows to compute Hamming distance alignments (by setting gap penalties very high and using a score cutoff that will allow only valid semiglobal matches to be reported), but it does not allow to exactly compute Levenshtein alignments. Still, we try to emulate Levenshtein distance by setting gap penalties equal to mismatch penalties and requiring a score cutoff that will guarantee not to miss valid Levenshtein alignments. Also Maq requires special treatment, as it minimizes the sum of mismatching quality values in its verification, and only the number of mismatches in the 28 bp prefix can be set. For setting A, we set the number of mismatches in the prefix to 2, for setting C to 3 (the maximum possible). We then post-filter

matches with a higher total number of errors > 2 for setting A and > 5 for setting C. Both Maq and Soap do not support gapped alignment, i.e. are only used for the Hamming distance test cases. Even though the Soap manual states 60 bp as the maximum read length, we experienced no problems running it on 63 bp reads (without trimming taking place). Zoom has a switch to guarantee 100% sensitivity for Hamming distance which works for $k = 2$ but gives an error message for $k = 5$. However, requiring 100% sensitivity for matches with up to 4 errors works, so we use this for setting C instead. We run RazerS in two sensitivity settings: 100% and 99% sensitivity. Details on program calls are given in the appendix.

Results

The results of running all tools on the three test settings are summarized in Table 3.2. By sacrificing 1% in sensitivity (RazerS100 to RazerS99), we gain a significant speed up of at least factor 2. In setting C (long human reads) speed is even improved by a factor of 13. For setting A (short Drosphila reads) RazerS100 is already fast, with only Zoom being faster. Except for mapping 1M reads in setting A, where Zoom is still slightly faster, RazerS99 is the fastest tool. The other q-gram counting based tool, Shrimp, is always slowest. In general, allowing mismatches only is always faster for all tools, with the exception of Shrimp which anyway computes Smith-Waterman alignments. The running time improvement in RazerS when going from Levenshtein to Hamming distance (in the most extreme case from 163 h in setting B to 10.6 h in setting A) is mainly explained by the use of gapped q-grams which immensely speed up the filtering phase.

In terms of memory consumption, RazerS is comparable to other tools. Setting C with the entire $\sim 10\,M$ reads requires $> 4\,GB$ of memory for all tools, except for Zoom. SeqMap is most

				RazerS100	RazerS99	Zoom	Shrimp	SeqMap	Soap	Maq
(A) Dm 36 bp	Hamming	1M	time (min)	2.13	1.63	**1.47**	15.3	6.70	9.27	4.10
			space (GB)	1.31	1.30	0.72	0.68	6.56	0.67	0.60
			#mapped	505,506	503,595	505,506	505,084	505,059	506,476	503,999$^{(605K)}$
		all	time (min)	10.6	**5.55**	7.80	145	12.8	116	9.68
			space (GB)	4.10	3.92	3.77	5.80	11.1	0.67	5.36
			#mapped	5,353,287	5,335,554	5,353,287	5,349,007	5,348,776	5,414,337	5,338,676$^{(6.5M)}$
(B) Dm 36 bp	Levenshtein	1M	time (min)	12.3	**5.92**	32.7	13.6	15.5	-	-
			space (GB)	0.48	0.53	0.72	0.68	8.38	-	-
			#mapped	512,477	511,695	512,139	515,080	512,477	-	-
		all	time (min)	163	**68.45**	267	146	abort	-	-
			space (GB)	4.58	4.59	3.77	5.90	-	-	-
			#mapped	5,431,142	5,424,088	5,427,589	5,486,467	-	-	-
(C) Hs 63 bp	Hamming	1M	time (h)	3.14	**0.40**	26.1	10.7	48.8	3.88	2.43
			space (GB)	1.14	1.86	1.27	6.10	8.10	6.20	0.70
			#mapped	352,725	351,767	352,617	352,742	349,721	354,020	323,893$^{(362K)}$
		all	time (h)	25.4	**1.95**	45.3	> 3 d	abort	33.8	5.74
			space (GB)	5.60	6.13	2.89	-	-	6.2	4.38
			#mapped	3,102,320	3,095,435	3,091,063	-	-	3,133,920	2,817,561$^{(3.0M)}$

Table 3.2: Results for mapping 1 M and ~ 10 M (all) reads of length 36 bp onto the Drosophila genome (Dm) allowing for up to 2 Hamming (A) or Levenshtein (B) errors, and results for mapping 1 M and ~ 8 M reads of length 63 bp onto the human genome allowing for up to 5 Hamming errors (C). Soap and Maq do not support Levenshtein distance. Maq also reports matches with more errors , the total count of mapped reads including reads with more errors than allowed is shown in brackets. Table from (Weese *et al.*, 2009).

space-consuming and aborted when run on the entire read sets except in setting A.

All tools report similar numbers of mapped reads. RazerS99 always recognizes between 99.62 and 99.87% of the matches found by RazerS100. As the sensitivity level applies to the highest number of allowed errors (i.e. RazerS99 has 99% sensitivity for 5-error matches in setting C but is most likely lossless for < 5 errors), the overall sensitivity is higher than 99%. Soap maps the highest number of reads due to its differential treatment of 'N's. It counts a match if 'N' is aligned with 'A', a behavior that could not be switched off. Also Shrimp treats 'N's differently, and aligns 'N' with 'N', therefore reporting more matches than RazerS100 in setting C. In setting B, Shrimp finds additional matches due to its Smith-Waterman verifier which cannot be set to report valid Levenshtein matches only. SeqMap does not detect matches that reach into reference regions consisting of 'N's, thereby always reporting less matches than RazerS100. Zoom reaches 100% sensitivity in setting A only. It loses matches in setting B, as it allows only one gap; for setting C, Zoom could not be set to 100% sensitivity for matches with 5 errors. As Maq uses a quality-based scoring method which is more permissive in the number of alignment errors, we filtered its output for matches conforming to our maximum error level. This produces numbers more comparable to those of the other tools (the total count of mapped reads is shown in brackets).

In conclusion, RazerS compares very well with other read mapping tools at the time. Its fine-grained sensitivity control is a unique feature among tools and gives a good tradeoff between mapping sensitivity and speed. Another important conclusion from this evaluation was that comparing read mapping tools is not a trivial task. Most read mappers have slightly different objectives and different parameters that can be set. For a fair comparison, and in particular a quantitative evaluation of accuracy, a benchmarking method that compares results to a reference truth is needed.

3.6 Improved Benchmarking of Read Mapping Tools

From the evaluation of RazerS it became clear that measuring the performance of read mapping tools is not a trivial task. In the previous section, we used the number of mapped reads as an indicator of mapping accuracy. This is the measure commonly used to compare read mapping tools (Li *et al.*, 2008b; Jiang and Wong, 2008; Langmead *et al.*, 2009), the tool mapping the highest number of reads being the best. Obviously, this is not a very sophisticated way of comparing tools, as 1) best matches could be missed and suboptimal matches reported instead, and 2) even false positive matches could be reported. Also, for different variants of the read mapping problem (e.g. *all* or *all-best*, see chapter beginning "Semiglobal read mapping") this measure is insufficient, as no information about the completeness of the found match set is revealed. Ideally, we would like to compare the result of read mapping tools to a gold standard, i.e. the perfect answer a read mapper should find.

Therefore, we devised a method to do exactly this: a benchmarking method called *Rabema* (Holtgrewe *et al.*, 2011). This is mainly work by Manuel Holtgrewe. Own contributions are mostly in the evaluation of results including a comparison of several read mapping tools (see section 3.6.2). In the following we first outline the method and then show results.

3.6.1 Rabema - Generating a Gold Standard

The Rabema method computes a gold standard for a set of reads by using RazerS in 100% sensitivity mode. However, some additional considerations and computations are necessary for creating the gold standard and for comparing a set of reported matches to it.

First of all, given two matches we need to be able to decide if they are the same. This is surprisingly involved and represents the core difficulty in creating a well-defined benchmarking method. As indicated earlier in Figure 3.4, there can be some ambiguity in match begin and end positions whenever gaps are allowed in the alignment, making it insufficient to simply check if two matches' begin and end positions are exactly the same. Additionally, it is not straightforward how to treat matches in repeat regions. Repeat regions can lead to many possibilities to align a read, all of which are essentially the same. For example it is clearly not the goal of read mapping to report every single matching position of read "AAA" in a long "A" homopolymer run.

We therefore compute *match-equivalent* intervals within which all matches are defined as equal. By definition, we identify each match by its end position (or interval of end positions) in the reference genome.

Given a maximum number of allowed errors k, we define two equivalence classes that we need in order to define such intervals: *k-trace-equivalence* for the first case (alignment ambiguity, Figure 3.11A), and *neighbor-equivalence* for the second case (repeat regions, Figure 3.11B). We first introduce *trace-equivalence* which we need in order to understand k-trace-equivalence. In simple words, two matches are trace-equivalent if they share a part of their trace. This is the case if their end positions trace back to the same begin position (by definition, the rightmost one). End positions that lie between two trace-equivalent matches that have $\leq k$ errors are "smoothed", i.e.

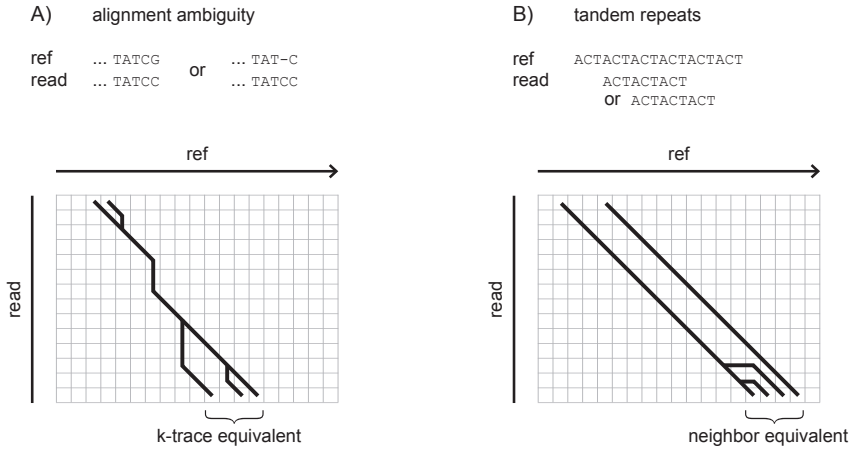

Figure 3.11: The two cases that complicate match equivalence definition: alignment ambiguity (A) and repeats (B). They are tackled through the definitions of k-trace equivalence and neighbor equivalence. Lines represent alignment traces with distance $\leq k$

included in the k-trace-equivalent interval, even if they have more than k errors. Two matches are neighbor-equivalent if their end positions are only separated by end positions that stay below error level k, which is the case in short tandem repeats. With the help of these two equivalence classes, we can finally compute our match-equivalent intervals. Formal definitions and further examples are given in (Holtgrewe *et al.*, 2011).

In the end, the gold standard consists of a set of match-equivalent intervals where each interval is annotated with its error level $e \leq k$. The output of a read mapper is then compared to this set of intervals where depending on the objective either all true matches, all best true matches, any best true match or any true match needs to be detected by the read mapper. The benchmark is also able to identify false positive matches (which in practise are rare). These are matches not contained in any interval, or only in an interval with error level larger than the reported one.

3.6.2 Rabema - Evaluation of Read Mapping Tools

In the following we show the results of a sample evaluation on three datasets, comparing mapping sensitivities of four popular read mappers: BWA (Li and Durbin, 2009), Bowtie (Langmead *et al.*, 2009), Soap2 (Li *et al.*, 2009b) and Shrimp2 (David *et al.*, 2011).

Evaluation Data

All data sets are from Drosophila melanogaster which has a moderately sized genome (~ 130 Mb).

- read set SRR026674 containing 36 bp Illumina reads.
- read set SRR049254 containing 100 bp Illumina reads.
- read set SRR034673 containing 454 reads of average length 273 bp.

From each set we use 10,000 randomly sampled reads.

Evaluation Setup

On the Illumina reads, we run Bwa in default mode as advised by its author. Also Shrimp2 is run in default, except for the use of weighted seeds (advised by the authors). For Bowtie and Soap2 we tested different parameter settings and include in our results the most sensitive ones we found as well as the default settings. For all tools we set the maximum number of matches to report to 100. On the 454 reads, we only evaluate Bwa (using bwasw) and Shrimp2, because Bowtie and Soap2 do not support gaps and are therefore practically unusable on 454 reads. Details on program calls can be found in the appendix.

To measure sensitivity we will use the measure of *normalized found intervals*. If a read matches in x intervals, each found interval contributes $1/x$ point, such that by summing up a read can contribute at most one point in total. We sum up all points of all reads and then divide (normalize) by the total number of reads. Multiplying by 100 gives us the percentage of normalized found intervals. We evaluate the *all* and the *any-best* problems.

Results

Sensitivity results for the *all* and *any-best* category are shown in Figure 3.12. Subfigures (A) and (B) show the results for the shortest read length tested (36 bp). Here, all tools achieve high sensitivities, particularly in the *any-best* category. The tuned versions of Soap2 and Bowtie (denoted by Soap2* and Bowtie*) perform the same as their default versions. Finding *all* matches, the tuned Soap2* performs significantly better than the default version Soap2, showing that Soap2's default settings are geared towards returning just one best match. For all tools, we see a slight decrease in sensitivity with increasing error rate.

On the longer 100 bp reads (subfigures (C) and (D)) we see a more dramatic decrease in sensitivity. Also, the difference between overall performance in the *all* compared to the *any-best* category is more pronounced. Most notably, the default versions of Soap2 and Bowtie experience a sudden drop in sensitivity, when error rate passes 2%. It appears that the default versions are geared to finding matches with at most 2 errors (corresponding to 2% error rate on the 100 bp reads). The tuned versions are able to achieve much better sensitivities (with a significant increase in running time, around factor 3 to 4). Overall, Bwa performs best in the *any-best* category, while in the *all* category Shrimp2 remains highly sensitive for all tested error rates.

On the 454 reads, we do not see a significant difference between the *all* and the *any-best* category (subfigures (E) and (F)). As the reads are much longer than the Illumina reads, they have fewer mapping locations and are thus less ambiguous, leading to less differences between *all* and *any-best*. We consistently see Shrimp2 being 10-20 percentage points more sensitive than Bwa. It is worth mentioning though, that Shrimp2 requires about an order of magnitude more memory and running time.

In conclusion, it seems that Soap2 and Bowtie are geared towards short reads (and gapless alignment), while Bwa and Shrimp2 perform well on different read lengths. Bwa is a very sensitive mapping tool, especially well suited for quickly reporting any best match of a read. Shrimp2 is overall the most sensitive at reporting all matches and therefore may be especially interesting in

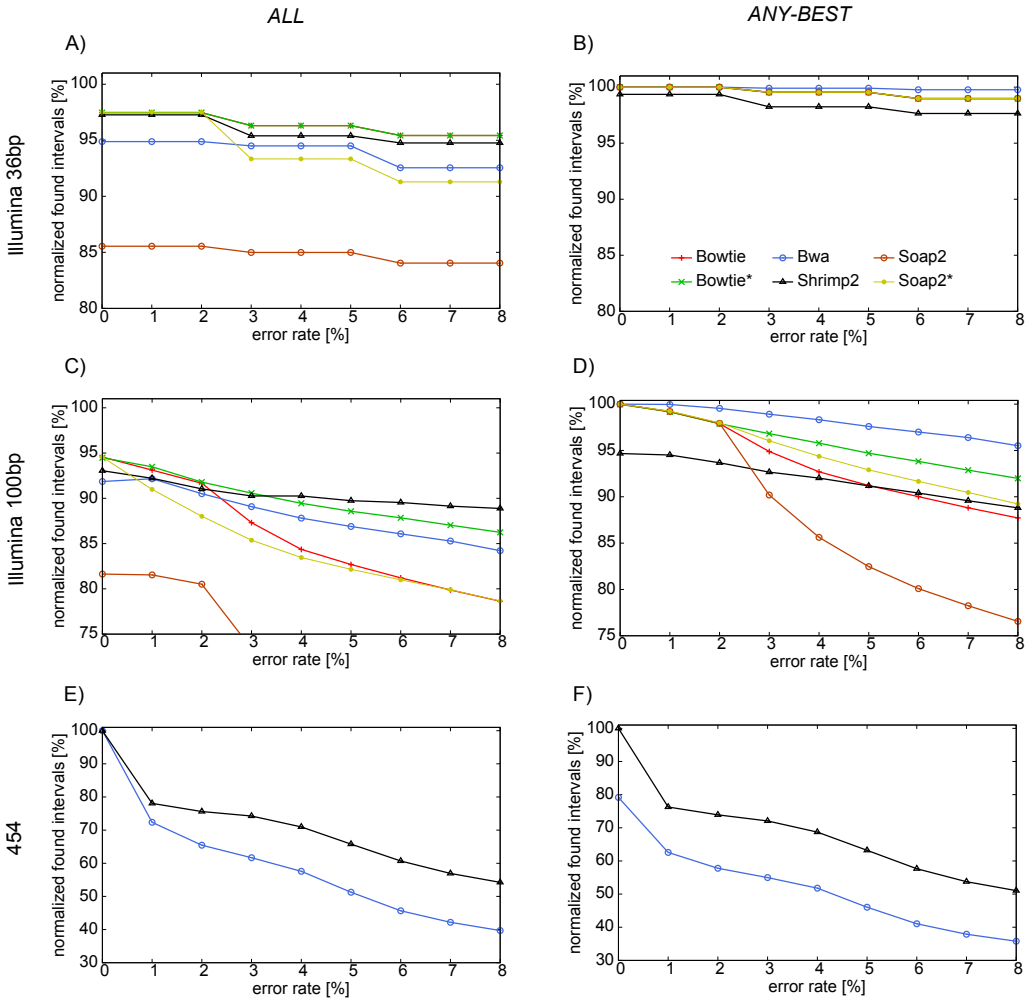

Figure 3.12: Comparison of second generation read mapping tools using the Rabema benchmarking method. Left column shows results for the *all* mapping problem, right column for the *any-best* problem. Plots in the first two rows are for Illumina reads (36 and 100 bp), and in the last row for 454 reads. Figure from (Holtgrewe *et al.*, 2011).

studies where sensitivity is valued over running time. We note that parameterization of tools is immensely important, as it can make a difference in sensitivity of more than 10 percentage points. This shows the importance of a benchmarking method, which can be used prior to deciding for a tool and a parameter setting for a certain task. Extending Rabema to other objectives in read mapping, e.g. for paired-end reads or for partial read mapping, will be even more involved, but might prove particularly fruitful for specific applications such as structural variant detection.

3.7 Chapter Summary

In this chapter, we

- gave an overview of algorithms and tools for read mapping.
- developed a semiglobal read mapping tool RazerS based on a q-gram counting approach that supports Hamming as well as edit distance.
- integrated a mapping sensitivity control switch, that is based on a dynamic programming procedure for filter sensitivity calculation, and that provides controlled runtime-sensitivity tradeoff.
- derived a heuristic for computing good (heavy) gapped shapes for Hamming distance mapping with 100% sensitivity.
- compared RazerS with other read mapping tools and tested its sensitivity estimation.
- introduced the benchmarking mehod Rabema that uses RazerS in 100% sensitivity mode (plus additional calculations) to create a well-defined read mapping gold standard (Rabema).
- finally gave a sample evaluation of read mapping tools using this gold standard.

We observed that

- q-gram counting is rare among read mapping tools, most tools rely on a pigeonhole-based filtering-and-verification approach, or use a (FM-) index backtracking procedure.
- RazerS' sensitivity estimation method gave a good lower bound on achieved mapping sensitivity.
- the speed up in read mapping was significant even when sacrificing only little in terms of sensitivity.
- the shape computation heuristic found an optimum-weight shape for all tested cases (up to 50 bp).
- using gapped shapes for Hamming distance mapping lead to a significant speed up compared to edit distance.
- RazerS could efficiently deal with NGS read data and was among the fastest tool, especially in 99% sensitivity mode.
- comparing read mapping tools is not straightforward: all tools had slightly different objectives and parameters.
- comparing read mapping accuracy of different tools is not trivial either and in most publications (including RazerS) simply the number of mapped reads is used as sensitivity estimate.

- creating a benchmarking method is quite involved, as defining when two matches are the same is complicated by alignment ambiguity and repeat sequences.
- many popular read mappers are lossy, especially for matches with more than 2 errors.

From this we conclude that

- RazerS' sensitivity switch, which is a unique feature among read mapping tools, gives it an effective sensitivity-speed tradeoff.
- RazerS can report all matches, i.e. guarantee 100% sensitivity for many different settings and including suboptimal matches, giving it an important advantage over other tools.
- we created the first benchmarking method that takes a more sophisticated approach at comparing the results of read mappers (compared to simply counting mapped reads).
- only in comparison to a gold standard we can quantitatively evaluate the sensitivity of read mapping tools.

Chapter 4

Partial Read Mapping and Applications

Semiglobal read mapping, the focus of the previous chapter, assumes reads to differ only slightly from the reference sequence. Reads that differ too much from the reference, e.g. because they contain or intersect with a genomic variant, will remain unmapped, or in the worst case may even be assigned to a wrong location. However, these are the reads we are especially interested in, as they carry the potential to identify these variants. Accurate variant discovery is of major importance for our understanding of genome evolution and for our power to identify disease-causing variants, ultimately on a diagnostic level (Voelkerding *et al.*, 2009). For variant-spanning reads, we require specialized read mapping types that are tailored to the problem at hand. In addition, the type of read data and use of reference sequence can necessitate different read mapping approaches.

Figure 4.1 gives an overview of some *partial* read mapping types with their diverse applications. In *prefix mapping* the read alignment starts at the first read base, but does not have to extend all the way to the last base. Prefix mapping has applications in small RNA read mapping (Emde *et al.*, 2010) and in SV (especially long insertion) breakpoint discovery (Suzuki *et al.*, 2011; Karakoc *et al.*, 2012). *Split read mapping*, where the read alignment is split into parts, has become a popular method for detecting SVs. Its main strength lies in the power to locate SVs with base pair precision, which is crucial for downstream analysis of SVs, such as predicting the functional impact on protein level or for studying variation formation processes (Lam *et al.*, 2010). Usually, the read is split into two parts that map in collinear order only interrupted by a longer gap in the read-to-reference alignment (*prefix-suffix mapping*). Prefix-suffix mapping has application in indel detection (Ye *et al.*, 2009; Emde *et al.*, 2012; Karakoc *et al.*, 2012). Another prominent application of split read mapping is in RNA-Seq for mapping reads crossing splice junctions (Au *et al.*, 2010; Wang *et al.*, 2010). If the read may be split into more than two parts (*multi-split mapping*) and the collinearity of mapped read parts is not enforced, we have the most generic form of split mapping. *Generic split mapping* (Trappe, 2012) has the potential to reveal more complex types of SVs such as interchromosomal rearrangements or gene fusions, which are of particular interest in cancer

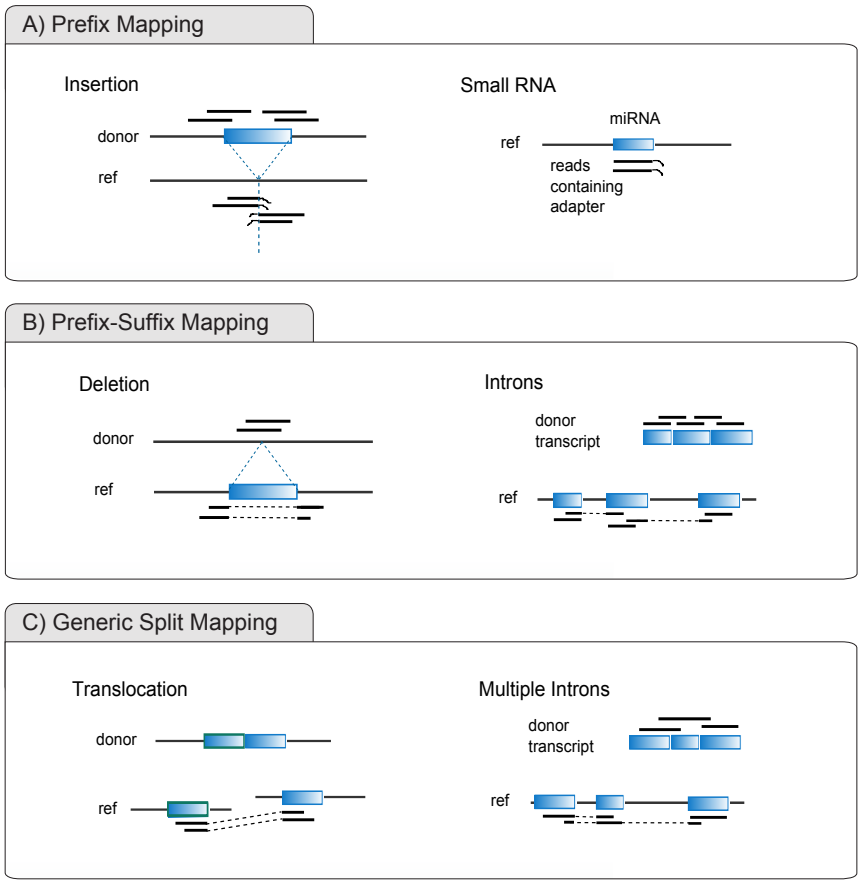

Figure 4.1: Overview of the different types of partial mapping. A) Prefix-based mapping for example finds application in insertion breakpoint discovery or in mapping of small RNA reads containing 3' adapter sequence. B) Prefix-suffix mapping can identify reads that span deletions or introns in transcriptomic data. C) The most generic form of split mapping can additionally map reads spanning complex SVs or multiple introns.

studies (Campbell *et al.*, 2008; Maher *et al.*, 2009).

Due to the already short read lengths and the partitioning of the read into even smaller parts, split read mapping struggles with even more alignment ambiguity and computational complexity challenges than semiglobal read mapping. Shorter sequence parts produce more random matches, and the more freely and distantly these match parts are allowed to be located on the genome, the more combinatorial explosion we face. So far split mapping has therefore been used mainly for paired end reads, where one end is mapped and serves as an anchor, while the other end could not be mapped, presumably because of an SV (Ye *et al.*, 2009). We speak of *one-end anchored* reads in this case. Through one-end anchoring we know where the unmapped read should map approximately. Thus anchoring significantly reduces search space for split mapping, thereby also reducing mapping ambiguity and uncertainty. As read lengths increase with advances in sequencing technology, split read mapping has become and is still becoming more and more powerful, and therefore important as a method for SV detection with base pair precision.

In this chapter we address two methods for partial read mapping that find application in small RNA read mapping and in indel discovery. First, we describe and evaluate the prefix mapping approach implemented in MicroRazerS (Emde *et al.*, 2010) for detection of miRNAs. To our knowledge, MicroRazerS was the first tool to implement a prefix-based mapping approach that makes adapter trimming of small RNA reads unnecessary and efficiently scales to large NGS data sets. MicroRazerS is joint work with Marcel Grunert. Then, we will turn to the main focus of this chapter: split read mapping with SplazerS (Emde *et al.*, 2012) for indel detection. Here, the novelty lies in its support for paired- as well as single-end data and especially its successful application to indel detection in a large-scale single-end resequencing data set.

4.1 Small RNA Read Mapping

Small RNA reads are special in that the sequenced read length may surpass small RNA length (19-25 bp in the case of miRNA). Sequencing may therefore reach into the sequencing adapter. When mapping small RNA reads onto a reference sequence (usually genome or database of known smallRNAs) this needs to be taken into account.

4.1.1 Strategies for Mapping Small RNA Reads

Mapping small RNA reads either requires the adapter sequence to be removed first such that traditional semiglobal read mapping can be applied, or involves an alignment method that can directly handle the adapter sequence. Adapter removal is far from trivial (Wang *et al.*, 2009a; Kong, 2011), as also the sequencing of adapter may produce sequencing errors, even more so as sequencing qualities degrade toward the 3′ end where the adapter is located. Unrecognized adapter will prevent the read from mapping semiglobally.

Instead of removing adapter sequence, early miRNA studies (Friedländer *et al.*, 2008; Morin *et al.*, 2008) therefore used the local alignment tool **Mega BLAST** (Zhang *et al.*, 2000) to map small RNA reads, rigorously filtering its output for matches adhering to certain criteria. These criteria include a certain minimum match length and a maximum number of errors; additionally, the match is required to start at the first read base.

Building on these strategies, we developed **MicroRazerS** which uses q-gram counting for finding a high quality prefix seed match which is then extended until the first mismatch is encountered. Adapter sequence does not have to be trimmed, thereby avoiding problems that come from undertrimming (cannot map undertrimmed read semiglobally) or also overtrimming (cannot map overtrimmed read uniquely). The algorithm is explained in the following.

4.1.2 MicroRazerS - Algorithm

MicroRazerS is joint work with Marcel Grunert. Marcel Grunert was mainly responsible for tool evaluation, while own work was mainly in tool implementation. Both contributed equally through discussion and manuscript preparation.

Given a minimum match length m and a number of allowed mismatches k, we define a valid prefix match of read r as an alignment $a_{r_{1,x}}$ for which holds:

1. $m \leq x \leq |r|$
2. $d(a_{r_{1,m}}) \leq k$
3. $d(a_{r_{m+1,x}}) = 0$
4. $d(a_{r_{x+1}}) = 1$

In other words, the first at least m bases of r need to be aligned (1). The m-prefix alignment (which we call seed from now on) is allowed to have at most k mismatches (2). Outside the seed, the alignment is required to be error-free (3) and the alignment needs to be maximally extended (4). Note that we only support Hamming distance in MicroRazerS. At most one mismatch is allowed in the seed, i.e. $k \leq 1$.

If multiple such matches exist, they are ranked according to 1) the number of mismatches in the seed (the lower the better) and 2) total alignment length x (the longer the better). All best matches or up to a user-defined maximum number of matches are reported.

