An Integrated Data Analysis Suite and Programming Framework for High-Throughput DNA Sequencing

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Dipl.-Inform. Felix Ott
aus Basel

Tübingen
2013

Tag der mündlichen Qualifikation:	18.12.2013
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Detlef Weigel
2. Berichterstatter:	Prof. Dr. Daniel Huson

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig und nur mit den angegebenen Hilfsmitteln angefertigt habe, und dass alle Stellen, die im Wortlaut oder dem Sinn nach anderen Werken entnommen sind, durch Angabe der Quellen kenntlich gemacht sind.

Beiträge Dritter, auf die im Text eingegangen wird, sind auf der Seite *Contributions* aufgelistet. Des Weiteren wird auf thematische Überschneidungen mit Publikationen, an denen ich mitgewirkt habe, im Text explizit hingewiesen. Weitere Überschneidungen mit zukünftigen Publikationen sind möglich.

Tübingen, 19. Dezember 2013 Felix Ott

Contributions

The high-throughput DNA sequencing data analysis pipeline SHORE that this work is built around is the work of a variety of contributors. The foundations to the software were laid by Stephan Ossowski and Korbinian Schneeberger, and are meticulously described in Stephan Ossowski's Ph. D. thesis "*Computational Approaches for Next Generation Sequencing Analysis and MiRNA Target Search*" [1]. Concerned with a direct continuation of this work are sections 2.1, 2.3 and 2.6. The prerequisites are however discussed explicitly, and further delineation is provided in section 1.7. Jörg Hagmann has supported the development with ideas and has contributed a multitude of improvements as well as modules for RNA editing analysis, bisulfite sequencing data analysis and multi-reference variation calling, which are however not subject of this work. Jonas Müller has contributed RAD-Seq motivated extensions to the read demultiplexing code, which were incorporated into the methods described in section 2.4. Alf Scotland has supported parts of the coding effort, and Sebastian Bender has contributed further extensions to the software pipeline not discussed in this work.

In accordance with standard scientific protocol, first-person plural pronouns will be used to refer to the reader and the writer, or to my scientific collaborators and myself.

Acknowledgment

First of all I would like to thank my supervisors Detlef Weigel and Daniel Huson for supporting my thesis and the opportunity to work on countless interesting and challenging questions at the Max-Planck-Institut Tübingen. Further I am grateful to Steffen Schmidt and Karsten Borgwardt, for helpful and encouraging advice given over the years in their role as members of my Thesis Advisory Committee.

Jörg Hagmann, Stephan Ossowski and Korbinian Schneeberger must be singled out, but I am grateful for the excellent collaboration with all other contributors to the SHORE software. Norman Warthmann, Jun Cao, Sang-Tae Kim, Stefan Henz out of many others have been of great help by using, testing and suggesting improvements to the application in various areas.

Markus Schmid, Levi Yant, David Posé Padilla, François Parcy, Edwige Moyroud, Stephan Wenkel, Ronny Brandt, Richard Immink and others deserve my gratitude for excellent ChIP-Seq collaborations, discussions and explanations.

There are plenty of other people I would like to thank for outstanding professional collaborations, advice or interesting discussions and input. Among them are Lisa Smith, Brandon Gaut, Jesse Hollister, Xi Wang, Marco Todesco, Felipe Fenselau de Felippes, Pablo Manavella, Ignacio Rubio Somoza and Christa Lanz. To Rebecca Schwab, Jörg Hagmann and Jonas Müller I am immensely grateful for doing a great job proofreading this thesis.

Finally, I apologize to those I forgot.

Zusammenfassung

Die Dideoxynukleotid-Kettenabbruchmethode wird seit dem Jahr 2005 durch eine Vielzahl an parallelen DNA-Sequenzierungsmethoden ergänzt. Diese haben sich als eine Schlüsseltechnologie mit einer großen Anzahl von Einsatzmöglichkeiten in der Genetik erwiesen, stellen andererseits mit einer Flut an erzeugten Daten auch eine Herausforderung dar. Eine Vielfalt an Algorithmen, die sich mit unterschiedlichen Aspekten der Sequenzierdaten-Verarbeitung und Analyse befassen, wurde bereits erarbeitet und implementiert. Zur routinemäßigen Analyse dieser Daten benötigt es jedoch zudem eine optimal aufeinander abgestimmte Sammlung von Analysewerkzeugen, die alle notwendigen Schritte von der initialen Rohdatenverarbeitung bis zum Erlangen der primären Analyseergebnisse abdeckt.

Mit der Software SHORE wird eine modulare, vielfältig nutzbare Datenanalyse-Lösung für die parallele DNA-Sequenzierung bereitgestellt. In der vorliegenden Arbeit werden die SHORE zugrunde liegenden Konzepte erweitert und verallgemeinert, um den Entwicklungen der neuen Sequenziermethoden Rechnung zu tragen. Diese Entwicklungen schließen unter anderem einen deutlich erhöhten Datenertrag ein, sowie die Verbreitung von Protokollen, die das Multiplexen von Proben mittels spezieller Kennzeichnungssequenzen ermöglichen. Um eine breitgefächerte Unterstützung grundlegender Funktionen wie Datenkompression und indexierter Suchalgorithmen zu ermöglichen, wurde eine allgemein nutzbare C++-Programmierumgebung *libshore* entwickelt, die als gemeinsame Basis aller Datenverarbeitungsmodule in SHORE fungiert. Ein weiteres Ziel war hierbei, Programmierschnittstellen bereitzustellen, die modulares Design sowie Parallelisierung von Sequenzierdaten-Verarbeitungsalgorithmen unterstützen.

Ein weiterer Schwerpunkt dieser Arbeit war die Entwicklung von Analysealgorithmen für mittels der Chromatin-Immunopräzipitation gewonnener Daten. Eine Schlüsselrolle in der Regulierung der DNA-Transkription wird von einer Klasse von DNA-bindenden Proteinen, den Transkriptionsfaktoren, eingenommen. ChIP-Seq-Studien erlauben die Darstellung von Transkriptionsfaktor-Bindestellen für das gesamte Genom, und besitzen daher großes Potential, zum Verständnis der Funktionsweise der komplexen regulatorischen Netzwerke beizutragen. In dieser Arbeit wird ein Analysemodul SHORE *peak* vorgestellt, welches zur Verarbeitung von Transkriptionsfaktor-Immunopräzipitationsdaten dient. Das Computerprogramm vereint statistische Detektion angereicherter Sequenzabschnitte mit empirischen Regeln zur Ausfilterung von Artefakten, um so die zuverlässige Erkennung der entscheidenden Bereiche des Genoms zu gewährleisten. Da Wiederholungsexperimente, die durch fallende Sequenzierungskosten verstärkt ermöglicht werden, ein wichtiges Mittel zur Identifizierung der biologisch relevanten Sequenzbereiche darstellen, wurde zudem besonderer Wert auf die simultane Auswertung dieser Datensätze gelegt.

Die entwickelten Algorithmen wurden zudem auf weitere Analysemodule übertragen, die mit dem Ziel der möglichst flexiblen Konfigurierbarkeit entwickelt wurden. In Kombination mit der bereits enthaltenen Funktionalität zur Detektion genomischer Varianten sollte die Software SHORE von Nutzen für einen großen Teil des Anwendungsbereiches der neuen DNA-Sequenzierungstechnologien sein. Mit der allgemein nutzbaren Funktionalität der *libshore*-Programmierumgebung sollte zudem die einfache Erweiterbarkeit im Hinblick auf zukünftige Ansätze zur Datenanalyse gewährleistet sein.

Abstract

The various parallel DNA sequencing methods that have been introduced since 2005 to complement the established dideoxynucleotide chain termination method have proved a key technology with a wide range of applications in genetic research, but also challenge with a constant flood of data. A multitude of algorithms have been proposed and implemented addressing different aspects of sequencing data processing and data analysis for a variety of high-throughput sequencing applications. Routine data analysis however requires a consistently assembled tool chain to ensure smooth end-to-end processing starting from raw sequencing read data up to primary analysis results.

The software SHORE aims to be a modular, general-purpose data analysis suite for high-throughput DNA sequencing. With this work, we extend and generalize the concepts of the SHORE analysis pipeline to accommodate increased throughput of sequencing devices and development of novel sequencing protocols such as bar-coded sample multiplexing. To provide universal support of basic features such as data set compression or indexing and query mechanisms, we develop a generic C++ programming framework *libshore* to form the foundation of all of SHORE's modules. Furthermore, we aim to provide a generic application programming interface facilitating modular design as well as parallelization of high-throughput sequencing data processing and analysis algorithms.

A further focus of this work was on the development of data analysis algorithms for chromatin immunoprecipitation and other sequence enrichment and expression profiling studies. Through genome-wide binding-site profiling for transcription factors, DNA-binding proteins occupying a key role in transcription regulation, ChIP-Seq studies have the potential to greatly improve the ability to understand the functioning of complex regulatory networks. We present an analysis module SHORE*peak* oriented towards processing of transcription factor immunoprecipitation data. Our program combines statistical enrichment detection with empirical artifact removal rules to ensure robust identification of the most relevant sites. As replicate experiments are an important factor for the identification of biologically relevant sites and with dropping sequencing cost are rendered more feasible, emphasis was put on the simultaneous processing of such data sets.

By transferring enrichment detection and expression analysis algorithms to further auxiliary modules emphasizing flexibility of configuration, as well as maintaining previously available variation analysis functionality, SHORE should represent a valuable resource for the majority of high-throughput sequencing applications. Furthermore, in combination with the generic functionality of the *libshore* framework, we hope to ensure extensibility to readily accommodate future analysis strategies.

Contents

1 Introduction

Next-Generation Sequencing *has evolved into a powerful tool for many areas of biological science, but has also introduced many new challenges related to data analysis and computing infrastructure into the field. Following a brief recapitulation of high-throughput DNA sequencing technology, this chapter highlights important areas of application while discussing previous and related work. We outline general sequencing data analysis methodology and conclude by establishing the motivation for this work.*

1.1 High-Throughput DNA Sequencing

Widely deployed for less than a decade, the impact of parallel next-generation sequencing technology on genetic research has already been considerable. The novel *high-throughput sequencing* approaches are characterized by massive parallelism implemented in a single device. From this design results the key advantage of a significantly reduced cost per sequenced base, which has allowed to quickly displace the previously predominant Sanger sequencing method in many areas of application.

Sanger sequencing, the prevalent sequencing method for the majority of time since its introduction in 1977 [2], needed to rely on extensive automation in dedicated sequencing centers to achieve time and cost effective readout of large amounts of sequence. Most prominently, this was put into effect during the effort of generating the first draft sequence of the human genome [3–6]. By contrast, high-throughput sequencing instruments are designed to analyze thousands to millions of DNA molecules simultaneously, and thus enable even smaller institutions to produce large quantities of sequence data on site.

Secondary to elucidation of DNA primary structure, sequencing utilized as a random sampling device delivers quantitative clues on sample composition. In this capacity, it has been used for diverse purposes such as inference of DNA methylation levels [7] or the analysis of environmental samples [8]. In concert with the economical advantage over the dideoxynucleotide chain-termination method, parallel DNA sequencing becoming widely accessible to researchers has served to promote such *deep sequencing* approaches that open up whole new areas of application beyond the domain previously occupied by Sanger sequencing.

As an alternative to microarray technologies, for example in gene expression profiling or chromatin immunoprecipitation assays, deep sequencing overcomes fundamental technological restrictions such as probe resolution and probe saturation. Furthermore, without the inherent requirement of a-priori knowledge of sequences to be detected or quantified, development of novel protocols of application has been furthered to transform high-throughput sequencing methods into a versatile tool for research.

1.2 Approaches to Parallel DNA Sequencing

An early parallel sequencing method, *massively parallel signature sequencing* (MPSS), was published in the year 2000 [9] and had already been available as a centralized, commercial service. However, as its short read length of only 17 to 20 nucleotides proved to be prohibitive in most potential areas of application, MPSS mainly found use as a gene expression profiling assay.

A technological and commercial breakthrough was marked by the deployment of integrated bench top devices through a variety of different providers. The first high-throughput sequencing instrument brought to the market was the 454 Life Sciences *Genome Sequencer FLX* in 2005 [10]. Soon after, in 2006, followed Illumina's *Genome Analyzer* [11] instrument and the Life Technology *SOLiD* system [12] in 2008.

Over a short period of time, considerable technological maturation has ensued, and vendors have started to broaden their portfolios by tailoring their respective solutions to different operational scenarios, exemplified by the Illumina *HiSeq* and *MiSeq* devices or the 454 Life Sciences bench top model *GS Junior*. On the other hand, diverse alternative approaches have been proposed and introduced into the market, notably a semiconductor-based solution marketed as *Ion Torrent* by Life Technology [13] and the Pacific Biosciences single molecule sequencing system *PacBio RS* [14].

As their defining feature, all high-throughput sequencing systems share the parallel interrogation of large numbers of DNA molecules. Sample DNA to be analyzed is typically applied to an expendable *flow cell* or *flow chip*, a specialized glass slide or microtiter plate particular to the respective technology. Aside from such fundamental analogies, the respective methods of sequence interrogation differ in key aspects. The basis of most current technologies is formed by the sequencing-by-synthesis (SBS) principle, with the notable exception of the SOLiD sequencing-by-ligation method. Sequencing-by-synthesis decodes the sequence of DNA molecules by keeping track of nucleotides incorporated by a DNA polymerase during complementary strand synthesis, with the distinguishing feature of the different technological implementations being the manifold methods employed for detecting events of nucleotide incorporation.

The 454 family of sequencers realize a *pyrosequencing* approach. The four deoxyribonucleoside triphosphates (dNTP) adenine, cytosine, guanine and thymine are flowed cyclically over microtiter wells holding clonally amplified DNA templates, where each flow of reagents delivers a specific type of dNTP. The amount of pyrophosphate released during nucleotide incorporation, which is determined by the number of nucleotides incorporated in each flow, is converted into light intensity through an enzymatic reaction and recorded by a CCD camera. [10]

The Illumina systems are based on reversible chain termination chemistry. A mixture of all four dNTPs is flowed over a slide with randomly distributed clusters of clonally amplified templates. The dNTPs are modified by a chain terminator, limiting elongation of the synthesized strand to a single nucleotide. Furthermore, each type of dNTP is distinguished by fusion to a specific fluorophore, allowing for identification of the respective incorporated base by means of laser excitation and a CCD detector. Reagents subsequently applied displace the reversible terminator and dye to enable further strand elongation in the following cycle. [11]

The Ion Torrent design adopts the homopolymer detection approach introduced by pyrosequencing. However, instead of pyrophosphate, the release of hydrogen ions serves as a proxy for strand elongation. The individual pH measurements are realized non-optically on an expendable semiconductor chip. [13]

PacBio RS sequencers implement a technology termed *single molecule real-time sequencing* (SMRT). The method is enabled by zero-mode waveguides (ZMW), nanostructures that enable probing of volumes smaller than the wave length of light [15]. In contrast to the other current technologies described, the SMRT approach therefore does not need to rely on clonally amplified

DNA templates, and instead single DNA molecules and polymerases suffice for obtaining a recognizable signal. DNA polymerases are immobilized in ZMW detection volumes, creating a slide suitable for the parallel observation of polymerase activity. Fluorescently labeled dNTPs are employed as the substrate, where nucleotides being incorporated are detected via laser excitation and a CCD detector. The fluorophore itself is displaced upon nucleotide incorporation. [14]

As opposed to sequencing-by-synthesis reliant on DNA polymerase, SOLiD sequencing constitutes a sequencing-by-ligation technique that exploits DNA ligase enzymes for sequence decoding. Probe oligomers partially consisting of degenerate and universal bases [16] are ligated to interrogate di-nucleotides at a spacing of three base pairs. When reaching the full read length, the template is reset and the ligation process is repeated five times at different offsets, and thus each of the template's bases is measured twice through different probe oligomers. Four different fluorophores serve to label the 16 types of probe to obtain an ambiguous *color space* code. The properties of the dibase color encoding are such that for any base, the ambiguity may be resolved as long as the identity of the preceding base is known. [12]

The following discussion centers on Illumina sequencing technology as the primary focus of the present work, although applicable — to varying extent — to multiple or all of the currently available high-throughput sequencing devices.

1.3 Applications of High-Throughput Sequencing

While high-throughput sequencing devices in general excel in the cost-effective generation of massive amounts of sequence data, the new technologies are still associated with weaknesses regarding error rates and maximal achievable sequencing read length (section 1.4). These weaknesses limit the capacity to generate base-by-base readouts of entire genome sequences. Nonetheless, concerted application of complementary technologies including Sanger sequencing has served to greatly accelerate the pace at which novel reference assemblies for manifold model organisms are being generated.

Certainly, despite the reduction in cost-per-base that has been achieved, decoding larger, repeat-rich genome sequences remains a massive task with current DNA sequencing technology. Genomic inter-individual and inter-species variation may however be assessed circumventing the issue of global-scale sequence topology (section 1.3.1), which has thus evolved as one of the primary applications of high-throughput sequencing. In turn, interpretation of data generated with the aim of such consensus-based characterization of sequence variation is greatly facilitated by the availability of a suitable reference genome assembly.

Similarly, the *resequencing* scenario facilitates data interpretation for an extensive family of deep sequencing and genome-wide screening applications. As throughput of sequencing devices has grown to enable routine sequencing even of large genomes to multiple depths of coverage, its potential for quantitative measurement has been leveraged for the assessment of diverse genomic properties. With stock genome sequencing protocol, deep sequencing allows estimation of quantifiable statistics such as gene copy number [17–19], application to forward genetic screens [20, 21] or dissection of environmental samples [22–24]. To exploit its potential for quantification in a wide range of further research settings, high-throughput sequencing is complemented by a host of different sample and library preparation protocols acting as adapters between the property of interest and DNA sequencing technology.

RNA conversion protocols facilitate investigation of transcript structure and abundance [25–29] and address structure, expression and distribution of specific types of transcript such as long non-coding RNA or small RNA [27, 30–33]. In ChIP-Seq, MeDIP-Seq or HITS-CLIP, immunoprecipitation-based sequence enrichment protocols provide insight into diverse genetic and epigenetic mechanisms including transcription factor binding events, histone and DNA modifications or specificity of RNA binding proteins [17, 34–38].

Conversely, depletion protocols such as MNase-Seq are for example applicable to investigation of nucleosome positioning [39]. An additional option for the study of epigenetic features by high-throughput sequencing is available in sequence conversion protocols utilizing bisulfite treatment of the sample DNA (BS-Seq) [27, 40, 41].

Further protocols aiming at a reduction of library complexity can supply significantly reduced cost of sequencing for certain applications, such as retrieval of genetic markers through sequencing of restriction site associated DNA (RAD-Seq) [42, 43].

By standard sample preparation protocols, a molecular mixture retrieved from millions of cells is assessed. This property is exploited deliberately for example by genomic enrichment or epigenetic sampling protocols such as ChIP-Seq as well as BS-Seq. On the other hand, single-cell sequencing protocols are being developed to enable the maximally fine-grained assessment of cellular populations [44].

While each sequencing protocol has on its own furthered our understanding of the molecular makeup of cells, integration and correlation of the different types and sources of data now available shows great promise to provide detailed insight into the complex interplay of the various mechanisms governing the cellular machinery.

1.3.1 Genome and Transcriptome Sequencing

Molecular genetics studies how an organism's genome interacts with its cellular environment, ultimately shaping the organism's phenotype inside its habitat. Learning how this interaction can be influenced may ultimately help to develop new capabilities to counteract diseases or enhance traits like disease resistance or crop yield. Analysis of the DNA-level differences between organisms, regardless of whether they are of the same species or not, presents an entry point to obtaining clues on this interaction. With the emergence of next-generation sequencing, for the first time researchers are able to employ this approach on the very large scale, exemplified by a variety of major sequencing projects both on the intra- and inter-species level like the 1000 genomes project [45, 46] for the human genome, the 1001 genomes project [19] for the *Arabidopsis thaliana* genome, the Genome 10K project [47] with the goal of analyzing 10000 vertebrate genomes or the metagenomic human microbiome project [48, 49].

Due to the short lengths of the sequence reads recovered by most high-throughput sequencers, most studies circumvent the complex task of whole genome assembly focusing on small scale sequence variants or localized larger scale rearrangements (section 1.5.4). These genetic markers thus recovered can be processed further allowing application to a wide range of scientific questions. Recovered on population scale, they provide insight into a species' population structure and the evolution of its genomes and of its subpopulations. When integrated with phenotype data, functional clues may further be obtained by means of genome wide association studies (GWAS) (e. g. [50]).

Knowledge of gene expression profiles proves valuable in many different contexts, and may on the one hand provide a means for the classification of entire cells, as well as on the other hand of certain genes or groups of genes with respect to their involvement in specific cellular pathways or function. While gene expression has been routinely assessed on the large scale using microarray technology, the method suffers from non-linear measurement, probe saturation and suboptimal signal-to-noise ratio.

High-throughput sequencing by contrast allows basically unlimited dynamic range only bounded by allocatable effort and amounts of cellular material. Other than microarrays, sequencing further enables recovery and quantification of unexpected transcripts and splice forms.

A variety of different transcriptome sequencing protocols are available, distinguished by their suitability for assessment of transcript structure and quantification as well as capabilities to detect certain types of transcript and to recover transcript directionality. As with all examples of quantification by sequencing, data interpretation may be intricate (section 1.5.5).

A special class of transcript is given by small RNA, short cellular RNA molecules ∼20–25 nucleotides in length ubiquitous in almost all well-studied eukaryotes. These have been identified as another vital factor for transcript regulation, and are further implicated in various important cellular processes such as DNA methylation and pathogen response [51, 52]. Transcript levels are affected by small RNA through RNA interference (RNAi). As the specificity-determining component of RNA induced silencing complexes (RISC), they direct the degrading enzymatic activity of ARGONAUTE proteins at the targeted, roughly complementary transcripts.

Small RNA molecules are attributed to multiple cellular pathways in plants and animals generally involving double stranded RNA or RNA stem-loop structures. According to the generating biochemical pathway, small RNA can be categorized into different classes, notably microRNA (miRNA) and small interfering RNA (siRNA), linked with overlapping, but distinct cellular roles.

High-throughput small RNA profiling by sequencing facilitates genome-wide discovery, categorization and relative quantification for the various small RNA species. However, small RNA quantification is sensitive to sequence specific bias [53], with their short length likely introducing increased variance to base composition bias (section 1.4) and emphasizing the importance of ligation biases. Furthermore, relative quantification with reference to library size is often deemed unsuitable for an investigation, and consensus on appropriate normalizing transformations is currently lacking (cf. [54]). Nonetheless, small RNA sequencing data prove a valuable quantitative resource on the genome scale, e. g. as a proxy for RNA-directed DNA methylation [55] or in correlation with further transcriptome or epigenetic data.

1.3.2 ChIP-Seq and further Immunoprecipitation Protocols

Expression of genes is thought to be connected in complex regulatory networks of positive and negative feedback. A key role in these networks is occupied by transcription factors, DNA-binding proteins involved with initiation of gene transcription at the promoter sequences. ChIP-Seq, the successor of the microarray-based ChIP-chip technology featuring improved signal-to-noise ratio and sharper signal bounds, facilitates the identification of putative transcription factor targets in a relevant in-vivo context. By elucidation of individual transcription factors' targets, our understanding of the complex transcriptional interplay can be gradually improved.

Immunoprecipitation (IP) methods exploit the antibody-antigen affinity to enrich specific molecules in a solution. In the course of the procedure, antibodies for the antigen to be concentrated are incubated with the solution to promote formation of antibody-antigen complexes. Antibodies are captured on solid-phase beads, thus enabling precipitation to enrich for the formed complexes.

Chromatin immunoprecipitation followed by high-throughput sequencing (ChIP-Seq) employs the IP technique for in-vivo detection of regions of co-localization of a specific molecule, usually a protein, with the genomic DNA. During the procedure, live tissue is prepared with formaldehyde or an appropriate alternative cross-linking reagent. Formaldehyde undergoes a reaction that induces cross-linking of amino groups in close vicinity, thereby promoting formation of DNA-protein complexes (DPC) (cf. Barker et al. [56]). Following lysis of the tissue, the DNA is sonicated, yielding DPC associated with only short fragments of genomic DNA. The DPC are immunoprecipitated and, by purification and amplification of the contained DNA, prepared as

Instrument	Max. Average Read Length	Read Pairs / Flowcell	Run Time
Illumina MiSeq	2x250bp	16M	39h
Illumina Genome Analyzer IIx	2x150bp	300M	14 days
Illumina HiSeq	2x100bp	1400M	11 days
454 Life Sciences GS Junior	∼400bp	100000	10h
454 Life Sciences GS FLX+	∼700bp	1M	23h
Life Technologies Ion PGM	up to 400bp	100000–5M	3.7h–7.3h
Life Technologies Ion Proton	up to 200bp	∼70M	2h–4h
Life Technologies 5500W (SOLiD)	1x75bp / 2x50bp	∼1600M	6 days
Pacific Biosciences PacBio RS	∼4.5kbp	∼22000	2h

Table 1.1: DNA Sequencer Specifications According to Device Vendors (3/2013)

a sequencing library. Following deep sequencing, thus precipitated tags facilitate identification and localization of putative protein binding sites.

Immunoprecipitation is complemented by techniques that seek to deplete segments of the DNA that are not protein-bound, e. g. using exonucleases like micrococcal nuclease (MNase) to digest the unbound parts of the DNA fragments following cross-linking and sonication. MNase-Seq is used on its own to investigate nucleosome positioning [39, 57], but can also be combined with ChIP-Seq for improved binding site resolution.

When available, immunoprecipitation of the DPC is achieved using a specific antibody raised against the protein of interest. Alternatively, the experiment is performed using transgenic organisms where the protein is tagged by an appropriate adapter protein, e. g. GFP [58], for which a high quality antibody is available. However, this fusion protein approach comes with the drawback of possible tag-induced alteration of protein function or binding affinity.

Apart from its application to the identification of transcription factor targets [34, 35], ChIP-Seq is most prominently applied for genome-wide screening of histone modifications [36]. Using 5-methylcytosine-specific antibodies, immunoprecipitation is further applicable for generating genome-wide maps of DNA methylation (MeDIP-Seq) [37].

RNA analogues to the ChIP-Seq strategy, *high-throughput sequencing of RNA isolated by crosslinking immunoprecipitation* (HITS-CLIP) [38] and *photoactivatable-ribonucleoside-enhanced crosslinking and immunoprecipitation* (PAR-CLIP) [59] are used in a similar fashion to investigate in-vivo binding of RNA associates like e. g. the miRNA binding protein ARGONAUTE.

1.4 Properties of Sequencing Data

Modifications to both the instruments themselves as well as to the sequencing chemistry utilized have led to vast improvements in per-run output since the first iteration of next generation sequencers, which has frequently drawn comparisons to the famous Moore's Law of the development of integrated circuits [60, 61].

A further reduction of overall sequencing cost remains a primary objective in the further development of the technologies, often illustrated by the catch phrase *1000 dollar* [human] *genome* [13, 62]. Efforts directed at this goal include the ongoing pursuit of new sequencing technologies as well as streamlining key operational aspects such as ease of library preparation, run time and quantization of throughput — i. e. the minimal amount of sequence that must be generated per sequencing run to achieve acceptable cost-per-base. However, with dropping cost-per-base further differentiated considerations including data handling and analysis cost as well as the unique capabilities of either respective technology are increasingly moving into focus.

Differences between the various technologies also imply different behavior regarding important parameters such as read length, frequency, type and randomness of sequencing errors or sequence dependent depth-of-coverage biases. With the targeted field of application, the impact of either property varies. Table 1.1 provides a comparison of read length, throughput and run time of sequencing instruments. Despite specifications of the young technologies being in constant flux, the overview highlights the technology's strengths and weaknesses discussed in the following.

Reversible chain terminator as well as sequencing-by-ligation approaches enforce in-phase measurement of all DNA templates being sequenced. Therefore, these devices provide a fixed, user selectable read length. Albeit currently providing the highest level of parallelism and of per-run throughput in terms of sequenced bases, they offer a significantly shorter maximal read length compared to the other methods discussed.

Sequencing-by-synthesis methods that as in the case of 454 and Ion Torrent instruments imply measurement of the homopolymer sequence of the respective DNA templates generate sequence reads of variable length determined by the actual nucleotide composition of the molecules. Whereas minimum read length equals the number of sequencing cycles, i. e. one quarter of the number of dNTP flows, the average read length thus depends on the type of DNA being sequenced.

By contrast, no phasing of strand elongation is enforced by the single molecule real time sequencing approach, neither on single base nor on the homopolymer level. The method instead attempts near-continuous tracking of the strand elongation process. SMRT devices produce variable length reads, with the read length largely dependent on the binding affinity of the polymerase enzyme utilized by the technology. The method currently achieves the longest average read length, with however a broad spectrum of read lengths and below-average level of parallelism.

The dideoxynucleotide chain-termination sequencing method has undergone a considerable time span of continued refinement. Therefore, although next-generation instruments are improving steadily with more and more sophisticated chemistry and signal processing, Sanger sequencing technology still defines the gold standard in terms of single read accuracy. Apart from stochastic measurement fluctuations, all present technologies feature an error profile with an increase in error frequency towards the end of the read. Typical sources of systematic decline in sequence quality include deterioration of the sequencing reagents over time or loss of phasing for methods relying on clonally amplified DNA. Orthogonally, overall read quality is influenced by factors like cross-talk between adjacent spots, mixed clusters or optical distortion. [63]

Due to phased probing of the template DNA, with the Illumina and SOLiD instruments measurement error mostly induces base substitution errors. Conversely, 454 and Ion Torrent instruments are primarily susceptible to over- or underestimation of homopolymer lengths, which results in an elevated frequency of base insertion and deletion errors. SMRT suffers from an elevated frequency of nucleotide deletion errors.

All platforms including Sanger sequencing are furthermore subject to sequence specific biases, where both the probability of sampling specific sequences as well as single base accuracy can be strongly influenced by the local sequence context. Besides the implications for sequencing data analysis and interpretation, uneven error rates and sampling probabilities result in an increase of required average sequence coverage and thus elevated cost-per-base.

Sampling bias may be introduced not only during the actual steps of the sequencing procedure, but also the weakly standardized sequencing library preparation, with e. g. PCR amplification an important and frequently cited source of bias [64]. Sequencing bias is classified either according to the introducing reaction or step of the library preparation, such as ligation or fragmentation bias, or according to the sequence properties it is attributed to, such as base composition or end biases.

Since quality and strength of these biases are dependent on an only vaguely understood interplay of variable factors like fragmentation size and exact PCR conditions, they are not easily corrected for and must be considered during data analysis and interpretation of results.

1.5 Sequencing Data Analysis

While data analysis strategies for different applications of high-throughput sequencing naturally diverge at some point following the initial base calling of the read sequences, they frequently share common data processing tasks or generic approaches such as reference genome-based resequencing. In addition to software solutions offered by device vendors or other commercial providers, countless computational utilities are developed by the scientific community implementing specific analysis strategies or prerequisites of generic analysis approaches such as read mapping.

Following the *base calling* step in which nucleotide sequences are deduced from signal intensities delivered by the sequencing device, raw sequence data undergo quality control and further processing according to parameters dictated by the respective application and analysis strategy. Subsequent strategies diverge into different classes such as reference sequence-guided resequencing approaches, de-novo sequence analysis such as genome assembly as well as reference-free multi-sample comparison for diverse purposes such as e. g. assessment of variation or microRNA content. Primarily motivated by the relatively short read lengths offered by current sequencing technology, reference sequence-guided resequencing remains the most widely used and practical method (section 1.5.3).

1.5.1 Base Calling and Read Quality Assessment

Base calling is the initial step of deducing an actual nucleotide sequence from the raw signal measurements delivered by sequencing machines, e. g. the brightness levels measured by the CCD detector for fluorescence-based techniques. Parallel sequencers operate by sampling the entire measurement area supporting large numbers of DNA molecules (e. g. flow cell) at certain intervals (section 1.2). Instrument measurements initially are stored as a series of images, where each image represents the signal intensities across the measurement area at a single sampling interval. The stack of images generated is subsequently processed in software.

In short, base calling software is required to locate spots within the measurement area that represent valid DNA templates, and further analyze the run of signal intensities at each spot to provide a nucleotide sequence that likely explains the observed intensities. In addition to this maximum likelihood estimate of a sequencing read's nucleotide sequence, a measure of reliability for the issued base calls is desirable. Due to their conceptual simplicity, PHRED-like per-base quality scores [65, 66] have become the de facto standard across all platforms. Additional measures that capture the peculiarities of the respective sequencing process, like e. g. individual base likelihood in the case of reversible chain terminator sequencing or homopolymer accuracy in the case of pyrosequencing, can be calculated by specific base callers but are widely disregarded by downstream analysis software.

Base calling software is usually highly device specific and thus provided by the instrument vendors; alternative software solutions exist in some cases [67–71], but have not become widely adopted.

Quality of high-throughput sequencing reads can be highly variable and is dependent on many different factors (section 1.4). Therefore, prior to further analysis it is often desirable to assess the quality of the generated sequencing reads, and to pre-filter the raw read data accordingly. Furthermore, depending on the sequencing protocol and application, the relevant sequences are often fused to utility oligomers like bar codes, sequencing adapters or linker sequences that are still present in the sequencer's output and interfere with direct application of the appropriate analysis algorithm. Countless tools and scripts have been developed for these purposes and are in some cases made freely available.

FastQC [72] for example is a simple Java application providing visualization for average read quality and various other properties of the data set. The FASTX *Toolkit* [73] provides tools for creating sequence quality charts, adapter removal and demultiplexing operations. *TagDust* [74] is a tool that identifies library artifacts such as sequencing primer dimers. Similar capabilities are offered by the tool *TagCleaner* [75], with additional read clipping, trimming and splitting options. A collection of Perl scripts for read filtering, quality-based trimming and creating read quality charts is provided by the NGS QC *Toolkit* [76].

1.5.2 Assembly

While the classical domain of large-scale DNA sequencing is the assembly of entire genomes, due to their short read length most high-throughput sequencing technologies do not produce data optimally suited to this approach. Computational tools based on alternative assembly algorithms and data structures such as De Bruijn graphs have been developed that can better cope with large amounts of short reads, for example the *Velvet* [77], *Oases* [78], SOAP*denovo* [79], ALLPATHS [80, 81] and ALLPATHS-LG [82, 83] assemblers. Despite these improvements, stock short read assemblies still do not reach the quality of typical laboriously generated reference genomes [84]. Generating high quality reference assemblies continues to require enormous effort often involving construction of BAC libraries and large amounts of man-hours both on the wet lab and data analysis sides for closing gaps in the assembly, ordering the assembled scaffolds or identification of mis-assembled regions.

These remaining issues prevent large-scale multiple whole-genome comparison approaches, and as a consequence, solutions to data representation and methodology for this type of analysis are not yet well established. In the medium term, sequencing technology can be expected to reach a sufficient mixture of sequencing cost, throughput, read length and accuracy to render whole-genome assembly a rather routine task. For genome-wide variation analysis, this should bring about a shift towards such large-scale assembly approaches.

1.5.3 Short Read Alignment

While direct genome assembly remains impractical, most applications of next-generation sequencing are analyzed through resequencing approaches that rely on the availability of a suitable reference genome sequence. Relating all data to a single assembled genome circumvents the issue of assembly, facilitates comparisons by establishing a common coordinate system and allows to examine analysis results in the context of a fully refined genome annotation.

Mapping millions of short reads to reference genomes up to multiple gigabases in size constitutes one of the computationally most demanding tasks of the resequencing strategy. In the presence of sequencing errors, erroneous reference sequences and possibly genomic variation such as SNPs, indels and rearrangements, perfect match queries between read and reference sequence cannot in general deliver satisfactory results.

The read alignment problem presents a string matching task that has quickly been met by a variety of proposed algorithms and software solutions, confronting the scientific community with a multitude of freely available alignment tool options like BWA [85], *Bowtie* [86], *Bowtie2* [87], SOAP [88], SOAP2 [89], SHRiMP [90], SHRiMP2 [91], MAQ [92], *GenomeMapper* [93], PAL*Mapper* [94] or *CloudBurst* [95].

