

# **Visual Analysis of Form and Function in Computational Biology**

Von der Fakultät für Mathematik und Informatik  
der Universität Leipzig  
angenommene

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

DOCTOR RERUM NATURALIUM  
(Dr. rer. nat.)

im Fachgebiet  
Informatik

vorgelegt  
von Master of Science Daniel Wiegrefe  
geboren am 14. Oktober 1989 in Lingen (Ems)

Die Annahme der Dissertation wurde empfohlen von:

1. Prof. Dr. Gerik Scheuermann, Universität Leipzig
2. Prof. Dr. Ivo L. Hofacker, Universität Wien

Die Verleihung des akademischen Grades erfolgte mit Bestehen  
der Verteidigung am 20. Juni 2019 mit dem Gesamtpredikat 'magna cum laude'.



# Abstract

In the last years, the amount of available data in the field of computational biology steadily increased. In order to be able to analyze these data, various algorithms have been developed by bioinformaticians to process them efficiently. Moreover, computational models were developed to predict for instance biological relationships of species. Furthermore, the prediction of properties like the structure of certain biological molecules is modeled by complex algorithms. Despite these advances in handling such complicated tasks with automated workflows and a huge variety of freely available tools, the expert still needs to supervise the data analysis pipeline inspecting the quality of both the input data and the results. Additionally, choosing appropriate parameters of a model is quite involved.

Visual support puts the expert into the data analysis loop by providing visual encodings of the data and the analysis results together with interaction facilities. In order to meet the requirements of the experts, the visualizations usually have to be adapted for the application purpose or completely new representations have to be developed. Furthermore, it is necessary to combine these visualizations with the algorithms of the experts to prepare the data. These in-situ visualizations are needed due to the amount of data handled within the analysis pipeline in this domain.

In this thesis, algorithms and visualizations are presented that were developed in two different research areas of computational biology. On the one hand, the multi-replicate peak-caller *Sierra Platinum* was developed, which is capable of predicting significant regions of histone modifications occurring in genomes based on experimentally generated input data. This algorithm can use several input data sets simultaneously to calculate statistically meaningful results. Multiple quality measurements and visualizations were integrated into the data analysis pipeline to support the analyst. Based on these in-situ visualizations, the analyst can modify the parameters of the algorithm to obtain the best results for a given input data set. Furthermore, *Sierra Platinum* and related algorithms were

---

benchmarked against an artificial data set to evaluate the performance under specific conditions of the input data set, e.g., low read quality or undersequenced data. It turned out that Sierra Platinum achieved the best results in every test scenario. Additionally, the performance of Sierra Platinum was evaluated with experimental data confirming existing knowledge. It should be noticed that the results of the other algorithms seemed to contradict this knowledge.

On the other hand, this thesis describes two new visualizations for RNA secondary structures. First, the interactive dot plot viewer *iDotter* is described that is able to visualize RNA secondary structure predictions as a web service. Several interaction techniques were implemented that support the analyst exploring RNA secondary structure dot plots. *iDotter* provides an API to share or archive annotated dot plots. Additionally, the API enables the embedding of *iDotter* in existing data analysis pipelines.

Second, the algorithm *RNApuzzler* is presented that generates (outer-)planar graph drawings for all RNA secondary structure predictions. Previously presented algorithms failed in always producing crossing-free graphs. First, several drawing constraints were derived from the literature. Based on these, the algorithm *RNAturtle* was developed that did not always produced planar drawings. Therefore, some drawing constraints were relaxed and additional drawing constraints were established. Building on these modified constraints, *RNApuzzler* was developed. It takes the drawing generated by *RNAturtle* as an input and resolves the possible intersections of the graph. Due to the resolving mechanism, modified loops can become very large during the intersection resolving step. Therefore, an optimization was developed. During a post-processing step the radii of the heavily modified loops are reduced to a minimum. Based on the constraints and the intersection resolving mechanism, it can be shown that *RNApuzzler* is able to produce planar drawings for any RNA secondary structure. Finally, the results of *RNApuzzler* are compared to other algorithms.



# Acknowledgments

First and foremost, I would like to thank my advisor Dirk Zeckzer. Without your patience and devotion to science, this thesis would not have been possible. Thank you for going this way with me and for always giving me your guidance.

I thank Gerik Scheuermann as my supervisor for his help and advice during difficult times. I am looking forward to the next stage of our journey.

I thank Peter F. Stadler for his ideas and his probably infinite knowledge, which I was privileged to use when needed.

I thank my collaborator Lydia Müller, it is always a fun to do research with you.

I thank my colleagues at the BSV and Bioinf for their feedback and suggestions for improvement.

I thank my numerous bachelor and master students for allowing me to work with them on many of my research ideas.

I thank the secretaries Karin Wenzel and Petra Pregel. No department can do research without a skilled secretary and I was fortunate enough to work with two of them.

Special thanks go to my family. Without your patience and help it would not have been possible to study and write this thesis.

Last but not least, I thank my wife Kim and my children Paulina and Elisa that they always understood long working days and that they always believed that I someday will finish my thesis.



# Contents

<b>1</b>	<b>Preface</b>	<b>1</b>
<b>2</b>	<b>Biological Background</b>	<b>7</b>
2.1	Molecular Biology . . . . .	7
2.2	Epigenetics . . . . .	14
2.3	RNA Secondary Structure Prediction . . . . .	20
<b>3</b>	<b>Sierra Platinum</b>	<b>27</b>
3.1	Introduction . . . . .	28
3.1.1	Goal . . . . .	28
3.1.2	Overview of the Multi-Replicate Peak-Calling Process	28
3.2	Related Work . . . . .	33
3.3	Methods . . . . .	36
3.3.1	Window Construction . . . . .	36
3.3.2	Window Joining . . . . .	42
3.3.3	Read Quality . . . . .	43
3.3.4	Poisson Distribution . . . . .	45
3.3.5	Tag Count Frequencies . . . . .	46
3.3.6	Scaling . . . . .	48
3.3.7	Normalized Poisson Distribution . . . . .	49
3.3.8	Neighborhoods . . . . .	50
3.3.9	Single p-Value . . . . .	52
3.3.10	p2q Transformation . . . . .	55
3.3.11	Significant Windows . . . . .	57
3.3.12	Pearson's Correlation between Replicates . . . . .	59
3.3.13	Combined p-Values . . . . .	63
3.3.14	Filtering and Weighting . . . . .	67
3.3.15	Agreement between the Multi-Replicate Result and the Single Replicate Results . . . . .	69

3.3.16	Computing Peaks . . . . .	70
3.3.17	Computing Peak Quality . . . . .	72
3.3.18	Exporting Peaks . . . . .	74
3.4	Implementation . . . . .	75
3.4.1	System . . . . .	75
3.4.2	Client GUI . . . . .	76
3.4.3	Replicates, Parameters, and Starting Computation . .	77
3.4.4	Quality Control . . . . .	78
3.4.5	Correlation Information, Recalculation Parameters, and Restarting Computation . . . . .	80
3.4.6	Peak Information . . . . .	82
3.4.7	Quality Information . . . . .	82
3.4.8	Additional Functionality . . . . .	83
3.4.9	Server . . . . .	84
3.4.10	Server Configuration File . . . . .	84
3.4.11	Service Implementation . . . . .	85
3.5	Benchmark Data Set . . . . .	89
3.5.1	Context . . . . .	89
3.5.2	State-of-the-Art and Gaps . . . . .	89
3.5.3	Goal . . . . .	90
3.5.4	Challenges . . . . .	91
3.5.5	Benchmarking Data Set Creation . . . . .	91
3.5.6	Benchmarking Replicates . . . . .	94
3.5.7	Benchmarking Data Sets . . . . .	95
3.6	Statistical Measures for Quality Assessment . . . . .	96
3.7	Evaluation . . . . .	98
3.7.1	Introduction . . . . .	98
3.7.2	Sierra Platinum Quality Measures and Visualizations .	98
3.7.3	Parameter Selection . . . . .	107
3.7.4	Approach Comparision . . . . .	115
3.8	Results . . . . .	124
3.8.1	Real World Data Sets . . . . .	124
3.8.2	Peak Agreement . . . . .	125
3.8.3	Peak-Calls . . . . .	126
3.8.4	Hox-C and Hox-D Clusters . . . . .	133
3.8.5	Peak Coverage Analysis . . . . .	138

<b>4</b>	<b>iDotter</b>	<b>141</b>
4.1	Introduction . . . . .	142
4.2	Background and Related Work . . . . .	143
4.3	Problem Statement: Current State and Issues . . . . .	145
4.4	Solution . . . . .	145
4.4.1	Data Import . . . . .	148
4.4.2	Visualization . . . . .	148
4.4.3	Interaction . . . . .	149
4.4.4	API . . . . .	151
4.4.5	Implementation . . . . .	151
4.5	Results . . . . .	154
4.6	Use Case . . . . .	154
<b>5</b>	<b>RNApuzzler</b>	<b>157</b>
5.1	Introduction . . . . .	158
5.2	Theory . . . . .	159
5.2.1	Directed Rooted Trees . . . . .	159
5.2.2	RNA Secondary Structure . . . . .	161
5.2.3	RNA-Trees . . . . .	164
5.3	Method RNAturtle . . . . .	169
5.3.1	Drawing Constraints . . . . .	169
5.3.2	Turtle Algorithm . . . . .	170
5.4	Method RNApuzzler . . . . .	173
5.4.1	Drawing Constraints . . . . .	173
5.4.2	Coordinate transformation and data structures . . . . .	174
5.4.3	Main Algorithm . . . . .	174
5.5	Optimization . . . . .	187
5.6	Benchmarks . . . . .	191
5.7	Results . . . . .	192
5.8	Comparison to other algorithms . . . . .	193
5.8.1	Comparison to NAView . . . . .	193
5.8.2	Comparison to other tools . . . . .	196
<b>6</b>	<b>Conclusion</b>	<b>209</b>
	<b>List of Figures</b>	<b>IX</b>
	<b>List of Tables</b>	<b>XX</b>

**7 Curriculum Scientiae**

**XXIII**

# Chapter 1

## Preface

*Verba docent, exempla trahunt.*

In the last years, the amount of available data in the field of computational biology steadily increased. In order to be able to analyze these data, various algorithms have been developed by bioinformaticians to process them efficiently. Moreover, computational models were developed to predict for instance biological relationships of species. Furthermore, the prediction of properties like the structure of certain biological molecules is modeled by complex algorithms. Despite these advances in handling such complicated tasks with automated workflows and a huge variety of freely available tools, the expert still needs to supervise the data analysis pipeline inspecting the quality of both the input data and the results. Additionally, choosing appropriate parameters of a model is quite involved.

Visual support puts the expert into the data analysis loop by providing visual encodings of the data and the analysis results together with interaction facilities. In order to meet the requirements of the experts, the visualizations usually have to be adapted for the application purpose or completely new representations have to be developed. Furthermore, it is necessary to combine these visualizations with the algorithms of the experts to prepare the data. These in-situ visualizations are needed due to the amount of data handled within the analysis pipeline in this domain.

In this thesis, algorithms and visualizations are presented that were developed in two different research areas of computational biology. On the one hand, a new multi-replicate peak-caller was developed, which is capable of predicting significant regions of histone modifications occurring in genomes based on experimentally generated input data. This algorithm can use several input data

sets simultaneously to calculate statistically meaningful results. On the other hand, this thesis describes two new visualizations for RNA secondary structures. First, an interactive dot plot viewer is described that is able to visualize RNA secondary structure predictions as a web service. Second, a new algorithm is presented that generates (outer-)planar graph drawings for all RNA secondary structure predictions. Previously presented algorithms failed in *always* producing *crossing-free* graphs.

The thesis starts with an introduction of the biological background (Chapter 2). The first section of this chapter gives an overview of DNA, RNA, and proteins (Section 2.1) and their interaction. It is followed by an overview of the research area Epigenetics, where the histone modifications are described in detail (Section 2.2). Finally, the basics of RNA secondary structure prediction algorithms that are able to generate the necessary input data for the visualization systems for RNA secondary structures are introduced (Section 2.3).

The chapter describing the new peak-caller 'Sierra Platinum' (Chapter 3) is mainly based on the following publications:

**Sierra platinum: a fast and robust peak-caller for replicated ChIP-seq experiments with visual quality-control and-steering:** Lydia Müller★, Daniel Gerighausen★, Mariam Farman, Dirk Zeckzer. *BMC Bioinformatics*, 2016. doi:10.1186/s12859-016-1248-6

**The Sierra Platinum Service for generating peak-calls for replicated ChIP-seq experiments:** Daniel Wiegrefe★, Lydia Müller★, Jens Steuck, Dirk Zeckzer, Peter F. Stadler. *BMC Research Notes*, 2018. doi:10.1186/s13104-018-3633-x

★ equally contributed

The chapter starts with an introduction of the peak-calling process of histone modifications and the goals of the new algorithm *Sierra Platinum* (Section 3.1). Then, a general overview of the analysis pipeline is given. In the following, related algorithms are presented and compared to the Sierra Platinum approach (Section 3.2). Afterwards, the methodology of the multi-replicate peak-caller is described in detail (Section 3.3). Moreover, the algorithms and visualizations of each pipeline step are described (Section 3.3). If it was possible to optimize an analysis step, those optimizations are discussed in detail. Then,



the implementation of Sierra Platinum and its usage to generate significant peak-callings are described (Section 3.4). Furthermore, the service implementation of Sierra Platinum is presented (Section 3.4.11).

To evidence the usefulness and usability of Sierra Platinum, an artificial benchmark data set that can be used to evaluate the performance of peak-callers was developed (Section 3.5). Therefore (Section 3.5.7), several data sets were created to simulating a variety of common quality issues of experimentally created input data sets for histone peak-callings. Together with statistical measures (Section 3.6) these benchmarks were used to evaluate the overall performance of Sierra Platinum. First, the quality measurements of Sierra Platinum are tested against the different data sets and it is discussed how the provided visualizations encode the errors of these data sets. Next, the parameter settings of Sierra Platinum are evaluated and general recommendations for using it are provided. Finally, the performance of Sierra Platinum is tested against other peak-callers and it is shown that Sierra Platinum performs better than the others in any test scenario.

Finally, results for experimental data using the Sierra Platinum peak-caller are presented (Section 3.8). First, the data sets and their pre-processing are introduced. Next, the methodology used to calculate the agreement of the results of different peak-calling tools is described, since the detected regions can be slightly different due to the different methods applied by the peak-callers. If the peaks overlap up to certain threshold, one can suppose that these results are similar. After that, the results of the comparison are presented. Then, the results are analyzed at specific loci of the human genome. For these regions biological experts provided background knowledge how the peak-callings should look like and the results are verified against this knowledge.

The chapter describing the interactive dot plot viewer (Chapter 4) is mainly based on the following publication:

**iDotter - an interactive dot plot viewer:** Daniel Gerighausen, Alrik Hausdorf, Sebastian Zänker, Dirk Zeckzer. 25. *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision WSCG*, 2017

This chapter starts with a brief introduction to motivate the necessity to develop a new dot plot viewer for RNA secondary structures. Furthermore, it describes what kind of interactions were added to the visualization system to

enhance it. Followed by that, the functions of related tools and visual representations of RNA secondary structures are discussed (Section 4.2). Based on that, the current state and the issues of current dot plot viewers for RNA secondary structures are elaborated upon (Section 4.3). Then, a solution for the mentioned issues is proposed and the newly developed interactive dot plot viewer *iDotter* is described (Section 4.4). Within this section, the complete data analysis pipeline is described and the features of the visualization system are presented in detail. *iDotter* is implemented as freely available web service to facilitate the usage of this tool. Finally, this chapter concludes with a discussion of the results produced by *iDotter* (Section 4.5) and an use case showing how to interpret RNA secondary structure dot plots (Section 4.6).

The subsequent chapter (Chapter 5) continues thematically in the domain of RNA secondary structure visualizations, addressing planar graph layouts of such structures. It is mainly based on the following publication:

**RNApuzzler: efficient outerplanar drawing of RNA-secondary structures:** Daniel Wiegreffe, Daniel Alexander, Peter F. Stadler, Dirk Zeckzer. *Bioinformatics*, 2018. doi:10.1093/bioinformatics/bty817

This chapter starts with an extensive introduction of the theoretical backgrounds of the graph representation of RNA secondary structures (Section 5.2). The theoretical background begins with a collection of definitions related to graphs that are used thereafter to describe the graph layout. Then, a formal definition of RNA and its secondary structure is provided as well as its interpretation as an RNA-Tree. Derived from these definitions, specific properties of the RNA-Tree are postulated, which will then be proven. Based on these properties, a new algorithm was developed to visualize RNA secondary structures as a graph that is provably planar for any RNA secondary structure. This algorithm consists of two principle components. The first component is called *RNAturtle* (Section 5.3). First, a collection of drawing constraints of RNA secondary structure graph layouts is described and matched with the related literature. These do not yet include planarity of the resulting drawing. Afterwards, the *RNAturtle* algorithm is described. It is based on a turtle-graphics algorithm and therefore, the RNA structure is drawn sequentially by iterating over all nucleotides of its sequence. The algorithm is evaluated against the previously postulated drawing constraints. Despite fulfilling most of the constraints with every possible structure, *RNAturtle* may not create planar drawings, especially

for large RNAs. Therefore, based on this result, RNApuzzler was developed (Section 5.4). First, the drawing constraints of RNAturtle are analyzed. Based on this, new constraints were added and some existing constraints were relaxed to enable RNApuzzler to create planar drawings. Furthermore, a more coarse grained data structure for the RNA-Tree is proposed. Each of the elements in this data structure represents a vertex (structural element: loop) and its incoming edge (structural element: stem). If a stem is connected to another loop segment, an edge between those two elements represents this connection. After that, the main algorithm of RNApuzzler is described (Section 5.4.3). Basically, RNApuzzler checks the output of RNAturtle for intersections within the graph layout and resolves them. To achieve this, RNApuzzler checks three different properties of the planarity in succession. As it was proven (Section 5.2), these properties build upon each other and if all of them are fulfilled, the whole graph layout is provably planar. If an intersection is detected during these tests, the algorithm resolves it and continues the processing. Based on the resolving mechanism, the loops of the RNA structure may become very large since their radii are increased during the resolving step. Therefore, an optimization step was developed that reduces the effects of the loop increase regarding drawing space and readability of the visualization (Section 5.5). Then, the benchmarks of RNApuzzler and RNAturtle are discussed (Section 5.6). It is shown that RNApuzzler fulfills all proposed drawing constraints at least in their relaxed form and compared to other drawing algorithms it is the only known algorithm that produces planar drawings for every possible RNA secondary structure. Furthermore, the computational time of RNApuzzler was evaluated. RNApuzzler needs only a fraction of the computational time that is needed to predict the folding of the structure but it is slower than the previous (non-planar) standard visualization algorithm, especially for large RNA structures. Finally, RNApuzzler was compared to existing drawing algorithms with respect to readability of the layout.

The last chapter summarizes the results of this thesis and an outlook for further research is given (Chapter 6).



# Chapter 2

## Biological Background

### 2.1 Molecular Biology

Every living cell has basically the same setup where molecules called proteins manage the cell, DNA (deoxy-ribonucleic acid) stores the construction plan for the proteins, and RNA (ribonucleic acid) transmits messages between the DNA and the proteins.

#### DNA

In eukaryotes, the DNA is stored inside the nucleus (see Figure 2.1) of the cell and it is organized in subunits, called chromosomes. Human DNA, for example, consists of 46 chromosomes. The DNA sequence is built up from four nucleotides: deoxyadenosine monophosphate (A), deoxyguanosine monophosphate (G), deoxycytidine monophosphate (C), and thymidine monophosphate (T). Each of them have a 2-deoxyribose sugar and a phosphate group is bonded to this sugar. Additionally, one of the four nucleobases adenine, guanine, cytosine, or thymine is bonded to the sugar. Altogether, each combination of sugar, phosphate, and one of the nucleobases forms a so-called nucleotide. These nucleotides can form a chain: a single strand of DNA (see Figure 2.2). Furthermore, the DNA can form a double strand with base pairs of nucleotides. As Watson and Crick [82] proposed, DNA forms only specific base pairs, the so-called *Watson Crick base pairs*: G-C and A-T (see Figure 2.3). The double stranded structure forms the well known double helical structure of the DNA like shown in Figure 2.1.

The DNA is divided into so-called *coding* and *non-coding* regions. In the coding regions, information is stored that can be used to generate proteins.

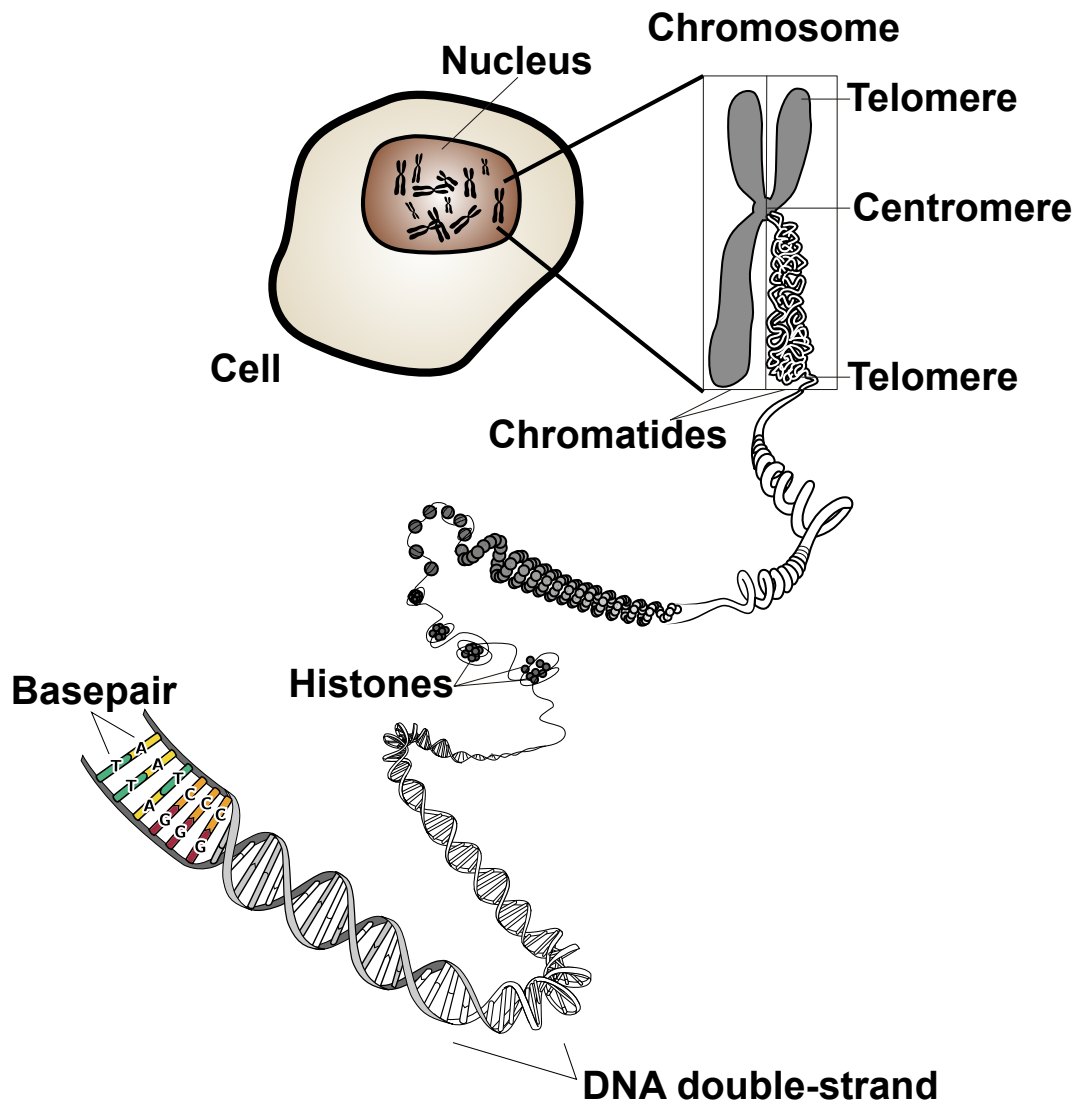


Figure 2.1: DNA and its structural organization in the nucleus of eukaryotic cells [1].

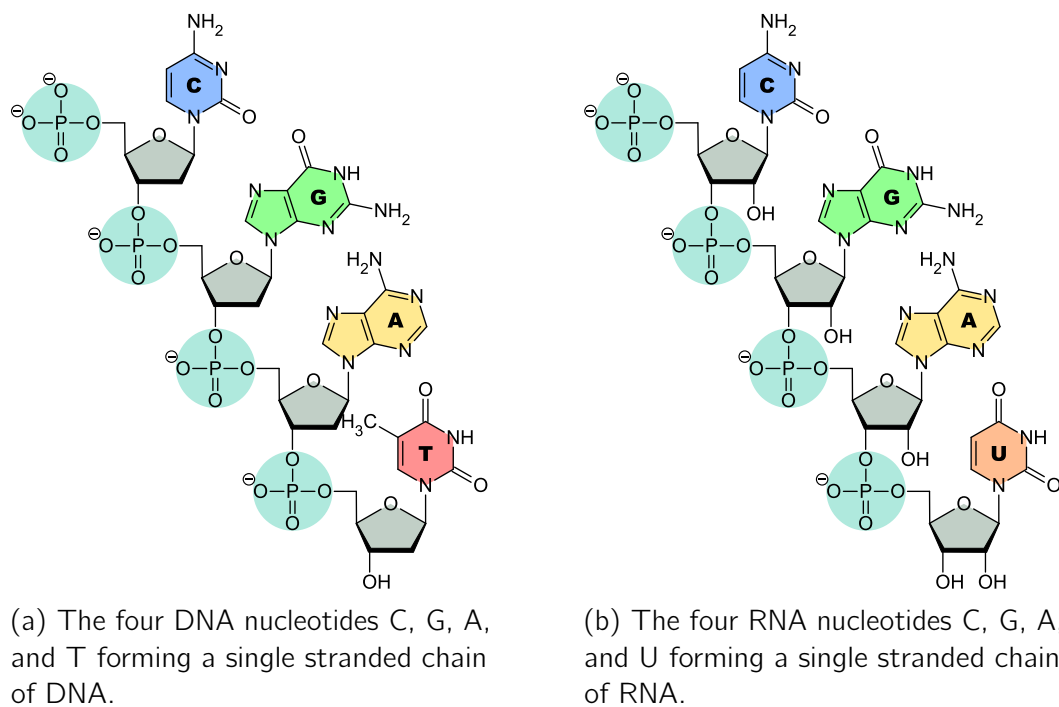


Figure 2.2: Single stranded pieces of DNA and RNA.

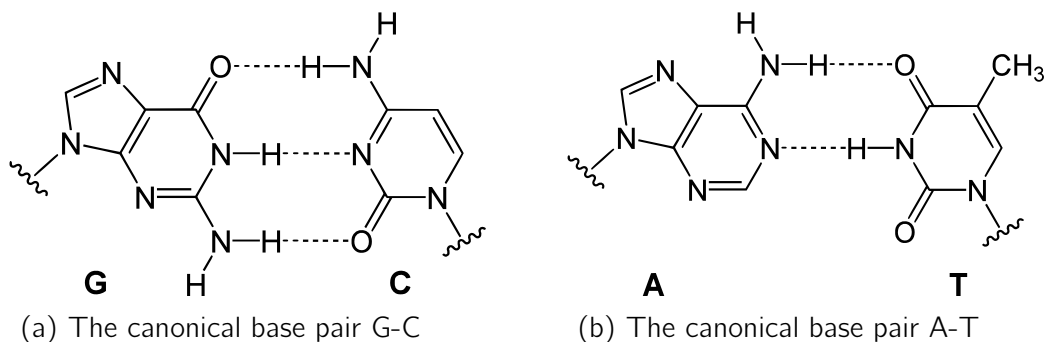


Figure 2.3: Watson Crick base pairs

Which protein is created depends on the order of the nucleotides in the respective DNA sequence. The non-coding regions can not be used to generate proteins. However, the DNA has regions that do not encode information for proteins, the *non-coding regions*. Nevertheless, these regions can be transcribed to RNAs and also have a function. Figure 2.5 shows an overview of transcription and translation. Functional regions of the DNA are called *genes*.

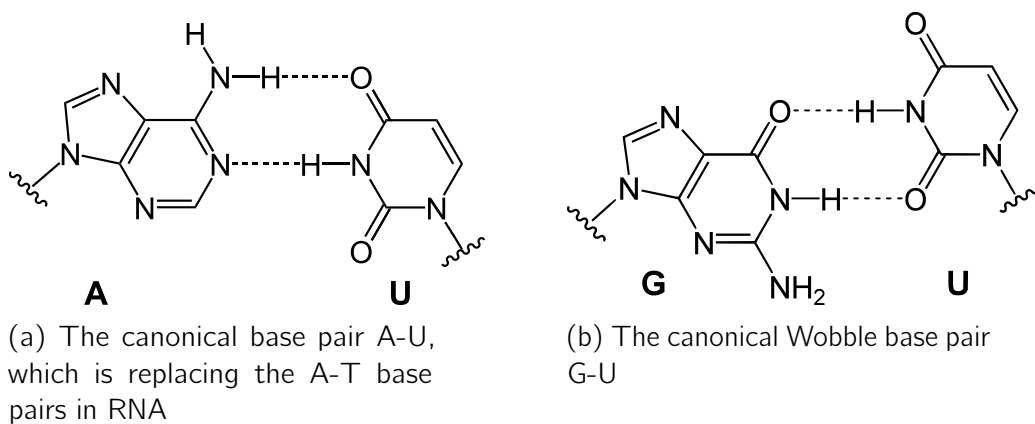


Figure 2.4: Additional possible base pairs occurring in RNA molecules.

## RNA

RNA molecules (Figure 2.2) are rather similar to DNA molecules (see Figure 2.2a). They are built up from a ribose sugar (compared to the desoxyribose sugar of the DNA, a hydroxyl group is added to the sugar at the 2' position of the ring). Additionally, one nucleobase is exchanged in RNA: RNA uses uracil instead of thymine. This base has an extra methyl group ( $H_3C$  in Figure 2.4) and like thymine forming a base pair with adenine. RNA molecules can also form a chain of molecules but they do not form the double stranded helical structures. Nevertheless, it can form a single stranded helical structure by complementary base pairings. This structure is called the RNA secondary structure and is described in more detail in Section 2.3. Besides Watson Crick base pairs, RNA can form additional base pairs like the *Wobble pair* G-U (see Figure 2.4). Altogether, Watson Crick and Wobble pairs [28] form the *canonical base pairs*. Additionally, even more base pair types exist in RNA, the so-called *non-canonical* base pairs. However, these pairings are not treated as secondary structures.

RNA molecules have different functions in the cell. Best known are the coding RNAs, that serve as intermediate messenger (mRNA) between the DNA and proteins. In addition to that, several other RNA types are known that do not serve as a messenger, the so-called non-coding RNAs (ncRNA). Most prominently are the groups of transfer RNAs (tRNA) and ribosomal RNAs (rRNA). They act as assistants and regulators during the translation of mRNAs into proteins. Besides those, there are many more ncRNA types.



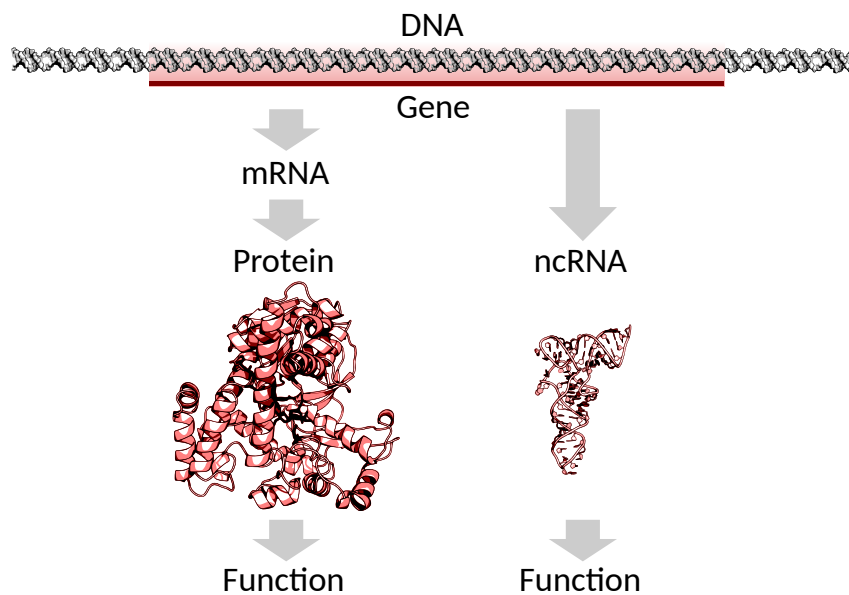


Figure 2.5: From a gene to function: either a region of the DNA is transcribed to an mRNA, which is translated into a functional protein or the DNA is transcribed into a functional ncRNA [9].

## Proteins

Proteins, on the other hand, are not built up from nucleotides but rather from amino acids. Amino acids are a composition of amine and carboxyl groups together with a specific side chain that determines the amino acid. Amino acids can also form chains that build up proteins. Similar to DNA and RNA, amino acids can establish additional pairings within a chain of amino acids and interact with other amino acid chains or other molecules forming large macromolecules. There are a multitude of different amino acids, but in known life only 22 variants are translated. These amino acids are called *proteinogenic amino acids*. From these 22 amino acids, 20 can be synthesized by translation, the remaining 2 are synthesized by other processes [16]. As shown in Figure 2.5, mRNAs are translated into proteins within a cell. This is done with the use of macromolecules called ribosomes. During the translation, a ribosome assembles the chain of amino acids with a specific order provided by the mRNA. Each triplet of nucleotides of the mRNA, a so-called codon, represents a specific amino acid and the ribosome reads the mRNA and forms the protein. Furthermore, mRNAs have a specific start and a specific stop codon for controlling the translation. Figure 2.6 provides the codons for the 20 common amino acids.

The transfer of information from DNA over RNA to protein is described in the *Central Dogma of Molecular Biology* [30, 29]. As shown in Figure 2.7,

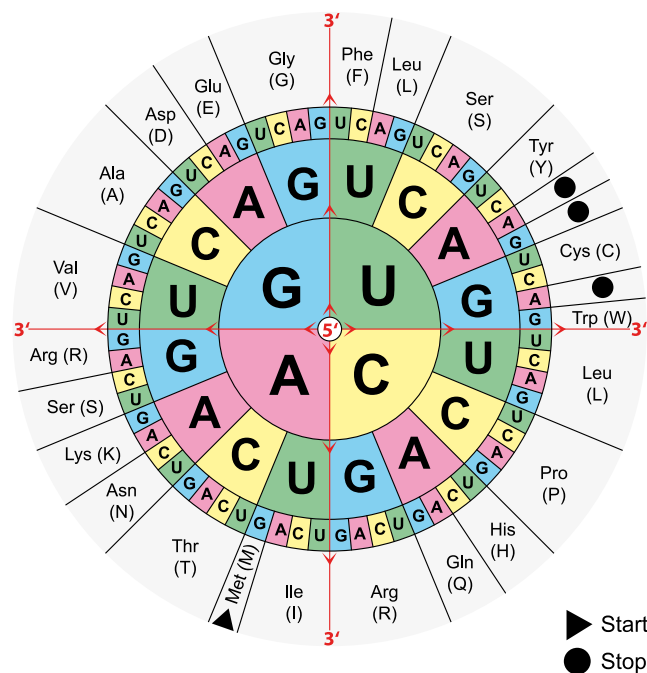


Figure 2.6: Overview of all amino acids that are encoded by codons. The visualization is read from the center of the circle outwards so that each sequence of nucleotides results in an amino acid or a start/stop codon.

the arrows show the direction of information transfer between DNA, RNA and proteins. The solid arrows encode information transfer that were included into the original model and describe the common information flow. However, the original model was not complete, e.g., RNA can be reversely transcribed to DNA. This mechanism is used by some viruses to transfer new information from their RNA to the host genome. Therefore, dashed arrows showing additional known information transfers were added to the original model.

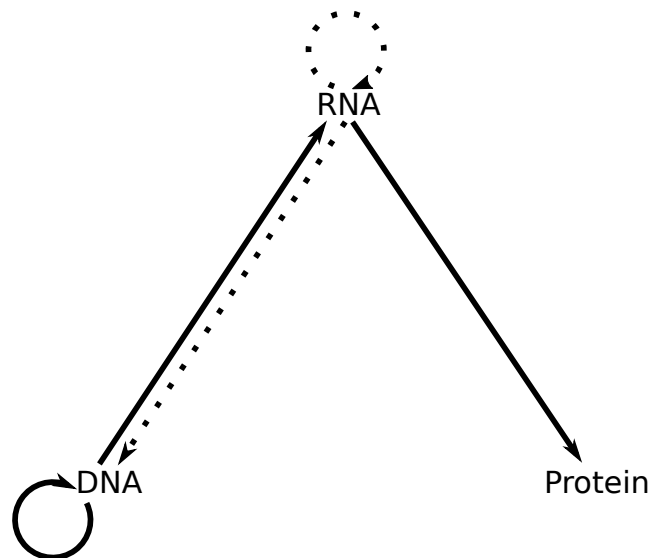


Figure 2.7: The *Central Dogma of Molecular Biology* proposed by Crick [30, 29]. The solid arrows show the common information flow and the dashed arrows specialized transfer directions.