MicroRazerS' core algorithm is very similar to RazerS (see pseudocode in appendix) and the implementation uses many parts of the RazerS code (mostly implemented by David Weese). An overview is given in Figure 4.2.

Parameter Choosing and Filtering

Like RazerS, MicroRazerS makes use of the parameter choosing functionality of ParamChooser (see section 3.4.3). In contrast to RazerS, the parameter choosing part only considers the seed of length m, independent of total read length. Parameter files were computed using a uniform error profile, the rationale being that the seed part (usually 16-20 bp) should be high quality with essentially equal sequencing error probabilities.

Given seed length m and k mismatches, the corresponding parameter files are searched for the best filter criterion. The q-gram index is built over all q-grams contained in the seed part of the reads. Filtering then proceeds in the same way as in RazerS, scanning over the genome sequence with the SWIFT filtering method (described in section 3.3.1).

Verification

We use the simple diagonal scanning approach as in RazerS (section 3.3.2). A seed match is found if the prefix of length m matches the reference sequence with at most k mismatches. Multiple seed matches may exist per parallelogram. A best seed match is one that has the minimum number of mismatches among all seed matches in the current parallelogram. Each best seed match is extended until the first mismatch is encountered. The longest total match is returned. In case of ambiguity, the first such match found is returned.

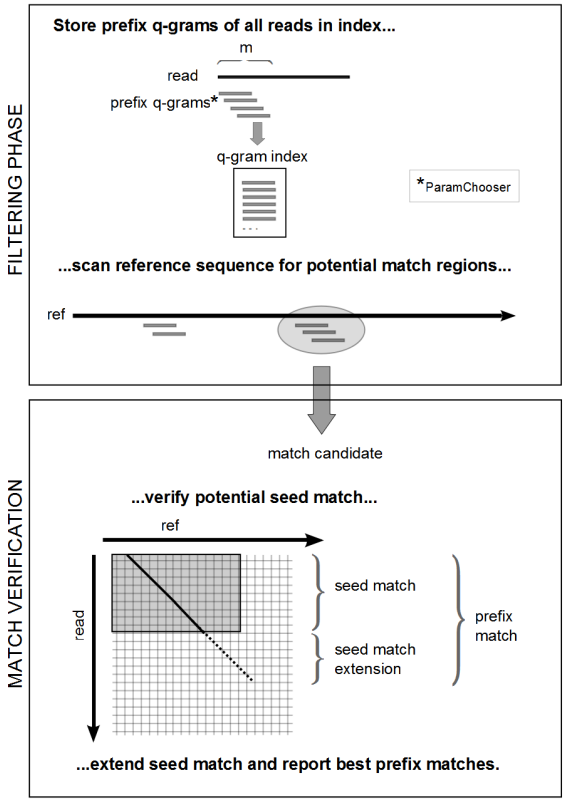

Figure 4.2: Overview of the MicroRazerS algorithm. In the filtering phase, a q-gram index is built over all prefix q-grams and the reference sequence is scanned for potential prefix seed matches. These are then verified and extended during match verification.

4.1.3 MicroRazerS - Evaluation and Comparison with Other Tools

In the following, we provide our evaluation of MicroRazerS in comparison to other mapping tools, as in (Emde *et al.*, 2010).

Evaluation Data

Small RNAs were isolated from human normal heart RNA and prepared for Illumina sequencing. Sequencing of the small RNA library yielded 9,286,222 36 bp reads. Small RNA read data sets are highly redundant, we therefore first sorted all reads and removed redundant sequences (keeping a count of how many times the read sequence occurred). In this way, the $\sim 9\,M$ reads were reduced to 2,402,361 unique read sequences. As reference sequence we use the human reference genome hg18 (NBCI build 36.1).

Evaluation Setup

We mapped the $\sim 2.4\,M$ unique reads to the reference genome using Mega BLAST, Soap2, Bowtie and MicroRazerS. For MicroRazerS, we set the seed length to 16, not allowing any errors. We set

the maximum number of matches to report to 20 (miRNAs can appear multiple times in the genome). Reads having more than 20 matches were discarded. Also Soap2 allows to set a seed length which accordingly was set to 16 bp. As Soap2 and Bowtie compute semiglobal alignments, we allow 20 mismatches (read length minus seed length) to guarantee to find all 16 bp prefix matches regardless of the rest of the read. Mega BLAST computes local alignments and was run in default mode. Details on program calls are given in the appendix. The output of Mega BLAST, Soap2, and Bowtie were filtered to get the best (longest) prefix matches. Reads with more than 20 such alignments were discarded.

Of all mapped reads, we count how many reads map to sequence annotated as miRNA according to miRBase (Release 13.0) (Griffiths-Jones *et al.*, 2008). Additionally, we count the number of different miRNAs that have a read count of more than 150 (as miRNAs detected with high confidence). All running times were measured on an AMD Opteron 2384 with 32 GB memory running a 64-bit Linux system.

Results

Our comparison of MicroRazerS with Mega BLAST, Soap2 and Bowtie is summarized in Table 4.1. Mega BLAST is an order of magnitude slower than the other tools. Soap2 and Bowtie are very fast but require an index construction preprocessing step that is computationally expensive. Bowtie's index construction takes even more time than the Mega BLAST search. The most space-consuming tool is Soap2 with more than 8 GB of memory usage. Output file size is huge (> 6 GB) for BLAST and Soap2, and filtering of their results to meet the prefix match criteria requires an additional, time-consuming post-processing step ($\sim 30\,min$ using Perl scripting). Of note, MicroRazerS provides the most convenient handling of small RNA reads, as no pre- or post-processing of reads is necessary. Of all programs, MicroRazerS is able to map the highest number of reads. As argued in the previous chapter, simply counting the number of mapped reads is not the best measure for sensitivity. However, MicroRazerS also yields the highest number of detected miRNAs together with Soap2, indicating highest sensitivity.

	MicroRazerS	**BLAST**	**SOAP2**	**Bowtie**
Running time [min]	24	194	6	5
Building index [min]	-	-	84	206
Output size [GB]	0.1	8.6	6.8	0.7
Memory usage [GB]	3.4	1.4	8.3	2.3
Unique sequences aligned	1,319,218	891,215	1,318,504	1,184,590
Mappable reads	7,743,516	7,001,832	7,742,266	7,410,239
Reads annotated as miRNA	5,819,189	5,746,588	5,819,184	5,667,027
MiRNAs	381	372	381	372
(read count > 150)	101	96	101	99

Table 4.1: Evaluation of small RNA mapping tools. We used a query dataset of ∼2.4M non-redundant read sequences (length 36 bp) representing a total of ∼9.3M reads.

Furthermore, Cordero *et al.* (2012) evaluated MicroRazerS and other more recent mapping tools specialized on, or providing a specific set of parameters for, small RNA read mapping. Their results on miRNA spike-in data "clearly indicate that SHRiMP and MicroRazerS provide the best miRNA detection rate" (Cordero *et al.*, 2012). According to Table 2 in their article, RazerS is also the fastest tool in the comparison.

4.2 Overview of Split Read Mapping Tools

Split read alignments are alignments of reads split into two or more parts. As introduced in section 2.2.4, split read methods are among the most popular approaches for discovering SVs using sequencing data. Usually, only reads that are unmapped, but anchored by a confidently mapped paired read, are considered; this technique is called anchored split-read mapping. Compared to read pair or read depth methods (see section 2.2.4), the strength of split-read methods lies in the potential to identify SVs at base pair resolution.

Ye *et al.* (2009) published the first tool, **Pindel**, for split mapping of short NGS reads. It requires reads to be in paired-end layout, performing anchored split mapping only. In its initial version, it only considered collinear prefix-suffix matches for insertion and deletion detection. Later versions have been extended to also detect inversions and tandem duplications. Pindel is based on a pattern growth algorithm and requires prefix and suffix match to be unique in order to be combined into a split read match.

Two more generic read mapping tools, that also provide split read mapping functionality are GSNAP (Wu and Nacu, 2010) and BWA (Li and Durbin, 2009). **GSNAP** indexes every third 12-gram in the genome and then processes each read one by one. Its filtering strategy combines the pigeonhole principle with a q-gram counting approach and guarantees full sensitivity only for matches containing at least an exact match of 14 contiguous base pairs. **BWA**, introduced in chapter 3, uses a heuristic approach that can allow one long gap in its FM-index backtracking procedure. Both GSNAP and BWA do not have indel calling functionality integrated, but output collinear prefix-suffix matches which can then be used as input for an indel calling tool. Moreover, neither tool provides direct support for anchored split mapping.

We thus developed our own split read mapping tool, **SplazerS**, which will be described in detail in the following section. Its main advantage over other tools (at that time and still) is that it has an anchored/paired-end mode as well as an unanchored/single-end mode. Especially with increasing read lengths, single-end split read mapping is becoming more valuable, as we will see later.

Two recent methods that were published around the same time as SplazerS are Splitread (Karakoc *et al.*, 2012) and SVseq2 (Zhang *et al.*, 2012). **Splitread** performs an anchored split read mapping step, artificially partitioning anchored reads into balanced and unbalanced splits. A maximum parsimony approach is taken to find the minimum number of breakpoint events indicated by the split reads, using a set cover approximation algorithm. Indels are required to have at least one balanced split read, as they are more trustworthy than unbalanced ones. **SVseq2** is to our knowledge the

first published tool to combine split read mapping with the read pair method. First, discordant read pairs identify candidate deletions, which are then called if also supported by split reads.

Further methods such as SpliceMap (Au *et al.*, 2010), MapSplice (Wang *et al.*, 2010), or SplitSeek (Ameur *et al.*, 2010) follow similar approaches but are geared toward splice junction discovery in RNA-Seq data. Typical drawbacks of these methods for indel detection are a lack of functionality for insertion detection, or their requirement for donor/acceptor splicing patterns.

In the following, we introduce our own SeqAn tool SplazerS in detail.

4.3 SplazerS - Split Read Mapping for Indel Detection

SplazerS (Emde *et al.*, 2012) is, as RazerS and MicroRazerS, implemented in the SeqAn library. It detects collinear prefix-suffix matches and supports anchored as well as unanchored split read mapping. In the anchored mode, it will only detect split matches of one-end anchored reads within the region defined by the mapped anchoring read. In the more general case, split matches of unanchored/single-end reads anywhere in the reference sequence will be reported. The input reads may be of arbitrary read length, and both Hamming and Levenshtein distance are supported.

First, we define what a valid split match is in SplazerS. Then, we turn to the algorithmic details of how these matches are found. Finally, we evaluate SplazerS' performance in indel detection by using its output in conjunction with a simple indel calling strategy. The remainder of this chapter is heavily based on and in part taken from (Emde *et al.*, 2012).

4.3.1 Split Match Definition

Given a (non-empty) prefix p of read r that aligns to genome subsequence $g_{i,j}$, and a (non-empty) suffix s of r that aligns to $g_{u,v}$ (see Figure 4.3). W.l.o.g. we assume our reads to only match to the forward strand, and define a collinear prefix-suffix read alignment as an alignment where the following holds:

1. $|p| + |s| = |r|$ and $j + 1 < u$ (read indicates a deletion) or
2. $|p| + |s| < |r|$ and $j + 1 = u$ (read indicates an insertion)

where $1 \leq i \leq j < u \leq v \leq |g|$.

In order for the split read alignment to be a *valid* prefix-suffix match, SplazerS employs a number of additional criteria. Given a minimum match length m, a maximum error rate ϵ, a maximum number of prefix (suffix) errors e_p (e_s) and a maximum gap length δ, SplazerS reports collinear prefix-suffix matches where the following holds:

1. $|p| \geq m$ and $|s| \geq m$
2. $d(a_{p_{1,m}}) \leq e_p$ and $d(a_{s_{|s|-m+1,|s|}}) \leq e_s$
3. $\frac{d(a_p)+d(a_s)}{|p|+|s|} \leq \epsilon$
4. $u - j - 1 \leq \delta$

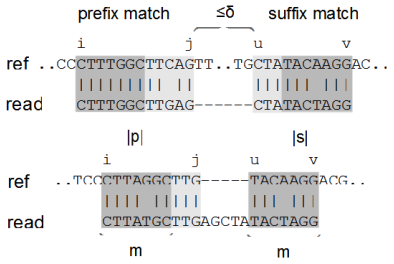

Figure 4.3: Two examples of split read alignments. Given parameters $m = 7$, $e_p = e_s = 1$ and $\epsilon = 0.1$. a) is a valid alignment spanning a deletion and b) spans an insertion but is not a valid match as the error rate condition is violated.

The first condition ensures that prefix p and suffix s have at least the minimum match length m. The second condition ensures that the number of errors in the minimum length prefix (suffix) match is at most a certain maximum number e_p (e_s). Condition 3 guarantees that the sum of the number of errors in the prefix and suffix match divided by the combined length of prefix and suffix lies within the allowed error rate ϵ. Note that we use an error rate instead of a number of errors k to make the number of errors allowed depend on the actually aligned read length $|p| + |s|$. The maximum gap length δ puts a constraint on the distance of prefix and suffix match. An example of a valid and an invalid split read match is given in Figure 4.3.

For *valid anchored* split read matches we further restrict the genomic region that the read is allowed to map to. This region is defined by the location of the confidently mapped mate and the expected insert size (e.g. insert size ±3 standard deviations).

If there are multiple split matches for a read, SplazerS ranks them according to a score. Each match with a middle gap of length ≥ 1 receives a score $sc = |p| + |s| - 2 \cdot (d(a_p) + d(a_s)) - c(r)$, while matches without a middle gap, i.e., matches that map normally and do not indicate an indel event, receive score $sc = |p| + |s| - 2 \cdot (d(a_p) + d(a_s))$. The parameter $c(r)$ puts a penalty on the existence of a gap independent of its length and is set depending on average error probability of the reads (by default $c(r) = \lfloor 0.03 \cdot |r| \rfloor$). Matches with the same score are ranked according to the length of the middle gap (the shorter the better). We here define a read match as unique, if it is the single match with highest score. Multiple best and also suboptimal matches can be reported.

Depending on total read length, minimum match length and numbers of errors allowed on prefix and suffix match, the probability of a random match can get quite high; especially for unanchored split read mapping, where the whole reference sequence is searched. We therefore provide an estimate of the expected number of random matches, given a split match validity criterion, which can help the user choose sensible parameters. For calculations see appendix.

4.3.2 Algorithm

Figure 4.4 shows an overview of the SplazerS algorithm. It is based on the same core algorithm as RazerS (see appendix for pseudocode). SplazerS uses two filters in the filtering phase, one identifying potential prefix matches, the other potential suffix matches. These are verified in the

match verification phase and combined into split matches during match combination. The steps will be explained in detail in the next subsections.

Filtering

Like all tools in the RazerS family, SplazerS is based on a SWIFT filter (section 3.3.1). More precisely, it is based on two filters (as originally implemented by David Weese in RazerS' paired-end module). The left filter detects potential prefix matches, while the right filter detects potential suffix matches. Parameters for filtering are chosen by ParamChooser (section 3.4.3). Assuming a uniform error profile here, we estimate the lower bound on sensitivity by the product of the sensitivity rates of the left and the right filter, as these operate independently of each other. This is just an approximation, as it does not take valid split match transcript frequencies into account. It is therefore just an estimate for the sensitivity of detecting an m-prefix match with e_p and an m-suffix match with e_s errors at the same time.

For the two filters, we need two q-gram indices: the left index containing all q-grams of all read prefixes of length m, and the right index containing all q-grams of all read suffixes of length m. We start scanning the reference sequence with the right filter until we encounter a potential suffix match of a read r. We let the left filter follow up to the current position of the right index, recording all potential prefix matches within the allowed distance δ in a queue. If there is at least one such potential prefix match for read r, this triggers the verification of the potential suffix match of r. Verification proceeds as described in the next paragraph. Only if the verification returns a valid suffix match, also all potential prefix matches are subjected to verification. In order to avoid having to verify a potential prefix match more than once, the queue keeps track of the verification status of each potential match. Potential prefix matches outside the allowed region as defined by δ are removed from the queue.

Match Verification

We will describe match verification for prefix matches only, as the procedure is directly transferable to suffix match verification by reversing read and reference subsequence. Our goal is to identify the longest prefix match, the *extended prefix match*, that contains up to $k = \lfloor |r| \cdot \epsilon \rfloor$ errors. This is the maximum number of errors allowed on the whole split read match. As prefix and suffix match verification are independent from each other, we have to assume the worst case error scenario. In the worst case for the prefix, the suffix match is error-free, such that k errors are allowed on the prefix match (maintaining the condition of at most e_p errors on the prefix of length m). A valid extended prefix match is thus a match $a_{r_{1,z}}$ with $m \leq z \leq |r| - m$ where $d(a_{r_{1,m}}) \leq e_p$ and $d(a_{r_{1,z}}) \leq k$. The longest possible read prefix match length is $|r| - m$ as a suffix of at least length m still needs to match.

For Hamming distance we use the diagonal scanning approach (section 3.1.3) and search for the longest extended prefix match that maintains the above criteria. In case of ambiguity, the first such match found is recorded. For Levenshtein distance, we first use Myers' bit vector algorithm to find an m-prefix match $a_{r_{1,m}}$ with at most e_p errors. This match is then extended using gapped

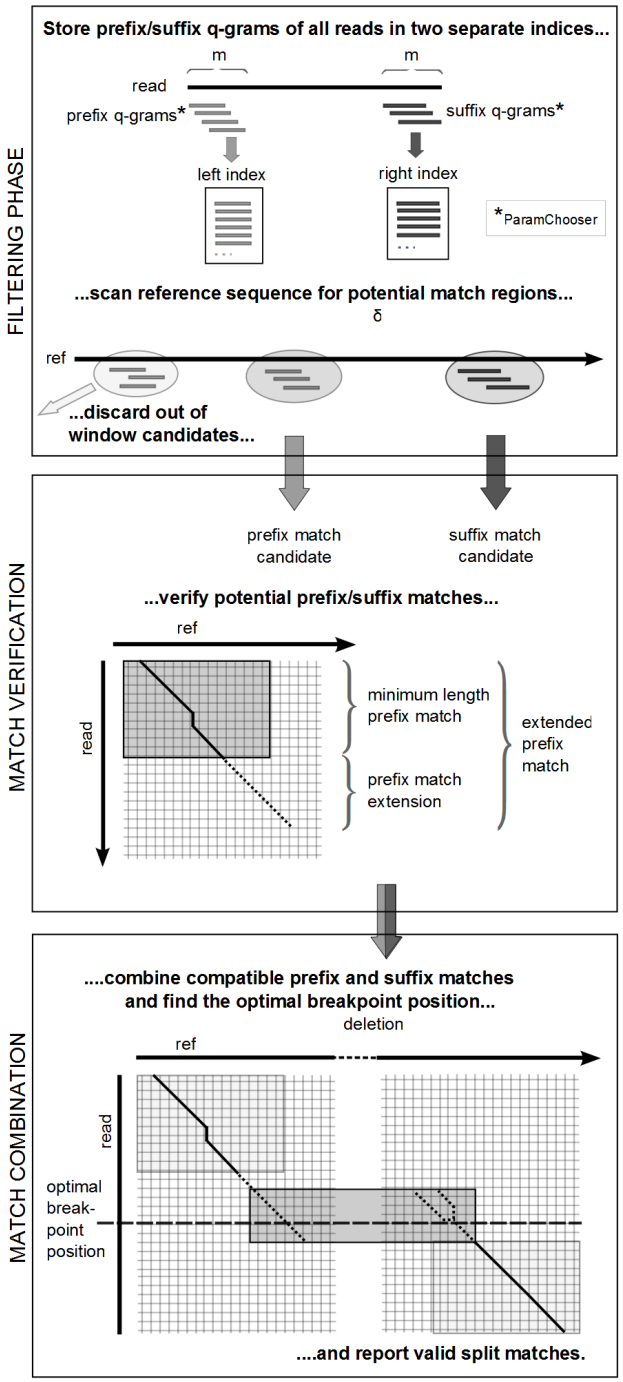

Figure 4.4: Overview of the SplazerS algorithm. During filtering, the reference sequence is scanned with two q-gram indices: the left index storing prefix q-grams and the right index storing suffix q-grams. Whenever a potential prefix and a potential suffix match are found within the allowed distance δ, first the suffix match is verified and if verified positively, then also the prefix match is subjected to verification. Match verification relies on a seed-and-extend approach first verifying the minimum length prefix (suffix) and then extending maximally to the right (left). During match combination, compatible extended prefix and suffix matches are combined into a split match and the optimal breakpoint position is located.

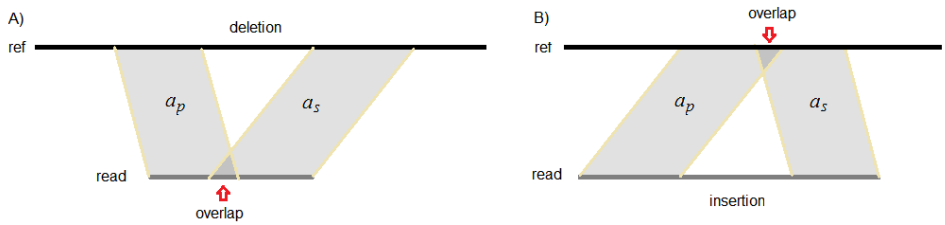

Figure 4.5: Extended prefix and suffix match overlap on A) read sequence if read spans a deletion, B) on genome sequence if read spans an insertion.

X-drop extension (Zhang *et al.*, 1999) allowing a score drop of at most $k - d(a_{r_{1,m}})$ where we use a Levenshtein distance scoring scheme (mismatch = gap = -1, match = 0). The extension stops once the additionally collected errors would exceed $k - d(a_{r_{1,m}})$, or when read position $|r| - m$ is reached. Again, in case of ambiguity, the first such match found is returned.

Breakpoint Computation

We now want to combine an extended prefix match $a_p^{g_{i,j}}$ and an extended suffix match $a_s^{g_{u,v}}$, where p is a prefix and s a suffix of read r. As $a_p^{g_{i,j}}$ and $a_s^{g_{u,v}}$ have been extended as far as possible, collecting up to k errors each, they will overlap (such as shown in Figure 4.5), if they can be combined into a valid split read match.

First, we check whether the basic criteria from section 4.3.1 can be fulfilled. We use function $d_g(x)$ to denote the maximum number of allowed gaps in x errors; we then have for Levenshtein distance $d_g(k) = k$, for Hamming distance $d_g(k) = 0$. We know that $a_p^{g_{i,j}}$ and $a_s^{g_{u,v}}$ cannot be combined into a valid split match if one of the following holds:

1. $v - i > \delta + d_g(k)$ (matches are too far apart, maximum gap length δ cannot be met)
2. $v - i < 2 \cdot m - d_g(e_p) - d_g(e_s)$ (matches are too close together, minimum prefix and suffix lengths m cannot be met)
3. $v - i \geq |r| + d_g(k)$ and $|p| + |s| < |r|$ (r would indicate a deletion, but matches do not touch with respect to read sequence, maximum error criterion cannot be met)
4. $v - i < |r| - d_g(k)$ and $j + 1 < u$ (r would indicate an insertion, but matches do not touch with respect to genome sequence, maximum error criterion cannot be met)

In the following, we explain breakpoint computation for the case of a potential deletion. In case of a potential insertion, we have the analogous case with r and g switched (insertion in r corresponds to a deletion in g). Following condition (3) above, the matches overlap on the read sequence, i.e. $|p| + |s| > |r|$, such that we need to resolve the overlapping read part. Read overlap begins at read position $|r| - |s| + 1$ and ends in position $|p|$, so our goal is to resolve the alignment of subsequence $r_{|r|-|s|+1,|p|}$. The non-ambiguous alignment parts $a_{r_{1,|r|-|s|}}^{g_{i,j'}}$ and $a_{r_{|p|+1,|r|}}^{g_{u',v}}$ are fixed. We denote with j' and u' the genomic positions aligned with read positions $|r| - |s|$ and $|p| + 1$, respectively.

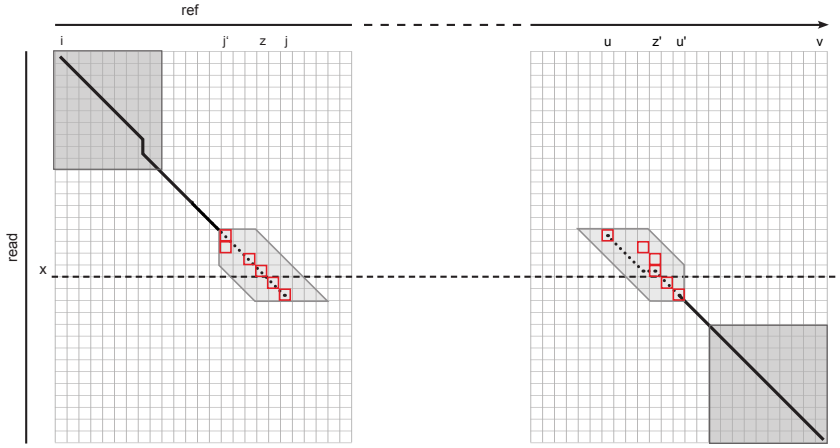

Figure 4.6: Example for the breakpoint computation of a deletion-spanning read. The fixed prefix and suffix alignment parts are represented by solid lines (where the prefix and suffix of minimum length m are indicated by dark gray background) and the dashed lines show the overlapping part that is resolved through a banded alignment procedure (light gray). Red squares represent cells with optimal scores per row.

To compute the optimal breakpoint position for Hamming distance, we again use a simple scanning approach. By scanning along the overlapping part of the prefix and suffix alignment, we find the read position x with $|r| - |s| + 1 \leq x \leq |p|$ where $d(a^{g_{i,f}}_{r_{1,x}}) + d(a^{g_{h,v}}_{r_{x+1,|r|}})$ is minimal. We denote with f and h the genomic positions aligned with read positions x and $x+1$, respectively. In case of ambiguity, we choose the leftmost position x, which is the common choice in variant detection methods.

For Levenshtein distance, the computation is more complex. Figure 4.6 illustrates the breakpoint computation procedure with an example. It is not sufficient to scan the existing prefix/suffix alignments, as these are not necessarily optimal with respect to the optimal breakpoint position. Instead, we compute two global alignment matrices: F^p for $r_{|r|-|s|+1,|p|}$ with $g_{j',j}$ and F^s for $r_{|r|-|s|+1,|p|}$ with $g_{u,u'}$, using a Levenshtein distance scoring scheme (mismatch = gap = -1, match = 0). Both alignment matrix computations can be banded by $e = k - d(a^{g_{i,j'}}_{r_{1,|r|-|s|}}) - d(a^{g_{u',v}}_{r_{|p|+1,|r|}})$, as at most k errors are allowed in total and $d(a^{g_{i,j'}}_{r_{1,|r|-|s|}}) + d(a^{g_{u',v}}_{r_{|p|,|r|}})$ have already been consumed by the fixed alignment parts.

We are interested in locating the combination of prefix and suffix match, i.e. read position x, where the sum of scores in F^p and F^s is optimal. Note that suffix sequences are actually reversed to compute scores for the alignment starting in read position $|p|+1$ and genome position $u'+1$, which we want to extend to the left. For easier notation, however, we use absolute read and reference positions here. Thus, we are searching for read position x and genome positions z and z' where $F^p_{x,z} + F^s_{x+1,z'}$ is optimal. In order to find this optimal breakpoint, we could keep the whole matrices and for each row (i.e. each read position x), check for the best combination of cells from F^p and F^s. This would require $o \cdot (2e+1)^2$ comparisons, where $2e+1$ is the bandwidth. However, as we are only interested in optimal combinations, i.e. those with maximal score, it suffices to store the optimal score value per row. We therefore keep two vectors f^p and f^s that store the best score for each row in F^p and F^s, respectively. In addition, we store in b^p and b^s the

corresponding genome position for each entry in f^p and f^s, respectively. If there is more than one cell with maximal score, we keep the leftmost (for suffix rightmost) position for technical reasons. This choice effectively maximizes middle gap length in the case of ambiguity. Then finding the read position x that maximizes $f^p_x + f^s_{x+1}$ gives the optimal score and the corresponding entries b^p_x and b^s_{x+1} provide the corresponding genome positions z and z'. The number of combinations that we need to check is thus reduced to o. As for Hamming distance, there may be multiple such optimal breakpoint positions, and we keep the leftmost one by convention.

Remarks:

Note that if $|r| - k \leq v - i \leq |r| + k$ the prefix alignment start i and the suffix alignment end v lie within the k-band of a semiglobal read alignment. If $v - i < |r|$ we are more likely to be dealing with a potential insertion, in case of $v - i > |r|$ a deletion is more likely. However, we cannot be sure, and we actually need to test both cases, insertion and deletion, and then store the one with the better score.

Also note that by choosing to store the outermost position of best score in b^p and b^s, we favor insertions over mismatches immediately next to the breakpoint position. Thus we may have a 1 bp insertion next to a 100 bp deletion, rather than a mismatch next to a 99 bp deletion. This seems reasonable as larger variants are often accompanied by micro-indels. However, this behavior is a matter of definition and can be easily changed during match output.

Our breakpoint computation method is very similar to a method developed independently around the same time. The AGE method (Abyzov and Gerstein, 2011) also computes two alignment matrices, but uses a Smith-Waterman like scoring scheme (rewarding matches). Instead of storing the best score value per row, it computes two additional matrices that store the maximum of the leading/trailing submatrix. It can therefore accommodate for combinations of SVs (for example a large deletion next to a large insertion), but does not necessarily guarantee unambiguous breakpoint placement.

4.4 SplazerS - Results

We extensively tested SplazerS and its ability to correctly detect indels in paired-end as well as single-end data. To this end we use SplazerS in conjunction with a simple indel calling method (using the SeqAn tool SnpStore which will be the subject of the next chapter). The indel calling procedure considers each pair of reference position and indel length observed in the split mapped reads as an indel candidate. All reads that overlap by a certain minimum number of base pairs with the indel candidate are considered spanning reads. Two thresholds are then checked for each indel candidate: a minimum number and a minimum percentage of spanning reads that are required to support it. In our analysis we will always require at least two or three and at least 25% or 50% supporting reads.

