Aus dem Institut für Medizinische Genetik und Humangenetik
der Medizinischen Fakultät Charité - Universitätsmedizin Berlin

DISSERTATION

Development of a bespoke algorithm to analyze ChIP-nexus genome-wide protein-DNA binding profiles

Zur Erlangung des akademischen Grades
Doctor medicinae (Dr. med.)

vorgelegt der Medizinischen Fakultät
Charité - Universitätsmedizin Berlin

von

Benjamin Sefa Menküc
aus Herdecke

Datum der Promotion: 10.03.2017

Contents

Abstract

Protein-DNA interaction plays a major role in gene regulation. As the human genome has been successfully decoded, the current focus of research is now on understanding the interaction of genes and what they are doing. Understanding these mechanisms is crucial to better understand and cure diseases where gene regulation is important.

The tools currently available to analyze living cells at a molecular level are still very limited. It is still not known how genes are regulated and which proteins can physically interact with them. ChIP-nexus is a protocol that quantitatively analyzes the interaction of proteins with deoxyribonucleic acid (DNA). Tools presented in this work aim to optimize the results obtained by this method and make it easy to use.
Results: The newly developed tools have been successfully validated against published data. Furthermore, the results could be improved in some aspects such as the number of useable reads. The useability could be improved by integrating all preprocessing tools into two programs that are still flexible enough to allow for possible future changes. Speed improvements in the order of 5 times were achieved by applying multithreading techniques and using vector instructions.

Abstrakt

Protein-DNA Interaktion spielt eine große Rolle in Genregulation. Seitdem das menschliche Genom vor einigen Jahren entschlüsselt wurde, liegt der Fokus nun darauf ein besseres Verständnis dieser Gene und deren Regulation zu erlangen. Verständnis dieser Mechanismen ist entscheidend, um Krankheiten, bei denen Genregulation eine Rolle spielt, besser zu verstehen und zu heilen.

Aktuell sind die zur Verfügung stehenden Werkzeuge, um lebende Zellen auf molekularer Ebene zu untersuchen, noch sehr begrenzt. Für viele Gene ist immer noch nicht bekannt, wie sie reguliert werden und welche Proteine mit den entsprechenden Stellen auf der DNA interagieren. ChIP-nexus ist eine Methode, welche geeignet ist neue Erkenntnisse über Protein-DNA Interaktion zu gewinnen. Die in dieser Arbeit vorgestellten Werkzeuge helfen dabei dies zu erleichtern und das Verfahren durch eine effiziente und einfache bioinformatische Analyse-Pipeline einem größerem Publikum zugänglich zu machen.
Ergebnisse: Die neu entwickelten Programme wurden erfolgreich anhand veröffentlichter Daten validiert. Die Ausbeute an nutzbaren Daten konnte durch neue Verarbeitungsmethoden gesteigert werden. Außerdem konnte die Benutzbarkeit durch Integration aller Verarbeitungsschritte deutlich erleichtert werden. Die Geschwindigkeit des Preprozessings wurde durch Anwendung von Multithreading-Techniken und Verwendung von Vektorinstruktionen durchschnittlich 5-fach gesteigert.

Objectives of this thesis

We are currently establishing the ChIP-nexus protocol at our institute for studying protein-DNA interaction in more detail than before. Since ChIP-nexus is a very new protocol, there are no software tools available that work with this protocol without altering the programming. In our group, we have already developed a software called Q for peak calling in ChIP-seq data [1].

Currently, we are enhancing this software, as part of a project called Q-nexus, to be able to also process ChIP-nexus data. The software tools for preprocessing of ChIP-nexus data are part of this project. Preprocessing of ChIP-nexus requires additional steps compared to ChIP-seq and ChIP-exo, because the random and fixed barcodes need to be processed. Until now, no all-in-one software is available that can perform the new tasks such as random and fixed barcode processing which is required by ChIP-nexus data. Therefore, the inventors of ChIP-nexus have delivered a pipeline that uses R-scripts to perform these tasks. However, scripts written in R are usually not very fast, nor very memory efficient.

Therefore the goals of this thesis are

1. To develop an efficient and accurate preprocessing algorithm for ChIP-nexus data
2. To optimize the preprocessing with regard to runtime, memory consumption and accuracy
3. To integrate the algorithm into our Q-nexus software package
4. To test the newly developed software on test and real data

1 Introduction

1.1 The human genome in contemporary medical research

Even though the human genome project was completed 13 years ago, medical genetics is still in its infancy. There are many reasons for this; for example, it is still a major challenge to go from DNA sequence to structure. Even though we have some basic understanding of the underlying mechanisms that govern protein folding [2], it is still not possible to confidently determine the protein structure of many genes with current methods such as molecular dynamics and protein crystallization. In 2014, a map of the human proteome was published [3] and the 17,294 proteins that were reported cover 84% of all known protein-coding genes.

Assuming that all proteins of the human organism will be known soon, this knowledge is still not directly useable to cure diseases. We need to understand the function of proteins and how they are regulated. This leads us to the very broad and still quite unexplored field of regulation of gene expression. Until 2005, the available data was still a limiting factor in this field, because Sanger sequencing [4], the method that was most widely used until that time, has a very limited throughput.

With the advent of next generation sequencing (NGS), there was suddenly ample data available, which enabled new insights but also required new methods to process this data. New fields of medicine such as personalized medicine emerged, but they have not yet reached the majority of patients.

Promising new technologies similar to gene therapy require a very deep understanding of the underlying mechanisms, such as gene regulation. Many aspects of gene regulation are being intensively studied today, including protein-DNA interaction. Protein-DNA interaction is a very important mechanism in gene regulation that has been intensively studied for many years. However, knowledge in this field is still far from complete and with every improvement in tools and protocols such as ChIP-nexus and ChIP-exo, which is a method similar to ChIP-seq but extended by applying an exonuclease, this knowledge can be taken a step higher [5]. Because of the progress with NGS, the amount of data that is being processed in research fields such as protein-DNA interaction has reached a volume of multiple gigabytes or even terabytes. As this trend will continue, efficient algorithms and tools are more important than ever to keep research fast and efficient.

1.2 Next generation sequencing

Next generation sequencing is currently being used in many areas of medical and biological research. It is also a key element of the ChIP-nexus protocol, where it is used to sequence DNA fragments. The application of of NGS in the ChIP-nexus protocol is described later in section 1.5. This section gives a general overview of how NGS is performed.

Before the actual sequencing can start, a sequencing library, which is a solution that contains different DNA fragments which make up all the sequences to be analyzed, needs to be prepared. This comprises fragmentation of DNA, repairing of ends, adapter ligation and Polymerase Chain Reaction (PCR) amplification. Because library preparation is quite different depending on the sequencing equipment used, I compare two library preparation protocols, the Illumina `TruSeq DNA PCR-Free` [6] and Illumina `Nextera DNA` [7]. The first step of library preparation is DNA fragmentation. For the `TruSeq DNA  PCR-Free` protocol, this is done mechanically using ultrasound, whereas the `Nextera DNA` protocol uses enzymatic fragmentation. The advantage of the latter method is that the fragmentation enzymes, transposomes, also tag the ends of DNA fragments. This eliminates the need for repairing blunt ends. Furthermore, less laboratory equipment is needed. End repair for `TruSeq DNA PCR-Free` is done by incubating the sonicated DNA fragments with a mix of enzymes that have a 3'-5' exonuclease and 5'-3' polymerase activity. The large Klenow fragment of E. coli DNA polymerase I or alternatively T4 DNA polymerase can be used to achieve this.

Next, a single A (adenine) base needs to be added to the 3'-end of the DNA fragment, so that the adapters that have a T (thymine) overhang can be ligated in a subsequent step. This can be done by using the exo-Klenow fragment, which is a mutated version of the Klenow fragment that lacks 3'-5' exonuclease activity.

Then, the 5'-end of the DNA fragment needs to be phosphorylated, so that the ligase can later bind the adapter at this place. T4 polynucleotide kinase can be used to achieve this (Figure 1.1). Tagged DNA fragments from `Nextera DNA` do not need to undergo this procedure.

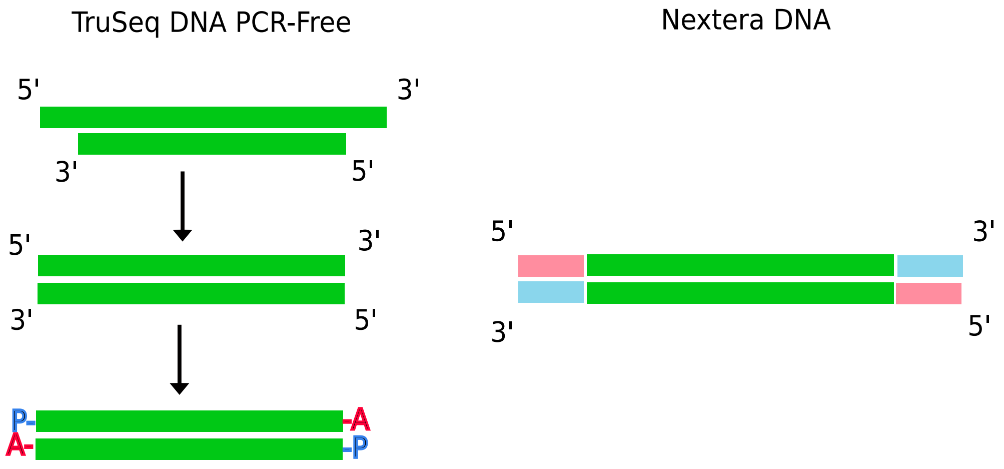


Figure 1.1: Adapter preparation step 1

Next, the adapters are ligated to the DNA fragments. This step is also different for each protocol. `TruSeq DNA PCR-Free` directly ligates the adapter sequences to the prepared DNA fragments, whereas `Nextera DNA` uses reduced cycle amplification to achieve the same goal. Figure 1.2 shows this for `Nextera DNA`. First, read 1 sequencing primer binds to the corresponding tag of the DNA fragment. Then DNA polymerase assembles a strand using the original DNA fragment as a template. Next, the original template is discarded and read 2 sequencing primer is used to assemble another sequence. This sequence now contains all the necessary regions for sequencing such as flow cell binding sites, index regions and sequencing start regions.

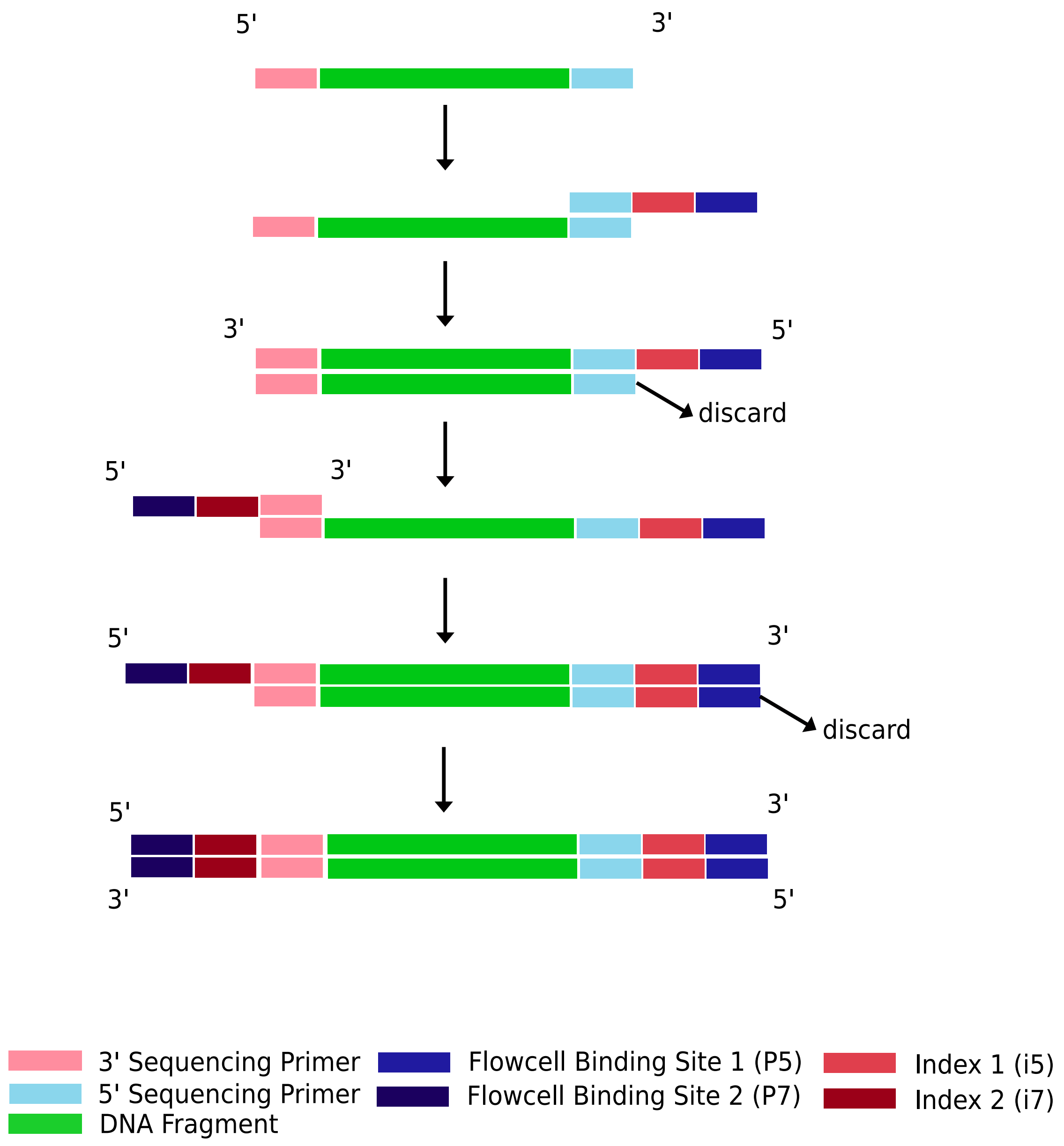


Figure 1.2: Nextera adapter preparation step 2

For `TruSeq DNA PCR-Free`, this step is quite different. As seen in figure 1.3, the adapters get ligated directly to the prepared ends of the DNA fragments. The resulting product is the same for both protocols. However, `Nextera DNA` employs two different index sequences at the 3' and 5', whereas `TruSeq DNA  PCR-Free` in this version only has one index region at the 5'-end. The use of two index sequences allows fragments from different experiments to be sequenced simultaneously. There are 8 different i5 and 12 different i7 index sequences

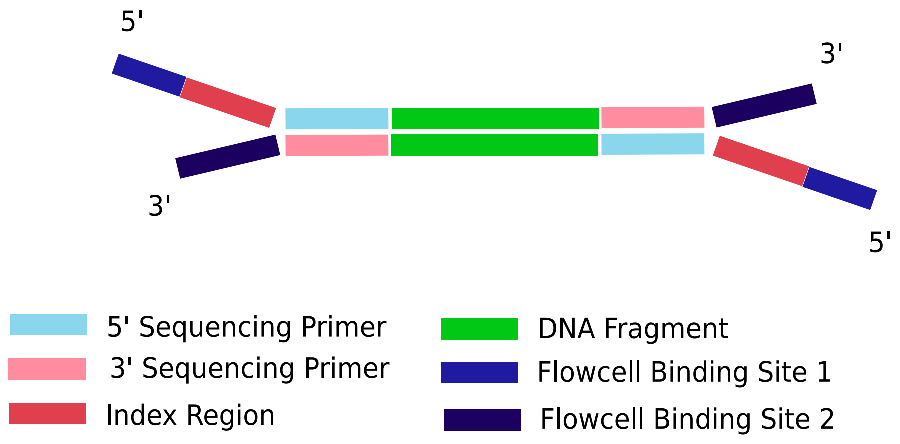
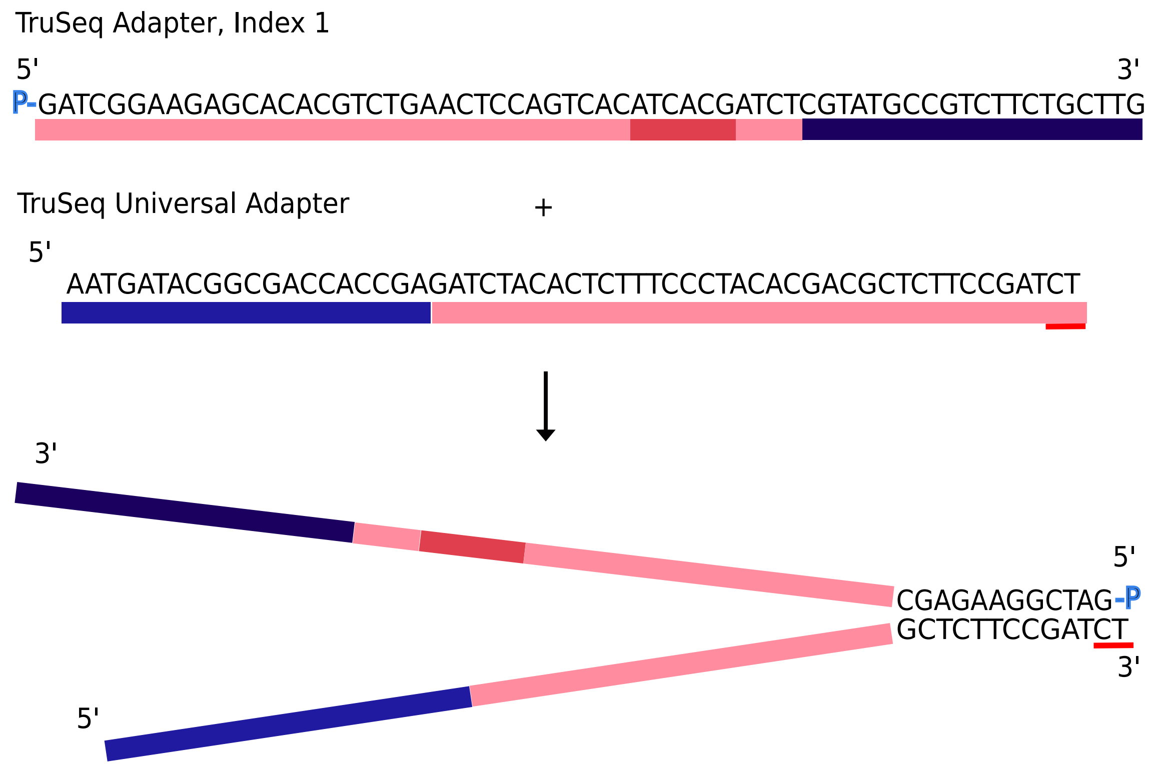


Figure 1.3: TruSeq adapter preparation step 2

available; therefore, the maximum number of different experiments that can be sequenced on the same flow cell is $8 \cdot 12 = 96$. To better understand the different regions of the adapter, see figure 1.4, which shows the situation for `TruSeq DNA  PCR-Free` in greater detail. The two different adapters combine to form a double stranded adapter. This is possible because 12 bases at the 3'-end of the universal adapter are reverse complementary to 12 bases at the end of the barcoded adapter. Figure 1.4 illustrates how these complementary bases facilitate the formation of the double stranded adapter.



TruSeq Adapter, Index 1
5'
3'
P-GATCGGAAGAGCACACGTCTGAACTCCAGTCACATCACGATCTCGTATGCCGTCTTCTGCTTG
TruSeq Universal Adapter
+
5'
AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCT
3'
5'
CGAGAAGGCTAG-P
GCTCTTCCGATCT
3'
5'

Figure 1.4: Note that there is a phosphothiorate bond, marked with a red bar. This prevents the adapter from being digested by exonuclease activity. Oligonucleotide sequences, ©2007–2011 Illumina, Inc. All rights reserved.

For `TruSeq DNA  PCR-Free`, the next step is gel purification. The goal of this step is to filter out PCR amplicons with the desired size without unwanted contamination such as primers and nucleoside triphosphates containing deoxyribose (dNTPs). This can be achieved by cutting out an appropriate slice from the electrophoresis gel. The width of the slice is determined by the range of fragment sizes we want to have. The selected fraction from gel electrophoresis is then amplified by PCR.

`Nextera DNA` uses `AMPure XP` instead of gel purification for post-PCR cleanup and size selection. This process binds PCR amplicons to paramagnetic particles, which then can be filtered out of the solution by using a magnet. Next, the PCR amplicons will be transferred to the flow cell. The Illumina flow cell is a small glass plate with 8 channels. Within each channel, there are many microscopic indentations, called wells. The flow cell binding sequences are present exclusively in these islands, which prevents mixing of different amplicons. A proprietary technology ensures that only one amplicon can bind to each well.

The next step is bridge amplification to create wells that have many clones of the same amplicon, the so called clusters. Figure 1.5 shows a small section of a flow cell well. The black bars at the base of the binding sequences represent the binding to the glass substrate. After the DNA template has hybridized with the binding sequence, a DNA polymerase starts to assemble a reverse complementary strand, which is anchored at the glass plate. After this is completed, the original DNA template is washed away.

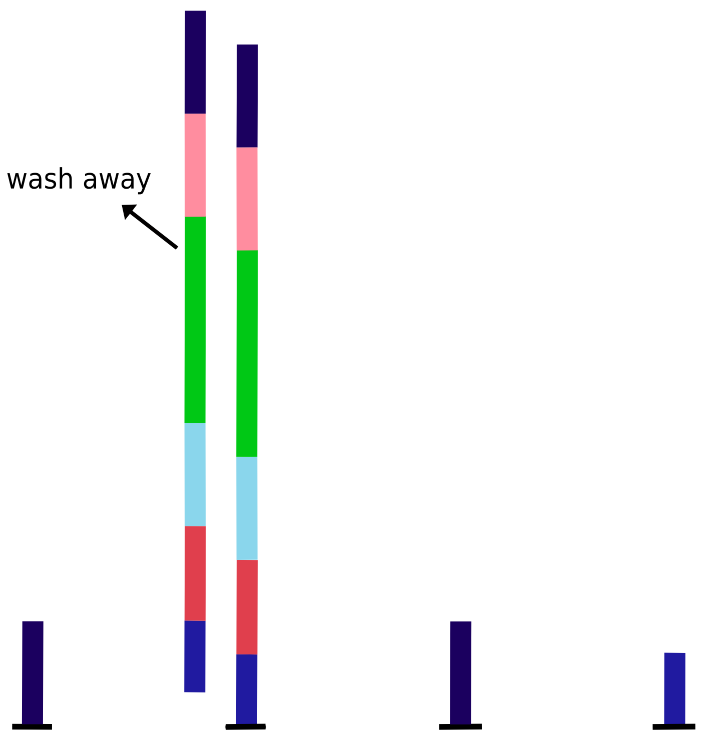


Figure 1.5: Bridge amplification step 1

The ends of the newly assembled DNA segment are now de-protected so that it can bind with its free end to a neighboring binding sequence on the glass substrate, which creates the typical U-shape as shown in figure 1.6. Then a DNA polymerase again assembles a reverse complementary strand.

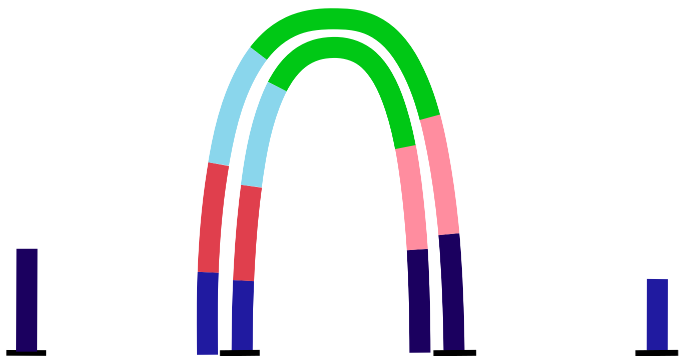


Figure 1.6: Bridge amplification step 2

Figure 1.7 shows how in the last step, after all binding sequences are populated with clones of the original template, the reverse strands are cleaved and washed away so that only the forward strands remain.

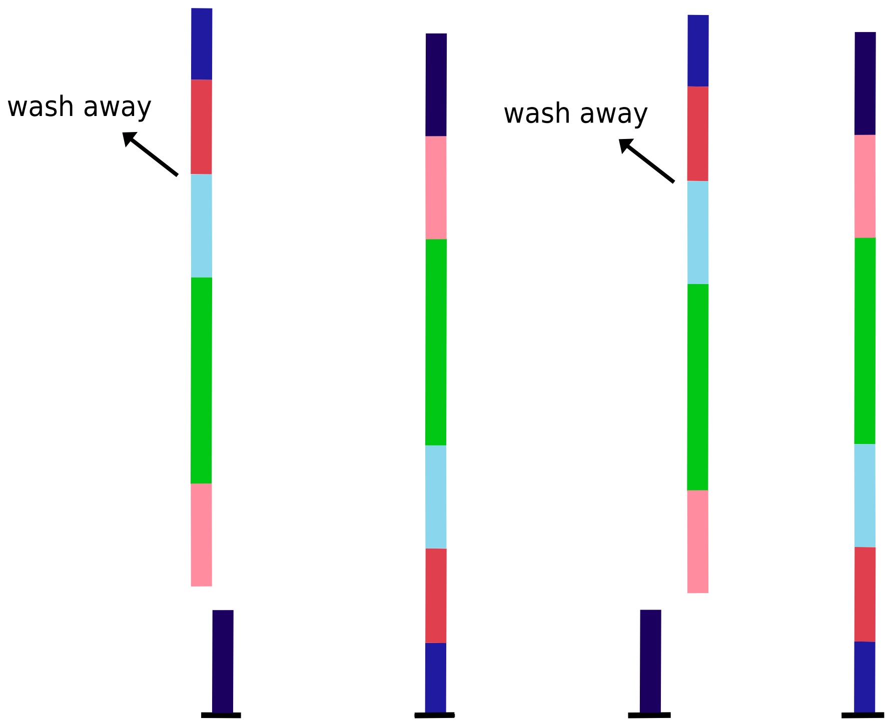


Figure 1.7: Bridge amplification step 3

Now, the actual sequencing can start. Most modern NGS systems employ a sequencing by synthesis (SBS) method. A common technique is to use cleavable fluorescent nucleotides [8], which are fluorescent for different wavelengths depending on the base that they carry. The readout of the sequence is then done by first adding a single cleavable fluorescent nucleotide to the sequence under construction. This causes all the labeled nucleotides within one cluster to light up in the same color, which is favorable for detection of the signal. The fluorescent nucleotides consist of a normal nucleotide, a fluorescent molecule and a sequence terminator, which prevents multiple nucleotides from being inserted in the same cycle. After readout of one base, the fluorescent molecule and the terminator are washed away, which leaves only the normal nucleotide in the sequence. This enables the next fluorescent nucleotide to be added, which happens in the next cycle.

After a number of cycles, the synthesis of the different clones within one cluster can get out of sync. In this case, more than one base is detected at the same cycle, which makes it harder to call the correct base. Therefore, the quality of reads usually deteriorates towards the end, which makes quality trimming necessary. The tools developed in this work can also perform this task.

1.3 Analyzing protein-DNA interaction using sequencing technologies

Protein-DNA interaction is one of the key elements in gene regulation. It happens, for example, when an estrogen-bound estrogen receptor migrates to the cell nucleus [9], or more generally speaking, every time a transcription factor controls the expression of a gene. Transcription factors control the transcription of a gene by physically interacting with DNA of the target gene or surrounding DNA.

ChIP-seq [10], which is a method that combines chromatin immunoprecipitation with next generation sequencing, was the first method widely used to analyze genome-wide protein-DNA interaction at large scale. It is based on chromatin immunoprecipitation (ChIP), which is a method that uses antibodies to isolate a Protein Of Interest (POI) bound to DNA. More details about ChIP are described later in section 1.5.1. After applying ChIP, there is a subsequent DNA sequencing to identify DNA sites that interact with the POI. The achievable positional resolution with ChIP-seq is about ± 50 bps (base pairs). Using ChIP-seq, many target genes of the transcription factor `TAL1`, which plays an important role in differentiation of multiprogenitor cells, could be identified [11].

ChIP-seq studies could show that `TAL1` [11] frequently forms a complex with `GATA1` [12] and binds to DNA sites that have a motif for `TAL` and `GATA` next to each other, as shown in figure 1.8.