To achieve satisfactory performance, alignment tools rely upon precomputed, static index data structures usually generated for the reference genome, ranging from simple k-mer hashes or spaced-seed strategies to suffix trees, suffix arrays or equivalent compressed full text indexes such as the Burrows-Wheeler transform (BWT) based FM-index [96]. The genome indexing approach has implications regarding both performance and hardware requirements of an alignment algorithm. Disk space and index memory required for k-mer indexes, suffix trees and plain suffix arrays amount to a multiple of the size of the indexed nucleotide sequence. In contrast, compressed BWT-based indexes have the advantage of an inherent representation of the genome sequence, with their size approaching that of the compressed sequence.

Different index data structures aside, read mapping algorithms are set apart by their choice of trade-offs between mapping speed, accuracy, sensitivity and supported alignment parameters. At the same time, mapping accuracy is partially influenced by mapping sensitivity, since a missed mapping location for a read may lead to spurious mappings to a different location of similar sequence. Read mapping algorithms commence by identification of potential origins of *seeds*, short substrings of the sequence read matching the reference genome sequence at high sequence identity. The tradeoff between mapping speed on the one side and accuracy and sensitivity on the other is greatly influenced by seed selection, seed length as well as the required level of sequence identity. Tools may decide to ignore seeds that occur with very high frequency in the reference genome, further accelerating the mapping process at the cost of sensitivity.

Following seed placement, follow-up alignment algorithms establish the actual base pairings between sequencing read and reference sequence, in the process identifying the best possible match for the entire read according to the scoring measure defined by the mapping tool. Furthermore, many mapping utilities in addition allow the explicit or implicit retrieval of the next-best mapping location, permitting to infer the reliability of the assigned location ("mapping quality") under the assumption of completeness and correctness of the provided reference genome sequence.

The set of features supported by mapping and alignment algorithms decides over their suitability for different kinds of sequencing data or application. However, omission of certain features may help with optimization of the algorithm. Gapped alignments for example come with a significant performance hit, but are indispensable for application in variation detection or for aligning sequencing reads obtained by pyrosequencing, Ion Torrent or SMRT technology. Further gap-related features such as affine gap costs or split-read approaches permit correct placement of the read in presence of longer insertions and deletions or intron-exon boundaries.

Sequencing reads with special properties such as color space or bisulfite converted data furthermore require fully customized mapping and alignment strategies.

1.5.4 Variation and Genotype Calling

Reversing the assembly and multi-genome comparison approach (section 1.5.2), resequencing has become the current method of choice for assessing genomic variation. In the resequencing setting, all read data for certain subspecies, strains or mutants are mapped to a single reference sequence of a closely related organism. The complexity of the analysis task is therefore shifted from establishing and examining complex relations between multiple assembled genomes towards obtaining a comprehensive and reliable readout of the structural information contained in the short read alignments.

Basic consensus calling is limited to the detection of conserved regions and small scale variation like single nucleotide polymorphisms and small insertions and deletions contained therein. Such regions of evident sequence conservation are directly suggested by the short read mappings. The task for analysis algorithms is therefore to assess each of the supported positions to identify the most likely combination of alleles to give rise to the observed data, along with a measure of confidence for the provided solution.

Identification of this most likely combination of alleles at each reference genome position is therefore the main problem to be solved by analysis algorithms. A major issue to be overcome by these algorithms is to achieve an integration of models for sequencing errors, read mapping and base alignment errors. While the probability of sequencing errors may be systematically influenced by the local sequence context, individual error events in different reads may be considered as largely independent. Sources of mapping and base alignment error are however mostly systematic in nature. Reads and aligned bases therefore do not represent independent evidence for a certain allele configuration, preventing these types of error from being as readily captured by statistical modeling.

The majority of consensus and genotype calling algorithms commence by examining the *pileup* of base calls and base qualities provided by the mapped reads overlapping a position. Given sufficient depth of coverage, eventual sequencing errors can be thereby identified at high confidence.

However, sequencing error may not only cause invalid readout of single positions, but also spurious cross-mappings to loci with similar sequence to that of the true origin of a read. The likelihood of such spurious mappings may be assessed using measures like the mapping quality provided by some alignment tools (section 1.5.3). Furthermore, heuristic parts of the mapping algorithms used to achieve improved mapping performance may also cause systematic cross-mappings that can be indistinguishable from true variants or give rise to invalid multi-allelic calls. Such pseudo multi-allelic regions may further arise due to copy number variants, under-representation of repetitive sequence in the reference assembly or sample contamination.

Another source of systematic miscalls presents base alignment error. Base alignment error is introduced by reads, albeit mapped to the locus on the reference genome from which they originate, whose alignment provided by the read mapping tool does not suggest the correct set of base pairings. This type of error mainly occurs at transition points to insertions, deletions, copy number variants or other types of rearrangement or at exon-intron boundaries in mRNA sequencing. Reads overlapping these transition points only with a small number of positions do not contain enough information to support the correct type of variant, causing spurious alignment of terminal nucleotides.

For these reasons, current consensus and genotype call algorithms employ combinations of statistical models as well as alignment correction algorithms and heuristics for removing or penalizing read alignment issues.

The Genome Analysis Toolkit (GATK) [97] combines Bayesian heterozygous genotype calling based on quality score recalibration with indel realignment and various heuristics for penalizing mapping artifacts [98].

SHORE [18] implements a flexibly calibratable scoring matrix approach [1, 99] that calculates SNP, indel and reference call scores based on a user-definable weighting of various properties of the read alignments overlapping a position. The method allows to impose a penalty on, or to ignore a number of terminal nucleotides towards the ends of each alignment, ruling out many cases of base alignment error at the cost of an effective loss of sequencing depth.

The SAM*tools* package [100] implements a profile HMM for estimation of a Base Alignment Quality (BAQ) for the assessment of potential mapping artifacts [101].

1.5.5 Quantification by Deep Sequencing

In principle, deep sequencing constitutes statistical sampling of the population of molecules that make up the sequencing library. Therefore, obtained sequencing data may be transformed into quantities that represent a frequency distribution that may be used to infer relative quantities with reference to this population.

Depending on the application, these measurements may be processed further for mere classification purposes (e. g. enriched versus not enriched in chromatin immunoprecipitation (ChIP) assays) or transformed into quantities appropriate for multi-sample comparison (e. g. normalized expression level in RNA sequencing).

For routinely used, but weakly defined concepts such as gene expression, definition of an appropriate reference for obtaining biologically meaningful quantities can be intricate. Depending on the application the appropriate unit of measurement might be absolute, i. e. the number of copies of a molecule in the cell or in the sample, a cellular or compartmental concentration, or relative to the total population of molecules of the same family. To a certain degree, relative measurements can be directly obtained from sequencing data, whereas calculation of absolute values is in general not possible. For identification of differentially expressed loci in a multi-sample comparison, it may however be sufficient to represent all measurements in relation to a common reference value. Given many loci contribute to the sequencing library with a majority following the same distribution in all conditions examined, the problem of calculation of a common reference value can be reduced to identification of outliers in a multi-sample comparison.

Next to basic compensation for sequencing library size, further specialized normalization procedures are therefore often applied to the raw quantities of sequenced reads. Some of these procedures attempt inference of quantities that are better suited to the comparison of the biological conditions in question, such as estimation of absolute quantities. Further aims of normalization include stabilization of measurement variance or removal of sequence specific biases. As an example, for whole transcriptome sequencing a normalizing procedure termed trimmed mean of M-values (TMM) [102] has been shown to compare favorably with plain relative quantification. For data with different properties such as microRNA profiles, consensus regarding data normalization is yet to be reached [103].

1.5.6 ChIP-Seq Analysis

While chromatin IP (section 1.3.2) enriches for sequence fragments that contain a binding site for the protein of interest, no perfect purification is achieved. Obtained sequencing data therefore tend to contain significant amounts of background noise in the form of reads that represent DNA fragments that were sequenced purely by chance rather than due to their affinity to the DNA-binding protein. Primary data analysis involves the identification of those genomic regions that potentially contain one or more binding sites, recognizable from the mapped read data by characteristic peaks in sequencing depth at the respective sites and their close vicinity.

Single transcription factors are often thought to contribute to the regulation of hundreds or thousands of different genes, and to take on different roles in various regulatory networks. This creates the need for *peak calling* algorithms that are able to process the ChIP-Seq data to identify all putative binding sites in a genome-wide scan. Multiple freely available computational tools including MACS [104], QuEST [105], SISSRs [106], *PeakSeq* [107], *cisgenome* [108] or *FindPeaks* [109] all implement different algorithmic approaches to the peak calling problem.

MACS employs estimation of local depth of coverage background using a sliding window. By this background estimate, p-values for enrichment are calculated based on the Poisson distribution. The program is able to operate with or without control libraries and calculates false discovery rates through a sample-control swap.

SISSRs calculates "net tag count" as the difference of forward and reverse strand read mappings in short jumping windows. Base line transitions of net tag count are retained as sites of putative enrichment, which are further compared to Poisson-motivated read support thresholds.

The *PeakSeq* program identifies candidate enriched regions through segment-wise calculation of a depth of coverage threshold obtained by simulation of random read distribution, taking into account a pre-computed “mappability map” to capture repetitive parts of the genome. Thus obtained candidates are assessed further using a binomial test comparing sample and scaled control data.

Particular to the QuEST algorithm, peak detection relies on score profiles generated by kernel density estimation over the reads' $5'$ mapping coordinates rather than depth of coverage. Peak calling is achieved by identification of local score maximums and a combination of score and fold change thresholds.

While a multitude of different analysis algorithms thus exist, due to the difficulty of assessing the quality of the provided solution there is generally no consensus which of the available algorithms performs best in a given scenario.

1.6 Storage and Representation of Sequencing Data

High and constantly improving throughput of sequencing instruments implies routine production of uncommonly large data sets. The amount of data produced poses challenges not only regarding data transfer, processing and analysis, but also in terms of data storage. In the desire to guarantee best possible traceability of scientific results and all potential sources of error, the benefit of long-term storage of all raw data must be balanced with storage cost, further considering the likelihood to again require certain types of data and the cost of eventually regenerating experimental data. Under these considerations it has become accepted practice to require long-term storage of at least nucleotide sequences and PHRED qualities of sequencing reads, whereas the preservation of raw signal measurements or further quality measures (section 1.5.1) is deemed optional.

Generally, large storage size of the data sets emphasizes the importance of the tradeoff between space and access efficiency. While off-line long-term storage solutions are able to mainly address space efficiency, with the only restriction being that the computational effort required to convert to and from the compressed representation of the data should remain in tolerable bounds, on-line representation of the data during the data analysis phase has to provide additional guarantees regarding data access.

1.6.1 Storage Formats for Next-Generation Sequencing Data

High-throughput sequencing data sets store elements like sequencing reads or read alignments, which define a tuple of *attributes*, such as a read identifier, nucleotide sequence and per-base qualities. Additional meta-data items like sequencing instrument, run or processing information may be required to be kept alongside the data set. On-line sequencing data representation has to at least guarantee efficient sequential traversal of a data set's elements, i. e. enable efficient retrieval of all of the respective next element's attributes. Furthermore, it is typically required to allow storage of the data set elements in a specific order not dictated by space efficiency considerations. Finally, accelerated queries for data set elements with specific properties, such as range queries for retrieval of elements overlapping with a given region of the genome, require a certain degree of random access support from the storage solution.

While generic solutions such as relational database management systems (RDBMS) address many of these requirements, they come with an excess of further features while impeding data set distribution. Therefore, most sequencing data analysis solutions rely on custom file-based storage solutions. File formats in the bioinformatics field have traditionally been ASCII text files following simple formatting rules. Examples include the de facto standard formats for representation of biological sequences, FASTA and FASTQ [110]. For description of large amounts of features that are relative to a reference sequence, simple tab-delimited tables have emerged as the predominant storage formats. For example, SAM [100, 111] is a standard file format used for storing sequencing reads and read alignments. GFF [112] is a generic format for description of genomic features that is primarily used, but not restricted to, representation of genome annotation data, and its subset specification GVF [113] is targeted at description of genomic variation like SNP or structural variants. VCF [114] provides a specification for storage of multi-sample variation and genotype information. SAM, GFF and VCF are structured similarly, with each data set element being represented on a single line. Common or mandatory element attributes like reference genome coordinates are stored as mandatory columns, whereas optional attributes are added in arbitrary order in the form of key-value pairs (tags). All three formats are generic formats that allow users to specify arbitrary additional attributes, and allow inclusion of meta-data in the file in the form of lines starting with a certain character sequence serving as meta-data indicator. The SAM format comes with an equivalent binary companion format BAM with the aim of reduced file parsing overhead.

While processed data like variant calls are usually moderate in size, space efficiency is a concern for raw read and read alignment data, suggesting the application of data compression algorithms. The generic compression algorithm DEFLATE [115] is one of the most widely used compression methods due to a tradeoff between compression efficiency and compression and decompression speed that is acceptable in many settings. Generic compression algorithms are also applied to sequencing data, e. g. the BAM format is routinely encoded as BGZF [111], a simple subset specification of the GZIP [116] format for DEFLATE-compressed data.

However, compression efficiency can be improved through use of sequencing data specific encoding methods or preprocessing. To some extent, high-throughput sequencing data compression is related to the issue of DNA sequence compression. Complete genome sequences are known to be hardly compressible beyond the typical 2-bit encoding of the four bases using common generic compression algorithms [117]. Most efforts preceding the emergence of high-throughput sequencing have thus been directed at generating efficient representations of such large biological sequences. However, algorithms that have been made available for this purpose are not readily applicable to short read sequencing data.

Common approaches used by most DNA compression algorithms in the literature rely on suffix arrays or similar index data structures for detecting and collapsing perfectly or imperfectly repetitive regions in a small set of large sequences. As such, these implementations are not suited to the task of compressing the huge sets of very short sequences generated by a high-throughput sequencing run. Analogous approaches are however available for short read sequences. Imposing a specific ordering on the elements of a data set potentially leads to an increase in entropy, and thereby reduced compression efficiency. A utility SCALCE implements a read reordering method using a similarity-based clustering approach that serves as a compression booster in combination with generic compression algorithms and formats [118]. However, as specific element order is often a requirement of analysis algorithms, SCALCE and similar reordering approaches are most suitable for off-line long-term storage and data distribution. A different approach is taken by the tool *Quip*, which implements an assembly-based short read compression scheme where reads are represented by their position on specifically assembled contigs [119]. Assembled contigs must be stored along with the data set elements, but allow efficient compression of data sequenced to multiple depth.

While these approaches focus on compression of short nucleotide sequences, those represent just one of the attributes to be stored in next-generation sequencing data sets, along with read identifiers, per-base PHRED qualities and possibly read alignments. Various methods therefore attempt to improve compression ratio for these heterogeneous collections of data. Notably, the CRAM format [120] aims at being a drop-in replacement for the SAM/BAM format, to large extent supporting the same logical structure and set of features. The main features of CRAM are reduced representations of aligned read sequences and per-base qualities. For aligned sequences, the format implements *reference compression*. In reference compressed alignments only the lengths of read subsequences that perfectly match the reference genome are stored, or an equivalent representation. While in principle a lossy operation, all discarded information may subsequently be recovered as long as the appropriate reference sequence is available. For PHRED quality data, CRAM offers a choice of various lossy binning schemes aimed at reduction of the entropy introduced with the read qualities. Following such preprocessing operations, CRAM encodes data into a custom binary format utilizing well-known encoding schemes such as Golomb [121] and Huffman coding [122].

1.6.2 Accelerated Queries on Sequencing Data

Support for accelerated queries on a data set is required to enable rapid data visualization [123, 124] and data set exploration. Indexed range queries are e. g. supported by the *BigWig* and *BigBed* formats [123], the BAM format or the utility *Tabix* [125] using R-tree [126] based indexing [124].

While stock generic compression algorithms permit streaming encoding and decoding and are thus compatible with the goal of sequential, ordered traversal of data sets, they do not comply with random access requirements. This disadvantage is commonly mitigated by enabling seeking to defined positions in the stream. These decoding offsets are realized by periodically either saving the complete state of the encoder, or by causing an encoder reset. In both cases, the cost is an effective decrease in compression efficiency, either because of the overhead of saving the series of encoder states along with the compressed data, or because the encoder needs to be retrained following each reset. Additionally, an index data structure must be integrated to allow conversion between compressed and uncompressed stream offsets. As a result, there is a tradeoff between compression ratio and random access granularity, and thus efficiency.

1.7 Contributions of this Work

The software SHORE is a processing and analysis pipeline for high-throughput DNA sequencing data [1, 18]. The pipeline is targeted at a reference sequence-based resequencing data analysis approach, combining raw read manipulation and filtering, read mapping and several analysis modules specific to different sequencing applications.

With this work, large parts of our pipeline were reworked for coping with increasingly large amounts of data, optimization of workflows and extended analysis requirements. In the process, SHORE was embedded into a generic programming framework to provide universal support of introduced features across all processing modules (section 3.1). Increasing storage sizes of data sets produced by high-throughput sequencing devices have been addressed by implementation of a variety of data compression options. At the same time, increased processing times were countered by extended parallel processing facilities, and specifically distributed memory parallelization of the read mapping process. Furthermore, with greater data set size, efficiency of retrieval of specific data set elements or subsets has gained importance for data set exploration and visualization. To meet these requirements, appropriate data indexing and query algorithms were incorporated. Exploiting the data set indexing facilities, within several analysis modules we address basic visualization of sequencing read alignment and depth of coverage distribution to mitigate the need for time-consuming data set copying and conversion for use with external visualization solutions.

Our application programming framework *libshore* was implemented as a common foundation to all SHORE processing modules. In addition to elementary functionality such as automatic input and output decoding and encoding, algorithms for indexing sequencing-related data and alignment of biological sequences are provided. Furthermore, we developed a generic processing framework with the aim of facilitating breakdown and implementation of rather complex sequencing data analysis algorithms as collections of manageable and self-contained processing modules. We further extended on this framework for straightforward parallelization of basic sequencing data processing.

The fundamental data analysis workflow introduced with the initial version of SHORE consists of subsequent application of read filtering, read mapping and application-specific analysis modules. While this basic approach has been maintained, it has been extended in many areas for simplification of a variety of routine tasks. An overview of the modified workflow is therefore given in section 2.1.

Significant modifications to core pipeline modules concern raw sequencing read import and filtering as well as read mapping. Data import was modified to enable recovery of raw data or iterative data set re-filtering (section 2.3), and complemented with extended functionality to address e. g. the more and more widespread use of a variety of library multiplexing and further adapter ligation protocols. For this purpose, flexible sequence-specific read partitioning and clipping facilities have been added (sections 2.4, 2.5). SHORE's read mapping module initially constituted a basic parallelization wrapper providing a unified interface to different short read alignment tools. To take advantage of an increasing diversity of read mapping algorithms, features have been added to enable integration of the results obtained using different alignment algorithms or parameters (section 2.6).

Application-specific data analysis modules initially implemented in SHORE mainly addressed the assessment of genomic variation. This functionality has largely been maintained, with only slight modifications to take advantage of the *libshore* infrastructure. A focus of this work however was on quantitative application of high-throughput sequencing, and primarily ChIP-Seq data analysis. Within the SHORE analysis environment, we have implemented an enrichment detection module targeted at the analysis of transcription factor immunoprecipitation data (section 2.7). In comparison to other approaches, our method emphasizes robustness towards read mapping artifacts and simplifies handling of replicate experiments. Our ChIP-Seq enrichment detection module is complemented by configurable auxiliary utilities allowing composition of custom workflows applicable to various expression profiling or enrichment detection applications.

In conclusion, with this work the SHORE software has been developed from an analysis pipeline consisting of a set of largely independent submodules targeted primarily at genomic variation assessment into a tightly integrated, general-purpose sequencing data analysis suite supporting a set of coordinated, interdependent features. The SHORE sequencing data analysis suite is open source software freely available[1] under the GNU General Public License (GPL) version 3.

[1] http://shore.sf.net

2 A High-Throughput DNA Sequencing Data Analysis Suite

The constant pace of development of high-throughput sequencing technology and protocols is a force driving continued adaptation and improvement of related software solutions. The present chapter is concerned with the data storage solutions and retrieval algorithms, high-throughput sequencing data analysis algorithms and software utilities that were developed in the course of this work for use with the SHORE DNA sequencing data processing and analysis pipeline.

2.1 Overview

Amounts of data produced by high-throughput sequencing devices have from the start posed a challenge with respect to data analysis. Since their introduction, countless computational utilities have been developed to satisfy different data processing and analysis needs in all areas of application. Large-scale high-throughput sequencing projects however emphasize the importance of standardized work flows and data storage. With data analysis extending over multiple phases as well as a significant time span and overall man-hours distributed among multiple individuals, coordinated efforts to ensure coherence and reproducibility of processing and analysis results become imperative.

A basic level of standardization is readily achieved by reliance on whole-sale software solutions, with all required data processing and analysis algorithms based on a common framework. However, apart from commercial offers few freely available high-throughput sequencing applications provide integration — to varying degrees — of more than just isolated processing and analysis steps, e. g. SOAP, SAM*tools* [100] or GATK [97].

With the SHORE software, we provide an integrated high-throughput sequencing read processing and analysis framework based on consistent data storage concepts. SHORE offers an end-to-end pipeline including all relevant steps starting from initial raw read data filtering and partitioning and proceeding up to primary analysis results. An overview of SHORE's general data analysis work flow is provided in figure 2.1. While in principle consistent with what has been described [1] (cf. section 1.7), we implement a variety of extensions including optional recovery of raw data, on-the-fly merging, subset selection and filtering as well as data visualization, as discussed in the following.

In a reference sequence-guided resequencing approach, primary data analysis proceeds in three main stages. First, raw sequencing read data obtained from the base calling software become subject to quality control and demultiplexing algorithms (figure 2.1(a)). Thus processed reads are subsequently mapped to the appropriate reference genome sequence (b). Application-specific algorithms then interpret the read alignment data as the final step of the primary analysis (d), such as genotype calling, assessment of structural variation, enrichment detection or differential expression analysis. Approaches to follow-up analysis of the generated primary results are manifold and highly divergent, and therefore not currently integrated with the pipeline.

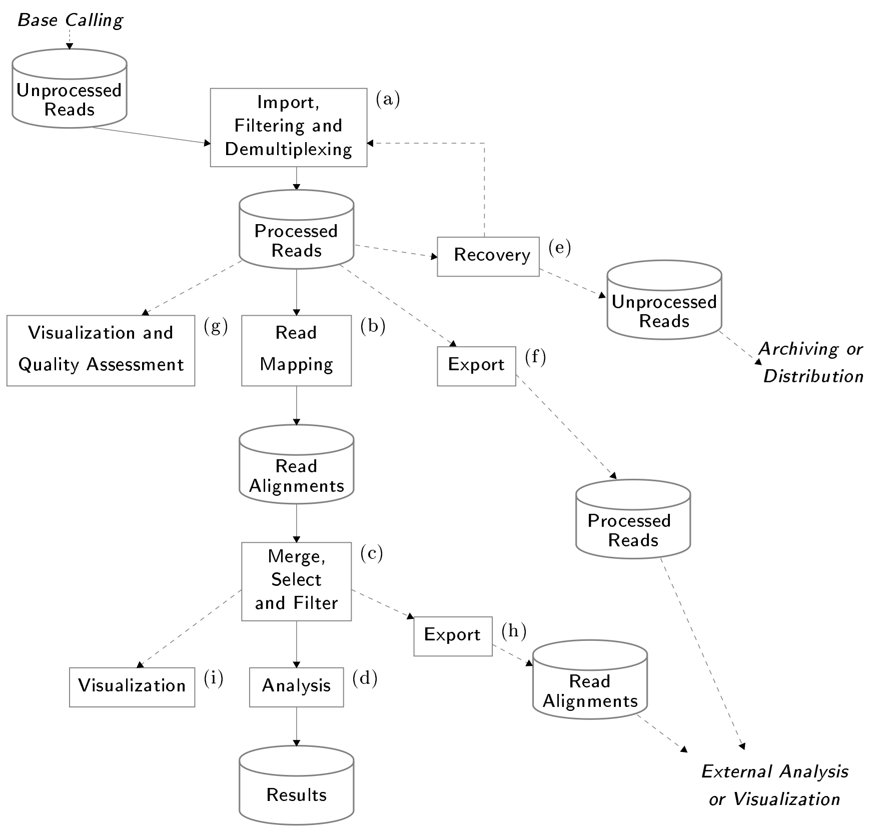


Figure 2.1: The SHORE Workflow

Our analysis pipeline is realized as a command line application organized as a collection of submodules. A module SHORE *import* provides initial processing of raw read data and logically organizes its output in a predefined directory hierarchy, which forms the basis for the operation of further modules (section 2.3). The module provides a comprehensive set of commonly required filtering operations such as quality based filtering and read trimming, removal of sequencing adapters or demultiplexing for bar-coded samples (section 2.4). To facilitate iterative adjustment of filtering parameters or distribution of raw read data to third parties, the module further provides functionality to fully restore its original input data (e).

Analysis of SHORE processed data by means of third-party analysis utilities is possible through format conversion. A module SHORE *convert* provides read and alignment data export to a variety of commonly supported data exchange formats such as FASTA, FASTQ, SAM, BED or GFF (f, h). To assess overall run quality, nucleotide composition and base quality distributions may optionally be visualized with a module SHORE *tagstats* (g, section 2.8).

Data analysis methods implemented in SHORE primarily follow a reference sequence-oriented resequencing approach (section 1.5). The provided module SHORE *mapflowcell* is responsible for the required mapping of the processed read data to a reference genome sequence. The mapping module is realized as a high-level command line interface and parallelization front-end to a variety of widely useful read mapping tools such as *GenomeMapper*, BWA or *Bowtie2* (section 2.6).

Read mapping data stored as a result of the SHORE *mapflowcell* module constitute the input for data analysis modules such as variant calling or enrichment detection (d). Available analysis modules cover SNP and indel detection (SHORE *qVar*, *consensus*), ChIP-Seq analysis (SHORE *peak*, section 2.7), reference-based small RNA analysis as well as versatile modules SHORE *coverage* and SHORE *count* applicable to a variety of expression analysis and enrichment sequencing approaches. A further module SHORE *mapdisplay* as well as SHORE *coverage* and *count* support data set exploration through visualization of read alignments or depth of coverage, respectively (i, section 2.8).

Alignment input data to analysis and visualization modules can on-the-fly be selected by region of interest, merged with data of e. g. further sequencing runs as well as passed through a variety of filtering operations such as duplicate sequence removal (c, sections 2.2.3, 2.7.2).

The SHORE data analysis suite is designed for seamless integration into established Unix command line workflows. Where appropriate, modules support streaming input and output to and from other commands. SHORE's data are stored in simple line based text file formats utilizing standards-compliant compression formats, facilitating data manipulation by stock text processing utilities. To support routine application in a specific scenario, all of SHORE's defaults may be pre-configured through user-specific configuration files.

2.2 Efficient Storage of High-Throughput Sequencing Data Using Text-Based File Formats

For large data sets as generated by high-throughput sequencing machines, space efficient on-disk representation is imperative. At the same time, during active analysis phases data must remain easily retrievable, requiring an appropriate compromise between compression ratio and access efficiency (section 1.6). This section presents data storage formats, indexing and retrieval mechanisms implemented in the SHORE pipeline.

Typical binary file formats for sequencing data come with both advantages and disadvantages compared to traditional text-based alternatives. SHORE makes use of compressed text file formats for read alignment data that achieve similar or better compression ratios compared to compressed binary formats such as CRAM. SHORE's file compression formats all support streaming input and output as well as near-random read access to decompressed file offsets. Users may flexibly choose the desired tradeoff between compression ratio, decoding and encoding speed and random access guarantees. Specialized utilities offer binary search like queries and range queries on compressed or uncompressed data sets for many types of text-based file format popular with the bioinformatics community.

2.2.1 Overview

Driven by the amounts of short read and alignment data generated during HTS data analysis, specialized binary storage formats have been introduced, notably the BAM format [100] and the CRAM format [120] both designed for storage of aligned as well as unaligned short read data. BAM employs binary encoding of values with the aim of reduced file parsing overhead, i. e. to improve efficiency of conversion between the disk and the in-memory representations of data. On the other hand, CRAM defines a specialized format with the aim of enabling increased compression ratios by supporting custom encoding for certain attributes of the data set elements.

In spite of the reasons in favor of specialized binary data encoding, there are also drawbacks compared to traditional ASCII text-based storage formats. Parsing custom binary formats requires careful consideration of technical details like machine endianness and representation of floating point numbers. Additionally, in depth knowledge of the relevant data encoding and compression algorithms is inevitable. Unless a parsing library is provided for the programming language of choice, readout of binary formats is therefore often too laborious or beyond basic programming skills, effectively limiting users of the format to choose from a narrow set of supported programming languages.

In contrast, basic parsers are straightforward to implement for simple tab-delimited table formats. Requirements such as string tokenization and conversion between character strings and numeric types are well supported by high level scripting languages popular in the bioinformatics field such as *Perl* or *Python*, and considered a basic skill in next to any programming language. The line-based formats are furthermore accessible to shell scripting and readily filtered through widely used *Unix* text processing utilities such as *grep*, *awk*, *sed* or *sort*. While conversion between file and in-memory representation is less efficient compared to optimized binary formats, the introduced overhead is marginal in relation to the overall CPU consumption of data analysis algorithms.

Furthermore, delimited text tables can serve as a basic framework for file format definitions such as SAM, GFF or VCF. This common foundation enables implementation of generic algorithms applicable to a large variety of different file formats, as demonstrated for example by the tool *Tabix* [125] for range indexing of genomic feature file formats. Adapting similar algorithms to different specialized binary formats is laborious, requiring reimplementation or implementation of sophisticated adapter mechanisms.

Format issues are easily identifiable with text-based file formats without requirement of expert knowledge or implementation of specialized format verification routines. Software dealing with data produced by a rapidly evolving technology as is high-throughput sequencing instrumentation is likely required to undergo constant adaptation. Besides, at high rates of data production storage systems may operate close to the limits of their capacity. In such potentially unstable settings, simplicity of file formats constitutes a significant advantage.

On grounds of these considerations, the SHORE analysis suite relies on tab-delimited table formats for storing reads, read mapping data and analysis results. Users may further choose to store files either directly as plain text, or between one of two widely used compression formats, GZIP [116] and XZ [127]. While such generic compression file formats themselves represent binary formats, decoding commands and routines are widely available across platforms and programming languages and are employed merely as a filtering layer, thus retaining most of the advantages of a text-based format. With compression, SHORE implements block-wise encoding for both the GZIP and the XZ format to enable near-random read access at any decompressed file offset. Our implementation enables free adjustment of random access and space efficiency trade-offs (section 2.2.2).

SHORE's text-based read alignment file format was extended to support reference compression as introduced by the binary CRAM format (section 1.6), as well as further simple preprocessing filters serving to boost compressibility in combination with generic compression algorithms. Depending on the choice of preprocessing filters, compression algorithm and random access resolution, our compressed text file formats enable similar or better read alignment compression ratios compared to CRAM (section 2.2.5).

Another common requirement in sequencing data analysis is the retrieval of data set elements with a certain attribute value, e. g. extraction of the set of single nucleotide polymorphisms included in a certain range of the reference genome or retrieving a read with a specific identifier for a short read data set. However, standard linear search is slow when processing large data sets. Given the data set is in sorted order with respect to the key attribute, accelerated queries are possible. SHORE *sort* offers text file sorting functionality similar to the standard *sort* command line utility, optimized for use with typical high-throughput sequencing file formats. Additionally, the tool is capable of performing binary accelerated queries on arbitrary sorted tab-delimited text files.

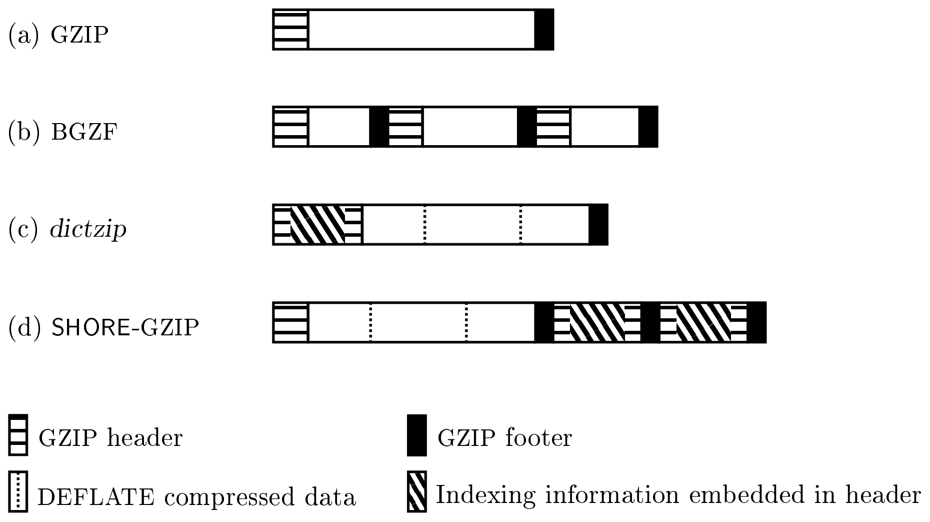


Figure 2.2: Basic Structure of the GZIP Format and its Sub-Formats

Binary search is however not applicable to elements like read mappings, gene models or variant information representing two-dimensional objects with a start and an end coordinate. Fast access to such data set elements overlapping a specific region of interest is crucial for data visualization and manual data set exploration. A text-file indexing approach capable of answering various types of range queries is implemented as a utility SHORE *2dex*, providing functionality similar to *Tabix* [125] or the BAM [100] and *BigBed* [123] indexing schemes. In comparison to the also-generic *Tabix* indexer, SHORE provides additional functionality such as flexible choice of compression format and random access resolution, indexing of files not sorted by start coordinate and fast queries on data sets with deep sequence coverage.

Both utilities SHORE *sort* and SHORE *2dex* make use of SHORE's generic random read access back end and can thereby seamlessly be applied to both compressed and uncompressed data sets.

2.2.2 Widely Compatible Indexed Block-Wise Compression

GZIP [116] is one of the most widely used generic compression formats with a high level of support across operating systems, programming languages, command line utilities and many further software applications. The employed DEFLATE compression algorithm [115] offers a reasonable compromise between encoding and decoding speed and compression ratio. Therefore, the format is an obvious choice for compression of sequencing related data. However, it lacks random read access support, forcing decompression of all data preceding a required section of the compressed file. As this interferes with the ability to perform accelerated queries on the compressed data sets, subset specifications have been developed to add this feature without breaking compatibility.

The GZIP file format embeds DEFLATE compressed data between a header and a footer sequence (figure 2.2(a)). Encoding parameters are specified within the header, whereas the footer consists solely of eight byte specifying a checksum as well as the decompressed file size modulo 2^{32}. Header, compressed data and footer constitute a GZIP *stream*, and a GZIP compliant file consists of one ore more concatenated streams. Upon decompression, data of concatenated GZIP streams are concatenated as well.

The BGZF format developed for the BAM format and the *Tabix* utility [100, 111, 125] exploits the concatenation feature to implement independently compressed blocks within the GZIP specification. Uncompressed data are split into 64 kilobyte (kB) blocks, compressed independently as GZIP streams and eventually concatenated (figure 2.2(b)). Although the resulting compressed size of a block is unpredictable and depends on the compressibility of the data, with the knowledge of the start offsets of all blocks in a compressed file it is possible to quickly locate the block containing a specific uncompressed file offset. Thereby excess data that have to be decompressed to access any specific position of the file are limited to at most the block size of 64 kB. BGZF however does not specify a means of storing the start offset of each of the compressed blocks. To realize accelerated queries on top of BGZF, block offset information must therefore be incorporated in external index files. This represents a source of potential error, as the contents of the compressed file may diverge from the stored index data. Furthermore, various software applications do not correctly implement the stream concatenation feature of the GZIP specification, resulting in BGZF compressed data being truncated after decompression of the first block.