SplazerS Parameter Setup

There are several parameters in SplazerS that can be set. In most cases, the choice of the error rate ϵ and therefore also e_p and e_s is rather straightforward, depending mainly on the error rate and pattern specific to the sequencing technology used and/or on the relatedness of sample and reference genome. The choice of parameters m and δ is a matter of tradeoff between sensitivity and specificity, and practically mainly a matter of runtime.

In the following results we set $\epsilon = 0.05$, and $e_p = e_s = 1$ unless stated otherwise. For the minimum match length m, we use values between 14 bp (anchored) and 23 bp (unanchored). The distance parameter δ will be set to values between $5,000$ and $50,000$.

Indel Comparison

Once we have a predicted set of indel variants, we measure its quality by comparing it to a reference set of variants. For this purpose, we implemented a SeqAn tool called variantComp (see appendix). Comparing a predicted indel with a reference indel is not trivial: indels, especially if located in a tandem repeat, can be placed at quite large distances while still constituting the same basic indel event. For indel rectification, we adopt the computation of the *equivalent indel region*, as defined by Krawitz *et al.* (2010), which accounts for repeats and gives a window of possible locations for each indel. In the following evaluation, predicted indel size is allowed to vary by 10% of reference indel size in all real data sets, but has to be exactly the same as implanted indel size in the simulation experiments.

We will use the measures *sensitivity* and PPV (positive predictive value):

$$Sensitivity = 1 - \frac{FN}{|I_m|} \qquad \text{and} \qquad PPV = \frac{TP}{|I_p|}$$

where true positives (TP) and false negatives (FN) are computed by comparing the set of predicted indels I_p with the set of implanted indels I_m using the indel comparison method of the previous subsection. The *F1*-measure serves as a combined measure:

$$F1 = 2 \cdot \frac{Sensitivity \cdot PPV}{PPV + Sensitivity}$$

4.4.1 Evaluation Datasets

We evaluate on four different datasets of real and simulated reads. As Pindel is SplazerS' main competitor we compare the two tools on two data different sets. Details on program calls are given in the appendix.

76 bp paired-end data from 1000 Genomes Project

We use two sets of Illumina reads for HapMap individual NA12878, available from the 1000 Genomes project page: One containing reads mapped or assigned to chromosome 22 and the other containing unmapped reads that could not be assigned to a chromosome (ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data/NA12878/alignment/NA12878.chrom22.ILLUMINA.bwa.CEU.high_coverage.

20100311.bam and NA12878.unmapped.ILLUMINA.bwa.CEU.high_coverage.20100311.bam). On this data set we compare SplazerS with Pindel and SVseq2.

We extracted a subset[1] of 76 bp reads accounting for $\sim$ 20x coverage and used all 952,401 one-end anchored reads as input for Pindel (version 2.2) and SplazerS in paired-end mode. Additionally, we used a total of 39,479,705 unmapped 76 bp reads as input for SplazerS in single-end mode. For a fair comparison, we use the same cutoff as Pindel and thus require at least 3 indel-supporting reads. We furthermore require the indel-supporting reads to constitute at least 50% of reads spanning the putative indel coordinate. Again similar to Pindel's settings, we set SplazerS' maximum distance parameter such that deletions up to 8 *kb* can be detected. Anchored split-mapping was performed with m = 14, unanchored split-mapping with m = 16. For SVseq2 (version 2.0.1), we use the whole set of reads mapped/assigned to chromosome 22 as input, as in addition to one-end anchored split mapping it also uses mapped read pairs to detect variants. No parameters on the number of supporting reads or variant size can be set, so we run SVseq2 in default mode. We compare the indel prediction results with two reference sets: 1) from the 1000 Genomes project (Durbin *et al.*, 2010) and 2) from a Sanger sequencing study (Mills *et al.*, 2011).

100 bp paired-end data for HapMap individual NA18507

We downloaded a set of 100 bp paired-end reads of chromosome 21 of HapMap individual NA18507, available from the Illumina webpage (http://www.illumina.com/truseq/tru_resources/datasets.ilmn). This file contained a total of 15,069,635 reads of which 904,616 reads are one-end anchored. Again, we use these reads as input for anchored split mapping and indel detection with SplazerS and Pindel. In contrast to the previous read set, this data does not only offer longer reads but also significantly higher quality reads (the average error rate from the alignment of mapped reads is more than three times lower than for the previous set). Also coverage is higher with $\sim$ 30 x. We use the same parameters as in the previous evaluation, but here conduct a second split-mapping step: all one-end anchored reads still unmapped after anchored Hamming distance split mapping are mapped with SplazerS using Levenshtein distance, i.e. allowing additional gaps in prefix and suffix matches. As a reference set, we use a set of indels provided by Illumina using the Casava 1.7 pipeline. For indels larger than 100 bp we check whether our predictions are present in DGV (Iafrate *et al.*, 2004) or dbSNP (Sherry *et al.*, 2001), as no high confidence reference set particularly for this individual is available.

Simulated single-end data

For the simulation of single-end reads (details given in appendix), we first generate a manipulated reference sequence by randomly choosing 1000 known indels from a reference set (dbSNP129 (Sherry *et al.*, 2001) and DGV (Iafrate *et al.*, 2004)) and implanting them into human chromosome 21. Furthermore, we add single base substitutions at a rate of 0.001 to simulate SNPs. We then generate single-end reads from the manipulated chromosome, using the SeqAn tool Mason (Holtgrewe, 2010) for read simulation with typical Illumina sequencing error settings.

[1] More precisely, we extracted 15 lanes: ERR003975-ERR003989 (consecutively numbered)

We repeated this simulation procedure with different read lengths: 100 *bp*, 125 *bp*, and 150 *bp*. Simulated coverages are 5, 10 and 30x. After mapping the set of simulated reads onto the whole human reference genome with "conventional" ungapped mapping with RazerS , we retrieve all unmapped reads for subsequent split mapping. The unmapped reads are split-mapped with SplazerS and, for comparison, with GSNAP (version 2010-07-27) and BWA (version 0.5.8a). For all tools, only one gap of at most length $\delta = 5,000$ was allowed on each read. For BWA we tested the "log-scaled gap penalty for long deletions" feature, but achieved better results with the n-difference mode which we consequently use. SplazerS was tested with different minimum match lengths, we will focus on the results with $m = 16$, but also show results for $m = 18$ and $m = 20$. We then use the same indel detection method for all three tools (snpStore, see chapter 5), as it can detect indels of all size ranges. Only unique best hits are used for indel calling for GSNAP, and only hits with mapping quality > 0 are used for BWA[2]. We set the indel detection method very sensitively: at least two reads and 25% of spanning reads are required to support the indel.

Real single-end data

We used our method in a large-scale targeted resequencing study of 248 male patients with X-linked intellectual disability collected by the EURO-MRX consortium (Kalscheuer *et al.*, tted). X chromosome exons were targeted by solution hybridization selection (SureSelect, Agilent) (Johnston *et al.*, 2010). Illumina sequencing of captured DNA yielded single-end reads of length 76 *bp*. After mapping onto the human reference genome with RazerS using edit distance, all unmapped reads were retrieved for mapping with SplazerS. This constituted a total of almost 1.5 billion unmapped reads (on average ∼6M per patient). With $m = 23$, split mapping was rather strict. Indel detection was not done using SnpStore but with different post-processing scripts (implemented by the author and by Stefan Haas). However, the same basic method was used, requiring that at least 3 reads and 50% of spanning reads support the indel.

4.4.2 Anchored Indel Detection on 1000 Genomes Project Data

The results for comparing indels predicted by SplazerS, Pindel and SVseq2 on anchored and unanchored paired-end reads are summarized in Table 4.2. Since we use previously mapped reads from the 1000 Genomes Project webpage, a large part of the chromosome is already covered by reads mapped also with small indels. Thus, our detected indels do not constitute the whole set of chromosome 22 indels, but rather additional ones discovered through split mapping.

Using anchored reads only, the SplazerS approach yielded a total of 392 indel calls, 301 (76.8%) of which are contained in at least one of the two reference sets (1000G (Durbin *et al.*, 2010) and Mills (Mills *et al.*, 2011)). Pindel called only 209 indels of which 142 (67.9%) are in one of the reference sets. Of the 129 variants predicted by SVseq2 only 14 are annotated (10.85%). SVseq2, using the combined split mapping and read pair approach, is clearly geared towards larger variants. Of all 39 detected indels ≤ 50 *bp* (38 of which are insertions), only one insertion is contained in the

[2]Mapping quality > 0 means unique best hits with respect to BWA's quality-based scoring

		Pindel	SVseq2	SplazerS PE	SplazerS PE+SE
small indels	Deletions	88	0	183	233
	Overlap 1000G	55	0	127	161
	Overlap Mills	56	0	112	142
	Insertions	62	18	105	145
	Overlap 1000G	35	0	58	82
	Overlap Mills	42	0	83	111
medium indels	Deletions	25	1	60	83
	Overlap 1000G	4	0	17	19
	Overlap Mills	8	0	24	32
	Insertions	24	20	28	40
	Overlap 1000G	0	0	0	0
	Overlap Mills	12	1	21	26
large deletions	Deletions	10	44	13	27
	Overlap 1000G	2	9	3	3
	Overlap Mills	0	5	1	1
	SV Deletions	0	21	3	6
	Overlap 1000G	0	3	3	5
	Overlap Mills	0	0	0	0

Table 4.2: Number of detected indels on 1000 Genomes Project data set for NA12878. Pindel and SplazerS PE use anchored reads only, SplazerS PE+SE additionally uses unanchored reads. SVseq2 uses anchored paired end reads for split mapping and read pair information. Small indels are ≤ 10 bp. Medium indels are $> 10\,bp, \leq 50\,bp$. Large deletions are $> 50\,bp, \leq 1000\,bp$. Large SV deletions are $> 1\,kb, \leq 5\,kb$.

reference set, indicating a high false positive rate or possibly but less likely an incomplete/biased reference set.

Pindel's and SplazerS' results are more comparable to each other, showing a similar distribution of predicted variant sizes. The Pindel and SplazerS call sets overlap in 130 indels (62.2% of Pindel call set). Of the 79 indels unique to Pindel, 44 (55.7%) correspond to an indel in the reference set. Of the 262 indels unique to SplazerS, 195 (74.4%) are in the reference set. These results do not only prove a significantly higher sensitivity for SplazerS, but also indicate higher specificity.

Adding SplazerS' unanchored split mapping results, the SplazerS set of called indels increases from 392 to 534 (last column in Table 4.2). Of the additional 142 indels, 90 (63.4%) are contained in one of the reference sets. This indicates a slight decrease in indel calling specificity, but at the gain of much improved sensitivity: more than 30% additional indels are attributed to single-end split mapping.

In summary, SplazerS was able to recover 391 known indels, while Pindel could only recover 142. Our analysis suggests that Pindel's sensitivity is less than 50% of SplazerS' sensitivity on anchored reads only. In the following evaluation, we compare SplazerS and Pindel on another read data set (longer, higher quality reads and higher coverage) and look more closely into the falsely predicted and missed indels.

4.4.3 Anchored Indel Detection on 100 bp Illumina Reads

On the set of paired-end 100 *bp* reads, our indel calling procedure yielded a total of 1481 indel calls of which 1196 (80.76%) overlap with a reference set indel. Pindel predicted a total of 1232 indels with 850 (68.99%) matching a reference variant. Table 4.3 summarizes the results and shows overlap with the Illumina and dbSNP/DGV reference set.

Again, our analyses suggests a much lower sensitivity for Pindel (∼70% of SplazerS' sensitivity). Also Pindel seems to be less precise than SplazerS, with percentages of predictions matching a reference indel being lower for small and medium sized indels. For large deletions, precision is comparable but sensitivity is lower.

Closer inspection of putative false positive (FP) indel calls of Pindel revealed a significant decrease in precision for the short arm of chromosome 21 (21p) as compared to the long arm (21q). Precision for called indels for Pindel is 18.43% in 21p and 79.9% in 21q; precision for SplazerS is 44.4% in 21p and 81.56% in 21q, thus also exhibiting an enrichment of FPs in 21p. 21p is a highly repetitive and polymorphic region (Lyle *et al.*, 2007) which leads to many low confidence split read matches (as indicated by many multiply mapped reads in SplazerS' mapping output). In addition, uniquely mapped split matches are often mutually contradicting, i.e., leading to indel candidates that do not pass the percentage filter criterion (at least 50% supporting reads) which explains a higher precision for SplazerS. Furthermore, 21p is homologous to other regions in the genome, possibly leading to wrongly anchored split reads, which may explain a general enrichment of FPs in 21p.

The results for adding the Levenshtein distance split-mapped reads ("SplazerS+L") are given in the last column of Table 4.3. We observe a small increase (2.6%) in detected indel variants while overlap percentage remains essentially the same. Levenshtein distance mapping also revealed additional indel-contradicting reads: The effect can be seen for medium-sized insertions where the number of predictions decreases by one.

As an example, Figure 4.7 shows two indels that were detected by SplazerS only and that

		Pindel	**SplazerS**	**SplazerS+L**
small	Deletions	533	662	675
	Overlap	375 (70.36%)	538 (81.27%)	550 (81.48%)
	Insertions	532	644	665
	Overlap	423 (79.51%)	580 (90.06%)	597 (89.77%)
medium	Deletions	64	85	87
	Overlap	17 (26.56%)	37 (43.53%)	37 (42.53%)
	Insertions	61	29	28
	Overlap	11 (18.03%)	8 (27.59%)	8 (28.57%)
large	Deletions	42	61	63
	Overlap	24 (57.14%)	33 (54.10%)	34 (53.97%)

Table 4.3: Number of detected indels by the different methods tested. Small indels are $\leq 10\,bp$. Medium indels are $> 10\,bp, \leq 100\,bp$. Large indels are $> 100\,bp$. Small and medium-sized indels were overlapped with an Illumina reference set. Large deletions were overlapped with a set of DGV and dbSNP indels $> 100\,bp$. Percentages in brackets give the fraction of predictions that are contained in the reference set.

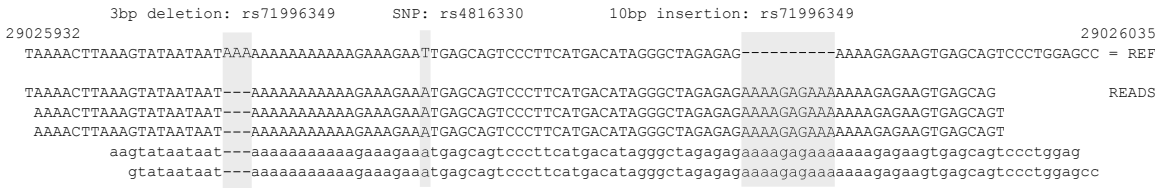

Figure 4.7: Example of a complex variant region: a 3 *bp* deletion, a SNP and a 10 *bp* insertion are colocated in a 65 *bp* window. The shown region is chr21:29025932..29026035. dbSNP accession IDs (rs numbers) are given for each variant. Reads in capital letters are mapped on the forward strand, while reads in small letters are mapped on the reverse strand. Note that the placement of both indels is ambiguous. By convention, SplazerS places the indel to the leftmost position.

		SplazerS		**GSNAP**		**BWA**	
		SN	PPV	SN	PPV	SN	PPV
Ins	10 - 30 bp	99.35	99.33	95.87	98.63	95.21	96.83
	4 - 9 bp	98.29	98.21	97.44	96.62	100.0	76.39
	1 - 3 bp	98.78	99.65	98.60	99.47	98.94	93.31
Del	1 - 3 bp	96.03	99.80	96.00	98.74	96.99	92.30
	4 - 9 bp	94.54	100.0	92.26	100.0	94.54	87.17
	10 - 50 bp	92.73	99.52	85.18	98.12	87.51	93.71
SV Del	51 - 500 bp	70.98	98.38	61.13	89.37	0	NA
	0.5 -1 kb	94.69	100.0	92.47	97.56	0	NA
	1 - 5 kb	100.0	94.44	96.67	90.61	0	NA

Table 4.4: Sensitivity (SN) and PPV results of simulations at 30x coverage with read length 125 *bp*, for different indel size categories. Each category has at least 50 representatives.

were also matched with entries in the reference set. Pindel does not detect these indels, as the 10 *bp* insertion is close to a 3 *bp* deletion (53 *bp* upstream) and a single nucleotide polymorphism (34 *bp* upstream). Both indels and also the T→A SNP are known variants contained in dbSNP. In particular, SplazerS is thus more sensitive and robust in variant-rich regions.

4.4.4 Unanchored Indel Detection on Simulated Data

We now turn to single-end data, i.e. unanchored split mapping. On the simulated read data set, we tested indel prediction accuracy of SplazerS, BWA and GSNAP for varying read lengths (100-150 *bp*) and coverages (5-30x). Figure 4.8 shows the results in terms of sensitivity, PPV and the combined accuracy measure, while Table 4.4 shows the results in terms of sensitivity and PPV for 125 *bp* reads at $\sim$ 30x coverage, dividing indels into different size ranges: three small indel classes for insertions/deletions of $1-3\,bp$, $4-9\,bp$, and $10-50\,bp$ in size; and three large structural variant classes for sizes $51-500\,bp$, $501\,bp-1\,kb$ and $>1\,kb$.

Figure 4.8A shows how sensitivity increases with coverage and with read length. In all settings, SplazerS is the most sensitive and most precise tool. At 30x, it already recovers $>90\%$ of implanted indels on the 100 *bp* reads; for the 150 *bp* reads sensitivity reaches $>95\%$. GSNAP is always a few percentage points behind SplazerS, with SplazerS' lead being more pronounced on longer reads

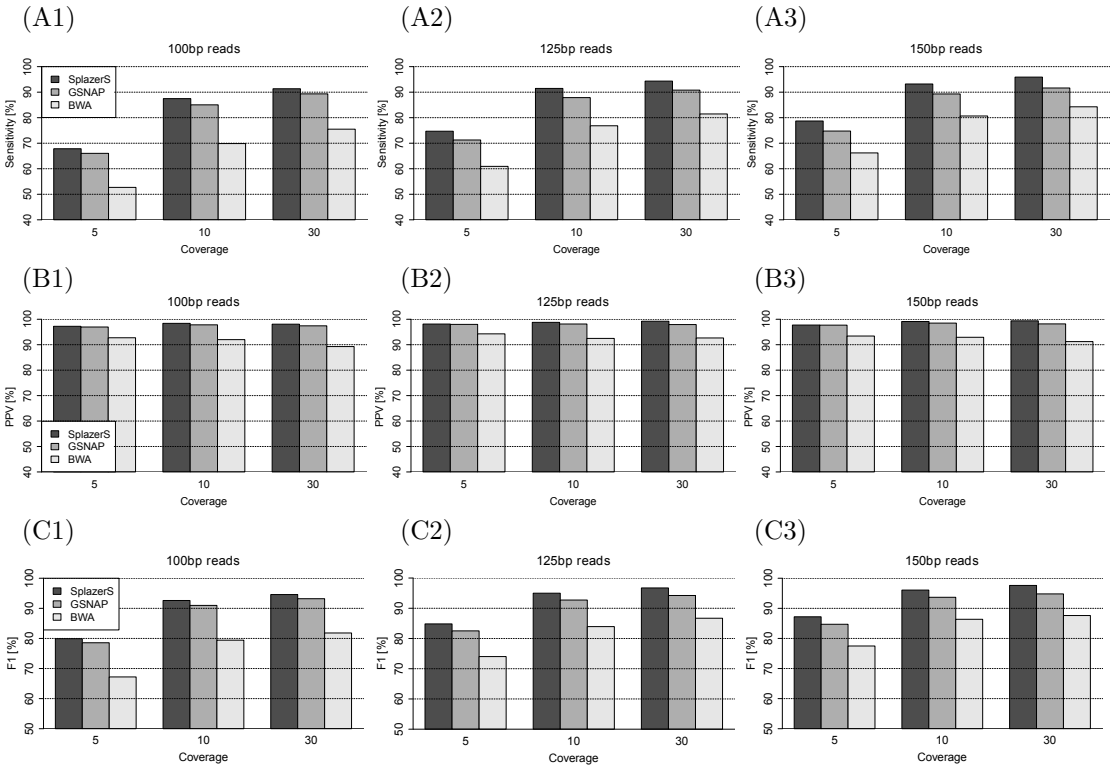

Figure 4.8: Sensitivity, PPV and combined F1-measure indel detection for increasing coverage and increasing read length. (A) Read length 100 *bp*. (B) Read length 125 *bp*. (C) Read length 150 *bp*. Note the different axis scaling.

	SnpStore		**Dindel**		**Dindel-Filtered** (q > 30)	
	Sensitivity	PPV	Sensitivity	PPV	Sensitivity	PPV
SplazerS	95.0%	99.2%	96.4%	82.7%	82.0%	95.5%
GSNAP	90.7%	97.6%	92.4%	57.2%	78.1%	84.9%
BWA	82.5%	89.2%	84.1%	85.9%	69.6%	94.8%

Table 4.5: Sensitivity and PPV when replacing SnpStore with Dindel on simulated data (125 bp reads, 30x). Dindel assigns a quality value to each predicted indel; when filtering out indels with quality lower than 30 (Dindel-Filtered) precision increases significantly; however, this comes at the cost of much decreased sensitivity.

and higher coverage (1.8% lead on 100 *bp* at 5x compared to 4.3% on 150 *bp* at 30x). Due to its exact 12-mer matching approach, GSNAP systematically fails to detect certain indels, even in high coverage data. BWA is less sensitive and less precise than the other tools (Figure 4.8B). Table 4.4 shows that BWA's sensitivity mainly suffers from missed deletions $> 50\,bp$.With increased read length, it can also detect larger indels and thus its overall sensitivity increases.

For indels $< 10\,bp$ BWA is the most sensitive tool, but with the lowest PPV. For indels $\geq 10\,bp$, SplazerS has the highest sensitivity. Furthermore, it maintains the highest PPV in all indel categories. Most notably, SplazerS achieves the highest sensitivities in the SV deletion categories. Both GSNAP and SplazerS exhibit a "temporary" drop in sensitivity for the smallest SV deletion class. The low sensitivity is due to low complexity and repeat sequences where reads are often

		BWA	GSNAP	SplazerS $m = 16$	SplazerS $m = 20$
chr21	index	49.8s	12.2s	-	-
	time	44.4s	49.7s	143.5s	52.9s
	space	154 MB	122 MB	185 MB	1.2 GB
genome	index	94.0m	16.9m	-	-
	time	10.8m	18.2m	193.2m	57.4m
	space	3.7 GB	4.6 GB	3.5 GB	5.6 GB

Table 4.6: Running time and memory measurements for 100,000 simulated 125 *bp* reads. SplazerS runs are shown for different minimum match lengths (m). BWA and GSNAP require an additional preprocessing step for index construction.

	100 bp		125 bp		150 bp	
	Sensitivity	PPV	Sensitivity	PPV	Sensitivity	PPV
$m = 16$	92.80	97.48	94.93	99.06	96.55	99.13
$m = 18$	92.55	97.94	94.70	99.06	96.55	99.23
$m = 20$	91.85	98.03	94.60	99.21	96.35	99.23

Table 4.7: Sensitivity and PPV when varying parameter m on the simulation datasets.

ambiguously mappable or even wrongly mapped without a middle gap. Nevertheless, SplazerS maintains a high PPV and higher sensitivity than GSNAP for these difficult-to-map indels. In order to investigate whether our results are robust with respect to different indel calling programs, we conducted an additional analysis replacing our in-house tool SnpStore with Dindel (Albers *et al.*, 2011). This analysis exhibits similar relative sensitivity results for SplazerS, GSNAP, and BWA, and furthermore demonstrated that SplazerS was again the most accurate tool in terms of sensitivity as well as PPV (see Table 4.5).

Running times and memory

Note that running time is not an issue for anchored split mapping (in the order of a few minutes for all tools tested). For single-end reads that may map anywhere in the reference sequence, however, running time can become prohibitive. Table 4.6 summarizes the running times for the simulated read data set. SplazerS' high sensitivity comes at the price of increased running time compared with index-based heuristic mappers. However, the parameter m can be used to achieve a significant speedup without compromising much sensitivity for high-coverage data sets and longer read lengths (see Table 4.7). The observed memory increase with m is explained by a larger value of q being used for q-gram index construction. Running time disadvantage nearly disappears when mapping onto a smaller reference sequences, as demonstrated by mapping the simulation reads onto chromosome 21 only.

4.4.5 Unanchored Indel Detection on Targeted Resequencing Data

We now turn to the hardest case: real 76 bp single-end reads from targeted exon resequencing. Using SplazerS we predicted on average 67 indels per patient. The overlap of predicted indels with dbSNP and DGV was between 38.89% and 71.79% per patient, with mean overlap of 54.99%. Of the total non-redundant set of indels predicted in at least one patient 39.02% were present in dbSNP or DGV.

Figure 4.9 shows the size distribution of all indels $> 5\,bp$ (the majority of smaller indels were predicted with a different method using edit distance alignments). As expected, the majority of indels is located in non-coding sequences (417 out of 456). Non-coding indels occur mostly in tandem repeat regions in units of 2, 3, or 4. Of the 39 coding indels, 29 (74.35%) are multiples of 3, usually having lesser impact on the protein level as the open reading frame may be maintained.

Large deletions $\geq 100\,bp$ are rather rare (61 in total). On average, 3 large deletions were predicted per patient. Interestingly, we observed a strong variance between patients. Closer inspection revealed that locations of large deletions often cluster on the chromosome. Figure 4.10A visualizes these clusters in 1Mb bins over the X chromosome. Surprisingly, deletions do not only colocalize, but often their boundaries also coincide exactly with annotated exon-intron boundaries. Figure 4.10B shows one such case where 5 predicted deletions span all introns of the PQBP1 gene. These "intron deletions" strongly suggest the presence of a retrocopy of this gene. PCR experiments with several primer pairs confirmed a complete, possibly functional retrocopy of PQBP1 (Vera Kalscheuer). This finding is particularly interesting, as it has been shown previously that mutations in PQBP1 cause X-linked intellectual disability (Kalscheuer *et al.*, 2003; Lenski *et al.*, 2004). Additional (partial) retrocopies were predicted for FAM104B, MSN, MPP1, EIF1AX, RBMX, and OPHN1. In the OPHN1 gene, large deletions spanned 19 introns, causing

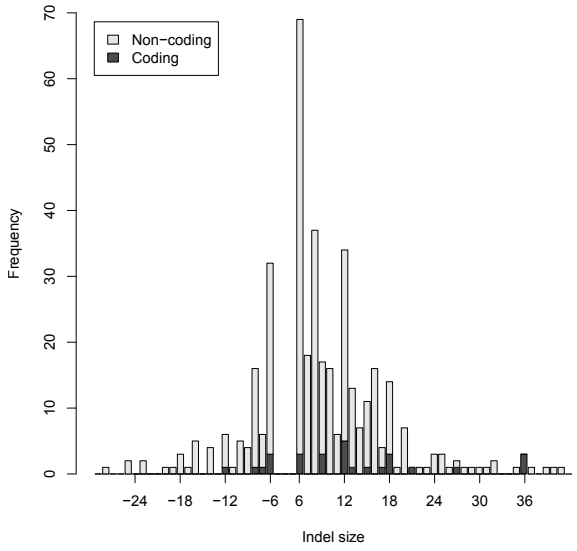

Figure 4.9: Histogram of indels of sizes $> 5\,bp$ and $\leq 40\,bp$. Indels are more abundant in non-coding sequences. The majority of indels in coding sequences are multiples of three, i.e., codon-length.

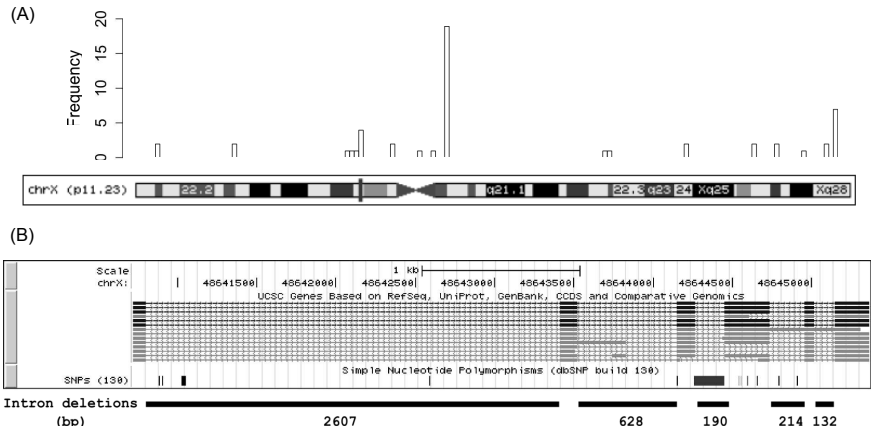

Figure 4.10: (A) Histogram of predicted deletions $\geq 100\,bp$ over their genomic coordinate on chromosome X. Clusters of large deletions are often due to retroposed genes, where spliced introns are missing. (B) A screenshot of the UCSC Genome Browser shows five large deletions that coincide exactly with the introns of the PQBP1 gene. The existence of this complete retrocopy of PQBP1 was confirmed by PCR.

the large peak close to the centromer in Figure 6A.

This finding is consistent with the recent publication of the tool Splitread (Karakoc *et al.*, 2012). In their exome study, they also detect many gene retrocopies (processed pseudogenes) and build a set of retrocopies which they propose to include in the reference sequence set for read mapping.