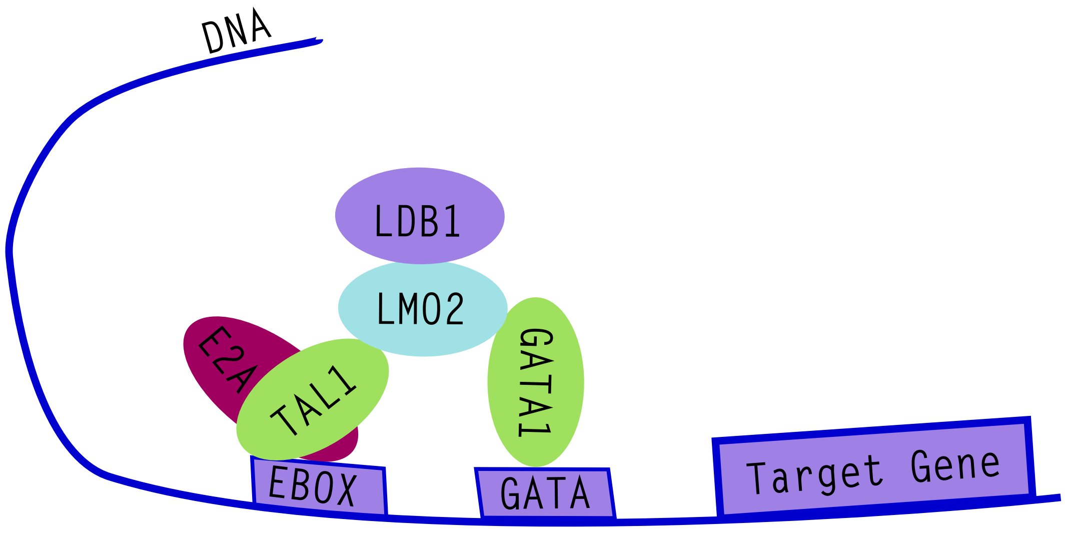


Figure 1.8: Interaction of `TAL1` and `GATA1` with DNA

Figure 1.8 illustrates that the interaction of proteins with DNA can be very complex

- much more complex than shown in this simplified illustration. Because of the 50 bps resolution limit, it was not possible to understand the mechanism of protein-DNA interaction at these sites where gene regulation happens in detail. Until recently, it was very difficult to gain further insight into the details of these gene regulatory networks and the details that govern the mechanisms of protein-DNA interaction.

With the advent of ChIP-exo [13], it was possible for the first time to achieve a positional resolution of 10 bps and below. This new method enabled us to better understand the interaction between `TAL1` and its target genes on the DNA. It helped to elucidate the mechanism of how `TAL1`, `E2A` and `GATA1` interact [14]. A better understanding of these mechanisms is also important in many fields of modern medicine. It can, for example help to find cures for diseases like leukemia, where the differentiation from multiprogenitor cells to red blood cells plays a role [15].

If we look more closely at how protein-DNA interaction takes place physically, we can often find recurring patterns such as the helix-loop-helix. In this case, the protein that interacts with DNA has two subunits, each of which consists of a helix followed by a loop followed by another helix.

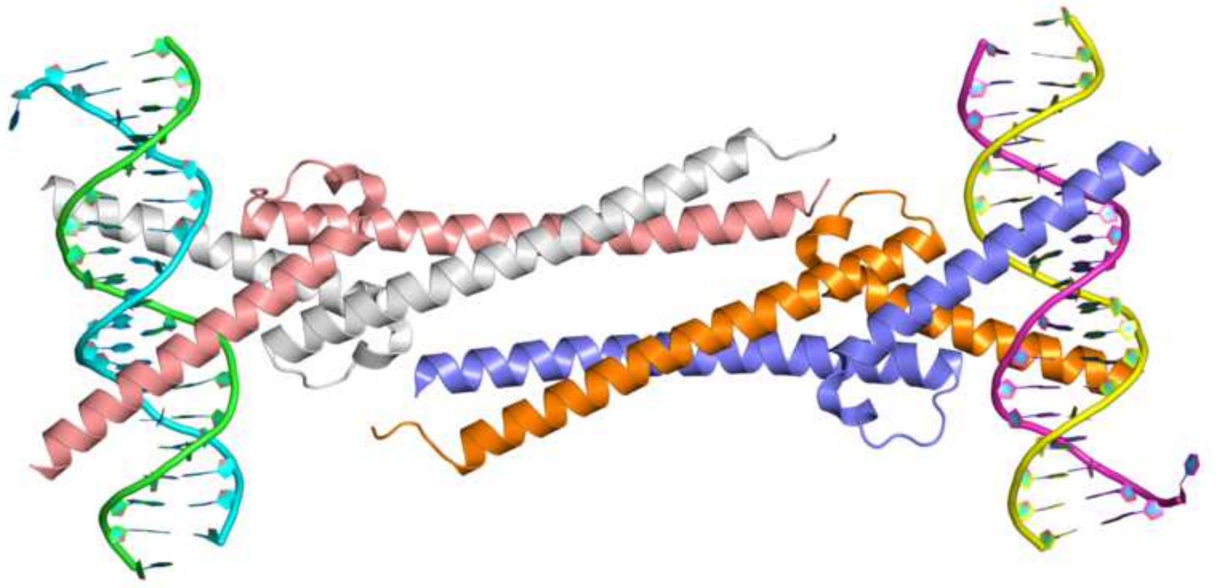


Figure 1.9: Interaction of `myc` with DNA, source PDB

Knowing the basic mechanism of interaction, like helix-loop-helix is not enough to fully understand the role of protein-DNA interaction in gene regulation. Proteins often form complexes with other proteins and recognize specific binding sites on the DNA. The methods presented in this work help to investigate these complex interactions.

1.4 Other types of gene regulation

While the methods presented in this work especially highlight the effect that protein-DNA interaction has on gene regulation, there are also other important mechanisms for gene regulation, such as methylation and histone modifications. Those mechanisms are also very important to recognize in order to understand the mechanisms that lead to cancer [16–18]. Methylation of DNA typically occurs at locations where a cytosine nucleotide is next to a guanine nucleotide, coupled by a phosphate (CpG). Recently, NGS methods have enabled a better picture of methylation patterns, especially in cancer cells.

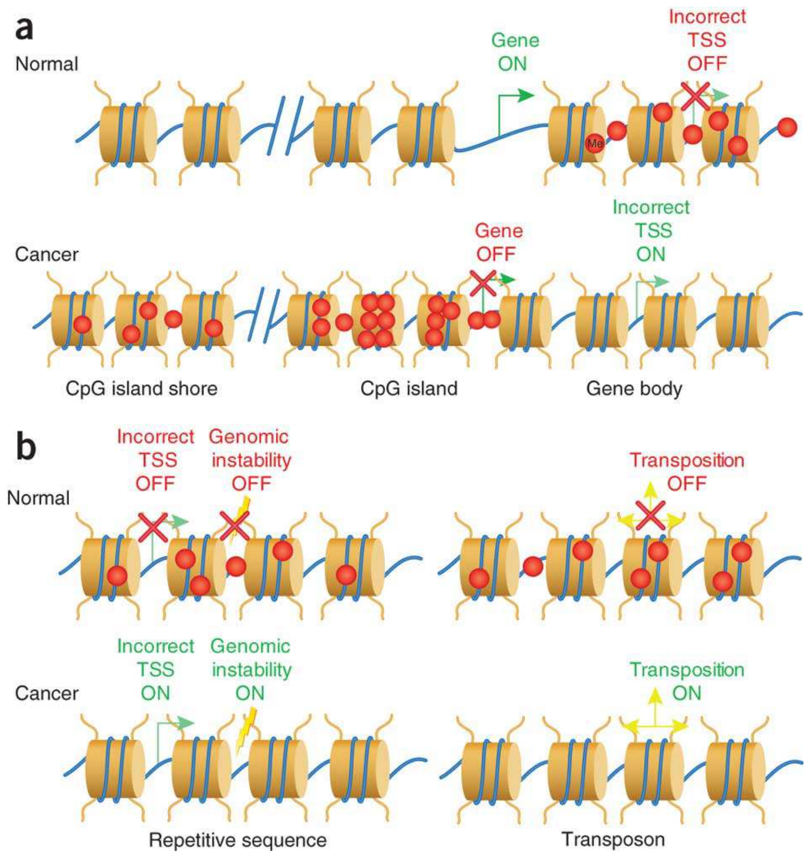


Figure 1.10: The cylindrical beige bodies depict histones with DNA around them. The red dots show methylated spots on the DNA. Source: Rodriguez-Paredes et al. [18]

Figure 1.10a shows that many regions of DNA that contains many C and G bases (CpG island) are unmethylated in normal cells but methylated in cancer cells. Methylation outside of CpG islands (CGIs) can also occur, as shown in the upper right corner of figure 1.10a. It is thought that the function of this type of DNA methylation is to suppress transcription from sites other than the genuine transcription start site (TSS) [18]. Cancer cells often develop abnormal methylation patterns that can disable "good genes" and also turn on wrong transcription start sites by not methylating them. CpG rich regions, which are typically in the promoter regions of genes, are also used to regulate transcription by getting selectively methylated or demethylated. This is one of the mechanisms used to enable tissue specific transcription of genes.

Figure 1.10b shows how methylation can reduce mutagenic events in healthy cells. This is achieved by methylating repetitive sequences which would otherwise be unstable. In cancer cells, these methylations are frequently lost, which leads to genomic instability and a more malignant tumor behavior.

Not only DNA, but also histones can be methylated. Histones are proteins, which are used to wind up DNA that is currently not being transcribed. This allows for a tight packing of DNA so that it can fit in the small cell nucleus. This tightly packed DNA is called heterochromatin, because when observing it through an electron microscope, it appears very dark and heterogeneous. Euchromatin is the lightly packed form of DNA which is currently being transcribed. Through an electron microscope, it looks bright or white with only little dark spots in it. Histone methylation, which is a specific form of histone modification, has been shown to play an important role in healthy as well as cancer cells. It exerts its effect on gene regulation by influencing chromatin structure as well as controlling the interaction with transcription factors [19].

Methods presented in this thesis can be used to analyze the regulation of histone modifications. ChIP-seq with antibodies against methylated H3K4 histone (H3K4me3) showed that CXXC finger protein 1 (Cfp1) depleted cells have reduced levels of H3K4me3 [20]. This means that the recruitment of H3K4me3 is positively influenced by the presence of Cfp1. Another type of histone modification that can be examined using ChIP-seq and its successors is acetylation. For example, ChIP-seq experiments indicated that dysregulated acetylation of lysine residues of histones correlated with memory impairment in mice [21].

The examples mentioned above give an impression of the complexity and diversity of gene regulation. Different flavors of ChiP experiments, such as histone modification, ChiP-seq [22] and DNase-seq help to further analyze these interactions [23]. Methods with high positional resolution such as ChIP-nexus applied to these areas lead to results that are valuable for medical research and a better understanding of biology in general.

1.5 ChIP-seq and ChIP-exo

In this section I briefly describe the workflow of ChIP-seq, because the ChIP-nexus workflow in many parts is based on it.

1.5.1 Chromatin immunoprecipitation

ChIP-seq [10] and ChIP-exo [13] are similar to ChIP-nexus [24] methods that uses chromatin immunoprecipitation [25] in order to identify protein-DNA binding profiles. The first step, which all of these protocols have in common, is chromatin immunoprecipitation. I give a brief overview on how this procedure is performed.

In a living cell, proteins usually bind very loosely to DNA, usually noncovalently. Chromatin immunoprecipitation requires that these loose bonds are converted to stable cross-links. This is usually done by adding formaldehyde to protein-DNA complexes [26]. Next, the protein-DNA complexes, many of which are still connected to each other by DNA strands, are then separated by sonication. Sonication uses sound energy to shear long strands of DNA into shorter strands. After DNA fragmentation, the complexes, consisting of the POI bound to DNA fragments, are isolated using antibodies that are specific for the POI. There are two methods to perform this isolation procedure. The first method is called direct method, because the antigenes that are used are directly coupled to beads when they are added to the protein mixture. The indirect method adds free antibodies to the protein mixture and lets them bind to the targets first. Then beads with universal antibody binding proteins called `protein A/G` attached to them are added to the protein mixture. From here on, both methods are the same.

1.5.2 Library preparation for ChIP-seq

The DNA fragments that were obtained as described above are then used to build a library as described in section 1.2. The fragments, which are usually selected by gel electrophoresis and subsequent excision, have a typical length in the range of 200 – 400 base pairs. Shorter segment lengths can give a better positional accuracy; however if the segments become too short, a greater fraction of them cannot be mapped uniquely to the reference genome anymore.

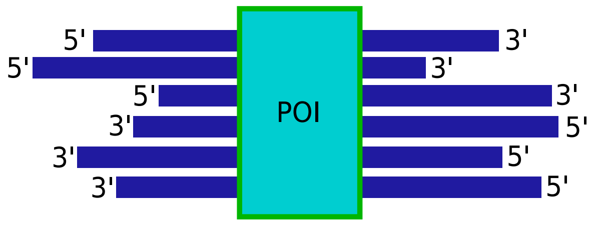


Figure 1.11: Possibilities for DNA fragments to bind to the POI in ChIP-seq experiments

Figure 1.11 shows how DNA fragments might be attached to the POI. Usually there is only one DNA fragment attached to one protein; therefore, each DNA fragment depicted as a blue line needs to be interpreted as one possible configuration how a DNA fragment can bind to the POI.

1.5.3 Library preparation for ChIP-exo

Library preparation for ChIP-exo involves an extra step, as it uses an exonuclease to digest bases at the 5'-end up to the POI. The exonuclease removes bases starting from the 5'-end until it is blocked by the POI, which is covalently bonded to the POI. The result of this process is shown in figure 1.12.

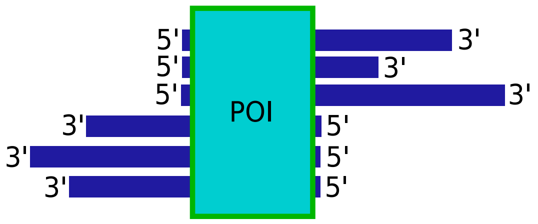


Figure 1.12: Possibilities for DNA fragments to bind to the POI in ChIP-exo experiments

For ChIP-seq, it was not possible to only use the 5'-end locations as inputs for peak-calling, because the position of the 5'-end relative to the POI was not deterministic. Now with the exonuclease digestion, the 5'-ends are usually right next to the POI and can therefore be used to better estimate the binding site of the POI within the genome. ChIP-nexus builds in many parts on the ChIP-exo protocol, and is described in more detail in the next section.

1.6 ChIP-nexus: A new method for analyzing protein-DNA interaction

1.6.1 Overview

ChIP-nexus is a method to study protein-DNA interaction similar to ChIP-seq and ChIP-exo. The first step of the experiment is to select a POI, for which we want to study the DNA binding profile. After the POI has been selected, it is important to select an appropriate state of the cell at which we want to study the protein-DNA interaction. Like all living organisms, a cell is never in a steady state, it can be growing, dying or undergoing mitosis. We also need to take into account that the protein-DNA extracts used in this analysis originate from an ensemble of cells and therefore usually represent a combination of different states. Chromatin immunoprecipitation, described in section 1.5.1, is used to filter out the POI-DNA complexes from a cell lysis concentrate.

Next, λ-exonuclease is added, as in the case of ChIP-exo [13]. This exonuclease has a 5'-3' exonuclease activity that digests the DNA strands starting from the 5'-end to the first cross-link (covalent bond between the DNA fragment and POI). The exonuclease does not always digest the DNA right to the first cross-link; for instance exonuclease activity can be blocked by other proteins than the POI bound to the DNA. One possible explanation is that other proteins are bound to the POI as shown in figure 1.8, and possibly block access of the λ-exonuclease to the POI.

After digestion, the sample needs to be prepared for sequencing. This step includes separating the POI from the DNA fragments and performing a library preparation similar to the one used in the `TruSeq DNA PCR free` protocol. However, there are some important variations. Most notably, random barcodes are added to every read primer, which make it possible to distinguish in later analysis duplicates that originate from PCR from duplicates that originate from different POI-DNA complexes. Duplicates from different POI-DNA complexes are valuable for our analysis, whereas duplicates from PCR do not carry additional information.

The data analysis for ChIP-nexus consists of several steps that are described below in more detail.

1.6.2 Library preparation

Library preparation for ChIP-nexus is based on the methods used for ChIP-exo [13]. After fragmentation, end-repair and dA-tailing is performed in a fashion similar to that described in section 1.2 for the `TruSeq DNA PCR free` protocol. The remaining part, however, is different and uses some elements from the `Nextera DNA` protocol.

Unlike in the `Nextera DNA` protocol, where the tag adapters are directly ligated to DNA fragments, ChIP-nexus uses special intermediate adapters that contain random and fixed barcodes and a special region, where cutting can be performed during later steps. Figure 1.13 shows the cutting region in pink color. The adapters are a reverse complementary pair, only the barcoded adapter has a non-complementary part that contains the barcodes at its 5'-end and a T-overhang at its 3'-end. The T-overhang is necessary to bind to the DNA fragment, which has an A-overhang at its 3'-end. Therefore, it is not possible to use completely reverse complementary adapters from the beginning.

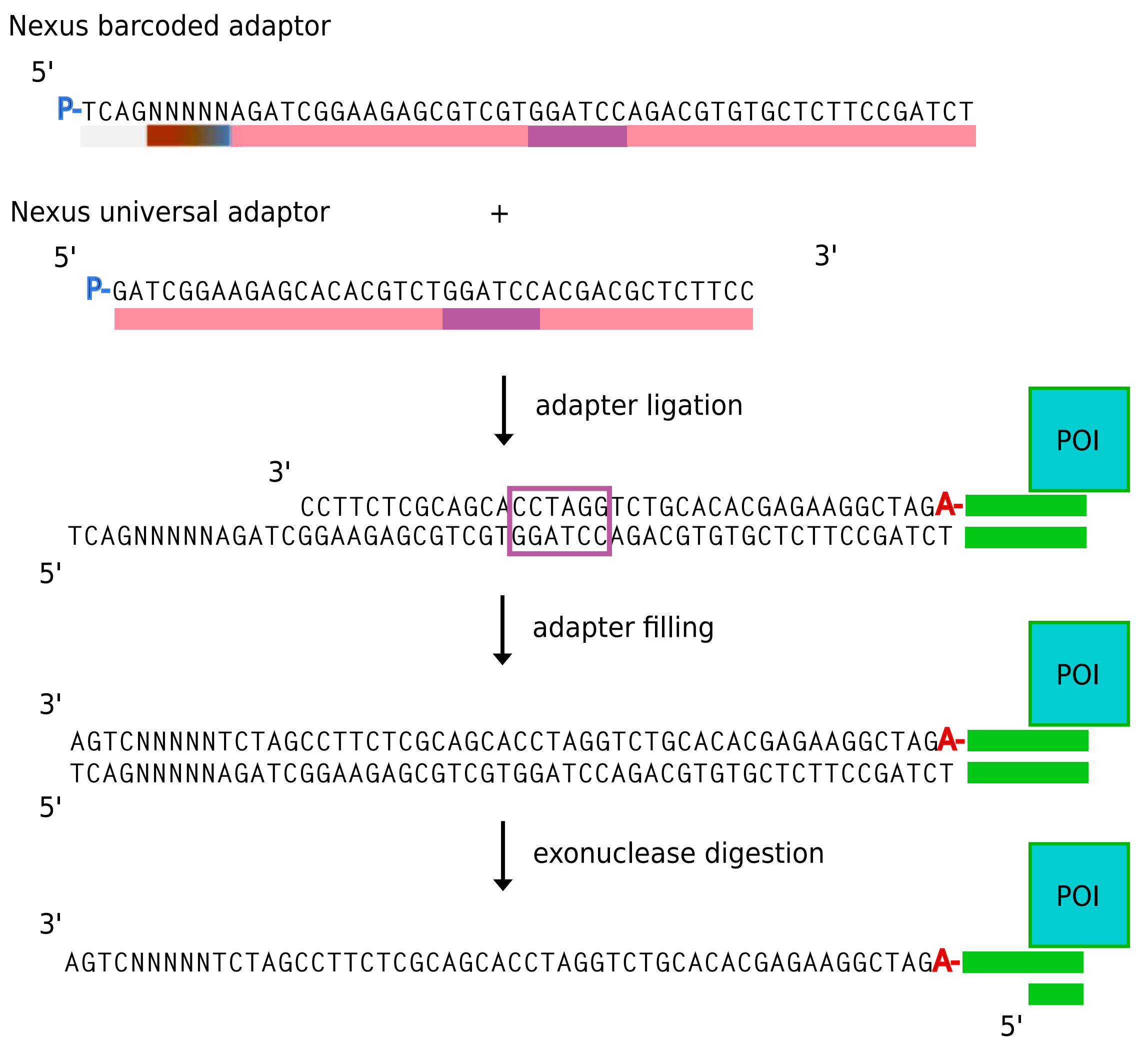

Nexus barcoded adaptor
5'
P-TCAGNNNNNAGATCGGAAGAGCGTCGTGGATCCAGACGTGTGCTCTTCCGATCT
Nexus universal adaptor
+
5'
3'
P-GATCGGAAGAGCACACGTCTGGATCCACGACGCTCTTCC
adapter ligation
POI
3'
CCTTCTCGCAGCACCTAGGTCTGCACACGAGAAGGCTAGA-
TCAGNNNNNAGATCGGAAGAGCGTCGTGGATCCAGACGTGTGCTCTTCCGATCT
5'
adapter filling
POI
3'
AGTCNNNNNTCTAGCCTTCTCGCAGCACCTAGGTCTGCACACGAGAAGGCTAGA-
TCAGNNNNNAGATCGGAAGAGCGTCGTGGATCCAGACGTGTGCTCTTCCGATCT
5'
exonuclease digestion
POI
3'
AGTCNNNNNTCTAGCCTTCTCGCAGCACCTAGGTCTGCACACGAGAAGGCTAGA-
5'

Figure 1.13: ChIP-nexus adapter bound to insert DNA

After ligating the adapters to the DNA fragment, missing nucleotides for the short universal adapter need to be added. This is done by using the same enzyme that was used before to repair the ends of the DNA fragments. As shown at the bottom of figure 1.13, we have the original DNA fragment attached to a double-stranded nexus adapter on both ends. Figure 1.13 shows only one side of the DNA fragment.

In the next step, digestion from the 5'-end to the first cross-link is performed by using λ-exonuclease in a manner similar to that described in the ChIP-exo protocol. This gives us two single-stranded DNA fragments.

The barcodes need to be moved directly in front of the sequence of interest, because the whole adapter region is not sequenced. The reason that the barcodes are not directly adjacent to the DNA fragment is that this region needs to be reverse complementary so that two adapters can form a pair. This region also serves as a matching sequence for the Illumina primers, which is used in a later step.

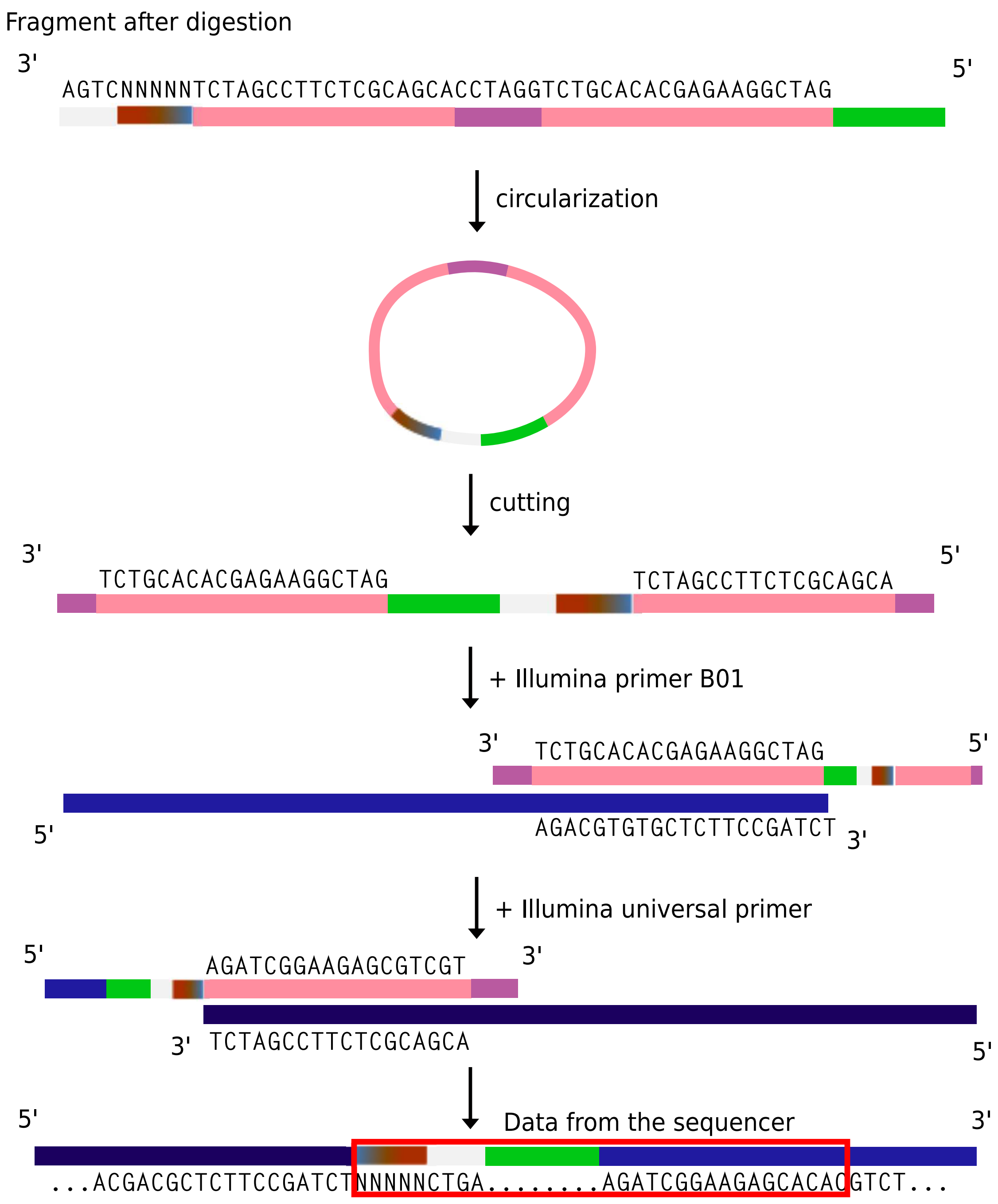


Figure 1.14: Circularization process during ChIP-nexus sequencing library preparation

The DNA fragment shown at the bottom of figure 1.14 is then transferred to the flow cell and sequenced.

1.7 Mapping of NGS Data

The data which comes out of the first stage of preprocessing consists of many short artifact eliminated DNA sequences, one for each DNA fragment. The next step in analyzing the data is called mapping. The mapper usually outputs a sam file, or the binary version of a sam file, which is a bam file. The sam file contains one line of ASCII text for each read. An example is given here:

```
SRR1175698.30 0 chr2L 21725514 255 35M * 0 0 NCTTCCTGATAAGAAGA
TGCGTAACGGCCATACATTGTTTTTGC   HHHBGHIIIIIIHIGIIIIIIIIIIIIIIHIIIIII XA:i:0
MD:Z:35 NM:i:0
```

This example shows that there are several tabular separated fields in one line. The first field is the read id. The second field is a bit flag that contains information about the mapping, the bit masks that matter the most to us are 0x04 for unmapped and 0x10 for reverse mapped reads. In the example shown above, this means that the read could be successfully mapped to the forward strand. The next field is the chromosome and the field after that the position within the chromosome where the read has been mapped to.

After mapping has been completed, it is recommended that the result is visually examined in a viewer like IGV [27].

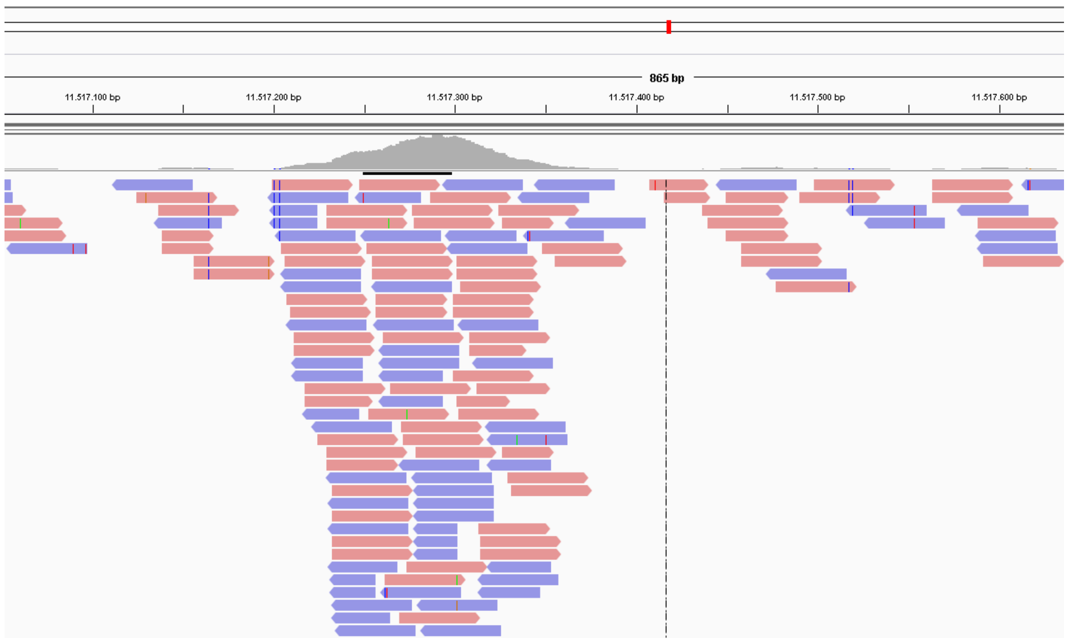


Figure 1.15: Mapped reads around a peak region shown in `IGV`.