## 2.2 Epigenetics

All cells of a multicellular organism share the same genetic information encoded as DNA and thus the same set of genes. However, different cell types have different functions and produce different transcripts of RNA by regulation of gene expression. Beside the regulation by transcription factors and RNAs, *epigenetics* was established as an additional research field in the domain of gene regulation. Epigenetics is defined as "the study of heritable changes in gene expression that are not mediated at the DNA sequence level" [27].

### Nucleosome

As shown in Figure 2.1, the DNA of eukaryotes is located in the nucleus of the cell and is partitioned in structural units called chromosomes. The DNA is wrapped around proteins and forms a "beads on a string" like structure. Together, they form the chromatin, which combine into the chromosome. In detail, each "bead" is made up from two copies of four different histones: H3, H4, H2A, and H2B. This protein complex is called *histone octamer* and with 146bp of DNA wrapped around it, it forms a *nucleosome* (see Figure 2.8). An additional histone H1 is attached to the open end of each nucleosome to stabilize the linker between two nucleosomes [21]. The chromatin can be packed differently tight, for example the Centromeres shown in Figure 2.1 are packed very tightly. These areas can not be accessed easily by the RNA-polymerase and therefore this effect decreases expression of the genes located in this region. Condensed regions of chromatin are called *heterochromatin* [27]. Additionally, the chromatin can be packed less tightly, and therefore easily can be accessed by the RNA-polymerase. These regions which are often highly transcribed are called *euchromatin* [27]. The histones of the octamer are modified by enzymes to form these regions. These enzymes attach specific molecules to the histones at certain amino acids. Due to this modification of the histone, the nucleosome can be packed more tightly or less tightly within the chromatin.

### Histone modifications

Many histone modifications with different effects on the chromatin organization are known (see Figure 2.10). Most common are acetylation, methylation, phosphorylation, and ubiquitination, which are all attached to so-called histone tails. As shown in Figure 2.9c, the histone complex is not covered completely

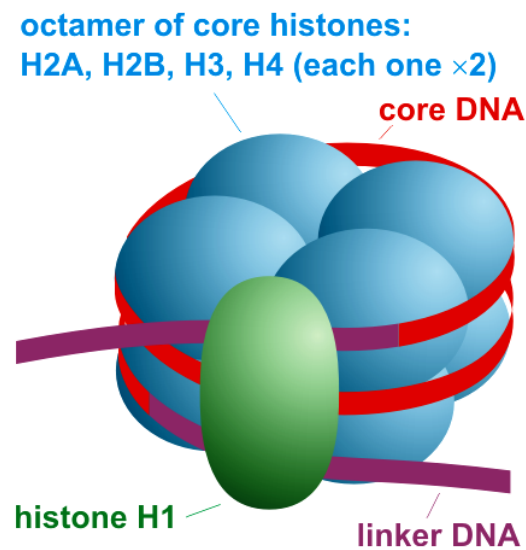


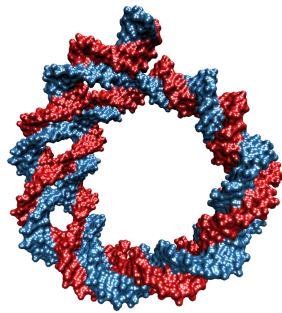
Figure 2.8: Schematic structure of a nucleosome [13].

by the DNA which is wrapped around it. The histone tails form the outside accessible regions that can be modified by enzymes. Each modification can have a different impact on the chromatin organization. Additionally, multiple modifications can occur at the same octamer and can have additional effects on the chromatin organization. For example, the histone H3 contains 8 lysine amino acids and a methylation group can be attached to most of them. The positions of the lysines are shown in Figure 2.11a. Currently, the effects of histone modifications are still a vivid subject of research.

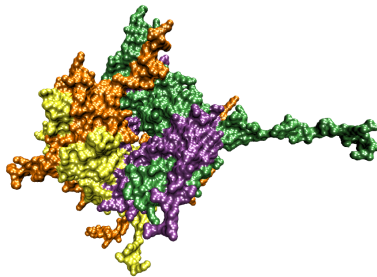
Due to the variety of histone modifications, the position and the type of a modification is described using a naming pattern. For example, the trimethylation ('me3') of the fourth lysine (biological abbreviation 'K') of the histone H3 is named 'H3K4me3'. In this thesis, the modifications H3K4me3, H3K27me3, H3K9me3, H3K27ac, and H3K9ac are analyzed:

**H3K4me3** This modification is positively correlated with transcription [71] and it occurs at tissue specific genes as well as so-called housekeeping genes that are necessary for maintaining a cell and its basic functions. Furthermore, it is often found in embryonic stem cells.

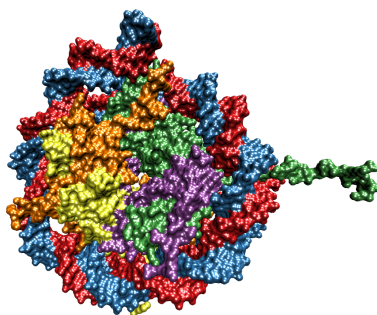
**H3K27me3** This modification is related to heterochromatin [25] and therefore it is correlated with the downregulation of nearby genes.



(a) The DNA component of the nucleosome (shown without the histone complex).



(b) The histone complex: H3 is colored in green, H2A in orange, H2B in yellow, and H4 in purple (shown without the DNA). The histone tail of the histone H3 can clearly be seen.



(c) DNA and the histone complex form the nucleosome.

Figure 2.9: A nucleosome divided into its components. This model was generated from the crystal structure published by Luger et al. [55].

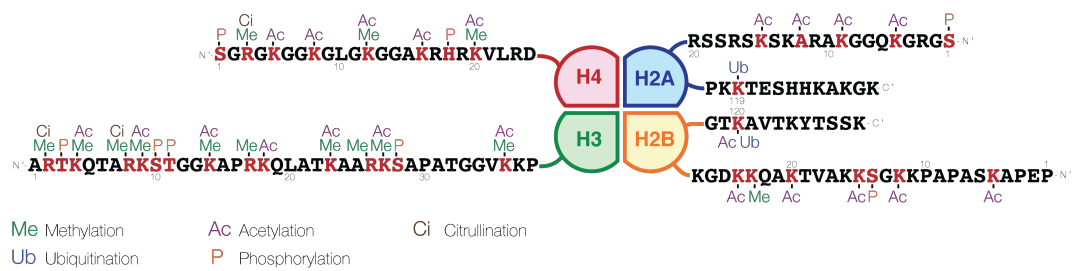


Figure 2.10: Overview of known histone modifications for the histones H4, H2A, H3, and H2B [14].

**H3K9me3** This modification acts as a repressor of transcription and it is correlated with DNA methylation that silences the DNA so it can not be expressed anymore [58].

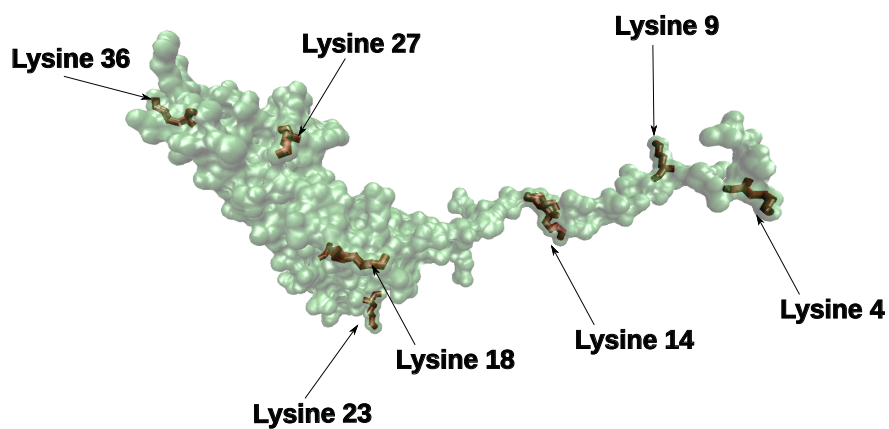
**H3K27ac** This modification is found predominantly in promotor regions and like H3K4me3 is positively correlated with transcription of nearby genes [80].

**H3K9ac** This modification is known to ease the packing density of a nucleosome and it occurs often at promotor regions [62]. It is correlated positively with the activation of nearby genes.

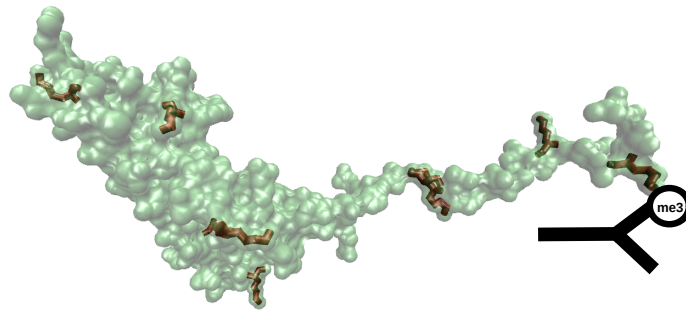
## Sequencing histone modifications

The most common method to measure histone modification is the chromatin immunoprecipitation sequencing (ChIP-seq) protocol. In a first step, the DNA is cross-linked with the histones that are bound to it. Afterwards, the DNA is fragmented into pieces having a size between 200 and 1000 bases by sonification. Then, specific antibodies are added to target a specific histone modification like H3K4me3. If an antibody encounters the targeted modification at a fragment, it binds to the modification as shown in Figure 2.11b. Next, all fragments that are not targeted by an antibody are removed by washing them away. Then, the cross-linking of the DNA with the histones is dissolved. After the dissolution, the histones are washed away, too. Through this procedure, all unmodified regions of the DNA are separated away. In the next step, the filtered DNA fragments are sequenced.

Since histone modifications are distributed over the genome of an organism and histone modifications occur frequently, it is recommended to use a next-generation sequencing (NGS) method. A common NGS method is the Illumina sequencing method. During this method, the DNA fragments are splitted



(a) Position of all modifiable lysines of the histone H3.



(b) An antibody binds to its target site (the histone modification H3K4me3).

Figure 2.11: A model of the histone H3 generated from the crystal structure published by Luger et al. [55].



into single strands that are amplified on a flow cell. After the amplification, the fragments form so-called DNA-clusters that should contain only DNA fragments, which were created by the amplification of a single DNA fragment from the previous step. In the following, the sequencing is performed with the help of modified nucleotides. These nucleotides are fluorescently-labeled to red and green laser light and are added step by step onto the single stranded DNA-clusters. After each step, two lasers emit their specific red and green light onto the DNA-clusters. A camera detects the specific reaction of each modified nucleotide. Since A and C are excited only by the red laser and G and T only by the green one, the camera needs to take four pictures after each step. The camera uses different filters to detect the small variations in the spectra emitted by A and C as well as G and T, respectively. Since the DNA-clusters contain many identical fragments, the emitted light is strong enough for a camera to capture it. Afterwards, a new nucleotide is added to the fragment until the process iterates over the whole fragment.

This method is fast but errors during the amplification are distributed exponentially within the DNA-cluster. Furthermore, the distinction between the different spectra of A and C as well as G and T can fail and produce errors. Therefore, it is recommended to sequence each experiment with a higher coverage to detect possible errors in the post-processing steps.

With this method, it is possible to sequence single histone modifications. To detect several histone modifications, it is necessary to repeat the process with specific antibodies for each histone modification. Furthermore, an unspecific antibody is used during an additional run to create a so-called background measurement. This background measurement is used during the post-processing steps to normalize the data and to reduce the noise caused by errors during the sequencing step.

## 2.3 RNA Secondary Structure Prediction

As mentioned before, RNA can form a single stranded helical structure, the *RNA Secondary Structure*. Since an RNA sequence is often not completely complementary, not all nucleotides of a sequence form a base pair and base pairs can not cross each other. The unpaired regions of an RNA secondary structure are called *loops*. Every paired region of an RNA sequence is called *stem*. Furthermore, one can distinguish between different types of loops. An *internal loop* is a region that is connected to two stems. A special case of the internal loop is the so-called *bulge* since only one side of the helical structure has unpaired nucleotides. A loop with only one stem attached to it is called *hairpin loop*. Additionally, a loop with more than two stems attached is called *multi-loop*. A special case is the unpaired region at the start and at the end of the RNA sequence. This region is called *exterior loop*. A formal description of these elements of the RNA secondary structure is given in Section 5.2. Figure 2.12 shows the different elements of an RNA secondary structure.

### Nussinov Algorithm

Predicting an optimal RNA secondary structure is called RNA folding and is still an ongoing research topic since the 1970s. Nussinov et al. [63] published the first algorithm to predict an optimal structure in which the number of base pairs in a given sequence has been maximized. Due to the fact that base pairs do not cross each other in a valid RNA secondary structure, every sequence can be decomposed into subsequences and the optimal substructures can be calculated recursively (Figure 2.13). Using this decomposition the maximal number of base pairings can be calculated using a dynamic programming approach. Therefore, a forward recursion is applied to fill a matrix  $E_{n, n}$ , which holds all possible substructures (see Algorithm 2.1). After this, the backward recursion is started at  $E_{1, n}$  to trace back the maximal number of possible base pairings for the given sequence. Since it can happen that a maximal number of possible base pairings can be formed by multiple valid structures, multiple equally optimized paths can be found during the backward step.

## Algorithm 2.1

Initialization:

$$\begin{aligned} E_{i, i} &= 0 \\ E_{i, i-1} &= 0 \end{aligned}$$

Forward Recursion:

$$E_{i, j} = \max \begin{cases} E_{i+1, j} \\ \max(E_{i+1, k-1} + E_{k+1, j} + S(i, k)) \end{cases} \quad (2.1)$$

where

$$S(i, k) = \begin{cases} 1, & \text{if } i \text{ and } k \text{ form a canonical base pair.} \\ 0, & \text{else} \end{cases}$$

and

$$i < k \leq j$$

## Nearest Neighbor Model

The Nussinov Algorithm is a simple algorithm since it takes neither the different binding energies of canonical base pairs nor the stacking effects of stems into account. Therefore, a more complex algorithm is usually used to predict a more realistic RNA secondary structure. It is based on thermodynamic principles and experimental observations to model a secondary structure that maximizes the free energy of the structure of the given sequence. For this purpose, the structure is decomposed into its elementary substructures such as stems, hairpin loops, internal loops, and multi-loops. Each of these elements adds a certain amount of free energy to the structure, which is summed up for the given sequence to find an optimal structure. Since it is not feasible to validate every possible substructure, only a large amount of smaller substructures was experimentally analyzed. The energy of larger structures is computationally derived. Therefore, a *nearest neighbor energy model* [72] is used. The basic

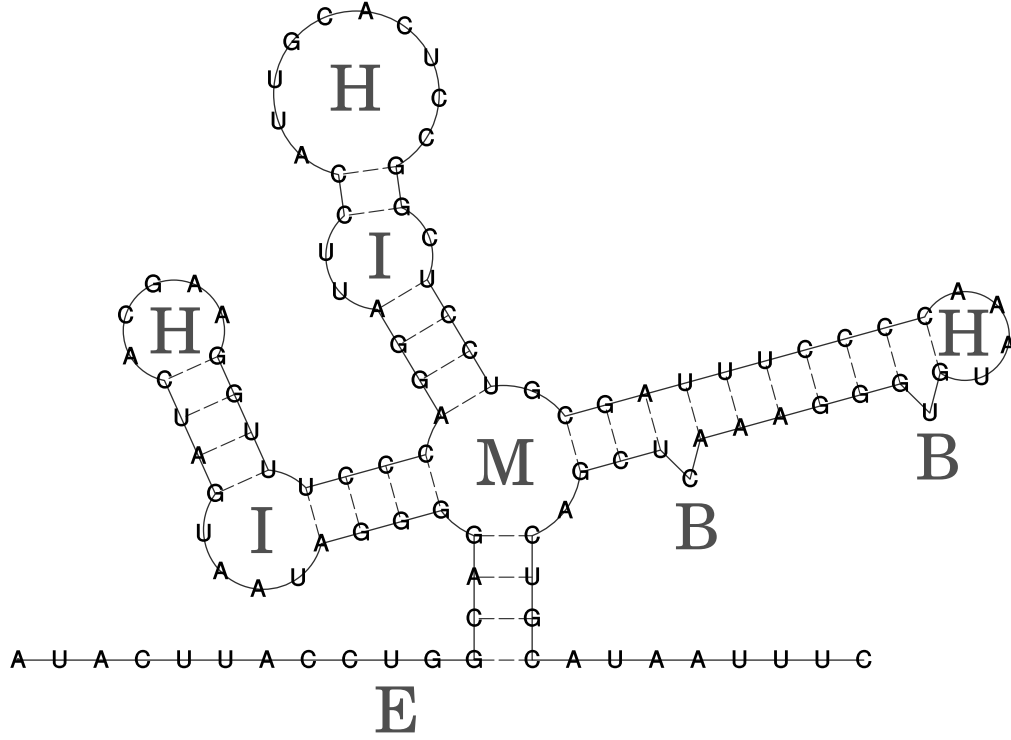


Figure 2.12: Overview of the substructures of an RNA secondary structure. E: exterior loop, I & B: internal loop, M: multi-loop, H: hairpin loop.

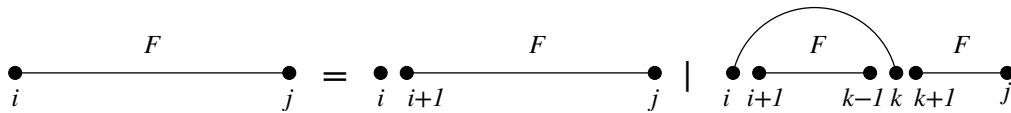


Figure 2.13: Decomposing an RNA secondary structure into substructures. The folding space  $F$  of the nucleotides  $i$  and  $j$  can be decomposed into a subfolding space of  $i+1$  and  $j$ , or into a bifurcation from  $i+1$  to  $k-1$  and from  $k+1$  to  $j$ , where  $i < k \leq j$ , and  $i$  and  $k$  form a base pair.

idea of this model is that only the base and its direct neighboring bases influence the free energy of a base pair. With this model, it is possible to break down the global free energy of a structure into multiple contributions of the substructures.

## Zuker Algorithm

Unfortunately, the nearest neighbor energy model impedes the prediction of multi-loops, since an unknown number of components can be attached to it. But since the number of components of a multi-loop influences the energy contribution, an accurate estimation of them is necessary. Therefore, Zuker published a customized variant [92, 91] of the Nussinov Algorithm by adding additional backtracking matrices to distinguish between the substructures of the RNA. Algorithm 2.2 denotes the Zuker algorithm in a version proposed by Hofacker et al. [43]. Many RNA folding software systems like the ViennaRNA Package [54] and RNAstructure [66] implemented this algorithm and developed it further. In general, the algorithm decomposes the structure into substructures and tracks the number of components of a multi-loop. Therefore, four tables track the optimal free energy for the substructures. In detail,  $F_{i,j}$  stores the optimal free energy of the substructure between  $i$  and  $j$ .  $C_{i,j}$  contains the optimal free energy for the substructure between  $i$  and  $j$ , if  $i$  and  $j$  form a base pair. Multi-loop substructures are stored in two tables.  $M_{i,j}$  stores the optimal free energy of a substructure between  $i$  and  $j$ , if the substructure is part of a multi-loop and has at least one component attached to it. The second table  $M_{i,j}^1$  holds the optimal free energy of a substructure between  $i$  and  $j$ , if the substructure is part of a multi-loop and has exactly one component attached to it. This second table for the multi-loop is not necessary to find an optimal solution but it accelerates the computation of multiple sub-optimal RNA foldings which is a common task in the area of RNA folding. The functions  $\mathcal{H}(i, j)$  and  $\mathcal{I}(i, j)$  (Algorithm 2.2) calculate the optimal free energy for a hairpin loop and an interior loop, respectively. Additionally, multiple energy constants are added to model the effects of base pairs. The backtracking step of this algorithm is applied just as the backtracking step of the Nussinov algorithm explained before.

## Algorithm 2.2

$$\begin{aligned}
F_{i,j} &= \min \begin{cases} F_{i+1,j} \\ \min_{i < k \leq j} (C_{i,k} + F_{k+1,j}) \end{cases} \\
C_{i,j} &= \min \begin{cases} \mathcal{H}(i,j) \\ \min_{i < k < l < j} (C_{k,l} + \mathcal{I}(i,j:k,l)) \\ \min_{i < u < j} (M_{i+1,u} + M_{u+1,j-1}^1 + a) \end{cases} \\
M(i,j) &= \min \begin{cases} \min_{i < u < j} ((u-i+1) \cdot c + C_{u+1,j} + b) \\ \min_{i < u < j} (M_{i,u} + C_{u+1,j} + b) \\ M_{i,j-1} + c \end{cases} \quad (2.2) \\
M_{i,j}^1 &= \min \begin{cases} M_{i,j-1}^1 + c \\ C_{i,j} + b \end{cases} \\
&\text{with}
\end{aligned}$$

$\mathcal{H}(i, j)$  = energy for a hairpin loop

$\mathcal{I}(i, j : k, l)$  = energy for an internal loop

$a$  = energy penalty for a closing base pair in a multi-loop

$b$  = energy penalty for a base pair in a multi-loop

$c$  = energy penalty for an unpaired base in a multi-loop

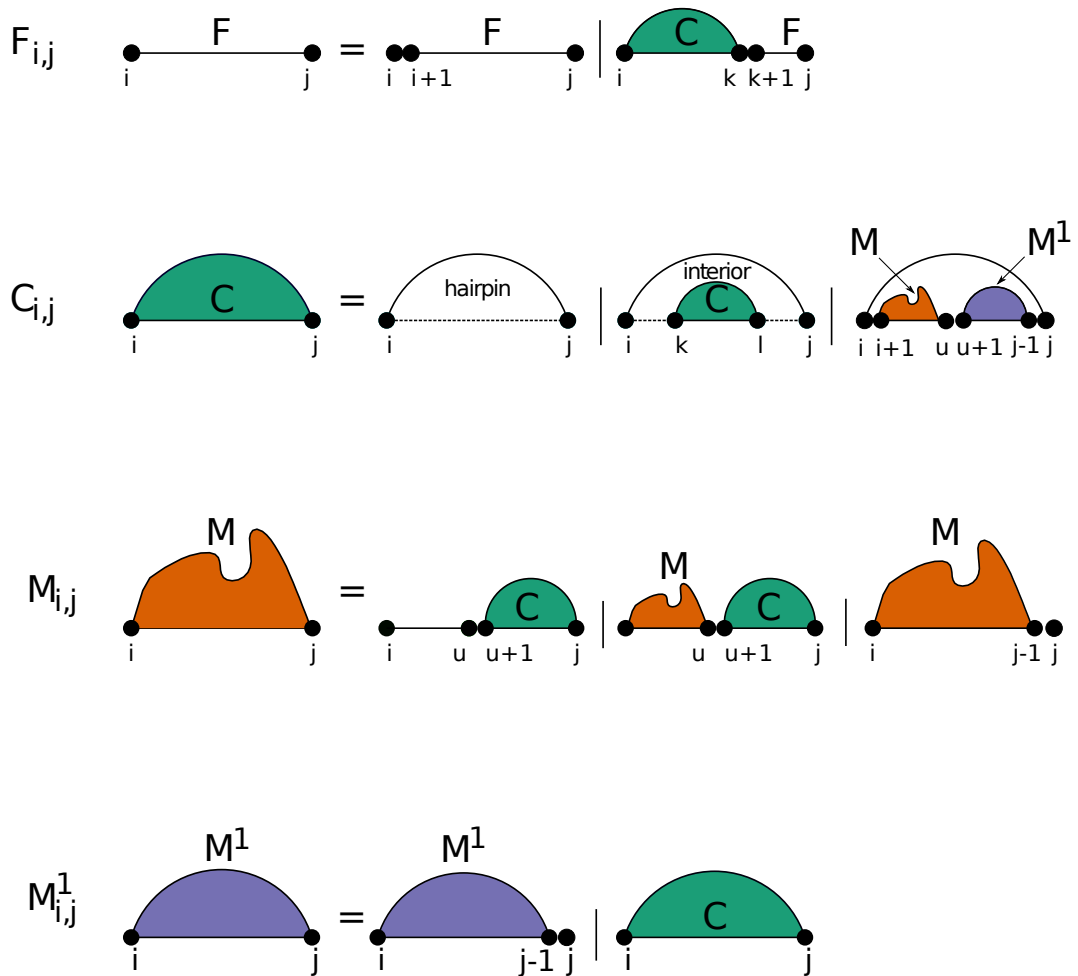


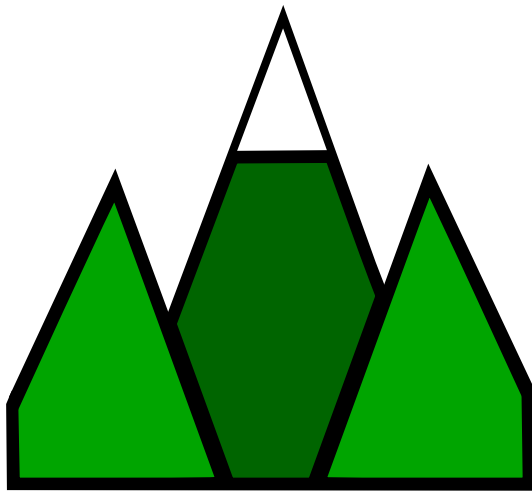
Figure 2.14: Graphical representation of the structural decomposition by the Zuker algorithm. Base pairs are shown as arcs, dotted lines represent unpaired subregions. A not fully calculated multi-loop region is visualized as mountain ridge.





## Chapter 3

### Sierra Platinum



## 3.1 Introduction

### 3.1.1 Goal

The goal of Sierra Platinum is to provide a mathematically sound method for combining the results of replicated experiments. Each replicated experiment consists of the experiment itself and the associated background, the experiment is compared to. Moreover, quality measurements and their respective visualizations are provided to make informed decisions about the quality of the replicates and to select those replicates that should be used and those that should be down-weighted or removed.

Sierra Platinum was developed with an equal contribution by Lydia Müller, as follows. Lydia Müller contributed mainly to the methods of Sierra Platinum, the creation of the benchmark data set, and the website of the service implementation. The visualizations, optimizations, and the implementation of Sierra Platinum as a service within a docker container were mainly contributed by the author of this thesis. The evaluation and the results were mainly produced within the thesis.

### 3.1.2 Overview of the Multi-Replicate Peak-Calling Process

The multiple-replicate peak-calling process of Sierra Platinum is depicted in Figure 3.1. The input data (circles on the left of the figure) consists of all tags of an experiment (black circles) and its associated background (white circles) which together form the replicate.

While computing the peaks, several quality measures for them are computed, too. These quality measures and those intermediate results from the method that allow assessing the quality of the input data are visualized. These visualizations support assessing the quality of the input data and making informed decisions. Moreover, this information can be used to remove or down-weight replicates (experiments) that are qualitatively weak and to rate the final result. All points, where visualizations are provided and interaction for changing parameters of the method is possible are marked with a magnifying glass in Figure 3.1.

In the following, the ‘method’ paragraphs describe the computations needed for establishing the peaks, the ‘quality measure’ paragraphs describe the computations that are used for quality assessment only, and the ‘visualization’ paragraphs describe the visualizations that support the quality assessment.

The steps “Window Construction”–“Single p-Value” are computed only once

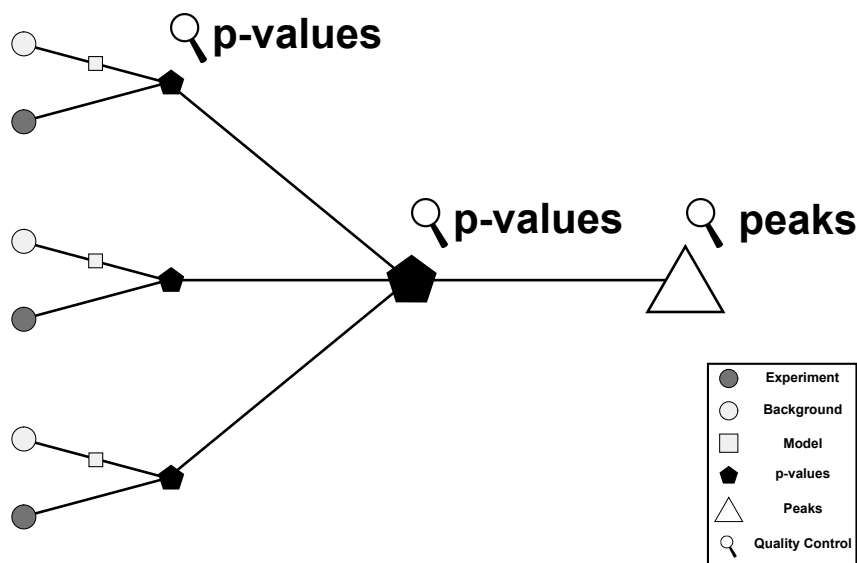


Figure 3.1: Overview of the multiple-replicate peak-calling process. Phase I: Windows are constructed and single replicate  $p$ -values for each window are computed (pentagons) Phase II: From the single  $p$ -values, combined  $p$ -values are computed by combining windows using the inverse normal method (large pentagon). Phase III: Suitable narrow and broad peaks (white triangle) are computed based on the windows' combined  $p$ -values. The magnifying glass symbolizes all points, where a visualization-based quality control is included in the peak-calling process.

to establish single replicate  $p$ -values. The steps “p2q Transformation” and “Combined p-Values” depend on the  $p$  to  $q$  value conversion method and are computed first with the default method. Moreover, the step “Combined p-Values” is first performed with all replicates included and equally weighted. After examining the results, the weights of the replicates can be changed or replicates can be excluded (Section “Filtering and Weighting”). Additionally, the method for converting  $p$  to  $q$  values can be changed. Then, the steps “p2q Transformation”—“Computing Peak Quality” can be performed again with the new configuration of replicates, weights, and  $p$  to  $q$  conversion method. This process of changing the configuration and recomputing the combined peaks can be repeated until a convincing configuration was found—i.e., a configuration with optimal quality. Table 3.1 gives an overview of the steps and if they contribute to the method, compute quality measures, or provide visualizations.

Table 3.1: The steps performed by Sierra Platinum: section and name of the step, and whether it is part of the method, a quality measure, or a visualization, respectively.

	Section	Method	Quality Measure	Visualization
Phase I				
3.3.1	Window Construction	✓		
3.3.2	Window Joining	✓		
3.3.3	Read Quality		✓	✓
3.3.4	Poisson Distribution	✓	✓	✓
3.3.5	Tag Count Frequencies	✓	✓	✓
3.3.6	Scaling	✓		
3.3.7	Normalized Poisson Distribution	✓	(✓)	✓
3.3.8	Neighborhoods	✓		
3.3.9	Single p-Value	✓	✓	✓
3.3.10	p2q Transformation	✓		
3.3.11	Significant Windows		✓	✓
Phase II				
3.3.12	Pearson's Correlation between Replicates		✓	✓
3.3.13	Combined p-Values	✓		✓
3.3.14	Filtering and Weighting			✓
Phase III				
3.3.15	Agreement between the Multi-Replicate Result and the Single Replicate Results		✓	✓
3.3.16	Computing Peaks	✓		
3.3.17	Computing Peak Quality		✓	✓
3.3.18	Exporting Peaks	✓		

Sierra Platinum combines two established methods:

1. The approach of Zhang et al. [90] (implemented in MACS) was adopted for splitting the genome into windows, for calculating the  $p$ -values for each window replicate, and for generating narrow and broad peaks.
2. The inverse normal method as presented by Hedges and Olkin [42] and as used by Wright et al. [87] is adapted to combine the  $p$ -values of the different replicates for each window into one  $p$ -value for each window.

**Phase I** The steps of this phase are computed for each data set—experiment and background of each replicate—or each replicate separately. However, two quality measures compare the windows per replicate to the mean over all replicates for the same window.

First, the windows are constructed (Section 3.3.1). Each window has a start position and a size. All tags overlapping this window are counted. Empty windows are discarded. In principle, this leads to a list of windows. However, to reduce the computation time, the first step is done in parallel on chunks of a certain size. This necessitates a complex data structure, which in turn needs to be transformed into a list (Section 3.3.2).

The next step is not needed for performing the peak-calling itself. Computing the mapped read quality serves as a quality assessment of experiment and background of a replicate (Section 3.3.3). It allows to decide, whether or not to use this replicate for the computation of the combined peaks.

Afterwards, the Poisson distribution of the tag counts of the windows is computed (Section 3.3.4). Similar to Zhang et al. [90], this serves as a model for computing the single replicate  $p$ -values (Section 3.3.9). As a quality estimate, the real tag count frequencies are computed and compared to the theoretical Poisson distribution (Section 3.3.5). Moreover, the results is also used later for computing the weights (Section 3.3.14). As in general the amount of used and mapped material differs between experiment and background, the experiments are scaled (Section 3.3.6). This allows for a better comparison between experiment and background. Based on the scaled experiments, the normalized Poisson distributions are computed (Section 3.3.7).

Next, the 1k, 5k, and 10k neighborhood of each window is determined (Section 3.3.8). Now, single replicate  $p$ -values are calculated for each window and each replicate based on the global  $\lambda$  of the normalized Poisson distribution and the  $\lambda$  values computed for each neighborhood (Section 3.3.9). These

$p$ -values determine the peaks of the replicates. To reduce the effect from the correlation between the  $p$ -values computed, they are transformed into so-called  $q$ -values (Section 3.3.10). As the  $q$ -values are (un-)corrected  $p$ -values, they are called  $p$ -values in the subsequent sections. As additional quality measurements, the amount of significant windows (Section 3.3.11) and the  $p$ -value distribution (Section 3.3.13) are determined.

**Phase II** During this phase, the information computed for the replicates is combined. First, the correlation between the replicates is determined (Section 3.3.12). This information is used for adapting the configuration of the replicates performed next. On the one hand, the replicates need to be correlated to compute justified, combined peaks. On the other hand, correlation is problematic for applying the combination method proposed. Therefore, the configuration has to be corrected for the correlation found.

The replicates are filtered and weighted based on their quality assessment (Section 3.3.14). To compute the combined  $p$ -values for each window, the inverse normal method is applied (Section 3.3.13). During this step, replicates, which are filtered out, are discarded, and the correlation coefficients and weights established previously are applied. Furthermore, the combined  $p$ -values are used for computing additional quality information as well as narrow and broad peaks. The combined  $p$ -values are again correlated and therefore converted into  $q$ -values. As the  $q$ -values are (un-)corrected  $p$ -values, they are called  $p$ -values in the subsequent sections.

**Phase III** During this phase, the agreement between each single replicate and the final combined results is computed and visualized (Section 3.3.15). This quality measure allows to assess the influence of each replicate on the final combined result. Finally, the narrow peaks (Section 3.3.16) and the broad peaks (Section 3.3.16) of the final combined results are determined together with their quality (Section 3.3.17). Finally, the computed peaks can be exported for further analysis (Section 3.3.18).

## 3.2 Related Work

Several peak-callers are available for single experiments. The largest difference between the different methods is the statistical framework used to model the background. Peaks are annotated at positions where the observed number of reads is significantly higher than the one expected by chance given the background model. Koohy et al. [49] and Wilbanks et al. [86] give a good overview and comparison of state-of-the-art peak-callers used in several published studies.

Peak-calling for replicated ChIP-seq experiments, however, is not well supported. Only two approaches exist that use a single experiment peak-caller and either combine the replicates before peak-calling or combine the peaks obtained from the single experiments (see Figure 3.2: MACS-CR and MACS-SA, respectively). Combining the replicates before peak-calling requires equal library sizes (total number of mapped reads) across all replicates or down-sampling to a common library size. For example, the NIH Roadmap Epigenomics Project uses the down-sampling approach to avoid artificial differences in the signal strength with uniform depth of at most 30 million reads before merging the replicates [68]. Similarly, MACS scales the two libraries which are compared to the same amount of reads to make experiment and background library comparable [90]. Down-sampling, however, may lead to an overestimation of the noise level. Moreover, it is not possible to incorporate weights for the replicates based on their quality or to backtrack the source of a specific signal to the supporting replicates.

Combining the peaks of the single experiments includes all peaks in any of the replicates. Thus, replicates with poor peak-call quality can have a large effect on the final result. Furthermore, very long peaks can occur in the final peak set by merging neighboring narrow peaks from different replicates. While none of the replicates predicts those broad peaks, the final result contains them.

On the other hand, PePr [89] explicitly supports replicates of the same condition during peak-calling (see Figure 3.2: PePr). It uses a binomial model that expects the same dispersion for experiment and background. However, for, e.g., experiments performed at different sequencing centers, it is unknown if this condition holds and consequently it is not guaranteed that the results obtained using this model are reliable. Also, PePr down-samples all libraries to the same size and thus might overestimate the noise level.