4.5 Chapter Summary

In this chapter, we

- developed a prefix-based read mapping tool MicroRazerS, based on the same core algorithm as RazerS and specialized on mapping small RNA reads.
- compared MicroRazerS with other read alignment tools on miRNA sequencing data, measuring the number of detected annotated miRNAs.
- developed a split read mapping tool SplazerS, also based on the RazerS core machinery, specialized on prefix-suffix mapping for indel detection, supporting Hamming as well as Levenshtein distance, and applicable to anchored paired-end as well as unanchored single-end reads.
- applied SplazerS to anchored paired-end reads from the 1000 Genomes project, and on newer reads from Illumina, comparing indel detection accuracy with Pindel and SVseq2.
- compared SplazerS with GSNAP and BWA on simulated single-end reads, measuring indel detection accuracy for different read lengths and coverages.
- applied SplazerS to a large-scale targeted resequencing data set of single-end reads from 248 X-linked disability patients.

We observed that

- MicroRazerS had highest miRNA detection sensitivity in our tests and tests conducted by others (Cordero *et al.*, 2012), together with Soap2 and Shrimp.
- in contrast to other tools, mapping small RNA reads with MicroRazerS did not require preprocessing (adapter-trimming) or postprocessing (read match filtering).
- SplazerS was significantly more sensitive and accurate than Pindel on anchored indel detection, especially in repeat and variant-prone genomic regions.
- SVseq2 is geared towards large deletions > 50 bp.
- on simulated single-end data, SplazerS showed highest indel detection accuracy compared with GSNAP and BWA, especially on large deletions.
- while on large genomes SplazerS was slower than the genome-index based tools GSNAP and BWA, on small reference sequences, such as single chromosomes, performance was similar.
- sensitivity can be effectively traded for speed through SplazerS' minimum match length parameter.
- on real single end data indel size distribution was as expected, with coding indels mostly conforming to reading frame and non-coding indels occurring with higher frequency.

- SplazerS discovered several variant/polymorphic gene retrocopies where spliced out introns look like large deletions.
- we tested one retrocopy and confirmed it by PCR, as it is particularly interesting as mutations in its gene of origination (PQBP1) have disease association.

From this we conclude that

- MicroRazerS' prefix-based approach is convenient, sensitive and efficient in its handling of small RNA reads.
- SplazerS is the first published tool (to our knowledge) that can perform anchored as well as unanchored split mapping for indel detection, and achieves highest sensitivity in both modes.
- single end split read mapping is feasible, even with 76 bp reads, and will become more powerful with increasing read length, and with increasing coverage.
- there are surprisingly many retrocopies, as was also discovered by a similar study using paired-end reads conducted around the same time (Karakoc *et al.*, 2012)

Chapter 5

Small Variant Detection

In the previous chapter, we saw how structural variants and indels can be detected using split read mapping. We now turn to smaller indels and single nucleotide variants (SNVs). SNVs and small indels are the most frequent source of genomic variation (Durbin *et al.*, 2010) and have furthermore been shown to account for many diseases (Stenson *et al.*, 2009). Accurate detection and genotyping is therefore of great importance, for diagnostics as well as for our understanding of genome evolution.

In general, small variant detection is based on inspecting columns in the multiple-read-to-reference alignment induced by the mapped reads. One can distinguish between multi- and single-sample variant calling. Here we only consider single-sample data, i.e. data from one individual that is expected to be diploid. Furthermore, one differentiates between variant calling, i.e. classifying a genomic variant as such, and genotyping, i.e. assigning a genotype to a variant (hetero-/homozygous). In our tool SnpStore - like all our tools implemented in the SeqAn library - we address both variant calling and genotyping.

In the following section, we will first point to the difficulties and challenges in small variant calling. Next, we introduce the currently most popular tools in this field (section 5.2), and then go into detail with SnpStore and its main features (section 5.3). Finally, we provide an evaluation of SnpStore in comparison with other tools on a 1000 Genomes Project data set and demonstrate its performance on a large-scale targeted resequencing data set (same data sets as used in section 4.4.2).

5.1 SNV/Indel Calling: Challenges in NGS Data

The main challenge of small variant calling and genotyping lies in distinguishing real variants from sequencing errors. Depending on the sequencing technology, variant callers have to cope with different types of sequencing errors (see section 2.2.1). For example, 454 sequencing is prone to miscalling the number of incorporated nucleotides in homopolymer runs, while in Illumina sequencing the main source of errors is base substitution miscalling. Targeted re-sequencing studies further suffer from preferential capture of the reference allele, leading to biases in allele distri-

Figure 5.1: An excerpt of an alignment of five reads (454 data) to a reference sequence. (A) Multiple sequence alignment induced by mapped reads. (B) Multiple sequence alignment after read realignment.

bution (Turner *et al.*, 2009). Additional biases can be introduced during PCR amplification of sample DNA fragments and through read mapping (Heinrich *et al.*, 2011).

Standard read mapping tools, as those introduced in Chapter 3, align reads independently of each other, i.e. produce pairwise read-to-reference alignments unaware of the multiple read context. Sequence variants may therefore not be consistently aligned among multiple reads covering the same genomic position. In addition, sequencing errors can cause further inconsistency in the alignment of multiple reads. Especially in repeat regions, alignment inconsistencies can complicate variant detection. Homer and Nelson (2010) first showed that read realignment taking into account all reads mapped to a genomic position significantly improves variant detection and genotyping accuracy (Homer and Nelson, 2010). Figure 5.1 shows an example of a multiple read alignment, induced by reads mapped in the standard pairwise fashion (A), and after realignment (B). After realignment, the homozygous deletion is clearly visible, demonstrating the benefit of realignment.

The next section reviews different variant detection methods, placing focus on realignment strategies. For a more detailed review of different SNP/genotype calling methods, the reader is referred to (Nielsen *et al.*, 2011).

5.2 Small Variant Detection Strategies

Variant detection tools scan the multiple read alignment for differences compared to the reference genome. When looking at a specific variant candidate, they either use certain threshold criteria, such as a minimum number of variant-supporting reads, to call a variant/genotype, or they use a probabilistic approach to determine the most likely event. The threshold approach was mainly used in earlier tools, as the probabilistic approach taking into account base quality values was soon proven to be superior. All tools mentioned in the following belong to this latter category employing a probabilistic framework. Other steps common to most variant detection tools include PCR artifact removal (removal of duplicates/pile-up correction), removal of low quality bases through read clipping, and assignment of quality (confidence) scores for post filtering. The most popular variant detection tools to date incorporate a realignment step, which we discuss in more detail in the following.

5.2.1 Variant Detection Tools and Their Realignment Strategies

SRMA (Homer and Nelson, 2010) is one of the first tools implementing read realignment for improved variant detection. The algorithm is based on constructing a *variant graph* incorporating all candidate variants observed in the pairwise read-to-reference alignments. Each read is then realigned to this variant graph and if an alignment superior to the original one (i.e. with better alignment score) exists, the original alignment is replaced with the improved one. As only a small window of the genome needs to be inspected at a time, the variant graph remains small and realignment can be done space-efficiently. The graph is further pruned by only maintaining variants that are observed in a significant fraction or number of original alignments. However, SRMA only performs realignment of reads, but does not implement a variant detection method.

The authors therefore used SRMA in conjunction with **Samtools** (Li, 2011), a suite of tools for analysis and manipulation of SAM files. Samtools does SNV and indel calling, but did not provide realignment functionality in early versions. Later versions do a simple realignment around indels, but the algorithmic strategy is not well documented.

One of the first indel detection tools incorporating realignment is **Dindel** (Albers *et al.*, 2011). Its realignment step takes into consideration combinations of candidate variants thereby creating up to a certain maximum number of *candidate haplotypes*. Reads are realigned to each candidate haplotype and posterior probabilities of each haplotype are calculated to obtain the most probable realignments and haplotype.

Currently, the most popular small variant detection tool with realignment method is probably **GATK**, the Genome Analysis Tool Kit (DePristo *et al.*, 2011). Similar to Dindel, it first constructs candidate haplotypes by considering indels observed in individual read alignments (and optionally known indel sites). Next, gapless realignment of each read to each haplotype is performed and the most likely haplotype is chosen. Reads are then assigned to either the reference haplotype or the alternative haplotype. Apart from realignment, GATK provides other functionality useful for variant detection. Its base quality score recalibration, for example, has been shown to improve variant detection accuracy (DePristo *et al.*, 2011). GATK also performs a read clipping step that discards the lowest quality read suffix with respect to a quality threshold value.

All realignment methods mentioned above are only empowered to detect indels that are present in at least one initial read-to-reference alignment. In order to avoid combinatorial explosion many tools further prune search space by discarding indels with low frequency. The realignment module in our SeqAn tool **SnpStore** aims at being less restrictive by performing essentially a de-novo multiple sequence alignment (MSA) of all reads covering a candidate position, and then realigning the reference sequence to the read MSA. We started developing SnpStore in 2009 when no other established variant calling tool was yet available. Many developments and real-data-relevant heuristics were therefore discovered and implemented simultaneously with their development in other tools. In the following we introduce SnpStore in detail.

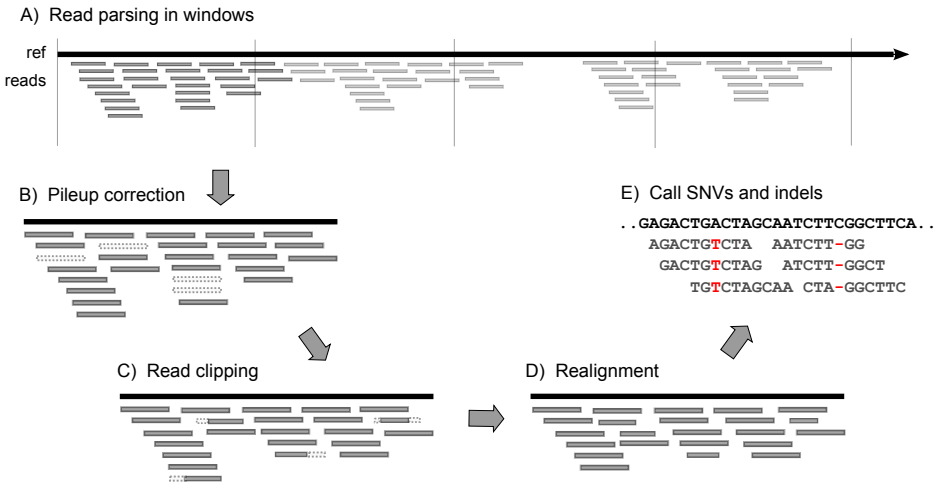

Figure 5.2: Overview of the SnpStore algorithm. A) Reads are parsed window by window. For each window B) pileup correction removes likely amplification artifacts, and C) read clipping removes low-quality bases and likely sequencing errors close to read borders. D) Each separate group of reads is realigned and finally D) SNVs and indels are called.

5.3 SnpStore - Variant Calling Algorithm

As all our developed tools, *SnpStore* is implemented in SeqAn. Most importantly, SnpStore supports SNV and indel detection as well as genotyping. It essentially implements the same strategy as Samtools/Maq for assigning posterior probabilities to genotypes assuming diploid data (section 5.3.2). For indel calling a threshold model is used. Alternatively, the threshold approach can also be used for SNV calling. SnpStore's realignment procedure is based on the Anson Myers' ReAligner (Anson and Myers, 1997) and will be explained in more detail in section 5.3.1. Before realignment and subsequent variant calling, SnpStore undertakes several steps (see also Figure 5.2):

1. Parsing reads in windows

As default, reads are parsed in genomic windows of $10,000\,bp$. This is done to keep a low memory footprint and to avoid large realignment units which would negatively impact runtime. Reads overlapping with window borders or with other reads which overlap with window borders are considered in both windows, in order to avoid realignment border artifacts.

2. Pileup correction

The pileup correction (or duplicate removal) procedure discards stacks of reads most likely resulting from PCR amplification artifacts. Of all reads mapping to the same chromosomal coordinates and strand, only the x highest quality reads are kept, where x is specified by the user and should be set to what is expected from sequencing coverage. Pileup correction is by default done on the merged set of reads from all input files. When multiple read files from separately prepared samples

(e.g. replicates with separate PCR amplification step) are used as input, pileup correction can be done on each file separately.

3. Read clipping

An additional feature of SnpStore is its read clipping routine. Read clipping is applied in order to trim low quality read ends, leading to a cleaner set of reads and hence variant calls. Clipping positions are determined by sliding a window of length w (by default $w = 10\,bp$) over the read starting at the suffix (prefix, respectively). If there is at least one base with quality value lower than a certain threshold value Q_t (by default $Q_t = 10$), the window is slid to the next position. Once all quality values in the current window are larger or equal to Q_t, the procedure is stopped and the suffix (prefix, respectively) outside the current window is trimmed. Optionally, it can be additionally required that there are no alignment errors within m base pairs of the clipping position (default $m = 3$), otherwise clipping is extended to remove the misaligned bases. Clipping positions are not computed within SnpStore, but are currently determined by a Perl script which adds clip tags to the mapped read files (clipping developed and implemented by Stefan Haas). These tags are read and applied by SnpStore.

5.3.1 Realignment

For realignment, SnpStore integrates the Anson-Myers' ReAligner (Anson and Myers, 1997) which was implemented in SeqAn by Tobias Rausch as part of the SeqCons tool for insert assembly (Rausch *et al.*, 2009). For each window of parsed reads, SnpStore treats each group of overlapping reads separately (see Figure 5.2). First, a multiple read alignment maximizing consistency among reads with respect to the implied consensus sequence is computed. In order to be able to call variants with respect to the reference genome, this multiple read alignment is then realigned to the reference sequence.

Notation

We define an alphabet $\Sigma^* = \{A, C, G, T, N, -\}$ of alignment characters or *observations*. Given an alignment column C, we denote the total number of alignment characters in this column as n. For $X \in \Sigma^*$ we further use n_X to refer to the number of times we observe character X. The consensus $\overline{c}$ records the most frequent character of the column. $\overline{c}$ records multiple characters if there are ties.

Anson-Myers' ReAligner

The ReAligner algorithm (Anson and Myers, 1997) expects as input a relatively good MSA that is then refined by a round robin procedure, extracting one sequence at a time and realigning it to the remaining MSA (called a profile). The extracted sequence is realigned in a band around its original alignment, which enforces it to differ only slightly from its original alignment, i.e. the global layout of the MSA is maintained. The scoring function used when realigning a read to a profile in a classical dynamic programming manner is a combination of two scores: the consensus

score δ_c and the fractional score δ_a. When aligning observation X to alignment column C, we compute

$$\delta_c(X, C) = \begin{cases} 0 & \text{if } X \in \overline{c} \text{ or } C \text{ empty} \\ 1 & \text{otherwise} \end{cases} \tag{5.1}$$

and

$$\delta_a(X, C) = \begin{cases} n_X/n & \text{if } n > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

The overall score δ_{c+a} is then a combination of both scores, giving them equal weight:

$$\delta_{a+c}(X, C) = 0.5 \cdot \delta_c(X, C) + 0.5 \cdot \delta_a(X, C) \tag{5.3}$$

This realignment strategy has been shown to perform well for assembly finishing (Anson and Myers, 1997). In contrast to the original scoring scheme which uses linear gap costs, we use affine gap costs which assigns a higher penalty when a gap is opened, i.e. a new column is inserted or the first gap in a column is inserted.

As we ultimately want to call variants with respect to the reference sequence, we finally realign the reference sequence to the multi read MSA.

5.3.2 Variant Calling and Genotyping

Given the initial or realigned read alignment, our objective now is to distinguish real variants from sequencing errors. After introducing the necessary notation, we describe our threshold model which can be used for indel as well as SNV calling and genotyping. Next, we explain the Bayesian SNV genotyping model that was originally developed in Maq (Li *et al.*, 2008a) and that we integrated into SnpStore with some adaptations. Note that while the Maq model assumes diploid data, the threshold model can in principle be applied to multi-sample data. However, we will not evaluate multi-sample data here. Independent of the variant calling model used, a number of heuristics for improvement of variant detection accuracy can be employed in SnpStore, which we explain before finally turning to the evaluation section.

Notation

In addition to the notation of the previous section, we record for each column the quality values associated with each involved alignment character. For each possible alignment character $X \in \Sigma^*$, we maintain a vector $Q^X = \{Q_1^X, ..., Q_{n_X}^X\}$ where Q_i^X with $1 \leq i \leq n_X$ is a Phred scaled quality value (see section 2.2.2). These quality values are either the raw base call quality values or some recalibrated quality values, e.g. possibly taking into account mapping quality of the respective reads. The gap character "$-$" receives the average quality value of its neighboring bases in the respective read. Further, we have the reference sequence character $\tau \in \Sigma^* = \{A, C, G, T, N, -\}$.

Threshold Method for SNV and Indel Calling

The threshold model in SnpStore is the only model offered for indel calling/genotyping. For SNV calling, the alternative probabilistic model will be introduced later. In the threshold model, each alignment column is checked for a number of different criteria. For the current alignment column, we denote the most frequent non-reference character as $\hat{X}$ with $\hat{X} \in \Sigma^* \setminus \{\tau\}$. A variant is called if the following requirements are satisfied:

- the alignment column needs to be covered at least by a certain minimum number of reads $\min_{\text{depth}}$, i.e.
$$n \geq \min_{\text{depth}} \tag{5.4}$$
- $\hat{X}$ needs to be observed at least a certain minimum number of times $\min_{\text{count}}$, i.e.
$$n_{\hat{X}} \geq \min_{\text{count}} \tag{5.5}$$
- $\hat{X}$ needs to be observed on at least a certain minimum fraction of reads $\min_{\text{frac}}$, i.e.
$$\frac{n_{\hat{X}}}{n} \geq \min_{\text{frac}} \tag{5.6}$$
- $\hat{X}$ needs to be observed with a certain minimum quality $\min_{\text{qual}}$, i.e.
$$\frac{\sum_i Q_i^{\hat{X}}}{n_{\hat{X}}} \geq \min_{\text{qual}} \tag{5.7}$$

A called variant is classified as homozygous if the variant is observed with a certain minimum fraction $\min_{\text{gfrac}}$:
$$\frac{n_{\hat{X}}}{n} \geq \min_{\text{gfrac}} \tag{5.8}$$
Otherwise, it is classified as heterozygous.

All threshold parameters can be specified by the user and may be set for indels and SNVs separately. On Illumina data, it makes sense to set thresholds for indels lower than for SNVs, as indel sequencing errors are less common and in general it is harder to correctly align indel-containing reads.

Bayesian Model for Genotyping (Maq Model)

The Bayesian model for SNV genotype calling is more involved and will be described only briefly here. More detailed formulae are given in the appendix. Given the two most frequent bases $\hat{X}$ and $\hat{Y}$ in an alignment column, we are ultimately interested in calculating the posterior probabilities of genotypes $(\hat{X}, \hat{X})$, $(\hat{Y}, \hat{Y})$, and $(\hat{X}, \hat{Y})$, given the observed data D. For each position along the reference sequence, we therefore first determine $\hat{X}$ and $\hat{Y}$. Given we observe base $\hat{X}$ k times and

base $\hat{Y}$ $n-k$ times, we denote the homozygote likelihoods as

$$P(D|(\hat{X},\hat{X})) = \alpha_{n,k} \quad (5.9)$$
$$P(D|(\hat{Y},\hat{Y})) = \alpha_{n,n-k} \quad (5.10)$$

Values $\alpha_{n,k}$ are derived by applying binomial statistics and then correcting for correlation of sequencing errors caused by mapping biases and amplification artifacts (see appendix). Reads mapped to different strands are modeled as being independent of each other, such that the final value $\alpha_{n,k}$ is calculated as the product of α_{n_F,k_F} and α_{n_R,k_R} where the subscripts F and R denote the corresponding value only considering the forward and reverse strand, respectively. Furthermore, the model takes into account individual base error probabilities.

Heterozygotes in Maq are also modeled using binomial distribution, with probability 0.5 of observing one or the other allele:

$$P(D|(\hat{X},\hat{Y})) = \frac{1}{2^n}\binom{n}{k} \quad (5.11)$$

To get to posterior probabilities, a prior needs to be set for each event. In Maq, the prior r for heterozygotes is set to 0.001 (SNP rate in humans) by default, and each homozygote prior is set to $(1-r)/2$. If we set $\hat{r} = 2r/(1-r)$, we arrive at posterior probabilities

$$P((\hat{X},\hat{Y})|D) = \frac{\hat{r}\cdot P(D|(\hat{X},\hat{Y}))}{\hat{r}\cdot P(D|(\hat{X},\hat{Y})) + P(D|(\hat{X},\hat{X})) + P(D|(\hat{Y},\hat{Y}))}$$
$$P((\hat{X},\hat{X})|D) = \frac{P(D|(\hat{X},\hat{X})}{\hat{r}\cdot P(D|(\hat{X},\hat{Y})) + P(D|(\hat{X},\hat{X})) + P(D|(\hat{Y},\hat{Y}))}$$
$$P((\hat{Y},\hat{Y})|D) = \frac{P(D|(\hat{Y},\hat{Y}))}{\hat{r}\cdot P(D|(\hat{X},\hat{Y})) + P(D|(\hat{X},\hat{X})) + P(D|(\hat{Y},\hat{Y}))}$$

The genotype $\hat{g_1}$ maximizing the posterior probability is called with a genotype quality confidence value describing how likely the second best genotype $\hat{g}_2$ is compared to $\hat{g}_1$, i.e. $P(\hat{g}_2|D)/P(\hat{g}_1|D)$ transformed to a Phred quality.

Adapted Maq Model in SnpStore

SnpStore implements the Maq model for genotype calling with some adaptations concerning 1) usage of quality values in computing homozygote probabilities, 2) additional computation of a SNV quality value, and 3) its calculation of heterozygote probabilities.

Firstly, while Maq requires a mapping quality value for each read, SnpStore resorts to raw base call qualities if no mapping qualities are given. For each base the minimum of its base quality and the mean read base quality of the respective read is used. The rationale here is to trust no read base more than the read it is contained in.

Secondly, SnpStore computes SNV quality values similar to genotype quality values by simply measuring how likely the homozygous reference genotype $\hat{g}_R$ is compared to the called genotype

$\hat{g}_1$, i.e. $P(\hat{g}_R|D)/P(\hat{g}_1|D)$, and transforming to a Phred quality value.

Finally, SnpStore offers an alternative approach for modeling heterozygote likelihoods. This model is based on the observation that the PCR amplification process leads to broader variance in observed allele frequency than what is expected from binomial statistics (Heinrich *et al.*, 2011). Heinrich et al. analysed this in exome data, seeing a mean fraction of 0.54 of the reference allele at heterozygote positions. The shift is due to enrichment bias (reference is more likely to be captured than variant allele) and due to read mapping (reads carrying the reference allele are more likely to be mapped). Apart from this shifted mean, they observed that the allele frequency is best described by a branching process. They finally derived an anaytical model, a normal distribution with larger variance, that fits well with empirical data.

In collaboration with the authors (Peter Krawitz, Verena Heinrich and Na Zhu), we adapted the Maq genotype calling model to incorporate this distribution. With reference allele probability $p = 0.54$, the mean reference allele frequency at a position covered by n reads is $\mu = n \cdot p$ and the variance is $\sigma^2 = n \cdot p \cdot (1-p) + \mathrm{Var}_B$. Var_B is the asymptotic variance of a random variable describing the allele distribution after amplification modeled with a branching process (Heinrich *et al.*, 2011). The variance is dependent on the number of amplification cycles K, initial number of fragments of each allele N, and amplification probability q.

$$\mathrm{Var}_B = \frac{2(1+q)^{-1} - 2(1+q)^{-k-1} + (1+q)^{-k} - 1}{8N} \tag{5.12}$$

We can then replace formula 5.11 with

$$P(D|(\hat{X},\hat{Y})) = \int_{k-0.5}^{k+0.5} f(k;\mu,\sigma^2)dk \tag{5.13}$$

$$\tag{5.14}$$

which we can efficiently calculate by using the cumulative distribution function of f describing the normal distribution. As default, we use the realistic values $N = 10$, $q = 0.3$, and $K = 18$ (Krawitz, personal communication), but parameters can be set on the command line.

Additional Filter Criteria for Both Models

A number of additional criteria are checked for each candidate alignment column, independent of whether the threshold or the Bayesian model is used. These criteria include

- the candidate variant needs to be observed on both strands
- a minimum number of different read positions need to be involved in the candidate observation
- additionally these positions can be required to have a minimum distance from read borders

- there may not be more than a certain fraction of noise reads, i.e. reads showing neither candidate variant nor reference character

These criteria were developed through inspection of many real-data cases, where without these heuristics wrong variant calls would be made.

5.4 SnpStore - Results

We evaluate and compare SnpStore with other tools on simulated and real data. In our evaluation we will not be able to investigate all parameters, but will limit the analysis to key parameters, such as whether realignment is performed or read clipping is applied. In the following, we explain the different tool settings we test and then introduce the different data sets we use for evaluation.

Tool Setup

We will compare SnpStore, GATK and Samtools, switching on a number of key features.

For SnpStore, the default mode will be denoted as "SnpStore". When feature X is enabled this will be denoted as "SnpStore+X". The features tested are realignment (R), read clipping (C), variance-corrected heterozygote probability computation (H), and using the threshold model (T) instead of the default probabilistic SNV calling model. For example, "SnpStore+R+C+H" then denotes that SnpStore was run with realignment, read clipping and heterozygote correction. Analogously, for GATK we switch on realignment (R), read clipping (C) and quality value recalibration (Q). Samtools is only run in its recommended default mode (which already employs a simple realignment step).

5.4.1 Evaluation Datasets

Simulation data

We simulated read data from a manipulated chromosome to study how well SnpStore, GATK and Samtools could retrieve the randomly implanted variants. To this end, we again use human chromosome 21 for simulation. We implant indels (1-5bp) and SNVs at a ratio of 1:8 and simulate diploid data with a 2/3 probability for heterozygotes. Indel length, sequence and position as well as SNV position and base are sampled randomly. We then simulate 10 million 100 bp reads ($\sim$ 30 x coverage) from the two generated haplotypes using the SeqAn tool Mason in Illumina default mode (including quality value simulation). These reads are mapped onto the original chromosome 21 using RazerS (in default mode with Levenshtein distance enabled).We use the mapped reads as input for GATK, Samtools and SnpStore. As RazerS does not assign a mapping quality, but mapping qualities are required by GATK and Samtools, we simply assign each uniquely mapped read a mapping quality of 254 (maximum possible value) and each non-uniquely mapped read a mapping quality of 0. For additional comparison, we map the simulated reads with BWA, the mapping tool probably most widely used in conjunction with GATK and Samtools.

Given a variant prediction set P and the reference set of implanted variants S, we then define sensitivity, precision and genotyping precision as:

$$\begin{aligned} \text{Sensitivity} &= 1 - \frac{FN}{|S|} \\ \text{Precision} &= \frac{TP}{|P|} \\ \text{Genotyping precision} &= \frac{TP\text{geno}}{TP} \end{aligned}$$

where TP denotes the number of true positive variant calls, FN the number of false negative variant calls, and TP_{geno} the number of true positive variants called with the correct genotype[1]. The genotype is considered correct if a predicted heterozygous variant is matched with a heterozygous variant in the reference set, or a predicted homozygous variant analogously with a homozygous reference variant. For SNVs, we count all predicted variants as true positives if they are detected at the exact same position and with the exact same base (and genotype for TP_{geno}) as a reference set SNV. For indels, the predicted variant may vary slightly in position ($\pm 5\,bp$) and size (10%) compared to the reference set indel. We furthermore employ the *extended indel region* computation (Krawitz *et al.*, 2010) to correct for larger indel placement uncertainty due to repetitive sequence.

1000 Genomes Data

Next, we evaluate SNV and indel calling and genotyping on a 1000 Genomes data set from whole genome resequencing of HapMap individual NA12878. This is the same data set as used in section 4.4.2, but instead of extracting unmapped reads, we use the whole data set of 38,632,492 reads mapped to chromosome 22. Again, we compare SnpStore with the two popular variant detection tools GATK and Samtools. In order to measure performance we compare our detected variants with the high confidence variants detected and published by the 1000 Genomes Project for NA12878 (daughter of CEU trio). Our NA12878 data set constitutes a good evaluation set, as 1) it contains reads from different Illumina sequencing instruments and with different read lengths and 2) a high confidence variant reference set is available from the 1000 Genomes Project (Durbin *et al.*, 2010). Note, that the 1000 Genomes Project used both GATK and Samtools to compile the SNV reference set, keeping the intersection of the two prediction sets. For indels, they used Dindel, which is very similar to the indel calling algorithm implemented in GATK. The obvious circularities should give GATK and Samtools an advantage over SnpStore. However, they used earlier versions of the tools (and additional filtering and knowledge from Mendelian segregation), and most importantly used a different read data set (including 454 as well as Illumina reads). This read data set was produced from earlier sequencing machines (yielding shorter, exclusively single end reads) and the covered portion of the genome was estimated to be 80% (Durbin *et al.*, 2010). Thus, the reference set is not complete and we expect to see additional variants in the prediction sets. However, it gives us high confidence genotype calls, as calls were made from trio

[1]Note that for SNVs we have $TP + FN = |S|$. For indels this does not have to hold necessarily, as more than one predicted indel may match the same reference set indel.

data where mother and father genotypes help to accurately assign child genotypes. We therefore use this set for sensitivity estimation and genotyping precision only and estimate precision with respect to variants contained in dbSNP (sensitivity, precision and genotyping precision definitions as in previous section).

Targeted Resequencing Data

Here, we use the same targeted resequencing data set of 248 male X-linked intellectual disability patients (Kalscheuer *et al.*, submitted) as described in section 4.4.1. We call variants on the RazerS and SplazerS mapped reads using SnpStore, and further inspect them with respect to functional classes, e.g. synonymous or non-synonymous mutation (see section 2.1.2).