Figure 1.15 shows a screenshot from IGV. The red bars represent forward mapped reads, the blue bars reverse mapped reads. It is also important to notice the difference between the 5'-end and the 3'-end. The 5'-end is depicted by a vertical bar, whereas the 3'-end is shown as an arrow.

There are several difficulties that can occur during mapping. Within the genome, there are many repeat or N-regions to which a short read cannot be unambiguously mapped. This problem can be solved by configuring the mapper to only report mappings that can be unambiguously mapped. A more difficult case is where the repeat regions in the reference genome could not be resolved during genome assembly and are replaced by N's. N is the abbreviation for any base and it can be A,C,G or T. These regions are also called low mappability regions. At the end or beginning of such a repeat region, there is usually one repeat that can still be unambiguously resolved. This can lead to situations which originate from repeat regions that can now be unambiguously mapped to the reference genome. At these positions, we can often find many reads, so called clusters, because reads from many different locations are collapsed into one location. These clusters are especially problematic when we use a global cross-correlation to estimate the average fragment length, since they usually cause a phantom peak in the cross-correlation at the read length. We need to distinguish carefully between read length and fragment length. A fragment is the DNA sequence that is bound to our protein of interest and the read length is the number of bases that the sequencer sequences for each fragment. Since the read length can be specified at the sequencing machine, it is a known value. Figure 1.16 shows a typical cluster and the read length after trimming is 42.

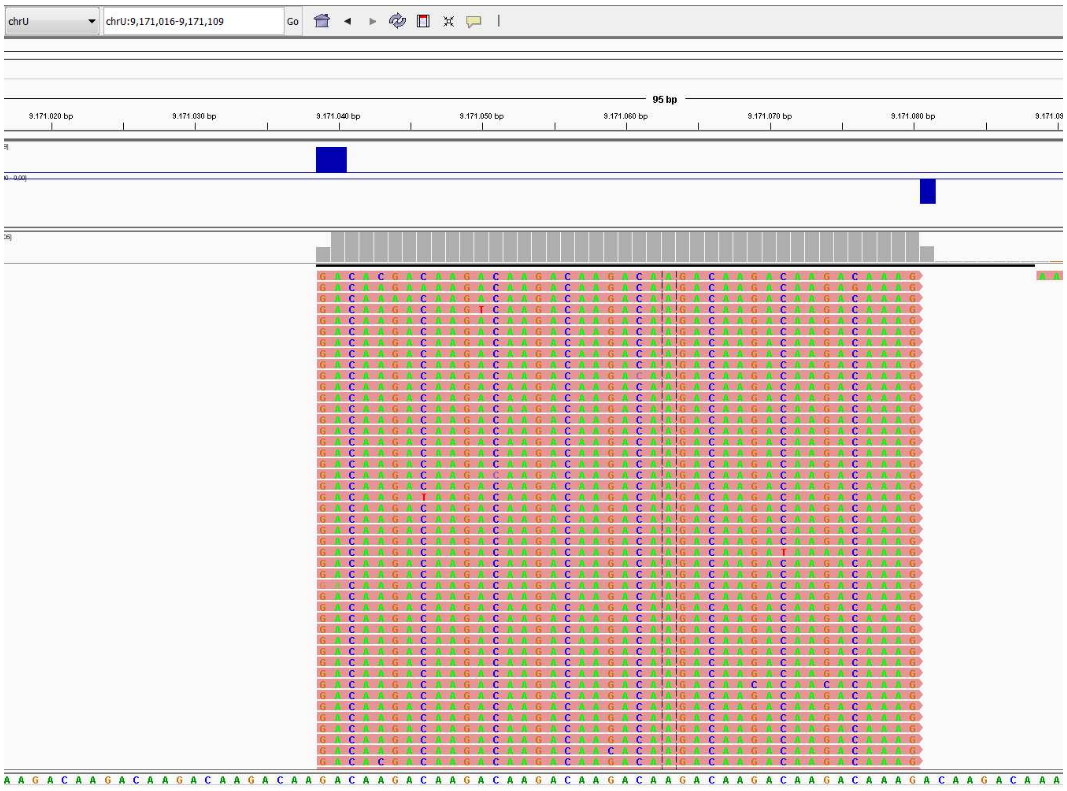


Figure 1.16: Mapped reads around a tower shown in `IGV`. Each horizontal red bar represents a mapped read. Red bars pointing to the right belong to reads mapped to the forward strand while the ones pointing to the left (not visible in this screenshot) represent reads mapped to the reverse strand. The gray vertical bars represent the read coverage. The two blue bars above it show the coverage for the 5'-ends, pointing upwards for reads mapped to the forward strand and pointing downwards for reads mapped to the reverse strand.

Figure 1.16 shows why we can now also understand, why these clusters cause phantom peaks. If there is a unambiguously mappable position within a low mappability region, many reads get collapsed to this position as mentioned before. However, the same is true for the reverse complementary strand. As I will describe in greater detail later in section 1.8, peak calling always uses information that is contained in the correlation between the forward and the reverse strand. Therefore, it is explainable why clusters which have a fragment length equal or similar to read length can lead to phantom peaks.

Until now, a straightforward solution for the problem of low mappability and phantom peaks does not exist. Methods proposed to remedy this issue include excluding repeat regions from the analysis [28], calculating a global mappability score [29], or using a mappability-sensitive cross-correlation [30]. For ChIP-seq, the phantom peak was not as big as a problem as for ChIP-nexus, because the read length is usually much smaller than the fragment length. For ChIP-nexus, however, the read length and the fragment length are in the same range; therefore, methods regarding cross-correlation analysis need to be modified.

1.8 Peak calling

Figure 1.15 shows how peaks form in some areas of the genome. A peak usually consists of a downward and an upward pointing summit. Since summits represent mappings of DNA fragments that were bound to the protein of interest, they usually appear in regions of the genome where the protein of interest interacts with DNA.

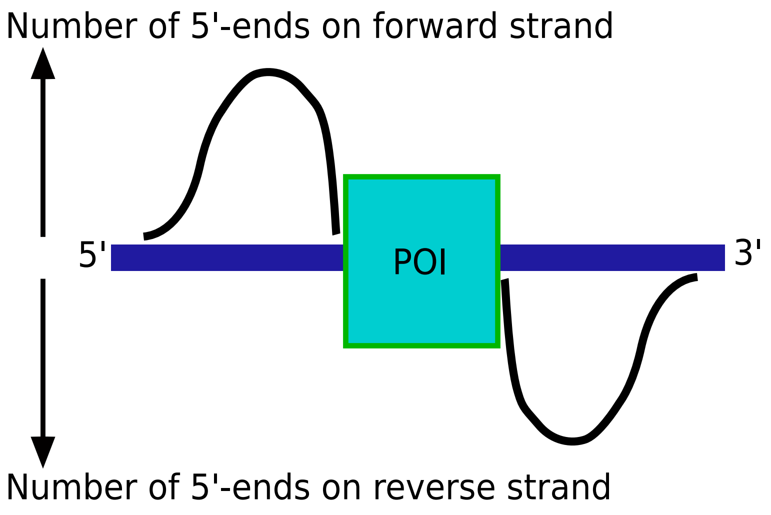

Number of 5'-ends on forward strand
5'
POI
3'
Number of 5'-ends on reverse strand

Figure 1.17: Typical distribution of 5'-ends around a peak

Figure 1.17 shows a typical distribution of 5'-ends around a peak. There are two summits visible. The upward pointing curve on the left hand side of the graph is a summit that is made up of the 5'-ends on the forward strand. The other summit is represented by the downward pointing curve on the right hand side and is made up of the 5'-ends on the reverse strand. Both summits together, one on the forward strand and one on the reverse strand, make up a peak. The center of the peak should be somewhere between them and represents the middle of the DNA binding site of the POI. Normally, the number of 5'-ends on the forward and reverse strand around a peak have roughly equal magnitudes. A deviation could be caused by other proteins obstructing the DNA on one side of the POI. Algorithms used in peak callers usually detect peaks by first scanning the mapped reads for summits and then using those summits to calculate the possible peak candidates. Detecting a summit seems easy at first, but it requires a threshold. Setting this threshold right is not an easy task and usually requires parameter tuning. If the threshold is too low, many small summits are detected and might impair the detection of larger summits which lie on top of them. A threshold that is too high might not detect weak protein-DNA interactions where only a small fraction of the POI is bound to DNA. The simplest peak calling algorithm would look for summits on the forward and reverse strands that are next to each other and define the exact middle between them as a peak. In reality, it is a bit more complicated, because one has to use a window to group summits on the forward and reverse strand together, otherwise one would get peaks made of summits that actually belong to different POIs. The rationale behind this is since we assume that a POI can have only a certain length, summits that are further apart cannot make up a peak.

It is common for peak calling software to take a bam or sam file as input. The bam or sam files contain the mapped reads as described in section 1.7. The peak-caller then generates a bed-file out of this. Bed-files have one line for each entry. There are several format types of bed-files; in our pipeline, we use the bed4 format, which has four fields in each line, viz., chromosome, start position, end position and name.

There are several tools available for peak calling [1, 31–34]; however, since ChIP-nexus is very new, none of them are optimized for Q-nexus yet. Most existing peak callers are optimized for ChIP-seq and only a few are optimized for ChIP-exo. The peak caller MACS has been shown to deliver useable results for ChIP-nexus and is therefore used by the inventors of the Q-nexus method. In our group, we have extended our peak-caller Q [1] to include algorithms that can convert the potential that lies in the raw data of ChIP-nexus into a better prediction of DNA binding sites. We have also put a lot effort into making peak calling as little parameter-dependent as possible. The updated version of Q as well as the preprocessing tools developed within the scope of this thesis are published together as a tool chain called Q-nexus [35].

1.9 Motif analysis and interpretation of results

Motif analysis takes the peaks calculated by the peak-caller and the reference genome as input. An algorithm then looks for similar sequences of nucleotides around peaks. The motivation behind this is to find patterns of bases that our POI frequently binds to. A popular tool set for motif analysis is `MEME` [36].

The first step in motif analysis is usually finding the most significant motifs around the detected peaks, this procedure is also called de novo motif search. A simple algorithm for this task would calculate the nucleotide frequency for every position relative to the detected peaks. These nucleotide frequencies are then divided by the number of reads at that position. The algorithm outlined above is only a simple illustration of the principle, modern tools like `MEME` [36] use more complex algorithms. The resulting motifs are reported in an IUPAC DNA ambiguity code together with a probability matrix. The meaning of the ambiguity code is shown in table 1.1.

A	adenine
G	guanine
C	cytosine
T	thymine
Y	pyrimidine (C or T)
R	purine (A or G)
W	weak (A or T)
S	strong (G or C)
K	keto (T or G)
M	amino (C or A)
D	A, G, T (not C)
V	A, C, G (not T)
H	A, C, T (not G)
B	C, G, T (not A)
X/N	any

Table 1.1: IUPAC ambiguity code

To describe a motif using these codes, one could, for example, write `GGAAAWNC`. One possible Position Weight Matrix (PWM) corresponding to this motif would be

$$\begin{array}{c} \\ 1\\2\\3\\4\\5\\6\\7\\8 \end{array}\begin{array}{c} \begin{array}{cccc} \scriptstyle A & \scriptstyle C & \scriptstyle G & \scriptstyle T \end{array} \\ \begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.0 & 0.5 \\ 0.25 & 0.5 & 0.25 & 0.25 \\ 0.0 & 1.0 & 0.0 & 0.0 \end{bmatrix} \end{array}$$

PWMs generated by `Dreme` [36] have one column for each nucleotide and one row for each position within the motif. For the PWM given above, we can read for example that the probability for an adenine nucleotide at the sixth position within the motif is 50%.

One can use the `-eps` command line option of `Dreme` [36] to generate a graphical representation of the position weight matrix, as shown in figure 1.18. A smaller letter in this graph means less probability for occurrence of the denoted nucleotide at this specific position.

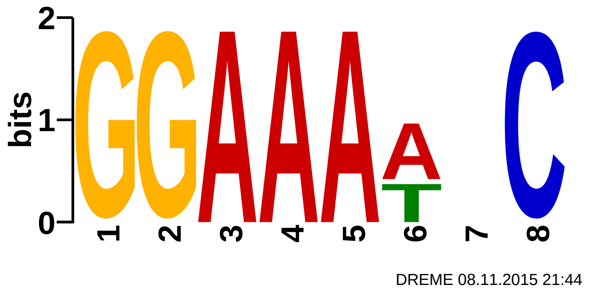


Figure 1.18: Motif GGAAAWNC

After the de novo motif search is completed, one can continue the analysis by separately analyzing the peaks in the vicinity of the most common motifs. Since every motif discovered by the de novo motif search can represent a different binding mechanism, it makes sense to analyze them separately. In order to do this, all reads around the motif, which is chosen from the list of discovered motifs, are selected using a motif search. Motif search searches the DNA sequence of reads that are mapped around peaks for the occurrence of this specific motif. Then the number of mapped reads, more precisely their 5'-end positions relative to the motif, are integrated over all selected peak regions to get a clearer picture of the binding mechanism for a given motif between the protein of interest and the DNA. In order to make this analysis less dependent on the peak calling algorithm, the relative position of the 5'-ends to the motif is usually used. An example of how this procedure can be performed is given in chapter 3.2.

2 Methods: Development of a preprocessing pipeline for ChIP-nexus

2.1 Overview of preprocessing

Preprocessing of ChIP-nexus data starts with raw reads that come from the NGS sequencer. The reads contain the following elements, starting from the 5'-end:

- Fixed barcode sequence
- Random barcode sequence
- Protein binding DNA sequence
- Adapter sequence

The first step of preprocessing filters all reads for the correct fixed barcode. This allows reads that were not bound to the protein of interest to be excluded from further analysis.

Next, the random barcode is removed from the sequence and saved in the meta data of the read for later use. After this, reads are adapter trimmed, which is done by specifying the sequences of the adapters that have been used in the experiment. These sequences are also called adapter templates. The preprocessing tool then tries to align the specified adapter sequences to the read. If there is a match, the matching bases are removed. This is necessary because adapter sequences would otherwise prevent reads from getting mapped to the reference genome.

The next step is mapping of the reads to the reference genome. We decided to use `Bowtie` [37] for this purpose because it has been proven to provide valid results for ChIP-nexus before [24]. `Bowtie` is a widely used program for mapping DNA sequences to a reference genome. We expect that other mappers which work for ChIP-seq or ChIP-exo can provide similar results, because ChIP-nexus has no special requirements in this regard. However, we believe that the data of ChIP-nexus experiments contains additional information that can be utilized by using peak callers specially optimized for ChIP-nexus.

An overview of all steps that are performed during preprocessing of ChIP-nexus reads is shown in figure 2.1. The figure shows that during preprocessing, the number of valid or mappable reads decreases. A noticeable amount of reads is usually filtered out, because they do not have a matching fixed barcode. Reasons for a non-matching fixed barcode could be problems during sequencing library preparation such as contamination with adapter dimers or cutting the circularized DNA strand at the wrong position.

Another large number of reads is filtered out during mapping, because some reads cannot be mapped unambiguously to the reference genome. This could be due to not properly removing adapter sequences or because of reads that originate from repetitive regions. Not properly removed adapter sequences can be caused by too many sequences errors so that the adapter does not match the adapter template good enough anymore.

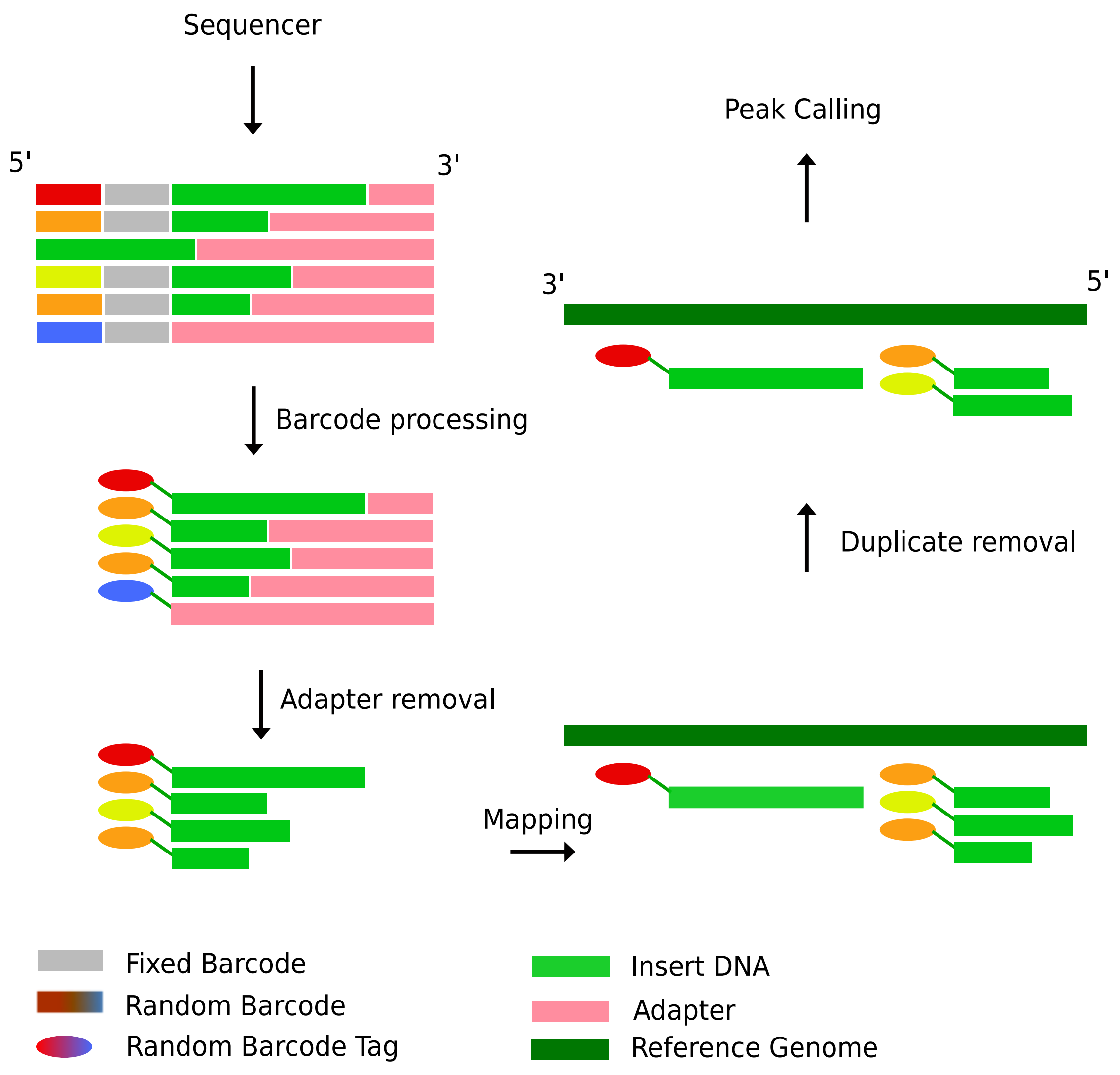


Figure 2.1: Preprocessing overview

Figure 2.2 shows the processing steps that are done before and after preprocessing. Within the scope of our Q-nexus project, we concentrate on the steps from preprocessing to motif analysis.

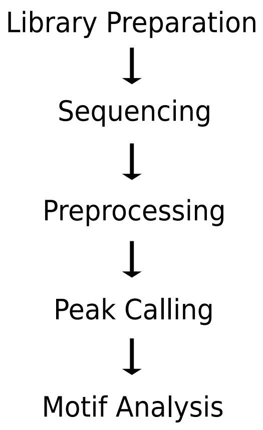


Figure 2.2: Overview of steps usually performed in a ChIP experiment

2.2 Flexcat: A tool for preprocessing NGS Data

2.2.1 Goals and functionality

Common steps in preprocessing NGS reads are quality trimming, adapter trimming and in some cases, barcode demultiplexing and length filtering. Existing tools [38, 39] can perform these tasks with certain limitations and a good overview can be found in a paper by Jiang et al. [38]. However, the existing tools are not optimized for ChIP-nexus; most importantly, they cannot process the random barcodes that were introduced by this protocol.

Normally, length filtering is not necessary because the settings that we use for mapping of reads to the reference genome ensure that only unambiguously mapped reads are used for further analysis. This is another method of filtering reads and ensures that reads that contain many errors are discarded. However, we have used different peak callers, such as `MACE` [34], for validation and testing purposes. Some of them require a certain read length in order to perform a bias compensation. Therefore, we have set the minimum read length to 13 for all peak callers in order to have comparable results.

Because I wanted to make this preprocessing pipeline as versatile as possible, I decided to implement all preprocessing options that can be used for any type of NGS reads, also including those for paired end reads. The preprocessing tool `Flexbar` [39] and a reimplementation [40] served as a basis for this implementation. I implemented most of the old functionality according to modern design principles, changed the multithreading architecture to enable parallel execution of the entire code, used parallel instructions whenever possible and added features which were necessary for this specific ChIP-nexus pipeline, such as random barcode processing and enhanced adapter trimming capabilities. The source code is available at `www.github.com\charite\Q` [41].

2.2.2 Implementation

The implementation of `Flexcat` was done in C++14. I created a produce-transform-consume library called ptc [42], which is used for enabling multithreading. This allows a very clean and maintainable design while delivering optimal multithreading performance. The Seqan library [43] is used for argument parsing and input/output of bam and sam files.

Most preprocessing tools that are known to me employ parallelization on a fairly low level, if any. This means, for example, that some loops in the function for adapter trimming and some loops in the function for quality trimming can be parallelized. This leads to a noticeable amount of code that is still executed serially, which is not optimal for scalability as we can see in figure 2.3. Unfortunately, during recent years, single thread performance has not improved much if we compare it to the gain in multithread performance. Therefore, employing efficient multithreading techniques for new software is essential.

As mentioned earlier, one important goal of multithreaded software development is parallelizing code on the highest level possible. We can have a closer look at this by writing the total processing time for a given piece of code with I_p parallelized and I_s non-parallelized instructions as a function of the number of threads N.

$$T(N) = \frac{I_p/N + I_s}{v} \tag{2.1}$$

In this equation, v is the execution time for a single instruction, and it can be seen as a constant in this context. Since we are actually more interested in the speedup that we get through parallel execution, we rewrite this equation and introduce the ratio $r = I_s/(I_s+I_p) \in [0,1]$, which is the proportion of instructions that are not executed in parallel.

$$S(N) = \frac{T(1)}{T(N)} = \frac{I_p + I_s}{I_p/N + I_s} = \frac{1}{\frac{1}{N}(1-r) + r} \tag{2.2}$$

This equation is called Amdahl's law. The graph shown in figure 2.3 shows the speedup that can be achieved for different r. The lock-free parallel implementation on the highest level used in `Flexcat` enables a speedup that is in the magnitude of the black curve.

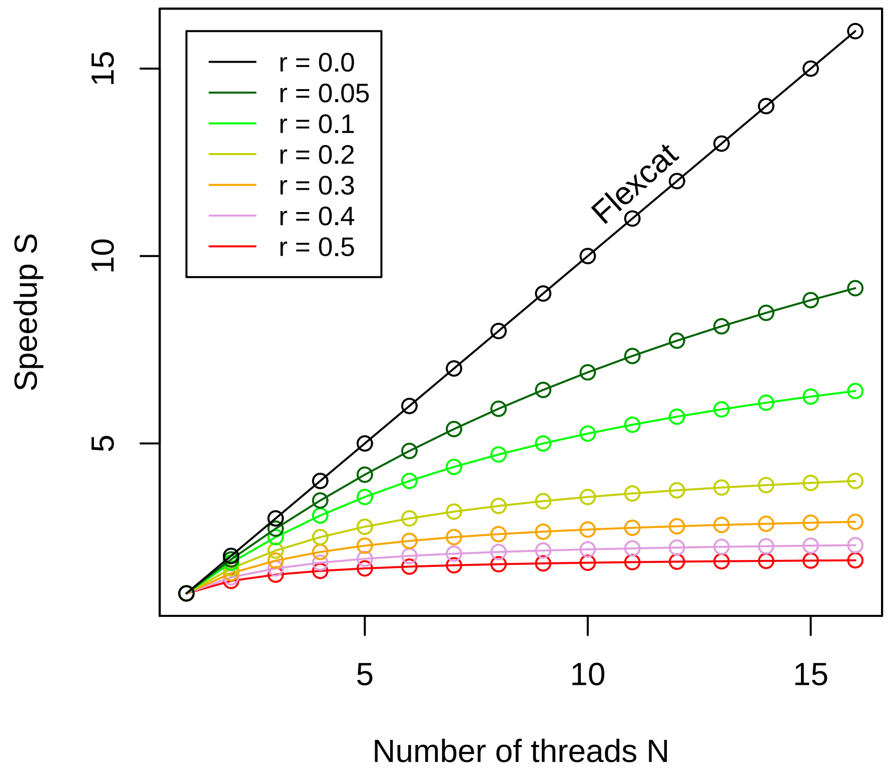


Figure 2.3: Amdahl's Law. The different speedups do not scale equally with the number of threads. Code that is only 50 % parallelized (red line) gets a very poor speedup when compared to the fully parallelized code (black line).

However, in real life, the speedup is usually not as good as Amdahl's law theoretically predicts. For `Flexcat`, the speedup when going from one to eight threads is about 5.5. Even though the hard disk I/O bandwidth of the testing machine was not yet exhausted when using only a small number of threads, we still see a decrease compared to the optimal speedup that Amdahl's law predicts for codes that run completely parallel. I assume, that the decreased speedup occurred because the test system's Intel i7 quad core CPU has only four hardware cores and also uses a shared L3 cache between all cores. When using eight threads, the I/O bandwidth of the testing machine becomes the bottleneck. Therefore, only a speedup of about 5.5 was achieved when using 8 cores.

The multithreading design in `Flexcat` consists of three main blocks, a producer thread called `ReadReader`, multiple processor threads called `ProcessingThreadX` and a consumer thread called `ReadWriter`. This design pattern is implemented using the ptc library [42], which is also open source. The program starts with the `ReadReader` reading data from the hard drive and then handing it over to the ptc library. The ptc library then hands this data asynchronously to the multiple transformation threads, which transform it and then give it back to the ptc library. The ptc library then asynchronously delivers this data to the `ReadWriter`, whose only responsibility is to write the received data from the ptc library to the hard drive.

A toy example shows how to use the ptc library to asynchronously calculate some Fibonacci numbers in listing 2.1. In this listing, only the relevant part of the code is shown; include lines as well as the definition of the fibonacci function are omitted.

Listing 2.1: Toy example shows how to use the ptc library to perform calculations of fibonacci numbers in parallel

```
int main()
{
    vector<int> vals = {10,20,30,40,42,20,30,40};
    auto numThreads = 10;
    auto myoptc = ptc::ordered_ptc(
        // produce
        [&vals]()->auto {
            if (vals.empty())
                return unique_ptr<int>();
            auto ret = make_unique<int>(vals.back());
            vals.pop_back();
            return ret;
        },
        // transform
        [](auto in)->auto {return make_unique<std::string>("fib(" +
            std::to_string(*in) + ") = " + std::to_string(fibonacci(*in)));},
        // consume
        [](auto in) {cout << *in << endl;},
        numThreads
    );
    myoptc->start();
    myoptc->wait();
    return;
}
```

As shown in listing 2.1, the process-transform-consume pipeline is assembled using a producer, a transformer and a consumer as building blocks. In listing 2.1 they are defined as lambda functions. There are some requirements for the interface of those building blocks by ptc library. The producer has to return a unique pointer to the data it generated. The transformer receives a unique pointer that points to data that it should transform, therefore it needs to have a unique pointer as argument. The very general term 'transform' is used here, because the the algorithm that the transformer runs is not limited by the ptc library. After transforming the data, the transformer returns a unique pointer that points to the transformed data. Finally, the consumer receives a unique pointer that points to the transformed data.