The open source command line tool *dictzip* comes with a specification for an indexed GZIP-compliant format. The GZIP format specification allows up to 64 kB of arbitrary extra data to be embedded within the header sequence. This extra data is simply to be ignored by basic GZIP decoders. The *dictzip* format takes advantage of the extra data field for storing block offset information (figure 2.2(c)). During the encoding process, the DEFLATE algorithm offers the possibility of manual insertion of reset points from which decompression may be started without processing the entire file. By exploiting this feature to compress blocks of input data independently, the *dictzip* tool avoids reliance on the stream concatenation feature as in the case of BGZF. However, with the extra data field in the GZIP header being limited to 64 kB by specification, the number of blocks that can be stored is limited as well, effectively prohibiting storage of data with an uncompressed size of more than 1.8 gigabyte (GB). Furthermore, the entire index data to be stored in the GZIP header only becomes available after the entire file has been encoded. Streaming output is therefore incompatible with the format, which is therefore suitable for medium size, static data sets, but is not well matched to the compression of large-scale sequencing data.

Due to the shortcomings of available formats, we introduce a further subset specification. The SHORE-GZIP format compresses the entire file as a single GZIP stream to provide optimal compatibility. To enable streaming output, all indexing information is stored at the end of the file. On decompression using either SHORE or arbitrary third party tools, block offsets stored in the index are discarded. Thereby, index information is guaranteed to remain consistent with the file contents. To allow fine-tuning the tradeoff between compression ratio and random access overhead, the format enables free choice of encoding block size.

SHORE-GZIP realizes block-wise encoding similar to the *dictzip* approach by requesting periodic resets of the DEFLATE encoder (figure 2.2(d)). During encoding, the method keeps track of the compressed block offsets. After encoding all input data, the recorded index information is split into blocks of approximately 64 kB. Each block is embedded as extra data field into a GZIP header, which is appended to the compressed file followed by two byte signaling empty compressed data to DEFLATE decoders, as well as eight further byte forming the appropriate GZIP footer. The indexing information is hence stored as consecutive empty GZIP streams that will be ignored by standard decoding of the data set. The SHORE-GZIP implementation is however able to recognize, collect and decode the trailing index blocks on demand to enable random read access.

An important feature of plain GZIP is the ability for users to concatenate compressed files without breaking format compliance. With SHORE-GZIP, concatenation results in index records being interspersed among the data streams. Through correct recognition of such interspersed index elements by SHORE's index parsing algorithm, random access functionality is preserved in concatenated files.

Apart from GZIP, SHORE offers file compression using the XZ [127] format. XZ employs the Lempel–Ziv–Markov chain algorithm (LZMA) to achieve considerably improved average compression ratio compared to DEFLATE. LZMA offers decompression performance close to that of DEFLATE, whereas compression requires significantly more computational resources. XZ is modeled after the GZIP format by specification of a similar structure of concatenated streams of encoded data. The file format however natively includes terminal index records that store all positions of an eventual encoder reset and can therefore be employed within SHORE without further extensions.

2.2.3 Efficient Queries on Text Files

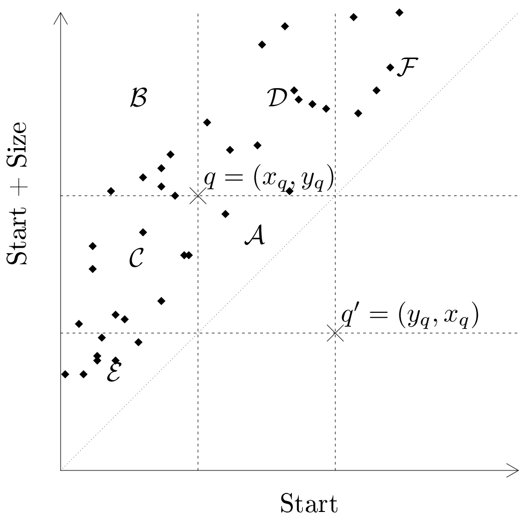
Queries for specific elements in a sorted sequence can be quickly answered by binary search in $\mathcal{O}(\log n)$ time, where n is the length of the input sequence. Stock binary search algorithms however are not directly applicable to text-formatted data sets where the total number of elements and their exact start offsets in the file are not known. We therefore implement a modified binary search for text file queries. Our algorithm bisects the data set by seeking to the central byte and subsequently seeking past the next newline marker following that position. Except for border cases that must be handled separately, the following line is compared to the user-provided search key and the algorithm is applied recursively to the appropriate subset of the file. In general, worst case time complexity of the modified binary search algorithm is $\mathcal{O}(m)$ on arbitrary text files with a total size of m byte, which however reduces to $\mathcal{O}(\log n)$ for an n-element data set if a static limit for the maximal byte size per element is assumed to exist.

Generic text file search functionality is made available through the SHORE *sort* utility. The program is applicable to arbitrary sorted tab-delimited text tables, offering the capability to retrieve specific data set elements as well as to bisect the data set using a search key.

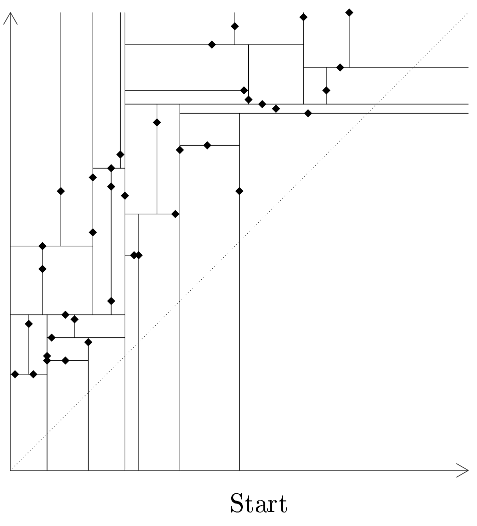
Automated data analysis can often narrow down the data relevant to the investigation to a small set of regions on the reference genome. Close examination of such individual genomic regions necessitates rapid retrieval of data associated with the appropriate range of positions. Unless consisting completely of non-overlapping entries, range queries can however not be answered by binary search.

Objects associated with a specific range are two-dimensional entities that can be interpreted as points in a plane defined by their start and end coordinate. A query range $q = (x_q, y_q)$ then is itself a point in the plane, which together with its reflection about the 45 degree line $q' = (y_q, x_q)$ partitions the search space into six different quadrants (figure 2.3(a)). Area $\mathcal{A}$ includes all objects fully included in the query range, defined by constraints $x \geq x_q$ and $y \leq y_q$. Subdivision $\mathcal{B}$ represents all elements whose range itself includes the entire query range, with the constraints $x \leq x_q$ and $y \geq y_q$. All objects intersecting the query are represented by the set union of areas $\mathcal{A}$ to $\mathcal{D}$ defined by $y > x_q$ and $x < y_q$, whereas elements completely disjoint from the query range fall into quadrants $\mathcal{E}$ and $\mathcal{F}$ ($y \leq x_q$ or $x \geq y_q$).

One of many ways to impose a spacial partitioning on multi-dimensional data represent k-d trees [128], recursively bisecting the data set with respect to alternating dimensions (figure 2.3(b)). In a perfectly balanced k-d tree, each recursion splits the current subset of the data at the median of the respective dimension's values. In that case, the tree may be represented implicitly by an array of its elements with a particular ordering [129]. k-d trees can answer region searches with a worst case time complexity of $\mathcal{O}(k \cdot n^{1-1/k} + r)$, i. e. $\mathcal{O}(\sqrt{n} + r)$ in the two-dimensional case, with n the total number of data set elements and r the number of elements to be retrieved [130].



(a) Range Query



(b) Search Space Partitioning

Figure 2.3: Range Search Using k-d Trees

SHORE implements a generic algorithm to impose 2-d tree ordering on array-like objects. Given a 2-d tree ordered array object, further algorithms facilitate retrieval of elements intersecting ($\mathcal{A}$–$\mathcal{D}$), included by ($\mathcal{A}$) or including ($\mathcal{B}$) a query range or entirely located upstream ($\mathcal{E}$) or downstream ($\mathcal{F}$) of a query position.

Based on the 2-d tree algorithms, we developed a persistent indexing scheme for text-formatted range data files, made available through the SHORE *2dex* utility. The indexing method processes input files in blocks of a fixed, user-provided byte size. For each block, we record a sequential identifier as well as the sequence range spanned by the combined elements whose storage entry starts inside the respective block. If a block is not associated with any storage entry, no information is saved. Blocks records are arranged in 2-d tree order and saved to disk as the concatenation of three arrays providing the reordered identifiers, genomic start positions and sizes. For answering range queries, array information is mapped into memory and supplied to the appropriate 2-d tree query algorithm, thereby providing the identifiers of all blocks that potentially contain elements relevant to the query. In combination with the index block size, block identifiers allow calculation of the uncompressed start offsets of the blocks in the file. A subsequent linear search phase scans the blocks indicated by the 2-d tree query to assemble the final search result.

Through the block size parameter, users are enabled to variably adjust the tradeoff between index file size and maximum length of the linear search phase. In sparse range data sets or data sets not ordered by sequence coordinate, a query range may be spanned by the combined elements associated with a block although none of the individual elements is of relevance to the respective query. In such cases irrelevant blocks must frequently be subjected to linear search unless the index block size is sufficiently small. To overcome this slight deficiency, we introduce a second user parameter *maxgap*. If subsequent data set elements in a block are farther apart on the genomic sequence than the provided *maxgap* size, then the respective block is stored multiple times associated with different ranges that do not include any coverage gap larger than *maxgap*.

```
     CCCTAAACCCTAAACCCTAAACCCTAAACCTCTGAATCCTTA
(a)  ||||||||||**|||||||||||||****|||||||||||||
     CCCTAAACCCGTAACCCTAAACCCT---TCTCTGAATCCTTA


(b)  CCCTAAACCC[TG][AT]AACCCTAAACCCT[A-][A-][A-][CT]CTCTGAATCCTTA


(c)  10[TA,GT]13[AAA,---][CT]13


     Query                                       CIGAR            MD
(d)  CCCTAAACCCGTAACCCTAAACCCTTCTCTGAATCCTTA   10=2X13=3D1X13=  10T0A13^AAA0C13
```

Figure 2.4: Example Read Alignment and its Representations in the *MapList* and SAM Formats

BAM indexing as well as the *Tabix* utility are two incarnations of a similar indexing scheme based on R-trees which was first introduced for the UCSC *Genome Browser* [124]. Utilities like the BAM indexer however are tied to their specific associated file format. The more versatile *Tabix* utility supports many text-based file formats, but requires BGZF compressed data. By separating the random access and range indexing concepts, the 2d-tree index is universally applicable to all storage formats supported by the random access back end, currently including plain text, XZ and SHORE-GZIP. Moreover, the ability of indexing data sets not strictly required to be ordered by sequence coordinate is unique among the stated indexing methods. As described, our method indexes chunks of the data set having a fixed byte size. The UCSC R-tree family of indexes differ in this respect by grouping data set elements into tiling windows on the genomic sequence at a maximum resolution of 16 kilobases. This binning approach implies the disadvantage of potential uncontrollable growth of the linear search phase for deeply covered regions of the genome. We further regard the 2d-tree index's homogeneous array layout a substantial implementation advantage.

2.2.4 Improved Compression of Read Mapping Data

To cope with increasingly large amounts of read mapping data, we implement reference based compression, quality reduction, read relabeling as well as a simple transposition algorithm capable of further boosting compressibility of SHORE alignment files and other types of tab-delimited table. Conversion from standard to condensed representations is implemented as a tool SHORE *coal*.

A read mapping by our definition is a multi-attribute entity composed of its *mapping coordinates* on the reference genome as well as a *read alignment* that specifies the exact set of base pairings, traditionally represented in a multi-line format (figure 2.4(a)). The line-based, tab-delimited *MapList* format [1] defined by SHORE stores mapping coordinates, read alignments along with the original read information and attributes describing the reliability of the mapping as well as read pairing information. Alignments are stored in an *alignment string* format resembling that originally introduced by *Vmatch* [131], a sequence matching tool based on enhanced suffix arrays [132]. In the *Vmatch* format, matching positions are represented literally by the matching nucleotide symbol, whereas edit operations are described by the pair of mismatching symbols enclosed in square brackets (figure 2.4(b)).

To adjust to the development of sequencing technology and read mapping tools, we have carefully amended the original alignment representation with a small set of additional features while maintaining full backwards compatibility. With the constant increase of achievable read length, it becomes possible for read mapping tools to align reads across longer stretches of inserted, deleted or polymorphic sequence. For more concise and readable representation of such alignments, we additionally allow consecutive edit operations of the same type to be indicated by two comma-separated strings representing the sequence of the reference and the read, respectively. Reference-based compression as introduced by CRAM in our format is supported by simple substitution of all substrings representing exact matches between read and reference sequence by their length (figure 2.4(c)).

The standard SAM/BAM format splits read alignment information across three attributes (figure 2.4(e)). In this format, the nucleotide sequence of a read (*query*) is stored unaltered. Actual alignment information is separated into a CIGAR string as well as an optional *MD* tag required to specify reference sequence information for mismatch and deletion positions. Omission of read information redundant with the reference sequence via reference-based compression is not supported by the format specification. In the opposite direction, the SHORE alignment string format specifies further enhancements such as inclusion of *soft clipped* sequence enclosed in angle brackets (not shown) to enable full feature compatibility with the CIGAR alignment representation.

Apart from read alignments, read qualities and identifiers can make up for a considerable fraction of the storage requirements of compressed read data (cf. section 2.2.5). SHORE offers a simple algorithm capable of lowering base quality resolution and thereby read quality entropy similar to the quality binning approach taken by *cramtools* [120]. We take a user-specified resolution threshold k to partition each read quality string into sections where the lowest and highest quality value differ by at most k. Each nucleotide is then assigned the lowest quality value found in the respective section. Information on edit operations is prioritized by always storing the exact quality value for read bases representing mismatches or insertions relative to the reference sequence.

Simplification of read identifiers can amount to a significant further reduction of storage requirements. Standard read identifiers are strings automatically generated by the base calling software. For example, Illumina software builds read labels from a sequencing run identifier as well as the reads' coordinates on the flow cell. Due to the randomness of the coordinates as well as the random order of occurrence of the reads in a typical mapping data set ordered by alignment coordinate, the sequence of identifiers to be stored is of high entropy and poor compressibility. Post-mapping systematic relabeling of reads can therefore drastically improve compression ratio. Following relabeling, information content of the sequence of stored identifiers is primarily determined by their actual function, which is identification of associated data set entries such as paired reads or repetitive read mappings. SHORE *coal* offers such relabeling combining simple enumeration by alignment coordinate with a static data set identifier.

In general, generic compression algorithms like DEFLATE or LZMA better pick up redundancy in homogeneously structured data. This suggests that e. g. read quality data might be compressed more effectively when grouped together rather than stored interleaved with the further attributes of the data set elements such as identifiers and read alignments. Isolated storage of these attributes however prohibits data streaming. Furthermore, given presence of variable-length attributes that cannot reasonably be stored in a fixed, predetermined amount of space, random access to the full set of attributes of a certain data set element requires additional index data structures that counter the aim of optimizing compression ratio. A compromise between such an *attribute centric* and the regular *element centric* representation constitutes block-wise coding of the data in an attribute centric representation, with block size chosen small enough to transform a block into an element centric representation on-the-fly.

To explore the effectiveness of such a mixed representation for boosting compressibility, we implemented a simple transposition operation applicable to tab-delimited file formats (algorithm 2.1). Our algorithm takes as parameter a fixed block size k for transformation of the

```
input  : Text $T$, block size $k$
output : Transposed text $T'$
foreach block $B$ in $T$ of size $k$
do
    first_row ← first row of $B$;
    remove the first row from $B$;
    last_row ← last row of $B$;
    remove the last row from $B$;
    append first_row to $T'$;
    if $B$ is a table then
        foreach column $C$ in $B$
        do
            append $C$ as row to
            $T'$;
        end
    else
        append $B$ to $T'$;
    end
    append last_row to $T'$;
end
```

Algorithm 2.1: Block-Wise Text Transposition

input data. Input is subdivided into blocks of k byte. First and last row of a block are defined as the characters up to, and including the first newline, as well as the characters following the last newline marker found in the block. For each block, first and last row are output unaltered, preserving lines spanning block boundaries. Data in between are checked whether they represent a table with fixed number of columns, and if so are transposed swapping rows for columns.

When applied to data represented as a tab-delimited table with a fixed number of columns, the transposition algorithm succeeds in grouping together all attribute values of the same type in a block, lines at block boundaries excluded. For tables with variable number of columns or other data, the input is left unaltered. Applied twice with the same block size parameter value, the original input is guaranteed to be restored.

SHORE supports a transposed format by addition of a special header marking transposed data and specifying the transformation's block size. On file access this header is recognized and input is passed through a filter applying the appropriate block-wise back-transposition. Random access is supported by reading the appropriate block of data, back-transposing and subsequently seeking to the correct offset inside the block.

2.2.5 Compression Results

We used a data set consisting of approximately 6.8 million aligned 40 base pair *Arabidopsis thaliana* reads obtained by Illumina sequencing for evaluation of effectiveness of the measures implemented to improve space efficiency.

In plain *MapList* format, the read mapping data set occupied about 1.1 Gb of disk space. To assess the potential for reduction of total data set size by optimizing encoding and representation of certain attributes, we extracted individual data set columns (table 2.1). Read alignments and

Attribute	Plain Size	Compressed Size	Compression Ratio
read alignments (ref. comp.)	349 Mb (33 Mb)	19 Mb (3.4 Mb)	5.7% (10.3%)
read IDs (relabeled)	227 Mb (117 Mb)	39 Mb (3.2 Mb)	17.5% (2.7%)
qualities (reduced)	341 Mb (341 Mb)	76 Mb (30 Mb)	22.3% (8.8%)
other fields	163 Mb	5.4 Mb	3.3%

Table 2.1: Compressibility of Individual Read Mapping Attributes

qualities accounted for 349 Mb and 341 Mb, each corresponding to approximately 32% of the total storage size, followed by standard read identifiers with 227 Mb (21%) and the collective size of all further attributes (163 Mb, 15%).

Column data were compressed in XZ format at 128 kB block size. Read alignments were thereby reduced to 5.7% (19 Mb) of their original size, further fields to 3.3% (5.4 Mb), whereas read identifiers and quality strings could not be compressed as effectively with compression ratios of 17.5% (39 Mb) and 22.3% (76 Mb), respectively.

By application of reference-based compression, storage size of read alignments was reduced to less than ten percent (33 Mb) in plain text format. This gain was however somewhat diminished in compressed format due to the worse compressibility of the data, resulting in a compressed size of 3.4 Mb at a compression ratio of 10.3%.

Relabeling of the reads resulted in a storage gain of roughly fifty percent relative to the original representation owing to the shorter read identifiers. Vastly improved however was the compressibility of the identifier sequence, resulting in a reduction to just 2.7% (3.2 Mb) relative to plain text format.

Effect of reduced quality value resolution was assessed using the *cramtools* quality binning configuration N40-R8 to allow direct comparison to the CRAM format, although the algorithm implemented in SHORE achieved similar results. The N40-R8 setting preserves quality values of all reference sequence mismatches, while assigning quality values of sequence matches to one of eight possible bins. Application of quality resolution reduction improved compression ratio of the quality data from 22.3% to 8.8%.

We further assessed the effect of various combinations of compression and data reduction configurations on total data set size by application of the SHORE *compress* and SHORE *coal* utilities to the data set in *MapList* format and compare the results with storage of SAM/BAM and CRAM formatted data (table 2.2).

Effect of compression block size was evaluated using SHORE's random access granularity presets "fast", corresponding to 128 kB block size, "medium", 2 MB, and "slow", 64 MB. Compression of the 1.1 GB data set in SHORE-GZIP format at 128 kB block size reduced space requirements to 205 MB, or 19% of the uncompressed file size. Increasing block size to 2 MB further improved space efficiency by just 1.5% to 202 MB storage size, and use of 64 MB blocks did not have a significant advantage over 2 MB.

Selecting the XZ compression format at 128 kB block size resulted in a storage size of 163 MB, 15% of plain data set size and a 20% improvement over SHORE-GZIP at the same block size. Effect of increased compression block size was more pronounced for XZ than for SHORE-GZIP. At 2 MB and 64 MB block size disk usage was reduced to 140 MB and 116 MB, respectively, improvements of 14% and 17% over the respective previous block size.

Block-wise transposition was applied using the block size parameter matching the respective compression block size. At the smallest block size, size of the transformed data in SHORE-GZIP format was 181 MB, a 12% improvement over untransposed storage. Block-wise transposition also augmented the effect of increased block sizes, achieving file sizes of 160 MB at 2 MB and of

File Format	Preprocessing	Compression Format	Block Size	Compressed File Size
SAM	-	-	-	1.3 GB
MapList	-	-	-	1.1 GB
BAM	-	BGZF	default	237 MB
SAM	-	GZIP	128 kB	216 MB
MapList	-	GZIP	128 kB	205 MB
MapList	-	GZIP	2 MB	202 MB
MapList	-	GZIP	64 MB	202 MB
MapList	BT	GZIP	128 kB	181 MB
MapList	-	XZ	128 kB	163 MB
MapList	BT	GZIP	2 MB	160 MB
MapList	BT	GZIP	64 MB	157 MB
MapList	BT	XZ	128 kB	150 MB
MapList	BT, RC	GZIP	2 MB	143 MB
MapList	-	XZ	2 MB	140 MB
CRAM	RC	-	default	135 MB
MapList	BT, RC	XZ	128 kB	132 MB
MapList	BT	XZ	2 MB	123 MB
MapList	-	XZ	64 MB	116 MB
MapList	BT, RC, RL	GZIP	2 MB	114 MB
MapList	BT, RC, QR	GZIP	2 MB	109 MB
MapList	BT, RC	XZ	2 MB	108 MB
CRAM	RC, QR	-	default	103 MB
MapList	BT, RC, QR	XZ	128 kB	100 MB
MapList	BT	XZ	64 MB	96 MB
MapList	BT, RC, RL	XZ	128 kB	95 MB
MapList	BT, RC	XZ	64 MB	84 MB
MapList	BT, RC, RL, QR	GZIP	2 MB	81 MB
MapList	BT, RC, QR	XZ	2 MB	80 MB
MapList	BT, RC, RL	XZ	2 MB	77 MB
MapList	BT, RC, QR	XZ	64 MB	62 MB
MapList	BT, RC, RL	XZ	64 MB	60 MB
MapList	BT, RC, RL, QR	XZ	64 MB	39 MB

BT:	block-wise transposition
RC:	reference-based compression
RL:	read relabeling
QR:	lossy quality resolution reduction (*cramtools* [120] preset N40-R8)

Table 2.2: Comparison of Alignment File Compression Efficiency

157 MB at the 64 MB configuration. This corresponds to advantages of 21% and 22% over the matching untransformed representations.

With XZ/LZMA compression, compression ratios benefited slightly less from block-wise transposition, at the three different block sizes gaining 8%, 12% and 17% over untransposed data to obtain file sizes of 150 MB, 123 MB and 96 MB, respectively.

We further list the effect of additionally applying various combinations of reference-based compression, read relabeling and quality resolution for transposed data encoded in GZIP format with 2 MB block size as well as encoded in XZ format at the three different block sizes. In all compression formats and block sizes assessed, reference-based compression could account for 11% to 12% of additional storage space savings. Added relabeling of reads resulted in file sizes reduced by further 20% in SHORE-GZIP format and by over 28% in the three XZ encoded files. Reduction of quality resolution gave similar improvements, producing slightly higher savings than read relabeling in GZIP compressed data, but had somewhat lesser impact compared to relabeling in XZ encoding.

For comparison of formats, table 2.2 also includes disk usage for the same data set converted to SAM, BAM and CRAM file formats. The SAM format representation of the data set entries was slightly more verbose compared to the *MapList* formatting, resulting in an uncompressed file size of approximately 1.3 GB. The BAM encoded data set was about 14% larger compared to *MapList*, and about 9% larger than the SAM formatted data compressed as SHORE-GZIP at 128 kB block size. When applying reference-based compression only, size of CRAM encoded data ranked slightly better than the *MapList* file compressed in XZ format at 2 MB block size or compressed as GZIP following block-wise transposition and reference-based compression, and slightly worse than the 128 kB block size XZ-encoded file with block-wise transposition and reference-based compression enabled. With additional reduction of quality value resolution, CRAM ranked somewhat worse than the XZ compressed *MapList* file for the same data encoded in 128 kB chunks with block-wise transposition.

2.2.6 Data Storage Considerations

Choice of sequencing data storage formats depends on three areas of application with overlapping, but distinct design goals. Off-line data archiving requires primarily space-efficient representation. Data exchange formats must conform to a highly standardized and stable format specification and allow combined distribution of all data and meta-data in a single file, while directly supporting simple analysis and processing tasks. Finally, on-line storage solutions should be optimized regarding the trade-off between space efficiency and certain access guarantees to provide optimal support for the respective approaches to analysis.

With the implementation of the SHORE storage formats, we demonstrate indexable, space efficient on-line data storage for high-throughput sequencing on the basis of simple text-based file format definitions and stock compression formats. While binary formats dedicated to sequencing data storage such as CRAM should permit further optimization for improved performance and space efficiency, the outlined simple formats and pre-processing methods should provide a valuable benchmark for future efforts.

The implemented indexing solution for compressed sequencing data files forms the basis for rapid data retrieval for iterative analysis and data set exploration. Furthermore, our indexing approach should facilitate the implementation of caching strategies e. g. for implementation of fully interactive visualization of large sequencing data sets.

2.3 A Non-Destructive Read Filtering and Partitioning Framework

To streamline further processing and analysis, it is in general desirable to initially prune high-throughput sequencing data sets of sequence with below-par quality. Sequencing of short polymers such as small RNA, bar-coded multiplexing and diverse other protocols in addition to that require sequence context sensitive read clipping and data partitioning.

With SHORE *import*, we provide a utility to apply such initial read filtering operations, in the process converting sequencing reads from various supported input formats into SHORE's *FlatRead* format and partitioning the data according to the predefined SHORE directory hierarchy as described [1]. This work further develops the application into a modular and extensible read filtering framework.

Data analysis frequently is an iterative process, where at advanced stages it may become clear that an initial choice of filtering parameters for a data set was suboptimal. However, retrieval of the original source data may be hampered by archiving in backup solutions, unless large amounts of duplicate data are to be supported and managed on disk. Our utility was therefore reworked to perform all sequencing read editing and filtering in a non-destructive, reversible manner, exploiting the predefined SHORE directory hierarchy for preventing loss of information. It is thus able to seamlessly recover the raw unprocessed sequencing data sets for straightforward re-filtering or distribution to third parties. Processing facilities provided by the application include custom quality and chastity filtering, low complexity read removal, quality-based trimming of read ends and small RNA adapter clipping, as described [1, 18]. We have further added filters for sequence context-dependent read splitting as well as a versatile demultiplexing system (section 2.4). Defaults of the program are configured such that no filtering is applied automatically, but quality filters indicated by the input format are respected. Supported read filtering operations must be selectively enabled by the user and are applied in a defined order.

SHORE *import* further provides optional functionality for partitioning its output into batches of fixed, user definable size, as well as additionally by the length of the processed reads. As sequence length is a highly significant property e. g. for small RNA data, length partitioning facilitates selection of subsets relevant to the respective analysis. Data partitioning may furthermore offer technical advantages. For example, large sequencing data sets may require a long period time to be written to disk, with a correspondingly high possibility of encountering system failures or other issues in the process. Reduction of the amount of data stored in a single file represents a trivial yet effective measure for limiting the significance of such failures for processing operations that are applied on a per-file basis. As SHORE is capable of merging sorted data sets on-the-fly with minor processing overhead, potential downsides of data set partitioning are minimized.

The original definition of the SHORE run directory hierarchy was amended to support non-destructive read filtering, sample demultiplexing and raw data recovery. The directory hierarchy by default includes four levels (figure 2.5). The root of the hierarchy represents a single run of a sequencing instrument. This directory is named according to the respective output directory parameter, an arbitrary user provided string. Subdirectory names have defined meanings. The first level below the root directory comprises of solely numeric names, representing the individual physically separated lanes of the sequencing run. The second level includes two different types of directory. Cleaned and demultiplexed read data are stored in directories corresponding to the appropriate sample identifier prefixed by the string `sample_`. On the same level, a directory `filtered` stores all data required to restore the original unedited data set.

A further sub-level resides within the sample directories. Valid paired-end sequencing reads are partitioned into directories named according to an integer enumeration of the paired-end read indexes. Reads obtained by single end sequencing or paired-end reads losing their partner due to read filtering are stored in a separate directory `single`. Data files are stored either

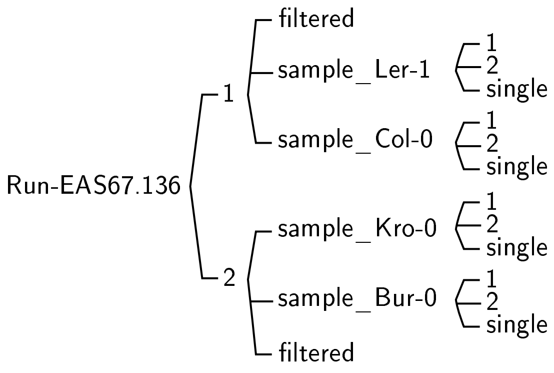


Figure 2.5: SHORE Run Directory Hierarchy

directly in the single and paired-end directories, or below a further sub-level for batching or read length-dependent partitioning.

On completion of a batch of reads, the associated file is sorted to enable accelerated queries on the data set (section 2.2.3) and compressed in a supported indexed compression format (section 2.2.4). Output is formatted in SHORE's *FlatRead* format; results of read trimming and clipping are on request reported via a set of optional tags rather than being applied directly to make filtering results available for further customized processing. Optionally, SHORE directory hierarchy output can be disabled to transform the program into a Unix pipeline-compliant standalone read filtering utility. The application reports basic read trimming and filtering statistics, whereas quality and nucleotide content statistics are implemented as a separate module SHORE *tagstats* (section 2.8).

SHORE *import* defines various different *importers* to be capable of dealing with a variety of input formats. For correct processing of read data, the program must be able to determine all reads belonging to the same read pairing. For pairing recognition to be possible without disproportional time or memory overhead, input data must be ordered according to adequate criteria. Currently available importers allow input where read pairs are provided as separate, correspondingly ordered files such as Illumina GA*Pipeline* directories, sets of FASTQ files or SOLiD color space FASTA files, as well as sorted input in *FlatRead*, FASTQ, Illumina QSEQ or 454 SFF format. Therefore, unordered input data require preprocessing, but can be conveniently passed to the application via Unix pipes using utilities SHORE *convert* and SHORE *sort*.

Our utility automatically recognizes any `filtered` directories residing within SHORE run or lane directories that are passed to it as input. The contents of these directories are automatically parsed to restore the original state of the data prior to any SHORE specific filtering, optionally re-filtering the data set with altered settings. Filtering defined by the original input format is by default restored as well, but can be explicitly ignored. By passing the restored output to the tool SHORE *convert*, raw data can be obtained for distribution in a variety of widely supported formats.

2.4 A Flexible Sequencing Read Demultiplexing System

To minimize cost per sequenced base, next-generation sequencing instruments must be uncompromisingly optimized for maximum throughput, at the expense of generating more data in a single run than would be required for many types of research project. Therefore, most current sequencers feature physically separated *sequencing lanes* or different types of flow cells or flow chips to allow either to distribute overall throughput to multiple samples or to enable reduced-output modes of operation. While physical compartmentalization allows simple sample multiplexing without any requirement for adaptation of work flows, it is tied to the sequencing hardware and therefore offers very limited capabilities.

Bar coded multiplexing strategies have thus quickly become the method of choice to achieve greater operational flexibility with the primarily throughput-optimized high-end sequencer models. Bar coding protocols offer near-perfect scalability by allowing to distribute per-run sequencing depth to an almost arbitrary number of samples.

Bar coded sequencing data require that prior to further analysis demultiplexing is performed in software to separate the respective samples as well as sample sequences from the artificial bar code oligomers. The SHORE *import* utility integrates a flexible demultiplexer capable of dealing with a wide variety of standard and custom bar coding protocols. The respective bar coding setup is provided by the user in a simple tab-delimited sample sheet format. Our tool supports all combinations of $5'$ bar code ligation and index read protocols, variable length bar codes and is able to take advantage of additional common subsequences such as e. g. restrictions sites in RAD-Seq applications (section 1.3).

2.4.1 Overview

Bar coding is realized by fusing sample DNA with index oligomers that serve as an unambiguous sample identifier. Ligation of these bar codes can be achieved in many different ways, each coming with a different set of advantages and disadvantages regarding library preparation, cost and flexibility. Therefore, various bar coding protocols as well as different bar coding kits provided by sequencing instrument vendors are available.

Using the Illumina platform, index sequences may be associated with particular sequencing primers to be read as a separate *index read*. Other protocols ligate bar codes to be read in one go with the sample DNA, resulting in index oligomer and sample DNA being fused into a single sequence read. Moreover, paired-end sequencing protocols imply various possible bar coding configurations, with oligomer labels either attached to the start of just one of the sequencing reads, identical labels attached to either read, or different labels inserted at the start of the first and the second read. Labeling reads individually, or even combining $5'$ and index read bar codes, enables large numbers of samples to be discriminated using only few different oligomer tags, at the cost of a more elaborate sample preparation procedure and an increased overhead in sequencing cycles.

Often multiple different labels are chosen to identify each respective sample. For example, strong bias in the sequenced sample may in certain cases interfere with sequencing instrument calibration, and due to that the set of bar code oligomers must be carefully compiled to achieve a largely balanced nucleotide composition in the multiplexed DNA.

Demultiplexing via tools provided by the sequencing instrument vendors is usually only possible for the vendor-specific indexing protocols. With the increasing variety in different bar coding protocols, we have gradually developed bar code matching integrated with the SHORE *import* utility into a flexible demultiplexing solution that is configurable for all compositions of index read and $5'$ ligated bar coding via a simple *sample sheet* format. We employ a read oriented format to avoid manual enumeration of all valid bar code combinations for multi bar code configurations. Prior to resolving tuples of bar codes to the appropriate sample identifiers, bar code matching is performed individually for each read for improved performance and detection of potential mislabeling. The current bar code matching implementation is able to tolerate a fixed, user-specified number of mismatches and no gaps.

```
#?sample  read  barcode
Col-0     2     TTCACG
Col-0     2     GGATGT
Ler-1     2     CTAGGC
Ler-1     2     AGACCA
```

Listing 2.1: Example of a Simple Demultiplexing Sheet

2.4.2 A Format for Description of Multiplexing Setups

A SHORE demultiplexing sample sheet is a simple tab-delimited text table with named columns. The recognized column names are `lane`, `sample`, `barcode_group`, `read`, `barcode`, `extbarcode` and `barcode_type`.

The sample sheet is parsed utilizing a generic table input API provided by the SHORE library. Column names are defined in the header line, the first non-empty line of the file that is not a line comment, or is introduced with a special comment tag "`#?`". Columns are recognized by name and may occur in arbitrary order; further columns may be present, but are ignored.

The only mandatory column, named `sample`, defines the sample identifiers that bar codes are to be translated to. All further columns may be added in arbitrary combinations and order. A typical simple index read demultiplexing configuration is described by additionally specifying the `read` and `barcode` columns (listing 2.1), with `read` specifying the index of the read that contains the bar code tag and `barcode` the actual sequence of the bar code oligomer. The read index column may be omitted, indicating that all sequence reads of a pair are attached to identical bar codes.