Another peak-caller for multiple-replicate is BinQuasi [37]. Its model is based

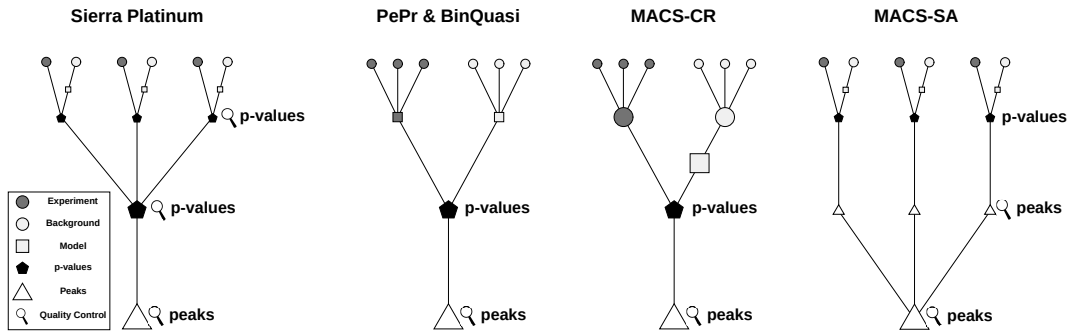


Figure 3.2: Overview of the multiple-replicate peak-calling process showing the basic steps of multiple-replicate peak-calling for Sierra Platinum, PePr, BinQuasi, MACS-CR (combine replicates approach using MACS as single-experiment peak-caller), and MACS-SA (combine peaks approach using MACS as single-experiment peak-caller). The MACS approaches and Sierra Platinum extract the parameters of the underlying model (squares) from the background data (white circles) and use the model and the experimental data to calculate  $p$ -values (pentagons) indicating how significantly enriched the experiment (black circles) is. PePr and BinQuasi generate also a model for the experiment and use both models to calculate  $p$ -values. Based on the  $p$ -values, peaks (triangles) are calculated. Quality control (magnifier) is provided usually alongside with the peaks. Only Sierra Platinum allows to examine the quality during the peak-calling process, while all other methods only allow to examine the quality of the peaks obtained.

on a negative binominal distribution and uses a one side quasi likelihood ratio test to detect peaks. Due to the model, BinQuasi has the same properties as PePr. The benchmark in Section 3.5 shows that BinQuasi reacts even more sensitively to noisy data and in some cases does not detect any peaks at all.

Besides the so far mentioned peak-callers, there are several peak-callers for differential peak-calling, i.e., finding peaks which occur in only one of two groups of samples. The underlying statistical model assumes that there are basically three types of peaks: peaks occurring in both groups of samples and peaks occurring in only one of the two groups. In particular, none of the groups is treated as background for which no peak should be found. Similar to PePr, those peak-callers apply methods from differential gene expression which fit two negative binomial distributions to the two groups for each locus and compare the data based on these distributions. For example, csaw [56] uses the edgeR package [69] to find differential peaks, while diffBind [70] uses peaks predicted on each sample and compares the peaks based on read counts within the peaks also using the edgeR package [69]. Sierra Platinum does not aim to find differential peaks but peaks with respect to a background measurements



which is a very different task from a statistical point of view [56]. Therefore, these peak-callers are not further evaluated in detail.

The currently available approaches for replicate peak-calling are neither designed to assess the replicates' quality nor to handle replicates of different quality. Moreover, the replicates' quality can not be incorporated during peak-calling. In the case of combining the peak-calls of the single experiments, it is in principle possible to introduce weights to account for different qualities of the replicates. However, doing this in a statistically sound way is hardly possible since non-significant positions are not provided by the peak-caller.

## 3.3 Methods

### 3.3.1 Window Construction

#### Method

The whole genome is split into overlapping windows of size  $w$  with offset  $o$ . According to the evaluations (Chapter 3.5), the window size should be the fragment size used in the experiment, while the window offset should be a quarter of the window size. Most frequently, ChIP-Seq data of histone modifications is fragmented with an average fragment size of 200nt. Therefore, the Sierra Platinum defaults are a window size of  $w = 200nt$  and a window offset of  $o = 50nt$ . However, these parameters are accessible through the graphical user interface (GUI, Section 3.4.2) and thus can be changed according to the data used.

Each window is compared to the tags obtained from the experiment and the background, respectively. The number of tags overlapping each window is stored separately for experiment and background for each replicate. Sierra Platinum assumes that the data is stored in files that are in bam format and that artifacts (PCR duplicates) were already removed using for example SAMtools [53] or Picard tools [3]. To access the data, the HTSJDK library [2] is used. Counts for experiment and background are calculated during the construction phase of the windows. Windows, which do not overlap with any tag in any data set, are removed since this is likely to be an artifact of either too low sequencing depth, unknown genome sequence, or highly repetitive sequences to which no tags were mapped (depending on the mapper).

#### Optimization

Constructing the windows is the first step of Sierra Platinum. Windows having a length of  $l$  nt are searched for. The offset between two subsequent windows is  $o$  nt. A window is constructed whenever the  $l$  nt under consideration overlap with tags from at least one experiment or background of one replicate. The number of overlaps is counted and stored with each window.

The general procedure for constructing windows is outlined in Algorithm 1.

---

**Algorithm 1** Construct Windows

---

- 1: Determine windows overlapping at least one tag in one data set
  - 2: Merge these windows (some strategies, see below)
  - 3: Flattening: determine final, ordered window list
- 

To accelerate this step, different parallelization strategies were developed and tested:

Tag Count Parallel: count tags in parallel

Chromosome Parallel: create one thread per chromosome

Chunk Parallel: count chunks in parallel

One important constraint during optimization was to keep the space consumption of all data structures—whether permanent or temporary—at a minimum. Thus, the computation time was minimized while always considering to keep the memory consumption low.

**Tag Count Parallel** The first idea was, to count tags in parallel. First, all windows are constructed sequentially. Then, for each window, the number of tags overlapping this window is counted for each data set. Parallelization was achieved by assigning a certain number of windows to each available thread. During flattening, only those windows are copied to the final list of windows that contain at least one tag in one data set.

This method is simple, but a lot of windows do not overlap any tags. These windows are empty and need not be generated and kept until flattening in the first place. They need to be removed by the garbage collector which takes additional time. The experiments show that the number of threads used for computation should be maximal.

The method uses more than 9 and a half hours (offset: 100nt, size: 400nt) and more than 37 hours (offset: 50nt, size: 200nt) of wall clock time, respectively (Table 3.2). The additional time used for flattening is  $\approx 4$  seconds (offset: 100nt, size: 400nt) and  $\approx 12$  seconds (offset: 50nt, size 200nt), respectively.

**Chromosome Parallel** The second idea is, to count the tags for each chromosome in parallel. For each chromosome, one thread is established and within this thread Algorithm 2 is performed.

Table 3.2: Constructing all windows for 6 replicates: time for different strategies

Strategy	100nt, 400nt (hh:mm:ss)	50nt, 200nt (hh:mm:ss)
Tag Count Parallel	09:38:22	37:02:14
Chromosome Parallel	15:13:13	29:56:02
Chunk – Window – Dataset	08:53:58	18:48:29
Chunk – Dataset – Window	09:28:09	19:49:34
Dataset – Chunk – Window	10:12:28	out-of-memory
Chunk Parallel Coherent	00:09:32	00:15:32

**Algorithm 2** Chromosome Parallel

---

```

for each chromosome (parallel) do
  for each window do
    for all data sets do
      count tags overlapping this window
    end for
    if at least one overlapping tag is found then
      add window to window list
    end if
  end for
end for

```

---

---

**Algorithm 3** Chunk – Window – Dataset

---

```
for for all chunks (parallel) do
  for for all windows of this chunk do
    for for all data sets do
      count tags overlapping this window
    end for
  end for
end for
```

---

As the chromosomes have very different lengths, this leads to an unbalanced use of threads. It can be observed that the number of threads used reduces over time until at the end only one thread is busy. This is reflected by the time used for computing the windows.

Overall, this method needs more than 15 hours of wall clock time; 5 and a half hours more than ‘Tag Count Parallel’ for a window offset of 100nt and a window size of 400nt (Table 3.2). However, for a window offset of 50nt and a window size of 400nt, it needs 7 hours less than ‘Tag Count Parallel’:  $\approx 30$  hours (Table 3.2). The additional time used for flattening is  $\approx 14$  seconds (offset: 100nt, size: 400nt) and  $\approx 41$  seconds (offset: 50nt, size: 200nt), respectively.

**Chunk Parallel** To achieve a more equalized distribution of the workload for each thread, while adding only those windows that contribute to the final result, chunks of equal size are created. Each chunk contains  $c$  windows, where  $c$  is a predefined chunk size. Each chunk is assigned to a thread such that chunks are computed in parallel. For each window, the tag count for each data set is computed. Thus, three loops are used for the computation: one for the chunks, one for the windows of the chunks, and one for the data sets. Rearranging these loops yields the three variations of this strategy described by Algorithms 3–5.

After computing all windows, the final, ordered window list is determined. This step is necessary, because nested data structures are used by the individual variations of the strategy.

**Chunk – Window – Dataset** In this variation (Algorithm 3), all chunks are handled in parallel, the windows of each chunk are constructed sequentially, and the overlaps of each window with tags from each data set are computed sequentially. Thus, it is always clear, if a window can be added to the list, or not.

**Algorithm 4** Chunk – Dataset – Window

---

```

for for all chunks (parallel) do
  for for all data sets do
    for for all windows of this chunk do
      count tags overlapping this window
    end for
  end for
end for

```

---

For this strategy, the following data structure is used. Each window is added to its chunk sequentially. All chunks of a chromosome are stored in a list, according to their number. This yields a list of chunks, with each chunk containing a list of windows. Finally, a map from each chromosome to its ordered list of chunks is used. This nested structure is flattened to obtain the final ordered window list.

For a window offset of 100nt and a window size of 400nt, the time needed by this approach ( $\approx 9$  hours, Table 3.2) is half an hour less than for ‘Tag Count Parallel’, while the order of magnitude is the same. However, for a window offset of 50 nt and a window size of 200nt, it is considerable faster ( $\approx 19$  hours, Table 3.2) than both ‘Tag Count Parallel’ and ‘Chromosome Parallel’. The time used for flattening is  $\approx 20$  seconds (offset: 100nt, size: 400nt) and  $\approx 53$  seconds (offset: 50nt, size: 200nt), respectively.

**Chunk – Dataset – Window** In this variation (Algorithm 4), all chunks are again handled in parallel. However, only the tags for the current data set are counted for each window.

Therefore, the data structure was changed. The chromosome is mapped to a map from window start position to window. Thereby, windows having tag counts for previously considered data sets are retrieved from the window map and the new tag counts for the current data set are added.

The run-time needed is similar to ‘Chunk – Window – Dataset’ (Table 3.2). Flattening takes  $\approx 20$  seconds (offset: 100nt, size: 400nt) and  $\approx 2 : 10$  minutes (offset: 50nt, size 200nt), respectively.

**Dataset – Chunk – Window** The data structure used for this variation (Algorithm 5), is the same as for the previous one. For a window offset of 100nt and a window size of 400nt, the computing times were similar to the previous two strategies (Table 3.2). The time used is 45 minutes longer while

---

**Algorithm 5** Dataset – Chunk – Window

---

```
for for all data sets do
  for for all chunks (parallel) do
    for for all windows of this chunk do
      count tags overlapping this window
    end for
  end for
end for
```

---

the time needed for flattening stayed the same. However, 26GB of memory are not sufficient for computing the windows using a window offset of 50nt and a window size of 200nt.

**Summary** All three strategies handling chunks in parallel described before produce runtimes in the same order of magnitude. The overall load is the same. It can be observed that the load is *not* IO bound as all threads use the complete available computing time and use a maximum of available threads. Moreover, a large number of disc locks can be observed. However, the strategy Dataset – Chunk – Window needs more memory and could not be used to compute the example with a window offset of 50nt and a window size of 200nt. Overall, the strategy Chunk – Window – Dataset is fastest.

**Chunk Parallel Coherent** During the development, it was found that IO is not yet the limit, an additional idea was exploited. The time consumption is large, as long as the tags overlapping individual windows are fetched from the file. As known from other domains like computer graphics, coherence is very important to reduce the amount of work. Scan line algorithms and raytracing are examples of using coherence to improve performance. At the same time, it is known from database technology that fetching a set of items one by one (several SQL statements) performs worse than fetching the set of items in one step (one SQL statement) due to the additional cost for handling an SQL statement that is independent of the number of items fetched.

Therefore, instead of fetching the tags for each window, the tags for the complete chunk are fetched, and then the tags are counted for each window belonging to this chunk. This strategy improved performance, i.e., reduced computation time, by an order of magnitude from more than 9 hours to less than 10 minutes (offset: 100nt, size: 400nt) and from more than 19 hours to less than 16 minutes (offset: 50nt, size: 200nt), respectively (Table 3.2).

Table 3.3: The time needed for constructing all windows for 6 replicates using the strategy Chunk Parallel Coherent with different thread pool sizes

Number of Threads	100nt, 400nt (mm:ss)	50nt, 200nt (mm:ss)
1	17:48	22:53
2	14:00	14:47
4	11:31	12:10
<b>6</b>	<b>08:53</b>	<b>10:45</b>
8	10:23	13:05
16	10:10	14:03
32	11:06	15:07

Further, the load shows that now IO is the limiting factor and no longer CPU power.

As all tags are already processed, this allows to compute the mapped read quality (Section 3.3.3) and the tag count of the data set (Section 3.3.5) at the same time as the windows instead of in separate steps. As the fetching of data is comparatively slow—in fact it is the bottleneck of the computation—this additionally reduces the time needed for these steps by a factor of three.

Testing sequential versus parallel execution showed that a certain number of threads is beneficial for reducing the computation time (see Table 3.3). However, after saturation, adding threads will decrease performance again. For the example and the computational environment used, an optimal number of 6 threads was determined. Please notice, that the optimal number of threads depends on the system architecture and might vary.

### 3.3.2 Window Joining

#### Method

Due to parallelization, windows are kept in a hierarchical data structure during window construction (Section 3.3.1). However, for the subsequent calculations, a linear data structure is more suitable. Therefore, the hierarchical data structure is flattened into a linear list of windows preserving order by genomic start site of the windows within the chromosomes.



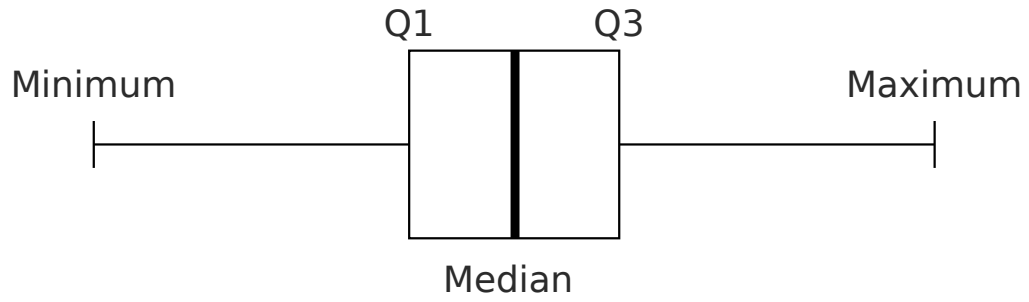


Figure 3.3: Example of a boxplot with lower and upper whisker representing the minimum and the maximum in the data, respectively.

### Optimization

This part is described together with the window construction in Section 3.3.1.

### 3.3.3 Read Quality

#### Quality Measures

Sierra Platinum provides the quality distribution of all mapped tags passed to it as a quality control for the user. Therefore, Sierra Platinum retrieves the quality as Phred Score for each base of each tag in the provided bam file. The Phred score ranges between 0 and 40 and is calculated as

$$Phred = -10 \cdot \log_{10} \left( \frac{p}{1-p} \right) \in [0; 40] \quad (3.1)$$

where  $p$  is the probability that the base call is incorrect. However, using the HTSJDK library [2], Sierra Platinum can obtain the base-wise Phred score directly from the data.

For each data set, Sierra Platinum calculates the median, the lower and the upper quartiles, as well as the minimum and the maximum value of the Phred score distribution.

#### Visualization

The statistics computed are displayed in a boxplot. Boxplots show the median as a line in a box between the lower and upper quartile. This box is extended by the so-called whiskers, whose two ends represent the lowest and the highest value in the data set (Figure 3.3), respectively.

The range of Phred-scores is divided into three categories following the approach of *FastQC* [18]. The colors and category ranges for the quality

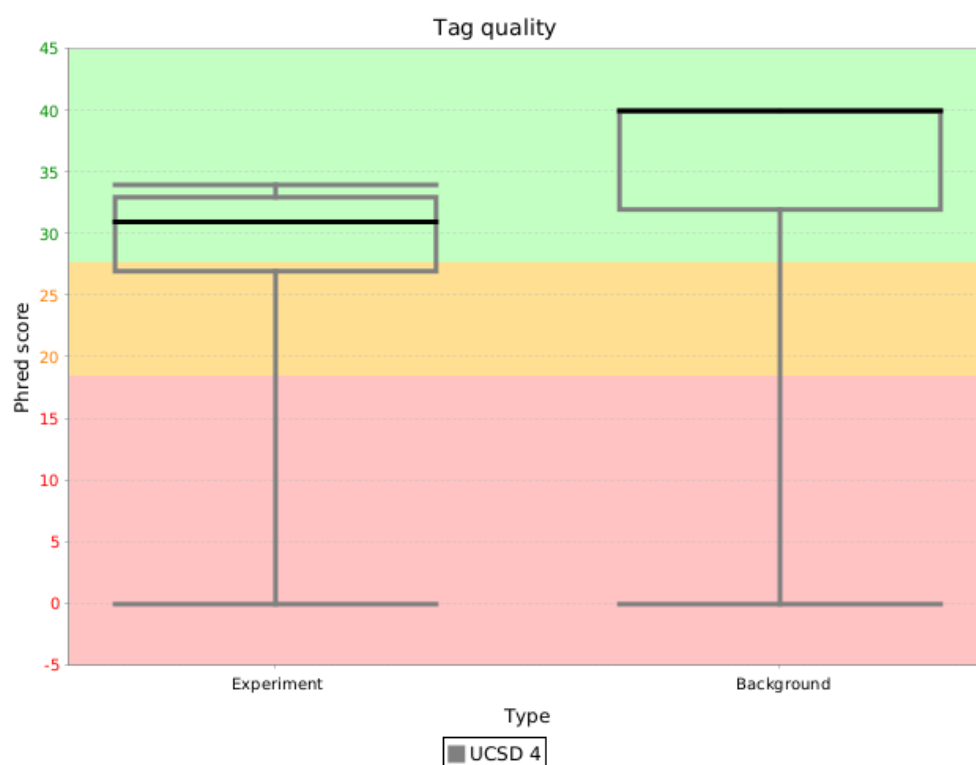


Figure 3.4: Example boxplot figure created by Sierra Platinum with the arranged background.

scores used there are also used here. For each replicate, the boxplots for both experiment and background are shown in the same figure for easing their comparison (Figure 3.4).

Ideally, the boxplots for experiment and background are similar, i.e., the read quality was very similar, and most of the bases have a good quality. For real data, this might not be the case. If the read quality distributions are very different between experiment and background of the same replicate, one may want to exclude or to down-weight the replicate. In particular, a bad background quality and a good experiment quality can lead to a high rate of false positive peaks. The other way around, a bad experiment quality and a good background quality might be acceptable but comes with the side effect, that it is likely that many peaks are missed. A result of reads with bad quality can be miss-mapped reads and thus, decreased reliability of the peak-calls based on this data.

## Optimization

To obtain the mapped read quality, Sierra Platinum calculates for each data set the median, the lower and the upper quartiles, and the minimum and the maximum value of the Phred score distribution.

This step can be joined with the window construction step, if strategy 'Chunk Parallel Coherent' is used (Section 3.3.1). In this case, it will not use additional time.

Otherwise, it takes  $\approx 11$  minutes (offset: 100nt, size: 400nt) and  $\approx 13$  minutes (offset: 50nt, size: 200nt), respectively. Parallelization using threads does not decrease the time needed, as this step is essentially IO limited.

### 3.3.4 Poisson Distribution

#### Method

Following MACS [90], Sierra Platinum models the tag distribution of the background with a Poisson distribution and uses this as the noise model for the experiment. For each window, it is then decided whether the observed tag counts are significant according to the noise model. This will be discussed in detail in Section 3.3.9.

A Poisson distribution is defined by one parameter  $\lambda$  which describes both the mean and the variance of the distribution. Thus, the noise model is generated from the background, by simply calculating the mean of the number of tags in each window for experiment and background of each replicate.

#### Quality Measures

Even though Sierra Platinum only needs the  $\lambda$  based on the background, Sierra Platinum also calculates the  $\lambda$  of the experiment. The latter serves as quality control for the user. Very different means between experiment and background indicate very different library sizes and large differences between experiment and background measurement. It is expected that they do not fit perfectly and we account for this fact by scaling the experiment (see Section 3.3.6). However, scaling may lead to over-estimation of the noise level. Thus, the Poisson distribution of the raw counts (namely the non-scaled counts) of the experiment indicates whether the noise level might be over-estimated.

#### Visualization

To allow the user to observe such issues, we show the Poisson distribution of the raw counts for both data sets, experiment and background, of each replicate as two curves in a line chart (Figure 3.5a). The horizontal axis shows the number of occurrences  $k$  of tags in a window and the vertical axis is the probability for

each number ablated. Since the number of occurrences can only be integers, the lines between the  $k$ -values are only guidelines for a better perception of the distribution. The red line represents the estimated Poisson distribution for the experiment data and the blue line the estimated Poisson distribution for the background data. It is possible to zoom the line chart to handle also bigger  $\lambda$  values, since their peaks can be far apart from each other.

### Optimization

Computing the distribution of the data and its noise model is already fast— $\leq 15$  seconds (offset: 100nt, size: 400nt),  $\leq 45$  seconds (offset: 50nt, 200nt)—and therefore was not optimized.

## 3.3.5 Tag Count Frequencies

### Method and Quality Measures

Even though in theory the tags should be Poisson distributed, real data usually does not perfectly fit the theoretical distribution. A small deviation from the theoretical model is acceptable. However, a high deviation or a completely different distribution would mean that the model estimated from the background data is not a good noise model and therefore would result in uninterpretable  $p$ -values.

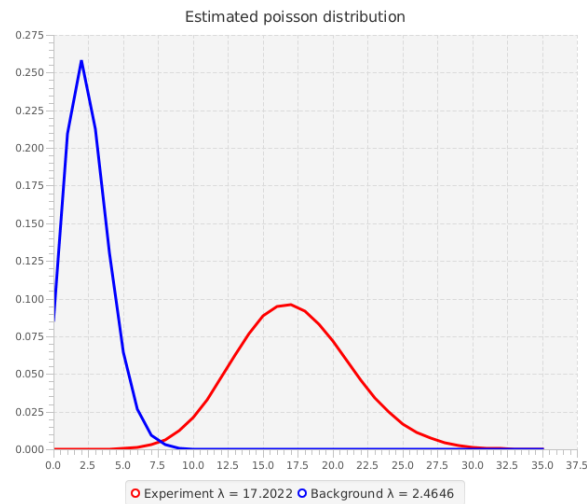
As a quality estimate, Sierra Platinum computes the tag count distribution of the real data, i.e., it calculates the relative frequency for each observed tag count. This distribution corresponds to the theoretically estimated Poisson distribution. As a measure of fitness, Sierra Platinum calculates the least squares difference between the theoretical and the real distribution for both, experiment  $\beta_{exp}$  and background  $\beta_{back}$ .

For each replicate, Sierra Platinum uses the sum of the least square distances of background and experiment to the corresponding theoretical Poisson distributions as combined least square  $\beta$  for the whole replicate, i.e.:

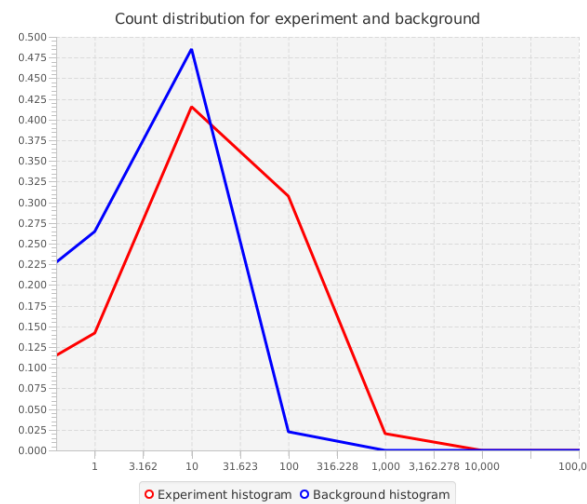
$$\beta = \beta_{exp} + \beta_{back} \quad (3.2)$$

Since the tool is able to weight replicates during peak-calling, Sierra Platinum estimates weights  $\omega$  based on the least square distance  $\beta$  as follows:

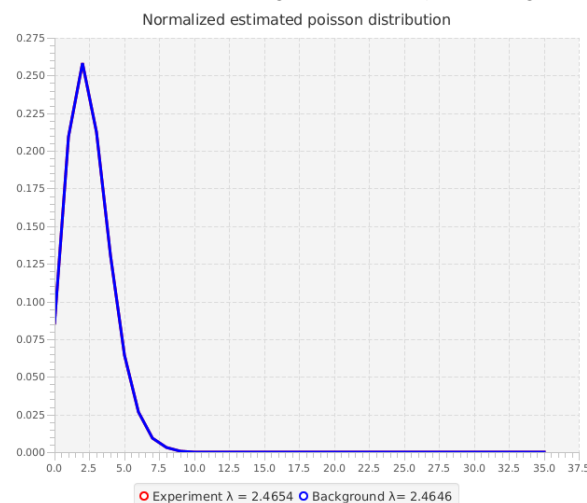
$$\omega = \frac{1}{1 + \beta} \quad (3.3)$$



(a) Example of an estimated Poisson distribution for an experiment and the corresponding background.



(b) Example of the tag count distribution visualization. The data corresponds to the data used in Figure a. The number of tags are displayed using a logarithmic scale.



(c) Example of the normalized estimated Poisson distributions. The normalized data was derived from the data used in Figure a.

Figure 3.5: Distribution of tags

## Visualization

The tag count distribution is visualized for each replicate using line charts allowing to compare between the tag count distribution and the estimated Poisson distribution (Figure 3.5b). Since real data might contain large outliers, it is necessary to display the number of occurrences of tags  $k$  using a logarithmic scale on the horizontal axis. The vertical axis represents the relative frequency  $k$  of tags. As for visualizing the Poisson distribution (Section 3.3.4), the experiment is mapped onto red lines and the background data onto blue lines. As shown in Figure 3.5a and Figure 3.5b, the overlap and differences between the tag count distribution and the estimated Poisson distribution can easily be seen. It is clearly visible that the estimated Poisson distributions fit inherently to the data distributions in this example.

## Optimization

Computing the tag count distribution of the real data, i.e., the relative frequency for each observed tag count is performed in parallel for all replicates by assigning each replicate to a thread. At the same time, the least square distances to the Poisson distribution (for each data set), the bins (for each data set), and the final weights (for each replicate) are computed. Overall, the computation times are  $\approx 50$  seconds (offset: 100nt, size: 400nt) and  $\approx 26$  seconds (offset: 50nt, size: 200nt), respectively.

### 3.3.6 Scaling

#### Method

A good noise model can only be estimated if the used and the mapped material are of a comparable amount. For real data, this is usually not the case. Moreover, real data usually suffers from very different library sizes. Therefore, scaling the libraries to the same size is necessary.

In Sierra Platinum, the experiment is scaled such that the library sizes measured as total number of mapped tags are equal. In more detail, we count the total number of mapped tags in the input data separately for experiment and background as  $t_{exp}$  and  $t_{back}$ , respectively. Then, the scaling factor for the experiment  $sf$  is:

$$sf = \frac{t_{back}}{t_{exp}} \quad (3.4)$$

The raw tag counts  $t_{raw}$  of the experiment are then normalized to fit the background counts. The normalized counts  $t_{norm}$  of the experiment are calculated as:

$$t_{norm} = sf \cdot t_{raw} \quad (3.5)$$

Any further reference to experiment counts will refer to the normalized experiment counts from now on.

### Optimization

In Sierra Platinum, each experiment is scaled such that the library sizes of experiment and background of a replicate measured as total number of mapped tags are equal. Scaling is done for all windows in parallel. Therefore, each window is assigned to a thread and all experiments of all replicates are scaled. This step is very fast taking  $\leq 5$  seconds (offset: 100nt, size: 400nt) and  $\leq 8$  seconds (offset: 50nt, size: 200nt), respectively. Thus, no further optimization is necessary.

### 3.3.7 Normalized Poisson Distribution

#### Methods

The normalization ought scale background and experiment to the same level. As a result also the differences between their respective Poisson distributions should be reduced to a minimum. This can serve as an additional quality check. If the theoretical Poisson distributions estimated from the normalized data are still very different, the normalization was not able to make background and experiment comparable. As a consequence, any  $p$ -values estimated based on this data will be spurious and likely to be wrong.

The theoretical Poisson distributions for the normalized data are estimated in the same way as the theoretical Poisson distributions for the raw counts (Section 3.3.4) by simply exchanging the raw counts by the normalized counts.

#### Visualization

It is possible to monitor the results of the scaling and normalization with a visualization of the normalized estimated Poisson distribution in the GUI of Sierra Platinum. Since the normalized Poisson distributions are created like their raw Poisson distribution counterparts, the same visualization is used.

An example is shown in Figure 3.5c, where the normalized data was derived from the data used in Figure 3.5a. It is clearly visible that the experiment data was normalized onto the background data since the  $\lambda$  of the background did not change. If for some reason the experiment and the background data are too different, the experiment data can not be normalized that accurately and the experiment and the background lines in the chart do not overlap each other anymore.

## Optimization

Computing the distribution of the scaled data and its noise model is already very fast taking  $\leq 15$  seconds (offset: 100nt, size: 400nt) and  $\leq 35$  seconds (offset: 50nt, size: 200nt), respectively, and therefore was not parallelized.

### 3.3.8 Neighborhoods

#### Method

Neighborhoods of sizes 1k, 5k, and 10k are established for each window. These neighborhoods are required during the  $p$ -value calculations for the single replicates (Section 3.3.9) to account for local sequence composition biases.

The neighborhood  $N_s(w_i)$  of the window  $w_i$  of size  $s$  is the set of windows that overlaps with the interval of size  $s$  centered at the mid-point of window  $w_i$ . The neighborhood consists of windows on the same chromosome, only.

Several options for storing and computing neighborhoods were explored. The final solution does not store neighborhoods, but directly computes the corresponding  $\lambda$  that is then directly used for computing the single replicate  $p$ -values (Section 3.3.9). Therefore, each of the neighborhoods is initialized by its range, the window list, and the replicate list. Internally, the index into the window list of the first and of the last neighbor is stored. As the windows are processed sequentially to obtain the corresponding  $\lambda$ -values, the neighborhood indices are updated for each window. At the same time, the tag count of the window's neighborhood is updated. Dividing the tag count of the neighborhood of window  $w_i$  by the number of windows in this neighborhood yields the  $\lambda_s^d(w_i)$  used for the  $p$ -value computation:

$$\lambda_s^d(w_i) = \frac{\sum_{w \in N_s(w_i)} t_w^d}{|N_s(w_i)|} \quad (3.6)$$



where  $t_w^d$  is the tag count of window  $w$  for data set  $d$  (background or experiment of a replicate) and  $|N_s(w_i)|$  is the number of windows of the neighborhood  $N_s(w_i)$  of window  $w_i$ .

## Optimization

The neighborhoods of 1k, 5k, and 10k for each window are required during the  $p$ -value calculations for the single replicates to account for local sequence composition biases (Section 3.3.9).

Several options for storing and computing neighborhoods were explored:

1. Compute and Store
  - (a) Store with window (computed and stored during flattening)
  - (b) Store separately (computed and stored before  $p$ -value computation)
2. Compute and Use during  $p$ -value computation

**Compute and Store with Window** The first solution computes the neighborhood while constructing the window list during the flattening phase. The indices of the first and of the last window of the respective neighborhood are stored with each window, which requires six additional values per window. As flattening is performed sequentially, the neighborhoods are also created sequentially.

**Compute and Store Separately** Storing the neighborhood with each window is not necessary as it is only used for one step (Section 3.3.9). To minimize space consumption, the two steps were separated and the neighborhoods are constructed just before they are used. Thus, the second solution computes all neighborhoods for all windows and stores them in a separate class. The storage of the neighborhoods is released immediately after the single replicate  $p$ -value computation (Section 3.3.9).

This step is time consuming and therefore was parallelized. As neighborhoods of a window belong to the same chromosome as the window itself and coherence should be exploited, a parallelization over chromosomes was chosen.

**Compute and Use During  $p$ -Value Computation** The final solution does not store neighborhoods any more, but directly computes the corresponding  $\lambda$  that is then directly used for computing the single replicate  $p$ -values (Section 3.3.9). Therefore, each of the neighborhoods is initialized by its range, the window list,

Table 3.4: Computing window neighborhoods and  $p$ -values for 6 replicates and 3 different neighborhood sizes: time for different strategies.

Strategy	Neighborhood (mm:ss)	$p$ -value (mm:ss)
Compute and Store with Window	< 00:30	02:57
Compute and Store Separately	00:55	02:57
Compute and Use During $p$ -Value Computation		02:06

and the replicate list. Internally, the indices into the window list of the first and of the last neighbor are stored. As the windows are processed sequentially to obtain the corresponding  $\lambda$ -values, the indices are updated for each window. At the same time, the tag count of the window's neighborhood is updated. The parallelization is again over all chromosomes.

**Summary** The time consumption of the different solutions (including  $p$ -value computation) is given in Table 3.4. It shows that the most space efficient solution - computing and using the neighborhood directly during  $p$ -value computation - is also the fastest solution.

### 3.3.9 Single $p$ -Value

#### Method

After preparing the experiment counts and the noise distribution, Sierra Platinum uses this data to calculate for each window and replicate a  $p$ -value. The  $p$ -value is the probability that one observes an at least as high tag count in random data as observed in the experiment. Hereby, the random data is modeled by the Poisson distribution with mean  $\lambda$ . Thus, the  $p$ -value for observing at least  $c$  tags in the experiment is calculated as the reciprocal of the cumulative Poisson distribution with mean  $\lambda$ .

$$\begin{aligned}
 p &= P(X \geq c, \lambda) \\
 &= 1 - P(X < c, \lambda) \\
 &= 1 - \sum_{i=0}^{c-1} \frac{\lambda^i \cdot e^{-\lambda}}{i!}
 \end{aligned} \tag{3.7}$$

It is known that due to biases in the library preparation and the local sequence composition, local estimates of the mean tag count would serve as a better noise model than the global estimate. Therefore, we use the same approach as MACS [90] and calculate the mean tag counts in the 1k, 5k, and 10k neighborhood of each window resulting in  $\lambda_{1k}$ ,  $\lambda_{5k}$ , and  $\lambda_{10k}$ , respectively.

The final mean of the Poisson distribution of the noise model is defined as:

$$\lambda = \max\{\lambda_{global}, \lambda_{1k}, \lambda_{5k}, \lambda_{10k}\} \quad (3.8)$$

where  $\lambda_{global}$  is the lambda estimated over all windows.

### Quality Measures and Visualization

For each replicate, Sierra Platinum computes the distribution of the  $p$ -values and visualizes this distribution using a bar chart histogram to check the quality of the single replicate peak-calling step (Figure 3.6). The combined  $p$ -values are binned into intervals from 1 to  $10^{-18}$  and visualized as bars. Both x- and y-axis are logarithmically scaled. The numerical method behind the  $p$ -value distribution produces values down to  $10^{-16}$ . Smaller values are assigned to bin  $10^{-18}$ . The gap between  $10^{-16}$  and  $10^{-18}$  is intentional pointing at this fact visually.

### Optimization

The final  $\lambda$  value for each window is computed as the maximum of the global  $\lambda$  value and the  $\lambda$ -values of each neighborhood (Equation 3.8). For each window and for each replicate, a  $p$ -value is computed from the final  $\lambda$  value and the tag counts.

When computing the neighborhoods explicitly for all windows, the single replicate  $p$ -values are computed in parallel over all windows assigning an equal number of windows to each thread:

$$n = \frac{|windows|}{|cores|} + 1 \quad (3.9)$$

except for the last thread, which handles the remaining windows. This heavily reduces the time needed. It scales with the number of threads available. The time used for the given configuration is  $\approx 3$  minutes (Table 3.4).