Using unique pointers has the advantage that it avoids multiple memory allocations. These memory allocations are usually very slow, since they are system calls which are not guaranteed to be lock-free either. Another feature that was implemented to reduce the number of memory allocations even more is the recycling of allocated memory. Since all the elements that are generated by the producer have the same type and are also about the same size, this is a good opportunity to use some kind of pool memory allocation. In the ptc library, memory recycling is implemented in a generic way, such that the producer can be optionally given a unique pointer that has been processed by the consumer. It is then the user-defined producer function's responsibility to use the memory pointed to by the given unique pointer to create a new producer element.

Listing 2.2 shows a toy example where memory recycling has been implemented using the ptc library. The producer initializes a string with Hello World, the transformer converts the string to upper case and the consumer prints it to the terminal. In contrast to the first toy example, the definition of the consumer lambda was moved out of the ptc declaration and into variable. This does not make a difference for runtime or functionality; however, for larger functions, it can increase the readability of the code. The function `producerReuse` now has an argument called `usedItem`. If there is an argument present in this function, the ptc library knows that memory recycling is possible and automatically enables it. The next interesting part is where the producer tests if `usedItem` is a valid pointer. This test is necessary because memory recycling cannot be used for the first element. If the argument `usedItem` of the function `producerReuse` is a valid pointer, the allocation of new memory is skipped. Skipping the allocation of new memory can be very beneficial for runtime, because memory allocation usually involves API calls to the operating system, which are very slow and usually not lock-free. Another small piece of code needed to be added to enable memory reuse. The consumer was modified so that it returns the consumed element. This way, the destructor of the unique pointer does not get called and the memory can get reused.

Listing 2.2: Toy example for using memory recycling with the ptc library

```
int main()
{
    unsigned count = 1000;
    auto producerReuse = [&count](std::unique_ptr<std::string>&& usedItem)->auto {
        if (count == 0)
            return unique_ptr<std::string>();
        --count;
        if (usedItem == nullptr)  // necessary for first element
            usedItem = make_unique<std::string>();
        usedItem->assign("Hello World");
        return std::move(usedItem);
    };
    auto myptcReuse = ptc::unordered_ptc(
        // produce
        producerReuse,
        // transform
        [](auto in)->auto {boost::to_upper(*in); return in;},
        // consume
        [](auto in) {cout << *in << endl; return in;},
        10 // number of threads
        );
    myptcReuse->start();
    myptcReuse->wait();
    return;
}
```

In the concrete case of `Flexcat` doing read processing, the producer returns an array of bases and the consumer also receives an array of bases of about the same size. Therefore, memory recycling could be efficiently implemented and the runtime significantly decreased.

As this example shows, using the ptc library is very simple and keeps the application code free from multithreading implementation details. The library internally employs atomic instructions that are thread-safe by nature. This eliminates the need for mutexes and improves scalability. The ptc library can be easily used to enhance existing tools that do all kinds of data transformation with multithreading capabilities. The data flow within the ptc library for the case of N processing threads is summarized in figure 2.4.

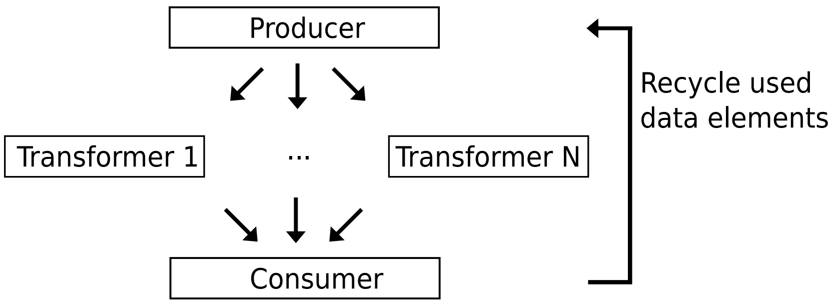


Figure 2.4: Data flow within the ptc library. Each rectangle represents a thread.

Another key part of an efficient implementation is using Single Input Multiple Data (SIMD) instructions whenever it makes sense. The part of `Flexcat` where they provide the greatest performance gain is when the sequences are compared to the adapter templates. The adapter templates are mapped to each read as good as possible, then the part of the read that has mapped to the adapter template is trimmed if the mapping satisfies the adapter trimming constraints such as error rate and minimum overlap. On a lower level, the adapter template is shifted base-wise along each read. For each shift position, the number of mismatches and the overlap is calculated. The following example with shift position $p = 0$ shows a mapping with 4 matches and 12 mismatches. The overlap can be calculated by adding matches and mismatches, the error rate can be calculated by dividing the number of mismatches by the overlap. In the example shown below, the overlap is 16 and the error-rate is 0.75.

```
CATTTAGAGAGAGAGAGAGCATTTAGATTGG    (read)
   |   |     | |
AGATCGGAAGAGCACA                   (adapter template)
```

The optimal shift position $p = 24$ is found by shifting the adapter template a few more bases to the right.

```
CATTTAGAGAGAGAGAGAGCATTTAGATTGG             (read)
                        |||| ||
                        AGATCGGAAGAGCACA    (adapter template)
```

Now the overlap is 7 and the error rate is 0.14. If the adapter trimming settings were set to an error-rate larger than 0.14 and a minimum overlap smaller than 7, the read would be trimmed like this

```
CATTTAGAGAGAGAGAGAGCATTT           (read)
```

It is evident that finding the optimal mapping requires a lot of base-compare operations. If the read length is l_r and the adapter template length is l_a, then the number of required base comparisons is $l_r l_a$. Starting with shift position $p = 0$, the number of comparisons is l_a for each shift position until the adapter template is sticking out at the end of the read. For example, shift position $p = 24$ only requires 7 comparison operations. However, the operations that are saved as the adapter template has an overhang to the right side and are used for overhangs to the left side, for example $p = -1$

```
 CATTTAGAGAGAGAGAGAGCATTTAGATTGG   (read)
  |   |     | |
AGATCGGAAGAGCACA                   (adapter template)
```

We can imagine that shift position $p = -1$ can fill up shift position $p = 30$

```
CATTTAGAGAGAGAGAGAGCATTTAGATTGG                 (read)

                              AGATCGGAAGAGCACA  (adapter template)
```

Therefore, the number of required comparisons is $l_r l_a$ if negative overhang is allowed. If we look how many instructions that would be for a file containing 10 million reads of length 40, the resulting number is already pretty big, $10^7 \cdot 40 \cdot 16 = 6.4 \cdot 10^9$, which is almost 10 billion. If more adapter templates are used, the number easily increases to about 50 billion comparisons. If the CPU could do one instruction per cycle, this would not be too bad, because modern CPUs usually have a clock rate over 1 billion per second. However, there is one thing that we did not take into account so far, namely cache misses. They usually occur if a conditional branch happens in the code and the branch prediction mechanism of the CPU cannot predict it correctly. If the read adapter mapping algorithm is implemented with straight forward base comparisons, cache misses occur approximately with every second comparison, because the mismatch counter needs to be increased depending on the result of the comparison. The penalty for a cache miss is roughly 15 cycles which increases the number of required cycles for the example above to 375 billion. This would take $6 : 15$ minutes on a hypothetical 1 GHz CPU that does one instruction per cycle.

This situation can be significantly improved by using SIMD instructions. Not only less instructions are required, but also less cache misses occur. Listing 2.3 shows how this is implemented. The functions that perform the actual matching between read sequence and adapter template are called `apply(...)`. They are located inside a templated struct called `compareAdapter` with the number of bases to compare as template parameter to allow for loop unrolling using template metaprogramming techniques. In listing 2.3, only the template that checks 32 bases is shown as an example. The function `apply(...)` is adapted from `AdapterRemoval 2` [44] and extended to use more and wider SIMD instructions. `AdapterRemoval 2` uses only SIMD instructions from the Streaming SIMD Extensions 2 (SSE2) standard, which are 128 bits wide. In this work, a speedup was achieved by using Streaming SIMD Extensions 4 (SSE4) instructions which are 256 bit wide and a SSE4 instruction that is optimized for counting set bits, called `popcnt`. SSE4 instructions are available on all Intel Haswell and AMD Barcelona or newer devices as well as VIA Nano-based processors. The algorithm shown here is not only fast because of the SIMD instructions used, but also because it avoids conditional jumps and therefore causes less cache misses. We can roughly estimate the runtime cost for this algorithm by assuming that every instruction takes only one cycle and that no cache misses occur. Also as before, we only count instructions that do the actual comparison; the code for increasing pointers, variable initialization, etc. is neglected. Six instructions needed are needed for every 32 bases, two for comparison `_mm256_sub_epi8` and `_mm256_cmpeq_epi8`, and four `_mm_popcnt_u64` instructions to count the number of matches. Taking the same example as above with 50 billion comparisons, this totals in $50 \cdot 6/32 = 9.375$ billion instructions. For the same hypothetical 1 GHz CPU as before that does one instruction per cycle, this would take only about 9 seconds which is a large difference compared to $6 : 15$ minutes. For this example, the calculated speedup factor is roughly 41. Because of memory bandwidth limitations and other factors, the actual speedup is not as large as in this calculation example, but depending on the input data and processor type, it is still a factor of 10 − 30 and therefore very considerable. The actual algorithm used in `Flexcat` is a bit more complex than the one shown in listing 2.3, because it also takes into account ambiguous bases and counts them separately.

Listing 2.3: Algorithm for adapter matching

```
inline size_t popcnt256(__m256i value) noexcept
{
  return _mm_popcnt_u64(value.m256i_u64[0]) +
    _mm_popcnt_u64(value.m256i_u64[1]) +
    _mm_popcnt_u64(value.m256i_u64[2]) +
    _mm_popcnt_u64(value.m256i_u64[3]);
}

const __m256i ZERO_256 = _mm256_set1_epi8(0);
const __m256i ONE_256 = _mm256_set1_epi8(1);


template <>
struct compareAdapter<32>
{
  template <typename TReadIterator, typename TAdapterIterator, typename TCounter>
  inline static void apply(TReadIterator& readIt, TAdapterIterator& adapterIt,
    TCounter& matches, TCounter& ambiguous) noexcept
  {
    const __m256i read = _mm256_loadu_si256(reinterpret_cast<const __m256i*>(&(*readIt)));
    const __m256i adapter = _mm256_loadu_si256(reinterpret_cast<const __m256i*>(&(*adapterIt)));

    const __m256i matchesMask = _mm256_sub_epi8(ZERO_256, _mm256_cmpeq_epi8(read, adapter));

    matches += popcnt256(_mm256_and_si256(matchesMask, ONE_256));
    readIt += 32;
    adapterIt += 32;
  }
};
```

2.2.3 Feature description

The main functionality of the preprocessing software developed in the scope of this thesis comprises adapter trimming, fixed and random barcode filtering, quality trimming and length filtering. Barcode demultiplexing can be activated by specifying a file that contains the barcodes in FASTA format, for example, by adding `-b barcodes.fasta` to the command line. `Flexcat` then creates a separate output file for each barcode. The suffix of the output files can be specified in the id line of the barcode FASTA file. In order to process fixed barcodes which are used in the ChIP-nexus protocol, one has to specify a barcode file that contains only one entry with the appropriate barcode sequence. One example for a possible barcode file is shown later in section 3.1.

The adapter trimming functionality of `Flexcat` is very versatile. It is possible to use both 3' and 5' adapters and it is also possible to specify if they adapters are anchored to their corresponding end of the read. For reads that a mapped to the forward strand, a 3' adapter usually starts after the protein-binding site and a 5' adapter before the protein-binding location. Therefore, it is necessary to distinguish during adapter trimming on which side of the adapter remaining DNA sequence should be removed. For example, if the 3' adapter template matched to a sequence within the read, all the bases that follow this matching sequence in 3' direction are probably adapter sequence as well and should therefore be removed. This specific case is illustrated in the top left corner of figure 2.5. The 5' adapter option has been added because it has been observed that the 5'-end of a read can have a bias [45] or even adapter contamination. Some researchers suspect that the 5'-end adapter contamination might be due to adapter dimer PCR artifacts; however at this point there is no published information that analyzes this phenomenon. Specifying an adapter sequence as anchored can be useful for 5' adapters that usually do not appear in a degraded form. For 3' adapters, this option is rarely used, because the number of bases before the adapter usually varies for every read.

As mentioned before, the base quality of sequenced reads usually deteriorates towards the end. This is because the clones inside a cluster of the flow cell become out of sync. Therefore, it is common to perform quality trimming, which removes bases from the sequence whose confidence level is below a certain threshold. `Flexcat` implements three different methods for quality trimming, which can be selected by the `-m` option. The option `-m WIN` uses a window that moves from 5' to 3'. For each window position, a quality score is calculated. Reads are trimmed starting from the first window whose quality score is below a user-defined threshold. The option `-m TAIL` trims the read at the position where the quality score is above or equal to the threshold for the first time, starting from the 5'-end. This option is most commonly used. The option `-m BWA` uses the method introduced by BWA [46]. The quality for each base is usually stored in a file that has a FASTQ format. FASTQ files are similar to FASTA files but contain two extra lines for each read. A read entry might look like this:

```
@SRR1175698.71 SEB9BZKS1:84:D1F78ACXX:4:1101:9390:2122 length=51
NAAAAACATGTCACAAATTCCCACTAACAATGCGGGCTTCCTCACGTCCTG
+SRR1175698.71 SEB9BZKS1:84:D1F78ACXX:4:1101:9390:2122 length=51
#4BDFFFFHHHHHJJJJJJJJJJJJJJJJIJJJJJIJJJJJIHIIIJJI
```

The first two lines are identical to the lines in a FASTA file and the third line is another description field which is not used most of the time. The fourth line contains logarithmic quality scores also called phred score, which is one byte for each base. It is defined as

$$Q = 10 \times log_{10}(P) \tag{2.3}$$

or

$$P = 10^{\frac{-Q}{10}} \tag{2.4}$$

for example the quality score for a certain base is 20, the error probability for this base is 1:100. Makers of sequencing machines use different ASCII encoding for the phred score. The most commonly used codings are the Sanger and Illumina 1.8+ standards, where the numbers 0 – 40 or 0 – 41 are mapped to the ASCII characters !..J like this

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJ
|        |                              |
0........9.............................40  (Sanger)
|        |                               |
0........9..............................41 (Illumina 1.8+)
```

`Flexcat` automatically detects the coding of the FASTQ file.

Trimming of `n` nucleotides can be activated by adding `-tl n` for the left side or `-tr n` for the right side to the command line. The additional option `-tt` writes the trimmed sequence of each read into the corresponding read id. For processing ChIP-nexus reads, we use `-tl 5 -tt` to remove the random barcode from the read sequence and store it for later processing in the id field. This allows PCR duplicates, which are reads that originate from the same DNA fragment, to be distinguished from real events in later steps of the analysis.

Table 2.1 shows an overview of Flexcat features compared to other preprocessing tools. The data for this table was adapted from Jiang et al. [38].

	Adapter Trimming					General Trimming			Misc						
Tool	3'	5'	m	a	o	left	right	tag	Nxs	MT	Q	X	SE	PE	Ns
Flexcat	+	+	+	+	+	+	+	+	-	+	+	+	+	+	+
Cutadapt [47]	+	+	+	+	-	+	-	-	-	-	+	+	+	+	-
Flexbar [39]	+	-	-	+	-	+	-	-	-	(+)	+	+	+	+	-
Skewer [38]	+	+	+	+	-	+	-	-	-	-	+	+	+	+	-
Scythe [48]	+	-	-	+	-	-	-	-	-	-	-	-	+	-	-
AR2 [44,49]	+	+	-	-	-	-	-	-	-	-	+	-	+	+	+

Table 2.1: m: multiple adapters, a: anchored adapters, o: allow optional adapter overlap at the opposite end, NXs: ChIP-nexus preprocessing, MT: multi-threading, Q: quality trimming, X: demultiplexing, SE: single-end, PE: paired-end, Ns: filter out reads with many unknown bases

2.2.4 Command line options

General Options

-o [prefix]	The output file will be `[prefix].input_file_ending`
-tnum [n]	n processing threads are used. There are two additional threads for io-processing. The default value is the number of hardware threads in the systems.
-r [n]	each processing thread processes n reads at a time. The default value of 1000 is suitable for most applications.
-st	write a statistics file at the end of the preprocessing. The name of the statistics file will be `[input_file_prefix]_statistics.txt`
-ss	show processing speed in the terminal window.
-od	keep the reads in the same order. This option requires additional synchronization between the processing threads and is therefore somewhat slower.
-fr n	process only the first n reads. This option can be useful for testing, where processing the whole file would take too long.
-ni	do not print parameter overview to the terminal.
-ml [n]	required minimum length after the entire preprocessing.

General Trimming

-tl [n]	n bases of the 5'-end are removed. This option can be useful to remove fixed barcodes and store them in the id line of the read (needs -tt option).
-tr [n]	n bases of the 3'-end are removed.
-tt	write removed bases to the id line of the read.

Adapter Trimming

-a [adapter-file]	specifies an adapter-file in the format explained in section 3.1.
-er [x]	maximum allowed error-rate.
-ol [n]	minimum required overlap of adapter and read.
-oh [n]	allowed overhang of the adapter at the opposite end of the read.
-times [n]	adapter trimming is done [n] times for each read. This is useful if reads can contain multiple adapters.
-nler	use non-linear error-rate.

Quality Trimming

-q [x]	quality threshold
-m [x]	method used for quality trimming, x can be TAIL, BWA or WIN
-qml [l]	minimum read length after quality trimming
-t	tag reads that were quality trimmed. The tag is added to the id line of the read.

Demultiplexing

-b [barcode-file]	specifies a file in FASTA format that contains barcodes.
-app	approximate barcode matching, allows one mismatch per barcode.
-ex	exclude reads with no matching barcode from further processing.
-hc	hard clip mode, removes the barcode from the input sequence whether is was matched or not

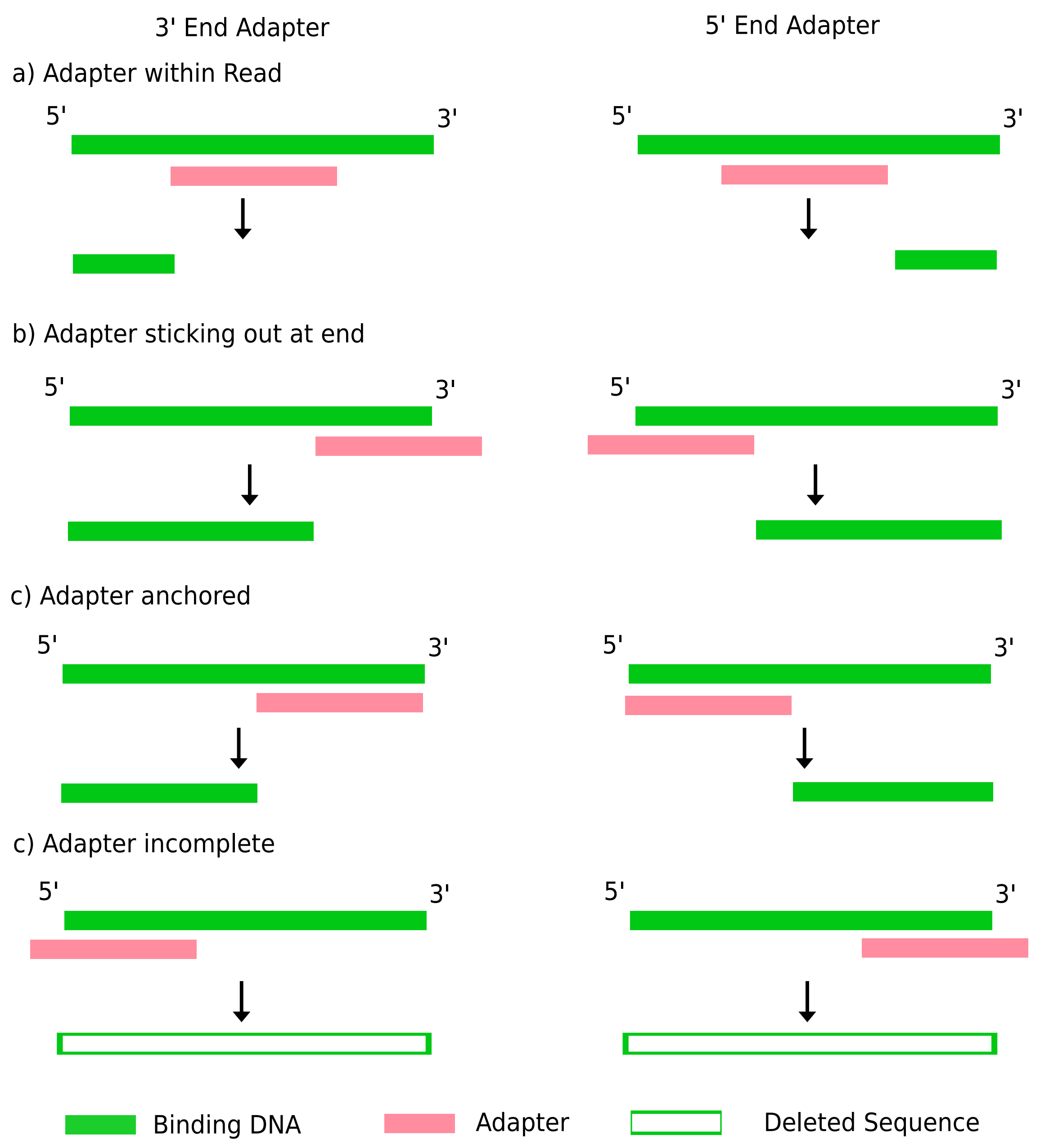


Figure 2.5: Matching of adapter sequences during the process of adapter trimming

2.3 Nexcat: A tool for filtering random barcodes

After mapping, there are some reads with identical random barcodes that are mapped to the same position. These reads must be filtered out except one, because they are caused by PCR amplification, which makes it difficult to use them to quantify the protein-DNA interaction. Since during peak calling, only the 5'-end of the read is used, it does not matter which of the identically mapped reads is kept.

The algorithm that does this filtering generates a unique key for each read, which consists of the mapping position and the random barcode. These keys are then inserted into a map. For every read, it is checked if the map already contains the same key, if so, the read is discarded. This is the part of the preprocessing where PCR duplicates are removed.

`Nexcat` also generates statistical information about the experiment, which can be used for quality control. For each read-length, it calculates the number of duplicate reads regardless of the random-barcode (total duplicates) and duplicate reads with the same random barcode (real duplicates). This information can then be used to generate a duplication rate plot that shows the total and real duplication rate over the read-length. The experiment setup should be optimized in such a way that there are as few real duplicates as possible, because we have to pay to sequence them, but do not use them in the final analysis. Until now, there is no software available to perform this task, except the R-scripts supplied by the inventors of the ChIP-nexus method. However, these R-scripts have some limitations with regard to processing accuracy, speed and memory consumption. A detailed comparison of the newly developed software to the existing R-scripts is shown in the following chapter. The source code of this tool is available in an open source repository [50].

2.4 Verification of correctness

All the newly developed tools have been unit-tested. The unit tests are supplied with the software and are located inside the `test_*.cpp` files. They can be compiled and executed independently. Besides unit testing, artificial input data has also been used for verification purposes. The newly developed tool `ReadSim`, which is described in more detail in the following chapter, was used to generate artificial data. For example, generating artificial data, without barcodes, with error-rate 0, without PCR artifacts and without quality deterioration allowed to test all the features regarding adapter trimming by comparing feeding the generated input data into the preprocessing pipeline and then comparing it to the optimal preprocessed output file, which is also generated by `ReadSim`. Another indirect marker for the correctness of preprocessing is the number of mappable reads, because random errors always destroy similarity between reads and the reference genome and thus make less reads mappable. As shown in the following chapter, the number of mappable reads could be markedly increased by adding new functions such as improved fixed barcode processing and improved adapter trimming.

2.5 Assembling the preprocessing pipeline

2.5.1 Overview

The different steps of preprocessing pipeline have been divided into two different binaries, `Flexcat` and `Nexcat`. This choice was reasonable, because the mapping, which is done by external tools, should be callable in a modular way. Splitting the preprocessing into two binaries also has the advantage that `Flexcat` and `Nexcat` can be combined with other tools. Therefore, I have implemented all the preprocessing steps that occur before mapping into `Flexcat` and the remaining preprocessing step, which is only necessary for ChIP-nexus data, into `Nexcat`.

2.5.2 Mapping preprocessed ChIP-nexus reads

After mapping the preprocessed FASTQ file using `Bowtie`, the resulting sam-file contains one line of ASCII text for each read. An example is given here:

```
SRR1175698.30:TL:NCTTC:AdapterRemoved 0 chr2L 21725514 255 35M * 0 0
TAAGAAGATGCGTAACGGCCATACATTGTTTTTGC HHHBGHIIIIIIHIGIIIIIIIIIIIIIHIIIIII
XA:i:0 MD:Z:35 NM:i:0
```

The first field is the read id, which was originally `SRR1175698.30`; however, during the preprocessing additional information that `Flexcat` generated were appended here, namely the random barcode `NCTTC` and a notification that an adapter has been removed. Additionally, in the DNA sequence shown, the fixed barcode has also been removed by `Flexcat`.

`Nexcat`, which is the next tool in downstream analysis, extracts the random barcode from the id field of the SAM file and uses it for further processing as described above.

2.5.3 Convenience wrapper script

In order to make the use of this preprocessing pipeline more convenient, I have created a python script, called `preprocess.py`, that wraps the calls to the two different binaries and the mapper. The advantage of using Python here instead of C++ is that it can be easily customized without the need for recompilation.

Editing `preprocess.py` can be useful if one wants to add support for another mapper besides `Bowtie` or if one wants to change the default parameters.

2.5.4 Command line options

The most common settings can be specified via command line parameter to `preprocess.py`.

General Options

[first filename]	input file for preprocessing, can be FASTA or zipped FASTA format.
[second filename]	reference genome used for mapping.
–verbose	print status information to the terminal.
–overwrite	overwrite existing files.
–clean	delete temporary files.
–tnum [n]	number of threads used.
–bowtie_location [path]	this is only necessary for `Windows` or if `Bowtie` is not present in the path.
-[additional options]	any additional options will be directly passed to `Flexcat`

An example call could look like this:

```
preprocess.py ~\data\raw_fastq\SRR1175698.fastq.gz ~\data\ref\dm3\dm3.fa --verbose
--output_dir ~\data\preprocessed\SRR1175698 --tnum 8
```

In this example, the file `\data\raw_fastq\SRR1175698.fastq.gz` contains the raw reads from the sequencer and the file `\data\ref\dm3\dm3.fa` contains the sequence of the reference genome.

3

Results: Comparison to existing tools and application to experimental data

Validation for the newly developed tools has been done in two ways. One way was using raw ChIP-nexus data from [24] and preprocessing with the newly developed pipeline. The other way was to use artificial reads generated by a read simulator. Using artificial reads also made it possible to calculate several quality metrics which are useful when comparing different tools. In order to demonstrate possible applications of the results, I have done a motif analysis and extracted information relevant to medicine and biology out of the raw data.

3.1 Comparison using real data

The first step in preprocessing is demultiplexing. This means that every read that does not have a matching fixed barcode is filtered out. The fixed barcode used for the data sets from [24] is `CTGA`. The barcode definition file used in this case, which is also used as a default by the `preprocess.py` script, is shown in listing 3.1.

Listing 3.1: barcodes.fa

```
>matched_barcode
CTGA
```

By default, the `-app` option of `Flexcat` that allows one mismatch when matching fixed barcodes is enabled in the `preprocess.py` script. As is shown below in figure 3.4, allowing a mismatch for fixed barcode matching increases the number of unambiguously mappable reads significantly.

The adapter file that has been used is shown in listing 3.2.

Listing 3.2: adapters.fa

```
>adapter1.2:3':
AGATCGGAAGAGCACACGTCTGGATCCACGACGCTCTTCC
>adapter1.2:3':
AGATCGGAAGAGCACA
>adapter1.2:5':
TGTGCTCTTCCGATCT
```

The first adapter in this file is the Illumina read primer as shown in figure 1.14. The second adapter is just a fragment of the first adapter. It is not immediately obvious why this one is useful. However, during the wet lab phase of the experiment, the different preparation steps cannot be executed as precisely as in a digital world; inevitably there are some errors. One of these errors can be an adapter that has a degraded 3'-end and thus is not matched to the original adapter sequence if the sequence as a whole is used as a template. To account for these situations, a 5'-end fragment of the original adapter is added to the adapter trimming template list. The motivation for the 3rd adapter template which is used, is also not very obvious. It turned out, that a noticeable amount of sequences contain part of a Illumina read primer at the 5'-end. It is as if the red box at the bottom of figure 1.14 were shifted a bit to the left. We improved the mappability of the data by removing those fragments using 5'-end adapter filtering. For mapping, the default settings as described in section 2.5.4 were used.

Figure 3.1 shows a comparison between the mapped reads resulting from this preprocessing pipeline with the mapped reads that were reported in [24]. There is roughly a 8 percent increase in the number of mapped reads across all data sets. The command lines that have been used to generate the results are shown in listing 3.3.

Listing 3.3: command lines