Index read and $5'$ bar coding require suitable treatment of the respective bar code oligomers. While $5'$ bar codes must be clipped, with the remaining part of the read sequence to be retained, index reads must be removed completely from the data set. The default implemented in SHORE is to apply bar code clipping if the sample sheet entry refers to the first read or if the read index was omitted, and to filter bar-coded reads where the read index is greater than one. The `barcode_type` column allows to explicitly control this behavior. SHORE accepts bar code types `read`, `5prime` and `none`, where the type is associated with the read index, i. e. all of a lane's entries with the same read index must also specify the same bar code type. Bar codes of type `read` will trigger filtering of the entire bar code associated read, whereas `5prime` bar codes will be clipped.

For configurations with bar code information distributed across multiple sequence reads, several entries with different read index values are for each sample added to the definition (listing 2.2, lines 4–7). The sample sheet entries will then be grouped on the sample identifier, with each possible combination of bar codes with differing read indexes considered a valid bar code tuple for the respective sample. Sample sheet entries with the special sample identifier "$*$" are interpreted as valid for each sample specified for the respective sequencing lane (e. g. lines 16–17). For example, listing 2.2 defines two valid bar codes for the *Ler-1* sample for each of the three sequencing reads, and thus $2^3 = 8$ valid 3-tuples of bar codes that can be resolved to that sample.

Automatic combination of all entries listed for a sample can be controlled by the user via specification of the `barcode_group` column. Sample sheet entries with different bar code group values are not able to form a valid bar code tuple (e. g. listing 2.2, lines 10–13). For example, the first read bar code specified in line 10 for the *Col-0* sample may only combined with the second read entry from line 11 and not the one from line 13. Furthermore, a combination of line 10 and 13 would collide with the bar code tuples including entries from line 4 and 7, which are already specified to resolve to the *Ler-1* sample. As with the sample column, the bar code group may be set to "$*$" to specify and entry that refers to all groups of the respective sample in the respective

```
 1  #?lane  sample  barcode_group  read  barcode  extbarcode  barcode_type

    # Bar codes for the Ler-1 sample.
    1       Ler-1   0              1     AACT     TGCAG       5prime
 5  1       Ler-1   0              1     TAGC     TGCAG       5prime
    1       Ler-1   0              2     AACT     *           read
    1       Ler-1   0              2     CCCT     *           read

    # Bar codes for the Col-0 sample.
10  1       Col-0   0              1     AACTT    GCAG        5prime
    1       Col-0   0              2     GGAC     *           read
    1       Col-0   1              1     TTGCT    GCAG        5prime
    1       Col-0   1              2     CCCT     *           read

15  # Third read has the same bar codes for all valid samples.
    1       *       *              3     GGAC     *           5prime
    1       *       *              3     TTGC     *           5prime

    # Lane 2 is not multiplexed, discard the index read.
20  2       Bur-0   0              2     *        *           read
```

Listing 2.2: Example of a Full Demultiplexing Sheet

lane (e. g. lines 16–17).

For certain applications, the identity of several nucleotides immediately following 5′ bar code sequences is known, like for example restriction site sequence in RAD-Seq (section 1.3). While such sequences can be exploited to correctly assign each read to the appropriate sample, they usually should in contrast to the bar code oligomers not be removed from the output. Parts of the recognition sequence that should not be clipped from the read can be specified in the extended bar code (`extbarcode`) field of the table. Internally, the sequence to be recognized is contructed by concatenation of the `barcode` and `extbarcode` fields, while the division of sequence among both fields is translated into a bar code cut position. For bar code types other than `5prime`, the split into bar code and extended bar code has no effect. The bar code cut position is determined in the context of the respective bar code tuple, i. e. for the same recognition sequence different samples may define a different split between bar code and extended bar code, as demonstrated by lines 4 and 10 of listing 2.2. If no part of the recognition sequence is to be removed from the output, then the entire oligomer can be provided as extended bar code, with column `barcode` either omitted or set to "$*$".

If neither bar code nor extended bar code are provided with a value other than "$*$", then the respective sample sheet entry will match any read sequence. With bar code type either `none` or `5prime`, such an entry can be utilized for assigning a certain sample identifier to an entire sequencing lane. On the other hand, this property may be exploited to completely remove all reads with a certain read index from the output (e. g. listing 2.2, line 20).

The sample sheet column `lane` serves to allow independent demultiplexing specifications for different sequencing lanes in a single sample sheet file. Rows with differing sequencing lane fields are completely independent of each other. If the sequencing lane column is omitted, then the entire demultiplexing specification is considered valid for all lanes of the instrument run.

2.4.3 Barcode Recognition and Sample Resolution

Despite current sequencing technologies having reached rather high levels of stability, single cycle quality dropouts can not completely ruled out. Therefore, algorithms matching reads and bar code oligomers should be able to tolerate at least limited hamming or edit distance. Rather than directly mapping each tuple of reads to one of the potentially huge number of possible bar code tuples with a certain cumulative edit distance, our algorithm proceeds by individually assigning each sequence read to a specific bar code.

Sample demultiplexing is thus broken down into three successive operations. First, each sequencing read is matched to the set of allowed bar code sequences associated with the respective read index. Subsequently, given all sequences for a flow chamber spot could be matched to one of the oligomers, the resulting tuple of bar codes is mapped to the appropriate sample identifier. Following successful bar code resolution, reads are then pruned of the index oligomers as required.

The initial bar code recognition step tolerates up to a fixed, user specified hamming distance between prefix of the sequencing read and bar code sequence. We perform a staged prefix matching to precomputed sorted sets of sequences, where each stage corresponds to a certain hamming distance to the exact tag sequences. The set of sequences for each stage is generated from the set of valid tags for the respective read index by recursively mutating the sequences of the respective previous set, with the mutated sequences carrying a pointer to the original unmodified bar code. Potential collisions in bar code resolution are automatically detected and the corresponding entries pruned. At each stage, the read being assessed is used to query the set of tag sequences utilizing a binary relation that defines a pair of sequences as equivalent given that one is the prefix of the other. If the query returns no result, the algorithm proceeds to the next stage, until the user specified number of mismatches is reached and the read is discarded as unresolvable.

Given a tuple of reads could thus be successfully mapped to a tuple of valid index sequences, the set of valid bar code tuples is interrogated to resolve the bar code combination to the appropriate sample identifier. Each of the reads is then either pruned of its $5'$ end or flagged as filtered according to the sample sheet's bar code type specification. Since the bar code cut position can in general only be resolved in the context of the complete bar code tuple, this clipping operation is postponed until sample identifier and bar code group have been resolved.

Due to its restriction to hamming distance rather than edit distance, our demultiplexer is of limited applicability to indel-prone sequencing technologies such as 454, Ion Torrent of SMRT sequencing. However, in certain cases perfect matching may be sufficient, or probable indels might be represented explicitly in the sample sheet. In setups requiring very long bar code oligomers and a high mismatch tolerance, recursive generation of the mutated bar code sets threatens to become overly resource intensive. However, bar code matching is implemented as one of several isolated modules with a simple interface, such that additional matching options might easily be added in the future as required, e. g. utilizing the dynamic programming alignment algorithms provided by the SHORE library. For highly customized context sensitive read clipping and splitting allowing for gaps and user defined scoring schemes, we provide a separate utility SHORE *oligo-match* (section 2.5).

2.5 Versatile Oligomer Detection and Read Clipping

In addition to sample demultiplexing, a wealth of different library preparation protocols and sequencing applications exist that involve ligation of certain adapter or linker sequences, or otherwise require sequence context aware clipping or splitting of the unprocessed read sequences. Depending on the protocol or application, the respective recognition sequences may be degenerate or truncated in various ways. Therefore, computational tools are required that are capable of detecting optimal sequence matches given defined constraints, and can utilize detected matches for pruning, splitting or otherwise manipulating sequencing read data according to user requirements.

In this work, a highly configurable utility SHORE *oligo-match* for pair-wise sequence matching and match based read manipulation was implemented. The program utilizes dynamic programming alignment algorithms with customized alignment matrix and backtrace initialization, modes of backtracing and match filtering. Various modes of sequencing read manipulation or annotation are provided, and full match information may be retrieved for advanced customization needs.

Our application supports both threaded as well as distributed memory modes of parallel operation. It is applicable to matching, manipulating and filtering even large data sets of sequencing reads by a small set of provided oligomer sequences.

2.5.1 Overview

Optimal pairwise sequence alignments can be computed using well known dynamic programming algorithms, given that no larger-scale sequence rearrangements need to be accounted for. Pairwise alignment algorithms are grouped into two classes, *global alignment* methods like the Needleman-Wunsch algorithm [133] and *local alignment* such as the Smith-Waterman algorithm [134]. However, matching pairs of short sequences such as high throughput sequencing reads and adapter oligomers usually implies specific sequence overlap configurations, and therefore represents neither a global nor local alignment use case.

Mate pair sequencing libraries for Roche 454 Genome Sequencer instruments are constructed through the *Cre/lox* system. *Cre* is an enzyme that catalyzes site specific recombination at pairs of *lox* sequences. The protocol exploits this affinity for circularizing multiple kilobase DNA fragments. Circularized DNA is then cut in relative proximity to the recombination site to obtain a shorter linear fragment consisting of the inverted ends of the original fragment joined by a linker oligomer that includes a *lox* sequence. To obtain mate pair reads, the compound insert is sequenced and then split computationally adjacent to the detected location of the linker oligomer.

Cre/lox mate pair sequencing protocols have also been introduced to Illumina technology [135]. Due to the by comparison short read lengths, both ends of the circularized fragment cannot in general be retrieved as a single sequence read and thus the insert is sequenced from both ends. The linker oligomer may be detectable in none, only one, or both sequences of a read pair, depending on read length, linker position and size of the insert.

Recognition and removal of adapter oligomers is further required in all situations where there is an overlap between the distributions of insert size and sequencing read length. In small RNA sequencing, all relevant sequencing reads will contain the $3'$ sequencing adapter due to the short length of the sequenced polymer. Alternative multiplexing protocols are enabled by this property where the bar code oligomer is ligated to the $3'$ library adapter. In total RNA libraries adapter sequences are in contrast only picked up in a minority of reads.

Adapter and linker recognition require finding the optimal pair-wise alignment with the constraint of specific sequence overlap configurations. We define five different keywords `global`, `local`, `dangling_qry`, `dangling_ref` and `dangling_any` for description of pair-wise sequence overlap configurations. At either end of the sequence alignment there are four different possible overlap configurations — overhang of the reference sequence (`dangling_ref`), overhang of the query sequence (`dangling_qry`), unaligned sequence in both reference and query (`local`) as well as no sequence overhang (`global`) — and therefore 16 different alignment configurations in total (table 2.3).

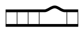
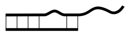
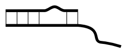
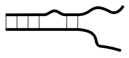
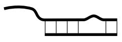
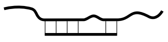
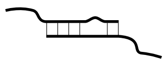
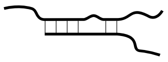
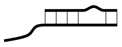
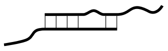
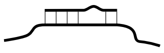
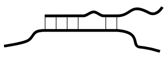
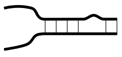
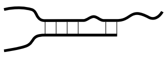
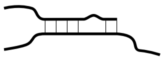
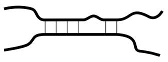
	Alignment	Matrix initialization (left end)	Backtracing (right end)
1		global	global
2		global	dangling_qry, dangling_any
3		global	dangling_ref, dangling_any
4		global	local
5		dangling_qry, dangling_any	global
6		dangling_qry, dangling_any	dangling_qry, dangling_any
7		dangling_qry, dangling_any	dangling_ref, dangling_any
8		dangling_qry, dangling_any	local
9		dangling_ref, dangling_any	global
10		dangling_ref, dangling_any	dangling_qry, dangling_any
11		dangling_ref, dangling_any	dangling_ref, dangling_any
12		dangling_ref, dangling_any	local
13		local	global
14		local	dangling_qry, dangling_any
15		local	dangling_ref, dangling_any
16		local	local

Table 2.3: Dynamic Programming Alignment Modes

The configuration depicted in the first column indicates the type of end overhang that should not be penalized by the alignment algorithm's scoring method, with the lower line defined as *reference* (`ref`) and the upper line as *query* (`qry`). The different end alignment modes form a hierarchy of constraints, with `dangling_any` a subset of `local`, `dangling_qry` and `dangling_ref` subsets of `dangling_any`, and `global` a common subset `dangling_qry` and `dangling_ref`.

The term *query* will be used in the following to always indicate the sequencing read, and *reference* may thus refer to a possibly much shorter oligomer sequence. Alignment of a short read to a part of a genomic reference sequence constitutes an example of overlap configuration 11, defined by the keyword pair `(dangling_ref;dangling_ref)`. Detection of sequencing adapters or $3'$ bar codes in small RNA sequencing data corresponds to configurations 6 or 7, depending on whether the read contains the entire or only the partial adapter sequence. This subset of configurations is defined by the pair `(dangling_qry;dangling_any)`. Detection of bar codes in $5'$ bar-coded sequences is represented by case 2 `(global;dangling_qry)`.

For Cre/lox-type recombinant mate pair sequencing varied overlap constraints may be appropriate depending on the desired sensitivity-specificity tradeoff. Conservative detection of linker sequences in valid 454 type mate pairs corresponds to configuration 6. Illumina Cre/lox mate pair protocols obtain read pairs where the linker sequence is expected towards the end of either read (configuration 6 or 7). Occasionally one of the enzymatic cuts of the circular DNA may also occur inside the linker DNA. Such cases are described by configuration 10. The subset of configurations 6, 7 and 10 constitutes a case of mutually dependent end alignment configurations. Thus, it can not be accounted for by a single keyword pair, but the two pairs `(dangling_qry;dangling_any)` and `(dangling_ref;dangling_qry)` (or alternatively, `(dangling_any;dangling_qry)` and `(dangling_qry;dangling_ref)`).

The SHORE *oligo-match* utility is capable of for each sequencing read selecting the optimal alignment or alignments out of multiple pairs of end alignment modes, multiple *reference* oligomers and optionally their reverse complemented sequence.

By utilizing a fully customizable 16x16 scoring matrix, the program enables adjustable handling of ambiguous IUPAC nucleotide codes as well as asymmetric base mismatch penalties with respect to the direction of the match.

A pair-wise sequence mapping is composed of two pairs of end coordinates as well as the alignment describing actual base pairings and sequence gaps. To increase specificity of read clipping and splitting operations, it is desirable to ensure that optimal pair-wise mappings feature a unique pair of end coordinates, whereas potential alternative alignments can be considered irrelevant. Our alignment algorithm is capable of either generating an exhaustive list of all possible alignments, a list featuring a representative of all alignments with a different pair of end coordinates, or just a single representative for each pair-wise mapping, which is optionally assessed for uniqueness of end coordinates.

The utility provides filters for pair-wise mappings with respect to uniqueness of oligomer selection and end coordinates. Alignments valid following filtering may be comprehensively reported and applied to clipping or splitting sequencing reads at either, or at both ends of the detected oligomer.

2.5.2 Dynamic Programming Alignment and Backtracing Pipeline

Our alignment pipeline proceeds in three subsequent passes. Initially dynamic programming alignment of the sequencing read is performed to each oligomer and for each pair of provided end alignment modes. The optimal alignment or alignments are passed to the backtracing module. Finally, generated traces are filtered and may subsequently be applied to read manipulation.

While alignment scores become available after the initial stage, filtering is delayed until after backtracing for reasons of threshold calculation.

The end alignment mode for the left end of the alignment determines the mode of alignment matrix initialization and alignment score calculation. With left end alignment mode `local` initialization and score calculation proceeds according to standard local alignment, with first row and column initialized to zero and the alignment score at each field of the matrix truncated at zero from below. Left end mode `dangling_any` utilizes the same matrix initialization, but does not clip alignment scores at zero. With `dangling_ref` and `dangling_qry`, only the first row or column is zero-initialized, respectively, given the width of the matrix is determined by the size of the reference sequence. Sequence overhangs for the respective other sequence are valued with the regular gap penalty. Mode `global` does not perform pre-initialization of the matrix, as in standard global alignment.

For selection of the best out of multiple permitted pairs of end alignment mode for a *reference* oligomer, multiple passes of dynamic programming alignment must be performed due to the distinct requirements of matrix initialization. For each distinct left end alignment mode a corresponding right end alignment mode may be defined. However, alignment is only performed if the configuration is not included by a different pair of modes specified, i. e. if the constraint on the right end of the alignment is weaker than that specified by the next left end mode that includes the current one. Due to the hierarchical nature of tolerated sequence overhang configurations there is nonetheless a chance that the same alignment will be produced multiple times by different keyword pairs. For example, `(dangling_ref;dangling_qry)` and `(dangling_qry;dangling_ref)` both define supersets of the `(global;global)` configuration. Such redundancies can be eliminated prior to backtracing by determining the subset of the respective end alignment mode that has already been covered by previous alignments and excluding it by assigning corresponding fields of the alignment matrix the maximum penalty.

For backtracing initialization and alignment score calculation, the alignment algorithm keeps track of values and locations of last row, last column and global score maximums for the matrix. Mirroring matrix initialization, backtracing starts at global score maximums for right end alignment mode `local`, at last row and last column maximums for modes `dangling_ref` and `dangling_qry`, respectively, at the combined last row and last column maximums for `dangling_any`, and at the lower right corner for `global`.

Whenever the backtracing algorithm encounters multiple possible fields of origin for an alignment score, alternative paths are stacked for later completion, unless retrieval of only a single representative alignment was requested. While exhaustive tracing of all possible alignment paths can be combinatorially unfavorable, generating a representative for all distinct pairs of mapping end coordinates can be performed efficiently. For this purpose, each field of the matrix that has been reached by the backtracing algorithm from the same right end coordinates is marked as visited. If a field marked as visited is reached from one of the stacked alternative paths, the respective path is aborted and the algorithm proceeds to assessing the next path.

For read clipping purposes it is typically only relevant whether an oligomer can be assigned to unique coordinates on the sequencing read. For assessment of coordinate uniqueness, the backtracing algorithm proceeds only until alternative end coordinates have been encountered, and the alternative trace is not emitted. If the backtrace completes without encountering coordinates different from the primary trace, its end coordinates are marked as unique.

Reliability of a sequence match is determined by both the length of the match and its relative amount and type of edit operations. Alignment score filtering is therefore configured by setting slope and offset of a linear function. By default, the function's parameter is the length within the shorter sequence out of *reference* and *query* that is spanned by the match. Alignments with a score below the threshold thus calculated are not propagated to the list of results and sequencing read manipulation. Alternatively, the score threshold may be set up to be calculated using the total length of the selected sequence as the function parameter.

2.6 A Parallelization Front-End for Short Read Alignment Tools

A great variety of sequencing read mapping and alignment programs are nowadays available to users. While most of these applications are similar in operation, each comes with its own specific user interface and usage peculiarities. Despite the huge selection of different implementations and extensive optimization of the underlying mapping and alignment algorithms, read mapping is furthermore still one of the computationally most expensive steps for resequencing applications (section 1.5.3). While parallel processing can often solve the issue of extensive run times, parallelization support is not uniform among all available solutions. Therefore, as each read alignment tool comes with its own set of trade-offs, strengths and weaknesses, users are often left to decide between performance and required or desired features, and frequently need to adapt to different interfaces.

The SHORE *mapflowcell* application is a parallelization, pre- and post-processing wrapper for a variety of widely used and freely available short read mappers [1]. In addition to shared memory parallelization, the application has been reimplemented supporting networked distributed memory architectures through the Message Passing Interface (MPI) standard. Further added capabilities include automatic adjustment of parameters for handling of data sets with heterogeneous read lengths, as well as straightforward integration of mapping results obtained using different aligner back-ends or sets of mapping parameters. The program thus constitutes a meta-alignment tool able to amend mapped sequencing read data sets using different aligners, parameters or additional reference sequence information. Such multi-pass read mapping may especially benefit computationally expensive approaches such as spliced read mapping.

SHORE *mapflowcell* currently provides a common user interface to the tools *GenomeMapper*, BWA, *Bowtie*, *Bowtie2*, ELAND and BLAT. User parameters are automatically transformed into the appropriate equivalent for the respective mapping back-end and results consistently reported in compressed, seekable SHORE *MapList* format easily convertible to various data exchange formats such as SAM.

Ion or pyrosequencing read data sets as well as other types of sequencing data set following quality based read trimming (section 2.3) or sequence context specific read clipping (section 2.5) feature read length distributions with broad support range. However, many read mapping tools only support a set of static alignment parameters provided at program startup, resulting in below-par mapping specificity for reads on the lower tail of the read length distribution and insufficient sensitivity for reads at the upper end of the length spectrum.

The SHORE *mapflowcell* application automates read length-dependent selection of all relevant alignment parameters such as maximum tolerated edit distance, gap extension lengths or seed sizes. These user parameters may be provided as either absolute values or percentage of read length; additionally, edit distance may be bounded according to the alignment seed lemma. Given a certain read length l and edit distance d, a read alignment is guaranteed to contain a perfectly matching seed of size $s = \lfloor l/(d+1) \rfloor$, and therefore the maximum edit distance that guarantees finding a seed for each valid mapping is $d = \lfloor l/s \rfloor - 1$. We provide two modes of reducing the initially specified edit distance. A mode "strict" limits maximum edit distance to $d_{max} = \lfloor l/s \rfloor - 1$, thereby ensuring that either all valid alignments with optimal edit distance are found for a read, or it is left unmapped (further seed selection heuristics implemented by the mapping tool excluded). An alternative setting "on" sets a weaker limit $d_{max} = \lfloor l/s \rfloor$, such that some of multiple mappings with the same edit distance may be missed, but reads are left unmapped if valid alignments at lesser edit distance can not be excluded.

Sequencing of DNA that is highly diverged from available reference genome sequences as well as transcriptome sequencing due to intron exon junctions imply that many sequencing reads can not be mapped to the reference as a contiguous unit. Such data must be mapped using specialized spliced sequencing read mapping programs like PAL*Mapper* [94] or generic matching tools such as BLAT [136]. Specialized spliced mapping algorithms may however represent a suboptimal choice for mapping the entire data set, either for reasons of performance or specificity. A different, yet related problem pose cases where the respective reference sequence is too large to be handled with a certain combination of mapping tool and available hardware. Our utility is able to mitigate these issues through automatic combination of the results of multiple read mapping passes using different alignment tools, parameters or reference sequence information. Two different re-mapping modes offer either full reassessment of the entire data set utilizing the information of prior read mapping passes, or processing and integrating additional mapping results for read previously not mappable to the reference genome.

For robust handling of variable reference sequence information provided to the program, SHORE *mapflowcell* maintains a sequence dictionary used to record reference sequence identifiers, metadata and checksums along with the alignment result file. With each read mapping pass the reference dictionary is updated appropriately and possible additional sequences are assigned new unique sequence identifiers as required.

For reassessment of read previously not mappable to the reference genome, mapping proceeds as in single pass application, and results are the union of all newly discovered read mappings with the results of the previous pass. For full refinement of entire data sets, both previously mapped and unmapped reads become subject to realignment. Previous mapping results are however factored into calculation of alignment parameters and determination of batch identity. On completion, results of the current and previous mapping run are merged while at the same time pruning read alignments rendered suboptimal by the newly obtained results from the mapping data set.

Read mapping tools incorporating computationally more expensive spliced alignment algorithms further emphasize the need for efficient and capable parallelization of the mapping process. While many aligners implement simple threaded parallelization, support is not universal, and only a few mapping tool-specific solutions exist for distributed memory settings or cloud environments, like e. g. *CloudBurst* [95]. SHORE *mapflowcell* implements a simple batch-based parallelization approach, in which automatically generated batches of input data may be distributed to either an alignment tool running in multi-threaded mode, to multiple aligner processes instantiated on the same machine, to alignment tools running within MPI processes distributed over a network, or a hybrid combination of all of the above.

SHORE *mapflowcell* defines a binary relation on read data by which reads are equivalent if they are set to be aligned using the same combination of static mapping parameters, and divides the data set into batches accordingly. Reads are partitioned into batch-specific files in SHORE's temporary file directory and, on reaching a user specified maximum batch size, submitted for parallel processing. Parallel operations include actual read mapping as well as format conversions and sorting where required. With fine tuning of the maximum batch size parameter, a simple measure is provided for adjustment of the tradeoff between minimal parallelization overhead and improved load balancing. For distributed memory parallelization, the temporary file location specified is required to be shared over the relevant network, with only meta information on each batch being transmitted via MPI channels for simplicity of implementation. On completion of all batches related to a specific input file, mapping results are automatically merged back into the permanent storage location.

2.7 Robust Detection of ChIP-Seq Enrichment

Investigation of protein-DNA interaction by ChIP-Seq can help to further our understanding of cellular regulatory networks (section 1.3.2). Primary data analysis for ChIP-Seq experiments requires software tools that can pinpoint hundreds to thousands of putative protein binding sites from data sets of high-throughput sequencing read alignments (section 1.5.6).

This section discusses the peak calling pipeline SHORE *peak* integrated into the SHORE analysis suite. The utility is able to discriminate true enrichment from common artifact patterns using various robust heuristics, calculates false discovery rates (FDR) by interrogation of control experiments and implements a unique "joint detection, separate evaluation" approach to handling of replicate experiments. The program outputs locations of possible binding regions together with their FDR and various additional features potentially useful for ranking, filtering or fine-tuning the detection. It further enables users to generate graphical representations of peak regions, put detected loci into the context of available genome annotation or to extract binding region sequences for subsequent motif sampling and analysis.

Our pipeline is pre-configured for straightforward detection of defined loci of ChIP-Seq enrichment as produced by transcription factor (TF) binding sites. Settings may however be adjusted to support the detection of clusters of enrichment that occur e. g. in data representing states of histone modification. Individual algorithms are implemented as pluggable modules and have been brought over to multi-purpose tools SHORE *coverage* and SHORE *count*, allowing fine-grained control and flexible applicability of enrichment detection in diverse scenarios such as MNase-Seq assays or assessment of copy number variation (CNV).

The SHORE *peak* pipeline has been outlined [58, 137] and further employed [138–141] in multiple biological publications.

2.7.1 Overview

Were sequencing reads randomly sampled from the genome, sequencing depth should be distributed according to the Poisson distribution whose parameter equals the average depth [142]. Although high-throughput sequencing is known to be highly non-random, with biases that bring about considerable deviation from the Poisson model (section 1.4), the distribution can still present a powerful tool for the detection of irregularities in sequencing depth at high sensitivity. Our peak calling algorithm consists of two major steps. The first pass is a detection phase that employs a relaxed Poisson assumption to the read alignment data generated through the ChIP-Seq experiment to identify regions of putative enrichment at high sensitivity. In the subsequent second pass implemented to improve specificity of the prediction, these *candidate regions* are subjected to several robust empirical rules for artifact removal and further tested against control experiment data to assess significance of the enrichment.

2.7.2 Correcting for Duplicated Sequences

The ChIP-Seq procedure yields quantities of DNA that are typically orders of magnitude below the amounts retrieved by other methods of DNA sampling. The excess PCR rounds required to obtain a compliant sequencing library often bring about significant amounts of duplicated sequences, reducing library complexity. PCR may entail significant sequence bias (section 1.4) with erratic rates of fragment duplication.

Duplicate sequences represent the same fragment of DNA and therefore violate the assumption that each sequencing read is sampled independently from the set of sequences under examination. During detection of sequence enrichment, the presence of duplicate sequences must thus be explicitly accounted for by the predictive model, or these sequences must be pruned from the data set prior to application of the actual detection algorithm. Most current ChIP-Seq analysis algorithms including SHORE*peak* implement a pruning approach.

In the presence of sequencing errors or variable read length, PCR duplicates cannot be recognized by simply collapsing all reads having the same nucleotide sequence. Furthermore, at enriched loci identical sequences may be sampled independently. In the sequencer output, these identical sequences are thus indistinguishable from PCR artifacts.

Duplicate recognition is therefore implemented as a filtering step that is applied after mapping the reads to the reference genome. ChIP-Seq algorithms typically restrict usage of duplicate reads by applying a static limit to the number of reads having their $5'$ end mapped to the same position of the genome. While reliably removing duplicated sequences, the approach does not distinguish between duplication events and independent sampling, and thus suffers from saturation once all reference positions are becoming occupied.

We therefore adaptively remove PCR duplicates, using a heuristic based on the assumption that the number of non-duplicated reads n^*_x that have their $5'$ end at any position x of the genome is sampled from a Poisson distribution with unknown distribution parameter λ_x, and that each of these reads is represented on average by an unknown number of copies α_x. Given the assumption holds, α_x equals the dispersion index $D_x = \sigma^2_x/\mu_x$ for the read count distribution including duplicates. We further assume that all λ_i and all α_i for positions that are in close vicinity take similar values, and that the read count distribution for any position x can therefore be approximated using a short range of positions adjacent to x. Based on these assumptions, we estimate α for each position using a sliding window. In a window, the dispersion index is estimated as the ratio of the empirical variance and mean of the read counts. To increase the adaptiveness of the sliding window calculation, the ratio is calculated independently for a *left* window that includes all positions from $x - k$ to x, and a *right* window including positions x to $x + k$

$$\hat{D}_{left} = \frac{s^2_{(x-k)..x}}{\bar{n}_{(x-k)..x}}$$

$$\hat{D}_{right} = \frac{s^2_{x..(x+k)}}{\bar{n}_{x..(x+k)}}$$

with s^2 and $\bar{n}$ the empirical variance and mean over the positions indicated, respectively. The left and right results are then averaged using the *root mean square* (RMS) of both values to obtain an estimate $\hat{\alpha}_x$ of α_x

$$\hat{\alpha}_x = \sqrt{\frac{1}{2} \cdot (\hat{D}^2_{left} + \hat{D}^2_{right})}$$

The division into left and right windows should allow for sharper boundaries between enriched and background sites. RMS average over the two windows was chosen to ensure that re-evaluation using a scaled read count vector $\mathbf{n}' = \mathbf{n} \cdot 1/\hat{\alpha}_x$ as input will always yield $\hat{\alpha}'_x = 1.0$.

In scenarios where it is desirable to retain the complete set of sequences with their associated base qualities, the estimated multiplicity factor $\hat{\alpha}_x$ may be incorporated into the analysis by assigning each read starting at x a weight of $1/\hat{\alpha}_x$ during calculation of sequencing depth. If retaining all sequences is not a concern, the maximal number of reads may alternatively limited to $\lfloor n_x/\hat{\alpha}_x \rfloor$, with n_x the observed read count at a position x. The limiting approach is currently implemented in SHORE, discarding all excess alignments when the read count limit for the respective position has been reached.

The duplicate correction algorithm does not provide any guarantees regarding the correctness of $\hat{\alpha}_x$. However, given that the variance of the read counts of adjacent positions is likely to exceed the Poisson assumption, average multiplicity should be overestimated. The algorithm is therefore conservative in the sense that the true number of non-duplicated reads per position will be underestimated. In contrast to imposing a static limit on the number of reads accepted at each position, the adaptive heuristic still allows to discriminate different levels of extreme enrichment as well as peak calling in presence of deep background coverage.

2.7.3 Detection Phase

The ChIP-Seq procedure enriches for short DNA fragments with above average affinity to the protein of interest. One or both ends of these fragments are subsequently sequenced, allowing detection of its origin on the reference genome assembly. In the following, the enriched DNA fragments subjected to sequencing will be referred to as *inserts*, whereas the sequenced ends of the fragments will be referred to as *reads*. The number of inserts putatively overlapping a position will be termed *insert depth*, and the number of reads overlapping the position *read depth*.

The primary peak detection phase, resembling that of MACS [104] (section 1.5.6), is realized as a sliding window analysis of insert depth. A window of fixed, user-selectable width is shifted along the reference sequence in single base steps. In each step, the algorithm calculates the average depth of insert coverage over the current window. In the case of paired-end sequencing, insert depth is calculated from the ranges defined by read pairs. For single end sequencing, each read is extended in $3'$ direction to match the estimated average insert size. To exclude regions not accessible to the sequencing experiment, positions with sequencing depth zero are ignored, i. e. the average depth is calculated as

$$\bar{d}'_W = \frac{\sum_{x \in W} d(x)}{|\{x \in W : d(x) > 0\}|}$$

where W is the set of reference sequence positions included by the sliding window and $d(x)$ signifies the depth at a position x. Since the sliding window may or may not contain enriched sites, $\bar{d}'_W$ may be regarded a conservative estimate of the minimum average background signal over the window. $\bar{d}'_W$ is used as the average coverage depth parameter to evaluate the potential enrichment at the central position x_c of the sliding window by a one-sided Poisson test. The test calculates the p-value for the respective coverage depth from the cumulative distribution function for the upper tail of the Poisson distribution

$$p(x_c) = 1 - e^{-\bar{d}'_W} \cdot \sum_{i=0}^{\lfloor d(x_c) \rfloor} \frac{(\bar{d}'_W)^i}{i!}$$

Consecutive positions falling below a certain significance threshold, by default set to 0.05, are joined to form a candidate region.

Detection is fine-tunable using two auxiliary significance thresholds. A relaxed *probing* significance threshold is accepted to automatically join candidate regions separated only by short drops in depth of coverage into a contiguous region. Furthermore, candidate regions may be pre-filtered using an *acceptance* threshold, discarding all regions that do not include at least one position that satisfies this more stringent significance criterion.

2.7.4 Recognition of Read Mapping Artifacts

Final significance scores for enrichment are calculated based on the number of read mappings providing evidence for a respective site (section 2.7.5). For multiple reasons, significance of enrichment is on its own of limited value for singling out relevant protein binding sites (section 2.7.6). Peak significance is therefore primarily used as a means of imposing a ranking on the detected loci.

Strong oversampling of a region of the reference sequence, for example due to collapsing of repetitive sequence onto few representations in a reference (section 1.5.3), may result in high levels of significance for such loci even for minimal strengths of enrichment. Thus oversampling entails high-ranking loci that are of limited relevance to the analysis when compared to significant loci sampled at regular depth.

SHORE*peak* offers various filtering heuristics that allow for reliable removal of typical oversampling artifacts. Oversampled regions are marked by elevated depth of coverage in both experiment and control sample. By specification of a maximum coverage depth tolerated in the control for valid candidate regions, many oversampled positions are ruled out. SHORE filters candidate regions based on their average depth of control sample coverage, discarding regions whose depth lies above a certain threshold. The rejection threshold is calculated adaptively in terms of a user specified multiple of standard deviations distance from the median chromosomal depth of coverage. When analyzing multiple replicate experiments, such suspiciously high depth of coverage in any of the controls specified triggers removal of the proposed candidate region by our algorithm. By default, regions with average control sample read depth more than six standard deviations above median depth are ignored.

Fold change, calculated as the ratio of read counts or average depths of experiment and control sample coverage at a candidate locus, in some cases presents a more sensitive criterion for recognition of loci that satisfy a strict significance cutoff due to oversampling, but are of limited relevance to further analysis. Oversampling is then marked by large amounts of supporting evidence in the form of reads mapped to the region, but comparably weakly pronounced fold change. SHORE calculates normalized fold change values for a region as

$$f = \frac{\bar{d}_e}{k_{e,c} \cdot \bar{d}_c} \tag{2.1}$$

where $\bar{d}_e$ and $\bar{d}_c$ represent the mean depth of coverage for experiment and control, respectively, and $k_{e,c}$ a scaling factor calculated for experiment and control over the respective chromosome to accommodate differences in overall sequencing depth (section 2.7.5). Regions with a normalized fold change $f < 2$ are not considered by default. For experiments comprising multiple replicates, the user-specified fold change criterion must be satisfied by at least one replicate.

For true sequence enrichment, mapped reads represent precipitated fragments of DNA typically larger than the sequenced read length. By contrast, oversampling artifacts arise due to over-representation solely of the part of the sequence utilized for read mapping. Given this *mapping length* is shorter than the actual *insert length*, a characteristic difference in the strand specific *read depth* curves ensues for enrichment compared to oversampling. In the case of enrichment, read depth curves for the forward and the reverse strand are shifted, with the shift offset representing the missing $3'$ parts of the inserts that are not taken into account for the read depth calculation (figure 2.6). Contrarily, for oversampled regions not comprising enriched sequence, no such shift in strand specific read depth may be observed. The *peak shift* of single-end read depth therefore may be exploited as an acceptance criterion for enriched regions. SHORE*peak* estimates a peak shift value for a region based on the two areas under the read depth curves for each strand. For both these areas, the center of gravity (COG) is calculated. The estimated peak shift then equals difference between the x-coordinate of the COG for the reverse strand and the x-coordinate of the forward strand COG. In multi-replicate experiments, the user-specified peak shift threshold must be satisfied by at least one replicate.