Using the space efficient version, the single-replicate  $p$ -values for each window are directly computed from the window and its neighborhood in one

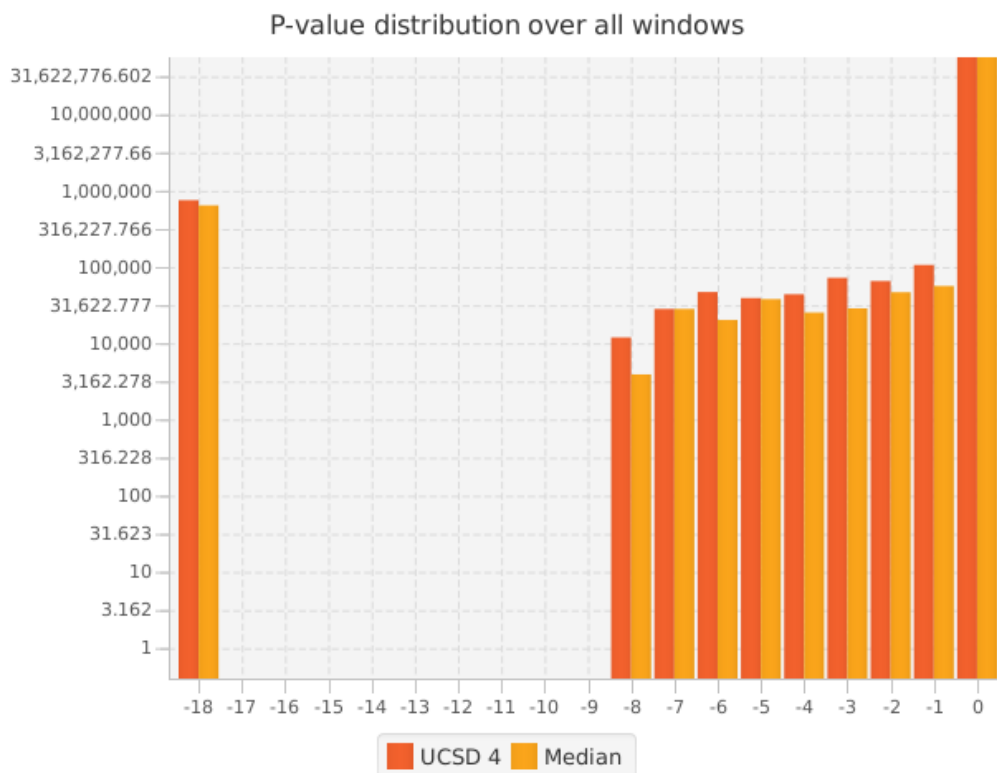


Figure 3.6: Example of the  $p$ -value distribution generated during the single peak-calling step. Red:  $p$ -values of one replicate. Orange: median of  $p$ -values over all replicates. Please note, that the x-axis shows only the exponent to the basis 10 and that both axes are scaled logarithmically.

step. All chromosomes are computed in parallel by assigning each chromosome to a thread that then computes the  $p$ -values (and the neighborhoods) for all windows on this chromosome. The combined approach uses only  $\approx 2$  minutes (offset: 100nt, size: 400nt) (Table 3.4) and  $\approx 10$  minutes (offset: 50nt, size: 200nt), respectively, and the space consumption is minimal.

Computing the distribution of the  $p$ -values uses  $\approx 50$  seconds (offset: 100nt, size: 400nt) and  $\approx 1 : 45$  minutes (offset: 50nt, size: 200nt), respectively. The first part of the computation is parallelized assigning each replicate to a thread, while the second part is sequential.

### 3.3.10 p2q Transformation

#### Method

Within one replicate and for the final significance test, Sierra Platinum repeats the same test for each window to obtain  $p$ -values for each window. However, it is well known that this leads to the so-called multiple testing problem. The more often a test is performed, the higher the chance to obtain a false positive result. With other words, the resulting  $p$ -values from the tests are too low due to multiple testing.

Several methods exist that allow correcting the  $p$ -values thus controlling the false discovery rate. The corrected values are referred to as  $q$ -values. In Sierra Platinum, two different methods are implemented: one proposed by Holm-Bonferroni and one proposed by Storey.

**Holm-Bonferroni** The Holm-Bonferroni correction [45] is a rather conservative method and thus may reduce the number of significant windows dramatically. It assumes that the list of  $p$ -values is sorted ascending, i.e.,  $i < j \rightarrow p_i \leq p_j$ .

Sierra Platinum obtains such a list using a parallelized merge sort. The  $i$ -th  $p$ -value  $p_i$  is corrected to  $q_i$  by

$$q_i = \min(p_i \cdot (N - i), 1) \quad (3.10)$$

where  $N$  is the number of tests performed. In this case, the number of tests is equal to the number of windows since Sierra Platinum performs one test for each window.

**Storey** Storey's  $q$ -values [77] also calculate the correction factor for the  $p$ -values. However, the underlying method is different. It uses the fact that random  $p$ -values are uniformly distributed but significance tests usually skew the distribution towards 0 or 1. A good estimate for the rate of false positives  $\hat{\pi}_0$  is the height of the uniform distribution while the rate of true positives is the height of the  $p$ -value distribution without the uniformly distributed part. Similarly to the Holm-Bonferroni correction, the correction is done stepwise on the sorted list of  $p$ -values. Storey's  $q$ -value calculation uses a bottom-up approach, i.e., starting with the largest  $p$ -value  $p_N$ :

$$q_N = \hat{\pi}_0 \cdot p_N \quad (3.11)$$

The subsequent  $q$ -values are calculated as follows:

$$q_i = \min \left( \frac{N}{i} \cdot \hat{\pi}_0 \cdot p_i, q_{i+1} \right) \quad (3.12)$$

where  $i$  is running from  $N - 1$  to 1.

To obtain the height of the uniform distribution  $\hat{\pi}_0$ , Storey proposes several methods. Sierra Platinum provides two of them: 'Storey Simple' and 'Storey Bootstrap'.

**Storey Simple** The most simple method is to estimate the height  $\hat{\pi}_0$  from a representative  $p$ -value in the  $p$ -value distribution, i.e., 0.5.

**Storey Bootstrap** For the bootstrap method, a cubic spline is fitted to the  $p$ -value distribution and the spline is used to estimate the height of the distribution  $\hat{\pi}_0$ . Sierra Platinum provides the bootstrap approach that makes fitting the cubic spline more robust [77].

## Interaction

The  $p$ -value correction method can be set using a drop-down-box (Figure 3.10, Section 3.3.14). The available  $p$ -value correction methods are 'None', 'Holm-Bonferroni', 'Storey Simple', and 'Storey Bootstrap'. The default is set to 'Holm-Bonferroni' to obtain a conservative correction.

As the  $q$ -values are (un-)corrected  $p$ -values, they are called  $p$ -values in the following sections.

Table 3.5: Time needed for transforming  $p$ - to  $q$ -values using different strategies

Strategy	100nt, 400nt (mm:ss)	50nt, 200nt (mm:ss)
Holm-Bonferroni	01:45	03:35
Storey-Simple	07:09	25:26
Storey-Bootstrap	04:04	03:52

### Optimization

The  $p$ -value correction methods are run in parallel for each replicate. Each method was split into two parts: sorting the  $p$ -values and converting  $p$ - to  $q$ -values. For sorting the  $p$ -values, 32 threads are used, while for converting 6 from the 32 threads available are used, one per replicate. In principle, the Holm-Bonferroni correction could be performed in parallel for each replicate, further speeding up the computation. However, for the Storey methods, the  $q$ -value computation is sequential per replicate and can not be sped up further. The overall time needed by the three different methods implemented are shown in Table 3.5. For a window offset of 100nt and a window size of 400nt, the Holm-Bonferroni correction can be computed relatively fast using 1:45 minutes, both Storey-Simple with 7:09 minutes and Storey-Bootstrap with 4:04 minutes take longer. For a window offset of 50nt and a window size of 200nt, the Holm-Bonferroni correction can be computed relatively fast using 3:46 minutes, Storey-Bootstrap with 3:52 minutes takes approximately the same time, and Storey-Simple takes much longer using 25:26 minutes. However, the Storey-Bootstrap correction is dependent on random numbers and thus does not produce consistent results over different runs.

### 3.3.11 Significant Windows

#### Quality Measures

The last quality measurement that Sierra Platinum provides for each replicate is the distribution of the significant windows. More precisely, for each chromosome  $c$  and each replicate  $i$ , Sierra Platinum counts the number of significant windows  $s_i^c$ :

$$s_i^c = |\{w \in W^c | p_i^w < \hat{p}\}| \quad (3.13)$$

where  $W^c$  is the set of all windows of chromosome  $c$ ,  $p_i^w$  is the  $p$ -value of window  $w$  for replicate  $i$ , and  $\hat{p}$  is the significance cutoff. Additionally, Sierra Platinum calculates the median distribution of the significant windows for each

chromosome. In detail, for each chromosome  $c$ , the algorithm calculates the median significant window  $\bar{s}^c$ :

$$\bar{s}^c = \text{median}\{s_i^c \mid \forall i \in [1, n]\} \quad (3.14)$$

where  $n$  is the number of replicates.

This measurement allows the user to investigate two facts.

**Has the current replicate an odd distribution of the significant windows compared to the median distribution?** If the overall distribution of significant windows is very different from the median distribution, then the peak-calling of this replicate will not overlap strongly with other replicates and may reduce the quality of the combined peak-calling over all replicates. An odd distribution can have several reasons. One possibility is that the conditions for the experiment of one replicate were very different from the conditions of the other replicates, which might have induced changes in the epigenetic state. Furthermore, it might mean that part of the library preparation or sequencing did not work out as they should. Since this approach is designed to do peak-calling for multiple replicates, i.e., peak-calling for the repeated measurement of the same state, one might prefer to exclude such replicates from peak-calling.

**Is there a chromosome with an odd number of significant windows?** If the overall distribution is similar to the median but only a few chromosomes diverge from the median, the replicate is suited for the multiple-replicate peak-calling step. However, one might want to perform peak-calling on this single replicate afterwards to investigate the differences in the peaks between this replicate and the multiple-replicate peak-calling on this chromosome.

## Visualization

The visualization shown in Figure 3.7 provides the necessary information to help the user to deal with the afore-mentioned facts. The amount of significant windows for each chromosome is visualized as a bar chart. The red bars show the number of significant windows for the replicate and can be compared to the orange bars that show the median of significant windows for each chromosome. The user can easily determine if the distribution of significant windows for each replicate is anomalous by comparing it with the median distribution or detect single chromosomes that deviate from the median.



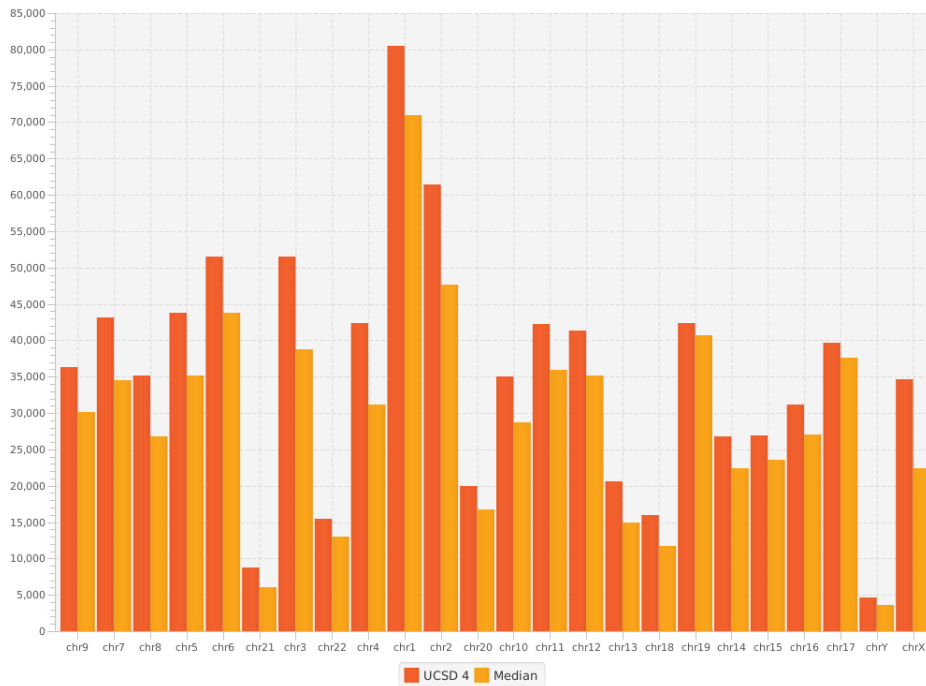


Figure 3.7: Example of a significant window distribution over all chromosomes. Red: significant windows of one replicate. Orange: median of significant windows over all replicates.

## Optimization

Sierra Platinum provides for each replicate the distribution of the significant windows: for each chromosome  $c$  and for each replicate  $i$ , we count the number of significant windows  $s_i^c$ . Additionally, we calculate the median distribution of the significant windows for each chromosome.

This step is already very fast using  $\leq 15$  seconds (offset: 100nt, size: 400nt) and  $\leq 30$  seconds (offset: 50nt, size: 200nt), respectively, and therefore was not parallelized.

### 3.3.12 Pearson's Correlation between Replicates

#### Quality Measures

Sierra Platinum also provides quality measurements between replicates in addition to those within replicates. The Pearson's correlation between the replicates allows to justify whether the replicates seem to agree on the significance of the windows in consensus (positive correlation).

As a logical consequence from the fact that all replicates measured the same modification (or chromatin bound protein) under comparable conditions and in the same cell line, positive correlations between the replicates are expected.

A positive correlation close to 0 may result from differences in the protocol, the conditions, the sequencing, or the mapping method for the tags. The resulting peak-calls may be biased by this fact and likely contain false negatives and false positives.

Negatively correlated replicates have to be treated with caution for two reasons.

1. Negative correlation indicates that windows in one replicate are significant while in the other replicate they are clearly not significant. This indicates that something went wrong in one of the experimental procedures.
2. For the inverse normal method that we use to combine the  $p$ -values of the replicates (see Section 3.3.13), negative correlations are problematic and alter the significance level. Therefore, all replicates used for the multiple-replicate peak-calling have to be positively correlated to each other.

Pearson's correlation assumes that the tested variables are normally distributed. However,  $p$ -values are uniformly distributed and are therefore not suited for correlation estimates. This problem was solved by estimating the correlations from the so-called probits instead of from the  $p$ -values. Probits are obtained by transforming the  $p$ -values with the inverse cumulative standard normal distribution. They are calculated using the inverse normal method, which is also used to combine the  $p$ -values of each window of the replicates into one single  $p$ -value for each window (see Section 3.3.13).

Let  $\tau, \tau'$  be the vectors of all probits for two replicates and let  $n_w$  be the number of windows and thus, also be the length of the vectors  $\tau$  and  $\tau'$ . The mean  $\bar{\tau}$  of  $\tau$  is computed as

$$\bar{\tau} = \frac{1}{n_w} \sum_{i=1}^{n_w} \tau_i \quad (3.15)$$

and the standard deviation  $s_\tau$  of  $\tau$  is computed as

$$s_\tau = \sqrt{\frac{1}{n_w - 1} \sum_{i=1}^{n_w} (\tau_i - \bar{\tau})^2} \quad (3.16)$$

The mean  $\bar{\tau}'$  and the standard deviation  $s_{\tau'}$  of  $\tau'$  are calculated analogously. The Pearson's correlation  $\rho_{pearson}(\tau, \tau')$  is thus

$$\rho_{pearson}(\tau, \tau') = \frac{\left( \sum_{i=1}^{n_w} \tau_i \tau'_i \right) - n_w \bar{\tau} \bar{\tau}'}{(n_w - 1) s_{\tau} s_{\tau'}} \quad (3.17)$$

## Visualization

The correlations between all replicates are visualized in a heatmap (Figure 3.8). Each cell in the heatmap describes the strength of the correlation between the corresponding column and row. Positive correlations are encoded with red and negative correlations with blue, respectively. The strength of the correlation is mapped to the saturation value, using the *HSB* color model.

$$color_{i,j} = \begin{cases} \text{HSB}(0, c_{i,j}, 1.0), & \text{if } c_{i,j} \geq 0 \\ \text{HSB}(240, (-c_{i,j}), 1.0), & \text{if } c_{i,j} < 0 \end{cases} \quad (3.18)$$

In the GUI of Sierra Platinum, it is possible to see the strength of the correlation for each cell by a tool tip. With this visualization it is easy to see how the replicates are correlated to each other and the user can recognize replicates that are problematic for the *p*-value combination step (see also Sections 3.3.14 and 3.4.5 describing how to enable/disable replicates and how to set weights).

## Optimization

To compute the Pearson's correlation between the replicates, the mean and the standard deviation of the probits over all windows for each replicate are computed, as well as the correlation between the replicates themselves. Three possible strategies were evaluated:

1. Sequential computation: one thread is used for all computations
2. Parallel: compute sequentially each of the following steps in parallel
  - (a) Computing the mean values of the replicates: one thread per replicate
  - (b) Computing the standard deviations of the replicates: one thread per replicate
  - (c) Computing the Pearson correlation of each pair of replicates. This results in a correlation matrix. The main diagonal of this correlation matrix is always one and not computed. Further, upper and lower

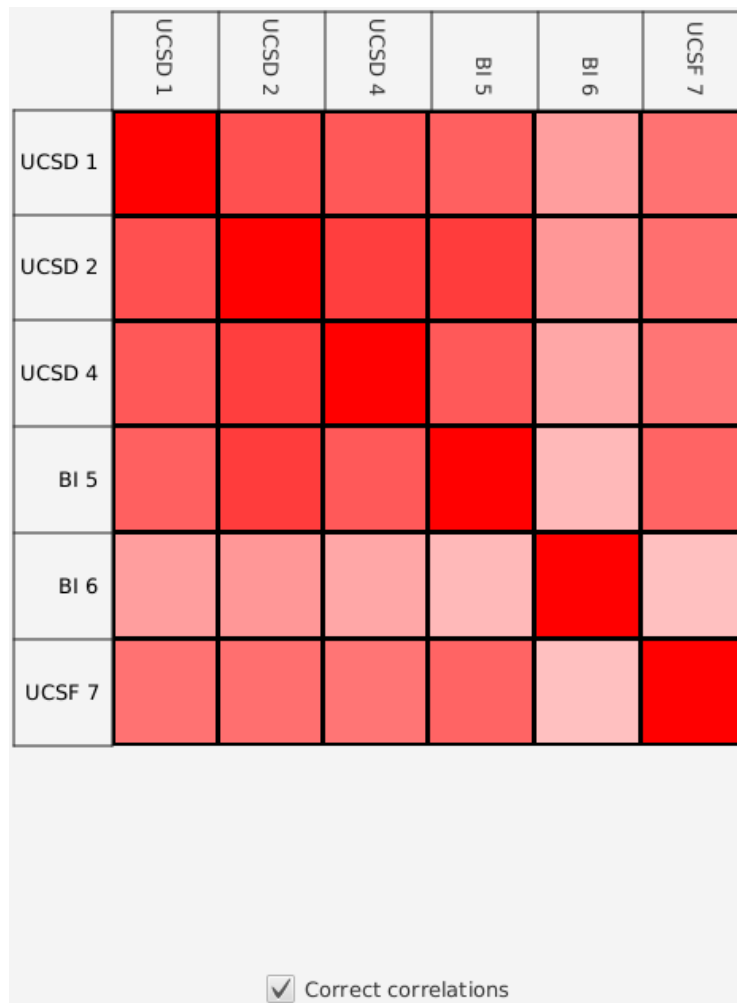


Figure 3.8: Top: heatmap of the Pearson's correlations between the replicates. Bottom: checkbox for enabling or disabling the correlation correction while computing the combined  $p$ -value. If the checkbox is checked,  $\hat{p}^*$  is computed as described in Section 3.3.13, otherwise  $\hat{p}^*$  is set to 0.

Table 3.6: Computing Pearson's correlation for 6 replicates: time for different strategies.

Strategy	100nt, 400nt (mm:ss)	50nt, 200nt (mm:ss)
Sequential	3:38	8:00
Parallel row	0:53	1:50
Parallel all	0:22	0:45

triangle of the correlation matrix are symmetric. Therefore, only the upper triangle is computed.

- i. Row: compute one row after each other, one thread per column
- ii. All: use one thread per correlation computed

The results for computing the correlation between six replicates are shown in Table 3.6. Computing all correlations in parallel is fastest followed by computing each row in parallel. Computing all steps sequentially is very slow, as expected.

### 3.3.13 Combined p-Values

#### Method

While MACS would generate a peak list using the calculated  $p$ -value at this point, Sierra Platinum first applies the inverse normal method to calculate the combined  $p$ -value and then generates the peak list based on the combined  $p$ -value.

Let  $p_i$  be the  $p$ -value for replicate  $i$ . To calculate the combined  $p$ -value, the  $p$ -values  $p_i$  of all replicates are transformed into probits  $\tau_i$  using the inverse cumulative standard normal distribution  $\Phi^{-1}$ :

$$\tau_i = \Phi^{-1}(p_i) \quad (3.19)$$

Since the  $p$ -values  $p_i$  are uniformly distributed for each window, the respective probits are normally distributed with mean 0 and standard deviation 1 due to the transformation. This step is done for each window and each replicate.

For each window, the combined probit is calculated based on the probits of the replicates. In the simplest case, the replicates are not correlated. Then, all replicates have equal weight and the combined test statistic  $\bar{\tau}$  for the  $n_r$  replicates is computed as:

$$\bar{\tau} = \frac{1}{\sqrt{n_r}} \cdot \sum_{i=1}^{n_r} \tau_i \quad (3.20)$$

Again, the test statistic  $\bar{\tau}$  follows a standard normal distribution.

The corresponding combined  $p$ -value  $p$  is calculated based on the cumulative standard normal distribution  $\Phi$ : the  $p$ -value is the one-sided, left cumulative probability calculated using the normal distribution with mean 0 and standard deviation 1:

$$p = \Phi(\bar{\tau}) \quad (3.21)$$

However, the replicates and thus the probits are expected to be correlated (see Section 3.3.12 for more details). Therefore, Sierra Platinum uses a weighted version of the inverse normal method based on the extension proposed by Hartung [40] that can cope with correlations. Hence, Sierra Platinum assigns weights to each replicate to be able to down-weight replicates that are of lower quality (see Section 3.3.3).

According to Hartung [40], the approximated correlation  $\hat{\rho}$  between the probits for  $n_r$  replicates of a window is

$$\hat{\rho} = 1 - s_\tau^2 \quad (3.22)$$

where  $s_\tau$  is calculated as given by Equation 3.16.

The correlation estimate  $\hat{\rho}^*$ , which will be used for the calculation of the combined probit is then calculated as

$$\hat{\rho}^* = \max \left\{ -\frac{1}{n_r - 1}, \hat{\rho} \right\} \quad (3.23)$$

For better readability of the equation to calculate the combined probit, the sum of weights is defined as

$$\omega = \sum_{i=1}^{n_r} \omega_i \quad (3.24)$$

and the sum of the squares of the weights is defined as

$$\hat{\omega} = \sum_{i=1}^{n_r} \omega_i^2 \quad (3.25)$$

whereby  $\omega_i$  is the weight for the  $i$ -th replicate. The default for Sierra Platinum is to use  $\omega_i$  of replicate  $i$  according to Equation 3.3 (Section 3.3.5).

The resulting combined probit is calculated as

$$\tau = \frac{\sum_{i=1}^{n_r} \omega_i \tau_i}{\sqrt{\hat{\omega} + [\omega^2 - \hat{\omega}] \left[ \hat{\rho}^* + \kappa \sqrt{\frac{2}{n_r+1}} (1 - \hat{\rho}^*) \right]}} \quad (3.26)$$

The parameter  $\kappa$  controls the significance level of the  $p$ -value as calculated in Equation 3.21. Hartung [40] experimentally estimated two values for  $\kappa$  from which one should be chosen. Sierra Platinum uses the first value proposed, i.e.,  $\kappa = 0.2$ .

Both values of  $\kappa$  described control the significance level well for positive correlations independent of the number of replicates and the variance of the weights. Negative correlations, however, are problematic. In particular in combination with a large variance of the weights, the actual significance level is higher than the one calculated with Equation 3.21. This would lead to false positive peaks.

In the case of multiple-replicate peak-calling, Sierra Platinum combines biological and/or technical replicates of the measurement of the same chromatin bound protein under approximately the same conditions in the same cell type or at least in similar cell types. Thus, it is expected that the replicates correlate positively (or at least not negatively) with each other. Therefore, negative correlation indicates that the replicates do not fit together and that they may result from an error in the experimental protocol. As a consequence, one would exclude those replicates from peak-calling. Hence, the chosen value of  $\kappa$  is suitable for the use in Sierra Platinum.

Sierra Platinum can also be used for peak-calling single replicates. In this case, the inverse normal method step is skipped and the final  $p$ -value is identified with the  $p$ -value of the replicate. All other steps are not affected by the number of replicates.

Similar to the conversion of the single replicate  $p$ -values into  $q$ -values (Section 3.3.10), Sierra Platinum converts the combined, multiple-replicate  $p$ -value

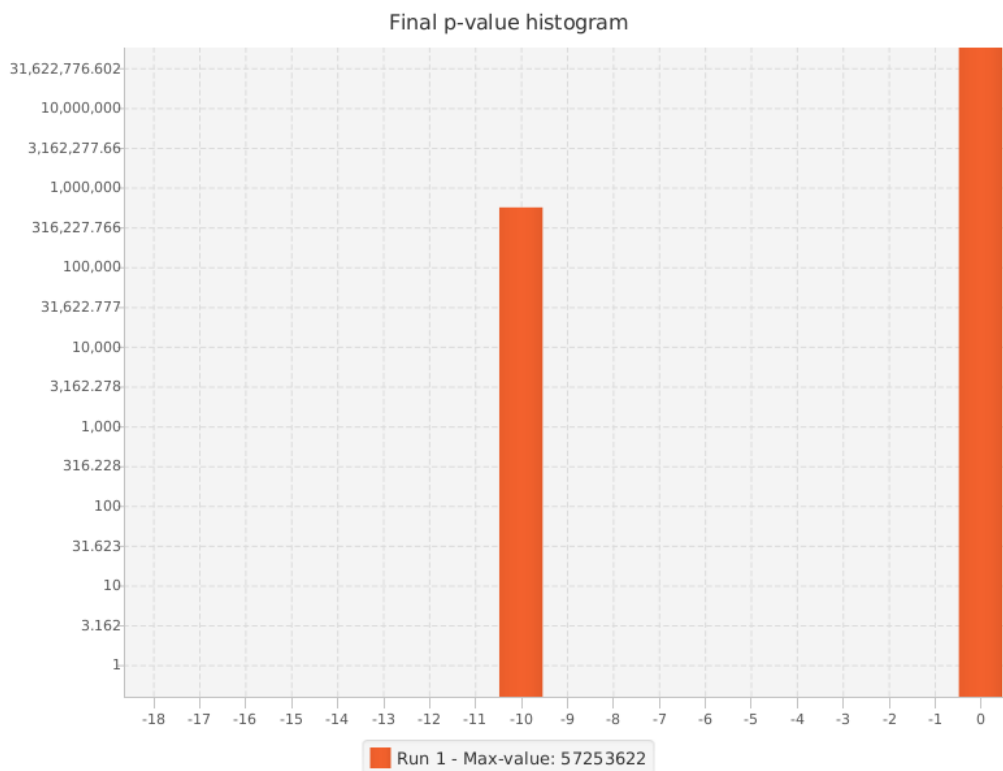


Figure 3.9: Example of a combined  $p$ -value distribution.

into a  $q$ -value for each window. The methods described in Section 3.3.13 are applied for the conversion and the resulting  $q$ -values are used in the subsequent computations.

**Interaction** The  $p$ -value correction method can be set using a drop-down-box (Figure 3.10). The available  $p$ -value correction methods are 'None', 'Holm-Bonferroni', 'Storey Simple', and 'Storey Bootstrap'. The default is set to 'Holm-Bonferroni' to obtain a conservative correction.

As the combined  $q$ -values are (un-)corrected combined  $p$ -values, we will refer to them as  $p$ -values in the subsequent sections.

## Visualization

The result of the combination of the  $p$ -values is shown in Figure 3.9. The combined  $p$ -values are binned into intervals from 1 to  $10^{-18}$  and visualized as bars. Both x- and y-axis are logarithmically scaled. The numerical method behind the  $p$ -value distribution produces values down to  $10^{-16}$ . Smaller values are assigned to bin  $10^{-17}$  for values of  $\bar{\tau} > -20$ , while values of  $\bar{\tau} \leq -20$  are assigned to bin  $10^{-18}$ .



Replicate	Weights	Status
UCSD 1:	0.999972	OFF
UCSD 2:	0.999906	ON
UCSD 4:	0.999956	ON
BI 5:	0.999961	ON
BI 6:	0.999941	ON
UCSF 7:	0.999856	ON

☐ Enable quality counting
 q-Value correction method: Holm Bonferroni
Recalculate

Figure 3.10: Overview of the options to weight replicates provided by the GUI of Sierra Platinum. The upper part contains one row per replicate showing the replicate identifier (left column), the assigned weight (middle column), and whether the replicate is used (ON) or not (OFF, right column). The weight checkbox on top of the middle column allows for disabling weights altogether (if unchecked). The lower part contains a checkbox that allows to enable or disable the computation of the quality of the peaks ('Enable quality counting', Section 3.3.17) and a drop-down-box that allows to select the  $q$ -value correction method (Sections 3.3.10 and 3.3.13). Pressing the button ('Recalculate', bottom right) starts the recomputation.

### 3.3.14 Filtering and Weighting

#### Visualization and Interaction

Sierra Platinum uses several parameters that influence the computation of the combined peaks. First of all, the user can decide whether or not to use the correlation correction based on Equation 3.23 (Section 3.3.13). If the correlation correction is disabled using the checkbox shown in Figure 3.8,  $\hat{p}^*$  is set to 0. Moreover, the user can decide whether or not to use a replicate (right column of Figure 3.10). Further, she can assign a weight to each active replicate (middle column of Figure 3.10) or disable weights altogether (weight checkbox in the first row of Figure 3.10). Finally, she can decide whether or not to compute the quality of the peaks (checkbox at the bottom of Figure 3.10; for a description please see Section 3.3.17).

During the initial run, default parameters are used: correlation correction is enabled, all replicates are enabled, the  $p$ -value correction method is set to "Holm-Bonferroni", and the weights are set to the  $\omega$  values of each replicate

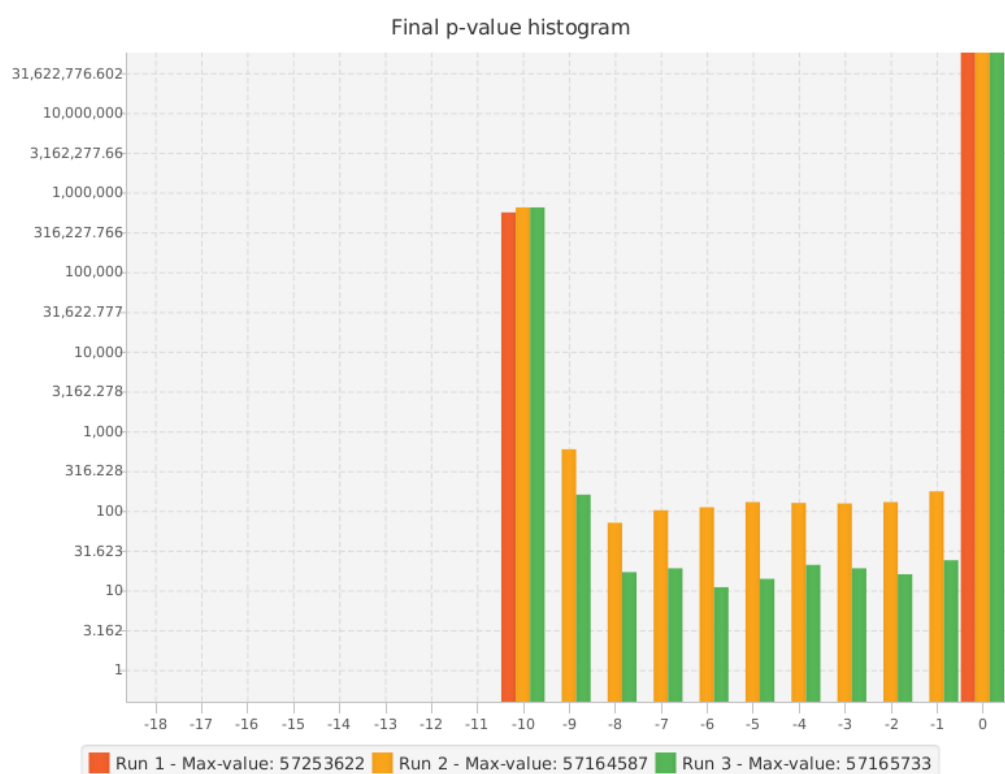


Figure 3.11: Comparison of the final  $p$ -value distributions with different weight settings. Each run is assigned a different color according to the color legend below the figure. The  $p$ -values of the bins are mapped to the x-axis while the amount of windows having the respective  $p$ -value are mapped to the y-axis. A logarithmic scale is used in both cases. Here, three different runs are shown assigned to the colors red (run 1), orange (run 2), and green (run 3). From the  $p$ -value distribution alone, run 1 would be preferred over run 3 over run 2.

(Equation 3.3, Section 3.3.5). Further, the peak quality is computed by default.

These parameters can and should be changed based on the results of the initial run. The single replicate peak-calling steps from the first part are not influenced by these parameters and thus this part does not have to be recomputed again. However, the second part of the computation (Section 3.3.13–3.3.18) should be performed again (see also Section 3.4.5). The results of all runs can again be analyzed using a histogram (shown in Figure 3.11) to compare the  $p$ -value distributions for different parameters settings. The color of each bar encodes a run with a specific parameter setting. As it is not advisable to use more than 8 different colors [81], only the results for the last 8 runs are provided. The results of each run are also stored separately by adding the number of the run to the file name (Section 3.3.18).

**Optimization**

This step is purely interaction. Only its results are used as parameters for the subsequent steps.

### 3.3.15 Agreement between the Multi-Replicate Result and the Single Replicate Results

**Quality Measures**

As a further quality measurement, Sierra Platinum calculates the agreement between the multiple-replicate result and the single replicate results. More precisely, it calculates the fractions of the significant windows that according to the combined  $p$ -value are also significant in the different replicates. Let  $C$  be the set of windows that are significant according to the combined  $p$ -value, i.e.,

$$C = \{w \in W | p^w < \hat{p}\} \quad (3.27)$$

where  $W$  is the set of all windows,  $p^w$  is the combined  $p$ -value of window  $w$ , and  $\hat{p}$  is the significance cutoff value.

Analogously,  $R_i$  is the set of windows that are significant according to the  $p$ -value for replicate  $i$ :

$$R_i = \{w \in W | p_i^w < \hat{p}_i\} \quad (3.28)$$

where  $p_i^w$  is the  $p$ -value of replicate  $i$  for window  $w$ , and  $\hat{p}_i$  is the significance cutoff value of replicate  $i$ . The agreement between the multiple-replicate result and the result of replicate  $i$  is thus

$$a_i = \frac{|C \cap R_i|}{|C|} \quad (3.29)$$

**Visualization**

This mutual agreement is visualized using a bar chart (Figure 3.12a). Each bar represents a replicate and the y-axis gives the percentage of agreement with the multiple-replicate peak-calling result.

This quality measurement eases assessing if the replicates agree with each other and thus with the combined results or if one replicate contributes much less than the other replicates to the final result. A high agreement of all replicates

with the multiple-replicate result shows that the replicates themselves agree in their peak-calls.

Again, for multiple runs, the overlap bar chart (Figure 3.12b) is extended with the overlap information of each run.

### 3.3.16 Computing Peaks

#### Method Narrow Peaks

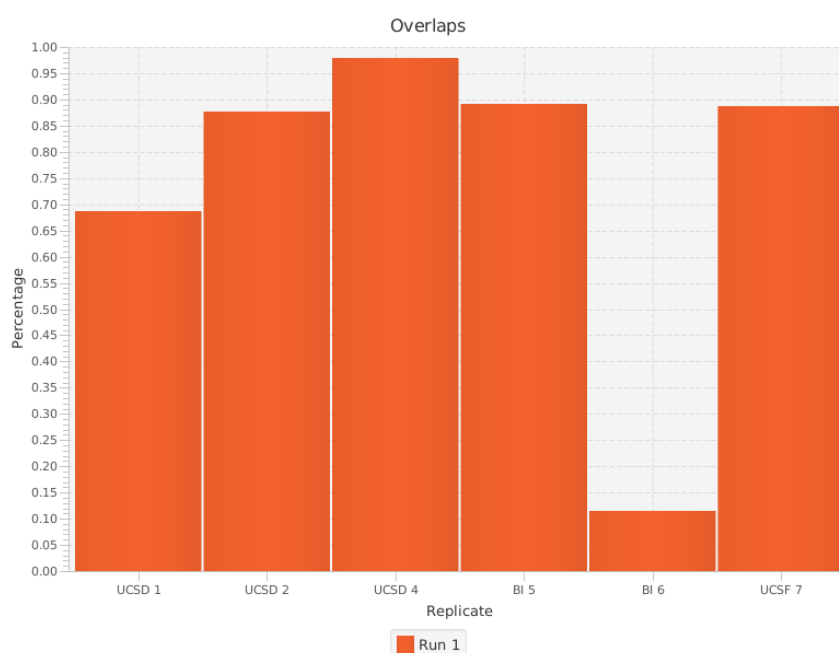
Sierra Platinum adopts the computation of the narrow peaks from MACS [90]. Significant windows overlapping in their genomic position are merged into the same peak. A window is significant if the combined  $p$ -value of this window does not exceed the user-defined significance cutoff. Sierra Platinum reports the lowest combined  $p$ -value of the combined  $p$ -values of the windows contributing to the peak.

Narrow peaks should be calculated for those modifications known to produce very sharp peaks such as H3K4me3.

#### Method Broad Peaks

It was found that many histone modifications form broad domains of consecutive modified nucleosomes. Therefore, Sierra Platinum also includes the computation of broad peaks into Sierra Platinum and again adopted the procedure from MACS [90].