```
flexcat filename.fastq.gz -o filename_processed.fastq -a ../adapters.fa -b \
  ../barcodes.fa -tl 5 -tt -app -er 0.2 -ol 4 -ml 13 -r 1000
bowtie -S --chunkmbs 512 -k 1 -m 1 -v 2 --strata --best \
  /media/sf_data2/genomes/dm3 filename_processed_matched_barcode.fastq \
  out.sam
nexcat out.sam
```

For the results shown as red bards, the argument `-q 10` has been added to the command line. It is visible that the number of mappable reads was only marginally increased when enabling quality trimming. Because of the additional runtime cost and limited benefit, I decided not to use it for the experiments carried out in the scope of this thesis. However for data sets that contain more reads with poor quality, enabling quality trimming may be worth the additional runtime. The in-depth analysis shown in section 3.1.2 shows where the main improvement with regard to the number of mappable reads comes from.

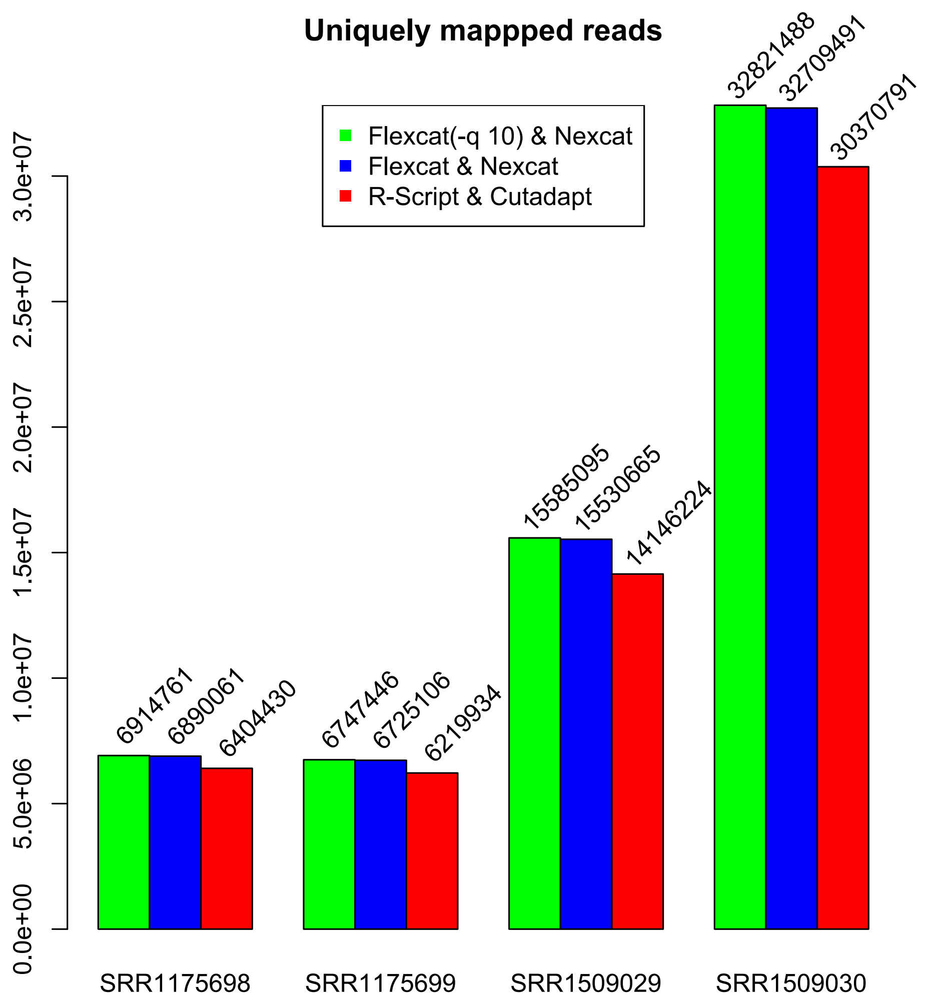


Figure 3.1: Number of unambiguously mapped reads after preprocessing for different data sets

The duplication rate plot shown in figure 3.2 generated from output data of `Nexcat` makes it possible to get a quick overview of the average peak coverage and number of PCR artifacts. The x-axis represents the number of reads mapped at the same position in the reference genome. The y-axis shows how many reads for each duplication rate exist. The solid line shows the number of reads for no matter what the random barcode is, while the dotted line only counts those reads that have a different random barcode. Therefore, the difference between solid and the dotted line quantifies the PCR artifacts. I further discuss information that this graph gives in section 4. As described earlier in section 2.3, the duplication rate plot can be useful for quality control of the experiment. Duplication rate plots of more data sets are shown in appendix C.

Before proceeding with the downstream analysis, it is recommended to filter out chromosomes that contain unlocalizable sequences, such as `chrU` and `chrXhet`. This is achieved by invoking `Nexcat` with the filter-chromosomes option together with an appropriate regular expression: `nexcat -fc '(.*)[H|U]+(.*)'`. During this analysis, chromosomes have not

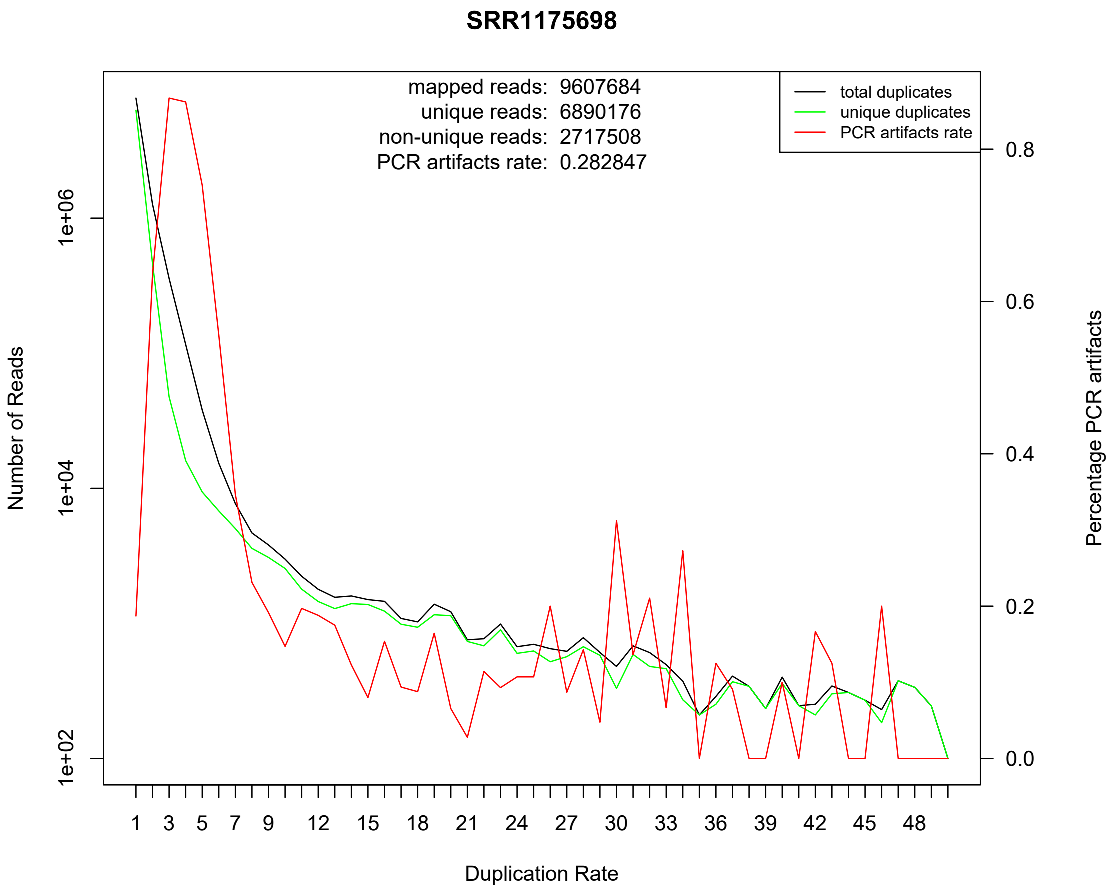


Figure 3.2: Duplication rate plot of data set SRR1175698 (*Drosophila Melanogaster*, transcription factor Dorsal). Unique reads are the area under the curve of the green line. Non-unique reads are the area between the black and green line. The total number of mapped reads is the area under the black curve (here 9354144). The total PCR artifacts rate can be calculated as the non-unique number of reads divided by the total number of reads (here 0.2825)

been filtered out during preprocessing in order to better compare the results to the original pipeline based on R-scripts where they also have not been filtered out. It is also possible to filter out unwanted chromosomes during a later stage of the analysis, for example during peak calling.

3.1.1 Comparison using simulated data

Until now, there has been no single tool available that supports all the operations necessary during preprocessing of ChIP-Nexus data. Therefore, I chose the original R-Script-based preprocessing pipeline that was developed by the authors of the ChIP-nexus method as a reference point for this benchmark. The original pipeline uses a variety of different tools that are executed sequentially. In order to test a wider range of existing tools, I have extended the R-scripts provided by He et al. [24] to optionally use different tools for adapter trimming. The adapter trimmers that I have selected for comparison are `Skewer` [38], `AdapterRemoval 2` [49], `Cutadapt` [47] and `Flexcat`. This selection is based on [44], where it was shown that `AdapterRemoval 2` and `Skewer` were the two fastest adapter trimmers available and also produce valid results.

The parameters used to assess the quality of preprocessing are the Matthews Correlation Coefficient (MCC) and receiver operating characteristics (ROC) curves. ROC curves show the performance of binary classifiers while the discrimination threshold, here error rate, is varied. The closer the curve is to the upper left corner of the plot window, the better the discrimination capability. A curve that goes in a straight line from the lower left to the upper right corner means that discrimination between true and false is happening only randomly. In order to generate the statistical metrics necessary for generation of MCC and ROC curves, false positives (FP), true positives (TP), false negatives (FN), true negatives (TN) and MCC need to be calculated. The MCC is a quality measure which has been used to compare adapter trimmers before [38]. It takes into account FP, FN, TP and TN and combines them into a single number.

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN))}} \tag{3.1}$$

The false positive rate (FPR) and True Positive Rate (TPR) are calculated as follows

$$FPR = \frac{FP}{FP + TN} = (1 - \text{specificity}) \tag{3.2}$$

$$TPR = \frac{TP}{TP + FN} = \text{sensitivity} \tag{3.3}$$

Calculating these values from experiments with real data is difficult, because no reference (perfectly trimmed file) is available. Therefore, synthetic data is usually used for benchmarking and validating NGS preprocessing software. There are already a number of read simulators for NGS data available [51–55]; however, none of them is suitable for generating ChIP-nexus data, as they cannot generate barcodes and have limited capabilities for simulating adapters. Therefore, I have programmed a new read simulator called `ReadSim` that is particularly suitable for generating ChIP-nexus data.
`ReadSim` takes a reference genome and randomly picks DNA fragments from it, which are then used to generate the synthetic reads. The features of `ReadSim` include, adding random and fixed barcodes, simulating errors and quality deterioration towards the end of reads and adding PCR artifacts. The command line options for `ReadSim` are as follows

-g : reference genome

-o : output filename

-a : FASTA file that contains adapter sequences

-fb [**b**]: fixed barcode sequence

-rb [**n**]: number of bases for random barcode

-n [**n**]: number of reads to generate

-er [**n**]: error rate

-q : simulate quality deterioration

-p : add PCR artifacts

The command line that has been used to generate data input for figure 3.3 looks like this:

```
ReadSim -q -p -g D:\data2\genomes\dm3.fa -a D:\git\nexus-tools\data\adapters.fa
-fb CTGA -n 1000000 -rb 5 -o D:\data2\fastq\readsim_out -er 0.1
```

Simulation of errors is done using the error rate specified via command line options. For every base, the error probability is calculated using a random number generator; if an error occurs, the base is reset to a random base. This error injection is applied to the whole read, including barcodes and adapters. The algorithm that is used for simulating quality deterioration is very simple and calculates the starting point for quality deterioration within the read by choosing a random position between the middle and the end of the read. Starting from this position, the quality is decreased by 5 with each base until the quality reaches 5, after that it is decreased by 2 with each base until it reaches 0 or the read ends. After calculating the quality deterioration, additional errors are inserted according to the error probability given by equation 2.4. Simulation of PCR artifacts is implemented by cloning every tenth generated read. `ReadSim` generates two output files, one with the name as specified with the `-o` option. This file contains the raw simulated reads and can be used as input for preprocessing software. The second file has the same name, but an additional `_preprocessed` suffix. This file is a gold file (theoretically optimally trimmed) that can be used to measure the performance of the preprocessing pipeline. `ReadSim` can assist in calculating the performance of any preprocessing pipeline by calculating the statistical values described above. The output might look similar to the following