COG-based shift estimation may be considered robust for defined loci such as transcription factor binding sites. For larger stretches of enriched sequence representing clustered binding sites or histone methylation, the COG is easily skewed by local biases or random fluctuations in sequencing depth, and therefore does not provide reliable estimation of peak shift.

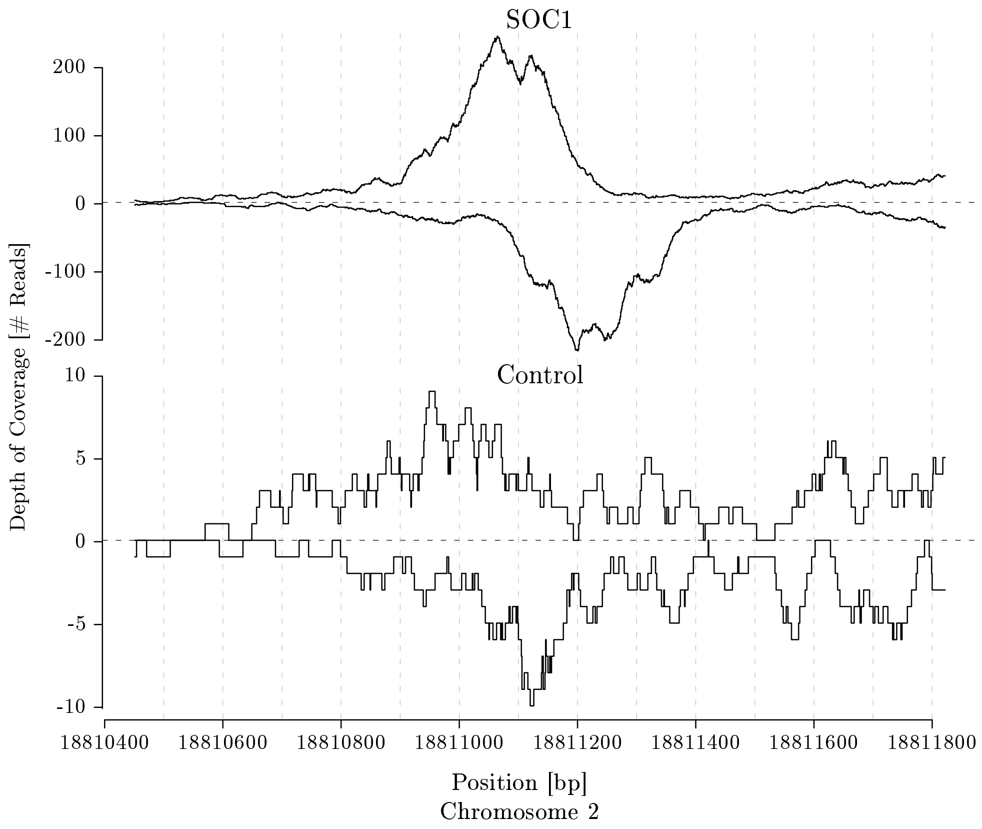


Figure 2.6: ChIP-seq Depth of Coverage in the Vicinity of a Transcription Factor Binding Site

2.7.5 Ranking and Assessment of Peak Significance

In a final step, the peak calling algorithm assesses detected candidate regions to establish a ranking of putative peak sites and determine their level of significance.

For any candidate region, given the read count for the experiment is a random variable X and the read count for the control is a random variable Y, then their joint read count is the convolution $Z = X * Y$.

Assuming the joint read count in a region takes an observed value $Z = z$, then the experiment read count is distributed according to a probability mass function $P(X = x|Z = z)$. Using Bayes' theorem, it is

$$\begin{aligned}
P(X = x|Z = z) &= \frac{P(Z = z|X = x) \cdot P(X = x)}{P(Z = z)} \\
&= \frac{P(Y = (z - x)) \cdot P(X = x)}{P(Z = z)} \qquad (2.2) \\
&= \frac{P(Y = (z - x)) \cdot P(X = x)}{\sum_{k=0}^{z} \left(P(Y = (z - k)) \cdot P(X = k)\right)} \qquad (2.3)
\end{aligned}$$

Were experiment and control read counts following Poisson distributions, $X \sim \text{Pois}(\lambda_1)$ and $Y \sim \text{Pois}(\lambda_2)$, then their convolution is Poisson distributed as well, $Z \sim \text{Pois}(\lambda_3)$, where

$\lambda_3 = \lambda_1 + \lambda_2$.

When applying the Poisson assumption to equation 2.2, it follows that

$$\begin{aligned}
P(X = x|Z = z) &= \frac{\frac{\lambda_2^{z-x} \cdot e^{-\lambda_2}}{(z-x)!} \cdot \frac{\lambda_1^x \cdot e^{-\lambda_1}}{x!}}{\frac{\lambda_3^z \cdot e^{-\lambda_3}}{z!}} \\
&= \frac{z!}{x! \cdot (z-x)!} \cdot \frac{\lambda_1^x \cdot \lambda_2^{z-x} \cdot e^{-(\lambda_1+\lambda_2)}}{\lambda_3^z \cdot e^{-\lambda_3}} \\
&= \binom{z}{x} \cdot \frac{\lambda_1^x \cdot \lambda_2^{z-x}}{(\lambda_1 + \lambda_2)^z} \\
&= \binom{z}{x} \cdot \left(\frac{\lambda_1}{\lambda_1 + \lambda_2}\right)^x \cdot \left(1 - \frac{\lambda_1}{\lambda_1 + \lambda_2}\right)^{z-x} \qquad (2.4)
\end{aligned}$$

Therefore, given Poisson distributed read counts and an observed joint read count for a region $Z = z$, the read count of the experiment follows a binomial distribution, $X \sim \mathrm{B}(z, p)$, with the probability parameter p as implied by equation 2.4

$$p = \frac{\lambda_1}{\lambda_1 + \lambda_2} \qquad (2.5)$$

Our algorithm relies on the Poisson assumption to assess the significance level of candidate peaks using a one-sided binomial test, i. e. by evaluating the upper tail of the cumulative distribution function (CDF) of the binomial distribution for the experiment read count x and joint read count z

$$P(X \geq \lfloor x \rfloor) = \sum_{k=\lfloor x \rfloor}^{z} \binom{z}{k} \cdot p^k \cdot (1-p)^{z-k}$$

The *floor* function $\lfloor\ \rfloor$ is applied to x to accommodate weighted read counts as e. g. used for reads mapping to multiple positions of the reference genome in a conservative way.

Normalization of experiment and control is therefore defined by calculation of an appropriate value for the probability parameter p of the binomial distribution. To obtain an estimate $\hat{p}$ of the probability parameter we define a sequencing depth scaling factor as $\delta = \lambda_1/\lambda_2$. By equation 2.5, the relation between the depth scaling factor and the probability parameter is

$$p = \frac{\delta}{\delta + 1} \qquad \delta = \frac{p}{1-p} \qquad (2.6)$$

Chromosomal, mitochondrial and plastid sequences may be present at considerably different number of copies in the cell, implying potentially different depth of coverage for these sequences. The probability parameter is therefore estimated independently for each chromosome or organelle sequence. The sequence is subdivided into fixed-size non-overlapping bins. The normalization algorithm then chooses $\hat{\delta}$ such that the median of the read counts associated with the bins for the experiment equals the median of the control sample bins multiplied by $\hat{\delta}$. This estimate is then used to replace δ in equation 2.6 to obtain a probability parameter for the binomial tests on the respective chromosome or organelle sequence.

While detection of candidate regions is applied to the joint data of all experiments, final assessment of peak significance is carried out separately for each replicate data set. This “joint detection, separate evaluation" approach ensures that replicate results are immediately comparable without requirement for peak overlap calculation involving arbitrary thresholds. At the same time, significance levels obtained still represent independent entities.

To account for the potentially large number of binomial tests performed, we finally process the p-values calculated to obtain false discovery rates (FDR) using Benjamini-Hochberg estimation [143].

Finally, to allow fine-grained custom filtering of peak lists, a variety of further peak properties are reported by our program, such as peak shift, peak height and fold enrichment ratio. Fold change values between sample and control are reported as a score F on the interval $[-1, 1]$ using the transformation $F = \frac{4}{\pi} \cdot \arctan(f) - 1$ on equation 2.1.

2.7.6 Experimental Relevance of Peak Significance

Besides true DNA binding affinity of the protein of interest and described artifact patterns, sources of bias may contribute to sequence enrichment in some cases indistinguishable in ChIP-Seq results.

For example, between-sample variation in the strength of sequence specific sampling bias might potentially be introduced due to slight differences in sample preparation parameters (section 1.4). Modeling varying sequencing depth as a scaling factor that globally applies to entire chromosomes then has the consequence that the definition of *enrichment* of a sequence in a two-sample comparison may include positions with above-average sampling bias, if the specific bias is pronounced more strongly in the experiment than in the control; as well as positions with below-average bias where the bias in the experiment is weaker when compared to control. Such differences in sampling bias strength can likely not be estimated globally, as the effects of multiple sources of bias might be superimposed. Adequate sampling bias estimation and correction therefore can be expected to require complex modeling based on specific sequence features and likely further investigation into sequencing and sample preparation processes. Strand specific effects may however be ruled out by assessment of properties like peak shape, peak shift or forward-reverse fold change. Furthermore, such technical sources of enrichment can sometimes be excluded by downstream binding motif analysis.

Moreover, correctly identified sites with above-average protein-DNA affinity might be biologically irrelevant. With multiple co-factors contributing to transcription initiation, evolutionary forces restricting the genome-wide development of individual binding sites are likely weak. Relevant sites should therefore be defined by presence of further promoter elements. Furthermore, both transcription factor binding as well as the ChIP procedure itself are governed by complex physico-chemical dynamics. DNA fragments with above-average affinity to the protein of interest might become detectable in ChIP-Seq, where the induced binding is however not sufficiently stable to result in transcription-regulatory relevance. Significance of such sites is however solely determined by sufficient depth of sequencing.

Further biological factors such as chromatin state and accessibility may contribute to significant enrichment not directly related to protein binding events. For these reasons, statistical significance of enrichment can on its own not provide a criterion for definition of biologically relevant binding sites.

Furthermore, the interpretation of the ranking imposed by binding site significance involves intricate considerations. Signal resulting from several different influences becomes convoluted through the ChIP procedure. Quality of the employed antibody, overall quality of the precipitation procedure, prevalence of the DNA-protein interaction across tissues as well as the actual affinity of the protein of interest to its target motifs all contribute to the overall signal observed. The first two effects globally affect the ratio of enriched sites to the magnitude of background coverage, and therefore impair the ability to observe global discrepancies in binding affinity between samples. By contrast, tissue composition and binding affinity are both locus specific effects that are indistinguishable during analysis, i. e. the procedure does not allow distinguishing between the strength of the Protein-DNA interaction and its predominance across the cells and tissue types from which the sample was retrieved. The large amount of tissue required for immunoprecipitation protocols likely contributes to a lack of controllability of the analyzed tissue states and composition, which may constitute a factor impeding experimental reproducibility.

2.8 Visualization of Sequencing Read and Alignment Data

Exploration and iterative analysis of high-throughput sequencing data require computational tools able to rapidly retrieve and visualize certain subsets or features of a data set. After obtaining raw or filtered read data, assessment of base call quality, nucleotide composition and read length distribution is desirable to verify overall data quality prior to further analysis (section 1.5.1). At later stages of data analysis, visual inspection of read alignment data can be a valuable tool for verification of loci of identified mutation candidates (section 1.3) or understanding the structure of complex genomic regions of clustered polymorphisms or rearrangements. For other applications such as mRNA-, small RNA- or ChIP-Seq, examination of local depth of coverage curves is an adequate approach to value structure or expression of individual transcripts or rule out a potential mapping artifact. Finally, different approaches to data visualization are required for gaining cues towards potential megabase scale or genome wide characteristics of the sequencing depth distribution in applications targeting e. g. epigenomic modifications.

While web browser or graphical user interface driven applications allow fully interactive navigation of data sets, they may require considerable effort in data preparation or are not easily operated remotely over a network.

The SHORE pipeline implements a variety of means to ad-hoc sequencing data visualization. The SHORE *tagstats* utility offers a summarized view of base quality distributions, per-base nucleotide composition as well as read length distribution. We further provide an application SHORE *mapdisplay* capable of rapidly generating comprehensible views of read alignment data, either as a text-based representation viewable in a terminal window, or in the form of static scalable vector graphics (SVG) images readily displayed in modern web browsers. In addition to that, versatile utilities SHORE *coverage* and SHORE *count* provide options for visualizing local depth of coverage as well as larger-scale genomic distribution of read mappings, respectively.

2.8.1 Gathering Run Quality and Sequence Composition Statistics

The SHORE *tagstats* utility summarizes content of unique read sequences in a data set, and additionally gathers base quality and per base nucleotide statistics. Statistics are collected in various tables and may additionally be visualized in the form of an all-in-one figure generated as a static Portable Document Format (PDF) image (figure 2.7).

Read quality distribution is represented as a per-base box plot, with indicators for median values, second and third quartile as well as the range included by a single standard deviation distance from the mean. Median and quartile values are corrected for quantization bias (section 2.8.4).

Total number of reads extending beyond a certain length is indicated by the *support* curve. The figure further includes the frequency of either nucleotide at each cycle of the sequencing run. Nucleotide frequencies are displayed multiplied by four to allow lineup with the support curve. In presence of sequencing errors and varying read length, uniqueness of read sequences cannot serve as a means of removal of duplicate sequences arising due to PCR amplification. Moreover, the

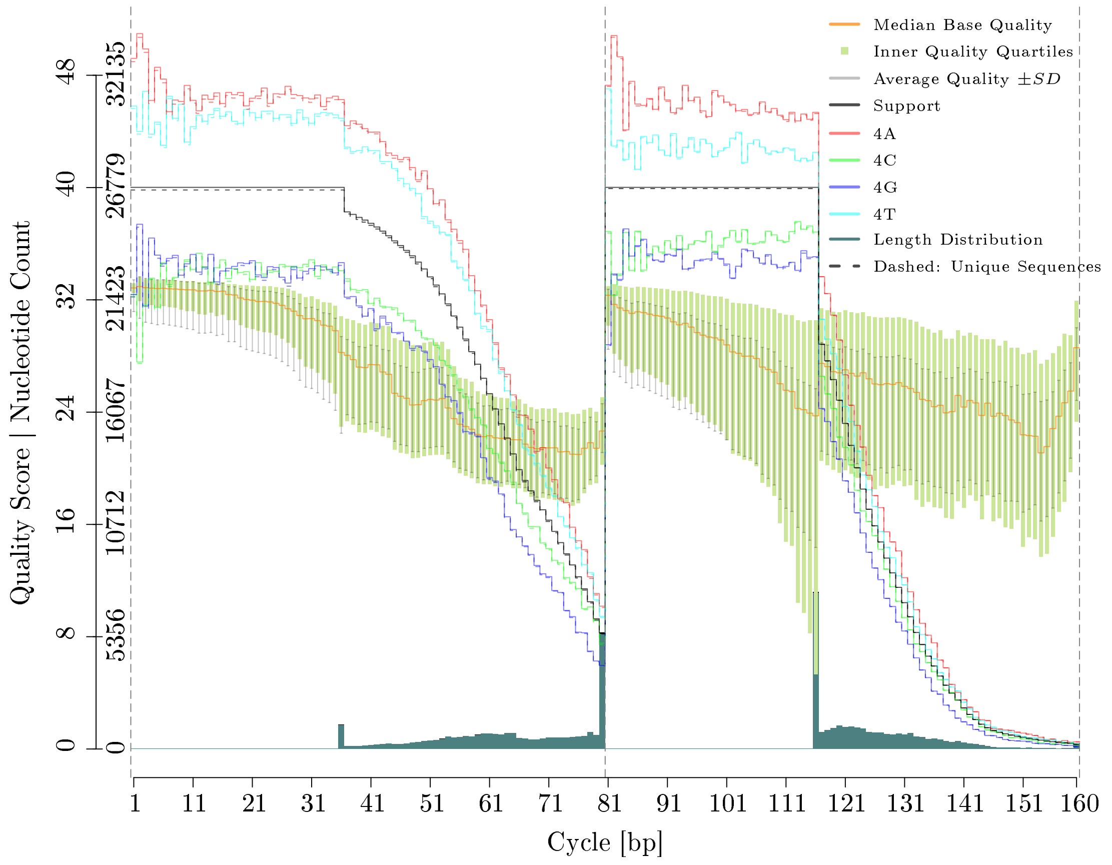
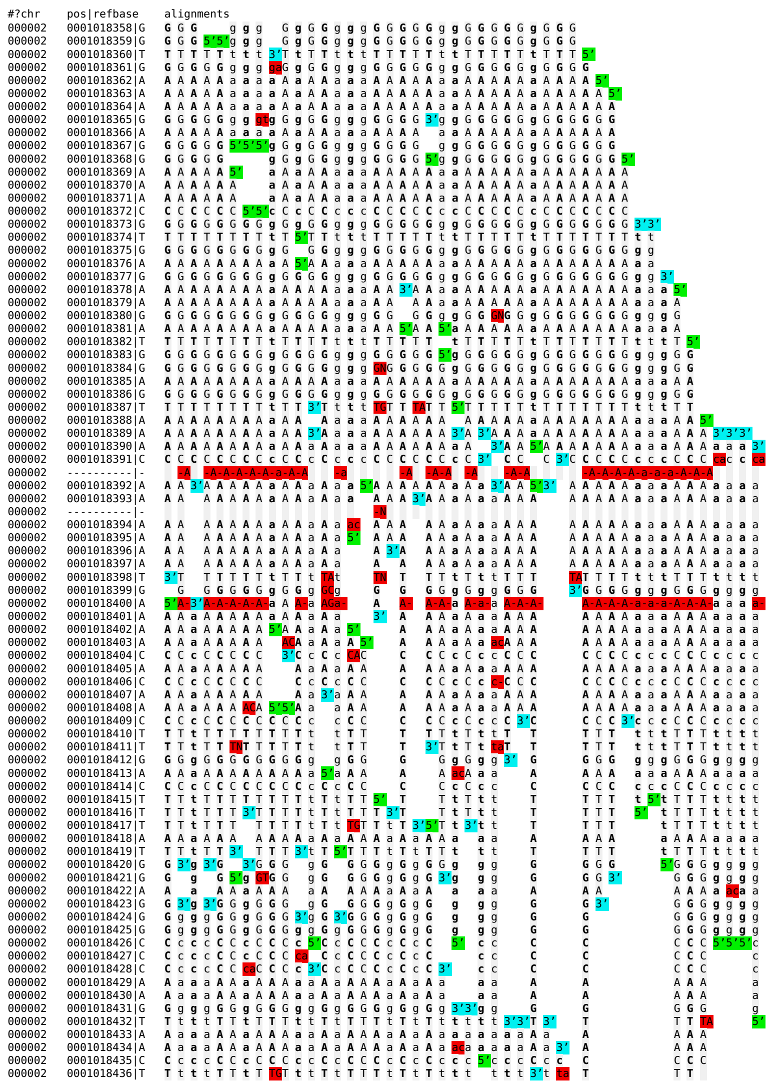


Figure 2.7: Read and Quality Statistics for a Paired-End Sequencing Run Following Quality-Based Read Trimming

likelihood of reads originating from duplicates having the same nucleotide sequence decreases with increasing read length (sections 1.4, 2.7.2). Frequency and nucleotide composition of unique read sequences can however still provide valid clues on ChIP-Seq library quality and complexity and reveal important sample properties in small RNA sequencing applications. Thus, support and nucleotide composition after pruning of redundant read sequences is indicated by a corresponding dashed line for each property.

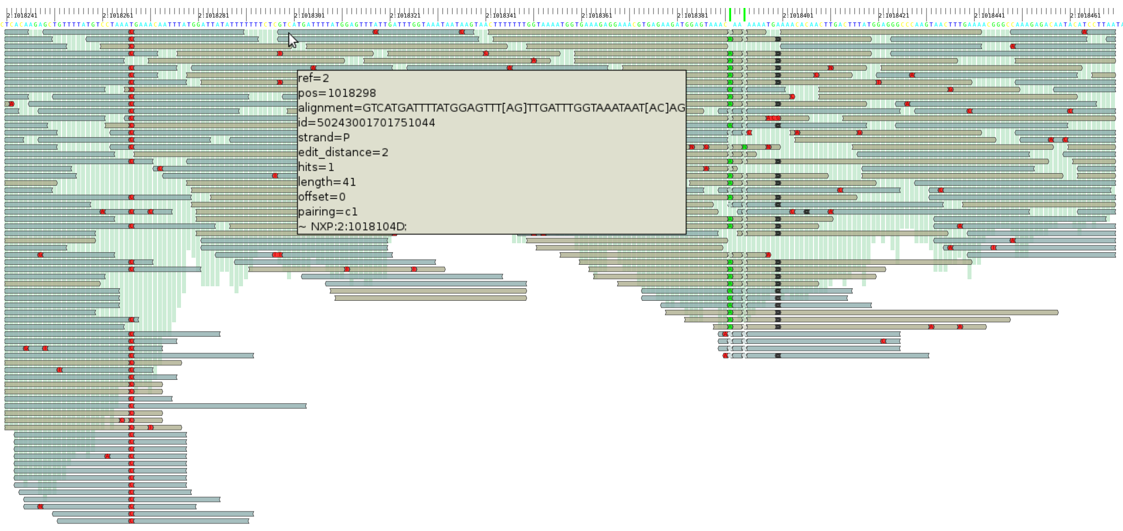
Finally, distribution of read lengths is difficult to pick up visually from support distributions alone. For improved appraisal, the distributions are therefore additionally indicated by short vertical bars at the bottom of the figure.

While the presented superimposition of quality and nucleotide composition curves may in some cases serve to obfuscate the interpretability of either individual property, it renders multivariate entities such as correlations between read quality issues and skews in nucleotide composition or the effectiveness of read trimming algorithms visually immediately appraisable.



```
#?chr    pos|refbase    alignments
000002   0001018358|G   G G G     g g g   G g G G g g g G G G G G g g G G G G G g G G G
000002   0001018359|G   G G G 5'5'g g g   G g G G g g g G G G G G g g G G G G G g G G G
000002   0001018360|T   T T T T T t t t 3'T t T T t t t T T T T T t t T T T T T t T T T 5'
000002   0001018361|G   G G G G G g g g gaG g G G g g g G G G G G g g G G G G G g G G G G
000002   0001018362|A   A A A A A a a a a A a A A a a a A A A A A a a A A A A A a A A A A 5'
000002   0001018363|A   A A A A A a a a a A a A A a a a A A A A A a a A A A A A a A A A A A 5'
000002   0001018364|A   A A A A A a a a a A a A A a a a A A A A A a a A A A A A a A A A A A A
000002   0001018365|G   G G G G G g g gtg G g G G g g g G G G G 3'g g G G G G G g G G G G G G
000002   0001018366|A   A A A A A a a a a A a A A a a a A A A A   a a A A A A A a A A A A A A
000002   0001018367|G   G G G G G 5'5'5'g G g G G g g g G G G G   g g G G G G G g G G G G G G
000002   0001018368|G   G G G G G       g G g G G g g g G G G G 5'g g G G G G G g G G G G G G 5'
000002   0001018369|A   A A A A A 5'    a A a A A a a a A A A A A a a A A A A A a A A A A A A A
000002   0001018370|A   A A A A A A     a A a A A a a a A A A A A a a A A A A A a A A A A A A A
000002   0001018371|A   A A A A A A     a A a A A a a a A A A A A a a A A A A A a A A A A A A A
000002   0001018372|C   C C C C C C 5'5'c C c C C c c c C C C C C c c C C C C C c C C C C C C C
000002   0001018373|G   G G G G G G G G g G g G G g g g G G G G G g g G G G G G g G G G G G G G 3'3'
000002   0001018374|T   T T T T T T T T t T 5'T T t t t T T T T T t t T T T T T t T T T T T T T t t
000002   0001018375|G   G G G G G G G G g G   G G g g g G G G G G g g G G G G G g G G G G G G G g g
000002   0001018376|A   A A A A A A A A a A 5'A A a a a A A A A A a a A A A A A a A A A A A A A a a
000002   0001018377|G   G G G G G G G G g G G G G g g g G G G G G g g G G G G G g G G G G G G G g g 3'
000002   0001018378|A   A A A A A A A A a A A A A a a a A A 3'A A a a A A A A A a A A A A A A A a a a 5'
000002   0001018379|A   A A A A A A A A a A A A A a a a A A   A A a a A A A A A a A A A A A A A a a a A
000002   0001018380|G   G G G G G G G G g G G G G g g g G G   G G g g G G GNG G g G G G G G G G g g g G
000002   0001018381|A   A A A A A A A A a A A A A a a a A A 5'A A 5'a A A A A A a A A A A A A A a a a A
000002   0001018382|T   T T T T T T T T t T T T T t t t T T T T T   t T T T T T t T T T T T T T t t t T 5'
000002   0001018383|G   G G G G G G G G g G G G G g g g G G G G G 5'g G G G G G g G G G G G G G g g g G G
000002   0001018384|G   G G G G G G G G g G G G G g g g GNG G G G G g G G G G G g G G G G G G G g g g G G
000002   0001018385|A   A A A A A A A A a A A A A a a a A A A A A A a A A A A A a A A A A A A A a a a A A
000002   0001018386|G   G G G G G G G G g G G G G g g g G G G G G G g G G G G G g G G G G G G G g g g G G
000002   0001018387|T   T T T T T T T T t T T 3'T t t t TGT T TAT T 5'T T T T T t T T T T T T T t t t T T
000002   0001018388|A   A A A A A A A A a A A   A a a a A A A A A A   A A A A A a A A A A A A A a a a A A 5'
000002   0001018389|A   A A A A A A A A a A A 3'A a a a A A A A A A 3'A 3'A A A a A A A A A A A a a a A A A 3'3'3'
000002   0001018390|A   A A A A A A A A a A A a A a a a A A A A A A a A   3'A A 5'A A A A A A A a a a A A A a a a 3'
000002   0001018391|C   C C C C C C C C c C C c C c c c C C C C C C c C 3'  C C   C 3'C C C C C c c c C C C cac c ca
000002   ----------|-     -A  -A-A-A-A-A-a-A-A    -a        -A  -A-A  -A    -A-A        -A-A-A-A-a-a-a-A-A-A
000002   0001018392|A   A A 3'A A A A A a A A a A a a 5'A A A A A A a A a 3'A A 5'3'  A A A A A a a a A A A a a a a
000002   0001018393|A   A A   A A A A A a A A a A a a   A A A 3'A A a A a a A A A     A A A A A a a a A A A a a a a
000002   ----------|-                                   -N
000002   0001018394|A   A A   A A A A A a A A a A a ac  A A A   A A a A a a A A A     A A A A A a a a A A A a a a a
000002   0001018395|A   A A   A A A A A a A A a A a 5'  A A A   A A a A a a A A A     A A A A A a a a A A A a a a a
000002   0001018396|A   A A   A A A A A a A A a A a     A 3'A   A A a A a a A A A     A A A A A a a a A A A a a a a
000002   0001018397|A   A A   A A A A A a A A a A a     A   A   A A a A a a A A A     A A A A A a a a A A A a a a a
000002   0001018398|T   3'T   T T T T T t T T t TAt     TN  T   T T t T t t T T T     TAT T T T t t t T T T t t t t
000002   0001018399|G     G   G G G G G g G G g GCg     G   G   G G g G g g G G G     3'G G G G g g g G G G g g g g
000002   0001018400|A   5'A-3'A-A-A-A-A-a A A-a AGa-    A   A-  A-A-a A-a-a A-A-A-      A-A-A-A-a-a-a-A-A-A-a a a a-
000002   0001018401|A   A A a A A A A A a A A a A a     3'  A   A A a A a a A A A       A A A A a a a A A A a a a a
000002   0001018402|A   A A a A A A A A 5'A A a A a 5'      A   A A a A a a A A A       A A A A a a a A A A a a a a
000002   0001018403|A   A A a A A A A A   ACA a A a A 5'    A   A A a A a acA A A       A A A A a a a A A A a a a a
000002   0001018404|C   C C c C C C C C   3'C c C c CAC     C   C C c C c c C C C       C C C C c c c C C C c c c c
000002   0001018405|A   A A a A A A A A     A a A a A A     A   A A a A a a A A A       A A A A a a a A A A a a a a
000002   0001018406|C   C C c C C C C C     C c C c C C     C   C C c C c c-C C C       C C C C c c c C C C c c c c
000002   0001018407|A   A A a A A A A A     A a 3'a A A     A   A A a A a a A A A       A A A A a a a A A A a a a a
000002   0001018408|A   A A a A A A ACA 5'5'A a   a A A     A   A A a A a a A A A       A A A A a a a A A A a a a a
000002   0001018409|C   C C c C C C C C C C C c   c C C     C   C C c C c c C 3'C       C C C 3'c c c C C C c c c c
000002   0001018410|T   T T t T T T T T T T T t   t T T     T   T T t T t t T   T       T T T   t t t T T T t t t t
000002   0001018411|T   T T t T T TNT T T T T t   t T T     T   3'T t T t taT   T       T T T   t t t T T T t t t t
000002   0001018412|G   G G g G G G G G G G G g   g G G     G     G g G g g 3'  G       G G G   g g g G G G g g g g
000002   0001018413|A   A A a A A A A A A A A a 5'a A A     A     A acA a a     A       A A A   a a a A A A a a a a
000002   0001018414|C   C C c C C C C C C C C c C c C C     C     C c C c c     C       C C C   c c c C C C c c c c
000002   0001018415|T   T T t T T T T T T T T t T t T T 5'  T     T t T t t     T       T T T   t 5't T T T t t t t
000002   0001018416|T   T T t T T T 3'T T T T t T t T T T 3'T     T t T t t     T       T T T   5'  t T T T t t t t
000002   0001018417|T   T T t T T T   T T T T t T t TGT T t T 3'5'T t 3't t     T       T T T       t T T T t t t t
000002   0001018418|A   A A a A A A   A A A A a A a A A A a A a A A a   a a     A       A A A       a A A A a a a a
000002   0001018419|T   T T t T T 3'  T T T 3't T 5'T T T t T t T T t   t t     T       T T T       t T T T t t t t
000002   0001018420|G   G 3'g 3'G   3'G G G   g G   G G G g G g G G g   g g     G       G G G       5'G G G g g g g
000002   0001018421|G   G   g   G 5'g GTG G   g G   G G G g G g G 3'g   g g     G       G G 3'        G G G g g g g
000002   0001018422|A   A   a   A A a A A A   a A   A A A a A a A   a   a a     A       A A           A A A a aca a
000002   0001018423|G   G 3'g 3'G G g G G G   g G   G G G g G g G   g   g g     G       G 3'          G G G g g g g
000002   0001018424|G   G g g g G G g G G G 3'g G 3'G G G g G g G   g   g g     G       G             G G G g g g g
000002   0001018425|G   G g g g G G g G G G g g G g G G G g G g G   g   g g     G       G             G G G g g g g
000002   0001018426|C   C c c c C C c C C C c 5'C c C C C c C c C   5'  c c     C       C             C C C 5'5'5'c
000002   0001018427|C   C c c c C C c C C C ca  C c C C C c C c C       c c     C       C             C C C       c
000002   0001018428|C   C c c c C C caC C C c 3'C c C C C c C c C 3'    c c     C       C             C C C       c
000002   0001018429|A   A a a a A A a A A A a a A a A A A a A a A a     a a     A       A             A A A       a
000002   0001018430|A   A a a a A A a A A A a a A a A A A a A a A a     a a     A       A             A A A       a
000002   0001018431|G   G g g g G G g G G G g g G g G G G g G g G g 3'3'g g     G       G             G G G       g
000002   0001018432|T   T t t t T T t T T T t t T t T T T t T t T t t t t t 3'3'T 3'    T             T T TA      5'
000002   0001018433|A   A a a a A A a A A A a a A a A A A a A a A a a a a a a a A a     A             A A A
000002   0001018434|A   A a a a A A a A A A a a A a A A A a A a A a aca a a a a A a 3'  A             A A A
000002   0001018435|C   C c c c C C c C C C c c C c C C C c C c C c c c 5'c c c C c c   C             C C C
000002   0001018436|T   T t t t T T t T TGT t t T t T T T t T t T t t t   t t t 3't ta  T             T T
```

Figure 2.8: Vertical Scrolling Text Mode Alignment View of SHORE *mapdisplay*



```
ref=2
pos=1018298
alignment=GTCATGATTTTATGGAGTTT[AG]TTGATTTGGTAAATAAT[AC]AG
id=50243001701751044
strand=P
edit_distance=2
hits=1
length=41
offset=0
pairing=c1
~ NXP:2:1018104D:
```

Figure 2.9: Vector Graphics Alignment View of SHORE *mapdisplay*

2.8.2 Visualization of Read Mapping Data

By utilizing SHORE's generic indexed query mechanisms (section 2.2.3), the SHORE *mapdisplay* program is able to rapidly retrieve all read mappings associated with a user specified genomic region. By default, the program generates a comprehensible colored ASCII/ANSI text representation of the read alignments, which is presented as a scrollable terminal view using the Unix file pager utility *less*. For technical reasons, read mappings are displayed in a vertical layout (figure 2.8).

Insertions are represented as additional positions in the alignment view. Analogous to SHORE alignment string format (section 2.2.4), mismatching positions in the alignment are displayed with the reference base on the left and the query base on the right side. When a more compact view is required, display of the reference nucleotide is omitted. To allow distinguishing between reads mapping to the forward or to the reverse strand of the reference sequence, reverse strand associated alignments are printed in lower case and forward strand sequences in upper case letters, and $3'$ and $5'$ ends are indicated by corresponding colored end markers. To allow identification of individual reads in the alignment view, inclusion of further information such as read identifiers and paired-end sequencing information may be requested on program startup.

The text mode mapping viewer provides a rapid, detailed view of read mapping data without interruption of workflows. It is however less suited to obtaining a medium- to larger-scale overview of complex genomic regions. Therefore, we additionally provide the possibility of generating an alternative alignment view as an SVG image (figure 2.9).

SVG files generated are easily and efficiently rendered in a web browser. In contrast to the text view, read sequences are omitted in the SVG view except for mismatching alignment positions to enable improved rendering performance. Mismatching alignment positions as well as insertions and deletions are color coded and their sequence included for direct examination at close zoom levels. Directionality of read mappings is indicated both by shape and color of the corresponding horizontal bars. As layout locations for the read mappings are generally assigned by a greedy algorithm that picks the lowest possible y coordinate immediately when an alignment

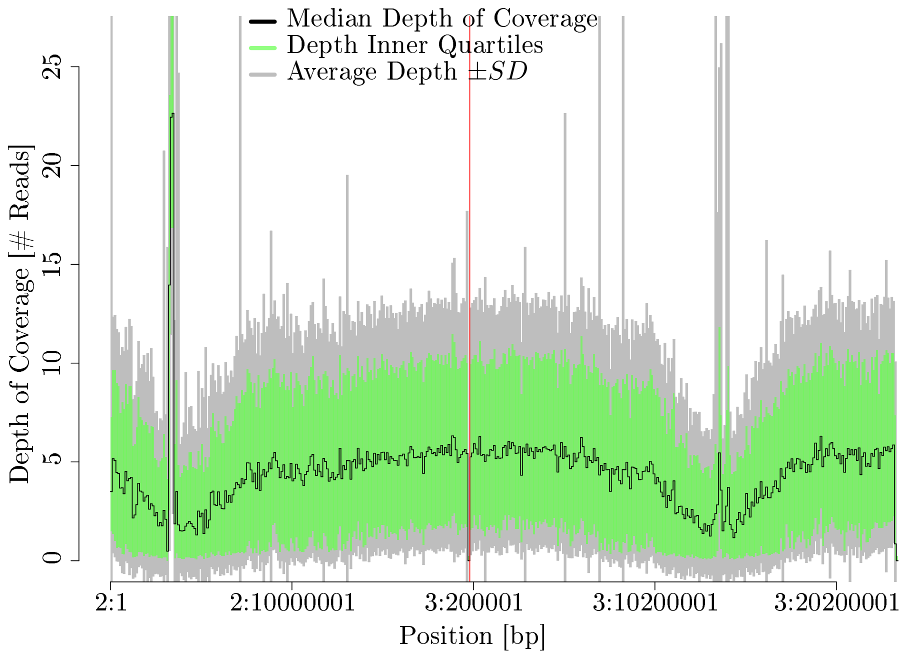


Figure 2.10: Segment-Wise Genome-Wide Box Plot

is encountered, depth of coverage can be hard to judge from the read alignments alone. Therefore, the SHORE *mapdisplay* program generates an overlay of depth of coverage, represented as vertical bars, and the horizontal bars of the read mappings. In addition, the SVG view allows retrieval of exact depth of coverage information for each position and detailed alignment information for each of the reads displayed in the form of mouse activated tool tips.