5.4.2 Comparison with Other Tools on Simulation Data

Table 5.1 summarizes the results for calling SNVs on the simulation data, comparing SnpStore, GATK and Samtools. For all tools we show results for the default mode and with realignment switched on where possible. For SnpStore we additionally test the threshold model as an alternative

	Sensitivity (*FN*)	**Precision** (*FP*)	**Genotyping prec.**
SnpStore+T	99.45% (132)	99.17% (199)	99.92%
SnpStore	99.79% (51)	98.31% (411)	99.95%
SnpStore+R	99.78% (52)	99.85% (37)	99.99%
SnpStore+R (BWA)	99.72% (68)	99.86% (34)	99.97%
GATK	99.49% (122)	99.34% (159)	99.93%
GATK+R	99.50% (120)	99.40% (145)	99.91%
GATK+R (BWA)	99.41% (141)	99.46% (130)	99.95%
Samtools	99.75 % (60)	99.89 % (26)	99.98 %
Samtools (BWA)	99.61 % (94)	99.88 % (28)	99.97%

Table 5.1: Sensitivity, precision and genotyping precision results for SNV calling on simulation data set. "SnpStore+T" uses the threshold model for SNV calling, "SnpStore" the default probabilistic model, "+R" means that realignment was performed before SNV calling. The additional "(BWA)" indicates that results are based on Bwa-mapped reads, otherwise RazerS-mapped reads were used.

	Sensitivity (*FN*)	**Precision** (*FP*)	**Genotyping prec.**
SnpStore	62.17% (1135)	39.37% (2,941)	89.11%
SnpStore+R	90.56% (283)	93.69% (183)	97.79%
SnpStore+R (BWA)	86.40% (408)	93.95% (167)	99.30%
GATK	56.07% (1318)	47.20% (1,932)	94.09%
GATK+R	88.60% (342)	85.05% (484)	91.94%
GATK+R (BWA)	87.63% (371)	99.34% (18)	93.33%
Samtools	74.87 % (754)	82.39% (480)	99.42%
Samtools (BWA)	96.93% (92)	97.70%(67)	98.52%

Table 5.2: Sensitivity, precision and genotyping precision results for indel calling on simulation data set. For legend on tool settings see Table 5.1.

to the probabilistic SNV calling model. To test how the choice of read mapper affects results, we show results for each tool's best setting on BWA- instead of RazerS-mapped reads. Table 5.2 compiles the corresponding results for indel calling. Note that there is no additional entry for "SnpStore+T" as indel calling is anyway done with the threshold model.

For SNV calling we see that all tools generally achieve very high sensitivity ($> 99.4\%$). For all tools, using the RazerS mapped reads as input gives higher sensitivity than using BWA mapped reads. However, GATK's and SnpStore's precision increases slightly with BWA mapped reads. The overall most sensitive setting is SnpStore without realignment and using RazerS mapped reads. But here we also see the significantly lowest precision (98.31% while all other precision values are $> 99\%$). When switching on realignment in SnpStore, SNV calling sensitivity essentially stays the same (losing only one true positive SNV call), while the number of false positive calls drops by a factor of more than 10 (37 compared to 411 false positives). The gain through realignment is also seen for GATK, but less drastically than for SnpStore. Using RazerS mapped reads and SnpStore with realignment or Samtools seem to be the best choices for SNV calling, where SnpStore is slightly more sensitive and Samtools slightly more precise. Interestingly, GATK shows lowest sensitivity and lowest precision. All tools perform well in genotyping (with genotyping precisions $> 99.9\%$). Genotyping on this simulation data set is easy because reads were sampled uniformly and with random errors, and no sequencing biases or amplification artifacts complicate genotyping.

For indel calling, we see much lower sensitivity and precision values than for SNV calling. For both GATK and SnpStore we see the realigned versions perform significantly better than the unrealigned ones. While Samtools with BWA mapped reads is the most sensitive combination, GATK with BWA is the most precise one. For all tools, precision is higher when using the BWA mappings as input. However, both GATK and SnpStore are more sensitive on RazerS than BWA mappings. Genotyping is also harder than for SNVs, indicating that indel-containing reads are harder to map and align correctly.

While unidentified SNVs are generally due to repetitive regions that make unique mapping difficult or impossible, unidentified indels are in addition due to mapping biases. In general, SNV identification is much more robust, while indel identification is quite vulnerable to the used mapping tools and the indel calling tool's setting.

Now we turn to real data, where we expect to see more difficulties, especially on indels as they tend to be in repetitive region.

5.4.3 Comparison with Other Tools on 1000 Genomes Data

On the 1000G data, we show results for SnpStore in five different modes. In addition to the ones tested previously, we test two more features that are mainly of relevance on real read data: read clipping (C) and variance-corrected heterozygote probability computation (H). GATK and Samtools were run in the same modes as before, but GATK additionally with quality recalibration (Q).

Table 5.3 compiles the results for SNV, Table 5.4 for indel calling. There are fewer rows in the indel calling table, as the heterozygote correction (+H) only concerns SNV calling and again there

	$\|P\|$	**Sensitivity** (FN)	**Precision** (FP_{dbSNP})	**Genotyping prec.**
SnpStore+T	46,768	97.67% (766)	92.43% (3,541)	99.39%
SnpStore	48,255	98.53% (485)	93.16% (3,303)	99.42%
SnpStore+R	47,673	98.51% (491)	93.81% (2,952)	99.36%
SnpStore+R+C	46,336	98.37% (538)	94.90% (2,363)	99.29%
SnpStore+R+C+H	46,931	98.45% (509)	94.33% (2,659)	99.32%
GATK	48,763	98.62% (456)	94.45% (2,704)	99.56%
GATK+R	48,139	98.57% (472)	95.09% (2,364)	99.55%
GATK+R+Q	48,303	98.61% (458)	94.97% (2,429)	99.55%
Samtools	44,260	97.31% (885)	97.64% (1,043)	99.53%

Table 5.3: SNV calling results on NA12878 data set.

	$\|P\|$	**Sensitivity** (FN)	**Precision** (FP_{dbSNP})	**Genotyping prec.**
SnpStore	4,898	64.17% (1,433)	70.25 % (1,457)	81.83%
SnpStore+R	4,194	71.24% (1,150)	80.04% (837)	92.74%
SnpStore+R+C	3,898	69.57% (1,217)	83.50% (643)	92.60%
GATK	4,031	67.17% (1,313)	90.03% (402)	97.06%
GATK+R	4,928	75.52% (979)	88.84% (550)	96.52%
GATK+R+Q	4,934	75.52% (979)	88.79% (553)	96.53%
Samtools	4,030	71.52% (1,139)	88.96% (445)	95.73%

Table 5.4: Small indel calling results on NA12878 data set.

is only one calling model for indel calling (threshold-based) as opposed to the two alternatives in SNV calling (default probabilistic or threshold-based). For SNV calling, we see high sensitivity ($> 97\%$) and precision ($> 92\%$) for all tools and settings. For indel calling on the other hand we see significantly lower sensitivities (for all tools $> 20\%$ lower sensitivity than for SNVs) and also considerably lower precision (between ~ 4 and 20% lower). Also genotyping is more precise for SNVs ($> 99.3\%$ for all tools) than for indels ($\sim$92-97% with exception of SnpStore without realignment).

For both SnpStore and GATK there is a significant increase in indel calling sensitivity ($> 7\%$) when realignment is switched on. This comes with a slight decrease ($< 0.06\%$) in sensitivity for SNVs but also a more significant increase in SNV calling precision ($> 0.6\%$). Interestingly, the total number of predicted indels increases with realignment for GATK but decreases for SnpStore. SnpStore seems to handle the BWA-mapped reads less accurately than GATK and produces many spurious indel calls as well as multiple calls for the same indel (called at slightly different positions due to inconsistency in the unrealigned reads).

As expected, SnpStore's default probabilistic SNV calling model is clearly superior to the threshold approach. Inspecting the other features tested, we see that SnpStore's read clipping is very effective for increasing precision for both SNVs and indels. At the same time sensitivity is decreased especially for indels. This implies that clipping discards read bases spanning or reaching into indels that are aligned as mismatches but carry potential for identifying indel variants. We also switched on GATK's read clipping procedure, but as BWA performs essentially the exact

same clipping procedure as GATK, this had no effect on variant calling results (not shown in tables). Also, quality recalibration did not exhibit a great influence on variant calling outcome, which is explained by the fact that 1000 Genomes Project reads available for download already come quality-recalibrated. Still, switching on quality recalibration has a small influence on GATK's results: we see a slight increase in SNV sensitivity at the cost of a slight decrease in precision. Finally, SnpStore's heterozygote correction effectively adds more false positive than true positive SNV calls. Also, the increase in genotyping precision is only marginal.

All in all, SnpStore is the least precise tool for genotyping, especially for indels where no sophisticated probabilistic model is implemented. When taking into consideration the nature of the reference set (for SNVs computed by GATK and Samtools, and for indels computed by a method similar to GATK), it is surprising that SnpStore achieves higher SNV calling sensitivities than Samtools. However, precision is higher for Samtools pointing to Samtools' default being stricter than that of SnpStore. Overall, GATK is the most sensitive method for both SNVs and indels, not only with respect to the reference sets, but also considering total number of predicted variants.

The Venn diagrams in Figure 5.3 show how the call sets of the three tools (SnpStore+R+C, GATK+R, Samtools) overlap. There is a much larger overlap between all three tools on SNVs than on indels. Samtools and SnpStore show the lowest overlaps with each other while GATK shares high overlap with both other tools. In SNV calling, SnpStore has the highest number of calls unique to itself, whereas in indel calling it has the lowest number. Again, this indicates that SnpStore is stronger in SNV calling where a probabilistic model is used than on indel calling where the simple threshold model is used. All in all, however, there is surprisingly much discordance on indels between the three tools showing that there is room for method improvement for indel detection.

In Figure 5.4, we see the tools' behavior at different read depths (SnpStore+R+C, GATK+R, Samtools). SNV calling reaches full sensitivity at read depth 20-30 and remains fully sensitive for

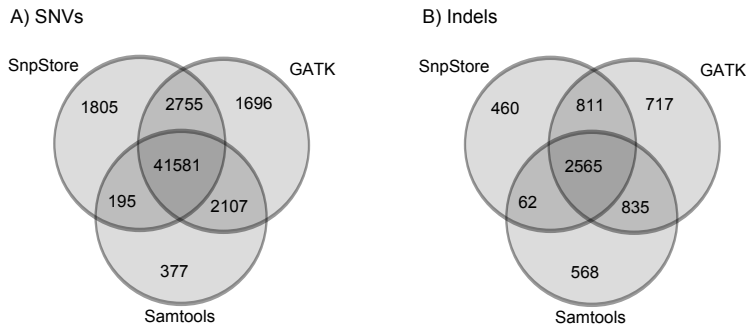

Figure 5.3: Venn diagrams comparing SNV (A) and indel (B) calling results of SnpStore (SnpStore+R+C), GATK (GATK+R) and Samtools.

higher depths, apart from some outliers where the total number of calls is low. Samtools reaches its peak sensitivity a bit slower than GATK and SnpStore. Indels are obviously harder to identify, with sensitivity increasing much slower than for SNVs and read depths of more than 50 required for full sensitivity. Even at high read depth, we see more outliers that show low sensitivity.

SNV calling precision is highest at depths 20-60 and decreases/destabilizes at higher depths, with Samtools consistently maintaining the highest precision. While real SNVs are unlikely to be missed at high read depth, also more artifacts accumulate (such as amplification artifacts and mapping artifacts due to repetitive regions), leading to decreased precision. The lower precision may also be due to a lesser power to ascertain SNVs in ambiguous/repetitive regions, i.e. true variants may simply be missing in the reference set. Especially, this may also be the case for indels where there is generally a lower level of completeness in the reference set. For indels, we see lower precision, with highest confidence for indels called at read depths 30-50.

SNV genotyping can be done with very high confidence for read depths larger than 20. For indels, this confidence is only achieved at depths larger than 50, again indicating the stronger mapping bias for the reference allele than for SNVs.

Mostly, differences between the tools are explained by their behavior in low coverage regions (< 15). Also, very high coverage regions give rise to differences (as seen in precision plots), but low coverage accounts for a much larger fraction of the dataset, giving rise to most false positives and false negatives.

Running times

All running times are roughly in the same order of magnitude. SnpStore's runtime is around 22 min without realignment, and increases by a factor of ~ 3 when doing realignment (71 min). GATK requires approximately 15 min, and an additional 20 min for realignment and 14 min for recalibration. Samtools takes about 28min.

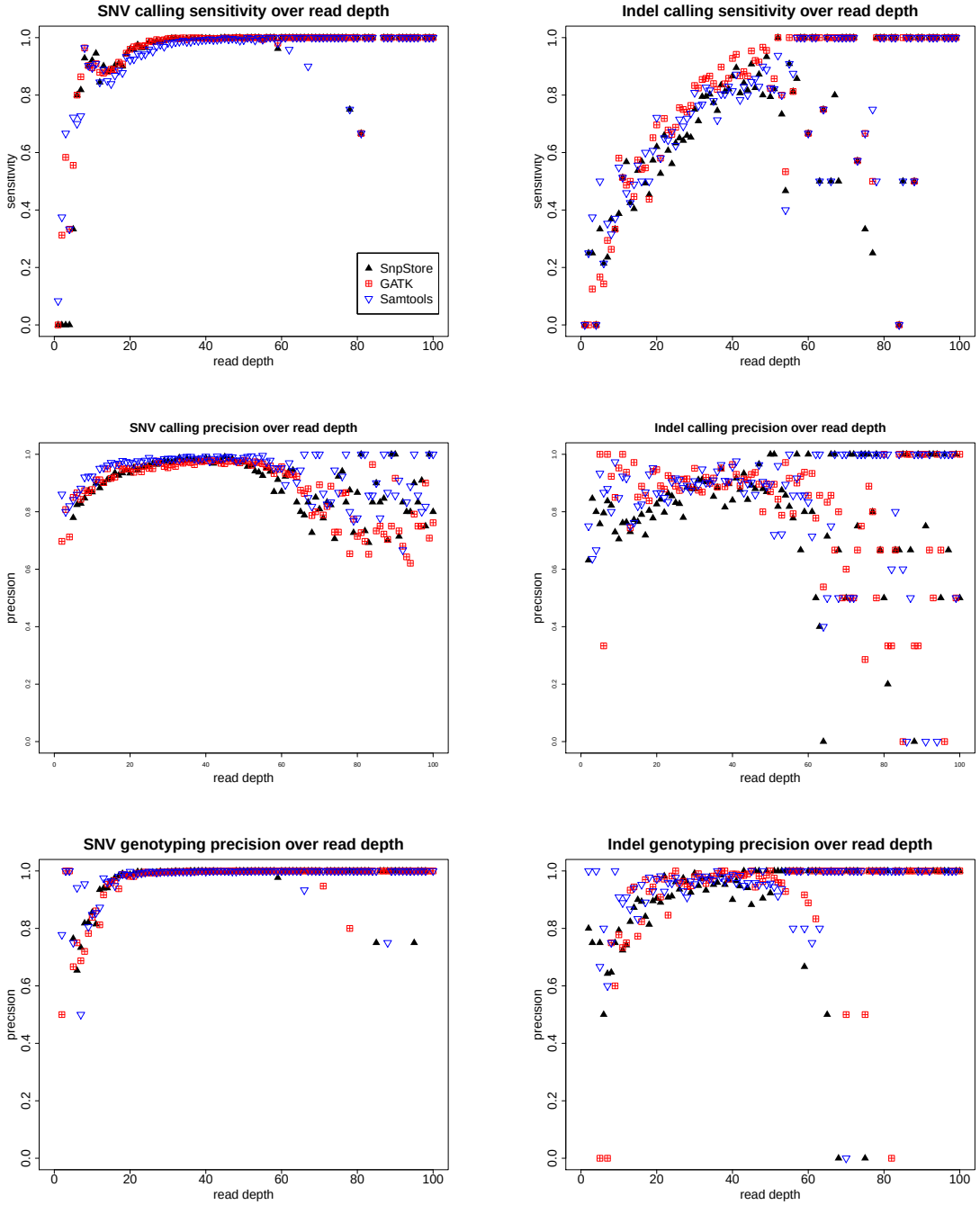

Figure 5.4: Precision and sensitivity of the three tested SNV/indel calling methods over read depth at calling positions. Results are for SnpStore with realignment and read clipping (SnpStore+R+C), GATK with realignment (GATK+R), and Samtools default (Samtools).

	$\|P\|$	overlap dbSNP	novel and unique to one individual
Non-coding	4605	2795 (60.69%)	1468 (31.88%)
Synonymous	1108	864 (77.98%)	235 (21.21%)
Non-synonymous	1570	1018 (64.84%)	531 (33.82%)
Missense	1438	962 (66.90%)	461 (32.06%)
Nonsense	23	10 (43.48%)	13 (56.52%)
In-frame indels	36	17 (47.22%)	18 (50.00%)
Frameshift indels	51	19 (37.25%)	29 (56.86%)
Splice-sites	22	10 (45.45%)	10 (45.45%)

Table 5.5: Small variant calling results on targeted resequencing data, split into functional categories.

5.4.4 Application to Targeted Resequencing Data

On the large-scale targeted resequencing data, SnpStore identified a total of 7283 distinct variants within target regions. Of this non-redundant set of variants called in at least one patient 35.6 % are novel (not contained in dbSNP), whereas in the redundant set (or per patient) we find on average only 17.4% novel variants. This is due to the fact that variants that are recurrent in the patient cohort are more likely to be in dbSNP, whereas variants unique to one patient are more likely to be novel and hence overrepresented in the non-redundant set. Table 5.5 separates the set of variants into functional categories (based on Table 2 from Kalscheuer *et al.* (manuscript submitted)). The majority of variants is in non-coding region. Focusing on variants in coding sequence only, we can distinguish between synonymous variants that do not lead to a change in amino acid and non-synonymous that (presumably) alter the amino acid sequence. Non-synonymous variants are less likely to be contained in dbSNP than synonymous ones. While the 1000 Genomes Project found an approximate 1:0.8 ratio of non-synonymous to synonymous variants in exon capture experiments (Durbin *et al.*, 2010), we see non-synonymous variants with a higher frequency in our data set (1:0.7 ratio). However, as we are looking at X chromosome exons from X-linked intellectual disability patients, we expect this ratio to be higher. In total, several novel candidate genes for X-linked intellectual disability were identified (Kalscheuer *et al.*, submitted).

5.5 Chapter summary

In this chapter, we

- developed a small variant detection tool SnpStore incorporating a read realignment step.
- introduced additional features of SnpStore such as read clipping, heterozygote variance correction in the Maq-like genotype probability calculation, and several filter criteria inspired by inspection of real data.
- applied SnpStore to simulated as well as real 1000 Genomes Project data and compared with the two popular variant detection tools Samtools and GATK
- applied SnpStore to large-scale targeted resequencing data of 248 X-linked intellectual disability patients.

We observed that

- SnpStore showed performance similar to GATK and Samtools.
- read realignment significantly increased the number of correctly called indels and reduced false positive SNV calls.
- SnpStore's read clipping effectively reduced the number of false positives for SNVs as well as indels, but also decreased sensitivity.
- variance corrected heterozygote probability calculation slightly improved sensitivity and genotyping accuracy, but also increased the number of false positive SNV calls significantly.
- on simulated data, SnpStore with RazerS mapped reads was the most sensitive tool on SNVs, while Samtools with BWA mapped reads was most sensitive and precise on indels.
- GATK showed highest sensitivity on 1000 Genomes Project data, but lowest sensitivity on simulated data.
- SNV calling was most accurate at read depths 20 to 60, while accurate indel calling required read depths > 50.
- the tested tools exhibited high agreement (overlap) on SNV calls but low agreement on indel calls.

From this we conclude that

- we developed a generic small variant calling tool that successfully employs the Myers' realigner algorithm on large-scale NGS data and offers several real-data relevant heuristics.
- especially when considering expected biases in the 1000 Genomes Project reference set and dbSNP towards GATK and Samtools, SnpStore is highly sensitive particularly for SNVs.
- the choice of read mapper plays a key role, especially in indel detection accuracy.

- there is still much room for improvement in indel detection, as indicated by high discordance in indel call sets.

Chapter 6

Discussion and Conclusions

Next generation sequencing has truly lead to a revolution in genomics research. Not only is this visible from the large number of scientific publications (currently Illumina lists 3360 and Roche 2380 peer-reviewed sequencing publications on their websites), but also from the broad spectrum of topics addressed ranging from genome resequencing (Bentley *et al.*, 2008; Wheeler *et al.*, 2008; Durbin *et al.*, 2010) to transcriptomics (Mortazavi *et al.*, 2008; Sultan *et al.*, 2008b) or metagenomics (Qin *et al.*, 2010; Gilbert and Dupont, 2011), and from fundamental research to diagnostics (Voelkerding *et al.*, 2009). Furthermore, the crucial role that computational analysis plays becomes obvious when looking at the multitude of software tools that are publicly available. For example, a very active NGS-centered forum (SEQanswers, 2012) currently lists 578 different tools for NGS data analysis, and this list is likely incomplete. The length of the list owes in part to the variety of NGS applications but also to the fact that sequencing technologies are advancing rapidly and thus new tools are continuously developed. However, this also points to the importance of methods that remain applicable with changes in sequencing characteristics. For example, increasing read length and changing error rates are factors that can be taken into account early on in method development.

In this work, we addressed a broad range of key computational aspects of resequencing applications, where a reference genome sequence is known and heavily used for interpretation of the newly sequenced sample. We developed tools for read mapping (Weese *et al.*, 2009) and benchmarking (Holtgrewe *et al.*, 2011), for partial read mapping of small RNA reads (Emde *et al.*, 2010) and for structural variant/indel detection (Emde *et al.*, 2012), and finally tools for detecting and genotyping SNVs and short indels. All methods are generic in that they can handle arbitrary read lengths and variable error rates. Furthermore, they are implemented in the SeqAn library (Döring *et al.*, 2008) making them open-source, easily available and potentially adaptable for the bioinformatics community.

Concerning semiglobal read mapping, we saw that the many published tools can be roughly divided into two generations. The earlier generation mainly makes use of the two-seed pigeonhole principle, as pioneered in Eland (Cox, unpublished) and Maq (Li *et al.*, 2008a), and usually uses an index built on the read sequences. Later tools usually build an index of the genome

(most commonly the space-efficient FM-index) and use index searching strategies to map the reads (Langmead *et al.*, 2009; Li and Durbin, 2009). While this later generation of tools is computationally very efficient, it is, however, more likely to output best matches only and is thus not as robust in the face of mapping ambiguity. Our SeqAn tool RazerS, as introduced in Chapter 3, uses a quite different strategy than most other tools and is based on a q-gram counting filter. Its novelty lies in its fine scale sensitivity switch which allows to set a lower bound on the desired mapping sensitivity. Most other read mappers use heuristics to speed up alignment and cannot guarantee 100 % sensitivity, at least not for many different error rates, read lengths and Hamming as well as Levenshtein distance. That most read mappers are not fully sensitive can be seen by comparing read mapping results to a gold standard, as we did with our benchmarking method Rabema in Chapter 3. Benchmarking turned out to be difficult, not only because tools have a variety of different parameters but also because the definition of match equivalence is not trivial. This difficulty has also been realized by others (Fonseca *et al.*, 2012). Our benchmarking results showed that the currently most popular read mappers are not fully sensitive and especially loose sensitivity for matches with more than 2 errors. Depending on the application, this can have potentially harmful effects in downstream analyses. For example, multi-read assignment methods (Hashimoto *et al.*, 2009; Ji *et al.*, 2011), especially important for transcript quantification in RNA-seq (Wang *et al.*, 2009b), rely on a complete set of possible mapping positions in order to assign multi-reads to their most likely location. In variant detection, wrongly mapped multi-reads can lead to wrong variant calls. The problem of mapping ambiguity becomes less with longer reads; in other words, as sequencing technologies advance and reads become longer, they become easier to assign. However, ambiguity in the human genome remains considerably high even with read lengths $> 100\,bp$ (Whiteford *et al.*, 2005). Also, while sequencing error rates tend to decrease with sequencing chemistry improvements, longer reads will invariably have to be mapped with more than 2 errors. Thus many currently still popular tools will not scale well with technology advances. On the other hand, RazerS should remain sensitive and especially in its newer, parallelized version efficient enough to be applied to the increasing amounts of read data (Weese *et al.*, 2012). Current limitations of the benchmark and also of RazerS are a lack of support for quality-value based mapping. However, Weese *et al.* (2012) show that quality values do not necessarily improve read placement, but may even decrease mapping accuracy of variant-spanning reads.

Increasing read lengths also make partial read mapping more powerful. In chapter 4 we saw two different applications of partial read mapping. The first one, based on read prefix mapping, is specifically designed for small RNA read mapping. As one of the first high-performance small RNA mappers at the time, our SeqAn tool MicroRazerS achieved highest sensitivities compared to other tools. Also in a more recent study involving different miRNA-specialized tools (Cordero *et al.*, 2012), MicroRazerS was shown to be the most sensitive and computationally efficient small RNA mapping tool. The prefix-based strategy is apparently an excellent approach to small RNA mapping, as it avoids errors made during adapter trimming that can prevent reads from mapping with their whole length. However, partial read mapping is especially interesting in SV detection, where reads spanning variants can only be mapped if split into two (or more) local match parts that map to the breakpoint-flanking sequences. Split mapping of these breakpoint-spanning reads

therefore carries the potential to identify SVs with base pair precision. In this work, we developed a strategy for split read indel detection that, in contrast to other methods, works not only on paired-end, but also on single-end read data. On paired-end read data, our split-mapping tool SplazerS significantly outperformed the state-of-the-art tool Pindel (Ye *et al.*, 2009) in variant detection sensitivity. In our evaluation, we also included the recently published tool SVseq2 (Zhang *et al.*, 2012) which uses a combination of two SV detection strategies: first discordant paired-end reads are used to identify SV candidates and then split mapping is performed in candidate regions. Our results indicate that this combination of read pair and split read approach works well for large deletions, but is unsuitable for small to medium-sized indels ($< 50\,bp$); presumably because read pairs are not classified as discordant. For these variants, the split read approach alone thus remains more powerful. Possibly a different way of combining the split-read approach with a different SV detection strategy can lead to a broader range of detectable variants and improved accuracy, but this remains for future work.

An advantage of SplazerS over other tools is its versatility. To our knowledge, SplazerS is still the only split mapping tool to be designed specifically for large deletion detection in single-end read data. While computational resource requirements are greater on single-end reads, as the whole genome and not just a target region needs to be searched, SplazerS still remains applicable to large-scale data sets. On the large-scale targeted-resequencing single-end read data of X-linked intellectual disability patients, we observed an interesting type of variation quite rarely described in humans: gene retroposition. Of particular interest is a retrocopy of PQBP1, a gene previously linked to intellectual disability, which if functional could have a dosage effect. In any case, the discovery of several retrocopies, some of which seem to be polymorphic in humans, was surprising and was also published by Karakoc *et al.* (2012) around the same time as SplazerS.

Used for indel calling on SplazerS' split reads previously, we finally evaluated our SNV and indel calling tool SnpStore. As shown by Homer and Nelson (2010), performing a read realignment step before variant calling leads to a significant improvement in calling accuracy. We can clearly confirm this for SnpStore's realignment feature. Additionally, SnpStore incorporates a number of heuristics that were developed in close inspection of real sequencing data. For example, read clipping turned out to be an effective real-data heuristic for reducing the number of false positive variant calls. Compared to other variant callers (DePristo *et al.*, 2011; Li, 2011), that were mostly developed as part of the 1000 Genomes Project (Durbin *et al.*, 2010), SnpStore performs similarly and is even among the most sensitive ones for SNV calling as observed on real as well as simulated data. In this work, we evaluated variant calling tools by using a 1000 Genomes Project data set and overlapping with 1000 Genomes Project calls and the dbSNP variant database (Sherry *et al.*, 2001). This way of evaluation is biased, as 1000 Genomes Project variant call sets are contained in the most recent versions of dbSNP and these calls were generated by tools that were part of the evaluation. A better way of evaluating would be to calculate agreement with SNP array variant calls. However, first of all, SNP array data needs to be available, and secondly, this strategy only works well for SNVs and not for indels. And especially indel detection seems to require more attention: While SNV calling results are quite similar between different tools, indel results are astonishingly dissimilar (even with the same mapped read input). This indicates that further

improvement in indel variant detection is necessary.

In summary, the methods developed in the course of this work are valuable to advance computational analysis of NGS data. They are real-world applicable and are part of the SeqAn C++ library and therefore have to adhere to certain coding standards. Their development took place simultaneously with other groups developing similar methods, as NGS data analysis has been a quickly advancing field. In fact, MicroRazerS, SplazerS and SnpStore were developed out of direct necessity, because there were no other established methods available at the time. In all our methods, we placed key emphasis on sensitivity and in our evaluations we saw that compared to other tools, our methods usually excel in this realm. Our methods furthermore aim at being compatible with different sequencing technologies and different reads lengths, hence aiding in the compatibility with sequencing technology advances. However, with coming technological advances (Schadt *et al.*, 2010; Branton *et al.*, 2008; Rothberg *et al.*, 2011) we expect novel computational approaches to emerge. Quite likely, assembly-based methods (Li *et al.*, 2011; Simpson and Durbin, 2012) that can reconstruct novel or difficult parts of a sample genome without the help of a reference sequence will receive more and more attention, especially with increased fragment insert sizes becoming available for read pair sequencing. In addition, we anticipate that methods for comparing multiple genomes (Darling *et al.*, 2010; Angiuoli and Salzberg, 2011), that can then directly compare (de-novo) assembled genomes, will gain importance. Further in the future, integrative approaches combining diverse sources of information, e.g. genomic, epigenetic, and transcriptomic, in a common framework promise a more sophisticated, unified view of genomics (Hawkins *et al.*, 2010).