```
Number of reads in reference file     : 100000
Number of reads in preprocessed file : 85681
---------- per read ----------
Number of correctly trimmed reads     : 56219 (56.219%)
Number of over trimmed reads          : 32714 (32.714%)
Number of under trimmed reads         : 11067 (11.067%)
Sensitivity_r                         : 0.810113
Specificity_r                         : 0.21583
PPV_r                                 : 0.590712
NPV_r                                 : 0.448607
MCC_r                                 : 0.0319383
---------- per base ----------
Number of correctly trimmed bases     : 56219 (56.2128%)
Number of over trimmed bases          : 422457 (16.2425%)
Number of under trimmed bases         : 52772 (2.02896%)
Sensitivity_b                         : 0.956012
Specificity_b                         : 0.427252
PPV_b                                 : 0.730812
NPV_b                                 : 0.856564
MCC_b                                 : 0.474468
```

The ROC curves in figure 3.3 show that the optimal error rate with regard to the maximum number of mapped reads and best MCC value is not identical.

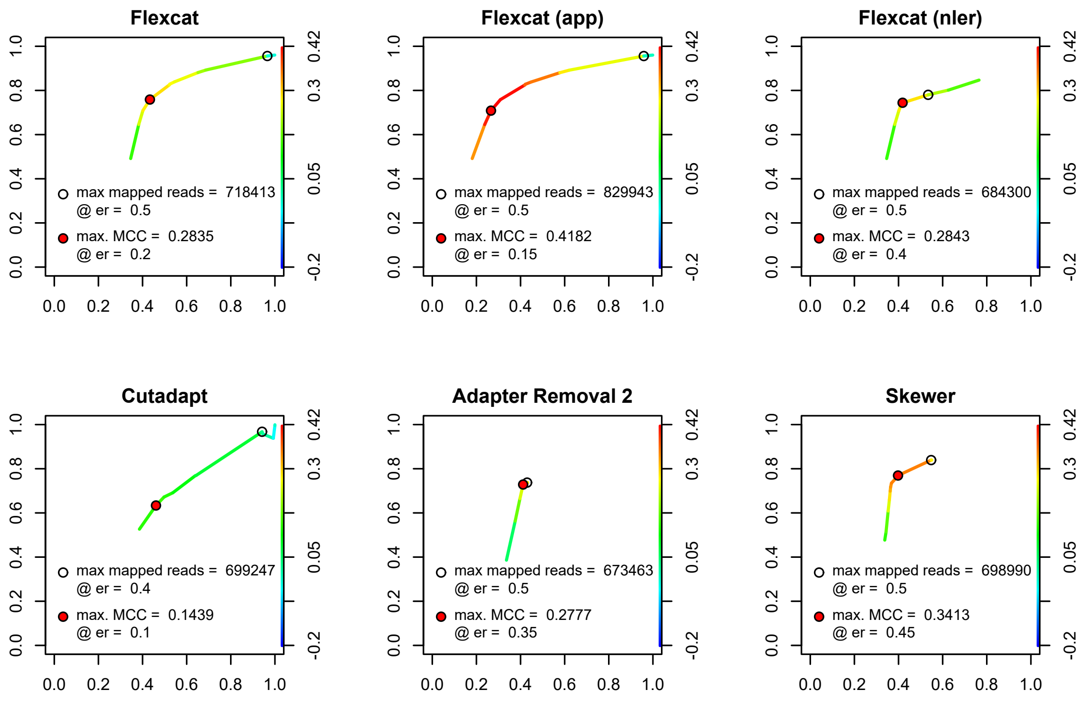


Figure 3.3: ROC Curve for different adapter trimmers using data set SRR1175698 (*Drosophila Melanogaster*, transcription factor Dorsal). The x-axis of each subgraph shows the FPR, the left y-axis shows TPR, the right y-axis shows the MCC.

The *MCC* is usually optimal for smaller error rates, because during its calculation, overtrimming and undertrimming are punished equally whereas for mappability; overtrimming is usually more unfavorable than undertrimming, because short reads can be mapped easier than reads that still contain adapter sequences. Therefore, the error rate used for experiments should be closer to the maximum number of mapped reads than to the *MCC*. In the experiments done with simulated reads, the maximum number of mapped reads was best for an error rate between 0.4 and 0.5. However, these numbers are specific to input data, in this case parameters of `ReadSim`. Therefore, the optimal error rate needs to be determined for every data set separately. Selecting parameters for preprocessing is a multidimensional optimization problem which is usually solved by using experience as well as by trial and error. For this benchmark, I used the same settings as used in [24]. Since `Flexcat` offers some options that other adapter trimmers do not have, I used three different settings of `Flexcat` to explore the effect of this new functionality. The first uses the default setting, the second one additionally uses the `-app` option, the third one additionally uses the `-nler` option. Figure 3.3 shows that `Flexcat` with the `-app` option can achieve the highest number of unambiguously mappable reads. The `-nler` option of `Flexcat` (see section 2.2.4), which reduces the number of possible mismatches for short overlaps during adapter trimming, did not show a benefit and therefore was not used during analysis of our data sets. However, we can see that the shape of the ROC curve of `Flexcat` with the `-nler` option enabled resembles the ROC curves of the other adapter trimmers that use a non-linear error rate for adapter trimming by default.

3.1.2 Benchmarks

Besides trimming quality, processing speed and memory footprint are two other important factors when preprocessing NGS data. Therefore I have also compared the tools mentioned above in these categories. The test platform was an Amazon c3.4xlarge instance with 8 Intel Xeon E5-2680 cores (16 virtual cores) and 30 GB RAM. Figure 3.4 shows the result comparing the same set of preprocessing tools as in section 3.1.1; however, this time real data, namely data set SRR1175698 (*Drosophila Melanogaster*) was used. The comparison was done in three categories, number of mapped reads, processing speed and used RAM. The parameters for preprocessing are the same as in [24]. To better explore the impact of the different newly developed preprocessing pipeline, `Flexcat` was used in three different settings: embedded into the original preprocessing scripts by He et al. [24] and stand alone with `Nexcat` with two different settings regarding fixed barcode detection. The first setting `-app` allows at maximum one error when matching the fixed barcodes, whereas the default setting does not allow any mismatch during fixed barcode detection. The original scripts written in R by He et al. [24] do not allow any mismatch when matching the fixed barcode; they allow up to one ambiguous base to be present within the fixed barcode. Figure 3.4 shows that the number of unambiguously mappable reads was also significantly increased for real data by allowing up to one mismatch during fixed barcode matching. Another aspect that was analyzed during these benchmarks is the influence of quality trimming on the number of mapped reads. All the different preprocessing setups were run with quality trimming enabled (blue bars) and also with quality trimming disabled (green bars). The settings used for quality trimming were such that all the bases with a phread score below 10 were trimmed; for `Flexcat` and most other adapter trimmers, this option is `-q 10`. The result of this comparison was that most preprocessing setups could map marginally more reads with quality trimming enabled, as shown in figure 3.1.

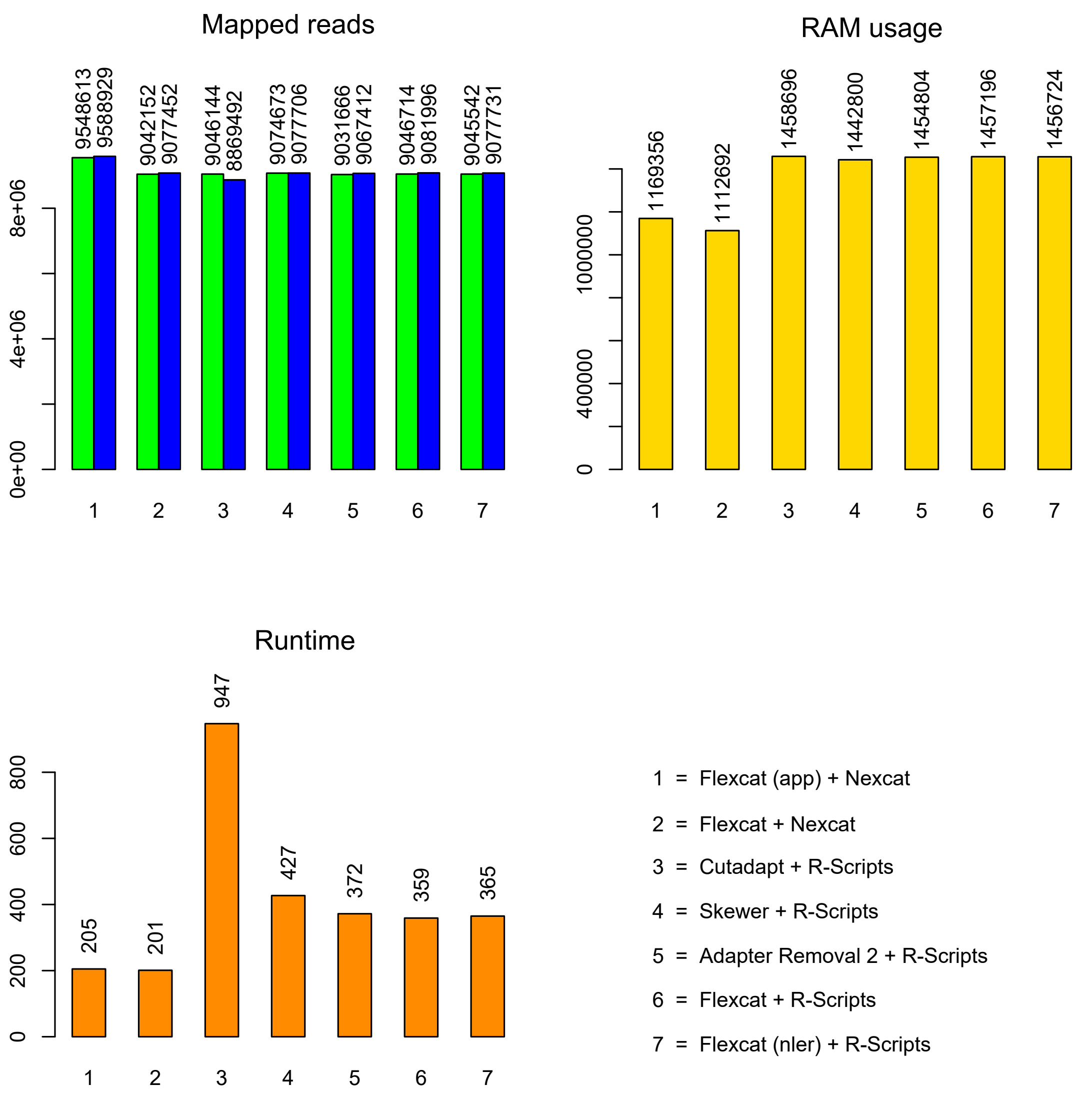


Figure 3.4: Comparison of runtime (seconds), RAM usage (kilobytes) and number of mapped reads for data set SRR1175698 (*Drosophila Melanogaster*, transcription factor Dorsal). The green bars show the number of mapped reads without quality trimming, the blue bars with quality trimming.

RAM consumption was smallest when using the newly developed preprocessing pipeline. This improvement has been achieved by avoiding R-Scripts which tend to have more overhead when loading large files into memory. It is generally desirable to avoid loading a file completely into memory, because this limits the size of processable input files to the size of the RAM available. However, for processing the random barcodes, keeping a list of at least some key elements of every read in memory cannot be avoided. Therefore, `Nexcat`, which does the random barcode filtering, has a higher RAM consumption than `Flexcat`, but is still more RAM and speed efficient than the R-scripts that were used before.

The runtime has been markedly reduced with `Flexcat` and `Nexcat` compared to the original solution by a factor of almost five. Figure 3.4 shows that the newly developed tools are faster in the part of preprocessing that handles barcodes and in the part that handles adapter trimming. In the original preprocessing pipeline, `Cutadapt` is responsible for adapter trimming. The effect of replacing `Cutadapt` with `Flexcat`, `Skewer` and `AdapterRemoval 2` is shown in figure 3.4. Just replacing `Cutadapt` already results in a runtime speed gain by a factor of about two. This means that `Flexcat`, `Skewer` and `AdapterRemoval 2` are almost equally fast if we look at adapter trimming alone. Another speed gain by a factor of about two is visible when using `Nexcat` for barcode processing (setups 1 – 2 in figure 3.4) instead of R-scripts. The runtimes shown in figure 3.4 also include mapping the preprocessed reads using `Bowtie`. For setups 1 – 2, `Bowtie` already takes up more than 90% of the time. Therefore, further improvements in runtime should concentrate on improving the performance of mapping.

To better analyze scalability with regard to performance, the runtime for the different preprocessing setups has been measured for different numbers of threads.

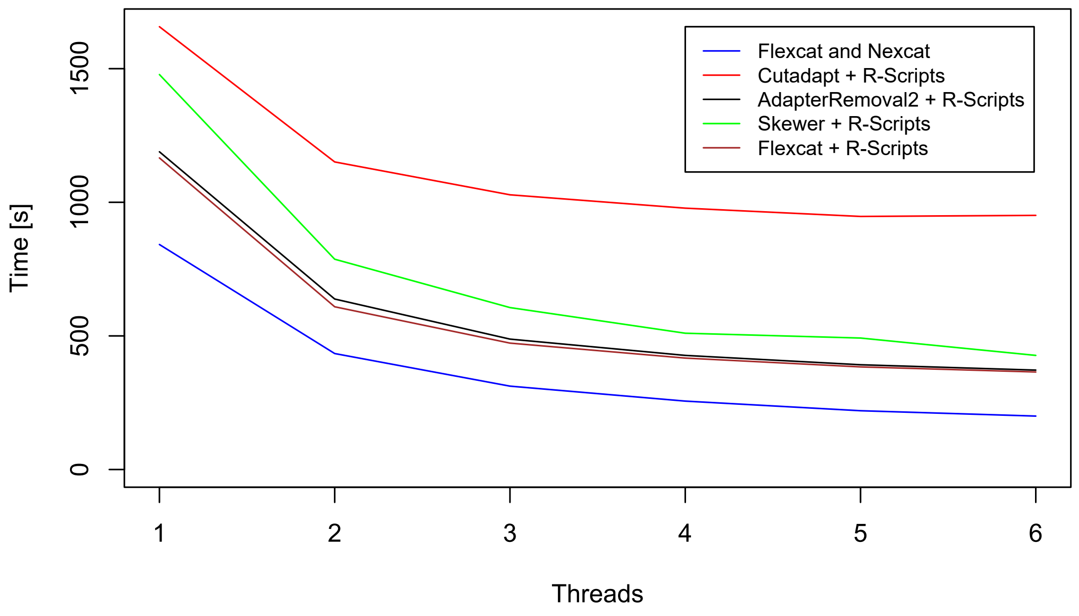


Figure 3.5: Runtime for different adapter trimmers using data set SRR1175698 (*Drosophila Melanogaster*, transcription factor Dorsal). The x-axis shows the number of threads used, the y-axis shows the runtime.

Figure 3.5 shows that the speedup is not optimal anymore after four or more threads are used. This is mainly because at this speed the IO-bandwidth (Input-/Output-bandwidth) is not sufficient anymore to allow maximum throughput. At the end of preprocessing, `Flexcat` shows a statistic with time spend for IO-tasks and for trimming-tasks which confirmed the IO-bottleneck for high numbers on threads. Because of this IO-bottleneck, the maximum number of threads used for benchmarks has been limited to 6.

3.2 Application of motif analysis to ChIP-nexus data

As described in 1.9, motif analysis requires DNA sequences. However, the output of peak-calling is a set of regions in the genome, the bed files. Since the regions that are reported by the peak caller have a width of one, I have first extended them to a pre specified width, here 20, using a shell script and the Linux line parser `awk`. Next, I used `bedtools` [56] with the `getfasta` option to retrieve the sequences for each region specified in the width-extended bed file. Following this, bases of repetitive sequences, which are written in lowercase letters, were replaced by N as suggested in [1].

Next, de novo motif discovery using `Dreme` [36] was performed. The result of this analysis is a ranked list of motifs, or more precisely, a `dreme.xml` file. The tool `Fimo` [57] was then used to select the motif-containing peaks as seen in the script shown in appendix B. This part can be difficult, because `Fimo` also reports non-exact matching motifs. Therefore, the p-value cutoff needs to be adjusted so that only the intended motifs are reported. If the cutoff value is too low, `Fimo` reports peaks that contain the selected motif with several mismatches. Since different motifs of the same protein can show very different binding patterns, selecting only those peaks that contain the motifs we want to analyze is very important.

These motif-containing peaks are then fed into a program that counts the forward and reverse strand 5'-ends in a given range around the input regions and outputs the result in a tab-separated text-file. This program has also been developed as part of this thesis and is called `5-PrimeEndCounter` [58]. The output text-file is then fed into an R-script that generates the 5'-end motif graphs.

For this specific analysis, the maximum motif length was set to 10. Since small variations in the input file often result in different but similar motifs being discovered, I first ran the motif discovery separately for all experiments, and then selected one or multiple motifs for each target protein (replicate pair) for further analysis. It would also be possible to use the highest ranked motif for each experiment and then compare possibly different motifs in the 5'-end motif graph.

The graphs on the left side in figure 3.6 show a global view of the 5'-ends around the motif location. This can be useful for assessing the noise present in the data or the peak density around a certain motif. The graphs on the right side show a detailed view of the 5'-end distribution around the motif location. These graphs are usually used to draw conclusions about the binding mechanisms of the protein of interest with the target DNA.

The overview graphs on the left side in figure 3.6 show that the peaks that are shown in more detail in the graphs on the right hand side are in fact located on top of a much broader peak. An explanation for the visible accumulation of 5'-ends around peaks could be that protein-DNA binding sites are often located closely together; therefore, 5'-ends from neighboring peaks can show up in the overview plot. Another possible explanation is that the exonuclease that trims DNA fragments on the 5'-end side until it reaches the first cross-link between the protein and DNA stops prematurely. This can happen if the fragmentation

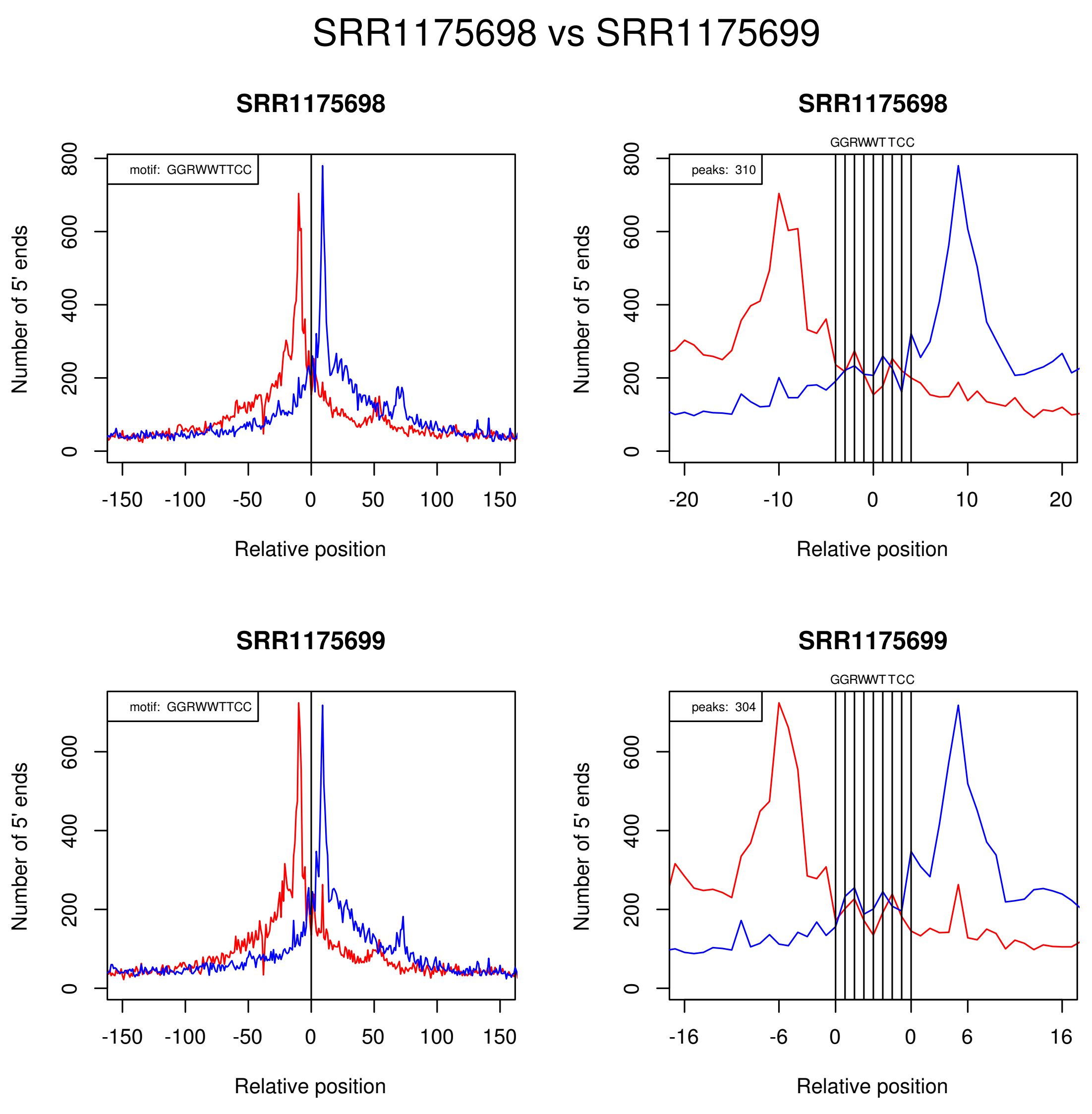


Figure 3.6: Comparison of two replicates of the same experiment for the same motif. The x-Axis shows the position of 5'-ends relative to the motif. The y-axis shows the number of 5'-ends for each position. Red denotes 5'-ends on the forward strand, while blue denotes 5'-ends on the reverse strand.

of the DNA is not fine enough so that multiple protein-DNA complexes remain connected together.

Additional motifs analysis plots are shown in appendix D. In these plots different preprocessing methods are compared, depicted as 'new' and 'orig'. It was shown that the new preprocessing method can reproduce the results from the work presented by He et al. [24] with an increased number of motif containing peaks. Furthermore, there are plots like figure 3.6 available that compare the binding characteristics of the same motif for different replicates of the same experiment using the new preprocessing tools.

The software tools developed as part of this thesis have been integrated and published as part of our Q-nexus project [35]. The goal of this research project is to provide a complete solution from preprocessing to peak calling specifically optimized for ChIP-nexus experiments. The website of this project provides source codes as well as a tutorial [59].

4 Discussion

4.1 Preprocessing of ChIP-nexus data

This newly developed preprocessing pipeline enables simple and quick evaluation of ChIP-nexus as well as ChIP-exo data. It can be used in a modular fashion, calling each step of preprocessing separately or by using the provided python script, which allows preprocessing to be performed by just executing one command. The tutorial on our website [59] shows how to do either one.

One improvement of this work regarding the first stage of preprocessing are the new possibilities in configuring adapter trimming, such as 5'-end adapters or multiple adapters. The 5'-end adapter that has been used as shown above resulted in a higher number of unambiguously mappable reads after preprocessing. Therefore, we assume that there are really 5'-end adapter sequences present in some reads; however, we have not yet fully understood the mechanism behind this. Having the option to trim an adapter several times within one read allowed us to use less adapter templates compared to the original pipeline which also resulted in a speed improvement. The biggest improvement with regard to the number of mappable reads was achieved by allowing up to one mismatch when detecting the fixed barcodes. At first, this is not intuitive because the phred score of the fixed barcodes is usually pretty good and does not explain why so many fixed barcodes contain errors. I suspect that the errors in the barcode sequence are probably caused by instabilities during library preparation and are therefore not adequately represented by phred scores, which represent only errors caused by the sequencing process.

Major improvements have been achieved in speed and scalability by applying multithreading techniques to every part of the program and by using SIMD instructions as much as possible. Recent performance improvements for single data instructions were only moderate when compared to performance improvements due to more cores or SIMD instructions. Now, the multithreading architecture as well as the use of SIMD instructions are already optimized for the preprocessing tools developed within the scope of this thesis. Therefore, the remaining bottlenecks are now the IO bandwidth of the hardware and the mapping software. `Bowtie` does not use SIMD instructions yet but it's scalability for multicore systems is already pretty good. It is possible to gain a considerable speedup by optimizing the mapping software to make use of SIMD instructions. Regarding the software developed as part of this thesis, a small speedup can still be gained by implementing a quality trimming algorithm that uses SIMD instructions. During processing of our data, we did not use quality trimming, because the reads with poor quality were very few. However, depending on the data, it may be worthwhile enabling quality trimming. Until now, only one modern adapter trimmer uses phred scores to optimize the adapter trimming algorithm [38]. This adapter trimmer always uses phred scores to optimize adapter trimming, even if quality trimming is disabled. Therefore, figure 3.5 shows that the difference in mappable reads for setup number 4 does not change much whether quality trimming is enabled or not. `Flexcat` does not use quality information inside the adapter trimming algorithm yet. For data that has very high quality scores, such as the data used in our experiments, adapter trimming does not benefit much from taking into account phred scores. However, for situations with lower quality scores, this is probably different. The idea is that if there is a mismatch between the read and a base of an adapter template, then this mismatch should be weighted by the error probability of the corresponding base in the read. If, for example, there is a mismatch between read and adapter for a base that has phred score 0, which means a 100% chance of error, then the penalty for this mismatch should also be zero. I have already experimented with adapter trimming algorithms that use phred scores and will probably implement them into `Flexcat` in the future.

Other improvements have been made in the area of usability. Until now, it was quite complicated to set up the environment for preprocessing ChIP-nexus data, because many different tools were needed, especially the setup of R of all the required libraries. With the newly developed tools, it is not necessary anymore to install and configure R. The configuration of preprocessing parameters also became more flexible and accessible. The monitoring and quality control of the preprocessing process became easier owing to the statistics output of the newly developed tools. The duplication rate plots generated by `Nexcat` visualize the impact of the random barcode on the number of useable reads. This allows estimates to be made of the amount of PCR artifacts present in the sequencing library and can be useful for detecting sources of errors during the wet lab part of the experiment. Figure 3.2 in section 3.1 shows that the number of PCR artifacts, which are reads that are mapped to the same position and have the same random barcode, is usually quite small compared to the number of duplicate reads, which are reads mapped to the same position but with different random barcodes. This shows the benefit of using barcodes to discriminate between PCR artifacts and useful duplicates. The useful duplicates, which could not be distinguished from PCR artifacts before ChIP-nexus, can increase the specificity and sensitivity of subsequent steps of the data analysis and make it possible to increase the resolution of detected peaks up to a single base [24]. However, sometimes when the wet lab part of the experiment flawed, the number of PCR artifacts can become quite large. This reduces the amount of useable reads and makes it necessary to sequence many more DNA fragments.

Figure 4.1 shows an example in which the library contains more PCR artifacts than usual. The high PCR artifact line shown in figure 4.1 is an indicator that many reads with the same random barcode are mapped to the same position on the reference genome. This means that the signal to noise ratio in this sequencing library is not as high as for the other replicates, where the PCR artifact line is lower. As described above, a high redundancy in the sequencing library usually needs to be remedied by sequencing more DNA fragments. However, sequencing more DNA fragments is not only expensive and time consuming, it also cannot recover DNA fragments that are completely absent from the DNA library. Figure 2 in the appendix shows that for replicate SRR1509030, only 20277544 reads

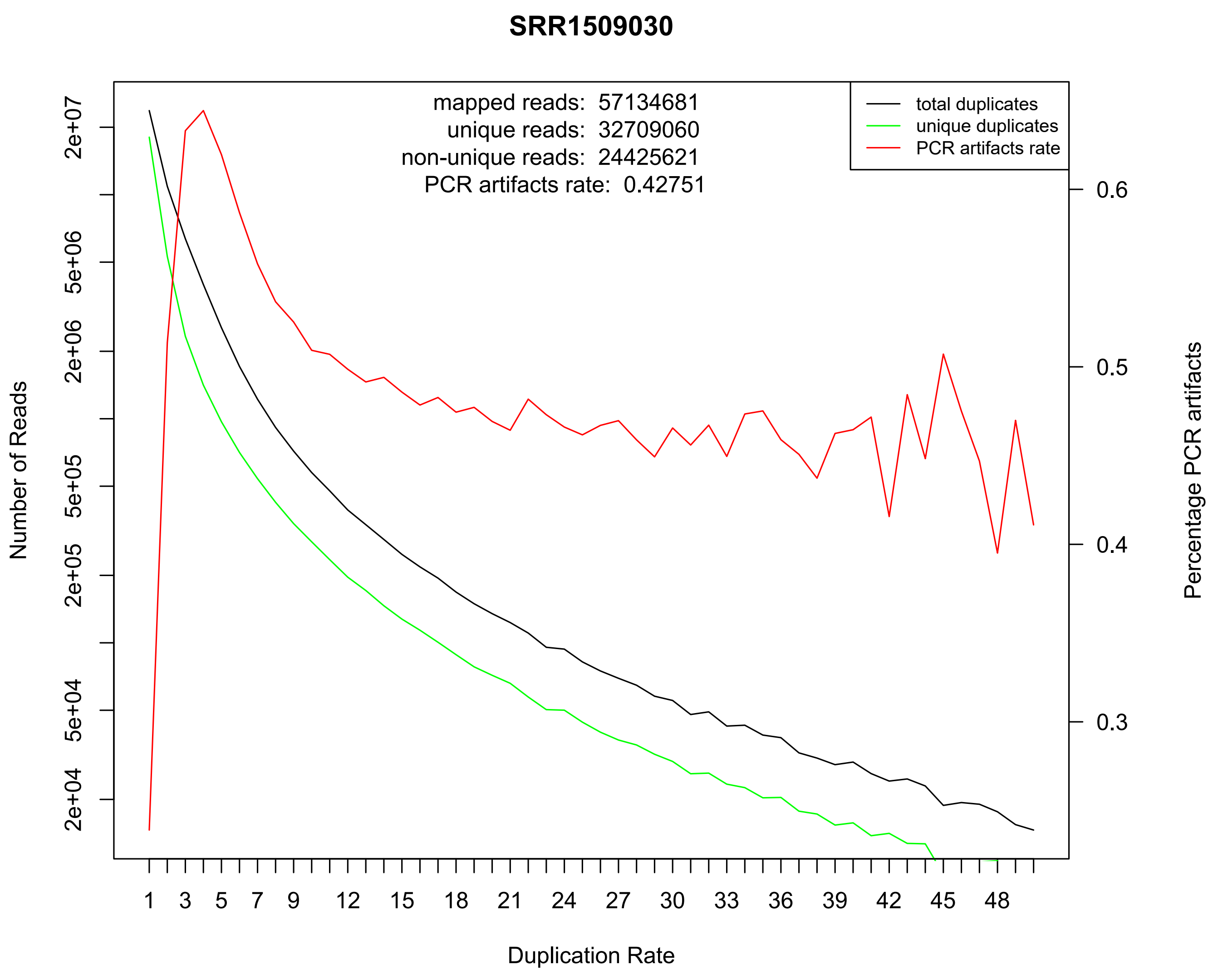


Figure 4.1: Duplication rate plot of data set SRR1509030 (Drosophila Melanogaster, transcription factor Dorsal)

have been mapped, whereas for replicate SRR1509029 57123681 reads have been mapped to the reference genome. The higher number of sequenced reads for replicate SRR1509029 was probably necessary because of the high redundancy in this sequencing library.

At this point, the most difficult step in analyzing of ChIP-nexus and ChIP-exo data is choosing the right parameters for adapter trimming and motif analysis. As described in section 3.1, there can be many different types of adapter contamination and some of them do not have an obvious reason, like the observed 5'-end adapter contamination. Other parameters that are not easy to select are the error rate for adapter trimming or the threshold for quality trimming. From my experience, optimizing the parameters for adapter trimming is not as critical as choosing parameters for motif analysis, because most adapters that are not completely trimmed only cause some reads not to be mappable to the reference genome. This causes a decreased signal in the downstream analysis but probably does not completely change the result, whereas slightly different parameters for motif analysis can change the picture drastically. In the future, it would be desirable to have a tool that can automatically suggest parameters for adapter trimming for a specific set of input data. This can probably be implemented by doing multiple runs with a small portion of the input data and then optimizing the parameters to yield the maximum mappable reads.

4.2 Using motif analysis to analyze the effect of protein-DNA interaction on gene regulation

As mentioned before, parameters play a crucial role in motif analysis. In section 3.2, it was shown that it is not trivial to exactly specify an exact motif sequence which is predominant. To illustrate this, I took the twist protein, which is a transcription factor, as an example. During development of the drosophila embryo, the twist transcription factor is necessary for differentiation of blastoderm to mesoderm [60]. Because of its ability to induce transformation of tissue, this transcription factor has also been shown to be relevant for metastatic tumor growth [61, 62]. The interaction of Twist with DNA shows a very complex behavior that is still not completely understood. Previous work using ChIP-seq indicated [63] that Twist binds to a so called ebox. Eboxes are enhancing regions that have a distinct pattern and can interact with proteins that increase target gene transcription when bound to the ebox. The motif class that is present in the Twist ebox is `CANNTG`. Results from previous work [63] were confirmed and further explained by applying the improved ChIP-exo protocol [64]. It was demonstrated that the variants with `CA` and `GA` at the `NN` location of the motif class `CANNTG` were major motifs and variants with `GC`, `TA` and `CT` relatively minor motifs. Upstream of the rho enhancer are two Twist eboxes next to each other, separated by only 5 bps, with motifs `CATATG` and `CACATG`, respectively. Furthermore, it was demonstrated that the motifs of these eboxes are not interchangeable. When they edited the cis regulatory module (CRM) so that both eboxes had the motif `CACATG`, the effect on gene transcription was not altered compared to wild type. However, editing both sites to have the motif `CATATG` revealed that the CRM was not fully functional anymore.

By applying the new ChIP-nexus protocol to the same target proteins, it was possible to see the details of protein-DNA interaction in greater detail [24]. Figure 4.2 shows a comparison of the binding profile generated from ChIP-nexus data for the motifs `CACATG` and `CATATG`. For motif `CATATG`, the pattern of two protein binding sites next to each other centered around the motif is recognizable. However, for motif `CACATG` this is not the case. Instead, the binding profile shows that there is a binding location centered on the motif and in addition to that two other binding locations to the left and the right of the motif.

These results demonstrate very impressively that protein-DNA interaction is very complex and cannot be fully explained by a single motif at a single position. Instead, multiple motifs at multiple locations have to be taken into account. In order to perform this task best, a high positional resolution of the tools is essential. The tools presented in this work together with the ChIP-nexus protocol help to improve the positional resolution and are useful for future research.

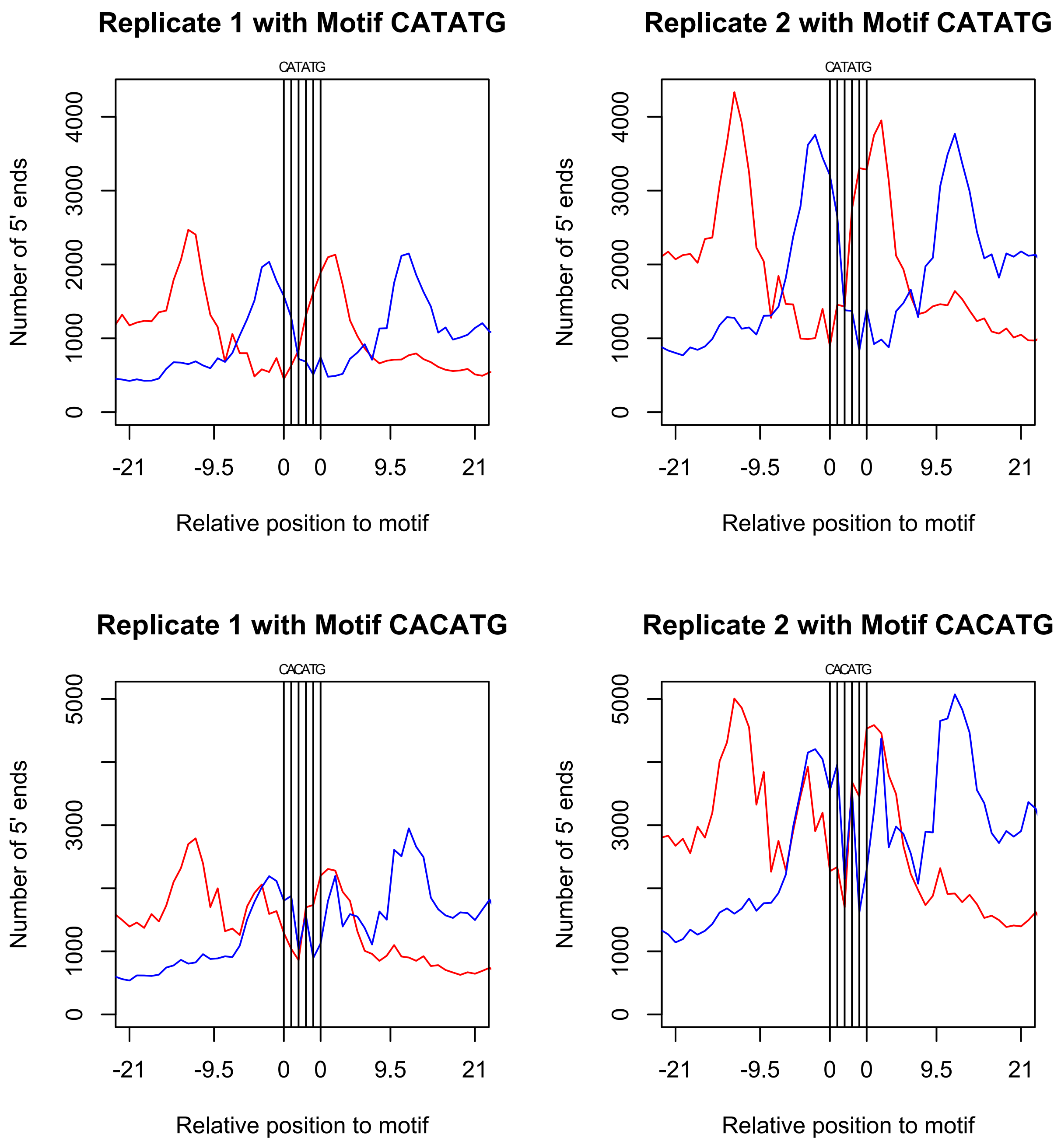


Figure 4.2: Protein-DNA binding patterns of Twist for different motifs. Protein binding sites are between red and blue peaks from left to right.

5 Summary

ChiP-nexus is a new method for analyzing protein-DNA interaction in a higher positional resolution than possible before. It is based on ChIP-exo and extends this methods by applying random barcodes to identify artifacts that originate from PCR amplification. Until now, there is only a very limited number of software available to analyze data produced by ChIP-nexus experiments. In our group we have developed an integrated pipeline that comprises all the tools necessary for the analysis of ChIP-nexus data, from preprocessing to peak calling.

With the newly developed preprocessing tools, `Flexcat` and `Nexcat`, the entire preprocessing has been accelerated and integrated into one pipeline. During the development of the tools presented in this thesis, modern design principles and special instructions for parallel data processing have been used. This led to a very big preprocessing speedup by a factor of about 5. The time required for preprocessing a specific data set, which used to be around one hour, could be reduced to 12 minutes. Moreover the complexity has been greatly reduced, so that persons who are not experts in programming and scripting can use it. Validation has been carried out by using a newly developed ChIP-nexus read simulator combined with statistical analysis and by applying unit testing. The newly introduced duplication rate plots allow to easily inspect the quality of the ChIP-nexus experiment and sequencing library. They are also very useful in visualizing the impact that the random barcodes, which were introduced by the ChIP-nexus protocol, have on the amount of useable reads. Since all the developed preprocessing tools are open source, they can easily be adapted or further optimized. The license also allows to use the source code in other programs.

To demonstrate the applications of the methods presented in this thesis to medicine and biology research, the analysis steps that follow preprocessing have been described and carried out for a concrete example. The alignment files which were generated by the preprocessing pipeline have been used to perform peak calling followed by motif analysis.

My experience from working with ChIP-nexus data is that the algorithms used from sequencing to peak calling are pretty mature already. However, motif analysis is still a challenge, because the resulting motifs can vary significantly by just changing the input parameters slightly. Furthermore, it has been shown that the same protein can bind to slightly different motifs in a completely different fashion. Therefore, this step of the analysis cannot be automated and requires expert knowledge.

As the sequencing costs continue to decrease, it will become possible to obtain higher sequencing depths by sequencing even more reads. This can help to increase the specificity and sensitivity of many ChIP-protocols, including ChIP-nexus. Because the time needed to process the huge amount of data is already a big factor today, highly efficient and scalable tools, as presented in this work, help to process the data in a reasonable time frame. The decrease in sequencing costs will almost automatically lead to more data on individual protein-DNA interactions. However, as it has been with the large expectations in gene therapy, more data does not automatically lead to more clinically relevant results. I think in order to overcome this problem and merge more data together, holistic approaches [65, 66] are necessary. Promising methods currently being applied to this problem are processing genomic data with machine learning [67] or using an integrative systems biology approach [68].

The tools presented in this thesis were developed and published as part of the Q-nexus project [35].

Appendices

A preprocess.py

Listing 1: command lines

```
#!/usr/bin/env python
# Author: Benjamin Menkuec
# Copyright 2015 Benjamin Menkuec
# License: LGPL

import sys
import os.path
import subprocess
import argparse
import platform
import multiprocessing

# this function calls a subroutine that will call samtools to
# sort the bam file and then build an index of it
def indexBamFile(filename, script_path):
        args = ("python", script_path + "/bam_indexer.py", filename)
        print args
        print "Creating indexed bam file..."
        popen = subprocess.Popen(args, stdout=subprocess.PIPE)
        popen.wait()
        output = popen.stdout.read()
        if results.verbose == True:
                print output
        if popen.returncode != 0:
                print "error"
                sys.exit()
        return;

script_path = os.path.dirname(os.path.realpath(__file__))
dataDir = os.getcwd() + "/"

parser = argparse.ArgumentParser(description="Preprocess fastq files and do mapping")
parser.add_argument('--adapters', type=str, default = "/../data/adapters.fa")
parser.add_argument('--barcodes', type=str, default = "/../data/barcodes.fa")
parser.add_argument('--flexcat_er', type=str, default = "0.2")
parser.add_argument('--flexcat_ol', type=str, default = "4")
parser.add_argument('--flexcat_ml', type=str, default = "19")
parser.add_argument('--flexcat_oh', type=str, default = "0")
parser.add_argument('--flexcat_times', type=str, default = "1")

parser.add_argument('--exo', action='store_true')
parser.add_argument('--clean', action='store_true')
parser.add_argument('--overwrite', action='store_true')
parser.add_argument('--verbose', action='store_true')
parser.add_argument('--bowtie_location', nargs='?', default = "")
parser.add_argument('--num_threads', nargs='?', default =
        str(multiprocessing.cpu_count()))
parser.add_argument('input_file')
parser.add_argument('--output_dir', type=str)
parser.add_argument('--filter_chromosomes', type=str, default="")
parser.add_argument('--random_split', action='store_true')
parser.add_argument('genome', type=str)

results, leftovers = parser.parse_known_args()

flexcatAdapterFilename = (os.path.dirname(os.path.realpath(__file__)) +
        results.adapters)
flexcatBarcodeFilename = (os.path.dirname(os.path.realpath(__file__)) +
        results.barcodes)

print "Reads: " + results.input_file
print "Genome: " + results.genome

genomeFilename = results.genome
bowtieLocation = results.bowtie_location
if len(bowtieLocation) > 0:
    bowtieLocation = bowtieLocation + "/"

if(platform.system() == "Windows" and results.bowtie_location == ""):
 print "Bowtie location is required under windows"
 sys.exit()

inputFile = os.path.abspath(results.input_file)

temp, inFileExtension = inputFile.split(os.extsep, 1)
# The output file of flexcat can not be a zipped fastq,
# because bowtie can not handle it.
# Therefore, remove .gz ending if its a fastq.gz file
inFileExtension, temp = os.path.splitext(inFileExtension)
inFileExtension = "." + inFileExtension
inFilenamePrefixWithoutPath = os.path.basename(inputFile)
inFilenamePrefixWithoutPath, temp = inFilenamePrefixWithoutPath.split(os.extsep, 1)