2.8.3 Visualization of Local or Genome-Scale Depth of Coverage

SHORE *coverage* is a multi-purpose tool for depth of coverage calculation and depth based segmentation (section 2.7). The complementary SHORE *count* utility gathers a wide variety of segment-related statistics. Like most SHORE modules, both programs are capable of rapidly retrieving and processing alignment information associated with defined regions of the genome.

SHORE *coverage* can be utilized to obtain visual representations of local depth of coverage (figure 2.6) or of kmer-phasing for small RNA data analysis (not shown). Per-base rendering of depth of coverage is however inappropriate for examination of larger scale distribution of sequencing depth. Visualization capability for such types of analysis is integrated with the SHORE *count* program. The software collects segment associated statistics for either user provided or fixed size *jumping window* segments. The segment data collected may be incorporated into a continuous genome- or chromosome-wide box plot (figure 2.10).

In the representation, the median line connects the median depth of coverage value for each of the segments provided, and is enclosed by boxes indicating second and third quartiles as well as the single standard deviation range surrounding the average depth of coverage of the segment. Red vertical bars indicate chromosome boundaries. As read depths mostly represent integer values, median and quartile values are adjusted to compensate quantization (section 2.8.4).

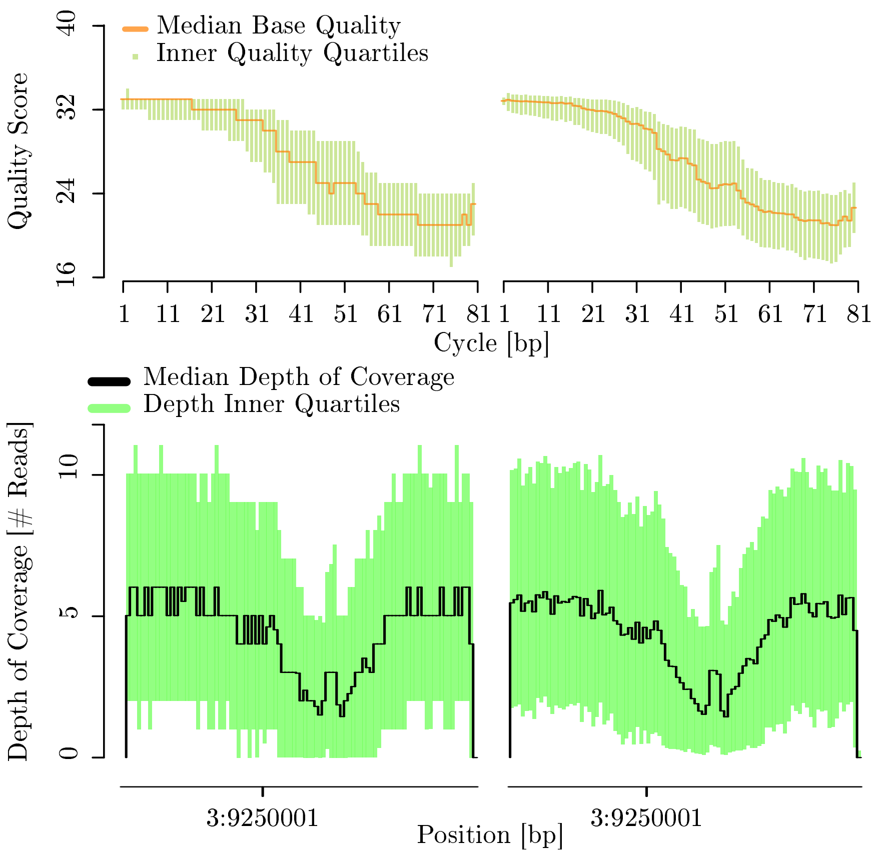


Figure 2.11: Quantile Interpolation Applied to Base Qualities and to Depth of Coverage

2.8.4 Quantile Correction

As in the case of base quality data or positional depth of coverage, quantile values of integer data distributed over a narrow range only coarsely represent the distribution of interest due to the small number of different values that such quantiles can assume (figure 2.11, left side).

Given that the integer data in fact represent real-valued entities that have at some point been rounded, then it can be assumed that the original value of an integer s follows an unknown distribution supported on the interval $[s - 0.5, s + 0.5)$. For an integer data set of size N, this implies for an arbitrary integer j-th c-quantile q_i, that the expected value for the quantile of the original, non-rounded data q_r data equals the minimal value to be considered for q_r plus the k-th order statistic for n values distributed according to the unknown distribution. Here, n is the number of values in the integer data set of equal value to q_i, and $k = \lceil (j \cdot N)/c \rceil - u$, where u denotes the number of integer data with smaller value than the quantile q_i.

Given n i. i. d. samples $X_1, \ldots, X_n$, then their k-th order statistic is a random variable $W_{(n,k)}$ with probability density function

$$f_{W_{(n,k)}}(x) = n \cdot \binom{n-1}{k-1} \cdot f_X(x) \cdot F_X(x)^{k-1} \cdot (1 - F_X(x))^{n-k} \tag{2.7}$$

Assuming X is distributed uniformly over the unit interval, then the k-th order statistic follows a beta distribution, $W_{(n,k)} \sim \mathrm{Beta}(k, n-k+1)$, with expected value and variance

$$\mathrm{E}(W_{(n,k)}) = \frac{k}{n+1} \tag{2.8}$$

$$\mathrm{Var}(W_{(n,k)}) = \frac{k}{(n+1)\cdot(n+2)} \cdot \left(1 - \frac{k}{n+1}\right) \tag{2.9}$$

The assumption of rounded data is reasonable for base quality values and arguable also for observed read count data. Therefore, it can be considered valid to estimate corrected quantile values as

$$\hat{q}_r = q_i - 0.5 + \frac{k}{n+1} \tag{2.10}$$

While the assumption of uniformity for the distribution of the original data to be considered may be questionable with special emphasis for read depth data with value zero, the adjusted quantiles still provide a considerable improvement for appraisal of the distribution compared to their integer counterparts. Equation 2.7 might be exploited where a better approximation of the actual distribution is available. For simplicity of implementation, we do not calculate the exact values of the $\hat{q}_r$, but employ a probabilistic algorithm that adds a random real value uniformly sampled from the interval $[-0.5, 0.5)$ to each integer value prior to quantile calculation (figure 2.11, right side).

3 A C++ Framework for High-Throughput DNA Sequencing

In a large, multi-purpose data analysis software solution, there are inevitably many recurring processing tasks and algorithmic challenges. To avoid large amounts of complex, redundant and error-prone code, the common subproblems must be isolated as reusable modules. With the SHORE DNA sequencing data analysis suite, we have striven to implement frequently required segments of code using consistent design patterns and interfaces. The resulting C++ programming framework libshore constitutes the subject of this chapter.

3.1 Overview

High-throughput sequencing has become adopted as a versatile tool applicable to a wide range of different scientific questions (section 1.3). While data analysis for different types of application must be approached from unique angles (section 1.5), in their elementary building blocks methods and algorithms often share considerable overlap on many levels.

Reading, parsing and formatted output of sequencing data in standard storage formats is a near universal requirement. Additionally, subsetting data sets by defined properties, achievable through different combinations of simple accept-reject filtering operations, is an often powerful tool for adjustment of an analysis' sensitivity-specificity tradeoff. More complex operations rely on the relationship between multiple data set elements, like e. g. removal of PCR duplicate sequences from read alignment data, or alter certain properties of the data set elements themselves, and are therefore sensitive to, and potentially useful not only in various combinations, but also orders of application. Finally, entire analysis processes can be modeled as modules or series of modules transforming the type of the data, for example read alignments into depth of coverage or read alignments into positional *pileups* into variant calls.

While isolation and encapsulation of subproblems facilitates correct implementation and comprehensibility of each individual step, the logic required to tie components into end-to-end processing pipelines can itself become extensive. For versatile modes of application, modularized components must therefore be embedded into a more generic framework capable of providing the glue between input, output and data filtering, manipulation and transformation.

The *libshore* C++ framework code is loosely categorized into ten different packages including application framework functionality, generic data processing infrastructure and sequencing specific processing modules and data structures. With figure 3.1 we present a package overview in a simplified UML-like representation, illustrating exemplary classes and associations.

The `base` package comprises elementary low level functionality and utilities as well as machine or system-dependent blocks of code. While its functionality is required for most other packages, it is self-contained and does not by itself depend on other parts of the library.

The class `program` from the package of the same name constitutes the base class of all SHORE command line utilities. The base class combines command line interface definition, documen-

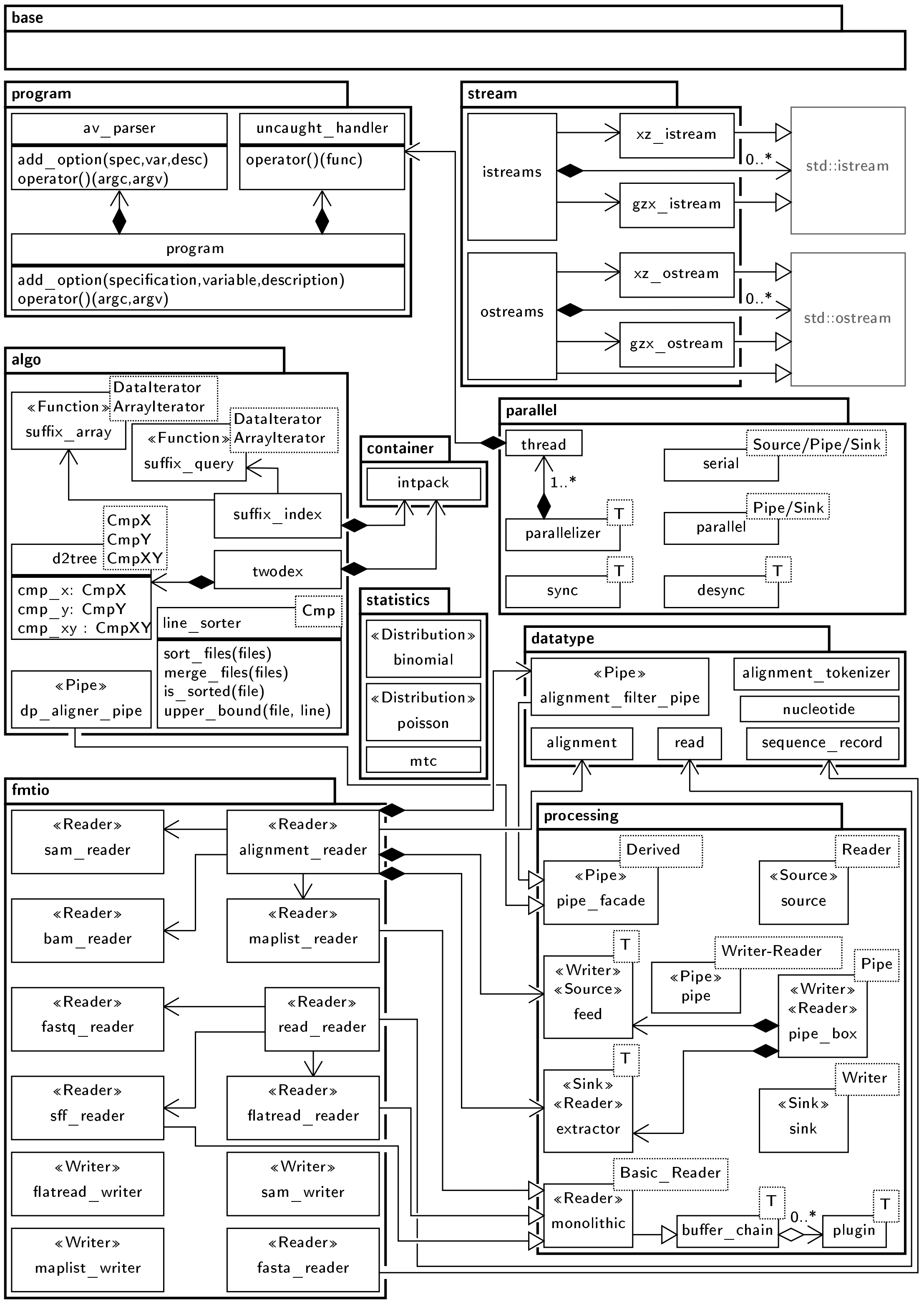


Figure 3.1: *libshore* Overview

tation and command line parsing through an auxiliary class `av_parser` and incorporates an object of class `uncaught_handler` to provide handling and reporting of fatal error conditions encountered by the application.

Classes `istreams` and `ostreams` of the `stream` package simplify handling of files or other input or output destinations. Interpretation of special location strings or prefixes facilitates specification of special files or other sources or destinations of input and output as option arguments or option default values, like e. g. standard input, output or Unix pipes. Input or output locations compressed in GZIP, BAM or XZ format are delegated to appropriate helper stream classes for automatic decoding, encoding or handling of random access requests (section 2.2.2). Both classes in addition provide input/output error checking and further stream-related functionality. Class `ostreams` furthermore implements simplified handling of temporary file destinations as well as optional temporary file compression for improved handling of large amounts of temporary data.

Package `algo` comprises of dynamic programming alignment (section 2.5.2) as well as indexing, sorting and query algorithms and associated persistent data structure implementations (section 2.2.3). The platform for range indexing and queries is provided by a generic 2-d tree algorithm template `d2tree`, which is further utilized by the class `twodex` that supplies a persistent index data structure configurable for a variety of genomic data formats. Multiple algorithm templates with the prefix `suffix_` implement generic suffix array construction and queries [144, 145]. The suffix array algorithms are employed by objects of class `suffix_index`, which provides persistent disk indexes for genomic sequences configurable and capable of answering various types of sequence match queries. To support up to 64 bit data at reasonable space efficiency, persistent `suffix_index` and `twodex` index data are encoded and decoded non-byte aligned with respective exact required bit width utilizing class `intpack` of the `container` package.

The *libshore* data processing infrastructure is defined and implemented as a set of templates forming the `processing` package (section 3.2). Package `parallel` extends on this infrastructure to build an abstracted parallel processing interface (section 3.3).

Despite growing acceptance of data exchange specifications such as SAM/BAM for short read mapping data, data format issues can present a major impediment to application of algorithms to input data from diverse sources. Therefore, the library includes support for a broad variety of sequencing read, mapping and further sequencing and DNA-related input and output file formats provided through the `fmtio` package. Format support is implemented as *Reader* and *Writer* classes with a consistent interface modeled after the requirements of the data processing infrastructure. To simplify working with various formats, for sequencing read and alignment data the library provides multi-format readers `read_reader` and `alignment_reader` which automatically incorporate the adequate *Reader* objects for each respective input file.

File format parsing and decoding requires adequate in-memory representation for the respective elements of the various types of data set, which are defined in the `datatype` package. Plain data structures `read` and `alignment` provide the standard representation of short sequencing read data, whereas large genomic sequences are stored as `sequence_record` objects with more sophisticated internal memory management. Additionally, processing modules and utilities specific to a certain type of data set element — or certain properties of data set elements — are also grouped under this package. For example, a class `alignment_filter_pipe` combines application of a variety of frequently required filtering and editing modules to read mapping data. Parsing and decoding the SHORE *alignment string* pair-wise alignment representation (section 2.2.4) into CIGAR string-equivalent tokens is provided as a class `alignment_tokenizer`. Class `nucleotide` provides decoding, encoding and efficient manipulation of IUPAC encoded DNA bases.

The additional `statistics` package supplies the basis for statistical hypothesis tests based on a variety of distributions. The complementary class `mtc` re-implements various multiple hypothesis test correction procedures in C++ following the R *multtest* package implementation [146].

Supported procedures include false discovery rate (FDR) calculation following Benjamini and Hochberg [143] as well as Benjamini and Yekutieli [147] and further family wise error rate (FWER) control via Bonferroni [148], Hochberg [149], Holm [150] as well as Sidak [151] correction algorithms.

3.2 A Modular Signal-Slot Processing Framework

3.2.1 Reader and Writer Concepts

The requirement to support large numbers of different input file formats emphasizes the advantage of consistent interfaces to provide access to the data. The *libshore* framework therefore defines the concept of a *Reader* as a class providing three methods named `has_data`, `current` and `next` which allow linear iteration over the data set's elements.

The `has_data` method returns a boolean value indicating whether further data set elements are available to the respective *Reader* object. The method `current` provides access to the element of the data set at the current position of the iteration, and `next` indicates the current element has been processed and may be discarded. Initialization of the following element for retrieval by `current` may be handled by either of the `next` or `has_data` methods.

Conversely, we define the concept of a *Writer* as a class supporting methods `append` and `flush`. The method `append` is used to pass the object a data set element for processing, whereas `flush` serves as a notification that no more data are available for processing, triggering finalizing actions such as addition of footer sequences for compressed data sets.

Intermediary processing and filtering modules can thus be realized as classes conforming to both the *Writer* and *Reader* concepts. Method `flush` in this case adopts the role of finishing data processing and propagating all remaining data to the object's *Reader* interface. For example, in a sliding window analysis output of data will be delayed until enough input data become available to cover the entire width of the window. On receiving a flush request, processing of remaining elements must however be completed without waiting for further data associated with the sliding window.

3.2.2 Definition of Processing Network Topology

With the *Reader* and *Writer* concepts, propagation of data through a network of processing modules might be accomplished through a series of nested processing loops (listing 3.1). The example illustrates propagation of data read from a single source of input through two cascaded filtering operations. Processing results are to be written to the application's standard output stream, whereas intermediate results following the first step of filtering are additionally recorded as a separate output file. The first set of loops of lines 8–27 constitute the actual propagation of input data. Read mapping data provided by the reader object are appended to the first filtering module (line 10). As a result, the filter object may or may not produce an arbitrary amount of data (e. g. due to sliding window based removal of PCR duplicated sequences), which are passed to the intermediate file writer as well as the second filter object (lines 12–24). Data produced by the second filter object must be transmitted to the final output accordingly through a further nested loop (lines 17–21). Additional sets of nested loops correspond to propagation of remaining data following appropriately stacked flush requests to each of the modules (lines 30–56).

The demonstrated mode of explicit data propagation is verbose and error prone. Our programming framework therefore implements a set of class templates providing a signal-slot pipeline architecture to allow more concise and comprehensible representations. A *signal* constitutes a function object that may be connected to an arbitrary number of function objects supporting

```
alignment_reader r("in.bam");
alignment_filter f1(filter1_config);
alignment_filter f2(filter2_config);
maplist_writer w1("filtered1.maplist");
maplist_writer w2("stdout");

// Process data.
while(r.has_data())
{
        f1.append(r.current());

        while(f1.has_data())
        {
                w1.append(f1.current());
                f2.append(f1.current());

                while(f2.has_data())
                {
                        w2.append(f2.current());
                        f2.next();
                }

                f1.next();
        }

        r.next();
}

// Finish.
f1.flush();

while(f1.has_data())
{
        w1.append(f1.current());
        f2.append(f1.current());

        while(f2.has_data())
        {
                w2.append(f2.current());
                f2.next();
        }

        f1.next();
}

w1.flush();

f2.flush();

while(f2.has_data())
{
        w2.append(f2.current());
        f2.next();
}

w2.flush();
```

Listing 3.1: Using Processing Modules with Nested Loops

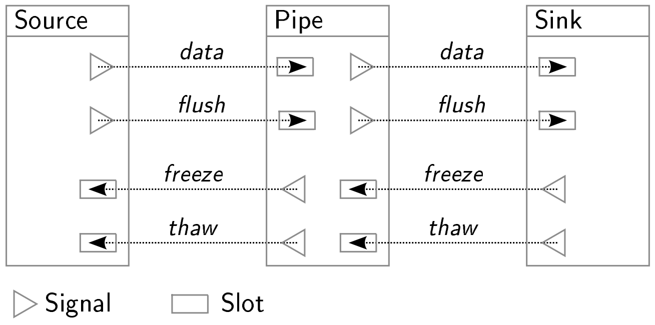


Figure 3.2: Pipeline Connections

```
1  alignment_source r("in.bam");
   alignment_filter_pipe f1(filter1_config);
   alignment_filter_pipe f2(filter2_config);
   maplist_sink w1("filtered1.maplist");
5  maplist_sink w2("stdout");

   // Connect modules.
   r | f1 | f2 | w2;

10 f1 | w1;

   // Process data and finish.
   r.dump();
```

Listing 3.2: Using Processing Modules with the *libshore* Pipeline API

the same types of function parameters (*slots*). Calling a signal function passes the respective set of function parameters to all connected slots in an unspecified order of invocation.

We thereby define the concept of a *Source* as a class providing signals named `data` as well as `flush`, and further slots labeled `freeze` and `thaw`. A *Sink* concept is conversely defined as a class providing slots named `data` and `flush`, as well as signals `freeze` and `thaw`. Furthermore, a *Pipe* amalgamates both the *Source* and the *Sink* concept. Thus defined *Pipe* processing modules immediately forward all elements of output data produced in response to flush requests or data appended the `data` slot to their respective `data` signal. A processing pipeline can thus be defined as a series of connected *Pipe* elements terminated by *Sources* and *Sinks* at either end, respectively (figure 3.2).

A *Source* or *Sink* can be realized using respective templates `source` and `sink` capable of appropriately transforming *Reader* and *Writer* objects. A further template `pipe` is provided for transformation of objects conforming to both the *Reader* and *Writer* concepts. For each file format *Reader* and *Writer* class, the library defines corresponding *Source* and *Sink* classes by means of the respective template, which are marked by suffixes `_source` and `_sink`.

Source, *Pipe* and *Sink* objects can be combined into extensive processing networks through connection of the appropriate signals and slots. To improve the clarity of definition of such networks, an overload of the pipe operator ("`|`") is provided for establishing pair-wise module connections. The code of the processing example can thereby be simplified to a large extent (listing 3.2). Lines 8 and 10 define the topology of the processing network. The `source` template further defines a method `dump`, which causes all data provided by the incorporated *Reader* to be passed to the `data` signal before triggering the `flush` signal to finalize processing (line 13).

For cases where application to entire data sets is not the intended use, modules must provide a means of suspending the flow of processing. Classes implementing the *Reader* concept may for example offer configurable standard filtering facilities comprising of multiple cascaded processing modules. However, while the *Reader* concept implies that each data set element is retrievable individually, *Pipe* elements may produce an arbitrary amount of output in response to input data processing, which would render such modules unsuitable for this type of reuse. We therefore introduce the additional module connections `freeze` and `thaw` serving to suspend and resume processing pipeline data flow, respectively (figure 3.2). These connections run anti-parallel to the `data` and `flush` signals. As signal invocation corresponds to a series of nested function calls transmitted along the connected modules, `source` and `pipe` templates are able to directly detect suspend and resume requests sent by downstream elements. A provided class template `extractor` utilizes the implemented suspend facilities to act as a bridge between the *Sink* and *Reader* concepts. Thus, `extractor` objects may be inserted at the end of a line of processing modules to retrieve generated output data individually. A complementary template `feed` provides bridging between the *Writer* and *Source* concepts.

3.2.3 Module Implementation

Encapsulation of subproblems as modules implementing both the *Writer* and *Reader* concepts comes with the disadvantage that all data generated in response to a single invocation of one of the *Writer* methods must be kept in an internal buffer for subsequent retrieval via the *Reader* interface. However, in the context of a processing pipeline each element of output data can in principle directly be transmitted to downstream modules for further processing, often avoiding the need to implement internal buffering. Direct propagation can be achieved by direct implementation of the `Pipe` concept, thus exposing direct access to the module's data and flush signals. Implementation logic of unbuffered, interruptible processing is however more complex than the buffered equivalent. We therefore provide a base class template `pipe_facade` enforcing a restricted programming model to support correct implementation of such modules.

The facade template requires distribution of processing implementation across four function calls `prepare`, `append`, `next` and `flush`. A function `emit` is further made accessible to subclasses to propagate generated output. The template enforces a single-emit rule by which invocation of the `emit` base class function is permissible exactly once during each call to either of the four implementation functions. The purpose of each of the four functions is illustrated by example (listing 3.3).

Raw sequencing read input data will typically be ordered by an identifier specific to the respective insert, whereas order of the individual reads retrieved from the insert is arbitrary. For some purposes further ordering of sequencing reads of the same insert by their paired-end sequencing index may however be convenient. Implementation of an appropriate reordering module demonstrates the purpose of each of the four facade functions.

The reordering module collects all reads belonging to the same insert in an internal buffer, and subsequently orders them by value of their paired-end index. The `prepare` method serves as an initial notification about next element that will be passed to the object's `append` method. In the case of the example module, identifiers are compared to determine whether further reads are available for the insert currently being processed. If the next read does not belong to the current insert, reordering of the reads by paired-end index is triggered. Reads are then transmitted for downstream processing by invocation of `emit` on the first of the collected reads. In concert with

```
class order_paired_reads_pipe
:public pipe_facade<order_paired_reads_pipe,
                    // Input datatype
                    read,
                    // Output datatype
                    read>
{
 private:

        // Grants pipe_facade access to private member functions.
        friend class pipeline_core_access;

        // Stores all reads with the same identifier.
        std::vector<read> m_buffer;

        // Orders all reads collected in buffer by their paired-end index.
        void order_reads_by_index();

        // Tests if a read has no mates.
        static bool is_single_read(const read &);

        void next()
        {
                if(!m_buffer.empty())
                {
                        m_buffer.pop_front();

                        if(!m_buffer.empty())
                                emit(m_buffer.front());
                }
        }

        void prepare(const read & r)
        {
                if(!m_buffer.empty() && (r.id != m_buffer.front().id))
                {
                        order_reads_by_index();
                        emit(m_buffer.front());
                }
        }

        void append(const read & r)
        {
                if(is_single_read(r))
                        emit(r);
                else
                        m_buffer.push_back(r);
        }

        void flush()
        {
                if(!m_buffer.empty())
                {
                        order_reads_by_index();
                        emit(m_buffer.front());
                }
        }
};
```

Listing 3.3: Usage Example for the `pipe_facade` Template

the method `next` described below, the internal sequencing read buffer thereby gets successively freed for processing of the following insert.

Processing of new input data must be handled by the method `append`. In the context of the reordering example, single-end and orphan reads — i. e. all reads that for any reason are the only ones for their respective insert — can be immediately transmitted downstream, whereas paired reads are appended to the internal sequencing read buffer for subsequent sorting.

The method `next` is automatically invoked through the facade class on completing propagation of an element of output data. Subclasses may implement the method with the purpose of discarding data no longer required. The `emit` method may be re-invoked inside the method body. In the example module, this property is exploited to iteratively emit all reads of the current insert and free the buffer for processing of the following insert.

By means of the division of implementation among the four processing functions in concert with the single-emit rule, the facade template is able to ensure correct data propagation and handling of suspend and resume requests triggered through the `freeze` and `thaw` slots. Subclasses may however omit implementation of individual methods not required for the respective processing task.

3.2.4 In-Place Data Manipulation

In the pipeline model, processing modules are presented with individual data set elements that are invalidated as the following element becomes available. As demonstrated by the example (listing 3.3), this implies that operations that rely on a relation between multiple elements must maintain an internal buffer to accumulate the relevant data. Furthermore, to avoid unintended side effects between different branches of the downstream processing setup, modules and data sources must prohibit direct manipulation of transmitted data. Copying of data is therefore also a prerequisite to implementing manipulation of certain properties of the data set's elements. Both restrictions may therefore result in a considerable level of unnecessary data copying overhead.

To mitigate this issue with the explicit model of data propagation (section 3.2.2), *Reader* classes are initially implemented following a lower level concept *Basic_Reader*, which is defined as a class with a single method `next`. The method can be used to attempt reading a single data set element into an externally provided buffer, and returns a boolean flag to indicate whether an element could be successfully retrieved. Data read into the provided buffer can then be freely manipulated prior to further processing. The *Basic_Reader* variant of a class is identified by the prefix `basic_`. The corresponding *Reader* classes compliant with the processing infrastructure are generated automatically via a class template `monolithic` (figure 3.1).

The processing framework provides a more generic mechanism to avoid data copying overhead. By implementation as plug-in class, operations with identical input and output data type can be realized using in-place manipulation of an arbitrary number of data set elements determined by the class.

For plug-in module support, *Reader*, *Source* or *Pipe* classes inherit a template class `buffer_chain`. The template maintains an extensible list of re-assignable buffers for the respective type of data set element associated with the data source. Prior to downstream propagation of data, objects of classes inheriting a class `plugin` of appropriate type may be granted access to the list of maintained buffers for manipulation, removal or requesting further buffering of data set elements.

As a simple example, we present the implementation of a plug-in for sort order verification (listing 3.4). Prior to propagation of the first of the buffered elements, plug-in modules are presented with the internal list of buffers, as well as a list of unused buffers to which discarded elements may be added. The module's `apply` method may freely manipulate, discard,

```
 1  class sequence_sorting_check
    :public plugin<read>
    {
     private:
 5
            // Generates an exception to indicate data are not sorted
            // according to expectation.
            void sorting_error();

10   public:

            virtual int apply(std::deque<read *> & buffers,
                              std::vector<read *> & unused,
                              const bool flush)
15          {
                    if(buffers.size() < 2)
                            return flush ? (PLUGIN_SUCCESS) : (PLUGIN_STARVED);

                    if(buffers[1]->sequence < buffers[0]->sequence)
20                          sorting_error();

                    return PLUGIN_SUCCESS;
            }
    };
```

Listing 3.4: Plugin Implementation Example

re-use buffers of the unused buffers list or allocate further buffers for its data manipulation task. Given enough data set elements have been buffered for the module to carry out its operation, a status value `PLUGIN_SUCCESS` is returned, and status `PLUGIN_STARVED` otherwise to cause the `buffer_chain` object to delay data propagation and collect further data. In case of the sort order example, the method indicates at least two elements are required to complete the operation (line 16). Given the prerequisite is fulfilled, the module proceeds to verify the ordering of the first two elements found in the buffer (line 19).

Support for plug-in modules can be readily combined with the `pipe_facade` approach (listing 3.5). The source code example presents a simple conversion module for sequencing read data types. To allow a more direct representation of the file structure, sequences read from 454 Standard Flowgram Format (SFF) files are initially stored in a data structure distinct from the *libshore* regular sequencing read representation. To be used with generic read processing modules, a conversion of read data is thus required. The `append` method of the conversion module requests an unused buffer and uses it to perform the data type conversion (lines 21, 23). Given plug-in modules succeed, a data set element is transmitted downstream (lines 25, 26). The module's `next` method invalidates the first of the buffered elements and transmits further data when appropriate. When combined with the `buffer_chain` class, the method `flush` of the `pipe_facade` template must be implemented to finalize processing of elements that were possibly delayed by request of plug-in modules (lines 29–33).

```
class sff_flatten_pipe
:public shore::pipe_facade<sff_flatten_pipe, sff_read, read>,
 public buffer_chain<read>
{
 private:

        // Grants pipe_facade access to private member functions.
        friend class shore::pipeline_core_access;

        // Performs the conversion from sff_read to read type.
        static void init_read_from_sff(read & r, const sff_read & sff);

        void next()
        {
                if(buffer_chain_pop())
                        emit(buffer_chain_front());
        }

        void append(const shore::sff_read & r)
        {
                shore::read & current = buffer_chain_push();

                init_read_from_sff(current, r);

                if(buffer_chain_prepare())
                        emit(buffer_chain_front());
        }

        void flush()
        {
                if(buffer_chain_flush(false))
                        emit(buffer_chain_front());
        }
};
```

Listing 3.5: Usage Example for the `buffer_chain` Template

3.3 A Simplified Parallelization Interface

While run time of simple high-throughput sequencing data processing tasks is often primarily determined by the available capacities for input and output, computationally more expensive operations such as mapping of reads to a reference genome sequence may require heavy parallelization to complete in an acceptable time span. Correctness of in general error prone implementation of parallel algorithms is difficult to assess. We build on the *libshore* pipeline infrastructure (section 3.2) to realize a parallelization framework that offers a simplified parallel programming model sufficient for typical processing of sequencing data. The framework provides an interface that abstracts from parallelization-related implementation details and is able to seamlessly support both shared and distributed memory modes of operation.

```
 1  const int NUM_THREADS = 16;
    const int BATCH_SIZE = 100;

    parallelizer<alignment> par(NUM_THREADS, BATCH_SIZE);
 5  sync<alignment> syn1;
    sync<alignment> syn2;

    serial<alignment_source> r("in.bam");
    parallel<alignment_filter_pipe> f1(filter1_config);
10  parallel<alignment_filter_pipe> f2(filter2_config);
    serial<maplist_sink> w1("filtered1.maplist");
    serial<maplist_sink> w2("stdout");

    // Connect modules.
15  r | par | f1 | f2 | syn2 | w2;

    f1 | syn1 | w1;

    // Process data and finish.
20  dump(r);
```

Listing 3.6: Parallel Processing Example

3.3.1 Parallelization Modules

Our parallelization library is based primarily on five different C++ class templates (figure 3.1). Class templates `parallelizer`, `sync` and `desync` can be instantiated to provide special parallelization modules for incorporation into a processing pipeline. Instantiated objects constitute special processing modules with identical input and output data type. The respective templates are parametrized on the type of data that is processable by the module. The template `parallelizer` is responsible for thread creation and management as well as dispatching of data to different threads or processes. Objects instantiated from the class template `sync` can be incorporated into a parallel pipeline to protect following pipeline modules from concurrent modification. The third type of parallelization module `desync` allows transition from pipeline modules protected by a `sync` object to further concurrently modifiable processing modules. For creation of parallel processing pipelines, the actual processing modules must further be incorporated by class templates `parallel` or `serial` to indicate whether the respective module may be used concurrently.

While the basic processing infrastructure only enforces compatibility of input and output data types of connected modules, the parallelization templates impose additional restrictions upon which classes of object can be connected. Outputs of `serial` objects may be connected to either other `serial` or to `parallelizer` or `desync` objects. Objects instantiated from the `parallel` template are the only objects capable of receiving input from `parallelizer` objects. The `parallel` objects can be connected downstream to either further objects instantiated from the `parallel` template, or to appropriate `sync` objects. Finally, `desync` outputs can only be connected to `parallel` objects.

Using these templates, the original data processing example (listing 3.2) can be reformulated to support parallel modes of operation (listing 3.6). As indicated by the example, the level of parallelism is requested through a `parallelizer` object (line 4). The `parallelizer` distributes batches of data collected from the connected reader object to the worker threads associated with it. The two subsequent filtering operations are carried out in concurrent mode, whereas the output operations must be protected by respective sync objects to prevent garbled output.

Our API abstracts from the actual implementation by which parallelization and synchronization of program flow is accomplished. The underlying framework currently supports multi-threaded operation as well as distributed memory configurations such as compute clusters utilizing the Message Passing Interface (MPI) standard. To avoid a hard dependency on MPI implementations, support for distributed operation is disabled by default, but may be enabled at compile time without further source code modification.