The wide range of high throughput sequencing applications has already enabled significant insights into fundamental genome research (Johnson *et al.*, 2007; Green *et al.*, 2010; Durbin *et al.*, 2010) and especially disease diagnostics (Ng *et al.*, 2010b; Johnston *et al.*, 2010; Voelkerding *et al.*, 2009). With sequencing costs decreasing, we can anticipate further great advancements, particularly in the active field of cancer genomics (International Cancer Genome Consortium *et al.*, 2010), that will truly fuel the shift toward personalized medicine.

6.1 Outlook

Building on the SplazerS split mapping tool, we are currently working on a new approach to multi split mapping for SV detection and for RNA-Seq read mapping. First evaluation of this method in the scope of a master thesis by Kathrin Trappe show promising results (Trappe, 2012). The proposed method uses a different, more generic approach to detect split matches and will have much wider application, e.g. also for alignment of assembled contigs. First, local matches of reads (or contigs) are computed using STELLAR (Kehr *et al.*, 2011), a fully sensitive local aligner with respect to Levenshtein distance. Next, we construct a read-centric compatibility graph for each read. In this graph, local matches are nodes and edges between nodes represent breakpoints. Exact breakpoints are computed with the dynamic programming method described in section 4.3.2. By traversing the graph from artificial start to end node, we obtain chains of local matches, i.e. (multi) split matches. Edges carry weights that represent penalties for chaining two adjacent matches. For example, if two adjacent nodes, i.e. two compatible matches according to read coordinates, map

onto two different chromosomes they will receive a higher penalty than two collinearly mapped matches. Furthermore, if many reads support the same breakpoint, corresponding edges will be penalized less than unsupported edges. The main difficulties in this approach concern the handling of repeat regions and designing a sensible scoring scheme for edge weights.

Appendix

Expected Number of Random Split Matches

The following formulae were derived together with Marcel Schulz. Given a set of reads R containing reads r all of the same length $|r|$. At a given position in a random DNA sequence s, the probability of a split read match with minimum match length m, maximum number of prefix errors e_p, maximum number of suffix errors e_s, and maximum number of total errors $k = \max(\lfloor \epsilon \cdot |r| \rfloor, e_s + e_p)$, can be calculated as

$$p(|r|, m, e_p, e_s, k) = \sum_{i_1=0}^{e_p} \sum_{i_2=0}^{e_s} \sum_{i=0}^{k-i_1-i_2} \binom{m}{i_1}\binom{m}{i_2}\binom{|r|-2m}{i}\left(\tfrac{3}{4}\right)^{i+i_1+i_2}\left(\tfrac{1}{4}\right)^{|r|-(i+i_1+i_2)} \tag{6.1}$$

Equation (6.1) assumes identical frequency=$\frac{1}{4}$ for each DNA character. We denote the number of possible breakpoint positions on the read as

$$cuts(|r|, m) = |r| - 2m + 1 \tag{6.2}$$

In the following, we calculate the expected number of deletion-indicating matches E_{del} and the expected number of insertion-indicating matches E_{ins} separately. Both E_{ins} and E_{del} are reported to the user when running SplazerS in verbose mode.

Deletions:

To compute the expected number of random matches indicating a deletion of length $\leq \delta$, with $\delta + |r| < |s|$, in the read sequence, we first determine the number of possible mapping locations of the read within s. Recall that we assume independence between the mapping locations.

$$\begin{aligned} del(|r|, |s|, \delta) &= \sum_{i=1}^{\delta} |s| - |r| - i + 1 \\ &= \delta \cdot (|s| - |r| + 1) - \tfrac{\delta(\delta+1)}{2} \end{aligned} \tag{6.3}$$

To get the total number of mapping configurations we can multiply $cuts(|r|, m)$ by $del(|r|, |s|, \delta)$. The expected number of random deletion-indicating matches for the whole set of reads then is

$$\begin{aligned} E_{del} = \quad & |R| \cdot p(|r|, m, e_p, e_s, k) \cdot cuts(|r|, m) \\ & \cdot del(|r|, |s|, \delta) \end{aligned} \tag{6.4}$$

Insertions:
The maximum size of an insertion in the read sequence is $|r| - 2m$. An insertion of length i essentially shortens the read by i characters. Therefore, the number of possible insertion breakpoints on the read, given by $cuts(|r| - i, m)$, depends on $|r| - i$ rather than $|r|$. Also, the probability of a random read match now depends on $|r| - i$ and the total number of errors then is $e = \max(\lfloor \epsilon \cdot (|r| - i) \rfloor, e_s + e_p)$. For a read indicating an insertion of length i, the total number of different mapping positions in the DNA sequence is $|s| - |r| - i - 1$. Hence, for the whole set of reads we expect to see E_{ins} many insertion-indicating matches:

$$\begin{aligned} E_{ins} = \quad & |R| \cdot \sum_{i=1}^{|r|-2m} (|s| - |r| - i - 1) \\ & \cdot p(|r| - i, m, e_p, e_s, k) \\ & \cdot cuts(|r| - i, m) \end{aligned} \tag{6.5}$$

Variant Comparison Tool

SeqAn tool variantComp:

VariantComp takes three input files: a file containing predicted variants (GFF), a file containing reference set variants (GFF), and a reference sequence (Fasta). It can compare SNVs, insertions, deletions, inversions and duplications (intrachromosomal events) and compute statistics about true positives, false positives, and false negatives, optionally for variants of different size ranges separately (SNVs are always treated separately). The most important parameters to set are

- Computation of the equivalent indel region, *eir*, as defined in (Krawitz *et al.*, 2010), to account for indel/SV placement ambiguity (for indels, inversions, duplications), requires inserted sequence to be supplied in input files.
- Position tolerance, allowing variant positions to be inexact by the specified number of base pairs. This is especially important if indel breakpoints are not necessarily expected to be exact, or insertion sequence is not given.
- Size tolerance, allowing predicted and reference variant to vary in size by the given percent of reference variant size.

Additionally, a file with variant size ranges for which to output statistics may be specified.

Implementation

As the most notable implementation detail, variantComp makes use of interval trees to quickly look up intersecting variants. The tool proceeds as follows:

1. File parsing: read input GFF files and reference sequence as well as range file if specified. SNVs are stored separately from other variants.

2. Compare variants (except for SNVs): for each chromosome build an interval tree of reference variants

   for each predicted variant on this chromosome query it against the interval tree (allowing position tolerance/*eir*)

   for each potentially matching pair of predicted/reference variant, check if all criteria are met

3. Compare SNVs: for each chromosome

   sort and stream over reference and predicted variants simultaneously, and check if called SNV matches reference SNV

Indel Simulator

The SeqAn tool indelSimulator was originally developed to only simulate indels in haploid data. Indels are either simulated at random positions and with random lengths (uniformly within bins) or from a given GFF file with indels (as used in Chapter 4). Later, SNV simulation and diploid data simulation were added (as used in Chapter 5). IndelSimulator takes as input a reference sequence/chromosome, and optionally a GFF file containing indel variants to draw from. Another optional input is a file describing which size ranges of indels should be simulated. If no insertion sequences are given, they are in half of the cases generated randomly with a 1/4 probability for each nucleotide, and in the other half of cases generated through duplication of the upstream sequence (emulating tandem duplication). The output is the manipulated reference sequence and a file of indels and SNVs that were implanted (with their coordinates with respect to the original sequence).

Implementation

Given a number of SNVs and indels to simulate, the indelSimulator tool proceeds as follows:

- File parsing: a reference sequence and optionally a GFF file containing reference indels is parsed.

- SNVs are placed randomly in a manipulated copy of the reference sequence. In case of diploid simulation, we have two copies of th reference sequence and the SNV is with 1/3 probability applied to both copies.

- A set of indels is either randomly generated or randomly drawn from the set of parsed reference variants. If multiple size ranges are specified, each such range receives equal probability. For each indel we check that it maintains a certain minimum distance from the next indel. The final list of indels is then sorted according to genome position.
- Indels are introduced one by one into the manipulated chromosome, updating the manipulated sequence from left to right.
- SNVs and indels are written into a file, writing out the original reference sequence positions.

Single-end read simulation

Simulation procedure and experiment design

We performed our simulation experiments according to the following steps (taken from (Emde *et al.*, 2012), supplement):

1. Choose 1000 indels from dbSNP130+DGV and implant these into chromosome 21. Indels are drawn uniformly from range buckets: $[-30,-10]$, $[-9,-1]$, $[1,9]$, $[10,30]$ and $[31,3000]$, where negative numbers indicate insertions. Each bucket has approximately 200 representatives. "Natural" distribution of indel sizes is only retained within range buckets. Note that our sampling procedure does not represent a realistic indel distribution, but gives us sufficient sample size for testing different indel size ranges, in particular medium sized to large indels. Maximum insertion length thus is 30 bp, maximum deletion length 3,000 bp. Insertion sequences are generated randomly. Indels are simulated at a distance such that no read will contain two indels. Single base substitutions are added at a rate of 0.001 to simulate SNPs. This yields the manipulated chromosome chr_m and a set of reference indels I_m.

2. Simulate a set of single-end reads R from chr_m, using the Mason read simulator[1] and using typical Illumina sequencing error settings: an average substitution rate of 0.005 (with error probabilities increasing from 5' to 3' end) and an insertion and deletion rate of 0.0002 each. The total number of reads $|R|$ is such that the coverage is approximately 30x. *Remark:* Indel sequencing errors in Illumina data have been observed to be strongly biased towards long homopolymer runs (Albers et al., *Genome Research*, 2010). The current version of the read simulator does not take this into account, but simulates indels at constant rates independent of sequence context. .

3. Map the whole set of reads onto the human reference genome (NCBI build 36) with an error rate of at most 5%. Only mismatches, i.e. Hamming distance, are allowed. To this end, we use RazerS in 100% sensitivity mode.

4. Map unmapped reads using Splazers, setting the minimum match length to 16 bp, allowing for one error in both prefix and suffix (i.e. $e_p = e_s = 1$) and requiring an overall sequence identity of $\geq 95\%$ ($\epsilon = 0.05$). The maximum gap length δ is set to 5 kb.

5. Call indels with snpStore. An indel is called whenever an absolute number of at least 2 reads, and a relative number of at least 30% of reads spanning the position support the indel event. Only unique matches are considered, i.e. only matches with single best scores. This yields a set of predicted indels I_p.

6. Compare I_m with I_p, counting the number of true positives (TP), false positives (FP) and false negatives (FN). In order to determine whether two indels are the same, we calculate the *extended indel region* as defined in (Krawitz *et al.*, 2010).

[1] www.seqan.de/projects/mason

SnpStore

Maq Model for genotype likelihoods of homozygotes

The following formulae are in large part taken form the Maq supplementary material. Given the two most frequent bases X and Y in an alignment column, we want to calculate the probabilities of homozygotes (X, X) and (Y, Y). To derive these probabilities, we first need to calculate the probability of observing exactly k errors in n bases. We denote this probability as α_{nk} and

$$\alpha_{nk} = (1 - \beta_{nk}) \prod_{i=0}^{k-1} \beta_{ni} \tag{6.6}$$

where β_{nk} is defined as

$$\beta_{nk} \widehat{=} \begin{cases} P(\text{at least k+1 errors}|\text{at least k errors in n bases }) & \text{if } k > 0 \\ P(\text{at least 1 error in n bases }) & \text{if } k = 0 \end{cases} \tag{6.7}$$

The probability of seeing exactly k errors is the product of the probability of seeing at least k errors in n bases ($\prod_{i=0}^{k-1} \beta_{ni}$) and the probability of not seeing the (k+1)-th error ($1 - \beta_{nk}$). As $\beta_{nn} = 0$, it holds that

$$\sum_{k=0}^{n} \alpha_{nk} = \sum_{k=0}^{n} (1 - \beta_{nk}) \prod_{i=0}^{k-1} \beta_{ni} = 1 \tag{6.8}$$

and

$$\beta_{nk} = 1 - \frac{\alpha_{nk}}{1 - \sum_{i=0}^{k-1} \alpha_{ni}} = \frac{1 - \sum_{i=0}^{k} \alpha_{ni}}{1 - \sum_{i=0}^{k-1} \alpha_{ni}} \tag{6.9}$$

Now, based on binomial distribution with n realizations (read bases) and k observations of an event (errors) that arise with probability ϵ, i.e. random variable $X \sim B(n, \epsilon)$, the probability of seeing k errors in n bases is

$$\bar{\alpha}_{nk}(\bar{\epsilon}) = \binom{n}{k} \bar{\epsilon}^k (1 - \bar{\epsilon})^{n-k}$$

with uniform base error rate $\bar{\epsilon}$. Furthermore:

$$\bar{\beta}_{nk}(\bar{\epsilon}) = \frac{1 - \sum_{i=0}^{k} \bar{\alpha}_{ni}}{1 - \sum_{i=0}^{k-1} \bar{\alpha}_{ni}}$$

In real sequencing data, the error independence does not hold due to amplification and mapping artifacts, which make errors accumulate. In Maq, β_{nk} is therefore modeled by

$$\beta_{nk}(\bar{\epsilon}) = \bar{\beta}_{nk}^{f_k}(\bar{\epsilon})$$

where $f_k = 0.85^k$ in practise. Thus we have

$$\alpha_{nk}(\bar{\epsilon}) = (1 - \bar{\beta}_{nk}^{f_k}(\bar{\epsilon})) \prod_{i=0}^{k-1} \bar{\beta}_{ni}^{f_i}(\bar{\epsilon}) = (1 - \bar{\beta}_{nk}^{f_k}(\bar{\epsilon})) \prod_{i=0}^{k-1} (\frac{\bar{\beta}_{ni}(\bar{\epsilon})}{\bar{\epsilon}})^{f_i} \cdot \bar{\epsilon}^{f_i} = c_{nk}(\bar{\epsilon}) \cdot \prod_{i=0}^{k-1} \bar{\epsilon}^{f_i}$$

with

$$c_{nk}(\bar{\epsilon}) = (1 - \bar{\beta}_{nk}^{f_k}(\bar{\epsilon})) \prod_{i=0}^{k-1} (\frac{\bar{\beta}_{ni}(\bar{\epsilon})}{\bar{\epsilon}})^{f_i}$$

Finally, individual base error probabilities are then incorporated by approximating

$$\alpha_{nk}(\epsilon_1, ..., \epsilon_k; \epsilon_{k+1}, ..., \epsilon_n) = c_{nk}(\bar{\epsilon}) \cdot \prod_{i=0}^{k-1} \epsilon_i^{f_i} \tag{6.10}$$

where

$$log(\bar{\epsilon}) = \frac{\sum_{i=0}^{k-1} f_i log(\epsilon_{i+1})}{\sum_{i=0}^{k-1} f_i} \tag{6.11}$$

Pseudocode for Computing Heavy Lossless Shapes

For an alignment of two sequences of length n with exactly k mismatches, we can compute a lossless shape *bestShape* with optimal threshold $t_0 \geq minT$ given a minimum allowed span $minSpan$ and maximum allowed span $maxSpan$:

```
1: function HEAVIESTSHAPE(n, k, minSpan, maxSpan, minT)
2:     for span in minSpan : maxSpan do
3:         shape = (1, span) // first and last position ofshape are fixed
4:         for i in 1:n-span+1 do
5:             initialize k-dimensional I_i with shape
6:         end for
7:         pos = 2
8:         while (pos < span) do
9:             tempShape = shape.insert(pos)
10:            for i in 1:n-span+1 do
11:                tempI_i = I_i
12:                add 1s to tempI_i for each entry indexed with pos
13:            end for
14:            M = sum(tempI_i)
15:            t = n - span+ 1 - max(M)
16:            if t ≥ minT then
17:                shape = tempShape
18:                for i in 1:n - span+1 do
19:                    I_i = tempI_i
20:                end for
21:            end if
22:            pos = pos+1
23:        end while
24:        if (length(shape)≥length(bestShape)) then
25:            bestShape = shape
26:        end if
27:    end for
28:    return sort(bestShape)
29: end function
```

Pseudocode for RazerS, MicroRazerS and SplazerS

The following pseudocodes give a very high-level description of the RazerS, MicroRazerS and SplazerS algorithms. We denote with G our reference genome sequence and with R our set of reads. We assume here that all reads have the same length $readlength$ and denote with k the maximum allowed number of errors. Other variables and functions are named such that they are self-explanatory.

RazerS

```
1: function RazerS(G, R, k, maxHits)
2:     chooseParameters(readlength, k)
3:     S = SwiftFinder(R)
4:     for each g ∈ G do
5:         while (hit = nextHit(S)) do
6:             if (match = verifyMatch(hit)) then
7:                 matches.insert(match)
8:             end if
9:         end while
10:    end for
11:    output matches
12: end function
```

Remark: Line 6 evaluates as "false" if function verifyPrefixMatch returns *invalid*.

```
1: function verifyMatch(swiftHit)
2:     r = swiftHit.read
3:     g = swiftHit.gInfix
4:     a = semiglobalAlignment(r,g)
5:     if (distance(a)≤ k) then
6:         return a
7:     end if
8:     return invalid
9: end function
```

MicroRazerS

In addition to the parameters above, MicroRazerS has a prefix seed length parameter *seedLen*.

```
1: function MicroRazerS(G, R, seedLen, k, maxHits)
2:     chooseParameters(seedLen, k)
3:     S = SwiftFinder(prefixes(R,seedLen))
4:     for each g ∈ G do
5:         while (hit = nextHit(S)) do
6:             if (match = verifyPrefixMatch(hit,seedLen,k)) then
7:                 matches.insert(match)
8:             end if
9:         end while
10:    end for
11:    output matches
12: end function
```

```
1: function verifyPrefixMatch(swiftHit, seedLen, k)
2:     r = swiftHit.read
3:     g = swiftHit.gInfix
4:     seed = semiglobalAlignment(prefix(r,seedLen),g)
5:     if (distance(seed)≤ k) then
6:         a = extendSeedRight(seed,0) // allow 0 errors in extension
7:         return a
8:     end if
9:     return invalid
10: end function
```

SplazerS

Here we will denote the minimum match length as m, the maximum number of prefix errors as e_p, the maximum number of suffix errors as e_s and the maximum distance of prefix and suffix match as δ.

```
1: function SplazerS(G, R, m, e_p, e_s, k, maxHits)
2:     chooseParameters(m, e_p)
3:     S_p = SwiftFinder(prefixes(R, m))
4:     chooseParameters(m, e_s)
5:     S_s = SwiftFinder(suffixes(R, m))
6:     for each g ∈ G do
7:         Q = empty queue
8:         while (hit_s = nextHit(S_s)) do
9:             clean(Q, δ) // remove out of window prefix hits
10:            while ((hit_p = nextHit(S_p)) & (hit_p.pos < hit_s.pos)) do
11:                if (hit_p within allowed distance δ of hit_s) then
12:                    Q.insert(hit_p)
13:                end if
14:            end while
15:            for each potential prefix match hit_p in Q do
16:                if (hit_s not yet verified) then
17:                    if (!(match_s = verifySuffixMatch(hit_s, m, e_s, k))) then
18:                        break
19:                    end if
20:                end if
21:                if (hit_p not yet verified) then
22:                    if (!(match_p = verifyPrefixMatch(hit_p, m, e_p, k))) then
23:                        remove hit_p from Q
24:                        continue
25:                    end if
26:                end if
27:                if (match = combineMatches(match_p, match_s, k, δ)) then
28:                    matches.insert(match)
29:                end if
30:            end for
31:        end while
32:    end for
33:    output matches
34: end function
```

```
1: function verifyPrefixMatch(swiftHit, m, e_p, k)
2:     r = swiftHit.read
3:     g = swiftHit.gInfix
4:     seed = semiglobalAlignment(prefix(r, m), g)
5:     if (distance(seed)≤ e_p) then
6:         a = extendSeedLeft(seed, k - distance(seed))
7:         return a
8:     end if
9:     return invalid
10: end function
```

```
1: function verifySuffixMatch(swiftHit, m, e_s, k)
2:     r = swiftHit.read
3:     g = swiftHit.gInfix
4:     seed = semiglobalAlignment(suffix(r, m), g)
5:     if (distance(seed)≤ e_s) then
6:         a = extendSeedRight(seed,k - distance(seed))
7:         return a
8:     end if
9:     return invalid
10: end function
```

The following function describes the breakpoint computation function as described in section 4.3.2. Remember that the two vectors f^p and f^s store the best score for each row in the banded alignment matrices for the prefix and suffix, respectively. The vectors b^p and b^s store the corresponding projected alignment end/begin position for each entry in f^p and f^s, respectively. Function rev reverses a sequence.

```
1: function combineMatches(match_p, match_s, k, δ)
2:     if (match_s and match_p do not overlap) then // or criteria from section 4.3.2 violated
3:         return invalid
4:     end if
5:     if (match_s and match_p share largest overlap on read sequence) then
6:         seq1_p = read infix that overlaps
7:         seq2_p = reference infix aligned to seq1_p through match_p
8:         seq1_s = seq1_p
9:         seq2_s = reference infix aligned to seq1_s through match_s
10:    else if (match_s and match_p share largest overlap on reference sequence) then
11:        seq1_p = reference infix that overlaps
12:        seq2_p = read infix aligned to seq1_p through match_p
13:        seq1_s = seq1_p
```

14: $seq2_s$ = read infix aligned to $seq1_s$ through $match_s$
15: **end if**
16: $fixed_p$ = fixed alignment part of $match_p$
17: $fixed_s$ = fixed alignment part of $match_s$
18: $band = k$ - distance($fixed_p$) - distance($fixed_s$)
19: (f^p, b^p) = bandedGlobalAlignment($seq1_p, seq2_p, band$)
20: (rev(f^s),rev(b^s)) = bandedGlobalAlignment(rev($seq1_s$), rev($seq2_s$), $band$)
21: set x such that $f^p[x] + f^s[x+1]$ is minimal
22: **if** $((f^p[x] + f^s[x+1] + \text{distance}(fixed_p) + \text{distance}(fixed_s)) > k)$ **then**
23: return *invalid*
24: **end if**
25: $z = b^p[x]$
26: $z' = b^s[x+1]$
27: set $match_p$ end positions according to x and z
28: set $match_s$ begin positions according to x and z'
29: return splitMatch($match_p$,$match_s$)
30: **end function**

Detailed program command lines

RazerS

Commands are given in the order: 1. Experiment A (Drosophila 36 bp, up to 2 Hamming errors), 2. Experiment C (Human 63 bp, up to 5 Hamming errors), and (if applicable) 3. Experiment B (Drosophila 36 bp, up to 2 Levenshtein errors).

RazerS with 100% sensitivity was run with:

```
razers -rr 100 -i 92 -m 1 genome.fa reads.fa -o razers.out
razers -rr 100 -i 92 -m 1 genome.fa reads.fa -o razers.out
razers -id -rr 100 -i 92 -m 1 genome.fa reads.fa -o razers.out
```

RazerS with 90% sensitivity was run with:

```
razers -rr 99 -i 92 -m 1 genome.fa reads.fa -o razers.out
razers -rr 99 -i 92 -m 1 genome.fa reads.fa -o razers.out
razers -id -rr 99 -i 92 -m 1 genome.fa reads.fa -o razers.out
```

SeqMap was run with:

```
seqmap 2 reads.fa genomeFasta seqmap.out /output_top_matches:1
seqmap 5 reads.fa genomeFasta seqmap.out /output_top_matches:1
seqmap 2 reads.fa genomeFasta seqmap.out /allow_insdel:2 /output_top_matches:1
```

Zoom was run with:

```
zoom -i reads.fa -g genome.fa -o zoom.out -mk 1 -mm 2
zoom -i reads.fa -g genome.fa -o zoom.out -mk 1 -mm 5 -sv r4
zoom -i reads.fa -g genome.fa -o zoom.out -mk 1 -ed 2
```

Shrimp was run with:

```
shrimp -m 1 -i 0 -e -100 -f -100 -g -100 -q -100 -h 34 -o 1 reads.fa genome.fa
shrimp -m 1 -i 0 -e -100 -f -100 -g -100 -q -100 -h 58 -o 1 reads.fa genome.fa
shrimp -m 1 -i -1 -e -1 -f -1 -g -1 -q -1 -h 32 -o 1 reads.fa genome.fa
```

Soap was run with:

```
soap -v 2 -a reads.fa -d genome.fa -f 2 -o soap.out
soap -v 5 -a reads.fa -d genome.fa -f 5 -o soap.out
```

Maq was run with:

```
maq map -n 2 -e 80  maq.out genome.bfa reads.bfq
maq map -n 3 -e 200 maq.out genome.bfa reads.bfq
```

Rabema

Let K denote the allowed number of errors.

Bowtie was run with (default and improved parameterization):

```
bowtie --sam -q -k 100 genome.fa reads.fq out.sam
bowtie --sam -q -k 100 --maxbts 800 --seedmms 3 -y --best --maqerr 400 \
  genome reads.fq out.sam
```

On Illumina reads, Bwa was run with:

```
bwa aln -f out.sai -n K genome reads.fq
bwa samse -n 99 -f out.sam -n K genome out.sai reads.fq
```

and on 454 reads with:

```
bwa bwasw -f out.sam -n K genome.fa reads.fq
```

Soap2 was run with (default and improved parameterization):

```
soap2 -v K -a reads.fq -D genome -o out.soap
soap2 -l 60 -r 2 -v K -a reads.fq -D genome -o out.soap
```

Shrimp was run with:

```
gmapper -E -o 100 reads.fa genome.fa -H -s w16 > output.sam
```

More detailed information about parameterization is given in the supplementary material in (Holtgrewe *et al.*, 2011).

MicroRazerS

MicroRazerS was run with:

```
micro_razers -m 20 -pa -sL 16 Hs.dna.fa reads.fa -o reads.out
```

SOAP2 was run with:

```
soap2 -I 16 -M 0 -v 20 -a reads.fa -D genomeIndex -o soap.out
```

Bowtie was run with:

```
bowtie -e 500 --strata --best -n 0 -l 16 genomeIndex reads.fq
```

For Bowtie and SOAP2 the outputs were filtered keeping the longest hits only with at most 20 mapping positions.

SplazerS - 1000 Genomes Data NA12878

In the 1000 Genomes Project data analysis, Pindel (version 2.2) was run with:

```
pindel -x 4 -u 0.05 -e 0.05 -c 22  -f chr22.fa -p pindelIn -o pindelOut
```

SVseq2 (version 2.0.1) was run with:

```
SVseq2 -c 22 -r /project/general/Genomes/Hs/GRCh37/Hs.dna.fa -b reads.bam --o delsOut
SVseq2 -insertion -c 22 -b reads.bam --o insOut
```

Output files containing predicted deletions and insertions were then converted to a sorted GFF file for subsequent overlap computation.

SplazerS in paired-end mode was run with:

```
splazers -an -sm 14 -ep 1 -es 1 -i 95 -s 11011011011 -t 1  -maxG 8092 -ll 300 -le 280 \
  -m 3 chr22.fa reads.sam -o splitMapped.gff
```

In single-end mode, SplazerS was run with:

```
splazers -sm 16 -ep 1 -es 1 -i 95 -s 11011011011011 -t 1  -maxG 8092 -pc 4 -m 3 \
  -o unanchoredSplitMapped.gff  Hs19.dna.fa reads.fa
```

Subsequent indel detection was performed with snpStore:

```
snpStore -mp 1 -oa  -it 3 -ipt 0.5 -mc 3 -ebi 6 -id indels.gff  chr21.fa \
  "splitMapped.gff unanchoredSplitMapped.gff"
```

SplazerS - Paired-end 100 bp NA18507

Pindel was run with:

```
pindel -x 4 -u 0.05 -e 0.03  -c 21 -f chr21.fa -p pindelIn -o pindelOut
```

SplazerS was run with:

```
splazers -an -sm 16 -ep 1 -es 1 -i 95 -s 11011011011011 -t 1 -maxG 8092 -ll 312 \
  -le 57 -m 3 chr21.fa reads.sam -o splitMapped.gff
```

For the edit distance mapping step, SplazerS was run with:

```
splazers -an -sm 16 -ep 1 -es 1 -i 95 -s 11111111 -t 1 -maxG 8092 -ll 312 \
  -le 57 -m 3 chr21.fa unmapped.sam -o splitMapped_edit.gff
```

Subsequent indel detection was performed with snpStore:

```
snpStore -mp 1 -oa  -it 3 -ipt 0.5 -mc 3 -ebi 6 -id indels.gff chr21.fa \
  "splitMapped_edit.gff splitMapped.gff"
```

SplazerS - Simulation Data Analysis

SplazerS was run with:

```
splazers -i 95 -sm 16 -ep 1 -es 1 -maxG 5000 -m 5 -o run01.100.split.gff \
  -s 11011011011011 -t 1 Hs.dna.fa run01.100.indeled.reads.unmapped.fa
```

GSNAP (version 2010-07-27) was run with:

```
gsnap -i 0 -m 5 -y 30 -z 5000 -d NHGD_R36 -D NHGD_R36 -n 5 -t 4 -A sam \
  run01.100.indeled.reads.unmapped.fa > run01.100.gsnap.split.sam
```

BWA (version 0.5.8a) was run with:

```
bwa aln -i 20 -l 20 -k 1 -n 5 -o 1 -e 5000 -d 1000 -t 4 -R 5 \
  hg18 run01.100.indeled.reads.unmapped.fa -f run01.100.bwa.split.bam
```

Formats are converted to snpStore input, i.e. GFF files. For GSNAP, a match is tagged as "unique" if the 0x100 bit (secondary alignment) is not set in the SAM flag. For BWA, matches with mapping quality > 0 are tagged as unique. Only unique reads are used for subsequent indel calling.

SnpStore was called with:

```
snpStore -it 2 -ipt 0.25 -mc 2 -id run01.100.indels.gff Hs.dna.fa \
  "run01.100.split.gff run01.100.indeled.reads.gff"
```

SplazerS - Targeted Resequencing Data

SplazerS was run with:

```
splazers -m 1 -pa -of 3 -i 95 -sm 23 -s 111001110011100111 -t 2 \
  -maxG 50000 genome.fa reads.fa -o splazers.out
```

Indel calling on split mapped reads was done with Perl scripts.