If two peaks are less than two window sizes apart from each other, then they are joined into the same broad peak. This copes with the fact that ChIP-seq is a measurement of a population signal rather than a single cell protocol and that the epigenetic state is controlled by stochastic processes. Thus, at the time of measurement, nucleosomes might be completely unmodified or modified in many cells of the population even though they are usually modified. This results in gaps between the peaks. These gaps can also be an artifact of the experimental or computational method. In both cases, one can close the gap computing broad peaks.



(a) Example of the overlap visualization for all replicates for a single run. The x-axis shows the replicate number while the percentage of overlap of the respective replicate is mapped onto the y-axis.



(b) Comparison of the overlap of the replicates for multiple runs. During the first run (red) all replicates were used while during run 2 (orange) only replicates 3 and 4, and during run 3 (green) only replicates 1, 2, 3, and 6 were used. While for run 3 the overlap of the individual replicates is more or less balanced, for run 2 this is not the case: replicate 4 overlaps more than 90% with the final result while replicate 5 overlaps less than 10% with the final result.

Figure 3.12: Comparison of the overlap of the peaks between the replicates.

### 3.3.17 Computing Peak Quality

#### Quality Measures

Finally, Sierra Platinum provides a last quality control: the read quality within the peaks. Therefore, Sierra Platinum calculates the median read quality for each peak (which is also exported together with the peaks, Section 3.3.18).

#### Visualization

For each replicate, Sierra Platinum again provides two boxplots for the median peak quality distribution in the experiment and in the background similar to those introduced in Section 3.3.3. As shown in Figure 3.13, the GUI of Sierra Platinum provides an overview over all data sets. The whiskers of the boxplots represent the minimum and the maximum value of the data set, respectively, and the background of the plot shows the different quality levels thus assisting the user to interpret the boxplots.

Replicates with a very low median peak quality distribution might be excluded since they only contribute with low quality—and thus suspiciously—data to the final result. Further, strong differences between the median peak quality distributions of experiment and background indicate suspicious results. As detailed analyses show, such replicates should be removed from the analysis (cf. Section 3.5).

#### Optimization

The peak quality method was optimized like the read quality method (Section 3.3.3).

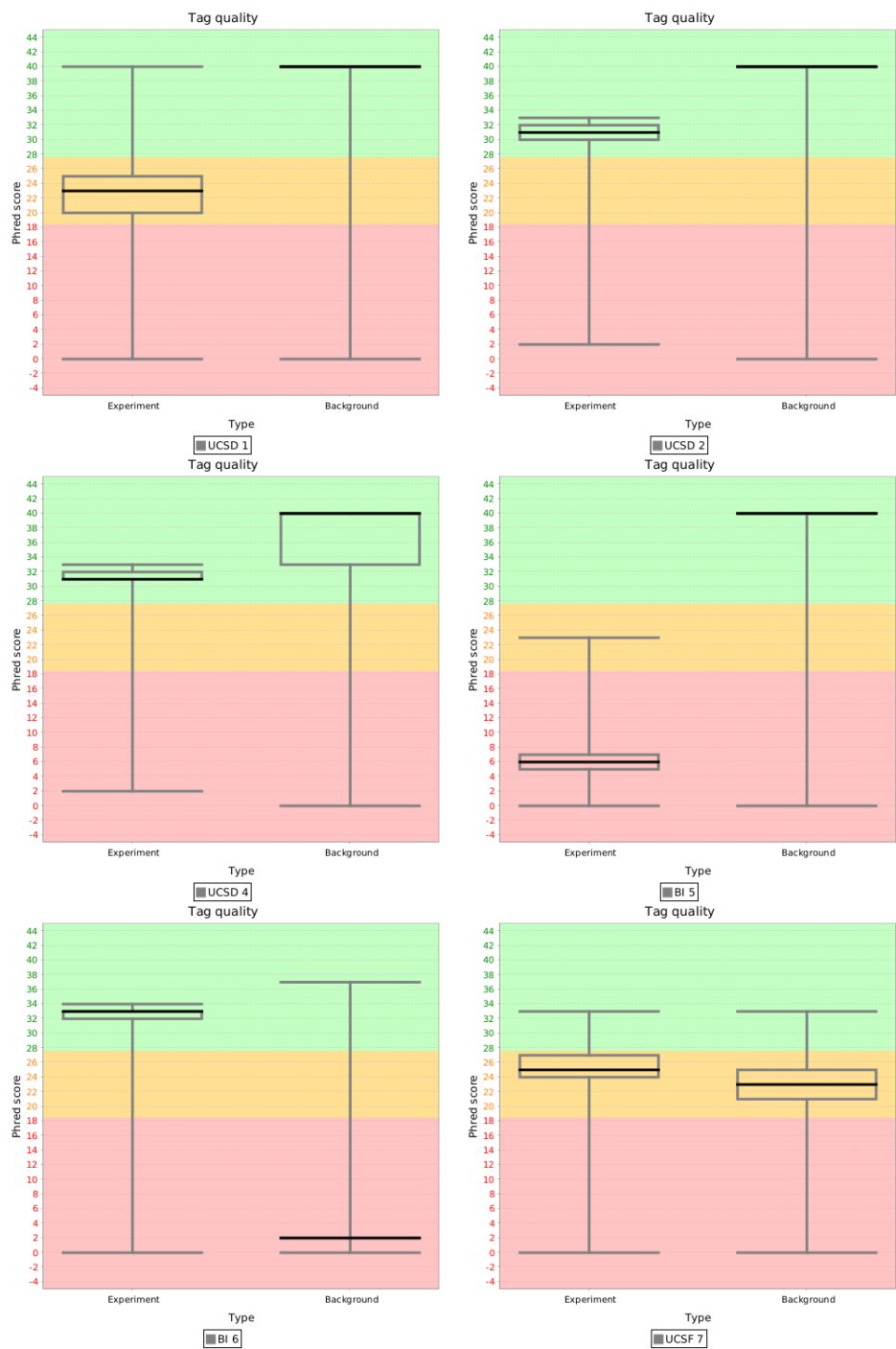


Figure 3.13: Final peak quality boxplots for experiment and background of 6 replicates.

### 3.3.18 Exporting Peaks

#### Method

After each run of the multiple-replicate peak-calling process the broad and the narrow peaks are exported and saved as *bed* or *csv* files with the *p*-value as the score field on the server. Additionally, it is possible to export the peaks on the client using the GUI of Sierra Platinum.

Sierra Platinum uses *Google GSON* [10] for all data storage. It is usually faster and more robust than the previously used, dated object serialization provided by Java. For efficiency reasons, all files are compressed.

More information is provided in Section 3.4.



## 3.4 Implementation

### 3.4.1 System

Sierra Platinum is completely written in JAVA 8 and uses JavaFX as API for the GUI. Therefore, it is possible to run Sierra Platinum on Windows, Linux, and Mac. The libraries used are Apache Commons IO [4], Apache Commons Logging [5], Apache Commons Math [6], Apache Commons Net [7], Apache Commons VFS [8], HTSJKD [2], JFreeChart [11], Gson [10], and JSch [12].

Choosing Java supports the generation of graphical user interfaces in a straight forward way. Moreover, its language concept based on software engineering principles supports maintainability.

While the computational requirements could be substantially optimized, they are still high and require a powerful workstation or a server. Since the input data for the peak-calling process can be very large, its calculation needs a performant workstation that is more powerful than current standard desktop computers. Therefore, the program is split into two parts:

**Sierra Platinum Server:** performs the peak-calling itself and computes all quality measures. It takes its parameters from the client and transfers the result to the client. The server is normally run on high-performance machines with sufficient resources, i.e., CPUs with several cores, a large amount of memory, and a high I/O bandwidth.

**Sierra Platinum Client:** the GUI that allows the user to select the data to use (replicate data sets), to set and adjust the parameters, to assess the results, and to export the results on the users's client. The client can be run on almost any current standard desktop computer.

Additionally, Sierra Platinum can be used in batch mode from a command line. To do so, the user has to create a configuration file for the server. This configuration file can be created with the client by selecting all replicates within the GUI and exporting the client configuration. This can be useful, if, e.g., no direct server connection is possible because of security policies in the lab environment. In principle, a skilled user can create the configuration files manually since they are gzipped JSON files. An example of the configuration is provided in the Section 3.4.10. With this configuration file, it possible to run the server in batch mode without the need of a client connection. The server will then read all the input data, perform the computations export all results,

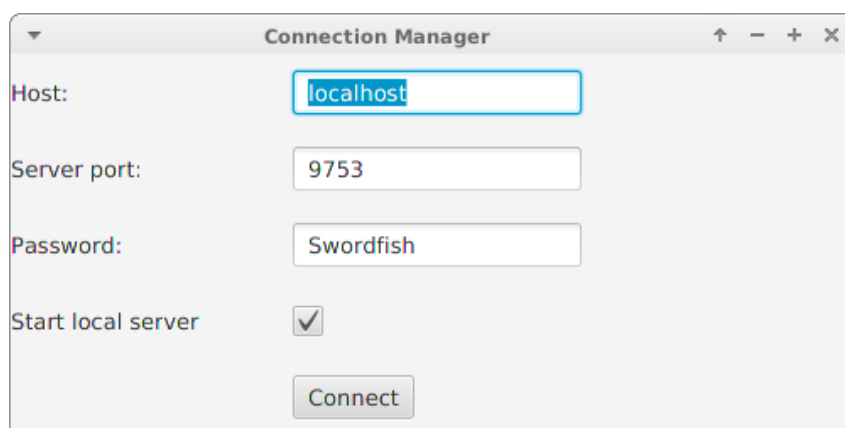


Figure 3.14: The server connection window of Sierra Platinum.

and then terminate. The user can import the results as a datamapper in the client and then check the results offline, that is, without any server running.

### 3.4.2 Client GUI

The client lets the user interact with the server for selecting the input data of the peak-calling process and for adjusting the parameters. After the calculation, the client visualizes the quality control steps allowing the user to assess the quality of the replicates and of the resulting peaks.

#### Communication with Server

First, the Sierra Platinum Client is connected to a server using the connection dialog (Figure 3.14). If the calculation should be performed locally, since the data set is small or because the local computer has enough resources, the client can create its own instance of the server ('Start local server' checkbox). Besides the name of the server ('Host'), the 'Server port' is provided on which the server is listening to connections and commands from potential clients. Additionally, it is possible to connect more than one client to the server and to secure the server with a password since it is possible to cancel jobs in the GUI. The communication protocol is designed as stateful API, where the client creates a connection to the server for each query. After accepting the query, the server responds on the same stream to the client and the client closes the connection. Since the server can not establish a connection to the client, the client constantly queries the state of the server using a heartbeat query.

After the calculation, the server sends a complete progress message with the next heartbeat signal of a client. After evaluating this message, the client

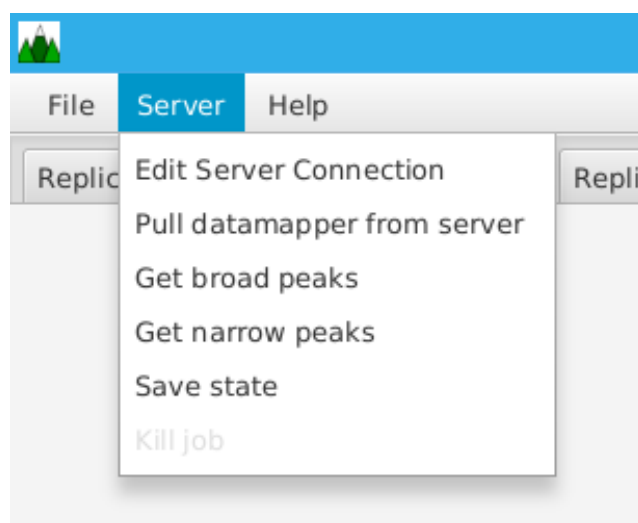


Figure 3.15: The Server Menu

queries for the data mapper object and shows it after receiving it. Furthermore, it is possible to pull the data mapper from the server if the client was not connected to the server when the server finished the job.

The connection dialog is opened automatically after starting the client. Further, a new connection can be opened any time using the Menu entry 'Server → Edit Server Connection' (Figure 3.15).

### 3.4.3 Replicates, Parameters, and Starting Computation

To call peaks for a set of replicates, first a list of replicates is created followed by setting the relevant parameters for the process (Figure 3.17). The complete settings—list of replicates and parameter settings—can be saved to file using the menu 'File → Config Management → Save config'. Alternatively, the settings can be loaded by using 'File → Config Management → Load config' (Figure 3.16).

Finally, the process is started by pressing the 'Start' button (Figure 3.17). The progress of the computation is shown by the progress bar to the right of the 'Start' button.

#### Editing the list of replicates

The user adds replicates using the 'Add replicate' button (Figure 3.17). For each of the added replicates, the file associated to the experiment and the file associated to the background of the replicate are selected on the local computer or on the server computer by using a file system browser [8]. A

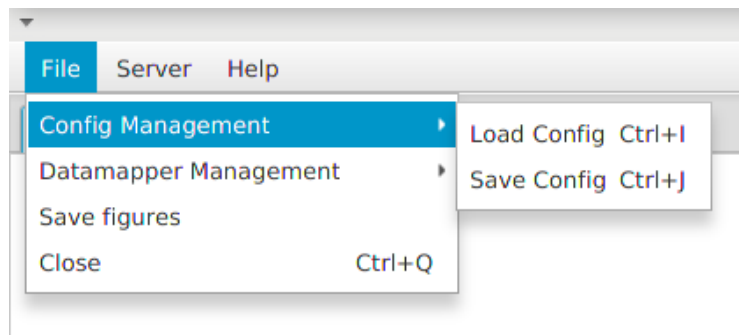


Figure 3.16: The File Menu showing the sub menu for storing and loading a configuration.

context sensitive menu in the main window allows to change the files associated with the replicates' experiment and background and to delete replicates.

### Setting parameters

The following settings for the current computation can be adjusted (Figure 3.17): 'window size' and 'offset' (Section 3.3.1), ' $p$ -value cutoff' (Sections 3.3.11, 3.3.15, 3.3.16), and the 'number of threads' that should be used for computation. Further, the 'job name' can be set. The job name is used for assigning names to the files used for the information exported.

### 3.4.4 Quality Control

After computation, for each replicate, a 'Replicate' tab is created showing all relevant information computed for this replicate (Figure 3.18):

- the estimated Poisson distribution for experiment and background (Figure 3.18, top left; Section 3.3.4, Figure 3.5a),
- the estimated Poisson distribution for adjusted experiment and background (Figure 3.18, bottom left; Section 3.3.7, Figure 3.5c),
- the count distribution for experiment and background (Figure 3.18, top middle; Section 3.3.5, Figure 3.5b),
- the  $p$ -value distribution over all windows (Figure 3.18, bottom middle; Section 3.3.4, Figure 3.6),
- the mapping quality (Figure 3.18, top right; Section 3.3.3, Figure 3.4),

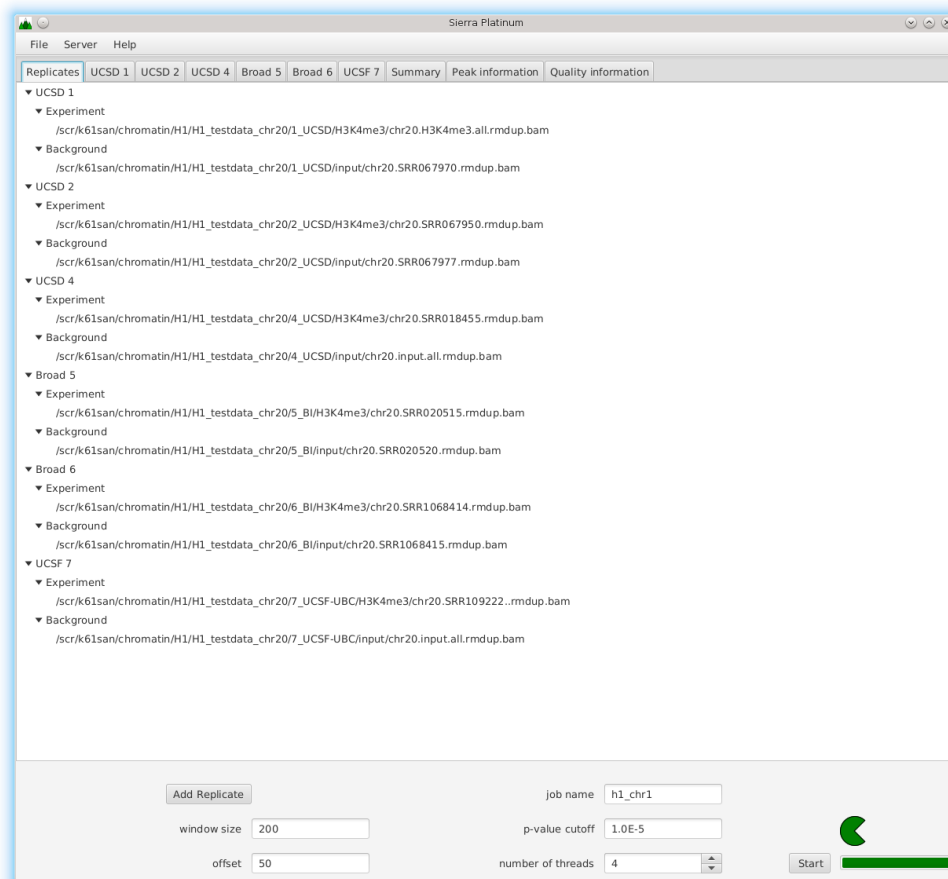


Figure 3.17: The settings-tab in the GUI of Sierra Platinum. In the large, middle window, the replicate information is shown. For each replicate, the file names for the experiment and the background are given. In the lower part, the 'Add Replicate' button allows adding additional replicates. Below this button, the parameters for the 'window size' and the 'window offset' can be changed. Further, the 'Job name', the ' $p$ -value cutoff', and the 'number of threads' can be assigned. Finally, to the right, the 'Start' button allows starting a computation and the progress bar to the right of this button shows the progress of the computation. A context sensitive menu in the main window allows to change the files associated with the replicates' experiment and background and to delete replicates.

- the distribution of significant windows per chromosome (Figure 3.18, bottom right; Section 3.3.11, Figure 3.7).

This supports assessing the quality of each of the replicates and for deciding, how to adjust the combination of replicates for the final results.

### 3.4.5 Correlation Information, Recalculation Parameters, and Restarting Computation

The ‘Summary’ tab (Figure 3.19)

- shows the Pearson Correlation of the replicates (Figure 3.19, left top; Section 3.3.12, Figure 3.8),
- allows for enabling or disabling the correlation based correction (Figure 3.19, left bottom, checkbox; Section 3.3.12, Figure 3.8),
- allows for setting weights affecting the combination of the single replicates for creating the combined peaks (Figure 3.19, right top, middle column; Section 3.3.14, Figure 3.10),
- allows for enabling (ON) or disabling (OFF) a replicate (Figure 3.19, right top, right column; Section 3.3.14, Figure 3.10),
- allows for setting two parameters and restarting the computation (Figure 3.19, right bottom row; Section 3.3.14, Figure 3.10).

The user can enable correlation correction and select a weight for each replicate, which is used while combining the  $p$ -values. Additionally, it is possible to exclude one or more replicates from the recalculation and to change the  $q$ -value correction method. Moreover, quality counting can be enabled or disabled.

The defaults used during the initial run (Section 3.3.14) are: correlation correction is enabled, all replicates are enabled, the  $p$ -value correction method is set to “Holm-Bonferroni”, and the weights are set to the  $w$  values of each replicate (Equation 3.3, Section 3.3.5). Further, the peak quality is computed by default.

Finally, the recalculation is started using the new settings. Therefore, it is necessary to send the job to the server again.

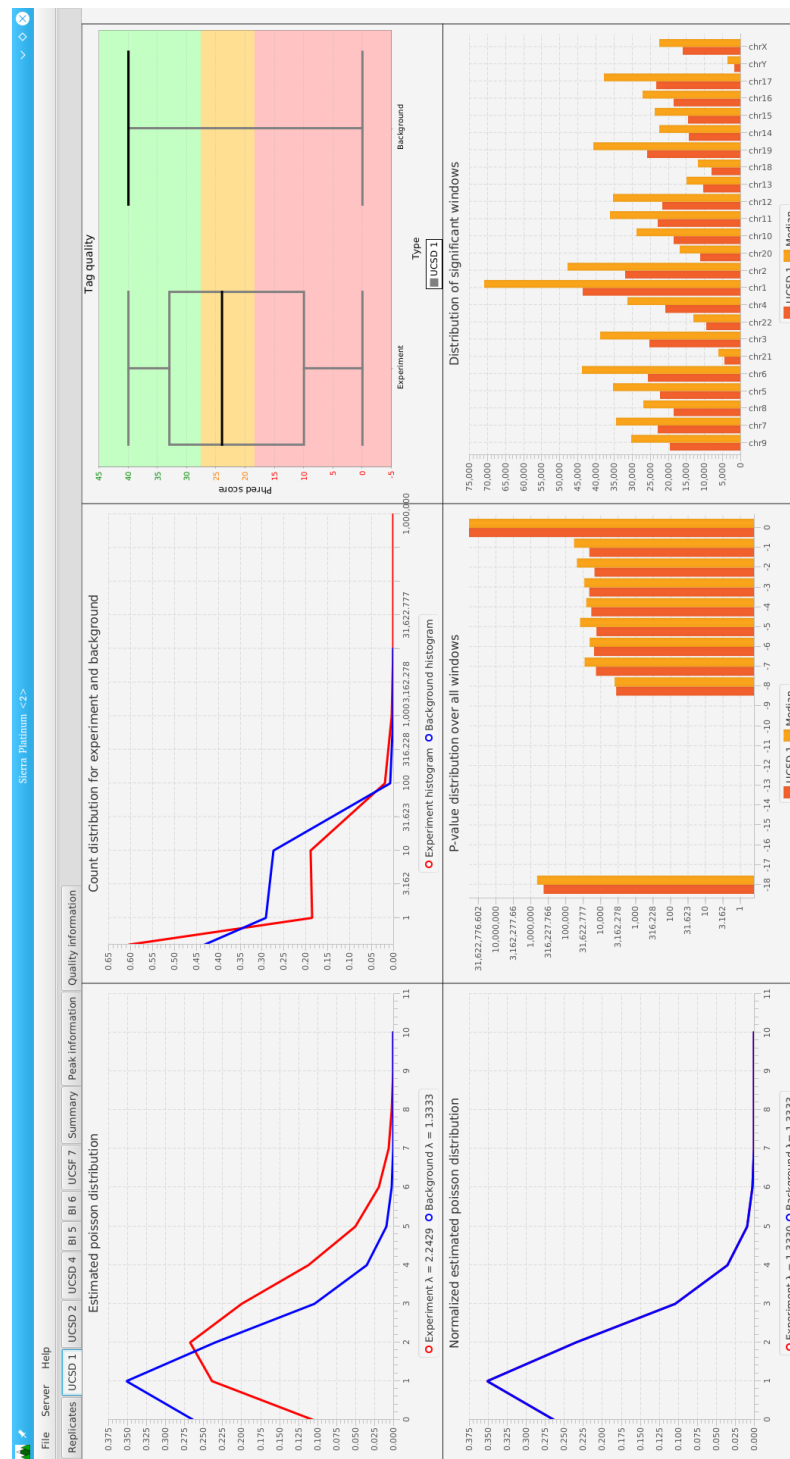


Figure 3.18: A replicate tab in the GUI of Sierra Platinum. Top left: the estimated Poisson distribution for experiment and background (Figure 3.5a). Top middle: the count distribution for experiment and background (Figure 3.5b). Top right: the mapping quality (Figure 3.4). Bottom left: the estimated Poisson distribution for adjusted experiment and background (Figure 3.5c). Bottom middle: the  $p$ -value distribution over all windows (Figure 3.6). Bottom right: the distribution of significant windows per chromosome (Figure 3.7).

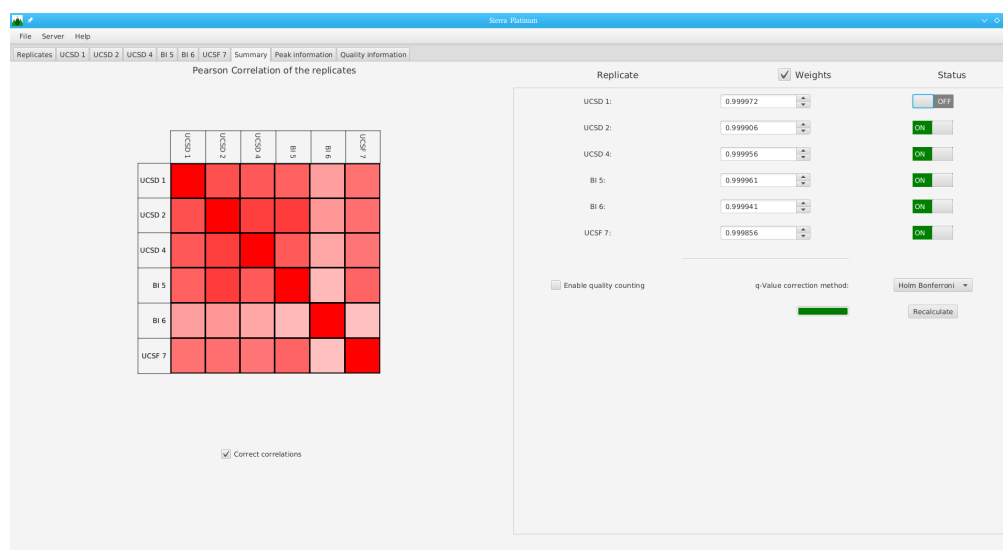


Figure 3.19: The 'Summary' tab in the GUI of Sierra Platinum. Left top: the Pearson Correlation of the replicates. Left bottom: checkbox allowing for enabling or disabling the correlation based correction (Figure 3.8). Right top, middle column: weights affecting the combination of the single replicates for creating the combined peaks. Right top, right column: enabling (ON) or disabling (OFF) a replicate. Right bottom row: two parameters and restarting the computation (Figure 3.10).

### 3.4.6 Peak Information

The 'Peak information' tab shows two diagrams:

- the final  $p$ -value histogram showing the combined  $p$ -values for all windows (Figure 3.20, left; Section 3.3.13, Figure 3.9; Section 3.3.14, Figure 3.11)
- the percentage of agreement of each replicate with the multiple-replicate result (Figure 3.20, right; Section 3.3.15, Figure 3.12a; Section 3.3.14, Figure 3.12b)

### 3.4.7 Quality Information

The 'Quality information' tab shows the distribution of the peak quality for all replicates (Section 3.3.17, Figure 3.13).



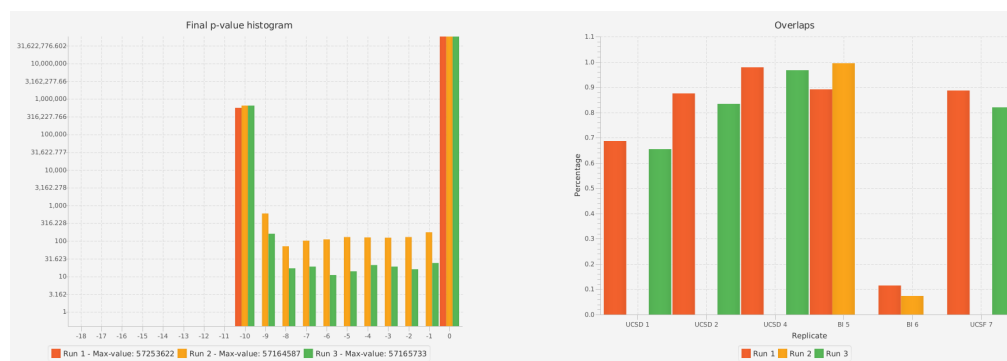


Figure 3.20: The ‘Peak information’ tab in the GUI of Sierra Platinum. Left (see also Figure 3.11): the final  $p$ -value histogram showing the combined  $p$ -values for all windows. Right (see also Figure 3.12b): the percentage of agreement of each replicate with the multiple-replicate result. Three runs with different setting for weights and different replicate combinations are shown.

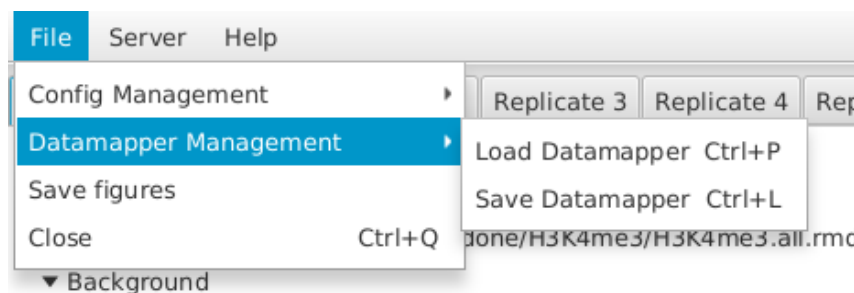


Figure 3.21: The File Menu showing the sub menu for storing and loading the data mapper.

### 3.4.8 Additional Functionality

#### Loading and saving the data mapper

Sierra Platinum automatically exports a data mapper file for each run, which contains all information that are presented in the GUI. With this mapper, it is possible to archive all the additional data created by Sierra Platinum for the peak-calling and to analyze this data even without a server connection. The data mapper can be exported by using the menu entry ‘File → Datamapper Management → Save Datamapper’ and imported by using ‘File → Datamapper Management → Load Datamapper’ (Figure 3.21).

The client configuration can be exported and imported like the Datamapper using the ‘Config Management’ menu entry.

## Export graphics

All figures created by Sierra Platinum can be exported as a bundle or as a single figure for later utilization as *.png* files. The user can export all figures by using the Menu Entry 'File → Save figures' (Figure 3.21). Single figures can be exported by clicking them with the right mouse button.

### 3.4.9 Server

When starting the server, the user can select the maximum number of threads, which are used during IO intensive operations. After every calculation, the server exports the results in bed and csv format to disk. Additionally, it stores the current data mapper and a log file with all information about the most recent calculation.

The server can be started in server or in batch mode. In server mode, it accepts connections from clients. When the server is running a job, it is sending progress information to each client. Only one job can be run at a time, due to the large amount of resources needed. Therefore, the server is locked while performing the job, meaning, that it does not accept any additional jobs. In batch mode a previously created job configuration file is provided to the server that will compute all results and terminate afterwards.

### 3.4.10 Server Configuration File

The server configuration file for batch mode can be exported from the client as described in Section 3.4.8. Since the configuration file is a gzipped JSON file, it is also possible to create it manually or automatically for using the server in pipelines. The syntax (format) of the file is given by the following example (white space and line breaks added for readability purposes):

```
{"replicates":  
  [{"experiment": "expA.bam",  
    "background": "backA.bam",  
    "name": "ReplicateA"}],  
  [{"experiment": "expB.bam",  
    "background": "backB.bam",  
    "name": "ReplicateB"}]},  
  "windowSize": 200,
```

```
"offset":50,  
"pvaluecutoff":1.0E-5,  
"peakmode":false,  
"numCores":4,  
"jobName":"Example"  
}
```

The first seven lines show, how an example with two replicates is generated. The keyword “replicates” is followed by a list (between ‘[’ and ‘]’) of replicates. Each replicate is delimited by the curly brackets and contains three fields: the “experiment” file name (e.g., “expA.bam”), the “background” file name (e.g., “backA.bam”), and the replicate “name” (e.g., “ReplicateA”). Furthermore, the following parameters have to be provided: the window size, the window offset, and the p-value cutoff. The variable ‘peakmode’ is only for internal use and should be always ‘false’ if the configuration file is created manually. The “numcores” entry determines the maximal number of threads used for the computation. Finally, the “jobName” describes the file prefix which is added to the output file’s names. After creating the configuration file, all white space (including tabs and line breaks) has to be removed to create a JSON file containing exactly one line, and the file has to be compressed with gzip.

### 3.4.11 Service Implementation

The computational efforts for ChIP-seq analysis require hardware that may not be available in labs without dedicated bioinformatics infrastructure due to the size of the input files and the complexity of the algorithms that combine multiple samples. To overcome this limitation, a service implementation of Sierra Platinum called Sierra Platinum Service was developed that provides access to the full functionality of Sierra Platinum in form of a web-based service hosted at [sierra.sca-ds.de](http://sierra.sca-ds.de). It provides a publicly available web service that combines user management, job control, and a queuing system as well as mechanisms for uploading the input data and for downloading all results. It creates a dedicated Sierra Platinum Server that allows the user to upload, analyze, inspect, and manipulate his ChIP-seq data using the Sierra Platinum Client with very little local resource consumption. Finally, the user can download all results—analysis results as well as the final peaks. A convenient docker image was developed for an easy deployment of private instances of the service, e.g., for institution-wide use.

## Technical Realization

The server is hosted within a docker container (see Figure 3.22), which provides a Java JRE for the Sierra Platinum Service, a fully configured nginx web server with php5 support, and an SQLite database that stores the user management of the service. The mail transmission is implemented by using sSMTP that allows using an existing email address without the need to setup an email server within the service.

Since the Sierra Platinum Service is embedded in a docker container, it can easily be deployed by pulling the Git repository <https://github.com/sierraplatinum/sierra-service> and running the scripts `build.sh` for building the container and `run.sh` for starting it. At this stage, the service can be configured by specifying TCP ports, the email address, and resource limitations such as the number of concurrent Sierra Platinum Service instances or threads. To handle the limited number of Sierra Platinum Service instances, a queuing system was implemented to handle all user requests. Within the docker container all services start automatically. The upload mechanism was implemented in the client/server core. To address security concerns, every user of the service is assigned his own FTPS directory and is jailed to it.

The client checks the validity of the uploaded files and the server can compute missing `.bam` indices. Interrupted uploads can be continued on the fly to accommodate for the large size of the input files.

## Usage and Interaction

The service requires registration with a valid email address and allows the user to start a dedicated Sierra Platinum Server for which he received the necessary credentials by email. A Sierra Platinum Service runs for 72 hours or until termination by the user. During this time, the user may disconnect from and reconnect to the server at any time. At the end of the Sierra Platinum Service's life time, all data is deleted from the server hardware.

To use the Sierra Platinum Service, the user connects with his credentials through the Sierra Platinum Client and first uploads his data as bam files using the integrated FTPS client. Then, the peak-calling can be started. Afterwards, quality control information can be visually inspected (see Figure 3.18) and parameters may be adjusted as for any local installation of Sierra Platinum. At any time, the results file can be downloaded for further, local analysis.

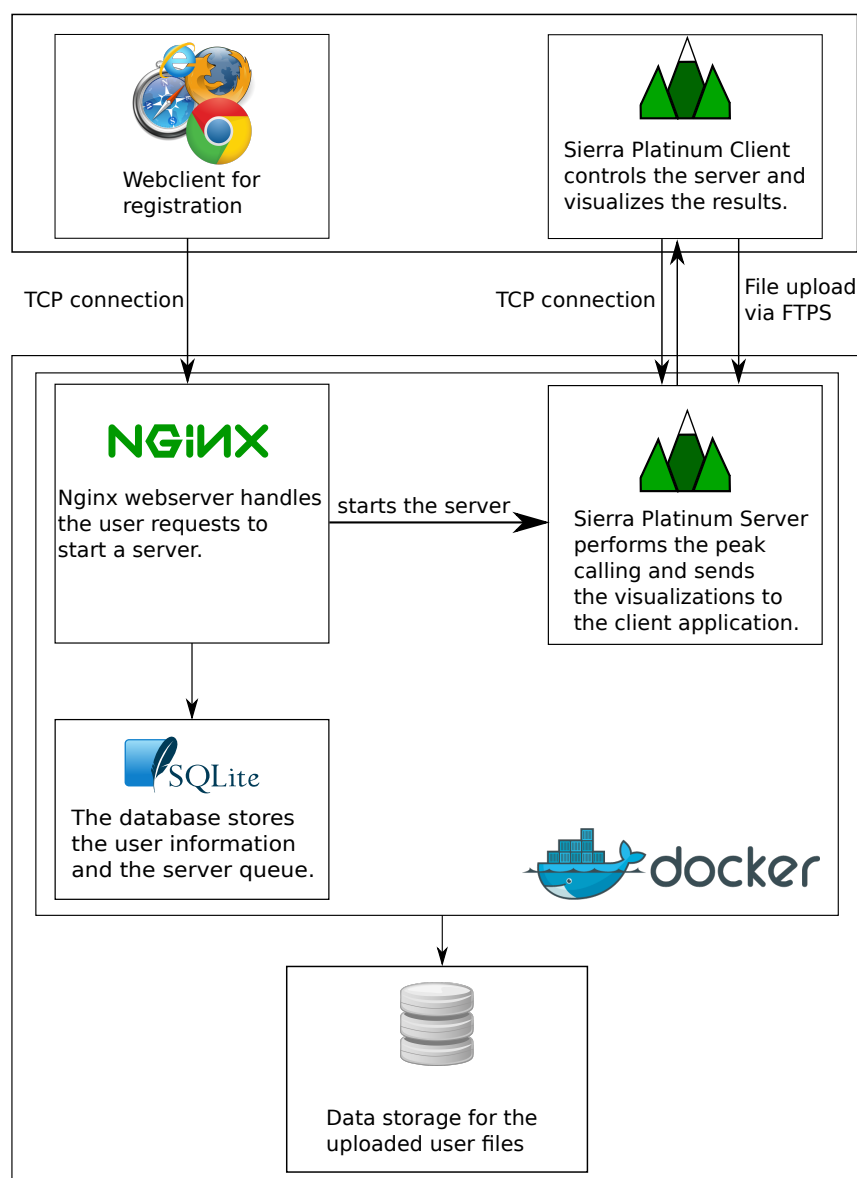


Figure 3.22: Overview of the Sierra Platinum Service container. The web server handles the user registrations and starts the Sierra Platinum Server after a valid activation. All necessary user information are stored in an SQLite database. After a successful registration the user can upload his data with the Sierra Platinum Client and start the computation. Afterwards, it is possible to export all results and visualizations within the client.