# set the output filename of flexcat
if results.output_dir is not None:
    outputDir = os.path.abspath(results.output_dir)
    head, tail = os.path.split(results.output_dir)
    if len(tail) > 0:
        inFilenamePrefixWithoutPath = tail;
    outputDir = outputDir + "/"
else:
    outputDir = inFilenamePrefixWithoutPath
flexcatOutputFilename = (outputDir + "/" + inFilenamePrefixWithoutPath +
        inFileExtension)

# if the exo option is set, there wont be any fixed barcode matching.
# therefore the output file of flexcat does not have a postfix
if results.exo:
 bowtieInputFilename = (outputDir + "/" + inFilenamePrefixWithoutPath +
        inFileExtension)
else:
 bowtieInputFilename = (outputDir + "/" + inFilenamePrefixWithoutPath +
        "_matched_barcode" + inFileExtension)
bowtieOutputFilename = outputDir + "/" + inFilenamePrefixWithoutPath + ".sam"

# platform independant way of calling flexcat
flexcat_path = script_path+"/../bin/flexcat"
if(platform.system() == "Windows"):
        flexcat_path += ".exe"

if results.exo:
 args = (flexcat_path, results.input_file, "-tt", "-t", "-ss", "-st", "-app",
        "-tnum", results.num_threads, "-times", results.flexcat_times, "-er",
        results.flexcat_er, "-ol", results.flexcat_ol, "-oh", results.flexcat_oh,
        "-ml", results.flexcat_ml, "-a",
        flexcatAdapterFilename,"-o", flexcatOutputFilename)
else:
 args = (flexcat_path, results.input_file, "-tl", "5", "-tt", "-t", "-ss", "-st",
        "-app", "-tnum", results.num_threads, "-times", results.flexcat_times, "-er",
        results.flexcat_er, "-ol", results.flexcat_ol, "-oh", results.flexcat_oh,
        "-ml", results.flexcat_ml, "-b",
        flexcatBarcodeFilename, "-a", flexcatAdapterFilename,"-o", flexcatOutputFilename)
if not os.path.exists(outputDir):
 os.makedirs(outputDir)
if (os.path.isfile(bowtieInputFilename) == False or results.overwrite == True):
    print "Filtering pre-mapping barcodes and trimming adapters..."
    if results.verbose == True:
        popen = subprocess.Popen(args + tuple(leftovers))
    else:
        popen = subprocess.Popen(args + tuple(leftovers), stdout=subprocess.PIPE)
    popen.wait()
    if popen.returncode != 0:
     print "error"
     sys.exit()
    if results.verbose == True:
        print flexcatOutputFilename + " created"

head, tail = os.path.split(genomeFilename)
genomeIndex, file_extension = os.path.splitext(tail)

# check if bowtie index already exists
# if yes, skip building the index
genomeIndexFile = os.path.dirname(genomeFilename) + "/" + genomeIndex;
if (os.path.isfile(genomeIndexFile + ".1.ebwt") == False):
 if(platform.system() == "Linux" or platform.system() == "Linux2"):
  args = (bowtieLocation + "bowtie-build", "-o", "1", genomeFilename,
        genomeIndexFile)
 else:
  args = ("python",  bowtieLocation + "bowtie-build", "-o", "1",
        genomeFilename, genomeIndexFile)
 popen = subprocess.Popen(args, stdout=subprocess.PIPE)
 popen.wait()
 output = popen.stdout.read()
 if results.verbose == True:
    print output
 if popen.returncode != 0:
  print "error"
  sys.exit()

# call bowtie if mapped file does not exist or overwrite is true
if (os.path.isfile(bowtieOutputFilename) == False or results.overwrite == True):
    args = (bowtieLocation + "bowtie", "-S", "-p", results.num_threads,
                "--chunkmbs", "512", "-k", "1", "-m", "1", "-v", "2", "--strata", "--best",
                genomeIndexFile, bowtieInputFilename, bowtieOutputFilename)
    if(platform.system() != "Linux" and platform.system() != "Linux2"):
     args = ("python",) + args
    print "Mapping reads..."
    popen = subprocess.Popen(args, stdout=subprocess.PIPE)
    popen.wait()
    output = popen.stdout.read()
    if results.verbose == True:
        print output
    if popen.returncode != 0:
     print "error"
     sys.exit()

# nexus-pre
nexusOutputFilename = (outputDir + "/" + inFilenamePrefixWithoutPath +
        "_filtered.bam")
if results.random_split == True:
        nexusOutputFilenameSplit1 = (outputDir + "/" + inFilenamePrefixWithoutPath +
        "_filtered_split1.bam")
        nexusOutputFilenameSplit2 = (outputDir + "/" + inFilenamePrefixWithoutPath +
        "_filtered_split2.bam")

# platform independant way of calling nexus-pre
nexcat_path = script_path+"/../bin/nexcat"
if(platform.system() == "Windows"):
        nexcat_path += ".exe"

# call nexcat if overwrite is true or output files dont exist
if (os.path.isfile(nexusOutputFilename) == False or
        (results.random_split == True and
                (os.path.isfile(nexusOutputFilenameSplit1) == False or
                        os.path.isfile(nexusOutputFilenameSplit2) == False)) or
                results.overwrite == True):
    args = (nexcat_path, bowtieOutputFilename,  "-fc", results.filter_chromosomes)
    if results.random_split == True:
                args += ("-rs",)
    print "Filtering post-mapping barcodes..."
    popen = subprocess.Popen(args, stdout=subprocess.PIPE)
    popen.wait()
    output = popen.stdout.read()
    if results.verbose == True:
        print output
    if popen.returncode != 0:
        print "error"
        sys.exit()

# special option for binding characteristic analysis
indexBamFile(nexusOutputFilename, script_path)
if results.random_split == True:
        indexBamFile(nexusOutputFilenameSplit2, script_path)
        indexBamFile(nexusOutputFilenameSplit1, script_path)

# cleanup
if results.clean:
    print "deleting intermediate files..."
    os.remove(bowtieOutputFilename)
    os.remove(nexusOutputFilename)
    if results.exo:
        os.remove(flexcatOutputFilename)
    else:
        os.remove(outputDir + "/" + inFilenamePrefixWithoutPath +
                        "_matched_barcode" + inFileExtension)
        os.remove(outputDir + "/" + inFilenamePrefixWithoutPath +
                        "_unidentified" + inFileExtension)
```

B motif_analysis.sh

Listing 2: command lines

```
#!/bin/bash
#Author Benjamin Menkuec
#License: LGPL

width=$1
reffile=$2
peakfile=$3
selected_motif=$4

#extend width of peaks
less $peakfile | awk -v width=$width 'BEGIN {OFS="\t"} $1!="track" \
        {print $1,$2-width,$3+width,$4}' > temp.bed

#get sequence for peaks
bedtools getfasta -fi $reffile -bed temp.bed -fo temp.fa

#convert repeats to NNNNNN
less temp.fa | awk 'BEGIN {OFS="\t"} \
        {if(substr($1,1,1)==">"){print $0} \
        else{gsub(/[a-z]/,"N",$0);print $0}}'\
        > temp_no_repeats.fa

#find top N motifs until the selected one, maximum motif size = 10
dreme -m $selected_motif -maxk 10 -p temp_no_repeats.fa

#select m01 for first motif, m02 for second ....
motif_idline=$(grep "m0$selected_motif" ./dreme_out/dreme.xml)
motif=$(awk 'BEGIN{FS="\""}{print $4}' <<< $motif_idline)
echo "selected motif: " $motif

#find top motif in peak sequences
fimo --motif $motif ./dreme_out/dreme.txt temp_no_repeats.fa

#generate centered motif locations
less ./fimo_out/fimo.gff | grep "^chr"| awk 'BEGIN{OFS="\t"} \
        {split($1,chr,":"); split(chr[2],loc,"-"); \
        offset = int(($4+$5)/2); \
        print chr[1],loc[1]+offset,loc[1]+offset+1,$9}' \
        > centered_motif_locations.bed