SnpStore - Simulation Data

RazerS mapping with:

```
razers genome.fa reads.fa -rr 100 -id -i 93 -m 2 -of 4 -o mapped_read.sam
```

Bwa (version 0.6.2) mapping was done with:

```
bwa aln genome reads.fq > aln.sai
bwa samse genome aln.sai reads.fq > mapped_reads.sam
```

GATK (version 1.4-37) realignment was run with:

```
java -jar GenomeAnalysisTK.jar -T RealignerTargetCreator -I mapped_reads.bam \
  -R genome.fa -o intervals.out
java -jar GenomeAnalysisTK.jar -T IndelRealigner -I mapped_reads.bam \
  -R genome.fa -targetIntervals intervals.out -o mapped_reads.realigned.bam
```

GATK SNV and indel calling was run with:

```
java -jar GenomeAnalysisTK.jar -T UnifiedGenotyper -R genome.fa \
  -I mapped_reads.realigned.bam -o snvs.out
java -jar GenomeAnalysisTK.jar -T UnifiedGenotyper -R genome.fa \
  -I mapped_reads.realigned.bam -o indels.out -glm INDEL
```

Samtools (0.1.18) was run with:

```
samtools mpileup -ug -m 1 -F 0.002 -d8000 -f genome.fa mapped_reads.fa \
  | bcftools view -bvcg - > var.raw.bcf
bcftools view var.raw.bcf | vcfutils.pl varFilter -D180 > variants.out
```

SnpStore was run (without and with realignment):

```
snpStore genome.fa mapped_reads.sam -o snvs.out -id indels.out -mc 2 \
  -if 1 -oa -dp 2 -eb 2 -of 1 -hq -it 2 -ipt 0.15 -mpr 10 -bsi
snpStore genome.fa mapped_reads.sam -o snvs.out -id indels.out -mc 2 \
  -re -if 1 -oa -dp 2 -eb 2 -of 1 -hq -it 2 -ipt 0.15 -mpr 10 -bsi
```

SNV calls were filtered to have at least read depth 3.

SnpStore - 1000 Genomes Data NA12878

SnpStore, Samtools and GATK were run with the same commands as in the previous section.

Additionally, GATK quality recalibration was run after realignment with:

```
java -jar GenomeAnalysisTK.jar -T CountCovariates -R genome.fa \
  -I mapped_reads.realigned.bam -recalFile reads.table.recal.csv \
  -knownSites dbsnp
java -jar GenomeAnalysisTK.jar -T TableRecalibration -R genome.fa \
  -I mapped_reads.realigned.bam -recalFile reads.table.recal.csv \
-o mapped_reads.realigned.recalib.bam
```

SnpStore's heterozygote correction was switched on with option "-ch". Read clipping was done with default values $Q_t = 10$, $w = 10$ and $m = 3$ using a Perl script.

SnpStore - Targeted Resequencing Data

SnpStore was run on the merged read data for each individual with:

```
snpStore -fc 10 -mc 3 -oa -mp 1 -mmq 10 -re genome.fa reads.gff
```

List of Figures

List of Tables

Bibliography

Abyzov, A. and Gerstein, M. (2011). Age: defining breakpoints of genomic structural variants at single-nucleotide resolution, through optimal alignments with gap excision. *Bioinformatics*, **27**(5), 595–603.

Aird, D., Ross, M. G., Chen, W.-S., Danielsson, M., Fennell, T., Russ, C., Jaffe, D. B., Nusbaum, C., and Gnirke, A. (2011). Analyzing and minimizing pcr amplification bias in illumina sequencing libraries. *Genome Biol*, **12**(2), R18.

Albers, C. A., Lunter, G., MacArthur, D. G., McVean, G., Ouwehand, W. H., and Durbin, R. (2011). Dindel: accurate indel calls from short-read data. *Genome Res*, **21**(6), 961–973.

Albert, T. J., Molla, M. N., Muzny, D. M., Nazareth, L., Wheeler, D., Song, X., Richmond, T. A., Middle, C. M., Rodesch, M. J., Packard, C. J., Weinstock, G. M., and Gibbs, R. A. (2007). Direct selection of human genomic loci by microarray hybridization. *Nat Methods*, **4**(11), 903–905.

Alkan, C., Coe, B. P., and Eichler, E. E. (2011). Genome structural variation discovery and genotyping. *Nat Rev Genet*, **12**(5), 363–376.

Ameur, A., Wetterbom, A., Feuk, L., and Gyllensten, U. (2010). Global and unbiased detection of splice junctions from rna-seq data. *Genome Biol*, **11**(3), R34.

Angiuoli, S. V. and Salzberg, S. L. (2011). Mugsy: fast multiple alignment of closely related whole genomes. *Bioinformatics*, **27**(3), 334–342.

Anson, E. L. and Myers, E. W. (1997). ReAligner: A program for refining DNA sequence multi-alignments. pages 9–16.

Applied Biosystems (2012). Capillary sequencing systems.

Au, K. F., Jiang, H., Lin, L., Xing, Y., and Wong, W. H. (2010). Detection of splice junctions from paired-end rna-seq data by splicemap. *Nucleic Acids Res*, **38**(14), 4570–4578.

Bailey, T. L. (2011). Dreme: motif discovery in transcription factor chip-seq data. *Bioinformatics*, **27**(12), 1653–1659.

Bentley, D. R., Balasubramanian, S., Swerdlow, H. P., Smith, G. P., Milton, J., Brown, C. G., Hall, K. P., Evers, D. J., Barnes, C. L., Bignell, H. R., *et al.* (2008). Accurate whole human genome sequencing using reversible terminator chemistry. *Nature*, **456**(7218), 53–9.

Bignell, G. R., Huang, J., Greshock, J., Watt, S., Butler, A., West, S., Grigorova, M., Jones, K. W., Wei, W., Stratton, M. R., Futreal, P. A., Weber, B., Shapero, M. H., and Wooster, R. (2004). High-resolution analysis of dna copy number using oligonucleotide microarrays. *Genome Res*, **14**(2), 287–295.

Branton, D., Deamer, D. W., Marziali, A., Bayley, H., Benner, S. A., Butler, T., Di Ventra, M., Garaj, S., Hibbs, A., Huang, X., Jovanovich, S. B., Krstic, P. S., Lindsay, S., Ling, X. S., Mastrangelo, C. H., Meller, A., Oliver, J. S., Pershin, Y. V., Ramsey, J. M., Riehn, R., Soni, G. V., Tabard-Cossa, V., Wanunu, M., Wiggin, M., and Schloss, J. A. (2008). The potential and challenges of nanopore sequencing. *Nat Biotechnol*, **26**(10), 1146–1153.

Burkhardt, S. and Kärkkäinen, J. (2001). Better filtering with gapped q-grams. In *CPM '01: Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching*, pages 73–85, London, UK. Springer-Verlag.

Burrows, M. and Wheeler, D. J. (1994). A block-sorting lossless data compression algorithm. Technical Report 124.

Campbell, P. J., Stephens, P. J., Pleasance, E. D., O'Meara, S., Li, H., Santarius, T., Stebbings, L. A., Leroy, C., Edkins, S., Hardy, C., Teague, J. W., Menzies, A., Goodhead, I., Turner, D. J., Clee, C. M., Quail, M. A., Cox, A., Brown, C., Durbin, R., Hurles, M. E., Edwards, P. A. W., Bignell, G. R., Stratton, M. R., and Futreal, P. A. (2008). Identification of somatically acquired rearrangements in cancer using genome-wide massively parallel paired-end sequencing. *Nat Genet*, **40**(6), 722–729.

Chamberlain, J. S., Gibbs, R. A., Ranier, J. E., Nguyen, P. N., and Caskey, C. T. (1988). Deletion screening of the duchenne muscular dystrophy locus via multiplex dna amplification. *Nucleic Acids Res*, **16**(23), 11141–11156.

Chen, K., Wallis, J. W., McLellan, M. D., Larson, D. E., Kalicki, J. M., Pohl, C. S., McGrath, S. D., Wendl, M. C., Zhang, Q., Locke, D. P., *et al.* (2009). Breakdancer: an algorithm for high-resolution mapping of genomic structural variation. *Nat Methods*, **6**(9), 677–681.

Chen, Y.-A., Lin, C.-C., Wang, C.-D., Wu, H.-B., and Hwang, P.-I. (2007). An optimized procedure greatly improves est vector contamination removal. *BMC Genomics*, **8**, 416.

Chikhi, R. (2012). *Computational Methods for de novo Assembly of Next-Generation Genome Sequencing Data*. Ph.D. thesis, École Normale Supérieure de Cachan.

Chikhi, R. and Lavenier, D. (2009). Paired-end read length lower bounds for genome re-sequencing. *BMC Bioinformatics*, **10**(Suppl 13), O2.

Collins, F. S., Morgan, M., and Patrinos, A. (2003). The human genome project: lessons from large-scale biology. *Science*, **300**(5617), 286–290.

Conrad, D. F., Pinto, D., Redon, R., Feuk, L., Gokcumen, O., Zhang, Y., Aerts, J., Andrews, T. D., Barnes, C., Campbell, P., Fitzgerald, T., Hu, M., Ihm, C. H., Kristiansson, K., Macarthur, D. G., Macdonald, J. R., Onyiah, I., Pang, A. W. C., Robson, S., Stirrups, K., Valsesia, A., Walter, K., Wei, J., , W. T. C. C. C., Tyler-Smith, C., Carter, N. P., Lee, C., Scherer, S. W., and Hurles, M. E. (2010). Origins and functional impact of copy number variation in the human genome. *Nature*, **464**(7289), 704–712.

Cordero, F., Beccuti, M., Arigoni, M., Donatelli, S., and Calogero, R. A. (2012). Optimizing a massive parallel sequencing workflow for quantitative mirna expression analysis. *PLoS One*, **7**(2), e31630.

Cox, A. J. (2006). Eland: efficient local alignment of nucleotide data. unpublished.

Darling, A. E., Mau, B., and Perna, N. T. (2010). progressivemauve: multiple genome alignment with gene gain, loss and rearrangement. *PLoS One*, **5**(6), e11147.

David, M., Dzamba, M., Lister, D., Ilie, L., and Brudno, M. (2011). Shrimp2: Sensitive yet practical short read mapping. *Bioinformatics*.

DePristo, M. A., Banks, E., Poplin, R., Garimella, K. V., Maguire, J. R., Hartl, C., Philippakis, A. A., del Angel, G., Rivas, M. A., Hanna, M., McKenna, A., Fennell, T. J., Kernytsky, A. M., Sivachenko, A. Y., Cibulskis, K., Gabriel, S. B., Altshuler, D., and Daly, M. J. (2011). A framework for variation discovery and genotyping using next-generation dna sequencing data. *Nat Genet*, **43**(5), 491–498.

Dohm, J., Lottaz, C., Borodina, T., and Himmelbauer, H. (2008). Substantial biases in ultra-short read data sets from high-throughput dna sequencing. *Nucleic Acids Res.*, **36**.

Döring, A., Weese, D., Rausch, T., and Reinert, K. (2008). SeqAn an efficient, generic C++ library for sequence analysis. *BMC Bioinf.*, **9**, 11.

Durbin, R. M., Abecasis, G. R., Altshuler, D. L., Auton, A., Brooks, L. D., Durbin, R. M., Gibbs, R. A., Hurles, M. E., and McVean, G. A. (2010). A map of human genome variation from population-scale sequencing. *Nature*, **467**(7319), 1061–1073.

Emde, A.-K., Grunert, M., Weese, D., Reinert, K., and Sperling, S. R. (2010). Microrazers: rapid alignment of small rna reads. *Bioinformatics*, **26**(1), 123–124.

Emde, A.-K., Schulz, M. H., Weese, D., Sun, R., Vingron, M., Kalscheuer, V. M., Haas, S. A., and Reinert, K. (2012). Detecting genomic indel variants with exact breakpoints in single- and paired-end sequencing data using splazers. *Bioinformatics*, **28**(5), 619–627.

Ewing, B. and Green, P. (1998). Base-calling of automated sequencer traces using phred. ii. error probabilities. *Genome Res*, **8**(3), 186–194.

Fonseca, N. A., Rung, J., Brazma, A., and Marioni, J. C. (2012). Tools for mapping high-throughput sequencing data. *Bioinformatics*.

Franklin, R. E. and Gosling, R. G. (1953). Molecular configuration in sodium thymonucleate. *Nature*, **171**(4356), 740–741.

Friedländer, M. R., Chen, W., Adamidi, C., Maaskola, J., Einspanier, R., Knespel, S., and Rajewsky, N. (2008). Discovering micrornas from deep sequencing data using mirdeep. *Nat Biotechnol*, **26**(4), 407–415.

Fu, Y. H., Pizzuti, A., Fenwick, Jr, R., King, J., Rajnarayan, S., Dunne, P. W., Dubel, J., Nasser, G. A., Ashizawa, T., and de Jong, P. (1992). An unstable triplet repeat in a gene related to myotonic muscular dystrophy. *Science*, **255**(5049), 1256–1258.

Gilbert, J. A. and Dupont, C. L. (2011). Microbial metagenomics: beyond the genome. *Ann Rev Mar Sci*, **3**, 347–371.

Green, R. E., Krause, J., Briggs, A. W., Maricic, T., Stenzel, U., Kircher, M., Patterson, N., Li, H., Zhai, W., Fritz, M. H.-Y., Hansen, N. F., Durand, E. Y., Malaspinas, A.-S., Jensen, J. D., Marques-Bonet, T., Alkan, C., Prfer, K., Meyer, M., Burbano, H. A., Good, J. M., Schultz, R., Aximu-Petri, A., Butthof, A., Hber, B., Hffner, B., Siegemund, M., Weihmann, A., Nusbaum, C., Lander, E. S., Russ, C., Novod, N., Affourtit, J., Egholm, M., Verna, C., Rudan, P., Brajkovic, D., Kucan, Z., Gusic, I., Doronichev, V. B., Golovanova, L. V., Lalueza-Fox, C., de la Rasilla, M., Fortea, J., Rosas, A., Schmitz, R. W., Johnson, P. L. F., Eichler, E. E., Falush, D., Birney, E., Mullikin, J. C., Slatkin, M., Nielsen, R., Kelso, J., Lachmann, M., Reich, D., and Pbo, S. (2010). A draft sequence of the neandertal genome. *Science*, **328**(5979), 710–722.

Greshock, J., Feng, B., Nogueira, C., Ivanova, E., Perna, I., Nathanson, K., Protopopov, A., Weber, B. L., and Chin, L. (2007). A comparison of dna copy number profiling platforms. *Cancer Res*, **67**(21), 10173–10180.

Griffiths-Jones, S., Saini, H. K., van Dongen, S., and Enright, A. J. (2008). mirbase: tools for microrna genomics. *Nucleic Acids Res*, **36**(Database issue), D154–D158.

Hajirasouliha, I., Hormozdiari, F., Alkan, C., Kidd, J. M., Birol, I., Eichler, E. E., and Sahinalp, S. C. (2010). Detection and characterization of novel sequence insertions using paired-end next-generation sequencing. *Bioinformatics*, **26**(10), 1277–1283.

Handsaker, R. E., Korn, J. M., Nemesh, J., and McCarroll, S. A. (2011). Discovery and genotyping of genome structural polymorphism by sequencing on a population scale. *Nat Genet*, **43**(3), 269–276.

Hashimoto, T., de Hoon, M. J. L., Grimmond, S. M., Daub, C. O., Hayashizaki, Y., and Faulkner, G. J. (2009). Probabilistic resolution of multi-mapping reads in massively parallel sequencing data using mumrescuelite. *Bioinformatics*, **25**(19), 2613–2614.

Hawkins, R. D., Hon, G. C., and Ren, B. (2010). Next-generation genomics: an integrative approach. *Nat Rev Genet*, **11**(7), 476–486.

Heinrich, V., Stange, J., Dickhaus, T., Imkeller, P., Krger, U., Bauer, S., Mundlos, S., Robinson, P. N., Hecht, J., and Krawitz, P. M. (2011). The allele distribution in next-generation sequencing data sets is accurately described as the result of a stochastic branching process. *Nucleic Acids Res*.

Herms, I. and Rahmann, S. (2008). Computing alignment seed sensitivity with probabilistic arithmetic automata. In *WABI*, pages 318–329.

Hollox, E. J., Huffmeier, U., Zeeuwen, P. L. J. M., Palla, R., Lascorz, J., Rodijk-Olthuis, D., van de Kerkhof, P. C. M., Traupe, H., de Jongh, G., den Heijer, M., Reis, A., Armour, J. A. L., and Schalkwijk, J. (2008). Psoriasis is associated with increased beta-defensin genomic copy number. *Nat Genet*, **40**(1), 23–25.

Holtgrewe, M. (2010). Mason – a read simulator for second generation sequencing data. Technical Report TR-B-10-06, Institut für Mathematik und Informatik, Freie Universität Berlin.

Holtgrewe, M., Emde, A.-K., Weese, D., and Reinert, K. (2011). A novel and well-defined benchmarking method for second generation read mapping. *BMC Bioinformatics*, **12**, 210.

Homer, N. and Nelson, S. F. (2010). Improved variant discovery through local re-alignment of short-read next-generation sequencing data using srma. *Genome Biol*, **11**(10), R99.

Hormozdiari, F., Alkan, C., Eichler, E. E., and Sahinalp, S. C. (2009). Combinatorial algorithms for structural variation detection in high-throughput sequenced genomes. *Genome Res*, **19**(7), 1270–1278.

Huang, X. and Madan, A. (1999). Cap3: A dna sequence assembly program. *Genome Res*, **9**(9), 868–877.

Iafrate, A. J., Feuk, L., Rivera, M. N., Listewnik, M. L., Donahoe, P. K., Qi, Y., Scherer, S. W., and Lee, C. (2004). Detection of large-scale variation in the human genome. *Nat Genet*, **36**(9), 949–951.

Illumina Inc. (2012). Hiseq systems.

Ingram, V. M. (1957). Gene mutations in human haemoglobin: the chemical difference between normal and sickle cell haemoglobin. *Nature*, **180**(4581), 326–328.

International Cancer Genome Consortium, Hudson, T. J., Anderson, W., Artez, A., Barker, A. D., Bell, C., Bernabé, R. R., Bhan, M. K., Calvo, F., Eerola, I., Gerhard, D. S., Guttmacher, A., Guyer, M., Hemsley, F. M., Jennings, J. L., Kerr, D., Klatt, P., Kolar, P., Kusada, J., Lane, D. P., Laplace, F., Youyong, L., Nettekoven, G., Ozenberger, B., Peterson, J., Rao, T. S., Remacle, J., Schafer, A. J., Shibata, T., Stratton, M. R., Vockley, J. G., Watanabe, K., Yang, H., Yuen, M. M. F., Knoppers, B. M., Bobrow, M., Cambon-Thomsen, A., Dressler, L. G.,

Dyke, S. O. M., Joly, Y., Kato, K., Kennedy, K. L., Nicolás, P., Parker, M. J., Rial-Sebbag, E., Romeo-Casabona, C. M., Shaw, K. M., Wallace, S., Wiesner, G. L., Zeps, N., Lichter, P., Biankin, A. V., Chabannon, C., Chin, L., Clément, B., de Alava, E., Degos, F., Ferguson, M. L., Geary, P., Hayes, D. N., Hudson, T. J., Johns, A. L., Kasprzyk, A., Nakagawa, H., Penny, R., Piris, M. A., Sarin, R., Scarpa, A., Shibata, T., van de Vijver, M., Futreal, P. A., Aburatani, H., Bays, M., Botwell, D. D. L., Campbell, P. J., Estivill, X., Gerhard, D. S., Grimmond, S. M., Gut, I., Hirst, M., Lpez-Otn, C., Majumder, P., Marra, M., McPherson, J. D., Nakagawa, H., Ning, Z., Puente, X. S., Ruan, Y., Shibata, T., Stratton, M. R., Stunnenberg, H. G., Swerdlow, H., Velculescu, V. E., Wilson, R. K., Xue, H. H., Yang, L., Spellman, P. T., Bader, G. D., Boutros, P. C., Campbell, P. J., Flicek, P., Getz, G., Guig, R., Guo, G., Haussler, D., Heath, S., Hubbard, T. J., Jiang, T., Jones, S. M., Li, Q., Lpez-Bigas, N., Luo, R., Muthuswamy, L., Ouellette, B. F. F., Pearson, J. V., Puente, X. S., Quesada, V., Raphael, B. J., Sander, C., Shibata, T., Speed, T. P., Stein, L. D., Stuart, J. M., Teague, J. W., Totoki, Y., Tsunoda, T., Valencia, A., Wheeler, D. A., Wu, H., Zhao, S., Zhou, G., Stein, L. D., Guig, R., Hubbard, T. J., Joly, Y., Jones, S. M., Kasprzyk, A., Lathrop, M., Lpez-Bigas, N., Ouellette, B. F. F., Spellman, P. T., Teague, J. W., Thomas, G., Valencia, A., Yoshida, T., Kennedy, K. L., Axton, M., Dyke, S. O. M., Futreal, P. A., Gerhard, D. S., Gunter, C., Guyer, M., Hudson, T. J., McPherson, J. D., Miller, L. J., Ozenberger, B., Shaw, K. M., Kasprzyk, A., Stein, L. D., Zhang, J., Haider, S. A., Wang, J., Yung, C. K., Cros, A., Cross, A., Liang, Y., Gnaneshan, S., Guberman, J., Hsu, J., Bobrow, M., Chalmers, D. R. C., Hasel, K. W., Joly, Y., Kaan, T. S. H., Kennedy, K. L., Knoppers, B. M., Lowrance, W. W., Masui, T., Nicols, P., Rial-Sebbag, E., Rodriguez, L. L., Vergely, C., Yoshida, T., Grimmond, S. M., Biankin, A. V., Bowtell, D. D. L., Cloonan, N., deFazio, A., Eshleman, J. R., Etemadmoghadam, D., Gardiner, B. B., Gardiner, B. A., Kench, J. G., Scarpa, A., Sutherland, R. L., Tempero, M. A., Waddell, N. J., Wilson, P. J., McPherson, J. D., Gallinger, S., Tsao, M.-S., Shaw, P. A., Petersen, G. M., Mukhopadhyay, D., Chin, L., DePinho, R. A., Thayer, S., Muthuswamy, L., Shazand, K., Beck, T., Sam, M., Timms, L., Ballin, V., Lu, Y., Ji, J., Zhang, X., Chen, F., Hu, X., Zhou, G., Yang, Q., Tian, G., Zhang, L., Xing, X., Li, X., Zhu, Z., Yu, Y., Yu, J., Yang, H., Lathrop, M., Tost, J., Brennan, P., Holcatova, I., Zaridze, D., Brazma, A., Egevard, L., Prokhortchouk, E., Banks, R. E., Uhln, M., Cambon-Thomsen, A., Viksna, J., Ponten, F., Skryabin, K., Stratton, M. R., Futreal, P. A., Birney, E., Borg, A., Brresen-Dale, A.-L., Caldas, C., Foekens, J. A., Martin, S., Reis-Filho, J. S., Richardson, A. L., Sotiriou, C., Stunnenberg, H. G., Thoms, G., van de Vijver, M., van't Veer, L., Calvo, F., Birnbaum, D., Blanche, H., Boucher, P., Boyault, S., Chabannon, C., Gut, I., Masson-Jacquemier, J. D., Lathrop, M., Pauport, I., Pivot, X., Vincent-Salomon, A., Tabone, E., Theillet, C., Thomas, G., Tost, J., Treilleux, I., Calvo, F., Bioulac-Sage, P., Clment, B., Decaens, T., Degos, F., Franco, D., Gut, I., Gut, M., Heath, S., Lathrop, M., Samuel, D., Thomas, G., Zucman-Rossi, J., Lichter, P., Eils, R., Brors, B., Korbel, J. O., Korshunov, A., Landgraf, P., Lehrach, H., Pfister, S., Radlwimmer, B., Reifenberger, G., Taylor, M. D., von Kalle, C., Majumder, P. P., Sarin, R., Rao, T. S., Bhan, M. K., Scarpa, A., Pederzoli, P., Lawlor, R. A., Delledonne, M., Bardelli, A., Biankin, A. V., Grimmond, S. M., Gress, T., Klimstra, D., Zamboni, G., Shibata, T., Nakamura, Y., Nakagawa, H., Kusada, J., Tsunoda, T., Miyano,

S., Aburatani, H., Kato, K., Fujimoto, A., Yoshida, T., Campo, E., Lpez-Otn, C., Estivill, X., Guig, R., de Sanjos, S., Piris, M. A., Montserrat, E., Gonzlez-Daz, M., Puente, X. S., Jares, P., Valencia, A., Himmelbauer, H., Himmelbaue, H., Quesada, V., Bea, S., Stratton, M. R., Futreal, P. A., Campbell, P. J., Vincent-Salomon, A., Richardson, A. L., Reis-Filho, J. S., van de Vijver, M., Thomas, G., Masson-Jacquemier, J. D., Aparicio, S., Borg, A., Brresen-Dale, A.-L., Caldas, C., Foekens, J. A., Stunnenberg, H. G (2010). International network of cancer genome projects. *Nature*, **464**(7291), 993–998.

International HapMap Consortium (2003). The international hapmap project. *Nature*, **426**(6968), 789–796.

Iyer, M. K., Chinnaiyan, A. M., and Maher, C. A. (2011). Chimerascan: a tool for identifying chimeric transcription in sequencing data. *Bioinformatics*, **27**(20), 2903–2904.

Ji, Y., Xu, Y., Zhang, Q., Tsui, K.-W., Yuan, Y., Norris, Jr, C., Liang, S., and Liang, H. (2011). Bm-map: Bayesian mapping of multireads for next-generation sequencing data. *Biometrics*, **67**(4), 1215–1224.

Jiang, H. and Wong, W. H. (2008). Seqmap: mapping massive amount of oligonucleotides to the genome. *Bioinformatics*, **24**(20), 2395–2396.

Johnson, D. S., Mortazavi, A., Myers, R. M., and Wold, B. (2007). Genome-wide mapping of in vivo protein-dna interactions. *Science*, **316**, 1497–1502.

Johnston, J. J., Teer, J. K., Cherukuri, P. F., Hansen, N. F., Loftus, S. K., N. I. H. Intramural Sequencing Center (NISC), Chong, K., Mullikin, J. C., and Biesecker, L. G. (2010). Massively parallel sequencing of exons on the x chromosome identifies rbm10 as the gene that causes a syndromic form of cleft palate. *Am J Hum Genet*, **86**(5), 743–748.

Jokinen, P. and Ukkonen, E. (1991). Two algorithms for approximate string matching in static texts. *Lecture Notes in Comp. Sci.*, **520**(06), 240–248.

Kalscheuer, V., Hu, H., Haas, S. A., Chelly, J., Esch, H. V., Raynaud, M., de Brouwer, A., Zemojtel, T., Weinert, S., Froyen, G., Frints, S. G., Laumonnier, F., Love, M. I., Richard, H., Emde, A.-K., Bienek, M., Jensen, C., Hambrock, M., Langnick, C., Feldkamp, M., Wissink-Lindhout, W., Lebrun, N., Castelnau, L., Shaw, M., Corbett, M. A., Gardner, A., Willis-Owen, S., Tan, C., Friend, K. L., Belet, S., van Roozendaal, K. E., Jimenez-Pocquet, M., Moizard, M.-P., Ronce, N., Sun, R., O'Keeffe, S., Chenna, R., Mysickova, A., Göke, J., Hackett, A., Field, M., Haan, E., Nelson, J., Turner, G., Baynam, G., Gillessen-Kaesbach, G., Müller, U., Steinberger, D., Budny, B., Badura-Stronka, M., Latos-Bielenska, A., Ousager, L. B., Wieacker, P., Criado, G. R., Bondeson, M.-L., Dufke, A., Cohen, M., Maldergem, L. V., Vincent-Delorme, C., Echenne, B., Simon-Bouy, B., Raymond, L., Kleefstra, T., Willemsen, M., Fryns, J.-P., Devriendt, K., Vingron, M., Wrogemann, K., Ullmann, R., Gecz, J., Tzschach, A., van Bokhoven, H., Wienker, T. F., Jentsch, T. J., Chen, W., and Ropers, H.-H. (submitted). Draining the pond: 14 novel candidate genes for x-linked intellectual disability.

Kalscheuer, V. M., Freude, K., Musante, L., Jensen, L. R., Yntema, H. G., Gécz, J., Sefiani, A., Hoffmann, K., Moser, B., Haas, S., *et al.* (2003). Mutations in the polyglutamine binding protein 1 gene cause x-linked mental retardation. *Nat Genet*, **35**(4), 313–315.

Karakoc, E., Alkan, C., O'Roak, B. J., Dennis, M. Y., Vives, L., Mark, K., Rieder, M. J., Nickerson, D. A., and Eichler, E. E. (2012). Detection of structural variants and indels within exome data. *Nat Methods*, **9**(2), 176–178.

Karlić, R., Chung, H.-R., Lasserre, J., Vlahovicek, K., and Vingron, M. (2010). Histone modification levels are predictive for gene expression. *Proc Natl Acad Sci U S A*, **107**(7), 2926–2931.

Kehr, B., Weese, D., and Reinert, K. (2011). Stellar: fast and exact local alignments. *BMC Bioinformatics*, **12 Suppl 9**, S15.

Kelley, D. R., Schatz, M. C., and Salzberg, S. L. (2010). Quake: quality-aware detection and correction of sequencing errors. *Genome Biol*, **11**(11), R116.

Kent, W. (2002). Blat – the blast-like alignment tool. *Genome Res.*, **12**(4), 656–64.