3.3.2 Parallel Pipeline Architecture

Internally, automation of parallel processing is accomplished by introduction of further signal-slot connections between the parallelization modules. The framework's mode of operation is illustrated with a block diagram depicting all possible module connections (figure 3.3(a)). The data *Source* for the processing pipeline is incorporated by the leftmost serial object. The data and flush signals of the source are connected to a multiplexer object internal to the parallelizer. The parallelizer further incorporates multiple thread objects (multiplicity is indicated by three dots in the lower compartment of the respective block diagram element). Data collected by the multiplexer are dispatched to threads by means of a condition variable and simple exchange of buffers.

Pipeline initialization is controlled by an auxiliary connection numthreads offered by all parallelization classes (not shown). Objects of class parallel on initialization create multiple separate processing modules. Each of these modules is associated with one specific thread of the parallelizer, to which it is connected to through its data and flush slots.

On the output side, each module incorporated by the parallel object is further connected to a unique track provided by the sync object. All of the tracks' data signals are connected to the same downstream serial object, whereas received flush requests must be integrated by the parent sync object to ensure that the downstream section of the pipeline gets flushed after processing has completed on all threads.

A sync incorporates a mutex variable which is used to protect downstream parts of the pipeline from concurrent use. On activation of a track object's data or flush slots, this mutex is acquired by the respective track before conveying the signal to the connected serial object. By the nested function call property of signals passed to connected pipeline elements, it is ensured that all contiguous, serial downstream operations are completed as the signal call returns. The mutex can thus be released immediately to free the protected modules for use by other threads.

The desync template is intended to enable correct resumption of parallel mode of operation downstream of a section of the processing pipeline controlled by a sync object. To accomplish that, desync objects must provide a decoupling of downstream processing modules from the upstream serial modules. For this purpose, the module receives locking information from the controlling sync object by means of four additional signal-slot connections. These connections between the sync and the desync object are established through objects lock_link provided by the serial modules composing the enclosed section of the pipeline. A signal postlock is transmitted by the sync object after the mutex variable has been acquired by a thread. In response, the desync object starts to accumulate all data that is generated by the upstream pipeline in a thread-specific buffer. The signal preunlock is received immediately before the mutex is unlocked by the upstream sync. The signal causes the thread-specific buffer to be detached from upstream pipeline input. Finally, a third signal postunlock indicates that the respective thread has just relinquished the sync's mutex variable, and causes the desync to dump all data collected in a buffer to the channel of the downstream pipeline that is associated with

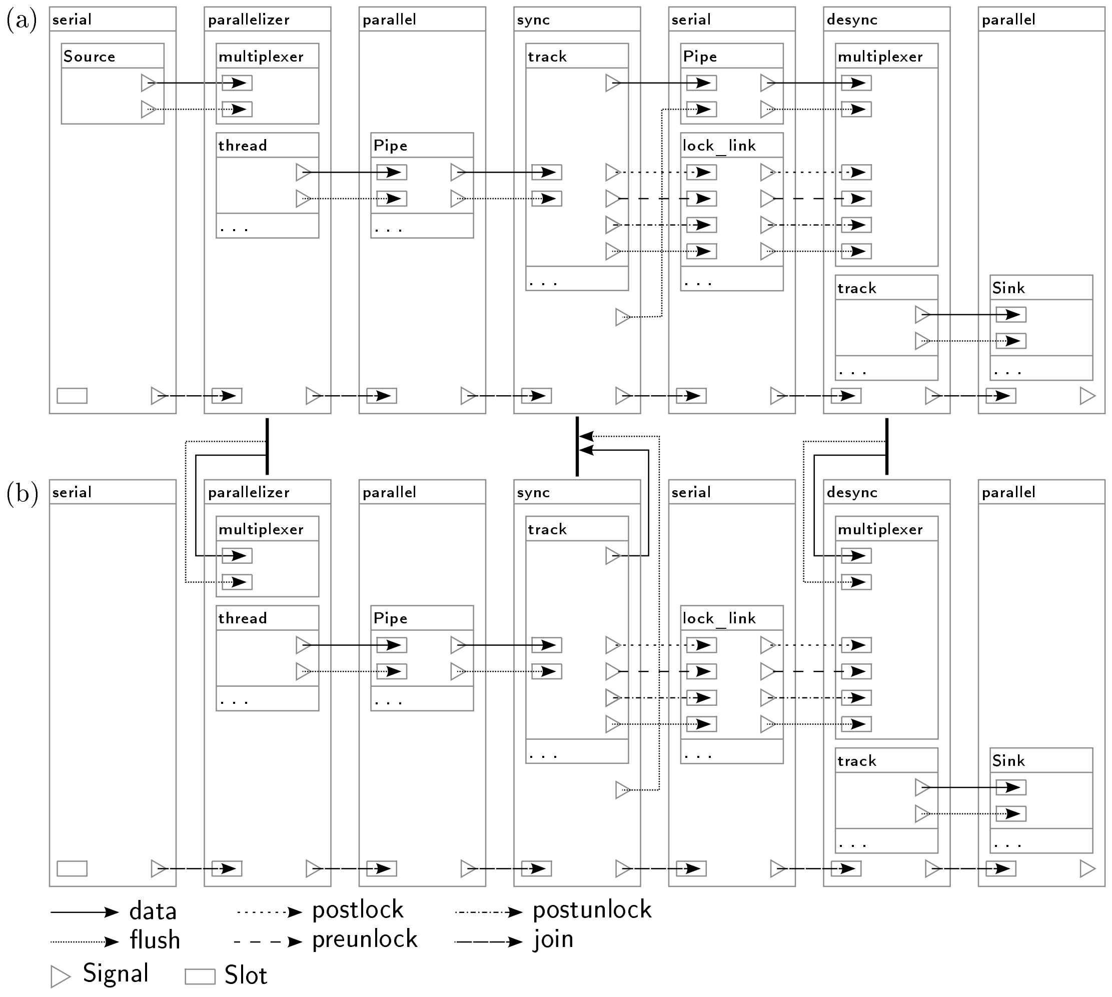


Figure 3.3: Connections between Parallel Processing Objects

the respective thread. An auxiliary `flush` signal in addition conveys the thread-specific flush state across synchronized sections of the processing pipeline.

Finally, a connection `join` is supported by all parallel processing modules to ensure correct termination of processing across all threads.

For distributed memory architectures, data are distributed from a single root process to several worker processes. Worker processes must receive the data to be processed concurrently from the root process, and must subsequently return their output data for processing by serial modules (figure 3.3(b)).

Objects of `parallelizer`, `sync` and `desync` initialized in worker processes automatically establish connections for exchange of data with their respective counterpart in the root process, where each connection is associated with an additional thread in the root process.

To prevent initialization conflicts and overhead, `serial` objects are associated with the additional function of preventing creation of their respective incorporated processing module when not in the root process.

Threads of worker processes' `parallelizer` objects operate by requesting data from the respective counterpart in the root process. Failure to retrieve data indicates that processing is complete, and triggers invocation of the thread's flush signal and subsequently thread termination. Otherwise, data received are transmitted to the subsequent `parallel` object, identical with threaded operation.

Subsequent `sync` objects submit received data and flush signals to the corresponding object of the root process. Sections formed by `serial` objects however cause an interruption in the pipelines of worker processes. Therefore, `sync` objects must exploit the `postlock` signal transmitted through `serial` modules to trigger a downstream `desync` object to request data generated by serial processing from its counterpart in the root process.

4 Closing Remarks

Routine analysis of high-throughput sequencing data not only requires development of an appropriate analysis strategy for the respective application as well as solving associated algorithmic challenges, but is further greatly facilitated by embedding the approach into an underlying framework optimized for handling large sequence data sets. With SHORE, we have implemented an end-to-end pipeline providing a powerful and modular analysis environment. Through tight integration of all required analysis modules, data flow is optimized, minimizing overhead and incompatibilities at the transition points between the various steps of an analysis. Through embedding into the pipeline environment, even supplementary utility algorithms are augmented by readily available complementary functionality such as capable read data filtering.

A variety of analysis modules dedicated to specific high-throughput sequencing applications are already made available with SHORE. With the SHORE *peak* module, this work introduced support for ChIP-Seq enrichment profiling offering highly configurable peak detection and filtering mechanisms. With an approach that renders analysis results of simultaneously processed data sets immediately comparable, obtaining conclusions from replicate experiments is in our view greatly facilitated. With further general-purpose modules SHORE *coverage* and SHORE *count* providing flexible depth of coverage-based sequence segmentation and comprehensive gathering of segment-associated statistics, SHORE should prove applicable to a wide range of expression profiling, enrichment or depletion protocols without further adaptation.

With the integrated approach, we were able to introduce a common foundation in the form of the *libshore* framework. This has allowed us to implement a common data storage solution allowing universal support of flexible and effective data compression and accelerated data set queries across all analysis modules. Thereby, we maintain the usefulness of the analysis suite in the face of vastly increased quantities of data produced by sequencing instruments. With general-purpose modules such as SHORE *oligo-match*, we further demonstrate that usability of stand-alone utilities also benefits from the common framework in terms of file format independence, familiar command line interfaces, automatic support for compressed or streaming input and output, or capable parallelization. With the *libshore* processing framework, we provide a means for designing complex analysis algorithms as a collection of manageable, self-contained modules. The ability to parallelize many such algorithms without further modification from our point of view has significant value with the still increasing rate of sequencing data production, and might serve as a basis for the development of further abstracted parallel processing approaches targeted at specific subclasses of sequencing application. For example, parallel implementations of reference-guided variant calling algorithms require appropriate data partitioning and result merging mechanisms, distributing read mapping data to different processing threads according to their location on the reference sequence. Such mechanisms are potentially intricate if multi-position traits such as allelic phasing of single nucleotide polymorphisms are to be assessed.

Despite next-generation sequencing technology now having been available for several years, for most applications surprisingly little consensus has been reached regarding optimal analysis algorithms. With the sample preparation procedure leading up to the sequencing device being application-specific, weakly standardized, involving significant amounts of manual work and thus ultimately highly variable, development of robust statistical models has proved intricate. Furthermore, strategies for result validation are usually laborious, cost-intensive or have yet to be developed. In many cases, specificity or sensitivity of predictions can be improved by intersection or union of results obtained through partially complementary algorithms, respectively. For example, with the SHORE *mapflowcell* utility, we provide a unified interface that allows to readily obtain and combine the results of a variety of available read mapping algorithms in consistent and immediately comparable output formats. Implementation of further wrapper infrastructure to similarly integrate third-party analysis algorithms for applications such as variation calling or enrichment profiling might be a future direction to obtain a valuable resource to readily determine individual algorithms' strengths and weaknesses through large-scale comparison. However, the various algorithms available for a specific high-throughput sequencing application often share similar fundamental ideas, and thus suffer from corresponding systematic issues.

The initially rapid pace at which innovations improving throughput, read lengths or error rates have been achieved now seems to be diminishing for the current generation of sequencing technologies. However, conceptually different high-throughput sequencing approaches such as nanopore sequencing are being investigated, and might in the medium term account for further drastic changes in the field. With cost reduced by a further magnitude and improved reliability and automation of sample preparation, DNA sequencing could find its way into routine medical application. However, other than cost-per-base, changes in the properties of the data should prove most relevant for scientific application and data analysis.

Sequencing read length has long been perceived as a critical factor for genome assembly or assessment of genome structure. It seems likely that a technology able to deliver read lengths of multiple tens of kilobases at competitive accuracy and price would bring about a major shift towards multiple whole-genome comparison strategies. In addition to whole-genome topology, such approaches will implicitly yield traits like copy number variation as well as single nucleotide polymorphisms and all other types of localized, small-scale variation, and possibly render the corresponding reference-based resequencing approaches irrelevant. However, to fully leverage the advantages compared to reference sequence-guided analysis, significant further challenges with respect to data analysis and representation will have to be overcome. Reasonable relations must be devised to capture homology, replacing the linear reference genome coordinate system. Providing manageable breakdown and visualization of such data will be essential, and likely require a variety of different approaches depending on the respective focus of the investigation. To assess the functional implications of primary analysis results, gene annotation or annotation transfer algorithms will have to be simplified and improved. Finally, ensuring immediate comparability of results across different studies should prove challenging.

Technological innovations that could potentially replace — or drastically improve the power of — current deep sequencing approaches such as whole-genome enrichment or expression profiling are more difficult to envision. Single-molecule sequencing methods however may in the future permit amplification- and ligation-free sequencing of samples from small amounts of starting material. From such elimination of steps of the sample preparation procedure might follow a significant reduction of sequence-specific biases. Furthermore, this simplification should present opportunities for standardization and automation of sequencing library production to further reduce technical variability. With minimization of the amount of required DNA and tissue, sample composition should correspondingly be rendered more controllable. Spike-in of standard quantities of DNA oligomers seems to have been abandoned by the scientific community, but might in future refined and standardized incarnations provide an additional resource to data normalization and estimation of measurement variance. Dropping sequencing cost and simplification of library preparation should further induce tolerance to employ the resource sequencing less sparingly. While most current studies due to economical considerations rely on no more than two replicate experiments, gathering extensive experimental evidence in the form of many replicate data sets could become standard in the future. Combined, these factors may entail improved statistical power from quantitative sequencing experiments, while at the same time facilitating statistical modeling of the sequencing and sample preparation process.

To this effect, the fundamental approaches to quantitative sequencing data analysis should require rather gradual modification compared to genome structure and variation analysis. Reference sequence-based analysis as implemented in our modules SHORE *peak* or SHORE *count* will probably remain the preferred strategy in the near future, although it may become more commonplace to generate appropriate reference genomes de-novo along with the quantitative data. A medium-term goal might be complementing the available algorithms for quantitative data with methods that allow to better leverage the statistical power acquired by increasing the number of replicate experiments. Short-term improvements could possibly be achieved by investing in more sophisticated models and algorithms to capture and estimate an increasing number of parameters of library preparation, sequencing and data analysis. However, the characteristics of current data sets may be the result of superimposition and complex interplay of a variety of effects, and are furthermore subject to constant change as protocols and sequencing chemistry evolve.

With progress with respect to sequencing read length, sequence assembly and automated genome annotation algorithms, reference-based variation detection as implemented by SHORE *consensus* or SHORE *qVar* modules could be superseded by fundamentally different strategies. However, SHORE and *libshore* may still constitute a valuable framework for implementation of such future methods. While increasing read length should automatically resolve many of the ambiguities that current analysis algorithms are confronted with, the biological questions to address by future algorithms are likely to gain complexity. They should therefore pose computationally intensive tasks that benefit from application programming frameworks oriented towards the efficient processing of large amounts of biological sequence data.

While the focus of this work was on the technical and algorithmic aspects of obtaining primary analysis results from various types of high-throughput DNA sequencing data set, a future challenge will lie with moving from elucidation of isolated properties of the cellular mechanism to building consistent theoretical frameworks through integration of the data made accessible by different sequencing applications and further complementary technologies.

Bibliography

[1] S. Ossowski. "Computational Approaches for Next Generation Sequencing Analysis and MiRNA Target Search". PhD thesis. Wilhelmstr. 32, 72074 Tübingen: Universität Tübingen, 2010.

[2] F. Sanger, S. Nicklen, and A. R. Coulson. "DNA sequencing with chain-terminating inhibitors". In: *Proc. Natl. Acad. Sci. U.S.A.* 74.12 (Dec. 1977), pp. 5463–5467. DOI: 10.1073/pnas.74.12.5463. PMID: 271968.

[3] J. C. Venter, M. D. Adams, G. G. Sutton, A. R. Kerlavage, H. O. Smith, and M. Hunkapiller. "Shotgun sequencing of the human genome". In: *Science* 280.5369 (June 1998), pp. 1540–1542. DOI: 10.1126/science.280.5369.1540. PMID: 9644018.

[4] T. S. Centre and T. W. U. G. S. Center. "Toward a complete human genome sequence". In: *Genome Res.* 8.11 (Nov. 1998), pp. 1097–1108. DOI: 10.1101/gr.8.11.1097. PMID: 9847074.

[5] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural, G. G. Sutton, H. O. Smith, M. Yandell, C. A. Evans, R. A. Holt, J. D. Gocayne, P. Amanatides, R. M. Ballew, D. H. Huson, J. R. Wortman, Q. Zhang, C. D. Kodira, X. H. Zheng, L. Chen, M. Skupski, G. Subramanian, P. D. Thomas, J. Zhang, G. L. Gabor Miklos, C. Nelson, S. Broder, A. G. Clark, J. Nadeau, V. A. McKusick, N. Zinder, et al. "The sequence of the human genome". In: *Science* 291.5507 (Feb. 2001), pp. 1304–1351. DOI: 10.1126/science.1058040. PMID: 11181995.

[6] E. S. Lander, L. M. Linton, B. Birren, C. Nusbaum, M. C. Zody, J. Baldwin, K. Devon, K. Dewar, M. Doyle, W. FitzHugh, R. Funke, D. Gage, K. Harris, A. Heaford, J. Howland, L. Kann, J. Lehoczky, R. LeVine, P. McEwan, K. McKernan, J. Meldrim, J. P. Mesirov, C. Miranda, W. Morris, J. Naylor, C. Raymond, M. Rosetti, R. Santos, A. Sheridan, C. Sougnez, et al. "Initial sequencing and analysis of the human genome". In: *Nature* 409.6822 (Feb. 2001), pp. 860–921. DOI: 10.1038/35057062. PMID: 11237011.

[7] M. Frommer, L. E. McDonald, D. S. Millar, C. M. Collis, F. Watt, G. W. Grigg, P. L. Molloy, and C. L. Paul. "A genomic sequencing protocol that yields a positive display of 5-methylcytosine residues in individual DNA strands". In: *Proc. Natl. Acad. Sci. U.S.A.* 89.5 (Mar. 1992), pp. 1827–1831. DOI: 10.1073/pnas.89.5.1827. PMID: 1542678.

[8] J. C. Venter, K. Remington, J. F. Heidelberg, A. L. Halpern, D. Rusch, J. A. Eisen, D. Wu, I. Paulsen, K. E. Nelson, W. Nelson, D. E. Fouts, S. Levy, A. H. Knap, M. W. Lomas, K. Nealson, O. White, J. Peterson, J. Hoffman, R. Parsons, H. Baden-Tillson, C. Pfannkoch, Y. H. Rogers, and H. O. Smith. "Environmental genome shotgun sequencing of the Sargasso Sea". In: *Science* 304.5667 (Apr. 2004), pp. 66–74. DOI: 10.1126/science.1093857. PMID: 15001713.

[9] S. Brenner, M. Johnson, J. Bridgham, G. Golda, D. H. Lloyd, D. Johnson, S. Luo, S. McCurdy, M. Foy, M. Ewan, R. Roth, D. George, S. Eletr, G. Albrecht, E. Vermaas, S. R. Williams, K. Moon, T. Burcham, M. Pallas, R. B. DuBridge, J. Kirchner, K. Fearon, J. Mao, and K. Corcoran. "Gene expression analysis by massively parallel signature sequencing (MPSS) on microbead arrays". In: *Nat. Biotechnol.* 18.6 (June 2000), pp. 630–634. DOI: 10.1038/76469. PMID: 10835600.

[10] M. Margulies, M. Egholm, W. E. Altman, S. Attiya, J. S. Bader, L. A. Bemben, J. Berka, M. S. Braverman, Y. J. Chen, Z. Chen, S. B. Dewell, L. Du, J. M. Fierro, X. V. Gomes, B. C. Godwin, W. He, S. Helgesen, C. H. Ho, C. H. Ho, G. P. Irzyk, S. C. Jando, M. L. Alenquer, T. P. Jarvie, K. B. Jirage, J. B. Kim, J. R. Knight, J. R. Lanza, J. H. Leamon, S. M. Lefkowitz, M. Lei, et al. "Genome sequencing in microfabricated high-density picolitre reactors". In: *Nature* 437.7057 (Sept. 2005), pp. 376–380. DOI: 10.1038/nature03959. PMID: 16056220.

[11] D. R. Bentley, S. Balasubramanian, H. P. Swerdlow, G. P. Smith, J. Milton, C. G. Brown, K. P. Hall, D. J. Evers, C. L. Barnes, H. R. Bignell, J. M. Boutell, J. Bryant, R. J. Carter, R. Keira Cheetham, A. J. Cox, D. J. Ellis, M. R. Flatbush, N. A. Gormley, S. J. Humphray, L. J. Irving, M. S. Karbelashvili, S. M. Kirk, H. Li, X. Liu, K. S. Maisinger, L. J. Murray, B. Obradovic, T. Ost, M. L. Parkinson, M. R. Pratt, et al. "Accurate whole human genome sequencing using reversible terminator chemistry". In: *Nature* 456.7218 (Nov. 2008), pp. 53–59. DOI: 10.1038/nature07517. PMID: 18987734.

[12] K. J. McKernan, H. E. Peckham, G. L. Costa, S. F. McLaughlin, Y. Fu, E. F. Tsung, C. R. Clouser, C. Duncan, J. K. Ichikawa, C. C. Lee, Z. Zhang, S. S. Ranade, E. T. Dimalanta, F. C. Hyland, T. D. Sokolsky, L. Zhang, A. Sheridan, H. Fu, C. L. Hendrickson, B. Li, L. Kotler, J. R. Stuart, J. A. Malek, J. M. Manning, A. A. Antipova, D. S. Perez, M. P. Moore, K. C. Hayashibara, M. R. Lyons, R. E. Beaudoin, et al. "Sequence and structural variation in a human genome uncovered by short-read, massively parallel ligation sequencing using two-base encoding". In: *Genome Res.* 19.9 (Sept. 2009), pp. 1527–1541. DOI: 10.1101/gr.091868.109. PMID: 19546169.

[13] J. M. Rothberg, W. Hinz, T. M. Rearick, J. Schultz, W. Mileski, M. Davey, J. H. Leamon, K. Johnson, M. J. Milgrew, M. Edwards, J. Hoon, J. F. Simons, D. Marran, J. W. Myers, J. F. Davidson, A. Branting, J. R. Nobile, B. P. Puc, D. Light, T. A. Clark, M. Huber, J. T. Branciforte, I. B. Stoner, S. E. Cawley, M. Lyons, Y. Fu, N. Homer, M. Sedova, X. Miao, B. Reed, et al. "An integrated semiconductor device enabling non-optical genome sequencing". In: *Nature* 475.7356 (July 2011), pp. 348–352. DOI: 10.1038/nature10242. PMID: 21776081.

[14] J. Eid, A. Fehr, J. Gray, K. Luong, J. Lyle, G. Otto, P. Peluso, D. Rank, P. Baybayan, B. Bettman, A. Bibillo, K. Bjornson, B. Chaudhuri, F. Christians, R. Cicero, S. Clark, R. Dalal, A. Dewinter, J. Dixon, M. Foquet, A. Gaertner, P. Hardenbol, C. Heiner, K. Hester, D. Holden, G. Kearns, X. Kong, R. Kuse, Y. Lacroix, S. Lin, et al. "Real-time DNA sequencing from single polymerase molecules". In: *Science* 323.5910 (Jan. 2009), pp. 133–138. DOI: 10.1126/science.1162986. PMID: 19023044.

[15] M. J. Levene, J. Korlach, S. W. Turner, M. Foquet, H. G. Craighead, and W. W. Webb. "Zero-mode waveguides for single-molecule analysis at high concentrations". In: *Science* 299.5607 (Jan. 2003), pp. 682–686. DOI: 10.1126/science.1079700. PMID: 12560545.

[16] D. Loakes. "Survey and summary: The applications of universal DNA base analogues". In: *Nucleic Acids Res.* 29.12 (June 2001), pp. 2437–2447. DOI: 10.1093/nar/29.12.2437. PMID: 11410649.

[17] X. Zhang, J. Yazaki, A. Sundaresan, S. Cokus, S. W. Chan, H. Chen, I. R. Henderson, P. Shinn, M. Pellegrini, S. E. Jacobsen, and J. R. Ecker. “Genome-wide high-resolution mapping and functional analysis of DNA methylation in arabidopsis”. In: *Cell* 126.6 (Sept. 2006), pp. 1189–1201. DOI: 10.1016/j.cell.2006.08.003. PMID: 16949657.

[18] S. Ossowski, K. Schneeberger, R. M. Clark, C. Lanz, N. Warthmann, and D. Weigel. “Sequencing of natural strains of Arabidopsis thaliana with short reads”. In: *Genome Res.* 18.12 (Dec. 2008), pp. 2024–2033. DOI: 10.1101/gr.080200.108. PMID: 18818371.

[19] J. Cao, K. Schneeberger, S. Ossowski, T. Gunther, S. Bender, J. Fitz, D. Koenig, C. Lanz, O. Stegle, C. Lippert, X. Wang, F. Ott, J. Muller, C. Alonso-Blanco, K. Borgwardt, K. J. Schmid, and D. Weigel. “Whole-genome sequencing of multiple Arabidopsis thaliana populations”. In: *Nat. Genet.* 43.10 (Oct. 2011), pp. 956–963. DOI: 10.1038/ng.911.

[20] K. Schneeberger, S. Ossowski, C. Lanz, T. Juul, A. H. Petersen, K. L. Nielsen, J. E. J?rgensen, D. Weigel, and S. U. Andersen. “SHOREmap: simultaneous mapping and mutation identification by deep sequencing”. In: *Nat. Methods* 6.8 (Aug. 2009), pp. 550–551. DOI: 10.1038/nmeth0809-550. PMID: 19644454.

[21] K. Schneeberger. “Whole genome analysis of Arabidopsis thaliana using Next Generation Sequencing”. PhD thesis. Wilhelmstr. 32, 72074 Tübingen: Universität Tübingen, 2010. URN: urn:nbn:de:bsz:21-opus-56685.

[22] E. A. Dinsdale, R. A. Edwards, D. Hall, F. Angly, M. Breitbart, J. M. Brulc, M. Furlan, C. Desnues, M. Haynes, L. Li, L. McDaniel, M. A. Moran, K. E. Nelson, C. Nilsson, R. Olson, J. Paul, B. R. Brito, Y. Ruan, B. K. Swan, R. Stevens, D. L. Valentine, R. V. Thurber, L. Wegley, B. A. White, and F. Rohwer. “Functional metagenomic profiling of nine biomes”. In: *Nature* 452.7187 (Apr. 2008), pp. 629–632. DOI: 10.1038/nature06810.

[23] J. Frias-Lopez, Y. Shi, G. W. Tyson, M. L. Coleman, S. C. Schuster, S. W. Chisholm, and E. F. Delong. “Microbial community gene expression in ocean surface waters”. In: *Proc. Natl. Acad. Sci. U.S.A.* 105.10 (Mar. 2008), pp. 3805–3810. DOI: 10.1073/pnas.0708897105. PMID: 18316740.

[24] X. Mou, S. Sun, R. A. Edwards, R. E. Hodson, and M. A. Moran. “Bacterial carbon processing by generalist species in the coastal ocean”. In: *Nature* 451.7179 (Feb. 2008), pp. 708–711. DOI: 10.1038/nature06513. PMID: 18223640.

[25] A. Mortazavi, B. A. Williams, K. McCue, L. Schaeffer, and B. Wold. “Mapping and quantifying mammalian transcriptomes by RNA-Seq”. In: *Nat. Methods* 5.7 (July 2008), pp. 621–628. DOI: 10.1038/nmeth.1226. PMID: 18516045.

[26] B. T. Wilhelm, S. Marguerat, S. Watt, F. Schubert, V. Wood, I. Goodhead, C. J. Penkett, J. Rogers, and J. Bahler. “Dynamic repertoire of a eukaryotic transcriptome surveyed at single-nucleotide resolution”. In: *Nature* 453.7199 (June 2008), pp. 1239–1243. DOI: 10.1038/nature07002. PMID: 18488015.

[27] R. Lister, R. C. O’Malley, J. Tonti-Filippini, B. D. Gregory, C. C. Berry, A. H. Millar, and J. R. Ecker. “Highly integrated single-base resolution maps of the epigenome in Arabidopsis”. In: *Cell* 133.3 (May 2008), pp. 523–536. DOI: 10.1016/j.cell.2008.03.029. PMID: 18423832.

[28] U. Nagalakshmi, Z. Wang, K. Waern, C. Shou, D. Raha, M. Gerstein, and M. Snyder. “The transcriptional landscape of the yeast genome defined by RNA sequencing”. In: *Science* 320.5881 (June 2008), pp. 1344–1349. DOI: 10.1126/science.1158441. PMID: 18451266.

[29] N. Cloonan, A. R. Forrest, G. Kolle, B. B. Gardiner, G. J. Faulkner, M. K. Brown, D. F. Taylor, A. L. Steptoe, S. Wani, G. Bethel, A. J. Robertson, A. C. Perkins, S. J. Bruce, C. C. Lee, S. S. Ranade, H. E. Peckham, J. M. Manning, K. J. McKernan, and S. M. Grimmond. "Stem cell transcriptome profiling via massive-scale mRNA sequencing". In: *Nat. Methods* 5.7 (July 2008), pp. 613–619. DOI: 10.1038/nmeth.1223. PMID: 18516046.

[30] C. Lu, K. Kulkarni, F. F. Souret, R. MuthuValliappan, S. S. Tej, R. S. Poethig, I. R. Henderson, S. E. Jacobsen, W. Wang, P. J. Green, and B. C. Meyers. "MicroRNAs and other small RNAs enriched in the Arabidopsis RNA-dependent RNA polymerase-2 mutant". In: *Genome Res.* 16.10 (Oct. 2006), pp. 1276–1288. DOI: 10.1101/gr.5530106. PMID: 16954541.

[31] J. G. Ruby, C. Jan, C. Player, M. J. Axtell, W. Lee, C. Nusbaum, H. Ge, and D. P. Bartel. "Large-scale sequencing reveals 21U-RNAs and additional microRNAs and endogenous siRNAs in C. elegans". In: *Cell* 127.6 (Dec. 2006), pp. 1193–1207. DOI: 10.1016/j.cell.2006.10.040. PMID: 17174894.

[32] R. D. Morin, M. D. O'Connor, M. Griffith, F. Kuchenbauer, A. Delaney, A. L. Prabhu, Y. Zhao, H. McDonald, T. Zeng, M. Hirst, C. J. Eaves, and M. A. Marra. "Application of massively parallel sequencing to microRNA profiling and discovery in human embryonic stem cells". In: *Genome Res.* 18.4 (Apr. 2008), pp. 610–621. DOI: 10.1101/gr.7179508. PMID: 18285502.

[33] P. Landgraf, M. Rusu, R. Sheridan, A. Sewer, N. Iovino, A. Aravin, S. Pfeffer, A. Rice, A. O. Kamphorst, M. Landthaler, C. Lin, N. D. Socci, L. Hermida, V. Fulci, S. Chiaretti, R. Foa, J. Schliwka, U. Fuchs, A. Novosel, R. U. Muller, B. Schermer, U. Bissels, J. Inman, Q. Phan, M. Chien, D. B. Weir, R. Choksi, G. De Vita, D. Frezzetti, H. I. Trompeter, et al. "A mammalian microRNA expression atlas based on small RNA library sequencing". In: *Cell* 129.7 (June 2007), pp. 1401–1414. DOI: 10.1016/j.cell.2007.04.040. PMID: 17604727.

[34] D. S. Johnson, A. Mortazavi, R. M. Myers, and B. Wold. "Genome-wide mapping of in vivo protein-DNA interactions". In: *Science* 316.5830 (June 2007), pp. 1497–1502. DOI: 10.1126/science.1141319. PMID: 17540862.

[35] G. Robertson, M. Hirst, M. Bainbridge, M. Bilenky, Y. Zhao, T. Zeng, G. Euskirchen, B. Bernier, R. Varhol, A. Delaney, N. Thiessen, O. L. Griffith, A. He, M. Marra, M. Snyder, and S. Jones. "Genome-wide profiles of STAT1 DNA association using chromatin immunoprecipitation and massively parallel sequencing". In: *Nat. Methods* 4.8 (Aug. 2007), pp. 651–657. DOI: 10.1038/nmeth1068. PMID: 17558387.

[36] A. Barski, S. Cuddapah, K. Cui, T. Y. Roh, D. E. Schones, Z. Wang, G. Wei, I. Chepelev, and K. Zhao. "High-resolution profiling of histone methylations in the human genome". In: *Cell* 129.4 (May 2007), pp. 823–837. DOI: 10.1016/j.cell.2007.05.009. PMID: 17512414.

[37] T. A. Down, V. K. Rakyan, D. J. Turner, P. Flicek, H. Li, E. Kulesha, S. Graf, N. Johnson, J. Herrero, E. M. Tomazou, N. P. Thorne, L. Backdahl, M. Herberth, K. L. Howe, D. K. Jackson, M. M. Miretti, J. C. Marioni, E. Birney, T. J. Hubbard, R. Durbin, S. Tavare, and S. Beck. "A Bayesian deconvolution strategy for immunoprecipitation-based DNA methylome analysis". In: *Nat. Biotechnol.* 26.7 (July 2008), pp. 779–785. DOI: 10.1038/nbt1414. PMID: 18612301.

[38] D. D. Licatalosi, A. Mele, J. J. Fak, J. Ule, M. Kayikci, S. W. Chi, T. A. Clark, A. C. Schweitzer, J. E. Blume, X. Wang, J. C. Darnell, and R. B. Darnell. “HITS-CLIP yields genome-wide insights into brain alternative RNA processing”. In: *Nature* 456.7221 (Nov. 2008), pp. 464–469. DOI: 10.1038/nature07488. PMID: 18978773.

[39] D. E. Schones, K. Cui, S. Cuddapah, T. Y. Roh, A. Barski, Z. Wang, G. Wei, and K. Zhao. “Dynamic regulation of nucleosome positioning in the human genome”. In: *Cell* 132.5 (Mar. 2008), pp. 887–898. DOI: 10.1016/j.cell.2008.02.022. PMID: 18329373.

[40] S. J. Cokus, S. Feng, X. Zhang, Z. Chen, B. Merriman, C. D. Haudenschild, S. Pradhan, S. F. Nelson, M. Pellegrini, and S. E. Jacobsen. “Shotgun bisulphite sequencing of the Arabidopsis genome reveals DNA methylation patterning”. In: *Nature* 452.7184 (Mar. 2008), pp. 215–219. DOI: 10.1038/nature06745. PMID: 18278030.

[41] A. Meissner, T. S. Mikkelsen, H. Gu, M. Wernig, J. Hanna, A. Sivachenko, X. Zhang, B. E. Bernstein, C. Nusbaum, D. B. Jaffe, A. Gnirke, R. Jaenisch, and E. S. Lander. “Genome-scale DNA methylation maps of pluripotent and differentiated cells”. In: *Nature* 454.7205 (Aug. 2008), pp. 766–770. DOI: 10.1038/nature07107. PMID: 18600261.

[42] N. A. Baird, P. D. Etter, T. S. Atwood, M. C. Currey, A. L. Shiver, Z. A. Lewis, E. U. Selker, W. A. Cresko, and E. A. Johnson. “Rapid SNP discovery and genetic mapping using sequenced RAD markers”. In: *PLoS ONE* 3.10 (2008), e3376. DOI: 10.1371/journal.pone.0003376. PMID: 18852878.

[43] E. M. Willing, M. Hoffmann, J. D. Klein, D. Weigel, and C. Dreyer. “Paired-end RAD-seq for de novo assembly and marker design without available reference”. In: *Bioinformatics* 27.16 (Aug. 2011), pp. 2187–2193. DOI: 10.1093/bioinformatics/btr346. PMID: 21712251.

[44] C. Zong, S. Lu, A. R. Chapman, and X. S. Xie. “Genome-wide detection of single-nucleotide and copy-number variations of a single human cell”. In: *Science* 338.6114 (Dec. 2012), pp. 1622–1626. DOI: 10.1126/science.1229164. PMID: 23258894.