**Limitations**

The Sierra Platinum Service architecture is currently designed as a split infrastructure. The registration and the validation of a new user is handled by a web interface whereas the data upload, the data processing, and the data presentation is implemented within a Java GUI application. Therefore, the user needs an up-to-date Java installation on his client. Moreover, the data upload of the input files can take a long time depending on the user's internet connection and the input size. If the user has a very slow upload rate, the maximal runtime of the service may be exceeded before finishing the upload. Further, the user needs a valid email address for the registration process.

Currently, the Sierra Platinum Service instance at `sierra.sca-ds.de` is able to compute 5 jobs simultaneously. If necessary, the service can be extended easily to provide more slots since it is running on a cluster system.

## **3.5 Benchmark Data Set**

### **3.5.1 Context**

Testing and benchmarking are essential steps while implementing new methods. Hereby, testing refers to the robustness of the new tool: showing that even with erroneous or bad data, the tool still produces reasonable results without crashing. Benchmarking means running a tool with data for which the ideal results are known and then calculating how close the computed results match the ideal results. Typically, benchmarking results are compared to those of existing tools/approaches performing the same task. While data sets for tests are relatively easy to design, designing benchmarking data sets poses a major challenge.

### **3.5.2 State-of-the-Art and Gaps**

Surprisingly, even though there are tools to simulate the ChIP-seq experiments, no benchmarking data set for peak-calling is published so far. Koohy et al. [49] and Wilbanks et al. [86] already described the problem and therefore compared the peak-calling results of different peak-callers based on the number of shared peaks of 3 different transcription factors.

As benchmarking data sets are not available, workarounds are used, e.g., using real data instead of artificial data. However, for real data the 'ground truth' is often not known and can just be approximated by verifying the results using known information.

Depending on the tools task, this can be done in different ways. If already other tools exist that fulfill the task, it can be asked how much of the results of the other tools can be recovered by the new tool. However, in those cases it is difficult to decide which tool is better because the reliability of the results can not be decided upon, if the results differ.

Another frequent approach is to use experimentally validated results. In the case of differential expression analysis, for example, qPCR results or spike-in genes are used. However, in this case the set of benchmarking data points is very small since experimental verification can not be done for all genes and also has it's own limitations.

In the case of peak-calls for ChIP-seq data, two other approaches are often chosen. One approach is to show that the resulting peaks are similar to the results obtained when measuring the same results with ChIP-chip, i.e., the first

step of the experimental procedure is the same but the sequences pulled out are measured using a chip instead of being sequenced. The other approach is to test the tool with a transcription factor for which the binding motif is known. The benchmark consists of testing whether the peaks detected with the new tool contain a binding site according to the binding motif.

For the first approach, recall and false discovery rate can only be estimated since ChIP-chip, too, does not guarantee to produce perfect results. For the second approach, one can not guarantee that all binding sites are found since the binding motif may be inaccurate and binding site prediction can be erroneous, too. Furthermore, a binding site does not ensure that the measured transcription factor was bound and thus, a peak is expected. In other words, the second approach does not provide an estimate for the negative outcomes of the experiment.

To be able to compare the new peak-calling method (Section 3.3), several benchmarking data sets were created. Four statistical measures were applied on the data sets for assessing the quality of peak-callers. In this section, the method for creating these benchmarking data sets (Section 3.5.5) and the proposed benchmarking data sets (Section 3.5.6) for assessing the quality are described.

### 3.5.3 Goal

In general, benchmarking data sets should provide a combination of data sets with the following properties:

*Ideal case:* Even though the real data sets will never be 'ideal', the ideal case should always be part of a benchmarking data set. A method, which produces wrong/bad results in the ideal case, can not be assumed to produce reliable results for non-ideal data and thus should not be applied to real data.

*Single noise case:* For any source of noise, there should be at least one data set simulating this type of noise varying the parameters representing this type of noise. In this way, the robustness of the method with respect to the different sources of noise can be estimated.



*Multi-noise case:* At least one data set should be designed to demonstrate the performance of the tool when different sources of noise appear together in the same data set. Therefore, data sets with a useful combination of different sources of noise should be part of the benchmarking data set.

*Real case:* A data set, which is inspired by the sources and the strength of noise of real data would be desirable.

### 3.5.4 Challenges

Major challenges for constructing benchmarking data sets are:

1. to define a model that produces artificial data that looks like real data
2. to define which sources of noise exist
3. to model the sources of noise in the chosen model system

### 3.5.5 Benchmarking Data Set Creation

ChIPsim [46], an R package to simulate ChIP-seq, was used to generate a benchmarking data set for peak-calling that shows characteristics of histone modifications such as broad domains. Instead of using the default model, the nucleosome density example in the manual was used as the basis and several changes were made to adapt it to the test scenario. In principle, the simulation procedure consists of the following steps:

1. Generate a genome sequence
2. Generate features using a Markov model
3. Generate a signal density for all features (i.e., how much signal at a specific base is given to the annotated feature)
4. Calculate the read density according to the signal density and the fragment length
5. Sample reads from the genome according to the read density
6. Sample the base quality for each base according to a defined quality distribution
7. Introduce sequencing errors based on base quality

Table 3.7: Chromosome names and lengths

chromosome	length
chr1	$1 \cdot 10^5$
chr2	$2 \cdot 10^5$
chr3	$3 \cdot 10^5$
chr4	$4 \cdot 10^5$
chr5	$5 \cdot 10^5$
chr6	$6 \cdot 10^5$
chr7	$7 \cdot 10^5$
chr8	$8 \cdot 10^5$
chr9	$9 \cdot 10^5$
chr10	$10 \cdot 10^5$

This procedure was used in two configurations: one time to generate the background and one time to generate the experiment data. Some aspects of the configurations stay the same in both. The genome for background and experiment is the same enabling mapping the simulated reads to the same genome. Ten chromosomes were generated with different lengths as given in Table 3.7. Each chromosome is obtained by sampling as many bases from the set of DNA bases as desired. The Markov chain generating the features requires a length parameter. This length corresponds to the length of the genomic region assigned to the feature. A unique feature length of 146 bases was used for all states, i.e., each feature corresponds to exactly one nucleosome. The fragment length for sequencing follows a normal distribution with a mean of 200 bases and a standard deviation of 4 bases. Furthermore, the values were bound to the interval [150; 250]. Thus, the simulated fragments will have a length between 150 bases and 250 bases and are on average 200 bases long.

### Background data

Trivial features were generated using a one state Markov chain as model for the background corresponding to an unspecific antibody. The density was taken from a  $\Gamma$ -distribution with shape parameter  $k = 1$  and scale parameter  $\theta = 20$ . This setting was suggested by the nucleosome density example in the tutorial of ChIPsim. Calculating the read density and sampling the reads are performed as provided by ChIPsim. The read quality is generated either in “high”, “mid”, or “low” mode, which are explained below. The optimal sequencing depth was taken from the tutorial (number of reads equal to 10% of the genome length)

and was varied to 1% for under-sequenced and 40% for over-sequenced data, respectively.

### Experiment data

A two state Markov chain of order 1 is used to generate the features for the experiment data. One state represents the background, i.e., the noise which is unspecific to the theoretical antibody used. Its characteristics are the same as those of the background data since both represent basically the same, unspecific binding to the chromatin.

The second state represents the experimental data, i.e., the modified histone the experiment wants to measure. Sticking to an example in the tutorial, a single parameter Pareto distribution was used to generate the density for the specific histone. The shape parameter  $r$  is set to 5 as in the tutorial. The average density is calculated based on the background model as

$$avgDens = k \cdot \theta \cdot e \quad (3.30)$$

where  $k$  is the shape parameter and  $\theta$  is the scale parameter of the background  $\Gamma$ -distribution, while  $e$  is the enrichment over the background. The optimal enrichment  $e$  is set to 5 and is varied to 2 for low and 4 for mid level enrichment, respectively.

Based on the average density, the lower bound for the density is estimated as

$$lb = (r - 1) \cdot avgDens \quad (3.31)$$

The density for each feature is then drawn from the single parameter Pareto distribution with shape  $r$  and lower bound  $lb$ .

The sequencing depth is set analogously to the background data. The base quality of the sequencing reads are generated in the modes "high", "mid", and "low". Poisson distributions bound to the interval  $[0; 40]$  were used to generate the probability for each possible Phred score. In detail, the probability of Phred score  $\phi$  is

$$P(\phi) = P_{Poisson}(40 - \phi, \lambda) \quad (3.32)$$

where  $\lambda$  is specific for the mode. In detail,  $\lambda = 7$  was selected for mode "high",  $\lambda = 17$  for mode "mid", and  $\lambda = 29$  for mode "low".

The probabilities are normalized such that they sum up to 1 in the interval  $[0; 40]$ .

### 3.5.6 Benchmarking Replicates

As described in the previous section, benchmarking data sets of different quality were generated. This section gives an overview of the benchmarking data sets generated and how they were processed. In addition to the generated genome and read files, the reference signal, i.e., the gold standard for the peak's location, is provided as annotation file in BED [47] format.

#### Replicates Generated

The replicates generated can be subdivided according to their type into 'noise free', 'wrong signal', and 'noisy signal'.

**Noise free** 6 replicates with optimal parameters were generated for experiment and background. These replicates can be used to evaluate the performance in the ideal case as well as for finding optimal parameters for the peak-caller.

**Wrong signal** Three replicates with different features each were generated. They can be used to test the performance of the peak-caller when ChIP-seq data may be erroneous. This might happen, for example, when the antibodies did not work correctly.

**Noisy Signal** Peak-calling can be affected by poor data quality. In particular, low sequencing quality, over-sequencing, under-sequencing, and low signal enrichment may affect the ability to find peaks and the quality of the predicted peaks. Two replicates were designed for each combination of low, middle, and high sequencing quality, sequencing depth, and enrichment (see Table 3.8).

In total, 27 replicates of different quality were generated yielding 54 read files; 27 for the experiment and 27 for the associated background. While the read files are generated in pairs, i.e., always one file for the experiment and one

Table 3.8: Parameter settings for the different levels of quality. See Section 3.5.5 for a description of the parameters.

	low	middle	high
sequencing quality	29	17	7
sequencing depth	0.01	0.1	0.4
enrichment	2	4	5

file for the background with the same characteristic, read files with different characteristics can be combined to replicates as well to study the effect of unequal quality of background and experiment.

### Post-Processing

Since no adapters were simulated, the read files of the replicates generated are directly mapped onto the artificial genome generated during simulation. The genome mapper `segemehl` [44] was used to index the genome and map the reads onto the genome. Mapping was performed with 80% accuracy (i.e., 80% of the alignment have to be identical between read and genome) and only the best hit is reported. The resulting SAM [53] files are converted into BAM [53] files and sorted using `SAMtools` [53]. Afterwards, `SAMtools` is used to remove PCR replicates.

### 3.5.7 Benchmarking Data Sets

The generated replicates can be combined to different data sets for benchmarking. As a minimal test for peak-calling, the following combinations are recommended:

**Noise free:** 6 noise free replicates—background and experiment data. Using this data shows if the method produces useful results. Methods that perform poorly on this data should not be used for real data.

**Noise:** 3 data sets with 6 replicates each combining 1, 2, and 3 replicates with noisy signal but good quality otherwise with 5, 4, and 3 noise free replicates, respectively. This will indicate the impact of experiments resulting in a wrong signal (e.g., failure of the antibody).

**Low quality:** 2 data sets with 3 and 4 replicates. In each data set, 2 noise free replicates are used together with either 1 or 2 replicates with low sequencing quality and good quality otherwise.

**Over-sequenced:** 2 data sets with 3 and 4 replicates. In each data set, 2 noise free replicates are used together with either 1 or 2 replicates with a (too) high sequencing depth and good quality otherwise.

**Under-sequenced:** 2 data sets with 3 and 4 replicates. In each data set, 2 noise free replicates are used together with either 1 or 2 replicates with a (too) low sequencing depth and good quality otherwise.

**Low enrichment:** 2 data sets with 3 and 4 replicates. In each data set, 2 noise free replicates are used together with either 1 or 2 replicates with low enrichment and good quality otherwise.

**Bad:** 2 data sets with 3 and 4 replicates. In each data set, 2 noise free replicates are used together with either 1 or 2 replicates with low sequencing quality, (too) low sequencing depth, and low enrichment. These data sets will show the combined effect of the different sources of noise.

### 3.6 Statistical Measures for Quality Assessment

The benchmarking data sets allow evaluating statistical parameters to assess the quality of the peak-calls produced with the peak-caller chosen. The gold standard provides a list of peaks specified by the genomic location. Based on the gold standard, the following statistical measures for quality assessment are recommended: the number of peaks  $n_p$ , the *recall*, the positive predictive value (*PPV*), and the false discovery rate (*FDR*).

Let  $PC$  be the set of peak-callers and  $pc \in PC$  a peak-caller. Let further  $p_{pc}$  be a peak detected by peak-caller  $pc$ ,  $p^g$  a peak in the gold standard, and  $p_{pc}^g$  a peak in the gold standard found by the peak-caller  $pc$ . Then,  $n_p$ , recall, PPV, and FDR are defined as:

$$n_p(pc) = |\{p_{pc}\}| \quad (3.33)$$

$$recall(pc) = \frac{|\{p_{pc}^g\}|}{|\{p^g\}|} \quad (3.34)$$

$$PPV(pc) = \frac{|\{p_{pc}^g\}|}{n_p(pc)} \quad (3.35)$$

$$FDR(pc) = \frac{|\{p_{pc}\}| - |\{p_{pc}^g\}|}{|\{p_{pc}\}|} \quad (3.36)$$

While  $p_{pc}$  and  $p^g$  are easy to determine,  $p_{pc}^g$  needs a careful definition. This definition has to consider the properties of the gold standard:

1. The length of the peaks in the gold standard are multiples of 146bp by construction since all the features generated with the Markov chain have a length of 146bp, i.e., individual histones were simulated rather than peaks. A peak in the gold standard will always start and end with a feature but may include more than one feature. Thus, peaks have a minimal length of 146bp but can extend much further in length.

2. Reads of length 36bp are generated assuming a fragment length of 200bp on average for sequencing. Thus, enrichment will have the same resolution. Therefore, peak-calling is restricted to the resolution given by the fragment length for sequencing and will not reach the 146bp resolution of the features.

As a consequence, gold standard peaks and predicted peaks will not match up completely. Thus, taking a 100% identical peak match as criterion for finding a peak of the gold standard would be too hard and will distort the performance of the peak-caller evaluated. Therefore, it is recommended to soften the criterion for re-discovering a peak of the gold standard. The criterion is motivated by the following:

- The minimum peak length is 146bp.
- The resolution expected for one nucleosome is 200bp on average but lies in the interval [150; 250].
- Given the 146bp resolution, even a perfect prediction would cover only up to 73% ( $= 146/200$ ) on average. However, the coverage might be as low as 58.4% when the sequencing fragment length is 250bp.
- Not only perfect matches should be counted.
- Incompletely found peaks should also be counted as “found” when most of the peak is found.

Therefore, the recommended soft criterion is a reciprocal overlap of at least 50% between prediction and gold standard:

$$p_{pc}^g := \{p_{pc} | \exists p^g : \text{reciprocal overlap}(p^g, p_{pc}) \geq 0.5\} \quad (3.37)$$

With this criterion, it is now possible to calculate the assessment parameters for any peak-caller based on the gold standard benchmarking data sets.

## 3.7 Evaluation

### 3.7.1 Introduction

This section presents multiple evaluations of Sierra Platinum. In a first step, Sierra Platinum will be evaluated on the different benchmark data sets described in Section 3.5. This evaluation is also a guide on how to recognize different types of noisy data with the quality measure visualizations provided by Sierra Platinum. Afterwards, the influence of the different parameter settings of Sierra Platinum is evaluated on the quality of its peak-calling results. In a last step, Sierra Platinum is compared against other state-of-the-art peak-calling tools.

### 3.7.2 Sierra Platinum Quality Measures and Visualizations

In Section 3.7.4 it is shown that down-weighting or deleting noisy replicates is beneficial for the performance of Sierra Platinum. While it is known which replicates have good quality and which replicates have bad quality in the case of the benchmarking data sets, for real data this information is commonly not available. However, the quality of the replicates is assessed using the quality measures and the visualizations included in Sierra Platinum. Several visualizations showing different quality measures of the replicates allow the user to judge whether a replicate can be used for peak-calling, should be excluded from peak-calling, or should be down-weighted during peak combination. This section shows how the different types of noise can be recognized using the visualizations provided by Sierra Platinum.

#### Noise-free data

Figure 3.23a shows a noise free replicate. The theoretical distributions (at least those calculated for the normalized ones) are (almost) identical (top and bottom, left). Each tag distribution has a single peak and the peak of the experiment is at lower tag counts than the peak of the background distribution (top, middle). However, the respective means of the two distributions are almost identical (location of the peak of the theoretical distribution). The “boxes” of the boxplots are in the green area (top, right). The replicate specific distributions for  $p$ -values (bottom, middle) and significant windows (bottom, right) do not differ much from the corresponding median distributions. The correlation with other replicates is high and thus, red is the predominant color



in the heatmap (Figure 3.23b). All significant windows of the replicates overlap largely and to approximately the same amount with the final significant windows (Figure 3.23c, right).

### **Poor Sequencing Quality**

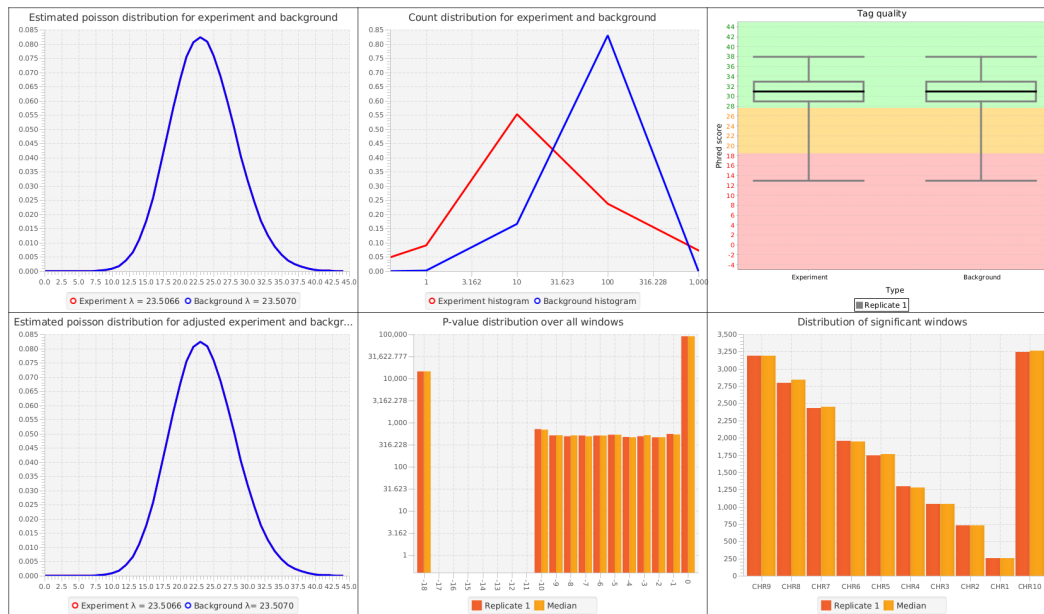
Replicates with low sequencing quality are easily recognizable since the “box” of the boxplot is not in the green area and may even drop into the red area of the boxplot (see Figure 3.24a, top, right). However, also the distribution of the significant windows and the amount of overlap with the final significant windows indicate the low sequencing quality. One can observe that the number of significant windows is lower than on average (see Figure 3.24a, bottom, right) and the amount of overlap of replicate 3 with the final significant windows is noticable lower (see Figure 3.24b, right).

### **Low enrichment**

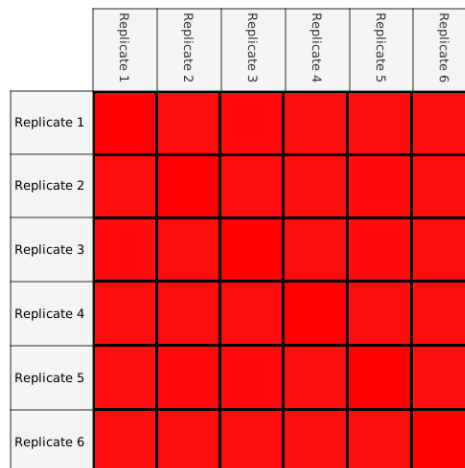
For replicates with a too low signal-to-noise ratio, e.g., due to ineffective antibodies, the visualizations look like those in Figure 3.25. Low enrichment usually leads to an unclear peak in the experiment tag count distribution, i.e., there is no peak but a plateau (Figure 3.25a, top, middle). Furthermore, the significant windows are less frequent in the replicate than in the median distribution of significant windows (Figure 3.25a, bottom, right). All other quality measurements and visualizations in the single replicate view are similar to those of the good replicates. The result charts show a lower amount of overlap for low enriched replicates with the final significant windows (Figure 3.25b, right).

### **Low Sequencing Depth**

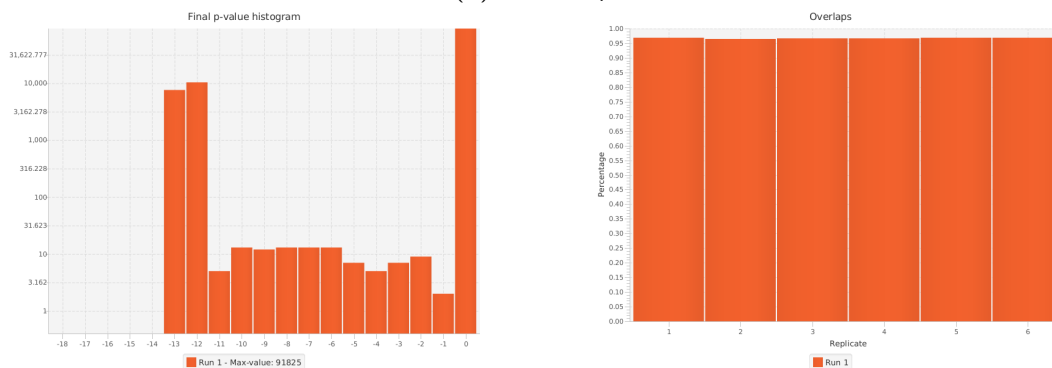
Under-sequencing of the experiment leads to peaks close to zero in the tag count distribution (Figure 3.26a, top, middle) since not enough reads are sampled from the data to cover the genome. Under these circumstances, it is hard to reliably estimate the parameter of the background model which affects the peak-calls. The number of significant windows in the replicate is much lower than that in the median replicate (Figure 3.26a, bottom, right) and the  $p$ -value distribution of the replicate differs strongly from that of the median (Figure 3.26a, bottom, middle). In particular, there are more windows with  $p$ -values close to one but



(a) Replicate view.

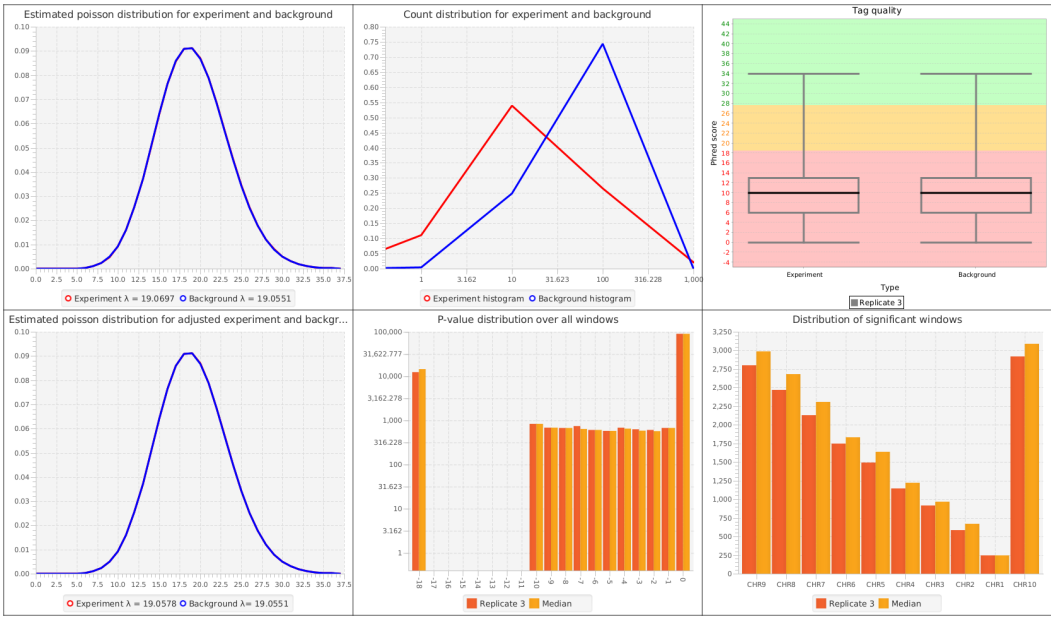


(b) Heatmap.

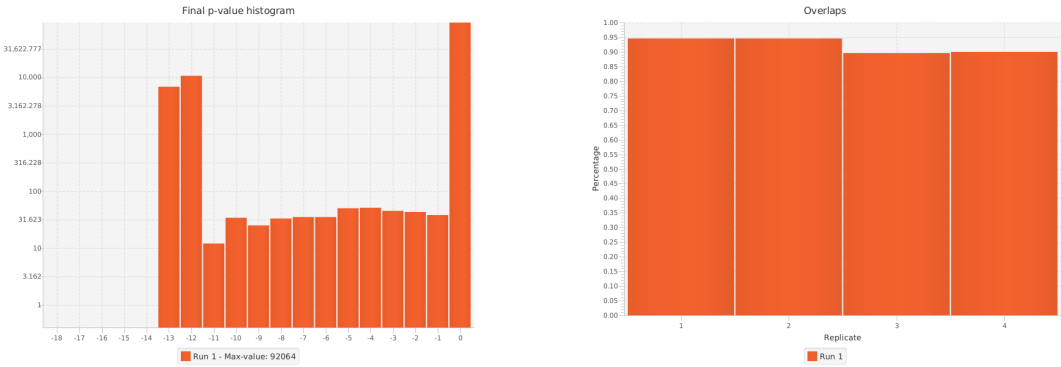


(c) Result charts.

Figure 3.23: Quality measurements for a noise free replicate.

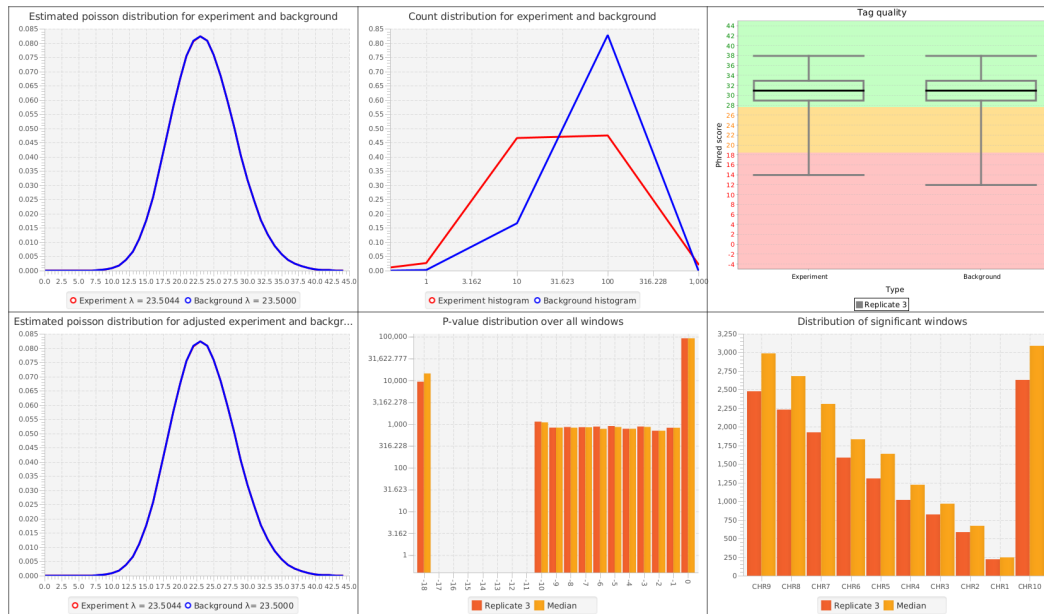


(a) Replicate view.

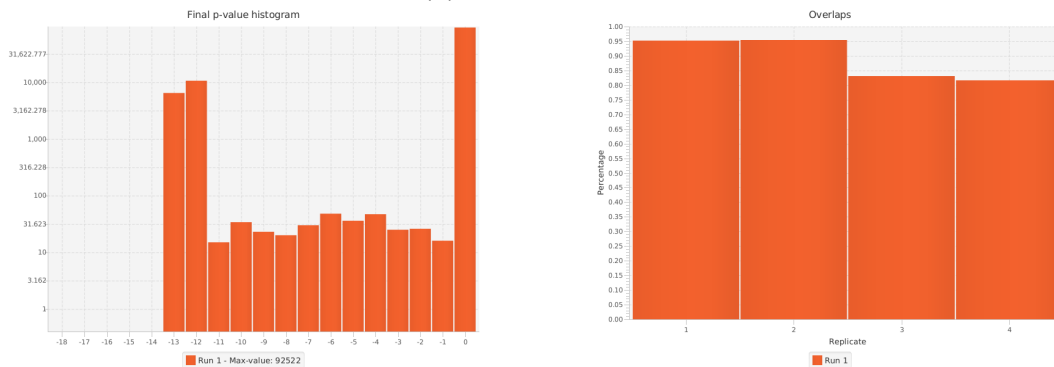


(b) Result charts.

Figure 3.24: Quality measurements for a replicate with low sequencing quality



(a) Replicate view.



(b) Result charts.

Figure 3.25: Quality measurements for a replicate with low enrichment

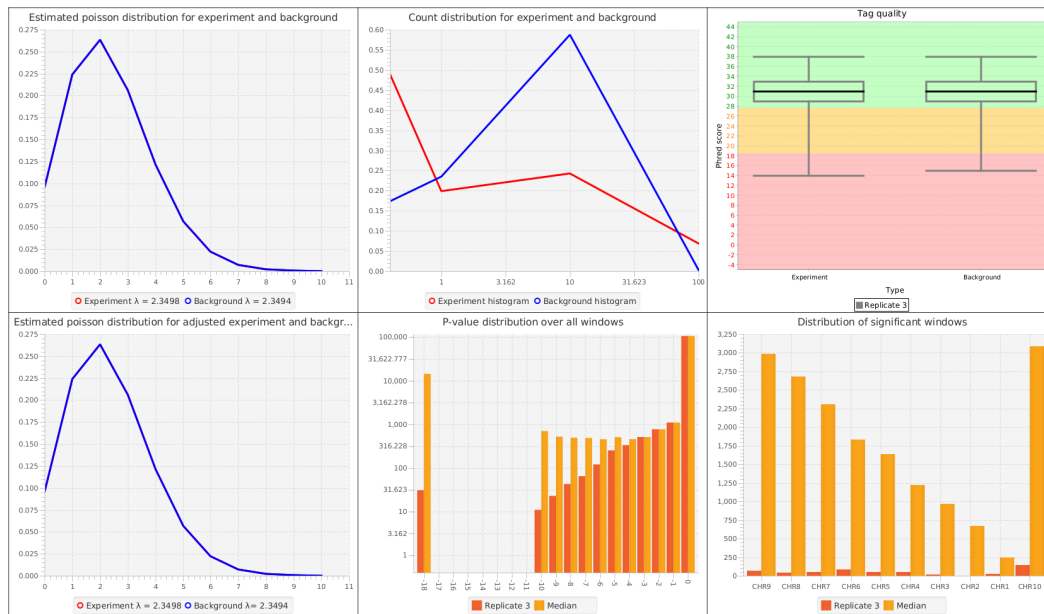
less with  $p$ -values close to zero in the replicate compared to the median of all replicates. The correlation with the other replicates is lower (Figure 3.26b, less saturated squares) and the overlap with the final significant windows is very low (Figure 3.26c, right). Furthermore, the  $p$ -value distribution of the final  $p$ -values has a bathtub shape with a large amount of very low and very high  $p$ -values, while there are only few windows with  $p$ -values between the lowest two and the highest  $p$ -value (Figure 3.26c, left).

### High Sequencing Depth

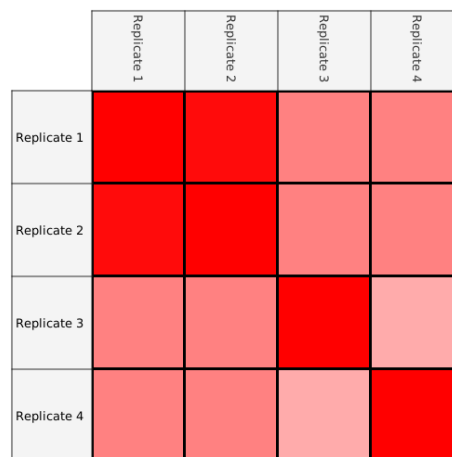
When replicates are over-sequenced, the tag count distributions have their peak at almost the same position and low tag counts are particularly rare (Figure 3.27a, top, middle). For most  $p$ -values in the  $p$ -value distribution the window count is lower than in the median replicate (Figure 3.27a, bottom, middle). Furthermore, high sequencing depth might lead to artifacts. Figure 3.27b, right, shows, for example, that the significant windows of replicate 3 and 4 overlap to 100% with the final significant windows. However, from the distribution of the good replicates in Figure 3.23 one can learn that even with perfect data the overlap is not 100% but slightly below 100% since the input signals differ. Likewise under-sequencing, also over-sequencing results in a strongly bathtub-shaped distribution of the final  $p$ -values (Figure 3.27b, left).

### Noisy Data Sets

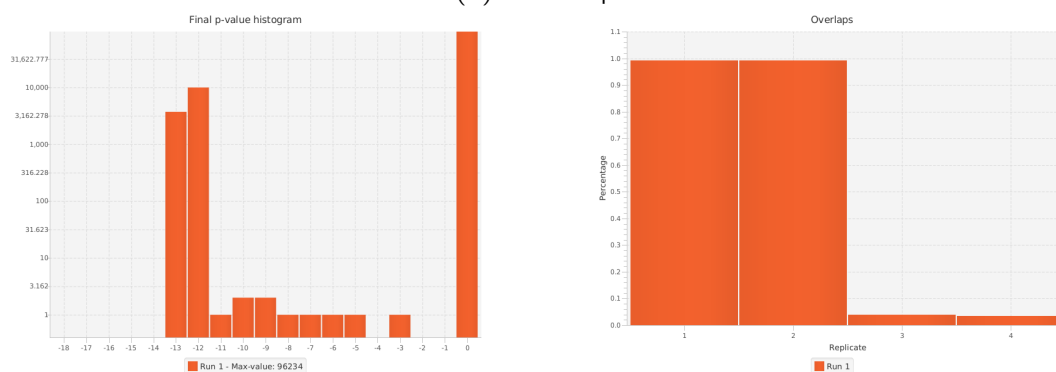
When something went completely wrong during the experimental procedure or when the antibody did not work correctly, this would result in a wrong signal. Then, a replicate does not reflect the same signal as the other replicates, This can be recognized using the heatmap. No or very low correlation with all other replicates indicates a wrong signal (Figure 3.28a, white or red with low saturation). Furthermore, the overlap with the final significant windows is also low (Figure 3.28b, right) and the final  $p$ -value distribution is that extremely bathtub-shaped that only three different  $p$ -values are observed, namely the lowest two and the highest  $p$ -value (Figure 3.28b, left).



(a) Replicate view.

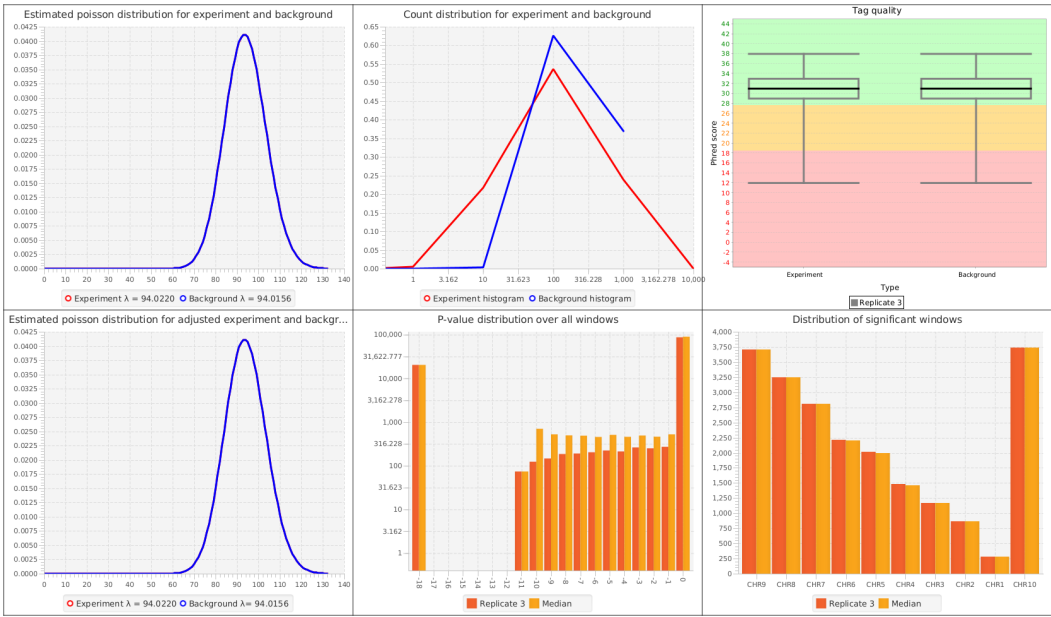


(b) Heatmap.