Kong, Y. (2011). Btrim: a fast, lightweight adapter and quality trimming program for next-generation sequencing technologies. *Genomics*, **98**(2), 152–153.

Korbel, J. O., Abyzov, A., Mu, X. J., Carriero, N., Cayting, P., Zhang, Z., Snyder, M., and Gerstein, M. B. (2009). Pemer: a computational framework with simulation-based error models for inferring genomic structural variants from massive paired-end sequencing data. *Genome Biol*, **10**(2), R23.

Krawitz, P., Rödelsperger, C., Jäger, M., Jostins, L., Bauer, S., and Robinson, P. N. (2010). Microindel detection in short-read sequence data. *Bioinformatics*, **26**(6), 722–729.

Kucherov, G., No, L., and Roytberg, M. (2005). Multiseed lossless filtration. *IEEE/ACM Trans Comput Biol Bioinform*, **2**(1), 51–61.

Lam, H. Y. K., Mu, X. J., Sttz, A. M., Tanzer, A., Cayting, P. D., Snyder, M., Kim, P. M., Korbel, J. O., and Gerstein, M. B. (2010). Nucleotide-resolution analysis of structural variants using breakseq and a breakpoint library. *Nat Biotechnol*, **28**(1), 47–55.

Lander, E. S., Linton, L. M., Birren, B., Nusbaum, C., Zody, M. C., Baldwin, J., Devon, K., Dewar, K., Doyle, M., FitzHugh, W., Funke, R., Gage, D., Harris, K., Heaford, A., Howland, J., Kann, L., Lehoczky, J., LeVine, R., McEwan, P., McKernan, K., Meldrim, J., Mesirov, J. P., Miranda, C., Morris, W., Naylor, J., Raymond, C., Rosetti, M., Santos, R., Sheridan, A., Sougnez, C., Stange-Thomann, N., Stojanovic, N., Subramanian, A., Wyman, D., Rogers, J., Sulston, J., Ainscough, R., Beck, S., Bentley, D., Burton, J., Clee, C., Carter, N., Coulson, A., Deadman, R., Deloukas, P., Dunham, A., Dunham, I., Durbin, R., French, L., Grafham, D., Gregory, S., Hubbard, T., Humphray, S., Hunt, A., Jones, M., Lloyd, C., McMurray, A., Matthews, L., Mercer, S., Milne, S., Mullikin, J. C., Mungall, A., Plumb, R., Ross, M., Shownkeen, R., Sims,

S., Waterston, R. H., Wilson, R. K., Hillier, L. W., McPherson, J. D., Marra, M. A., Mardis, E. R., Fulton, L. A., Chinwalla, A. T., Pepin, K. H., Gish, W. R., Chissoe, S. L., Wendl, M. C., Delehaunty, K. D., Miner, T. L., Delehaunty, A., Kramer, J. B., Cook, L. L., Fulton, R. S., Johnson, D. L., Minx, P. J., Clifton, S. W., Hawkins, T., Branscomb, E., Predki, P., Richardson, P., Wenning, S., Slezak, T., Doggett, N., Cheng, J. F., Olsen, A., Lucas, S., Elkin, C., Uberbacher, E., Frazier, M., Gibbs, R. A., Muzny, D. M., Scherer, S. E., Bouck, J. B., Sodergren, E. J., Worley, K. C., Rives, C. M., Gorrell, J. H., Metzker, M. L., Naylor, S. L., Kucherlapati, R. S., Nelson, D. L., Weinstock, G. M., Sakaki, Y., Fujiyama, A., Hattori, M., Yada, T., Toyoda, A., Itoh, T., Kawagoe, C., Watanabe, H., Totoki, Y., Taylor, T., Weissenbach, J., Heilig, R., Saurin, W., Artiguenave, F., Brottier, P., Bruls, T., Pelletier, E., Robert, C., Wincker, P., Smith, D. R., Doucette-Stamm, L., Rubenfield, M., Weinstock, K., Lee, H. M., Dubois, J., Rosenthal, A., Platzer, M., Nyakatura, G., Taudien, S., Rump, A., Yang, H., Yu, J., Wang, J., Huang, G., Gu, J., Hood, L., Rowen, L., Madan, A., Qin, S., Davis, R. W., Federspiel, N. A., Abola, A. P., Proctor, M. J., Myers, R. M., Schmutz, J., Dickson, M., Grimwood, J., Cox, D. R., Olson, M. V., Kaul, R., Raymond, C., Shimizu, N., Kawasaki, K., Minoshima, S., Evans, G. A., Athanasiou, M., Schultz, R., Roe, B. A., Chen, F., Pan, H., Ramser, J., Lehrach, H., Reinhardt, R., McCombie, W. R., de la Bastide, M., Dedhia, N., Blcker, H., Hornischer, K., Nordsiek, G., Agarwala, R., Aravind, L., Bailey, J. A., Bateman, A., Batzoglou, S., Birney, E., Bork, P., Brown, D. G., Burge, C. B., Cerutti, L., Chen, H. C., Church, D., Clamp, M., Copley, R. R., Doerks, T., Eddy, S. R., Eichler, E. E., Furey, T. S., Galagan, J., Gilbert, J. G., Harmon, C., Hayashizaki, Y., Haussler, D., Hermjakob, H., Hokamp, K., Jang, W., Johnson, L. S., Jones, T. A., Kasif, S., Kaspryzk, A., Kennedy, S., Kent, W. J., Kitts, P., Koonin, E. V., Korf, I., Kulp, D., Lancet, D., Lowe, T. M., McLysaght, A., Mikkelsen, T., Moran, J. V., Mulder, N., Pollara, V. J., Ponting, C. P., Schuler, G., Schultz, J., Slater, G., Smit, A. F., Stupka, E., Szustakowski, J., Thierry-Mieg, D., Thierry-Mieg, J., Wagner, L., Wallis, J., Wheeler, R., Williams, A., Wolf, Y. I., Wolfe, K. H., Yang, S. P., Yeh, R. F., Collins, F., Guyer, M. S., Peterson, J., Felsenfeld, A., Wetterstrand, K. A., Patrinos, A., Morgan, M. J., de Jong, P., Catanese, J. J., Osoegawa, K., Shizuya, H., Choi, S., Chen, Y. J., Szustakowki, J., and , I. H. G. S. C. (2001). Initial sequencing and analysis of the human genome. *Nature*, **409**(6822), 860–921.

Langmead, B., Trapnell, C., Pop, M., and Salzberg, S. (2009). Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biol*, **10**(3), R25.

Lee, S., Hormozdiari, F., Alkan, C., and Brudno, M. (2009). Modil: detecting small indels from clone-end sequencing with mixtures of distributions. *Nat Methods*, **6**(7), 473–474.

Lenski, C., Abidi, F., Meindl, A., Gibson, A., Platzer, M., Kooy, R. F., Lubs, H. A., Stevenson, R. E., Ramser, J., and Schwartz, C. E. (2004). Novel truncating mutations in the polyglutamine tract binding protein 1 gene (pqbp1) cause renpenning syndrome and x-linked mental retardation in another family with microcephaly. *Am J Hum Genet*, **74**(4), 777–780.

Li, H. (2011). A statistical framework for snp calling, mutation discovery, association mapping

and population genetical parameter estimation from sequencing data. *Bioinformatics*, **27**(21), 2987–2993.

Li, H. and Durbin, R. (2009). Fast and accurate short read alignment with burrows-wheeler transform. *Bioinformatics*, **25**(14), 1754–1760.

Li, H., Ruan, J., and Durbin, R. (2008a). Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Res.*, **18**(11), 1851–1858.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., and Durbin, R. (2009a). The sequence alignment/map format and samtools. *Bioinformatics*, **25**(16), 2078–2079.

Li, M., Ma, B., Kisman, D., and Tromp, J. (2003). Patternhunter II: highly sensitive and fast homology search. *Genome Inform.*, **14**, 164–175.

Li, R., Li, Y., Kristiansen, K., and Wang, J. (2008b). Soap: short oligonucleotide alignment program. *Bioinformatics*, **24**(5), 713–714.

Li, R., Yu, C., Li, Y., Lam, T.-W., Yiu, S.-M., Kristiansen, K., and Wang, J. (2009b). Soap2: an improved ultrafast tool for short read alignment. *Bioinformatics*, **25**(15), 1966–1967.

Li, Y., Zheng, H., Luo, R., Wu, H., Zhu, H., Li, R., Cao, H., Wu, B., Huang, S., Shao, H., Ma, H., Zhang, F., Feng, S., Zhang, W., Du, H., Tian, G., Li, J., Zhang, X., Li, S., Bolund, L., Kristiansen, K., de Smith, A. J., Blakemore, A. I. F., Coin, L. J. M., Yang, H., Wang, J., and Wang, J. (2011). Structural variation in two human genomes mapped at single-nucleotide resolution by whole genome de novo assembly. *Nat Biotechnol*, **29**(8), 723–730.

Lin, H., Zhang, Z., Zhang, M. Q., Ma, B., and Li, M. (2008). Zoom! zillions of oligos mapped. *Bioinformatics*, **24**(21), 2431–2437.

Lunter, G. (2007). Probabilistic whole-genome alignments reveal high indel rates in the human and mouse genomes. *Bioinformatics*, **23**(13), i289–i296.

Lupski, J. R., Reid, J. G., Gonzaga-Jauregui, C., Rio Deiros, D., Chen, D. C. Y., Nazareth, L., Bainbridge, M., Dinh, H., Jing, C., Wheeler, D. A., McGuire, A. L., Zhang, F., Stankiewicz, P., Halperin, J. J., Yang, C., Gehman, C., Guo, D., Irikat, R. K., Tom, W., Fantin, N. J., Muzny, D. M., and Gibbs, R. A. (2010). Whole-genome sequencing in a patient with charcot-marie-tooth neuropathy. *N Engl J Med*, **362**(13), 1181–1191.

Lyle, R., Prandini, P., Osoegawa, K., ten Hallers, B., Humphray, S., Zhu, B., Eyras, E., Castelo, R., Bird, C. P., Gagos, S., *et al.* (2007). Islands of euchromatin-like sequence and expressed polymorphic sequences within the short arm of human chromosome 21. *Genome Res*, **17**(11), 1690–1696.

Maher, C. A., Kumar-Sinha, C., Cao, X., Kalyana-Sundaram, S., Han, B., Jing, X., Sam, L., Barrette, T., Palanisamy, N., and Chinnaiyan, A. M. (2009). Transcriptome sequencing to detect gene fusions in cancer. *Nature*, **458**(7234), 97–101.

Manber, U. and Myers, E. (1993). Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, **22**(5), 935–948.

Manzini, G. and Ferragina, P. (2004). Engineering a lightweight suffix array construction algorithm. *Algorithmica*, **40**(1), 33–50.

Mardis, E. R. (2008a). The impact of next-generation sequencing technology on genetics. *Trends Genet*, **24**(3), 133–141.

Mardis, E. R. (2008b). Next-generation dna sequencing methods. *Annu Rev Genomics Hum Genet*, **9**, 387–402.

Medvedev, P., Stanciu, M., and Brudno, M. (2009). Computational methods for discovering structural variation with next-generation sequencing. *Nat Methods*, **6**(11 Suppl), S13–S20.

Mills, R. E., Pittard, W. S., Mullaney, J. M., Farooq, U., Creasy, T. H., Mahurkar, A. A., Kemeza, D. M., Strassler, D. S., Ponting, C. P., Webber, C., and Devine, S. E. (2011). Natural genetic variation caused by small insertions and deletions in the human genome. *Genome Res*, **21**(6), 830–839.

Mitelman, F., Johansson, B., and Mertens, F. (2007). The impact of translocations and gene fusions on cancer causation. *Nat Rev Cancer*, **7**(4), 233–245.

Morin, R. D., O'Connor, M. D., Griffith, M., Kuchenbauer, F., Delaney, A., Prabhu, A.-L., Zhao, Y., McDonald, H., Zeng, T., Hirst, M., Eaves, C. J., and Marra, M. A. (2008). Application of massively parallel sequencing to microrna profiling and discovery in human embryonic stem cells. *Genome Res.*, **18**, 610–621.

Mortazavi, A., Williams, B. A., McCue, K., Schaeffer, L., and Wold, B. (2008). Mapping and quantifying mammalian transcriptomes by rna-seq. *Nat Methods*, **5**(7), 621–628.

Muntoni, F., Torelli, S., and Ferlini, A. (2003). Dystrophin and mutations: one gene, several proteins, multiple phenotypes. *Lancet Neurol*, **2**(12), 731–740.

Myers, E. W. (1994). A sublinear algorithm for approximate keyword searching. *Algorithmica*, **12**(4-5), 345–374.

Myers, E. W. (1999). A fast bit-vector algorithm for approximate string matching based on dynamic programming. *JACM*, **46**(3), 395–415.

NCBI (2012). Entrez genome project.

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Molecular Biol.*, **48**, 443–453.

Ng, S. B., Turner, E. H., Robertson, P. D., Flygare, S. D., Bigham, A. W., Lee, C., Shaffer, T., Wong, M., Bhattacharjee, A., Eichler, E. E., *et al.* (2009). Targeted capture and massively parallel sequencing of 12 human exomes. *Nature*, **461**(7261), 272–276.

Ng, S. B., Bigham, A. W., Buckingham, K. J., Hannibal, M. C., McMillin, M. J., Gildersleeve, H. I., Beck, A. E., Tabor, H. K., Cooper, G. M., Mefford, H. C., Lee, C., Turner, E. H., Smith, J. D., Rieder, M. J., Yoshiura, K.-I., Matsumoto, N., Ohta, T., Niikawa, N., Nickerson, D. A., Bamshad, M. J., and Shendure, J. (2010a). Exome sequencing identifies mll2 mutations as a cause of kabuki syndrome. *Nat Genet*, **42**(9), 790–793.

Ng, S. B., Buckingham, K. J., Lee, C., Bigham, A. W., Tabor, H. K., Dent, K. M., Huff, C. D., Shannon, P. T., Jabs, E. W., Nickerson, D. A., Shendure, J., and Bamshad, M. J. (2010b). Exome sequencing identifies the cause of a mendelian disorder. *Nat Genet*, **42**(1), 30–35.

Nicolas, F. and Rivals, E. (2005). Hardness of optimal spaced seed design. In *Proceedings of CPM 2005*, LNCS, pages 187–209. Springer Verlag.

Nielsen, R., Paul, J. S., Albrechtsen, A., and Song, Y. S. (2011). Genotype and snp calling from next-generation sequencing data. *Nat Rev Genet*, **12**(6), 443–451.

Ning, Z., Cox, A. J., and Mullikin, J. C. (2001). Ssaha: a fast search method for large dna databases. *Genome Res*, **11**(10), 1725–1729.

O'Brien, S. J. (1994). Genetic and phylogenetic analyses of endangered species. *Annu Rev Genet*, **28**, 467–489.

Orr, H. T. and Zoghbi, H. Y. (2007). Trinucleotide repeat disorders. *Annu Rev Neurosci*, **30**, 575–621.

Park, P. J. (2009). Chip-seq: advantages and challenges of a maturing technology. *Nat Rev Genet*, **10**(10), 669–680.

Pop, M. and Salzberg, S. L. (2008). Bioinformatics challenges of new sequencing technology. *Trends Genet*, **24**(3), 142–149.

Qin, J., Li, R., Raes, J., Arumugam, M., Burgdorf, K. S., Manichanh, C., Nielsen, T., Pons, N., Levenez, F., Yamada, T., Mende, D. R., Li, J., Xu, J., Li, S., Li, D., Cao, J., Wang, B., Liang, H., Zheng, H., Xie, Y., Tap, J., Lepage, P., Bertalan, M., Batto, J.-M., Hansen, T., Le Paslier, D., Linneberg, A., Nielsen, H. B., Pelletier, E., Renault, P., Sicheritz-Ponten, T., Turner, K., Zhu, H., Yu, C., Li, S., Jian, M., Zhou, Y., Li, Y., Zhang, X., Li, S., Qin, N., Yang, H., Wang, J., Brunak, S., Dor, J., Guarner, F., Kristiansen, K., Pedersen, O., Parkhill, J., Weissenbach, J., , M. I. T. C., Bork, P., Ehrlich, S. D., and Wang, J. (2010). A human gut microbial gene catalogue established by metagenomic sequencing. *Nature*, **464**(7285), 59–65.

Rasmussen, K., Stoye, J., and Myers, G. (2005). Efficient q-gram filters for finding all epsilon-matches over a given length. In *Proceedings of the Nineth Conference on Research in Computational Molecular Biology*, pages 189–203. Springer.

Rausch, T., Koren, S., Denisov, G., Weese, D., Emde, A.-K., Döring, A., and Reinert, K. (2009). A consistency-based consensus algorithm for de novo and reference-guided sequence assembly of short reads. *Bioinformatics*, **9**(25), 1118–1124.

Redon, R., Ishikawa, S., Fitch, K. R., Feuk, L., Perry, G. H., Andrews, T. D., Fiegler, H., Shapero, M. H., Carson, A. R., Chen, W., Cho, E. K., Dallaire, S., Freeman, J. L., Gonzlez, J. R., Gratacs, M., Huang, J., Kalaitzopoulos, D., Komura, D., MacDonald, J. R., Marshall, C. R., Mei, R., Montgomery, L., Nishimura, K., Okamura, K., Shen, F., Somerville, M. J., Tchinda, J., Valsesia, A., Woodwark, C., Yang, F., Zhang, J., Zerjal, T., Zhang, J., Armengol, L., Conrad, D. F., Estivill, X., Tyler-Smith, C., Carter, N. P., Aburatani, H., Lee, C., Jones, K. W., Scherer, S. W., and Hurles, M. E. (2006). Global variation in copy number in the human genome. *Nature*, **444**(7118), 444–454.

Ren, B., Robert, F., Wyrick, J. J., Aparicio, O., Jennings, E. G., Simon, I., Zeitlinger, J., Schreiber, J., Hannett, N., Kanin, E., Volkert, T. L., Wilson, C. J., Bell, S. P., and Young, R. A. (2000). Genome-wide location and function of dna binding proteins. *Science*, **290**(5500), 2306–2309.

Rios, J., Stein, E., Shendure, J., Hobbs, H. H., and Cohen, J. C. (2010). Identification by whole-genome resequencing of gene defect responsible for severe hypercholesterolemia. *Hum Mol Genet*, **19**(22), 4313–4318.

Rothberg, J. M., Hinz, W., Rearick, T. M., Schultz, J., Mileski, W., Davey, M., Leamon, J. H., Johnson, K., Milgrew, M. J., Edwards, M., Hoon, J., Simons, J. F., Marran, D., Myers, J. W., Davidson, J. F., Branting, A., Nobile, J. R., Puc, B. P., Light, D., Clark, T. A., Huber, M., Branciforte, J. T., Stoner, I. B., Cawley, S. E., Lyons, M., Fu, Y., Homer, N., Sedova, M., Miao, X., Reed, B., Sabina, J., Feierstein, E., Schorn, M., Alanjary, M., Dimalanta, E., Dressman, D., Kasinskas, R., Sokolsky, T., Fidanza, J. A., Namsaraev, E., McKernan, K. J., Williams, A., Roth, G. T., and Bustillo, J. (2011). An integrated semiconductor device enabling non-optical genome sequencing. *Nature*, **475**(7356), 348–352.

Rumble, S. M., Lacroute, P., Dalca, A. V., Fiume, M., Sidow, A., and Brudno, M. (2009). Shrimp: Accurate mapping of short color-space reads. *PLoS Comput Biol*, **5**(5), e1000386.

Salmela, L. (2010). Correction of sequencing errors in a mixed set of reads. *Bioinformatics*, **26**(10), 1284–1290.

Sanger, F., Nicklen, S., and Coulson, A. R. (1977). DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*, **74**(12), 5463–5467.

Schadt, E. E., Turner, S., and Kasarskis, A. (2010). A window into third-generation sequencing. *Hum Mol Genet*, **19**(R2), R227–R240.

Schmieder, R., Lim, Y. W., Rohwer, F., and Edwards, R. (2010). Tagcleaner: Identification and removal of tag sequences from genomic and metagenomic datasets. *BMC Bioinformatics*, **11**, 341.

Schwartz, S. L. and Farman, M. L. (2010). Systematic overrepresentation of dna termini and underrepresentation of subterminal regions among sequencing templates prepared from hydrodynamically sheared linear dna molecules. *BMC Genomics*, **11**, 87.

SEQanswers (2012). Next-generation sequencing community forum.

Sherry, S. T., Ward, M. H., Kholodov, M., Baker, J., Phan, L., Smigielski, E. M., and Sirotkin, K. (2001). dbsnp: the ncbi database of genetic variation. *Nucleic Acids Res*, **29**(1), 308–311.

Shumway, M., Cochrane, G., and Sugawara, H. (2010). Archiving next generation sequencing data. *Nucleic Acids Res*, **38**(Database issue), D870–D871.

Simpson, J. T. and Durbin, R. (2012). Efficient de novo assembly of large genomes using compressed data structures. *Genome Res*, **22**(3), 549–556.

Smeds, L. and Künstner, A. (2011). Condetri–a content dependent read trimmer for illumina data. *PLoS One*, **6**(10), e26314.

Smit, A. F. A., Hubley, R., and Green, P. (1996-2004). Repeatmasker open-3.0. http://www.repeatmasker.org.

Smith, L. M., Sanders, J. Z., Kaiser, R. J., Hughes, P., Dodd, C., Connell, C. R., Heiner, C., Kent, S. B., and Hood, L. E. (1986). Fluorescence detection in automated dna sequence analysis. *Nature*, **321**(6071), 674–679.

Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *J. Molecular Biol.*, **147**, 195–197.

Stenson, P. D., Mort, M., Ball, E. V., Howells, K., Phillips, A. D., Thomas, N. S., and Cooper, D. N. (2009). The human gene mutation database: 2008 update. *Genome Med*, **1**(1), 13.

Sultan, M., Schulz, M. H., Richard, H., Magen, A., Klingenhoff, A., Scherf, M., Seifert, M., Borodina, T., Soldatov, A., Parkhomchuk, D., Schmidt, D., O'Keeffe, S., Haas, S., Vingron, M., Lehrach, H., and Yaspo, M.-L. (2008a). A global view of gene activity and alternative splicing by deep sequencing of the human transcriptome. *Science*, **321**(5891), 956–960.

Sultan, M., Schulz, M. H., Richard, H., Magen, A., Klingenhoff, A., Scherf, M., Seifert, M., Borodina, T., Soldatov, A., Parkhomchuk, D., Schmidt, D., O'Keeffe, S., Haas, S., Vingron, M., Lehrach, H., and Yaspo, M.-L. (2008b). A global view of gene activity and alternative splicing by deep sequencing of the human transcriptome. *Science*, **321**, 956–960.

Sun, R., Love, M. I., Zemojtel, T., Emde, A.-K., Chung, H.-R., Vingron, M., and Haas, S. A. (2012). Breakpointer: using local mapping artifacts to support sequence breakpoint discovery from single-end reads. *Bioinformatics*, **28**(7), 1024–1025.

Suzuki, S., Yasuda, T., Shiraishi, Y., Miyano, S., and Nagasaki, M. (2011). Clipcrop: a tool for detecting structural variations with single-base resolution using soft-clipping information. *BMC Bioinformatics*, **12 Suppl 14**, S7.

Trapnell, C., Pachter, L., and Salzberg, S. L. (2009). Tophat: discovering splice junctions with rna-seq. *Bioinformatics*, **25**(9), 1105–1111.

Trapnell, C., Roberts, A., Goff, L., Pertea, G., Kim, D., Kelley, D. R., Pimentel, H., Salzberg, S. L., Rinn, J. L., and Pachter, L. (2012). Differential gene and transcript expression analysis of rna-seq experiments with tophat and cufflinks. *Nat Protoc*, **7**(3), 562–578.

Trappe, K. (2012). *Multi-Split Mapping of NGS Reads for Variant Detection*. Master's thesis, Freie Universität Berlin.

Turner, E. H., Lee, C., Ng, S. B., Nickerson, D. A., and Shendure, J. (2009). Massively parallel exon capture and library-free resequencing across 16 genomes. *Nat Methods*, **6**(5), 315–316.

Tuzun, E., Sharp, A. J., Bailey, J. A., Kaul, R., Morrison, V. A., Pertz, L. M., Haugen, E., Hayden, H., Albertson, D., Pinkel, D., Olson, M. V., and Eichler, E. E. (2005). Fine-scale structural variation of the human genome. *Nat Genet*, **37**(7), 727–732.

Valouev, A., Johnson, D. S., Sundquist, A., Medina, C., Anton, E., Batzoglou, S., Myers, R. M., and Sidow, A. (2008). Genome-wide analysis of transcription factor binding sites based on chip-seq data. *Nat Methods*, **5**(9), 829–834.

Venter, J. C., ..., Reinert, K., *et al.* (2001). The sequence of the human genome. *Science*, **291**, 1145–1434.

Voelkerding, K. V., Dames, S. A., and Durtschi, J. D. (2009). Next-generation sequencing: from basic research to diagnostics. *Clin Chem*, **55**(4), 641–658.

Wang, K., Singh, D., Zeng, Z., Coleman, S. J., Huang, Y., Savich, G. L., He, X., Mieczkowski, P., Grimm, S. A., Perou, C. M., *et al.* (2010). Mapsplice: accurate mapping of rna-seq reads for splice junction discovery. *Nucleic Acids Res*, **38**(18), e178.

Wang, W.-C., Lin, F.-M., Chang, W.-C., Lin, K.-Y., Huang, H.-D., and Lin, N.-S. (2009a). mir-express: analyzing high-throughput sequencing data for profiling microrna expression. *BMC Bioinformatics*, **10**, 328.

Wang, Z., Gerstein, M., and Snyder, M. (2009b). Rna-seq: a revolutionary tool for transcriptomics. *Nat Rev Genet*, **10**(1), 57–63.

Wang, Z., Gerstein, M., and Snyder, M. (2009c). Rna-seq: a revolutionary tool for transcriptomics. *Nat Rev Genet*, **10**(1), 57–63.

Watson, J. D. and Crick, F. H. C. (1953). Molecular structure of nucleic acids. *Nature*.

Weese, D., Emde, A.-K., Rausch, T., Döring, A., and Reinert, K. (2009). RazerS–fast read mapping with sensitivity control. *Genome Res*, **19**(9), 1646–1654.

Weese, D., Holtgrewe, M., and Reinert, K. (2012). Razers 3: Faster, fully sensitive read mapping. *Bioinformatics*, **28**(20), 2592–2599.

Wheeler, D. A., Srinivasan, M., Egholm, M., Shen, Y., Chen, L., McGuire, A., He, W., Chen, Y.-J., Makhijani, V., Roth, G. T., *et al.* (2008). The complete genome of an individual by massively parallel dna sequencing. *Nature*, **452**(7189), 872–876.

Whiteford, N., Haslam, N., Weber, G., Prgel-Bennett, A., Essex, J. W., Roach, P. L., Bradley, M., and Neylon, C. (2005). An analysis of the feasibility of short read sequencing. *Nucleic Acids Res*, **33**(19), e171.

Wu, T. D. and Nacu, S. (2010). Fast and snp-tolerant detection of complex variants and splicing in short reads. *Bioinformatics*, **26**(7), 873–881.

Xie, C. and Tammi, M. T. (2009). Cnv-seq, a new method to detect copy number variation using high-throughput sequencing. *BMC Bioinformatics*, **10**, 80.

Yang, X., Chockalingam, S. P., and Aluru, S. (2012). A survey of error-correction methods for next-generation sequencing. *Brief Bioinform*.

Ye, K., Schulz, M. H., Long, Q., Apweiler, R., and Ning, Z. (2009). Pindel: a pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads. *Bioinformatics*, **25**(21), 2865–2871.

Yoon, S., Xuan, Z., Makarov, V., Ye, K., and Sebat, J. (2009). Sensitive and accurate detection of copy number variants using read depth of coverage. *Genome Res*, **19**(9), 1586–1592.

Zang, C., Schones, D. E., Zeng, C., Cui, K., Zhao, K., and Peng, W. (2009). A clustering approach for identification of enriched domains from histone modification chip-seq data. *Bioinformatics*, **25**(15), 1952–1958.

Zerbino, D. R. and Birney, E. (2008). Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Res*, **18**(5), 821–829.

Zhang, J., Wang, J., and Wu, Y. (2012). An improved approach for accurate and efficient calling of structural variations with low-coverage sequence data. *BMC Bioinformatics*, **13 Suppl 6**, S6.

Zhang, L., Miles, M. F., and Aldape, K. D. (2003). A model of molecular interactions on short oligonucleotide microarrays. *Nat Biotechnol*, **21**(7), 818–821.

Zhang, Y., Liu, T., Meyer, C. A., Eeckhoute, J., Johnson, D. S., Bernstein, B. E., Nusbaum, C., Myers, R. M., Brown, M., Li, W., and Liu, X. S. (2008a). Model-based analysis of chip-seq (macs). *Genome Biol*, **9**(9), R137.

Zhang, Z., Berman, P., Wiehe, T., and Miller, W. (1999). Post-processing long pairwise alignments. *Bioinformatics*, **15**(12), 1012–1019.

Zhang, Z., Schwartz, S., Wagner, L., and Miller, W. (2000). A greedy algorithm for aligning dna sequences. *J Comput Biol*, **7**(1-2), 203–214.

Zhang, Z.-F., Ruivenkamp, C., Staaf, J., Zhu, H., Barbaro, M., Petillo, D., Khoo, S. K., Borg, A., Fan, Y.-S., and Schoumans, J. (2008b). Detection of submicroscopic constitutional chromosome aberrations in clinical diagnostics: a validation of the practical performance of different array platforms. *Eur J Hum Genet*, **16**(7), 786–792.

Selbstständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Berlin, den 30.10.2012 (Anne-Katrin Emde)

Curriculum Vitae

For reasons of data protection, the Curriculum vitae is not published in the online version.