[45] G. R. Abecasis, D. Altshuler, A. Auton, L. D. Brooks, R. M. Durbin, R. A. Gibbs, M. E. Hurles, G. A. McVean, D. Altshuler, R. M. Durbin, G. R. Abecasis, D. R. Bentley, A. Chakravarti, A. G. Clark, F. S. Collins, F. M. De La Vega, P. Donnelly, M. Egholm, P. Flicek, S. B. Gabriel, R. A. Gibbs, B. M. Knoppers, E. S. Lander, H. Lehrach, E. R. Mardis, G. A. McVean, D. A. Nickerson, L. Peltonen, A. J. Schafer, S. T. Sherry, et al. “A map of human genome variation from population-scale sequencing”. In: *Nature* 467.7319 (Oct. 2010), pp. 1061–1073. DOI: 10.1038/nature09534. PMID: 20981092.

[46] G. R. Abecasis, A. Auton, L. D. Brooks, M. A. DePristo, R. M. Durbin, R. E. Handsaker, H. M. Kang, G. T. Marth, G. A. McVean, D. M. Altshuler, R. M. Durbin, G. R. Abecasis, D. R. Bentley, A. Chakravarti, A. G. Clark, P. Donnelly, E. E. Eichler, P. Flicek, S. B. Gabriel, R. A. Gibbs, E. D. Green, M. E. Hurles, B. M. Knoppers, J. O. Korbel, E. S. Lander, C. Lee, H. Lehrach, E. R. Mardis, G. T. Marth, G. A. McVean, et al. “An integrated map of genetic variation from 1,092 human genomes”. In: *Nature* 491.7422 (Nov. 2012), pp. 56–65. DOI: 10.1038/nature11632. PMID: 23128226.

[47] D. Haussler, S. J. O’Brien, O. A. Ryder, F. K. Barker, M. Clamp, A. J. Crawford, R. Hanner, O. Hanotte, W. E. Johnson, J. A. McGuire, W. Miller, R. W. Murphy, W. J. Murphy, F. H. Sheldon, B. Sinervo, B. Venkatesh, E. O. Wiley, F. W. Allendorf, G. Amato, C. S. Baker, A. Bauer, A. Beja-Pereira, E. Bermingham, G. Bernardi, C. R. Bonvicino, S. Brenner, T. Burke, J. Cracraft, M. Diekhans, S. Edwards, et al. “Genome 10K: a proposal to obtain whole-genome sequence for 10,000 vertebrate species”. In: *J. Hered.* 100.6 (2009), pp. 659–674. DOI: 10.1093/jhered/esp086. PMID: 19892720.

[48] B. A. Methe, K. E. Nelson, M. Pop, H. H. Creasy, M. G. Giglio, C. Huttenhower, D. Gevers, J. F. Petrosino, S. Abubucker, J. H. Badger, A. T. Chinwalla, A. M. Earl, M. G. FitzGerald, R. S. Fulton, K. Hallsworth-Pepin, E. A. Lobos, R. Madupu, V. Magrini, J. C. Martin, M. Mitreva, D. M. Muzny, E. J. Sodergren, J. Versalovic, A. M. Wollam, K. C. Worley, J. R. Wortman, S. K. Young, Q. Zeng, K. M. Aagaard, O. O. Abolude, et al. "A framework for human microbiome research". In: *Nature* 486.7402 (June 2012), pp. 215–221. DOI: 10.1038/nature11209. PMID: 22699610.

[49] C. Huttenhower, D. Gevers, R. Knight, S. Abubucker, J. H. Badger, A. T. Chinwalla, H. H. Creasy, A. M. Earl, M. G. FitzGerald, R. S. Fulton, M. G. Giglio, K. Hallsworth-Pepin, E. A. Lobos, R. Madupu, V. Magrini, J. C. Martin, M. Mitreva, D. M. Muzny, E. J. Sodergren, J. Versalovic, A. M. Wollam, K. C. Worley, J. R. Wortman, S. K. Young, Q. Zeng, K. M. Aagaard, O. O. Abolude, E. Allen-Vercoe, E. J. Alm, L. Alvarado, et al. "Structure, function and diversity of the healthy human microbiome". In: *Nature* 486.7402 (June 2012), pp. 207–214. DOI: 10.1038/nature11234. PMID: 22699609.

[50] P. R. Burton, D. G. Clayton, L. R. Cardon, N. Craddock, P. Deloukas, A. Duncanson, D. P. Kwiatkowski, M. I. McCarthy, W. H. Ouwehand, N. J. Samani, J. A. Todd, P. Donnelly, J. C. Barrett, P. R. Burton, D. Davison, P. Donnelly, D. Easton, D. Evans, H. T. Leung, J. L. Marchini, A. P. Morris, C. C. Spencer, M. D. Tobin, L. R. Cardon, D. G. Clayton, A. P. Attwood, J. P. Boorman, B. Cant, U. Everson, J. M. Hussey, et al. "Genome-wide association study of 14,000 cases of seven common diseases and 3,000 shared controls". In: *Nature* 447.7145 (June 2007), pp. 661–678. DOI: 10.1038/nature05911. PMID: 17554300.

[51] R. W. Carthew and E. J. Sontheimer. "Origins and Mechanisms of miRNAs and siRNAs". In: *Cell* 136.4 (Feb. 2009), pp. 642–655. DOI: 10.1016/j.cell.2009.01.035. PMID: 19239886.

[52] O. Voinnet. "Origin, biogenesis, and activity of plant microRNAs". In: *Cell* 136.4 (Feb. 2009), pp. 669–687. DOI: 10.1016/j.cell.2009.01.046. PMID: 19239888.

[53] S. E. Linsen, E. de Wit, G. Janssens, S. Heater, L. Chapman, R. K. Parkin, B. Fritz, S. K. Wyman, E. de Bruijn, E. E. Voest, S. Kuersten, M. Tewari, and E. Cuppen. "Limitations and possibilities of small RNA digital gene expression profiling". In: *Nat. Methods* 6.7 (July 2009), pp. 474–476. DOI: 10.1038/nmeth0709-474. PMID: 19564845.

[54] S. U. Meyer, M. W. Pfaffl, and S. E. Ulbrich. "Normalization strategies for microRNA profiling experiments: a 'normal' way to a hidden layer of complexity?" In: *Biotechnol. Lett.* 32.12 (Dec. 2010), pp. 1777–1788. DOI: 10.1007/s10529-010-0380-z. PMID: 20703800.

[55] J. D. Hollister, L. M. Smith, Y. L. Guo, F. Ott, D. Weigel, and B. S. Gaut. "Transposable elements and small RNAs contribute to gene expression divergence between Arabidopsis thaliana and Arabidopsis lyrata". In: *Proc. Natl. Acad. Sci. U.S.A.* 108.6 (Feb. 2011), pp. 2322–2327. DOI: 10.1073/pnas.1018222108.

[56] S. Barker, M. Weinfeld, and D. Murray. "DNA-protein crosslinks: their induction, repair, and biological consequences". In: *Mutat. Res.* 589.2 (Mar. 2005), pp. 111–135. DOI: 10.1016/j.mrrev.2004.11.003. PMID: 15795165.

[57] A. Valouev, J. Ichikawa, T. Tonthat, J. Stuart, S. Ranade, H. Peckham, K. Zeng, J. A. Malek, G. Costa, K. McKernan, A. Sidow, A. Fire, and S. M. Johnson. "A high-resolution, nucleosome position map of C. elegans reveals a lack of universal sequence-dictated positioning". In: *Genome Res.* 18.7 (July 2008), pp. 1051–1063. DOI: 10.1101/gr.076463.108. PMID: 18477713.

[58] L. Yant, J. Mathieu, T. T. Dinh, F. Ott, C. Lanz, H. Wollmann, X. Chen, and M. Schmid. "Orchestration of the floral transition and floral development in Arabidopsis by the bifunctional transcription factor APETALA2". In: *Plant Cell* 22.7 (July 2010), pp. 2156–2170. DOI: 10.1105/tpc.110.075606.

[59] M. Hafner, M. Landthaler, L. Burger, M. Khorshid, J. Hausser, P. Berninger, A. Rothballer, M. Ascano, A. C. Jungkamp, M. Munschauer, A. Ulrich, G. S. Wardle, S. Dewell, M. Zavolan, and T. Tuschl. "Transcriptome-wide identification of RNA-binding protein and microRNA target sites by PAR-CLIP". In: *Cell* 141.1 (Apr. 2010), pp. 129–141. DOI: 10.1016/j.cell.2010.03.009. PMID: 20371350.

[60] M. Muers. "Technology: Getting Moore from DNA sequencing". In: *Nat. Rev. Genet.* 12.9 (Sept. 2011), p. 586. DOI: 10.1038/nrg3059. PMID: 21808262.

[61] G. E. Moore. "Cramming More Components onto Integrated Circuits". In: *Electronics* 38.8 (Apr. 19, 1965), pp. 114–117. DOI: 10.1109/JPROC.1998.658762.

[62] R. F. Service. "Gene sequencing. The race for the $1000 genome". In: *Science* 311.5767 (Mar. 2006), pp. 1544–1546. DOI: 10.1126/science.311.5767.1544. PMID: 16543431.

[63] C. Ledergerber and C. Dessimoz. "Base-calling for next-generation sequencing platforms". In: *Brief. Bioinformatics* 12.5 (Sept. 2011), pp. 489–497. DOI: 10.1093/bib/bbq077. PMID: 21245079.

[64] D. Aird, M. G. Ross, W. S. Chen, M. Danielsson, T. Fennell, C. Russ, D. B. Jaffe, C. Nusbaum, and A. Gnirke. "Analyzing and minimizing PCR amplification bias in Illumina sequencing libraries". In: *Genome Biol.* 12.2 (2011), R18. DOI: 10.1186/gb-2011-12-2-r18. PMID: 21338519.

[65] B. Ewing, L. Hillier, M. C. Wendl, and P. Green. "Base-calling of automated sequencer traces using phred. I. Accuracy assessment". In: *Genome Res.* 8.3 (Mar. 1998), pp. 175–185. DOI: 10.1101/gr.8.3.175. PMID: 9521921.

[66] B. Ewing and P. Green. "Base-calling of automated sequencer traces using phred. II. Error probabilities". In: *Genome Res.* 8.3 (Mar. 1998), pp. 186–194. DOI: 10.1101/gr.8.3.186. PMID: 9521922.

[67] M. Kircher, U. Stenzel, and J. Kelso. "Improved base calling for the Illumina Genome Analyzer using machine learning strategies". In: *Genome Biol.* 10.8 (2009), R83. DOI: 10.1186/gb-2009-10-8-r83. PMID: 19682367.

[68] W. C. Kao, K. Stevens, and Y. S. Song. "BayesCall: A model-based base-calling algorithm for high-throughput short-read sequencing". In: *Genome Res.* 19.10 (Oct. 2009), pp. 1884–1895. DOI: 10.1101/gr.095299.109. PMID: 19661376.

[69] J. Rougemont, A. Amzallag, C. Iseli, L. Farinelli, I. Xenarios, and F. Naef. "Probabilistic base calling of Solexa sequencing data". In: *BMC Bioinformatics* 9 (2008), p. 431. DOI: 10.1186/1471-2105-9-431. PMID: 18851737.

[70] Y. Erlich, P. P. Mitra, M. delaBastide, W. R. McCombie, and G. J. Hannon. "Alta-Cyclic: a self-optimizing base caller for next-generation sequencing". In: *Nat. Methods* 5.8 (Aug. 2008), pp. 679–682. DOI: 10.1038/nmeth.1230. PMID: 18604217.

[71] A. R. Quinlan, D. A. Stewart, M. P. Stromberg, and G. T. Marth. "Pyrobayes: an improved base caller for SNP discovery in pyrosequences". In: *Nat. Methods* 5.2 (Feb. 2008), pp. 179–181. DOI: 10.1038/nmeth.1172. PMID: 18193056.

[72] S. Andrews. *FastQC: A quality control tool for high throughput sequence data.* May 2012. *WebCite*: `68W7OHlEL`. URL: `http://www.bioinformatics.babraham.ac.uk/projects/fastqc/` Feb. 27, 2013.

[73] A. Gordon. *FASTX-Toolkit: FASTQ/A short-reads pre-processing tools.* Feb. 2010. *WebCite*: `5zWQ6Eqh6`. URL: `http://hannonlab.cshl.edu/fastx_toolkit/index.html` Feb. 27, 2013.

[74] T. Lassmann, Y. Hayashizaki, and C. O. Daub. "TagDust–a program to eliminate artifacts from next generation sequencing data". In: *Bioinformatics* 25.21 (Nov. 2009), pp. 2839–2840. DOI: `10.1093/bioinformatics/btp527`. PMID: `19737799`.

[75] R. Schmieder, Y. W. Lim, F. Rohwer, and R. Edwards. "TagCleaner: Identification and removal of tag sequences from genomic and metagenomic datasets". In: *BMC Bioinformatics* 11 (2010), p. 341. DOI: `10.1186/1471-2105-11-341`. PMID: `20573248`.

[76] R. K. Patel and M. Jain. "NGS QC Toolkit: a toolkit for quality control of next generation sequencing data". In: *PLoS ONE* 7.2 (2012), e30619. DOI: `10.1371/journal.pone.0030619`. PMID: `22312429`.

[77] D. R. Zerbino and E. Birney. "Velvet: algorithms for de novo short read assembly using de Bruijn graphs". In: *Genome Res.* 18.5 (May 2008), pp. 821–829. DOI: `10.1101/gr.074492.107`. PMID: `18349386`.

[78] M. H. Schulz, D. R. Zerbino, M. Vingron, and E. Birney. "Oases: robust de novo RNA-seq assembly across the dynamic range of expression levels". In: *Bioinformatics* 28.8 (Apr. 2012), pp. 1086–1092. DOI: `10.1093/bioinformatics/bts094`. PMID: `22368243`.

[79] R. Luo, B. Liu, Y. Xie, Z. Li, W. Huang, J. Yuan, G. He, Y. Chen, Q. Pan, Y. Liu, J. Tang, G. Wu, H. Zhang, Y. Shi, Y. Liu, C. Yu, B. Wang, Y. Lu, C. Han, D. Cheung, S.-M. Yiu, S. Peng, Z. Xiaoqian, G. Liu, X. Liao, Y. Li, H. Yang, J. Wang, T.-W. Lam, and J. Wang. "SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler". In: *GigaScience* 1.1 (2012), p. 18. DOI: `10.1186/2047-217X-1-18`.

[80] J. Butler, I. MacCallum, M. Kleber, I. A. Shlyakhter, M. K. Belmonte, E. S. Lander, C. Nusbaum, and D. B. Jaffe. "ALLPATHS: de novo assembly of whole-genome shotgun microreads". In: *Genome Res.* 18.5 (May 2008), pp. 810–820. DOI: `10.1101/gr.7337908`. PMID: `18340039`.

[81] I. Maccallum, D. Przybylski, S. Gnerre, J. Burton, I. Shlyakhter, A. Gnirke, J. Malek, K. McKernan, S. Ranade, T. P. Shea, L. Williams, S. Young, C. Nusbaum, and D. B. Jaffe. "ALLPATHS 2: small genomes assembled accurately and with high continuity from short paired reads". In: *Genome Biol.* 10.10 (2009), R103. DOI: `10.1186/gb-2009-10-10-r103`. PMID: `19796385`.

[82] F. J. Ribeiro, D. Przybylski, S. Yin, T. Sharpe, S. Gnerre, A. Abouelleil, A. M. Berlin, A. Montmayeur, T. P. Shea, B. J. Walker, S. K. Young, C. Russ, C. Nusbaum, I. MacCallum, and D. B. Jaffe. "Finished bacterial genomes from shotgun sequence data". In: *Genome Res.* 22.11 (Nov. 2012), pp. 2270–2277. DOI: `10.1101/gr.141515.112`. PMID: `22829535`.

[83] S. Gnerre, I. Maccallum, D. Przybylski, F. J. Ribeiro, J. N. Burton, B. J. Walker, T. Sharpe, G. Hall, T. P. Shea, S. Sykes, A. M. Berlin, D. Aird, M. Costello, R. Daza, L. Williams, R. Nicol, A. Gnirke, C. Nusbaum, E. S. Lander, and D. B. Jaffe. "High-quality draft assemblies of mammalian genomes from massively parallel sequence data". In: *Proc. Natl. Acad. Sci. U.S.A.* 108.4 (Jan. 2011), pp. 1513–1518. DOI: `10.1073/pnas.1017351108`. PMID: `21187386`.

[84] K. Schneeberger, S. Ossowski, F. Ott, J. D. Klein, X. Wang, C. Lanz, L. M. Smith, J. Cao, J. Fitz, N. Warthmann, S. R. Henz, D. H. Huson, and D. Weigel. “Reference-guided assembly of four diverse Arabidopsis thaliana genomes”. In: *Proc. Natl. Acad. Sci. U.S.A.* 108.25 (June 2011), pp. 10249–10254. DOI: 10.1073/pnas.1107739108.

[85] H. Li and R. Durbin. “Fast and accurate short read alignment with Burrows-Wheeler transform”. In: *Bioinformatics* 25.14 (July 2009), pp. 1754–1760. DOI: 10.1093/bioinformatics/btp324. PMID: 19451168.

[86] B. Langmead. “Aligning short sequencing reads with Bowtie”. In: *Curr Protoc Bioinformatics* Chapter 11 (Dec. 2010), Unit 11.7. DOI: 10.1002/0471250953.bi1107s32. PMID: 21154709.

[87] B. Langmead and S. L. Salzberg. “Fast gapped-read alignment with Bowtie 2”. In: *Nat. Methods* 9.4 (Apr. 2012), pp. 357–359. DOI: 10.1038/nmeth.1923. PMID: 22388286.

[88] R. Li, Y. Li, K. Kristiansen, and J. Wang. “SOAP: short oligonucleotide alignment program”. In: *Bioinformatics* 24.5 (Mar. 2008), pp. 713–714. DOI: 10.1093/bioinformatics/btn025. PMID: 18227114.

[89] R. Li, C. Yu, Y. Li, T. W. Lam, S. M. Yiu, K. Kristiansen, and J. Wang. “SOAP2: an improved ultrafast tool for short read alignment”. In: *Bioinformatics* 25.15 (Aug. 2009), pp. 1966–1967. DOI: 10.1093/bioinformatics/btp336. PMID: 19497933.

[90] S. M. Rumble, P. Lacroute, A. V. Dalca, M. Fiume, A. Sidow, and M. Brudno. “SHRiMP: accurate mapping of short color-space reads”. In: *PLoS Comput. Biol.* 5.5 (May 2009), e1000386. DOI: 10.1371/journal.pcbi.1000386. PMID: 19461883.

[91] M. David, M. Dzamba, D. Lister, L. Ilie, and M. Brudno. “SHRiMP2: sensitive yet practical SHort Read Mapping”. In: *Bioinformatics* 27.7 (Apr. 2011), pp. 1011–1012. DOI: 10.1093/bioinformatics/btr046. PMID: 21278192.

[92] H. Li, J. Ruan, and R. Durbin. “Mapping short DNA sequencing reads and calling variants using mapping quality scores”. In: *Genome Res.* 18.11 (Nov. 2008), pp. 1851–1858. DOI: 10.1101/gr.078212.108. PMID: 18714091.

[93] K. Schneeberger, J. Hagmann, S. Ossowski, N. Warthmann, S. Gesing, O. Kohlbacher, and D. Weigel. “Simultaneous alignment of short reads against multiple genomes”. In: *Genome Biol.* 10.9 (2009), R98. DOI: 10.1186/gb-2009-10-9-r98. PMID: 19761611.

[94] G. Jean, A. Kahles, V. T. Sreedharan, F. De Bona, and G. Ratsch. “RNA-Seq read alignments with PALMapper”. In: *Curr Protoc Bioinformatics* Chapter 11 (Dec. 2010), Unit 11.6. DOI: 10.1002/0471250953.bi1106s32. PMID: 21154708.

[95] M. C. Schatz. “CloudBurst: highly sensitive read mapping with MapReduce”. In: *Bioinformatics* 25.11 (June 2009), pp. 1363–1369. DOI: 10.1093/bioinformatics/btp236. PMID: 19357099.

[96] P. Ferragina and G. Manzini. “Opportunistic data structures with applications”. In: *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*. FOCS ’00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 390–. ISBN: 0-7695-0850-2. DOI: 10.1109/SFCS.2000.892127.

[97] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, and M. A. DePristo. “The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data”. In: *Genome Res.* 20.9 (Sept. 2010), pp. 1297–1303. DOI: 10.1101/gr.107524.110. PMID: 20644199.

[98] M. A. DePristo, E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, C. Hartl, A. A. Philippakis, G. del Angel, M. A. Rivas, M. Hanna, A. McKenna, T. J. Fennell, A. M. Kernytsky, A. Y. Sivachenko, K. Cibulskis, S. B. Gabriel, D. Altshuler, and M. J. Daly. "A framework for variation discovery and genotyping using next-generation DNA sequencing data". In: *Nat. Genet.* 43.5 (May 2011), pp. 491–498. DOI: 10.1038/ng.806. PMID: 21478889.

[99] S. Ossowski, K. Schneeberger, J. I. Lucas-Lledo, N. Warthmann, R. M. Clark, R. G. Shaw, D. Weigel, and M. Lynch. "The rate and molecular spectrum of spontaneous mutations in Arabidopsis thaliana". In: *Science* 327.5961 (Jan. 2010), pp. 92–94. DOI: 10.1126/science.1180677. PMID: 20044577.

[100] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin. "The Sequence Alignment/Map format and SAMtools". In: *Bioinformatics* 25.16 (Aug. 2009), pp. 2078–2079. DOI: 10.1093/bioinformatics/btp352. PMID: 19505943.

[101] H. Li. "Improving SNP discovery by base alignment quality". In: *Bioinformatics* 27.8 (Apr. 2011), pp. 1157–1158. DOI: 10.1093/bioinformatics/btr076. PMID: 21320865.

[102] M. D. Robinson and A. Oshlack. "A scaling normalization method for differential expression analysis of RNA-seq data". In: *Genome Biol.* 11.3 (2010), R25. DOI: 10.1186/gb-2010-11-3-r25. PMID: 20196867.

[103] L. X. Garmire and S. Subramaniam. "Evaluation of normalization methods in mammalian microRNA-Seq data". In: *RNA* 18.6 (June 2012), pp. 1279–1288. DOI: 10.1261/rna.030916.111. PMID: 22532701.

[104] Y. Zhang, T. Liu, C. A. Meyer, J. Eeckhoute, D. S. Johnson, B. E. Bernstein, C. Nusbaum, R. M. Myers, M. Brown, W. Li, and X. S. Liu. "Model-based analysis of ChIP-Seq (MACS)". In: *Genome Biol.* 9.9 (2008), R137. DOI: 10.1186/gb-2008-9-9-r137. PMID: 18798982.

[105] A. Valouev, D. S. Johnson, A. Sundquist, C. Medina, E. Anton, S. Batzoglou, R. M. Myers, and A. Sidow. "Genome-wide analysis of transcription factor binding sites based on ChIP-Seq data". In: *Nat. Methods* 5.9 (Sept. 2008), pp. 829–834. DOI: 10.1038/nmeth.1246. PMID: 19160518.

[106] R. Jothi, S. Cuddapah, A. Barski, K. Cui, and K. Zhao. "Genome-wide identification of in vivo protein-DNA binding sites from ChIP-Seq data". In: *Nucleic Acids Res.* 36.16 (Sept. 2008), pp. 5221–5231. DOI: 10.1093/nar/gkn488. PMID: 18684996.

[107] J. Rozowsky, G. Euskirchen, R. K. Auerbach, Z. D. Zhang, T. Gibson, R. Bjornson, N. Carriero, M. Snyder, and M. B. Gerstein. "PeakSeq enables systematic scoring of ChIP-seq experiments relative to controls". In: *Nat. Biotechnol.* 27.1 (Jan. 2009), pp. 66–75. DOI: 10.1038/nbt.1518. PMID: 19122651.

[108] H. Ji, H. Jiang, W. Ma, D. S. Johnson, R. M. Myers, and W. H. Wong. "An integrated software system for analyzing ChIP-chip and ChIP-seq data". In: *Nat. Biotechnol.* 26.11 (Nov. 2008), pp. 1293–1300. DOI: 10.1038/nbt.1505. PMID: 18978777.

[109] A. P. Fejes, G. Robertson, M. Bilenky, R. Varhol, M. Bainbridge, and S. J. Jones. "FindPeaks 3.1: a tool for identifying areas of enrichment from massively parallel short-read sequencing technology". In: *Bioinformatics* 24.15 (Aug. 2008), pp. 1729–1730. DOI: 10.1093/bioinformatics/btn305. PMID: 18599518.

[110] P. J. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice. “The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants”. In: *Nucleic Acids Res.* 38.6 (Apr. 2010), pp. 1767–1771. DOI: 10.1093/nar/gkp1137. PMID: 20015970.

[111] T. S. F. S. W. Group. *The SAM Format Specification*. Sept. 2011. *WebCite*: 6FqvzGZJ8. URL: http://samtools.sourceforge.net/SAM1.pdf Feb. 27, 2013.

[112] L. Stein. *Generic Feature Format Version 3*. Feb. 2013. *WebCite*: 5R00Wxobq. URL: http://www.sequenceontology.org/gff3.shtml Mar. 4, 2013.

[113] M. G. Reese, B. Moore, C. Batchelor, F. Salas, F. Cunningham, G. T. Marth, L. Stein, P. Flicek, M. Yandell, and K. Eilbeck. “A standard variation file format for human genome sequences”. In: *Genome Biol.* 11.8 (2010), R88. DOI: 10.1186/gb-2010-11-8-r88. PMID: 20796305.

[114] P. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, S. T. Sherry, G. McVean, R. Durbin, R. Durbin, D. Altshuler, G. Abecasis, D. Bentley, A. Chakravarti, A. Clark, F. De La Vega, P. Donnelly, M. Dunn, P. Flicek, S. Gabriel, E. Green, R. Gibbs, B. Knoppers, E. Lander, H. Lehrach, E. Mardis, G. Marth, et al. “The variant call format and VCFtools”. In: *Bioinformatics* 27.15 (Aug. 2011), pp. 2156–2158. DOI: 10.1093/bioinformatics/btr330. PMID: 21653522.

[115] P. Deutsch. *DEFLATE Compressed Data Format Specification version 1.3*. RFC 1951 (Informational). Internet Engineering Task Force, May 1996. *WebCite*: 6Fr6mTscx. URL: http://www.ietf.org/rfc/rfc1951.txt.

[116] P. Deutsch. *GZIP file format specification version 4.3*. RFC 1952 (Informational). Internet Engineering Task Force, May 1996. *WebCite*: 6Fr6csgTT. URL: http://www.ietf.org/rfc/rfc1952.txt.

[117] X. Chen, M. Li, B. Ma, and J. Tromp. “DNACompress: fast and effective DNA sequence compression”. In: *Bioinformatics* 18.12 (Dec. 2002), pp. 1696–1698. DOI: 10.1093/bioinformatics/18.12.1696. PMID: 12490460.

[118] F. Hach, I. Numanagic, C. Alkan, and S. C. Sahinalp. “SCALCE: boosting sequence compression algorithms using locally consistent encoding”. In: *Bioinformatics* 28.23 (Dec. 2012), pp. 3051–3057. DOI: 10.1093/bioinformatics/bts593. PMID: 23047557.

[119] D. C. Jones, W. L. Ruzzo, X. Peng, and M. G. Katze. “Compression of next-generation sequencing reads aided by highly efficient de novo assembly”. In: *Nucleic Acids Res.* 40.22 (Dec. 2012), e171. DOI: 10.1093/nar/gks754. PMID: 22904078.

[120] M. Hsi-Yang Fritz, R. Leinonen, G. Cochrane, and E. Birney. “Efficient storage of high throughput DNA sequencing data using reference-based compression”. In: *Genome Res.* 21.5 (May 2011), pp. 734–740. DOI: 10.1101/gr.114819.110. PMID: 21245279.

[121] S. Golomb. “Run-length encodings (Corresp.)” In: *Information Theory, IEEE Transactions on* 12.3 (1966), pp. 399–401. ISSN: 0018-9448. DOI: 10.1109/TIT.1966.1053907.

[122] D. Huffman. “A Method for the Construction of Minimum-Redundancy Codes”. In: *Proceedings of the IRE* 40.9 (1952), pp. 1098–1101. ISSN: 0096-8390. DOI: 10.1109/JRPROC.1952.273898.

[123] W. J. Kent, A. S. Zweig, G. Barber, A. S. Hinrichs, and D. Karolchik. “BigWig and BigBed: enabling browsing of large distributed datasets”. In: *Bioinformatics* 26.17 (Sept. 2010), pp. 2204–2207. DOI: 10.1093/bioinformatics/btq351. PMID: 20639541.

[124] W. J. Kent, C. W. Sugnet, T. S. Furey, K. M. Roskin, T. H. Pringle, A. M. Zahler, and D. Haussler. "The human genome browser at UCSC". In: *Genome Res.* 12.6 (June 2002), pp. 996–1006. DOI: 10.1101/gr.229102. PMID: 12045153.

[125] H. Li. "Tabix: fast retrieval of sequence features from generic TAB-delimited files". In: *Bioinformatics* 27.5 (Mar. 2011), pp. 718–719. DOI: 10.1093/bioinformatics/btq671. PMID: 21208982.

[126] A. Guttman. "R-trees: a dynamic index structure for spatial searching". In: *SIGMOD Rec.* 14.2 (June 1984), pp. 47–57. ISSN: 0163-5808. DOI: 10.1145/971697.602266.

[127] L. Collin and I. Pavlov. *The .xz File Format version 1.0.4*. Aug. 2009. *WebCite*: 6FqpSZFsW. URL: http://tukaani.org/xz/xz-file-format.txt Nov. 4, 2012.

[128] J. L. Bentley. "Multidimensional binary search trees used for associative searching". In: *Commun. ACM* 18.9 (Sept. 1975), pp. 509–517. ISSN: 0001-0782. DOI: 10.1145/361002.361007.

[129] Y. Livnat, H.-W. Shen, and C. R. Johnson. "A Near Optimal Isosurface Extraction Algorithm Using the Span Space". In: *IEEE Transactions on Visualization and Computer Graphics* 2.1 (Mar. 1996), pp. 73–84. ISSN: 1077-2626. DOI: 10.1109/2945.489388.

[130] D. Lee and C. Wong. "Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees". English. In: *Acta Informatica* 9.1 (1977), pp. 23–29. ISSN: 0001-5903. DOI: 10.1007/BF00263763.

[131] S. Kurtz. *The Vmatch large scale sequence analysis software.* Dec. 2011. *WebCite*: 5EyNRAwVK. URL: http://www.vmatch.de Feb. 27, 2013.

[132] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. "Replacing suffix trees with enhanced suffix arrays". In: *Journal of Discrete Algorithms* 2.1 (2004), pp. 53 –86. DOI: 10.1016/S1570-8667(03)00065-0.

[133] S. B. Needleman and C. D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". In: *J. Mol. Biol.* 48.3 (Mar. 1970), pp. 443–453. DOI: 10.1016/0022-2836(70)90057-4. PMID: 5420325.

[134] T. F. Smith and M. S. Waterman. "Identification of common molecular subsequences". In: *J. Mol. Biol.* 147.1 (Mar. 1981), pp. 195–197. DOI: 10.1016/0022-2836(81)90087-5. PMID: 7265238.

[135] F. Van Nieuwerburgh, R. C. Thompson, J. Ledesma, D. Deforce, T. Gaasterland, P. Ordoukhanian, and S. R. Head. "Illumina mate-paired DNA sequencing-library preparation using Cre-Lox recombination". In: *Nucleic Acids Res.* 40.3 (Feb. 2012), e24. DOI: 10.1093/nar/gkr1000. PMID: 22127871.

[136] W. J. Kent. "BLAT–the BLAST-like alignment tool". In: *Genome Res.* 12.4 (Apr. 2002), pp. 656–664. DOI: 10.1101/gr.229202. PMID: 11932250.

[137] E. Moyroud, E. G. Minguet, F. Ott, L. Yant, D. Pose, M. Monniaux, S. Blanchet, O. Bastien, E. Thevenon, D. Weigel, M. Schmid, and F. Parcy. "Prediction of regulatory interactions from genome sequences using a biophysical model for the Arabidopsis LEAFY transcription factor". In: *Plant Cell* 23.4 (Apr. 2011), pp. 1293–1306. DOI: 10.1105/tpc.111.083329.

[138] R. Brandt, M. Salla-Martret, J. Bou-Torrent, T. Musielak, M. Stahl, C. Lanz, F. Ott, M. Schmid, T. Greb, M. Schwarz, S. B. Choi, M. K. Barton, B. J. Reinhart, T. Liu, M. Quint, J. C. Palauqui, J. F. Martinez-Garcia, and S. Wenkel. "Genome-wide binding-site analysis of REVOLUTA reveals a link between leaf patterning and light-mediated growth responses". In: *Plant J.* 72.1 (Oct. 2012), pp. 31–42. DOI: `10.1111/j.1365-313X.2012.05049.x`.

[139] R. G. Immink, D. Pose, S. Ferrario, F. Ott, K. Kaufmann, F. L. Valentim, S. de Folter, F. van der Wal, A. D. van Dijk, M. Schmid, and G. C. Angenent. "Characterization of SOC1's central role in flowering by the identification of its upstream and downstream regulators". In: *Plant Physiol.* 160.1 (Sept. 2012), pp. 433–449. DOI: `10.1104/pp.112.202614`.

[140] D. Pose, L. Verhage, F. Ott, L. Yant, J. Mathieu, G. C. Angenent, R. G. Immink, and M. Schmid. "Temperature-dependent regulation of flowering by antagonistic FLM variants". In: *Nature* (Sept. 2013). DOI: `10.1038/nature12633`.

[141] P. Merelo, Y. Xie, L. Brand, F. Ott, D. Weigel, J. L. Bowman, M. G. Heisler, and S. Wenkel. "Genome-Wide Identification of KANADI1 Target Genes". In: *PLoS ONE* 8.10 (2013), e77341. DOI: `10.1371/journal.pone.0077341`.

[142] E. S. Lander and M. S. Waterman. "Genomic mapping by fingerprinting random clones: a mathematical analysis". In: *Genomics* 2.3 (Apr. 1988), pp. 231–239. DOI: `10.1016/0888-7543(88)90007-9`. PMID: `3294162`.

[143] Y. Benjamini and Y. Hochberg. "Controlling the false discovery rate: a practical and powerful approach to multiple testing". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1995), pp. 289–300.

[144] U. Manber and G. Myers. "Suffix arrays: a new method for on-line string searches". In: *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*. SODA '90. San Francisco, California, USA: Society for Industrial and Applied Mathematics, 1990, pp. 319–327. ISBN: 0-89871-251-3.

[145] G. Nong, S. Zhang, and W. H. Chan. "Two Efficient Algorithms for Linear Time Suffix Array Construction". In: *IEEE Transactions on Computers* 60.10 (2011), pp. 1471–1484. ISSN: 0018-9340. DOI: `10.1109/TC.2010.188`.

[146] K. S. Pollard, H. N. Gilbert, Y. Ge, S. Taylor, and S. Dudoit. *multtest: Resampling-based multiple hypothesis testing*. R package version 2.4.0.

[147] Y. Benjamini and D. Yekutieli. "The control of the false discovery rate in multiple testing under dependency". In: *Annals of statistics* (2001), pp. 1165–1188.

[148] O. J. Dunn. "Multiple comparisons among means". In: *Journal of the American Statistical Association* 56.293 (1961), pp. 52–64.

[149] Y. Hochberg. "A sharper Bonferroni procedure for multiple tests of significance". In: *Biometrika* 75.4 (1988), pp. 800–802. DOI: `10.1093/biomet/75.4.800`.

[150] S. Holm. "A simple sequentially rejective multiple test procedure". In: *Scandinavian journal of statistics* (1979), pp. 65–70.

[151] P. H. Westfall and S. Young. *Resampling-Based Multiple Testing: Examples and Methods for P-Value Adjustment*. A Wiley-Interscience publication. Wiley, 1993. ISBN: 9 780471 557616.