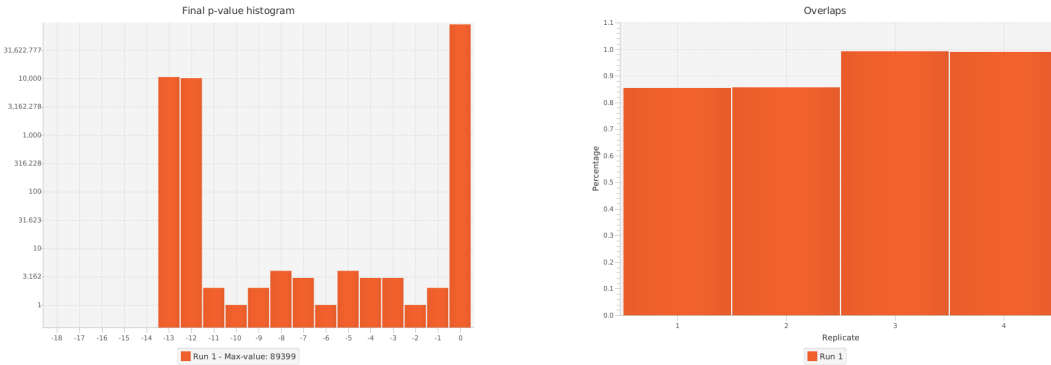


(c) Result charts.

Figure 3.26: Quality measurements for an under-sequenced replicate

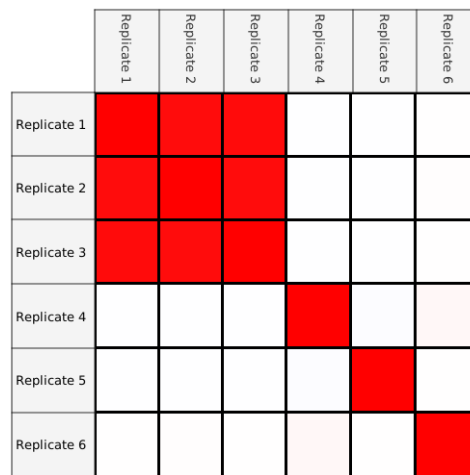


(a) Replicate view.

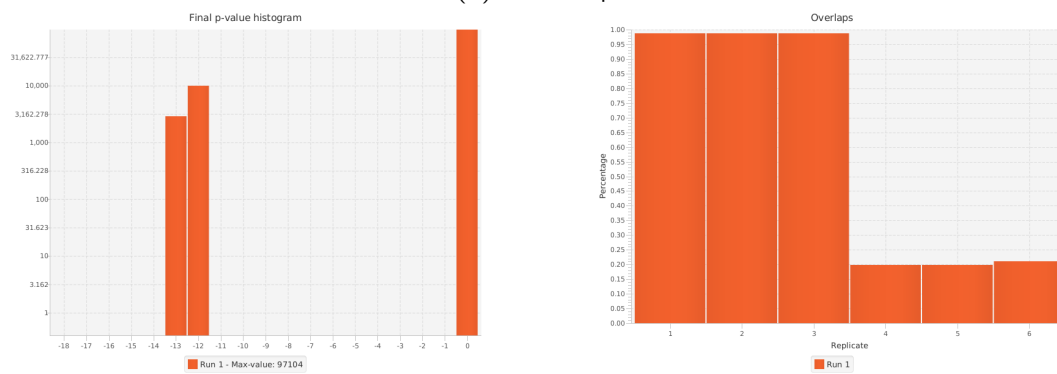


(b) Result charts.

Figure 3.27: Quality measurements for an over-sequenced replicate



(a) Heatmap.



(b) Result charts.

Figure 3.28: Quality measurements for a replicate having the wrong signal



## Summary

Sierra Platinum provides a wide range of visual quality controls. They allow judging, how well a replicate is suited for peak-calling in general and how well it fits to the other replicates. Hereby, each quality control provides insights into specific aspects of the data such as tag distributions or overall significance. This allows not only to identify replicates with poor quality but also the identification of the type of noise. Thus, the user of Sierra Platinum is able to react adequately by deleting or down-weighting certain replicates or by taking lower quality into account for further analysis of the data.

### 3.7.3 Parameter Selection

For obtaining optimal results, peak-callers frequently allow to change parameters like window size, window offset, or cut-off value that have a direct impact on the outcome of the peak-calling process. Moreover, most peak-callers provide either default settings or estimates for the parameters provided. Sierra Platinum provides default settings as a good starting point for initializing the parameters of the peak-calling process. Estimates for the parameters based on the data are helpful for good quality data where such estimates are reliable. Furthermore, an evaluation of different parameter settings enables the user to judge by himself/herself how to choose the parameters. Therefore, this section shows the effects of different parameters for the  $p$ -value cutoff, the window size, the window offset, and the method for calculating the  $q$ -values on the peak-calling process. The three data sets evaluated are the noise-free data set, the data set with 3 noisy replicates, and a bad quality data set.

#### How to choose the $p$ -value cutoff

The effect of the  $p$ -value cutoff is almost negligible (see Figure 3.29). Using a cut-off value of 1 produces only significant windows. Thus, there is only one peak on each chromosome covering the whole chromosome. Those peaks do not meet the criteria of the reciprocal coverage and thus, result in a recall and a PPV of 0 and a FDR of 1. There is also only a very small variation in the recall down to a cut-off value of  $10^{-4}$ . Therefore, a cutoff of  $10^{-5}$  is recommended to be sure to obtain best recall, PPV, and FDR.

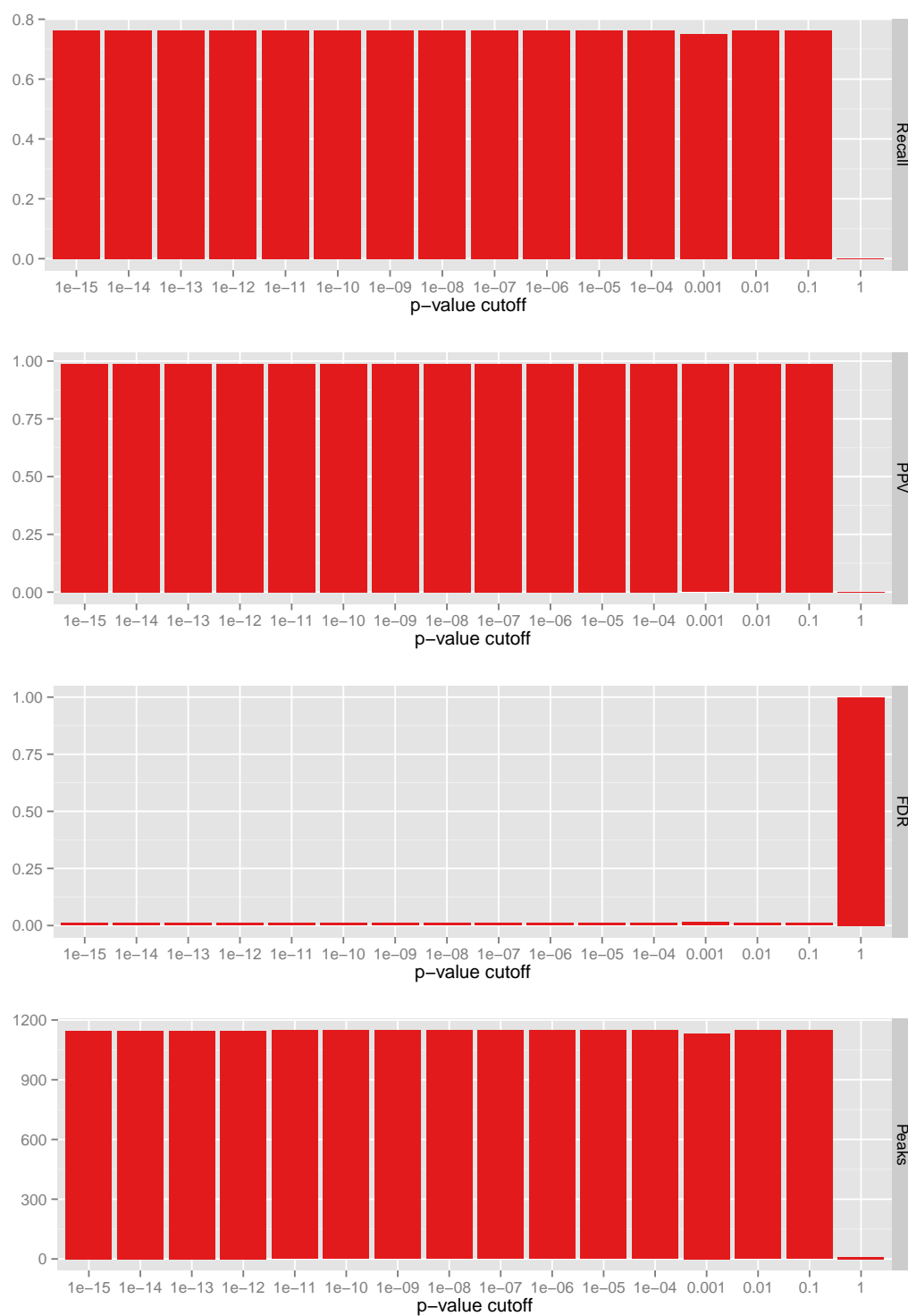
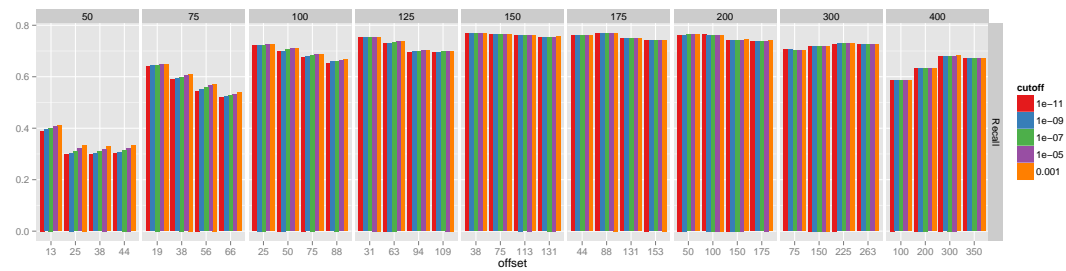


Figure 3.29: Evaluation of the  $p$ -value cutoff on the result quality for the noise-free data set.

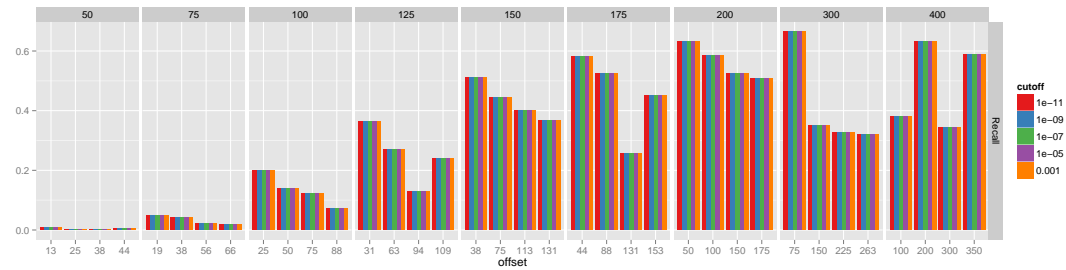
### **How to choose the window size**

Furthermore, the influence of the window size was tested on recall, PPV, and FDR. Note, that the fragment size of the simulated data is on average 200nt. To assure that the  $p$ -value cutoff is independent of the other parameters, not only different window sizes were tested but also the  $p$ -value cutoff was varied. With respect to the recall (see Figures 3.30a), one can observe that window sizes between 100 to 200 nt have the best recalls for good quality data (with 150nt being the very best). For the noise-3 and the bad-2 data set (see Figure 3.30b, 3.30c) larger window sizes yield better recalls. In these cases, the best recall can be achieved with a window size of 300nt. A similar behavior can be observed for the positive predictive value (see Figure 3.31) and the false discovery rate (see Figure 3.32).

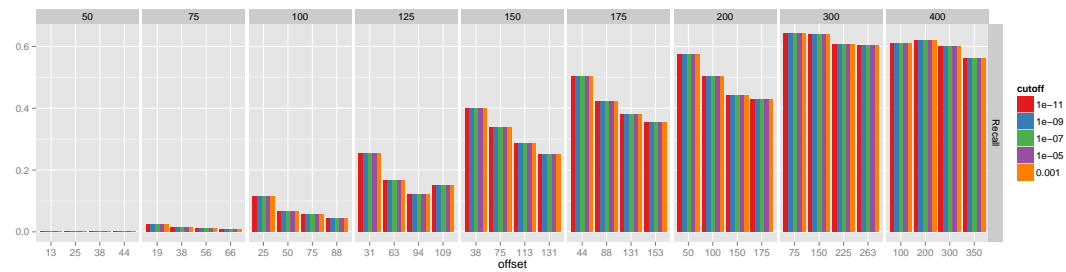
The reasons for the observed FDRs, PPVs, and recalls become clear when analyzing the number of peaks (see Figure 3.33). Small window sizes lead to an overprediction of the number of peaks in the case of the noise-free data set. It seems that some of the small windows are insignificant due to local sequence composition biases and thus lead to a gap in the peak. Such insignificant windows are less likely with large window sizes. In the case of bad quality data, the number of predicted peaks is too low for small window sizes. A reason therefore might be a low signal due to low enrichment of chromatin fragments during the experiment in combination with local sequence composition biases. Given the results, a window size equal to the average fragment length is recommended (200nt in this benchmark data set). If it is obvious that the data has bad quality, a larger window size should be chosen.



(a) Noise-free data set

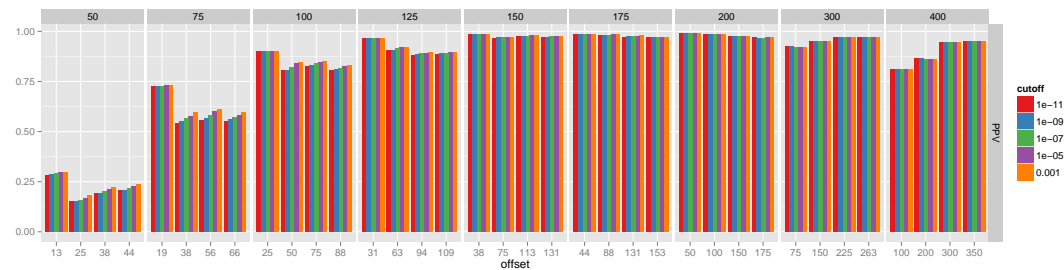


(b) Noise-3 data set

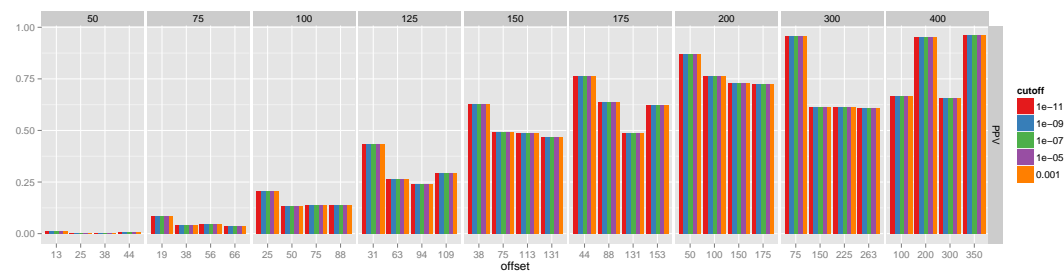


(c) Bad-2 data set

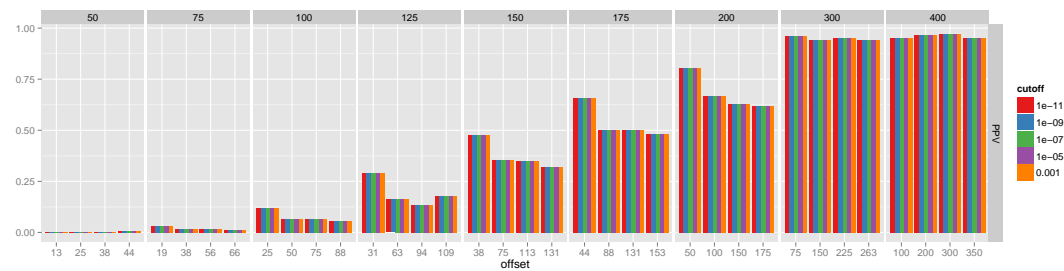
Figure 3.30: Recall of the peak-calls using different combinations of window size, window offset, and cutoff.



(a) Noise-free data set

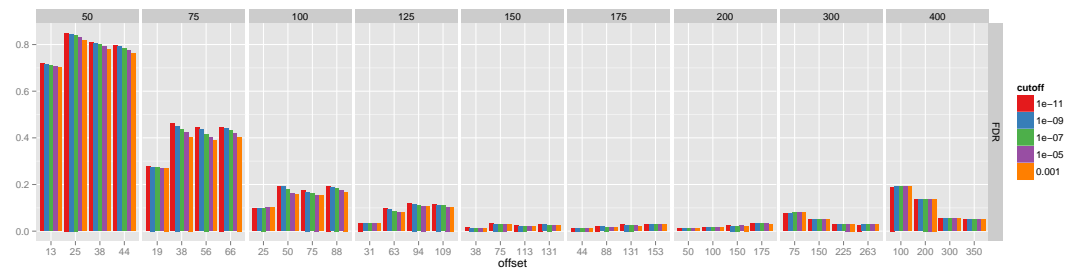


(b) Noise-3 data set

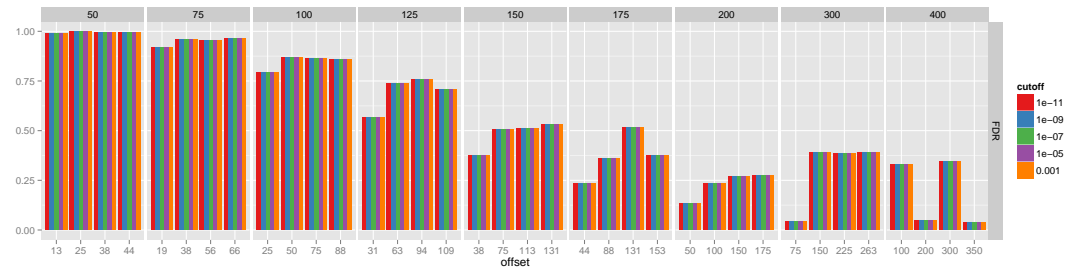


(c) Bad-2 data set

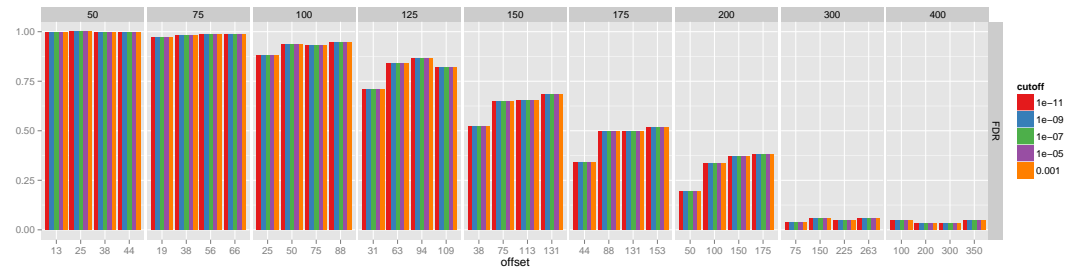
Figure 3.31: Positive predictive value of the peak-calls using different combinations of window size, window offset, and cutoff.



(a) Noise-free data set



(b) Noise-3 data set



(c) Bad-2 data set

Figure 3.32: False discovery rate of the peak-calls using different combinations of window size, window offset, and cutoff.

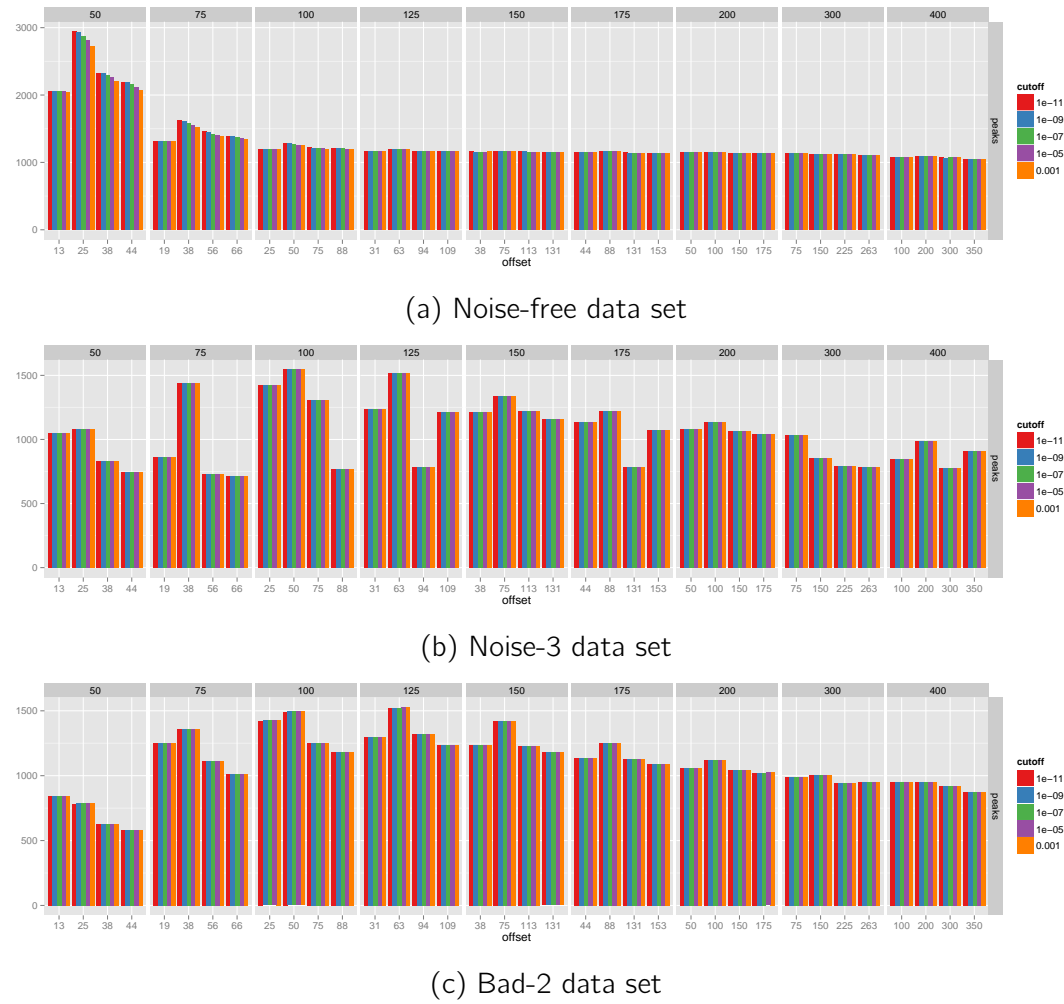


Figure 3.33: Number of peaks found using different combinations of window size, window offset, and cutoff.

### How to choose the window offset

To capture the effect of different offsets in combination with the window size and the  $p$ -value cutoff, 4 different window offset settings were evaluated for each window size with Sierra Platinum. Each combination of window size and offset was tested with 5 different  $p$ -value cutoffs. As window offset, one quarter, one half, three-quarters, and seven-eighth of the window size were selected. The results are shown in Figures 3.30–3.33.

For small window sizes (50–125 nt), one can see that small offsets (one quarter of the window size) give the best results (highest recall and PPV, and lowest FDR). The window offset for the proposed window size of 200nt does not strongly affect PPV, FDR, or recall on the noise-free data set. Nevertheless, small offsets (one quarter or one half) are slightly better than large offsets. For bad quality data, one quarter window offsets always produced the best results with respect to recall, PPV, FDR, and number of predicted peaks. When using a window size of 400nt, the recommendation for the window offset changes to the half of the window size (200nt), which reflects the fragment size of the input data. Consequently, the recommendation for the window offset is one quarter for window sizes  $\leq 200$ nt and half of the window size for window sizes  $> 200$ nt.

### Which method for the $q$ -value calculation should be used

Sierra Platinum provides different methods to compute the  $q$ -value. The most traditional but also most strict method is the Holm-Bonferroni method. Both methods from Storey are less strict and thus are expected to be more suitable for noisy data with bad quality where very strict methods may lead to non-significant windows only. Storey proposes both a simple and a bootstrap version. The latter one is more robust. All three methods were tested with window size 200nt, offset 50nt, and two different cutoff values ( $1e-5$  and  $1e-10$ ). The results are shown in Figure 3.34.

On noisy data without other quality issues, the Holm-Bonferroni correction performs best regardless of the cut-off value. Storey's less strict methods lead both to high FDRs as well as low recall and PPV in those cases. On data sets with replicates of bad quality, Storey's methods for the calculation of the  $q$ -value perform best. Using noise-free data, the recall is almost equal for the different methods but PPV is higher and FDR is lower for Holm-Bonferroni correction.



## Summary

Sierra Platinum has four parameters to fine tune the performance of the peak-calling results. The nature of these parameters implies that a bad parameter choice can strongly affect the quality of the results. This section showed, how to adapt the parameters of Sierra Platinum to the given data sets.

The window size should be about the size of the fragments in the experiment. If the fragment size varies across the replicates or is completely unknown, shorter windows are recommended rather than longer ones except if the data quality is overall very low when larger window sizes perform best. The offset should be one quarter of the window size  $\leq 200\text{nt}$ . When using an even larger window size, half of the window size is recommended. The  $p$ -value cut-off does not affect the performance strongly. However, a default  $p$ -value of  $10^{-5}$  is recommended since for higher  $p$ -value cut-offs variation in their performance could still be observed. The last parameter is the method for the  $q$ -value calculation. The performance differences between the different methods depend on the type of noise in the data and on the chosen  $p$ -value cut-off for significance. Holm-Bonferroni corrections perform well and fast in most cases but can not handle combinations of noise and quality issues very well. However, for those cases Storey's  $q$ -value methods perform well and robust.

### 3.7.4 Approach Comparision

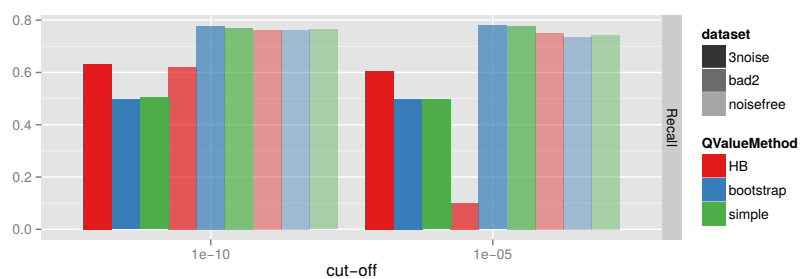
To compare the results generated by Sierra Platinum with existing other approaches, several tools were used to generate peak-calls based on the benchmark data set. The following methods were used:

**PePr** The multiple-replicate peak-caller PePr v1.0.1 [89].

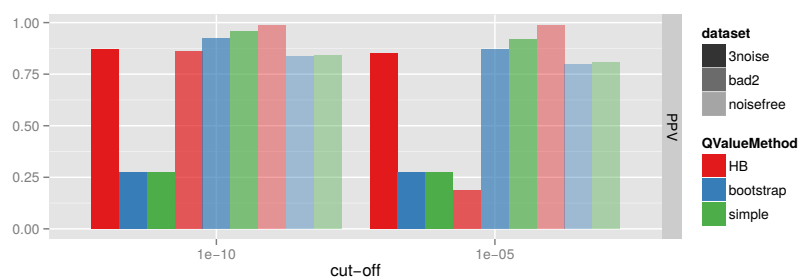
**MACS-SA** For each of the replicates, MACS2 v2.1.0 [90] was started in the 'callpeak' mode and called broad peaks. These peaks were then merged to obtain the final resulting peak list.

**MACS-CR** SAMtools [53] were used to merge the bam files of the experiments of the replicates into a single experiment bam file and the bam files of the background of the replicates into a single background bam file. MACS2 v2.1.0 [90] was started as for the single replicates.

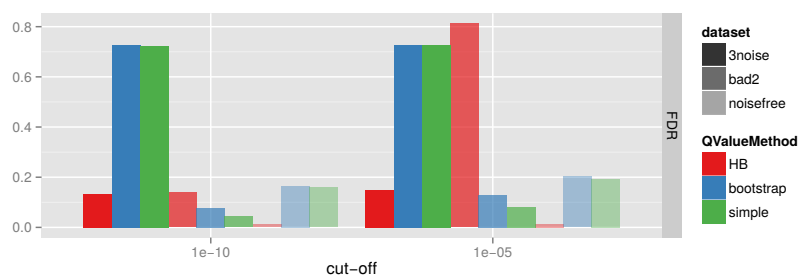
**BinQuasi** The multi-replicate peak-caller BinQuasi v0.1-3 [37].



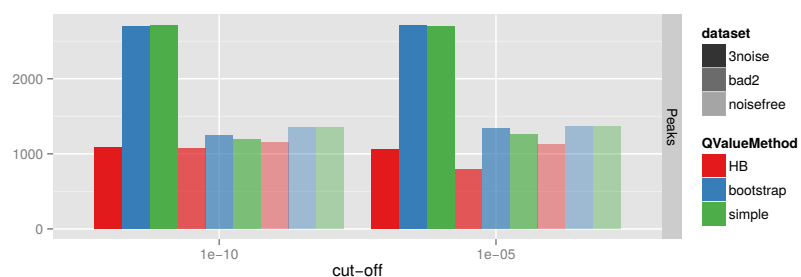
(a) Recall



(b) PPV



(c) FDR



(d) Number of peaks found

Figure 3.34: Evaluation of the  $q$ -value methods on the noisy, the bad, and the noise-free data set (Section 3.5.7).

If not stated explicitly, the parameter settings were the same for all peak-calling methods: window size =  $200nt$  (equal to the fragment size), window offset =  $50nt$ ,  $p$ -value cut-off =  $10^{-5}$  and peak type = 'broad'.

Except for the 'noise free' set of replicates in the benchmark, Sierra Platinum was used to generate peak-calls using all replicates with equal weight (Sierra1, without down-weighting or excluding bad replicates), excluding the 'noisy' replicates (Sierra2), and down-weighting the 'noisy' replicates (Sierra3). In all data sets (Section 3.5.7), the 'noisy' replicates could be identified using at least one quality measurement (Section 3.7).

The results are shown in Figures 3.35–3.37. From top to bottom Recall, Positive Predictive Value (PPV), False Discovery Rate (FDR), and Number of Peaks are reported. In addition to the results obtained by the different peak-calling methods, the number of peaks is provided that would be optimally found (gold standard, GS).

### Noise-free data

For the noise-free data (Figure 3.35), Sierra Platinum has the highest recall, followed by BinQuasi with a slightly smaller recall. Both variations of MACS show more than 10% less recall, while PePr has a recall of  $\approx 30\%$  less than Sierra Platinum. Although the recall of BinQuasi is comparable with Sierra Platinum, the PPV and the FDR of BinQuasi are much worse than Sierra Platinum. Both approaches of MACS show moderate results while PePr provides the worst results. The high number of peaks generated by BinQuasi and PePr comes at the expense of a high FDR, while both methods are relying on the same model. In total, Sierra Platinum gives significantly better results, since BinQuasi produces a similarly good recall, but also generates many false positives. With data sets created in laboratories, no gold standard is given. Then, the analyst can only rely on the peak-calling results and a low FDR rate, since it is not possible to distinguish between false and true positive results in these data sets.

### Poor Sequencing Quality

For poor sequencing quality, two benchmarking data sets were used. Each of them contains two good replicates and either one or two bad replicates (Figure 3.36, first column). It does not make a large difference whether one or two data sets with a low sequencing quality, and thus more sequencing errors, were added to two high quality data sets: the results are quite similar to those

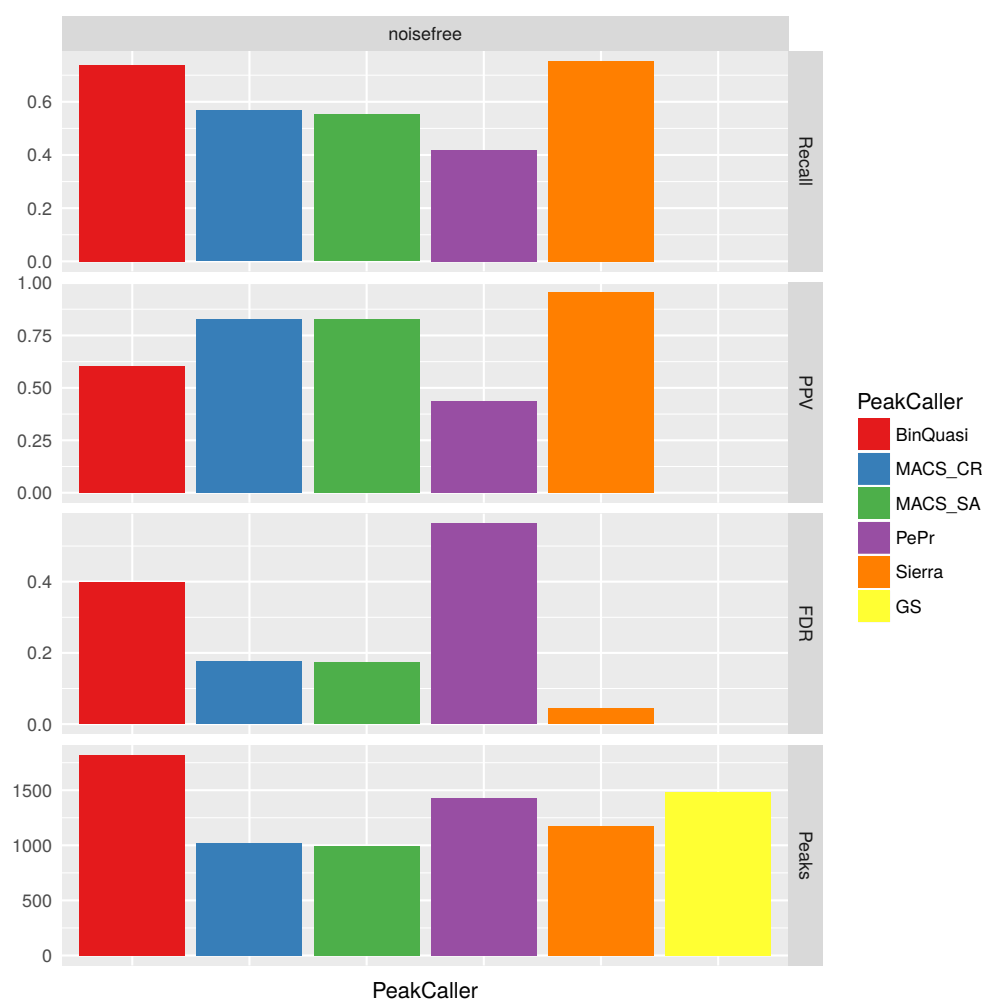


Figure 3.35: Evaluation results for the noise-free data set (6 replicates).

of the noise-free data set. Excluding the low quality replicates improves the results a bit with respect to recall, PPV, and FDR. Like with the noise-free data set, BinQuasi achieves a very good recall rate, but provides a considerably lower PPV and significantly higher FDR compared to Sierra Platinum. Again, BinQuasi provides a lot of false positive results. Compared to MACS and PePr, any approach of Sierra Platinum (all replicates, removing bad replicates, down-weighting bad replicates) performs better with respect to all three quality measurements.

### **Low enrichment**

A low enrichment, i.e., the signal to noise ratio is low, does not much affect the performance of all peak-callers (Figure 3.36, second column). The result of Sierra Platinum can be equally well improved by deleting or down-weighting the low enriched replicates.

### **Low Sequencing Depth**

A low sequencing depth does not have a strong influence on the peak-calling quality of Sierra Platinum (Figure 3.36, third column). Deleting or down-weighting the replicates with low sequencing quality improves the results even more. Deleting is just marginally better than down-weighting. Still, MACS-CR and MACS-SA have lower recall but a higher PPV than PePr. The recall is about 3% lower than in the noise-free data for MACS. BinQuasi provides a better recall rate than Sierra Platinum, but comes with a worse PPV and FDR comparable to PePr.