#delete temporary files
rm temp.bed
rm temp.fa
rm temp_no_repeats.fa
```

C Duplication rate plots

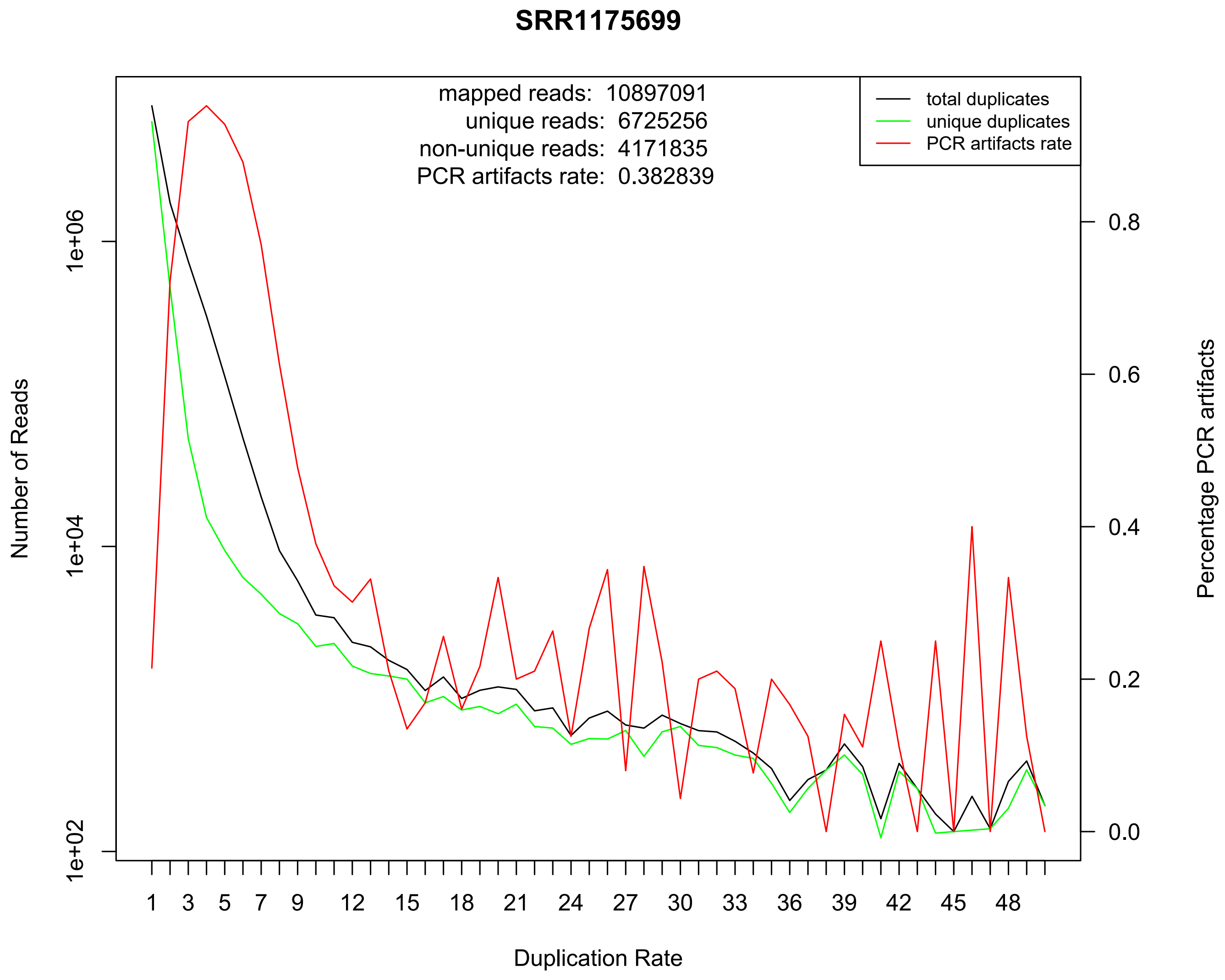


Figure 1: Duplication rate plot of data set SRR1175699 (Drosophila Melanogaster, transcription factor Dorsal)

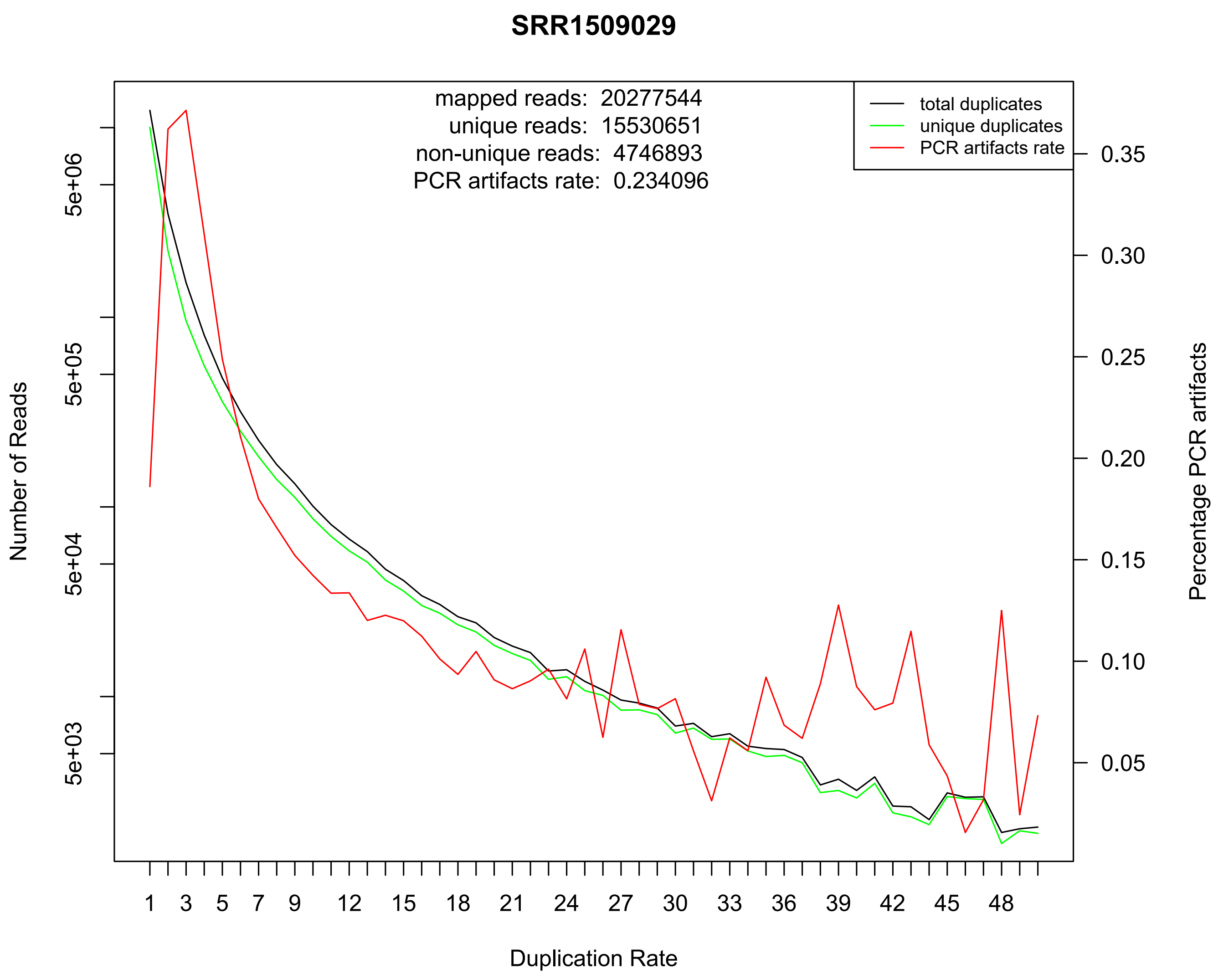
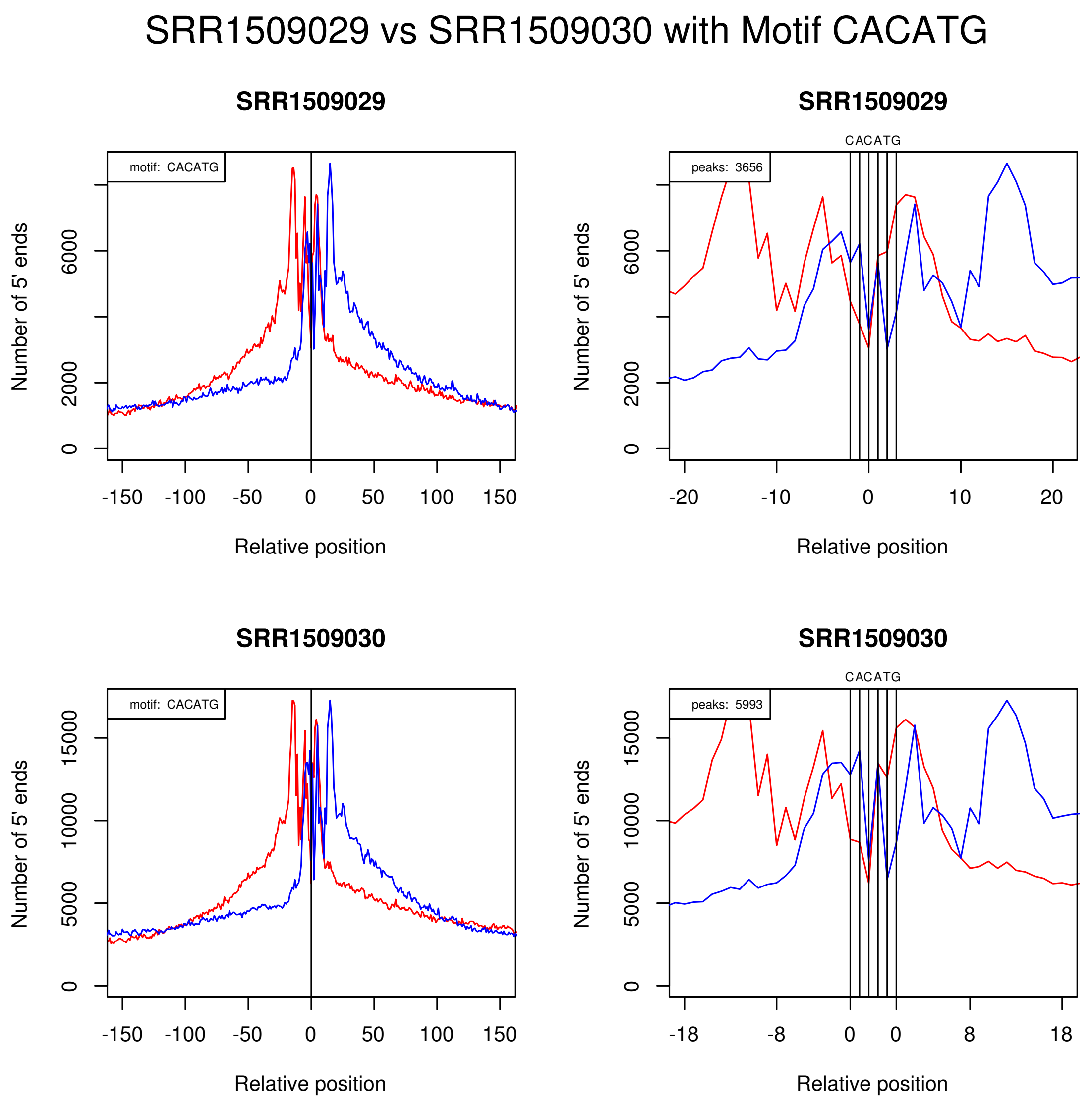
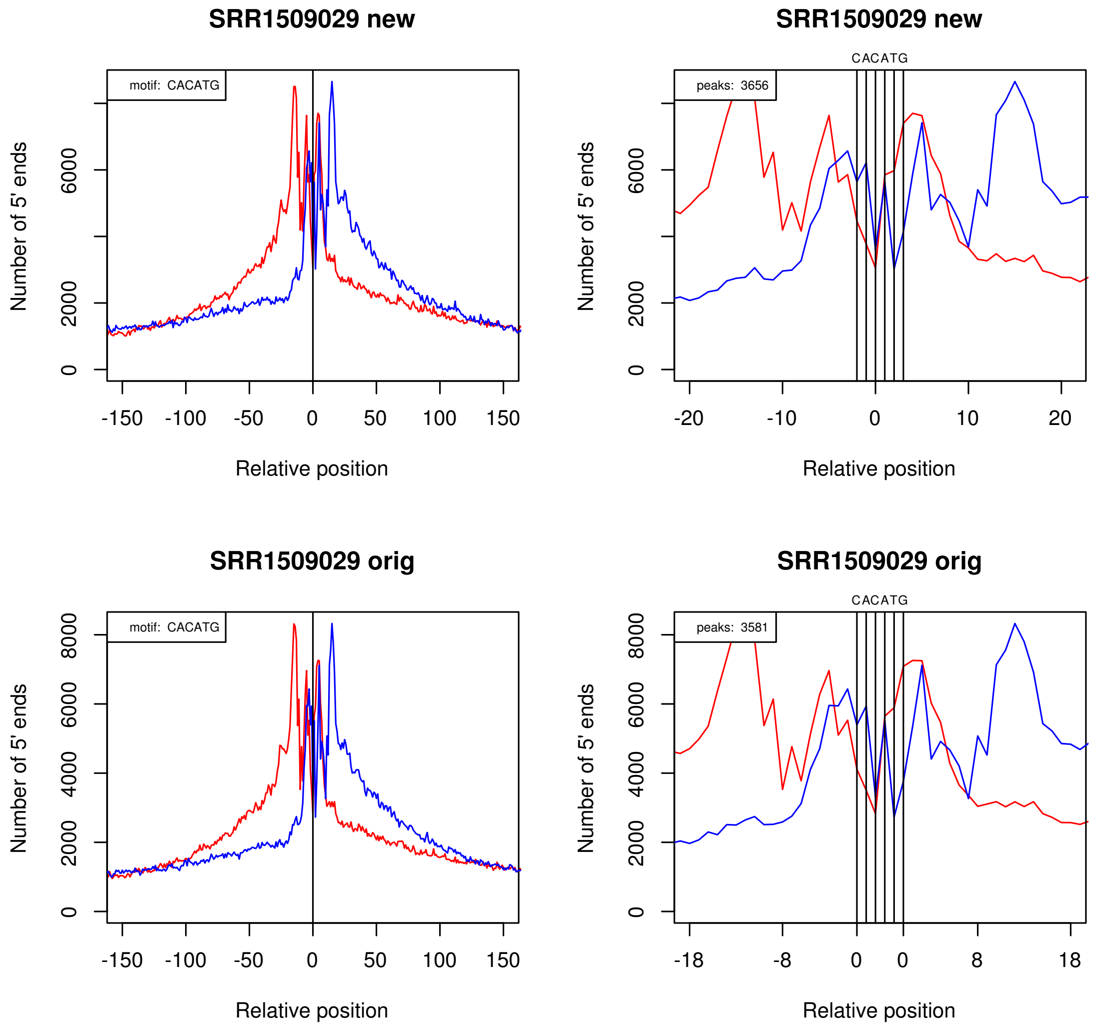


Figure 2: Duplication rate plot of data set SRR1509029 (Drosophila Melanogaster, transcription factor Twist)

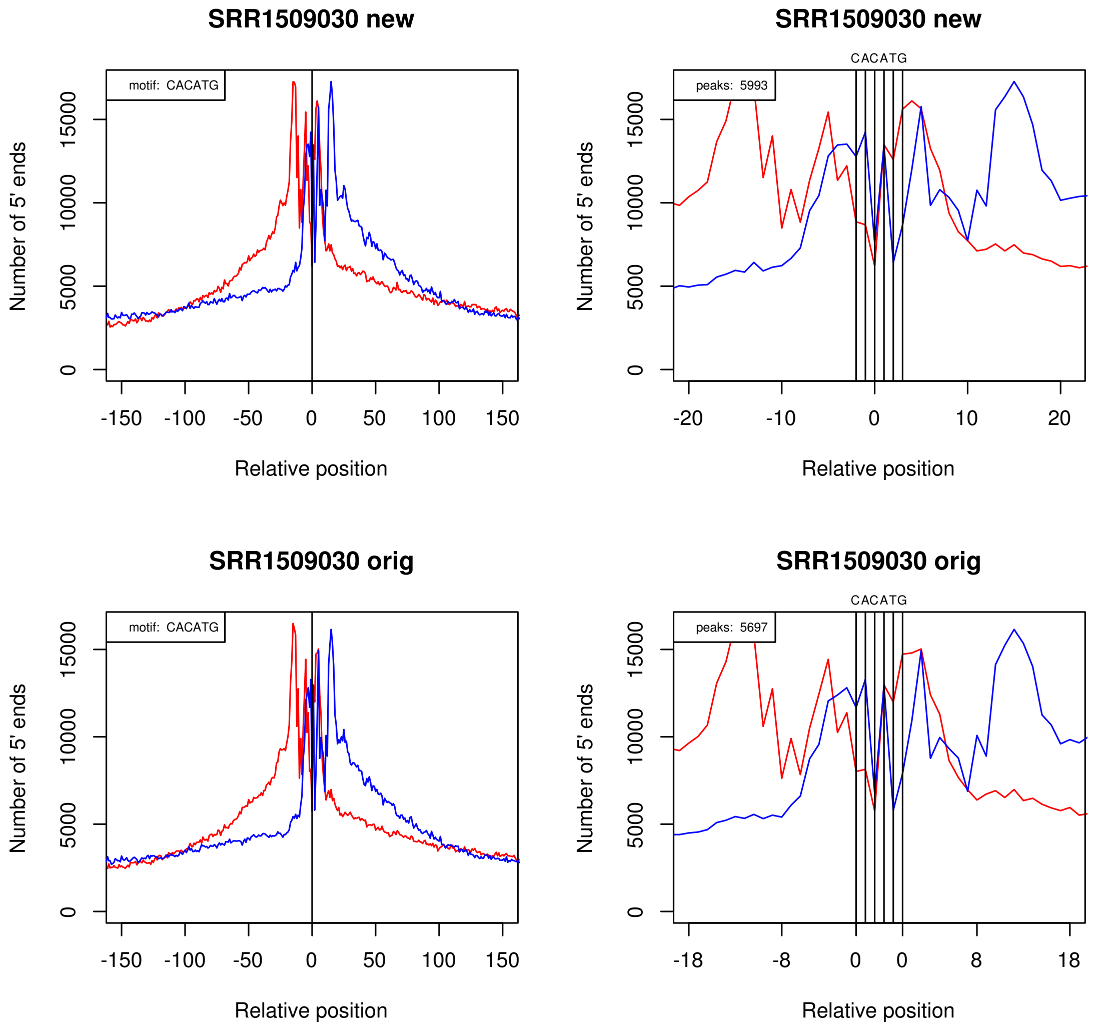
D 5'-end motif plots



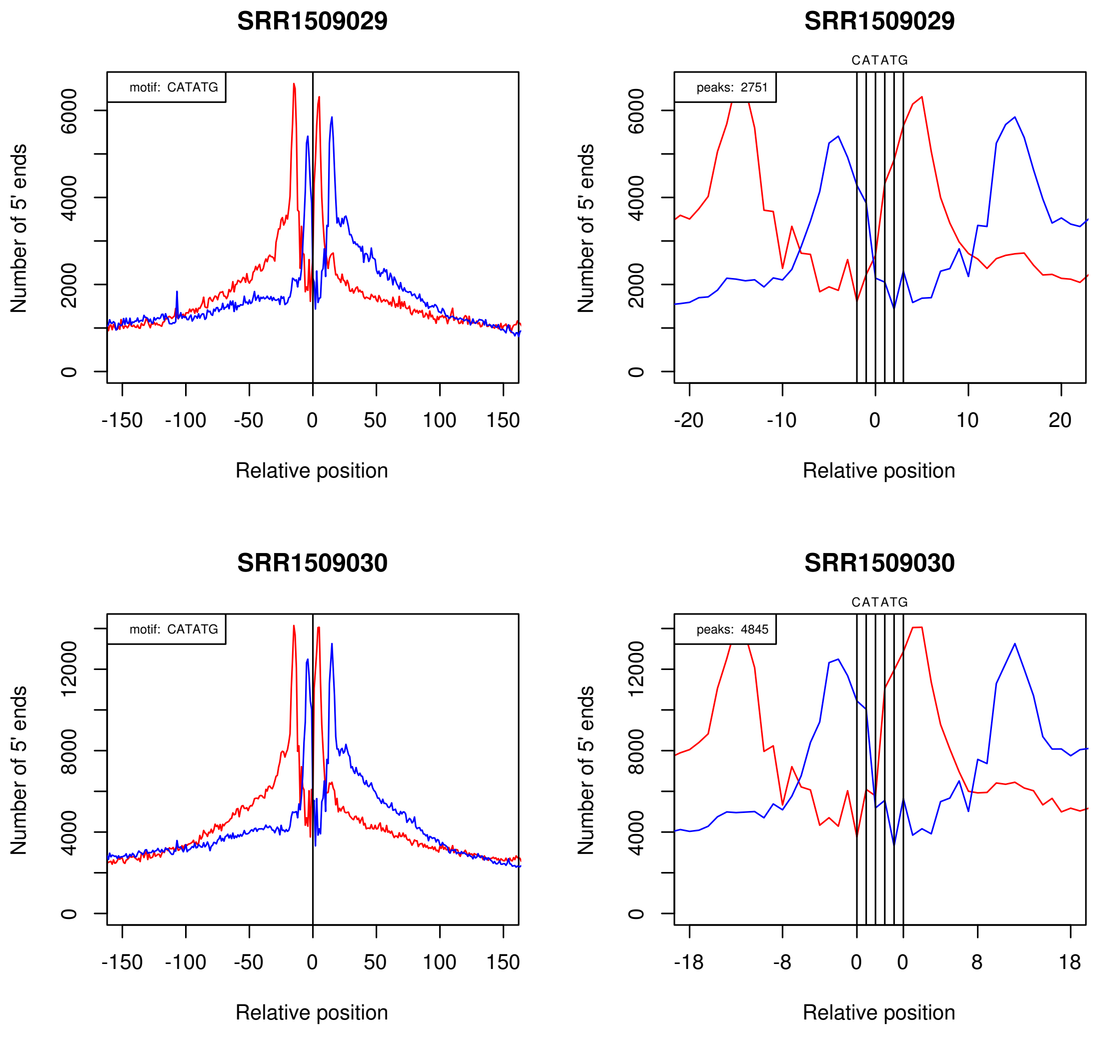
SRR1509029 new vs SRR1509029 orig with Motif CACATG



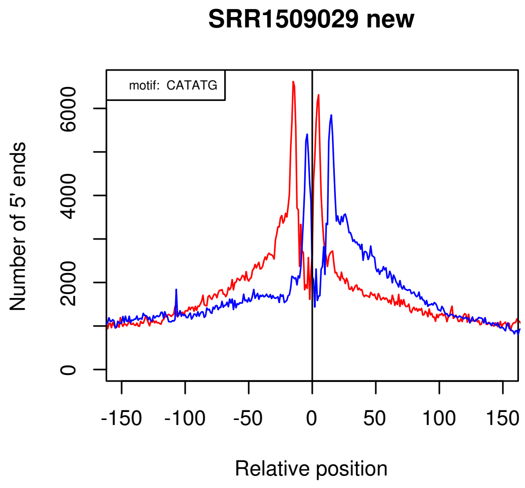
SRR1509030 new vs SRR1509030 orig with Motif CACATG

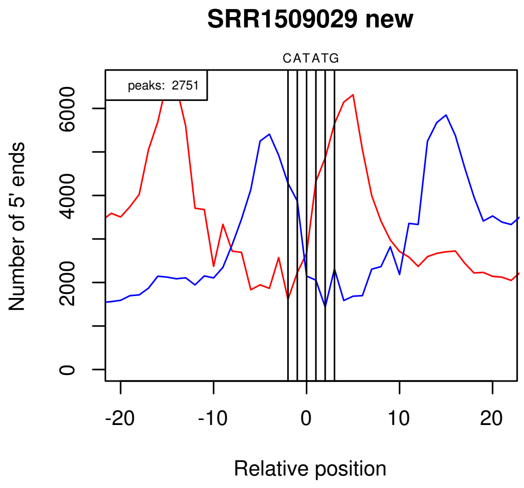


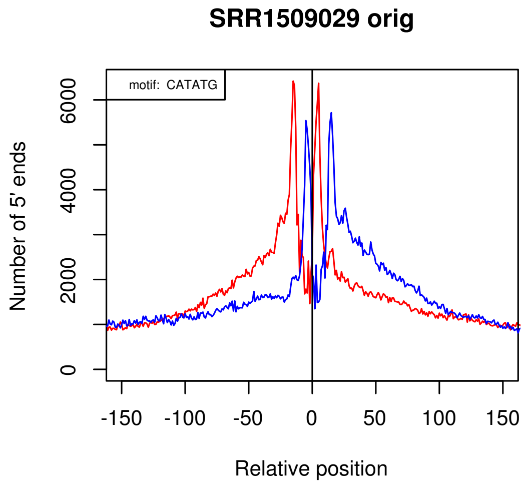
SRR1509029 vs SRR1509030 with Motif CATATG

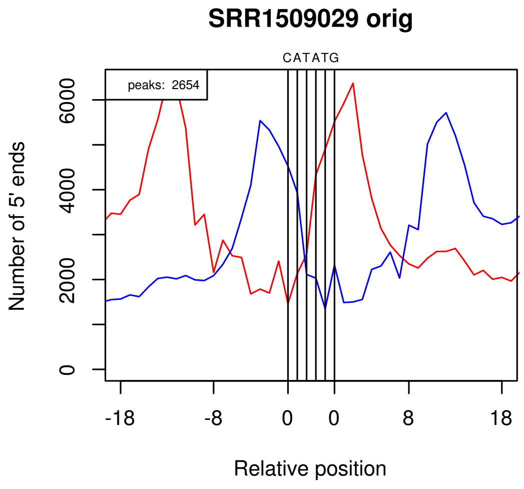


SRR1509029 new vs SRR1509029 orig with Motif CATATG

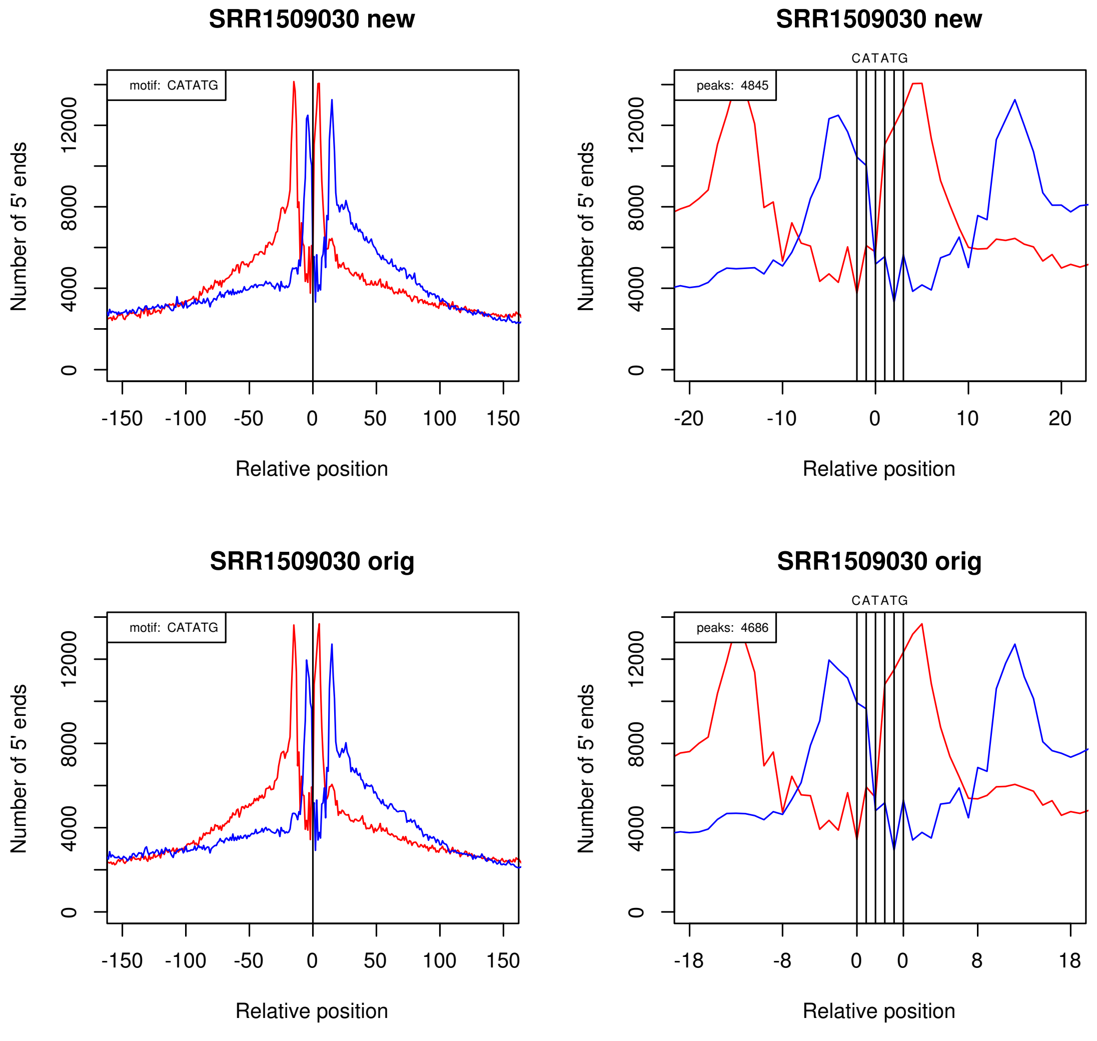








SRR1509030 new vs SRR1509030 orig with Motif CATATG



Nomenclature

Amplicon	A piece of DNA that is the source and/or product of amplification for example by PCR, page 5
ASCII	American Standard Code for Information Interchange, page 36
Cfp1	CXXC finger protein 1, page 10
ChIP	Chromatine immunoprecipitation, page 7
ChIP-exo	A method similar to ChIP-seq, but extends it by applying an exonuclease, page 1
ChIP-nexus	A method similar to ChIP-exo but extends it by adding random barcodes, page iii
ChIP-seq	A method that combines chromatin immunoprecipitation with next generation sequencing, page 7
CpG island	Region of DNA that contains many C and G bases, page 9
CRM	cis regulatory module, page 56
DNA	Deoxyribonucleic acid, page iii
dNTP	Nucleoside triphosphates containing deoxyribose, page 5
exo-Klenow fragment	A DNA Polymerase I which retains polymerase activity, but has lost the 5'-3' exonuclease activity, page 2
FN	False Negatives, page 43
FP	False Positives, page 43
FPR	False Positive Rate, page 44
H3K4me3	Methylated H3K4 histone, page 10
IO-bandwidth	Input-/Output-bandwidth, page 49
MCC	Matthews Correlation Coefficient, page 43
NGS	Next Generation Sequencing, page 1
PCR	Polymerase Chain Reaction, page 2
PCR duplicates	Copies of a read that originate from PCR, page 31

Phred score	Logarithmic score that describes error probability for sequenced base, page 31
POI	Protein Of Interest, page 7
PWM	Position Weight Matrix, page 19
ROC	Receiver Operating Characteristics, page 43
SBS	Sequencing by synthesis, page 6
SIMD	Single Input Multiple Data, page 28
SSE2	Streaming SIMD Extensions 2, page 29
SSE4	Streaming SIMD Extensions 4, page 29
TN	True Negatives, page 43
TP	True Positives, page 43
TPR	True Positive Rate, page 44
Transposome	A small piece of DNA that inserts itself into another place in a DNA fragment, page 2
TSS	Transcription Start Site, page 9

Bibliography

[1] P. Hansen, J. Hecht, D. M. Ibrahim, A. Krannich, M. Truss, and P. N. Robinson. Saturation analysis of chip-seq data for reproducible identification of binding peaks. *Genome Res*, 25(9):1391–400, 2015.

[2] F. U. Hartl, A. Bracher, and M. Hayer-Hartl. Molecular chaperones in protein folding and proteostasis. *Nature*, 475(7356):324–32, 2011.

[3] M. S. Kim, S. M. Pinto, D. Getnet, R. S. Nirujogi, S. S. Manda, R. Chaerkady, A. K. Madugundu, D. S. Kelkar, R. Isserlin, S. Jain, J. K. Thomas, B. Muthusamy, P. Leal-Rojas, P. Kumar, N. A. Sahasrabuddhe, L. Balakrishnan, J. Advani, B. George, S. Renuse, L. D. Selvan, A. H. Patil, V. Nanjappa, A. Radhakrishnan, S. Prasad, T. Subbannayya, R. Raju, M. Kumar, S. K. Sreenivasamurthy, A. Marimuthu, G. J. Sathe, S. Chavan, K. K. Datta, Y. Subbannayya, A. Sahu, S. D. Yelamanchi, S. Jayaram, P. Rajagopalan, J. Sharma, K. R. Murthy, N. Syed, R. Goel, A. A. Khan, S. Ahmad, G. Dey, K. Mudgal, A. Chatterjee, T. C. Huang, J. Zhong, X. Wu, P. G. Shaw, D. Freed, M. S. Zahari, K. K. Mukherjee, S. Shankar, A. Mahadevan, H. Lam, C. J. Mitchell, S. K. Shankar, P. Satishchandra, J. T. Schroeder, R. Sirdeshmukh, A. Maitra, S. D. Leach, C. G. Drake, M. K. Halushka, T. S. Prasad, R. H. Hruban, C. L. Kerr, G. D. Bader, C. A. Iacobuzio-Donahue, H. Gowda, and A. Pandey. A draft map of the human proteome. *Nature*, 509(7502):575–81, 2014.

[4] F. Sanger and A. R. Coulson. A rapid method for determining sequences in dna by primed synthesis with dna polymerase. *J Mol Biol*, 94(3):441–8, 1975.

[5] S. R. Starick, J. Ibn-Salem, M. Jurk, C. Hernandez, M. I. Love, H. R. Chung, M. Vingron, M. Thomas-Chollier, and S. H. Meijsing. Chip-exo signal associated with dna-binding motifs provides insight into the genomic binding of the glucocorticoid receptor and cooperating transcription factors. *Genome Res*, 25(6):825–35, 2015.

[6] Illumina. Truseq© dna sample preparation guide. Technical report, 2012.

[7] Illumina. Nextera© dna sample preparation guide. Technical report, 2012.

[8] G. Turcatti, A. Romieu, M. Fedurco, and A. P. Tairi. A new class of cleavable fluorescent nucleotides: synthesis and optimization as reversible terminators for dna sequencing by synthesis. *Nucleic Acids Res*, 36(4):e25, 2008.

[9] B. F. Luisi, W. X. Xu, Z. Otwinowski, L. P. Freedman, K. R. Yamamoto, and P. B. Sigler. Crystallographic analysis of the interaction of the glucocorticoid receptor with dna. *Nature*, 352(6335):497–505, 1991.

[10] D. S. Johnson, A. Mortazavi, R. M. Myers, and B. Wold. Genome-wide mapping of in vivo protein-dna interactions. *Science*, 316(5830):1497–502, 2007.

[11] M. T. Kassouf, J. R. Hughes, S. Taylor, S. J. McGowan, S. Soneji, A. L. Green, P. Vyas, and C. Porcher. Genome-wide identification of tal1's functional targets: insights into its mechanisms of action in primary erythroid cells. *Genome Res*, 20(8):1064–83, 2010.

[12] T. Fujiwara, H. O'Geen, S. Keles, K. Blahnik, A. K. Linnemann, Y. A. Kang, K. Choi, P. J. Farnham, and E. H. Bresnick. Discovering hematopoietic mechanisms through genome-wide analysis of gata factor chromatin occupancy. *Molecular Cell*, 36(4):667–681, 2009.

[13] H. S. Rhee and B. F. Pugh. Chip-exo method for identifying genomic location of dna-binding proteins with near-single-nucleotide accuracy. *Curr Protoc Mol Biol*, Chapter 21:Unit 21 24, 2012.

[14] G Celine Han, Vinesh Vinayachandran, Alain R Bataille, Bongsoo Park, Ka Yim Chan-Salis, Cheryl A Keller, Maria Long, Shaun Mahony, Ross C Hardison, and B Franklin Pugh. Genome-wide organization of gata1 and tal1 determined at high resolution. *Mol Cell Biol*, 2015.

[15] G. A. Calin, C. D. Dumitru, M. Shimizu, R. Bichi, S. Zupo, E. Noch, H. Aldler, S. Rattan, M. Keating, K. Rai, L. Rassenti, T. Kipps, M. Negrini, F. Bullrich, and C. M. Croce. Frequent deletions and down-regulation of micro- rna genes mir15 and mir16 at 13q14 in chronic lymphocytic leukemia. *Proc Natl Acad Sci U S A*, 99(24):15524–9, 2002.

[16] A. L. Statham, M. D. Robinson, J. Z. Song, M. W. Coolen, C. Stirzaker, and S. J. Clark. Bisulfite sequencing of chromatin immunoprecipitated dna (bischip-seq) directly informs methylation status of histone-modified dna. *Genome Research*, 22(6):1120–1127, 2012.

[17] A. B. Brinkman, H. C. Gu, S. J. J. Bartels, Y. Y. Zhang, F. Matarese, F. Simmer, H. Marks, C. Bock, A. Gnirke, A. Meissner, and H. G. Stunnenberg. Sequential chip-bisulfite sequencing enables direct genome-scale investigation of chromatin and dna methylation cross-talk. *Genome Research*, 22(6):1128–1138, 2012.

[18] M. Rodriguez-Paredes and M. Esteller. Cancer epigenetics reaches mainstream oncology. *Nat Med*, 17(3):330–9, 2011.

[19] S. M. Lauberth, T. Nakayama, X. L. Wu, A. L. Ferris, Z. Y. Tang, S. H. Hughes, and R. G. Roeder. H3k4me3 interactions with taf3 regulate preinitiation complex assembly and selective gene activation. *Cell*, 152(5):1021–1036, 2013.

[20] J. P. Thomson, P. J. Skene, J. Selfridge, T. Clouaire, J. Guy, S. Webb, A. R. W. Kerr, A. Deaton, R. Andrews, K. D. James, D. J. Turner, R. Illingworth, and A. Bird. Cpg islands influence chromatin structure via the cpg-binding protein cfp1. *Nature*, 464(7291):1082–U162, 2010.

[21] S. Peleg, F. Sananbenesi, A. Zovoilis, S. Burkhardt, S. Bahari-Javan, R. C. Agis-Balboa, P. Cota, J. L. Wittnam, A. Gogol-Doering, L. Opitz, G. Salinas-Riester, M. Dettenhofer, H. Kang, L. Farinelli, W. Chen, and A. Fischer. Altered histone acetylation is associated with age-dependent memory impairment in mice. *Science*, 328(5979):753–756, 2010.

[22] H. O'Geen, L. Echipare, and P. J. Farnham. Using chip-seq technology to generate high-resolution profiles of histone modifications. *Methods Mol Biol*, 791:265–86, 2011.

[23] T. S. Furey. Chip-seq and beyond: new and improved methodologies to detect and characterize protein-dna interactions. *Nature Reviews Genetics*, 13(12):840–852, 2012.

[24] Q. He, J. Johnston, and J. Zeitlinger. Chip-nexus enables improved detection of in vivo transcription factor binding footprints. *Nat Biotechnol*, 33(4):395–401, 2015.

[25] J. D. Nelson, O. Denisenko, and K. Bomsztyk. Protocol for the fast chromatin immunoprecipitation (chip) method. *Nat Protoc*, 1(1):179–85, 2006.

[26] V. Jackson. Studies on histone organization in the nucleosome using formaldehyde as a reversible cross-linking agent. *Cell*, 15(3):945–54, 1978.

[27] H. Thorvaldsdottir, J. T. Robinson, and J. P. Mesirov. Integrative genomics viewer (igv): high-performance genomics data visualization and exploration. *Brief Bioinform*, 14(2):178–92, 2013.

[28] T. S. Carroll, Z. Liang, R. Salama, R. Stark, and I. de Santiago. Impact of artifact removal on chip quality metrics in chip-seq and chip-exo data. *Front Genet*, 5:75, 2014.

[29] H. Lee and M. C. Schatz. Genomic dark matter: the reliability of short read mapping illustrated by the genome mappability score. *Bioinformatics*, 28(16):2097–105, 2012.

[30] P. Ramachandran, G. A. Palidwor, C. J. Porter, and T. J. Perkins. Masc: mappability-sensitive cross-correlation for estimating mean fragment length of single-end short-read sequencing data. *Bioinformatics*, 29(4):444–50, 2013.

[31] J. Feng, T. Liu, B. Qin, Y. Zhang, and X. S. Liu. Identifying chip-seq enrichment using macs. *Nat Protoc*, 7(9):1728–40, 2012.

[32] 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). *Genome Biol*, 9(9):R137, 2008.

[33] T. Liu. Use model-based analysis of chip-seq (macs) to analyze short reads generated by sequencing protein-dna interactions in embryonic stem cells. *Methods Mol Biol*, 1150:81–95, 2014.

[34] L. Wang, J. Chen, C. Wang, L. Uuskula-Reimand, K. Chen, A. Medina-Rivera, E. J. Young, M. T. Zimmermann, H. Yan, Z. Sun, Y. Zhang, S. T. Wu, H. Huang, M. D. Wilson, J. P. Kocher, and W. Li. Mace: model based analysis of chip-exo. *Nucleic Acids Res*, 42(20):e156, 2014.

[35] P. Hansen, J. Hecht, J. Ibn-Salem, B. S. Menkuec, S. Roskosch, M. Truss, and P. N. Robinson. Q-nexus: A comprehensive and efficient analysis pipeline designed for chip-nexus. *BMC Genomics*, 17:873, 2016.

[36] T. L. Bailey. Dreme: motif discovery in transcription factor chip-seq data. *Bioinformatics*, 27(12):1653–9, 2011.

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

[38] H. Jiang, R. Lei, S. W. Ding, and S. Zhu. Skewer: a fast and accurate adapter trimmer for next-generation sequencing paired-end reads. *BMC Bioinformatics*, 15:182, 2014.

[39] M. Dodt, J. T. Roehr, R. Ahmed, and C. Dieterich. Flexbar-flexible barcode and adapter processing for next-generation sequencing platforms. *Biology (Basel)*, 1(3):895–905, 2012.

[40] S. Roskosch. Implementierung eines schnellen ngs-data postprocessing toolkits in seqan, 2013.

[41] https://github.com/catkira/nexus-tools/tree/master/flexcat.

[42] https://github.com/catkira/ptc/tree/master/ptc.

[43] A. Doring, D. Weese, T. Rausch, and K. Reinert. Seqan an efficient, generic c++ library for sequence analysis. *BMC Bioinformatics*, 9:11, 2008.

[44] M. Schubert, S. Lindgreen, and L. Orlando. Adapterremoval v2: rapid adapter trimming, identification, and read merging. *BMC Res Notes*, 9:88, 2016.

[45] K. D. Hansen, S. E. Brenner, and S. Dudoit. Biases in illumina transcriptome sequencing caused by random hexamer priming. *Nucleic Acids Res*, 38(12):e131, 2010.

[46] H. Li and R. Durbin. Fast and accurate long-read alignment with burrows-wheeler transform. *Bioinformatics*, 26(5):589–95, 2010.

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

[48] https://github.com/vsbuffalo/scythe.

[49] S. Lindgreen. Adapterremoval: easy cleaning of next-generation sequencing reads. *BMC Res Notes*, 5:337, 2012.

[50] https://github.com/catkira/nexcat/tree/master/nexcat.

[51] W. Huang, L. Li, J. R. Myers, and G. T. Marth. Art: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–4, 2012.

[52] X. Hu, J. Yuan, Y. Shi, J. Lu, B. Liu, Z. Li, Y. Chen, D. Mu, H. Zhang, N. Li, Z. Yue, F. Bai, H. Li, and W. Fan. pirs: Profile-based illumina pair-end reads simulator. *Bioinformatics*, 28(11):1533–5, 2012.

[53] F. E. Angly, D. Willner, F. Rohwer, P. Hugenholtz, and G. W. Tyson. Grinder: a versatile amplicon and shotgun sequence simulator. *Nucleic Acids Res*, 40(12):e94, 2012.

[54] D. Pratas, A. J. Pinho, and J. M. Rodrigues. Xs: a fastq read simulator. *BMC Res Notes*, 7:40, 2014.

[55] A. Shcherbina. Fastqsim: platform-independent data characterization and in silico read generation for ngs datasets. *BMC Res Notes*, 7:533, 2014.

[56] A. R. Quinlan and I. M. Hall. Bedtools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–2, 2010.

[57] C. E. Grant, T. L. Bailey, and W. S. Noble. Fimo: scanning for occurrences of a given motif. *Bioinformatics*, 27(7):1017–8, 2011.

[58] https://github.com/catkira/nexus-tools/tree/master/5primeendcounter.

[59] http://charite.github.io/q/.

[60] M. Leptin. twist and snail as positive and negative regulators during drosophila mesoderm development. *Genes Dev*, 5(9):1568–76, 1991.

[61] S. M. Lu, L. Yu, J. J. Tian, J. K. Ma, J. F. Li, W. Xu, and H. B. Wang. Twist modulates lymphangiogenesis and correlates with lymph node metastasis in supraglottic carcinoma. *Chinese Medical Journal*, 124(10):1483–1487, 2011.

[62] M. A. Khan, H. C. Chen, D. Z. Zhang, and J. J. Fu. Twist: a molecular target in cancer therapeutics. *Tumor Biology*, 34(5):2497–2506, 2013.

[63] Y. T. Ip, R. E. Park, D. Kosman, K. Yazdanbakhsh, and M. Levine. dorsal-twist interactions establish snail expression in the presumptive mesoderm of the drosophila embryo. *Genes Dev*, 6(8):1518–30, 1992.

[64] A. Ozdemir, K. I. Fisher-Aylor, S. Pepke, M. Samanta, L. Dunipace, K. McCue, L. Zeng, N. Ogawa, B. J. Wold, and A. Stathopoulos. High resolution mapping of twist to dna in drosophila embryos: Efficient functional analysis and evolutionary conservation. *Genome Res*, 21(4):566–77, 2011.

[65] P. N. Robinson, S. Kohler, S. Bauer, D. Seelow, D. Horn, and S. Mundlos. The human phenotype ontology: a tool for annotating and analyzing human hereditary disease. *Am J Hum Genet*, 83(5):610–5, 2008.

[66] P. N. Robinson and S. Mundlos. The human phenotype ontology. *Clin Genet*, 77(6):525–34, 2010.

[67] S. Bauer, J. Gagneur, and P. N. Robinson. Going bayesian: model-based gene set analysis of genome-scale data. *Nucleic Acids Res*, 38(11):3523–32, 2010.

[68] S. Komili and P. A. Silver. Coupling and coordination in gene expression processes: a systems biology view. *Nature Reviews Genetics*, 9(1):38–48, 2008.

Eidesstattliche Versicherung

„Ich, Benjamin Sefa Menküc, versichere an Eides statt durch meine eigenhndige Unterschrift, dass ich die vorgelegte Dissertation mit dem Thema: „Development of a bespoke algorithm to analyze ChIP-nexus genome-wide protein-DNA binding profiles“ selbstständig und ohne nicht offengelegte Hilfe Dritter verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel genutzt habe.
Alle Stellen, die wörtlich oder dem Sinne nach auf Publikationen oder Vorträgen anderer Autoren beruhen, sind als solche in korrekter Zitierung (siehe „Uniform Requirements for Manuscripts (URM)“ des ICMJE -www.icmje.org) kenntlich gemacht. Die Abschnitte zu Methodik (insbesondere praktische Arbeiten, Laborbestimmungen, statistische Aufarbeitung) und Resultaten (insbesondere Abbildungen, Graphiken und Tabellen) entsprechen den URM (s.o) und werden von mir verantwortet.
Meine Anteile an etwaigen Publikationen zu dieser Dissertation entsprechen denen, die in der untenstehenden gemeinsamen Erklärung mit dem Betreuer, angegeben sind. Sämtliche Publikationen, die aus dieser Dissertation hervorgegangen sind und bei denen ich Autor bin, entsprechen den URM (s.o.) und werden von mir verantwortet.
Die Bedeutung dieser eidesstattlichen Versicherung und die strafrechtlichen Folgen einer unwahren eidesstattlichen Versicherung (§156, §161 des Strafgesetzbuches) sind mir bekannt und bewusst.“

Datum Unterschrift

Anteilserklärung an etwaigen erfolgten Publikationen

Benjamin Sefa Menküc hatte folgenden Anteil an den folgenden Publikationen:

Publikation 1: Hansen P, Hecht J, Ibn-Salem J, Menkuec BS, Roskosch S, Truss M, Robinson PN, Q-nexus: A comprehensive and efficient analysis pipeline designed for ChIP-nexus data, BMC Genomics, 17:873, 2016

Beitrag im Einzelnen (bitte kurz ausführen): Entwurf und Entwicklung der Softwaretools für das Preprocessing der rohen ChIP-nexus Daten

Unterschrift, Datum und Stempel des betreuenden Hochschullehrers

Unterschrift des Doktoranden

Mein Lebenslauf wird aus datenschutzrechtlichen Gründen in der elektronischen Version meiner Arbeit nicht veröffentlicht.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Dr. med. Peter Nick Robinson for the continuous support of my thesis, for his patience, motivation and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.
Last but not the least, I would like to thank my parents for supporting me throughout writing this thesis and in my life in general.