### **High Sequencing Depth**

Replicates with a too high sequencing depth are not affecting the peak-calls of MACS-CR and MACS-SA (Figure 3.36, fourth column). This might be an effect of the two good quality replicates always included in the data sets. Surprisingly, two replicates with a high sequencing depth produce better results than just one replicate with too many reads in the case of PePr (recall and PPV increase by about 10%). In the case of BinQuasi, the recall is comparable to Sierra Platinum, but again PPV and FDR rate is much worse. It is also noticeable that BinQuasi finds significantly more peaks in these data sets than in the previous benchmarks. The results of Sierra Platinum in its default settings are affected by



Figure 3.36: Evaluation results for data sets with noise. First column: low sequencing quality. Second column: low enrichment. Third column: too low sequencing depth. Fourth column: too high sequencing depth. First four rows: one bad replicate; three replicates in total. Second four rows: two bad replicates; four replicates in total.

the replicates with the too high sequencing depth. Deleting the replicates with too many reads, the recall drops slightly but the PPV increases. Down-weighting these replicate shows similar results. In the case of two over-sequenced samples, the results of the effect of down-weighting or deleting bad replicates can be seen even stronger.

### Bad replicates

In the next step, Sierra Platinum, MACS-SA, MACS-CR, and PePr were also evaluated on data sets with a mixture of noises used in the data sets proposed before. The data sets *bad1* and *bad2* (Figure 3.37) are composed of two good replicates, and 1 respectively 2 under-sequenced replicates with low enrichment and low quality. With both data sets the recall drops and the FDR increases with Sierra Platinum. Interestingly, the down weighting mechanism decreases the recall rate using the *bad1* data set. This behaviour only occurs with this data set. BinQuasi generates again a comparable recall rate, but also shows a higher FDR rate. In comparison to the other peak-callers, even using Sierra Platinum with equal weights generates a higher recall and lower FDR.

The data set *likeK4* contains a mixture of qualities (Figure 3.37), i.e., experiment and background may not have comparable data quality and the quality between replicates differs as well. Similarly to the previous data sets, Sierra Platinum outperforms the other peak-calling methods and BinQuasi shows an even higher FDR rate. Since the data set *likeK4* models a realistic data set from laboratories, this indicates a problem of BinQuasi with noisy replicates. Similarly to the previous data set, recall and FDR are better compared to the other peak-callers independently of the approach used for Sierra Platinum.

### Noisy data sets

All peak-callers were evaluated on data sets containing 1, 2, or 3 noisy replicates (Figure 3.37, bottom row), i.e., replicates with a different signal track. Each data set is filled up with replicates of perfect quality until they contain 6 replicates in total. Using the *noise1* data set, BinQuasi calculates only a very small amount of peaks. Depending on this, the recall rate drops considerably under 25%. Using even more noisy replicates, BinQuasi fails to build a model for the peak-calling process and can not find any peak any more.

The recall of the other peak-callers decreases with an increasing amount of noise. In particular, MACS-SA and PePr show a large drop in the recall. Furthermore, the FDR increases considerably. The largest increase of the FDR is found for MACS-SA since the peaks of all replicates are simply merged. Thus, all peaks from the noisy replicates are kept. Again, any approach of Sierra Platinum outperforms the other peak-callers with respect to all 3 quality metrics.

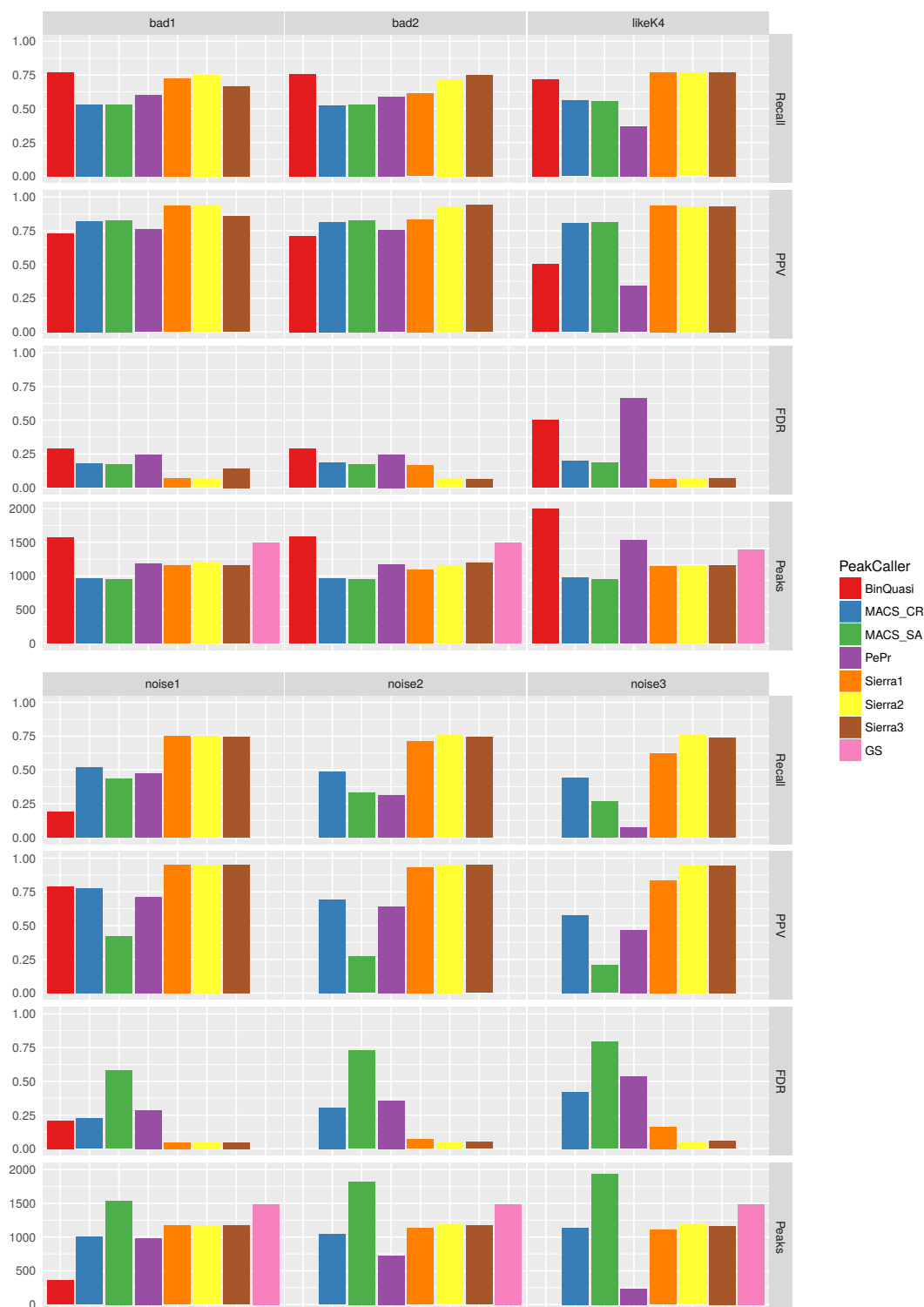


Figure 3.37: Evaluation results for quality deficits in some of the data sets. First four rows, left to right: one under-sequenced replicate with low enrichment and low quality, two under-sequenced replicates with low enrichment and low quality, and a mixture of quality inspired by real data for H3K4me3 in embryonic stem cells. Second four rows, left to right: one, two, and three noisy replicates.



## Summary

Even using the defaults settings—no deletion or down-weighting of replicates—the performance of Sierra Platinum on *noisy* data is superior compared to the performance of other peak-callers on *noise-free* data. In general, deleting or down-weighting replicates increases the performance of Sierra Platinum on noisy data reaching the performance of Sierra Platinum on noise-free data. Thus, the method implemented in Sierra Platinum is robust against any kind of noise in the data. Moreover, the implemented user interactions for deleting and down-weighting replicates in combination with the visual quality control features allow fine-tuning of the peak-calling results to obtain the optimal results for each data set.

## 3.8 Results

### 3.8.1 Real World Data Sets

#### Reference Data

The peak-calls for the epigenome E003 generated by the NIH Roadmap Epigenome project [68] were used as reference. From these, the consolidated broad peaks for the embryonic stem cell line H1 were used.

According to the Roadmap Epigenomics Consortium [68], the ‘MACS-CR approach’ (Section 3.7.4) was used to obtain the peak-calls for E003. In this approach, first all libraries for H1 are down-sampled to 30M reads per sample. Next, libraries measuring the same modification belonging to the input were merged into one file. Afterwards, MACS2 was applied on this data to generate the peak-calls for the different modifications.

#### Input Data of Sierra Platinum

**Data Set for H1** As test data set for Sierra Platinum, the replicates of the H1 cell line (an embryonic stem cell line) from the NIH Roadmap Epigenomics Project [68] were used. In a first step, the raw data for five histone modifications—H3K4me3, H3K27me3, H3K9me3, H3K27ac, and H3K9ac—and the corresponding ChIP-input as background was downloaded if available. Table 3.9 provides the GEO identifications and the sequencing center, which produced the data set. An ID was assigned to each replicate for further reference to the replicates.

**Data Set for ESCs** As a second test data set (Table 3.11), the data for all other embryonic stem cell lines available at the NIH Roadmap Epigenomics Project were also downloaded. For this data set, 4 modifications—H3K4me3, H3K27me3, H3K9me3, and H3K27ac—and the corresponding ChIP-Input as background were downloaded, if available. In all cases, only those modification data were used for which a fitting ChIP-input was available. For this data set, it was possible to obtain up to 16 replicates for each modification.

For each modification and replicate the available sra files from GEO [33, 20] were downloaded. Then, the sra files were converted into fastq files and an adapter clipping was applied. Afterwards, the reads were mapped using `segemehl` [44] with an accuracy of 80% against the human genome version

hg19 [51]. Using SAMtools [53], the results were converted into one bam file for each data set. Since not all embryonic stem cell lines were male, the chromosomes X and Y were removed in the data of the second test set. In the cases where the data set consists of multiple fastq files, the mapping results were merged in addition to sorting and indexing them. In a last step, the Picard Tools [3] were used to remove PCR duplicates.

### **3.8.2 Peak Agreement**

The analysis of the agreement of the peaks predicted by Sierra Platinum and by different publicly available peak-callers is based on H3K4me3 measurements of all three replicates of BMP4 Trophoblast Cells in the GEO Series GSE16256. The procedure described in Section 3.6 was used to calculate the agreement between the different results. Since there is no gold standard to decide, if a peak is valid or not, only the agreement between the methods was calculated. In a first step, all pairwise overlaps were calculated and for each overlap the original peak of both inputs was stored.

Calculating the overlap of three methods is slightly involved, since one has to take into account that pairwise overlapping peaks can not simply be intersected with each other. In a first step, one would overlap two pairwise overlapped peak sets, where one method was used in both pairwise overlappings. As shown in Figure 3.38, four different cases can occur while calculating the overlap of three different methods (without consideration of the symmetrical cases). The first case is the valid peak overlapping for the three methods, since each peak generated by the different methods overlaps at least 50% with the other peaks. Case 2 and 3 are the trivial cases where no overlap exists between all three methods. Case 4 is slightly more difficult, because the overlaps of the pairwise overlappings have to be intersected. Overlapping a peak with a merged overlap of two peaks would not lead to the correct result, since the overlap of the merged peaks is probably larger than the two peaks. Therefore, it is necessary to intersect the peaks from each method with each other to detect this non-overlapping case.

After calculating each triple overlap, it is necessary again to overlap the peaks of each method while calculating the quad overlap. Since BinQuasi was not able to detect any significant peak on the used data set, it is not necessary to calculate even higher overlaps.

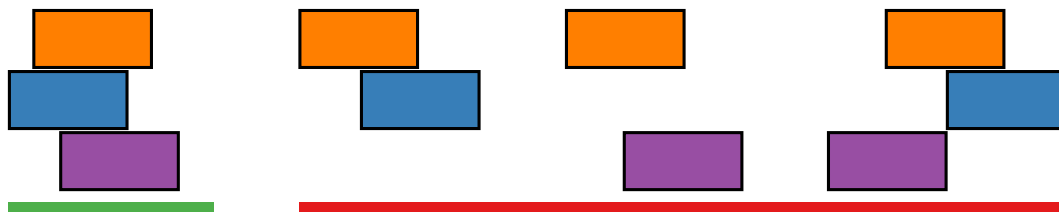


Figure 3.38: Overview of the possible peak overlaps with three replicates. Only the first overlap (marked with green) is a valid peak overlap and should be counted, the other three (marked with red) are discarded. Similarly, the merging process overlaps the peaks of all four peak-calling methods.

To calculate the agreement of the different methods, peaks were predicted with Sierra Platinum, the MACS-SA and MACS-CR approaches, PePr, and BinQuasi. Unfortunately, BinQuasi was not able to predict any peaks with the given data. In addition, BinQuasi was also tested with the other data sets from the NIH Roadmap Epigenome project [68]. Here, too, BinQuasi could not detect any peak. Due to a similar methodology, PePr can not find any peaks in many data sets. One explanation for this is the higher amount of noise between the replicates. This behavior of BinQuasi and PePr has already been demonstrated in the benchmarks (see Section 3.5), whereby BinQuasi reacts significantly more sensitive to noise than PePr.

The overlap of the peak predictions is shown in Figure 3.39. MACS-CR predicted the largest amount of peak with 68754 segments, followed by MACS-SA (44845 peaks) and Sierra Platinum (47442 peaks). PePr predicted 32186 peaks. As stated above, BinQuasi was not able to predict any peak. Only the MACS-SA side overlaps nearly completely with the MACS-CR side and all approaches show large overlaps among each other.

### 3.8.3 Peak-Calls

Sierra Platinum was configured to generate peak-calls with a window size of 200nt, a window offset of 50nt, and a  $p$ -value cutoff of  $10^{-5}$ . Probits were corrected for inter-replicate correlation. The Holm-Bonferroni method is used to calculate the  $q$ -value.

#### Peak-Calls for H1

Using the visual quality controls, the replicates for H1 were analyzed with respect to noise. Given the results, the decisions shown in Table 3.10 were made where the *replicate ID* refers to the ID given in Table 3.9, the *weight*

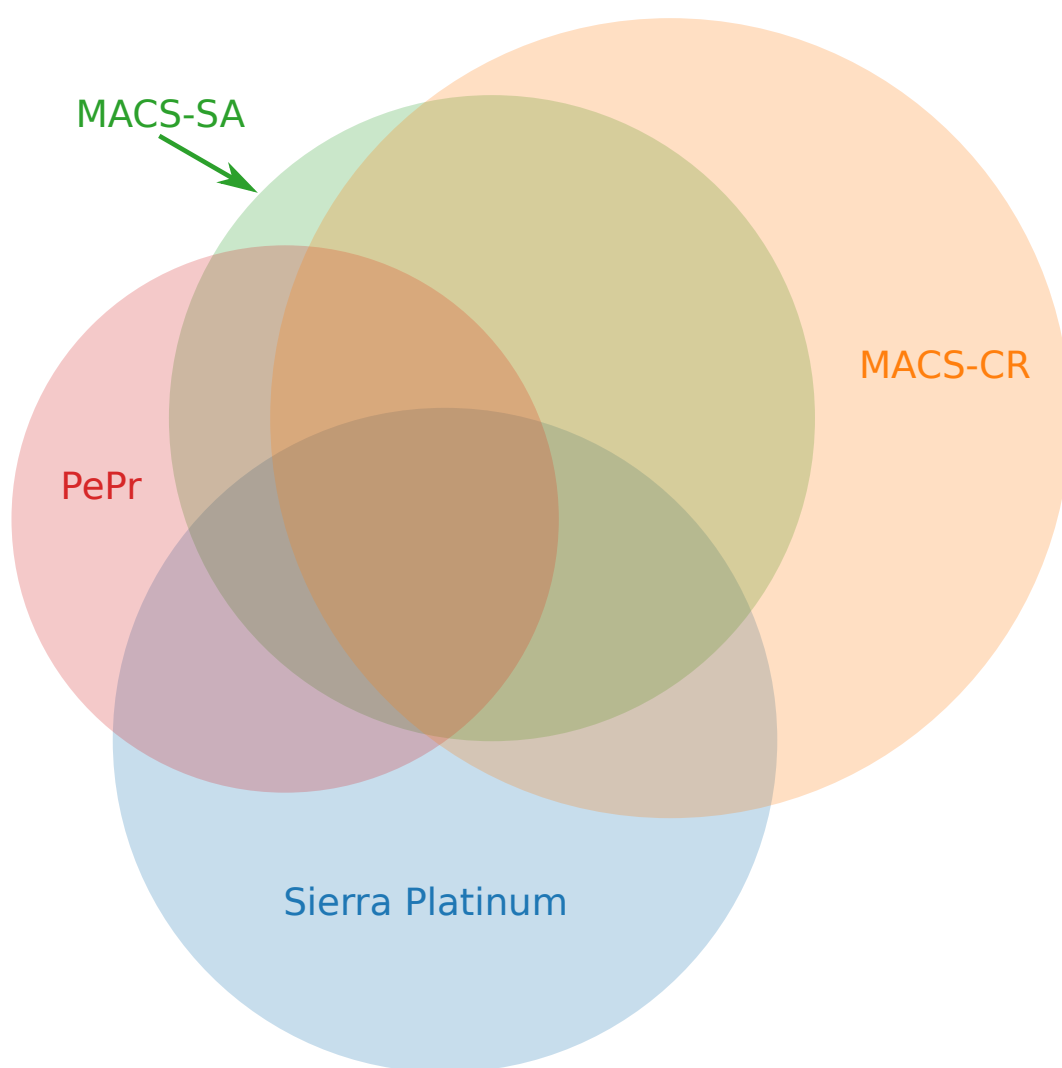


Figure 3.39: Agreement of the peak predictions: The overlap of the peaks predicted by Sierra Platinum (blue), MACS-SA (green), MACS-CR (orange), and PePr (red) is shown.

specifies the weight that was used for weighting the replicate, and *off* means that the replicate was excluded.

### **Peak-calls for ESCs**

Using the visual quality controls, the replicates for ESCs were analyzed with respect to noise. Given the results, the decisions shown in Table 3.12 were made where the *replicate ID* refers to the ID given in Table 3.11, the *weight* specifies the weight that was used for weighting the replicate, and *off* means that the replicaten was excluded.

Table 3.9: Overview of the data sets used for H1. IDs are only used internally to distinguish the different replicates available for H1. For each replicate, the sequencing center, which produced the data, and the GEO identifications are provided for the data sets H3K4me3, H3K27me3, H3K9me3, H3K27ac, H3K9ac, and ChIP-input.

ID	Center	H3K4me3	H3K27me3	H3K9me3	H3K27ac	H3K9ac	Chip-Input
1	UCSD	GSM469971	GSM466734	GSM605325	GSM466732		GSM605333
2	UCSD	GSM605315		GSM605327		GSM605323	GSM605339
3	UCSD			GSM818057			GSM667642
4	UCSD	GSM409308	GSM434776			GSM434785	GSM605334
5	BI	GSM433170	GSM433167	GSM433174		GSM433171	GSM433179
6	BI	GSM537681					GSM537682
7	UCSF-UBC	GSM432392		GSM450266			GSM450270
8	UCSF-UBC			GSM428291		GSM410807	GSM428289
#replicates		6	3	6	1	4	8

Table 3.10: Decisions based on visual inspection of the quality of the replicates listed in Table 3.9. *ID*: ID given Table 3.9, *Weight*: weight used, *off*: replicate excluded, *empty cell*: replicate not available.

ID	H3K4me3	H3K27me3	H3K9me3	H3K27ac	H3K9ac
1	0.1	1	0.05	1	
2	0.1		0.05		1
3			1		
4	1	0.05			0.1
5	0.1	0.1	0.1		1
6	off				
7	1		off		
8			1		off



Table 3.11: Overview of the data sets used for ESC: IDs are only used internally to distinguish between the different replicates available for embryonic stem cell lines. For each replicate, the sequencing center, which produced the data, and the GEO identifications are provided for the data sets H3K4me3, H3K27me3, H3K9me3, H3K9ac, and the ChIP-input.

Id	Center	H3K4me3	H3K27me3	H3K9me3	H3K9ac	ChIP-Input
1	UCSD	GSM409308	GSM434776		GSM434785	GSM605334
2	UCSD	GSM469971	GSM466734	GSM605325		GSM605333
3	UCSD	GSM605315	GSM605308		GSM605323	GSM434785
4	UCSD	GSM616128	GSM706066	GSM667633	GSM616129	GSM667643
5	UCSD			GSM605327		GSM605335
6	BI	GSM433170	GSM433167	GSM433174	GSM433171	GSM433179
7	BI	GSM537681	GSM537683			GSM537682
8	BI	GSM669889	GSM669887	GSM669886	GSM669963	GSM669888
9	BI	GSM669893	GSM669897	GSM669894	GSM669890	GSM669895
10	BI	GSM772978	GSM772977	GSM772856	GSM772980	GSM772913
11	BI		GSM773002		GSM773003	GSM772979
12	BI	GSM669936	GSM669942	GSM772799	GSM670013	GSM772794
13	BI	GSM772797	GSM772766		GSM772796	GSM772755
14	BI	GSM537665	GSM537648	GSM537639	GSM537670	GSM537647
15	UCSF-UBC		GSM428295	GSM428291		GSM428289
16	UCSF-UBC	GSM432392		GSM450266		GSM450270
#replicates		13	14	11	11	16

Table 3.12: Decisions based on visual inspection of the quality of the replicates listed in Table 3.11. *ID*: ID given Table 3.11, *Weight*: weight used, *off*: replicate excluded, *empty cell*: replicate not available.

ID	H3K4me3	H3K27me3	H3K9me3	H3K9ac
1	0.02	1		0.8
2	0.02	0.8	1	
3	1	1		0.8
4	off	0.02	1	0.01
5			1	
6	0.02	0.01	1	off
7	off	off		
8	1	1	1	0.01
9	1	1	1	0.01
10	0.02	1	1	off
11		1		1
12	1	1	1	1
13	1	1		1
14	off	0.02	1	off
15		0.02	1	
16	0.02		1	

3.8.4 Hox-C and Hox-D Clusters

In this section, the epigenome state of whole clusters of genes are analyzed, i.e., the Hox-C and the Hox-D clusters. The Hox clusters (there are also Hox-A and Hox-B) are clusters of transcription factors that are important for embryonal development and differentiation. Therefore, the regulation of these clusters is crucial.













They are conserved in all mammals. However, within the Hox-C cluster, the famous lncRNA HOTAIR is located. HOTAIR was found to drive the regulation of the Hox-D cluster by repressing it [67]. However, the function of HOTAIR might be specific to humans since it was so far not found in other mammal. Furthermore, its function was even disproved in mouse [17].

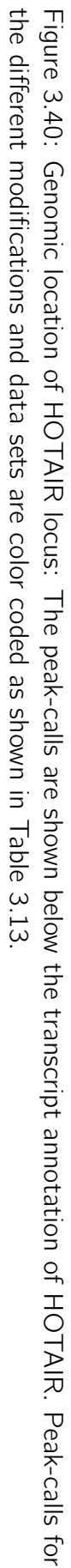
Firstly, the epigenomic state predicted with Sierra Platinum on the H1 data set (see Table 3.9). Secondly, the epigenomic state predicted with Sierra Platinum on the ESCs data set (see Table 3.11). Thirdly, the consolidated epigenomic state of the H1 cell lines downloaded from the NIH Roadmap Epigenomics Webportal of the Washington University (epigenome E003, only H3K4me3, H3K27me3, and H3K9me3).

In Figures 3.40–3.42, one can see the HOTAIR locus, the containing Hox-C cluster, and the Hox-D cluster. Given the knowledge about the clusters, one would expect that most of the promoters are inactive (marked with H3K27me3) or poised (marked with H3K4me3). Thus, the genes in the clusters are inactive or are already primed for activity. One would not expect to find H3K9me3 marks in there.

The presence of H3K27me3 at all loci is predicted by all three approaches. However, E003 predicts more H3K4me3 marks. While in the case of HOTAIR this might be correct (even though neither the H1 nor the ESCs based prediction of Sierra Platinum do predict H3K4me3 peaks), the abundance of peaks which

Table 3.13: Color coding used in the figures showing the peaks as UCSC tracks at selected genomic positions.

	H1	ESC	E003
H3K4me3			
H3K27me3			
H3K9me3			
H3K27ac			
H3K9ac			







do not co-occur with promoter regions is suspicious and may indicate an over-prediction of H3K4me3 marks. Even more suspicious is the massive amount of H3K9me3 marks predicted by the E003 epigenome. This would mean that these regions are strongly repressed, which was not found so far for embryonic stem cells. Furthermore, it is not possible to confirm the presence of H3K9me3 in the clusters using Sierra Platinum on the two embryonic stem cell data sets.

### 3.8.5 Peak Coverage Analysis

In this section it is analyzed whether the predicted peaks for the combined replicates are supported by the single replicates. Therefore, the read coverage was calculated of each peak by the corresponding replicate. The resulting coverage distributions are simplified by counting only how often peaks are not covered at all, by 1 - 200 reads and by more than 200 reads. For replicates having a high weight, one can expect to find a strong support of almost all peaks since those are the replicates having a strong influence on the final  $p$ -value. Replicates having a low weight are not expected to fully support the peaks predicted by Sierra Platinum. Those replicates having low weights are noisy or of bad quality and therefore, may not reflect the true epigenome state. Replicates, which are that are even worse, were excluded from peak-calling, since they are not expected to support the peaks.

Figure 3.43 shows the analysis results. For all replicates, most peaks fall into the categories 1-200 reads coverage and more than 200 reads coverage. Thus, in all replicates most peaks have at least a weak support.

Peaks with no support are almost only found when the replicate is strongly down-weighted during peak-calling. For example replicate 16 was down-weighted for peak-calling since the correlation with the other replicates was low and because the tag distribution indicated under-sequencing effects. About 10.000 peaks are unsupported by this replicate. This strongly supports the decision to not rely too much on the replicates and shows that Sierra Platinum overcomes the problem that peaks may not be supported by all replicates. While the support of replicate 16 was used for the remaining 50.000 peaks, the 10.000 unsupported peaks are not strongly affected by the presence of replicate 16.

On the other hand, replicates having a high weight (such as replicate 8 and 9) support most of the peaks. In the case of replicate 8, 9, and 10, most peaks are supported with more than 200 reads. In summary, the peak-calls are supported by the replicates.



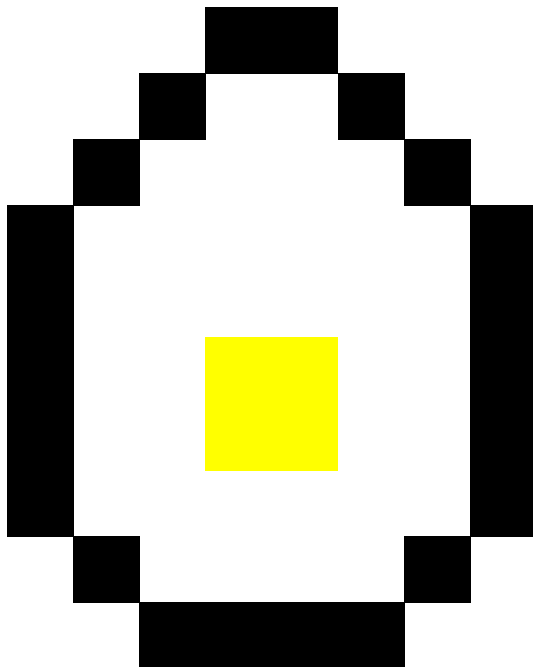


Figure 3.43: Peak coverage for each replicate: For each peak, the number of supporting reads is calculated. Peaks are counted in the categories no supporting reads (red), 1-200 supporting reads (blue), and more than 200 supporting reads (green). The peaks are the H3K4me3 peaks predicted by the data set ESC. Replicate numbers refer to those in Table 3.11.



## Chapter 4

### iDotter



## 4.1 Introduction

In bioinformatics, one frequent task is judging the likelihood of the overall RNA secondary structure. As described in Section 2.3, the secondary structure of an RNA is defined by its base pairs. These base pairs can be predicted with special algorithms and therefore, the probability for two nucleotides of an RNA sequence forming a base pair. Dot plots are used for displaying probabilities or similarity measures between a row and a column of a matrix. Hence, dot plots are frequently used for analyzing RNA secondary structure likelihood displaying the probability of a row and a column nucleotide forming a base pair.

Most currently available tools generate dot plots in post-script (ps) format (e.g., [54, 39]). These ps-images are then viewed using suitable postscript viewers. However, postscript itself is no longer actively developed and was replaced by the portable document format (pdf). Moreover, the images are static and possible interactions are restricted to standard *viewing* interactions like geometric zooming and panning the image. Further, the scalability of this approach is low as during zoom-in the nucleotide sequence that is displayed at the border of the image might not be visible any more.

Therefore, iDotter, an interactive tool for analyzing RNA secondary structures, was developed that overcomes these limitations. Concretely, the contributions of this tool are:

- Sophisticated zooming and panning methods, preserving the context of the zoomed and the panned area [75]
- Presenting details for each dot on demand [75]
- Highlighting of semantic units in the dot plot
- Recoloring of the dot plot for annotation and presentation
- Exporting parts or the whole dot plot for further analysis and for presentation
- A powerful API for using iDotter within analysis pipelines
- A sharing function for collaborative analyses

## 4.2 Background and Related Work

Dot plots were introduced by Gibbs and McIntyre [36]. Originally, dot plots were used to visualize alignments of two nucleotide sequences or proteins. A dot plot is a two dimensional matrix where the sequences 'A' and 'B', that are compared, are visualized on the x- and y-axis, respectively. A dot in a cell means that Sequence 'A' is similar to Sequence 'B' at this nucleotide/amino acid (position). Both color and size of a dot represent the strength of the similarity of the sequences calculated using application dependent measurements. With the aid of dot plots, identifying highly similar regions between two sequences is easily possible. These regions are the diagonal lines in the matrix. An example for an interactive dot plot viewer for alignments was introduced by Sonnhammer and Durbin [76]. In this thesis, however, the focus is on RNA folding structures that can not be handled by their program. Moreover, their tool does not provide additional interactions like highlighting, semantic zoom, and export of (sub-)sequences.

While the nucleotide sequence (RNA primary structure) is important for the analysis of RNA sequences, the folded structure of the RNA (RNA secondary structure) provides additional vital information. With the emergence of RNA folding tools [54, 57, 66], visualizing RNA secondary structure became more and more important to foster its analysis. Tools like Varna [31] or the NAVIEW algorithm [22] generate graph-based, node-link visualizations of a single RNA secondary structure showing *one* possible folding of the RNA, only.

In addition to the graph-based visualizations, dot plots were adapted to visualize the predicted base pair probabilities within a single RNA sequence. Furthermore, they support analyzing the changes of two similar RNA sequences of different species. Usually, the size of a dot describes the probability of a base pair between the corresponding nucleotides.

Static dot plots can be calculated with R using the R package R-CHIE [26]. The ViennaRNA package [54] can generate one dot plot in postscript format for each RNA secondary structure prediction (an example being shown in Figure 4.1). Moreover, the ViennaRNA Web Services [39] provide the functionality of the ViennaRNA package without the necessity to compile the package. Therefore, it can be used platform independently.

While the original dot plots of Gibbs and McIntyre [36] for alignments show the same information in the upper and the lower triangle, dot plots generated by RNA folding software contain two different folding predictions as shown

in Figure 4.1, e.g., the energetically best solution and all possible base pair probabilities in the lower and upper triangles, respectively.

An alternative for visualizing RNA secondary structure predictions is the arc diagram introduced by Wattenberg [83] and later implemented as arc plot in R [50]. The RNA sequence is plotted as a linear sequence and an arc between two nucleotides describes a base pair while the color of an arc might encode the probability of the pair. Besides the fact, that this approach has limited

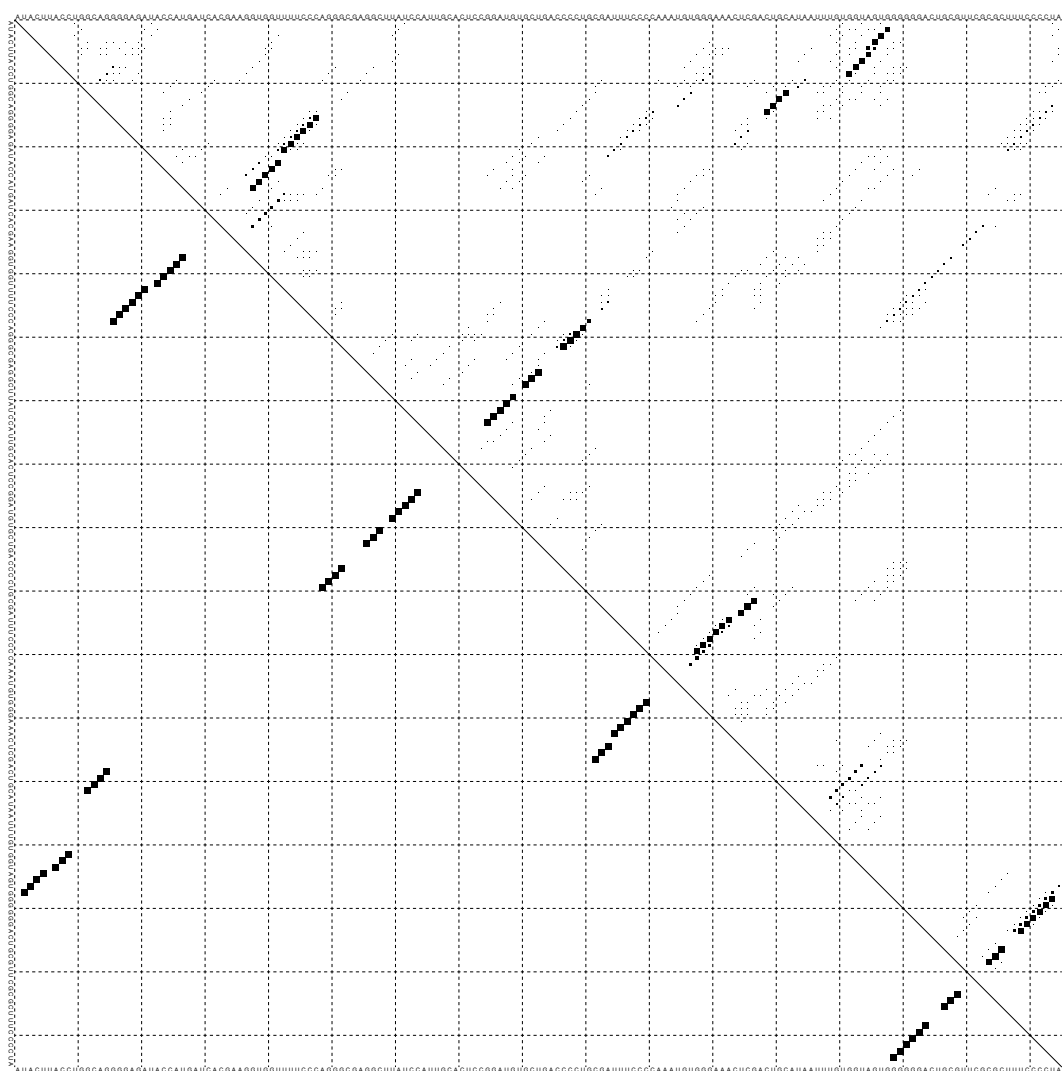


Figure 4.1: Overview of the postscript dot plot generated by ViennaRNA [54] showing base pair probabilities of an RNA. Black squares are used for showing the possibility of two nucleotides forming a base pair. The probability for forming this base pair is encoded in the size of these squares: large squares imply a high probability, while small squares imply a low probability. The diagonal is used as a landmark only. It divides the upper-right triangle showing the consensus probabilities from the lower-left triangle showing the probabilities according to the energetically best solution.

scalability, the arcs produce a lot of clutter and it is hard to determine the corresponding base pairs.

Arc diagrams and dot plots can be used for character sequence comparison (alignment) in general. For arc diagrams this was already introduced in the original paper [83]. Abdul-Rahman et al. [15] use dot plots to visualize text alignments between different documents.

### 4.3 Problem Statement: Current State and Issues

A dot plot fulfills the standard design goals taken from the information visualization literature [81]. Dot plots

1. are flexible (can be used for different tasks and application areas)
2. are space efficient
3. provide a good overview of the data
4. ease the identification of pattern in the data
5. are fast to create

However, dot plots are static images without any interaction provided. Figure 4.2 shows a relatively small RNA having a length 165nt. As can be seen in the ps version (Figure 4.1), the nucleotide names are barely readable. Moreover, it is difficult to impossible to spot small base pair probabilities. Zooming into a part of the ps view is possible (Figure 4.3a). Then, all dots become larger and small base pair probabilities are more easily spotted. However, due to the limitations of the ps-viewers, the nucleotide sequence related to the zoomed-in area might no longer be visible.

### 4.4 Solution

To overcome the limitations of existing dot plot generators, iDotter was developed: a fully interactive web-interface that supports users in analyzing RNA secondary structure. iDotter is based on the dot plots generated by existing folding tools. After importing the data (Section 4.4.1), the dot plot is shown in the web browser (Section 4.4.2). Then, the user can zoom in and out as well as pan the view (Section 4.4.3). Moreover, the user can mark rectangular regions of dots as well as single columns and single rows in the dot plot (Section 4.4.3).

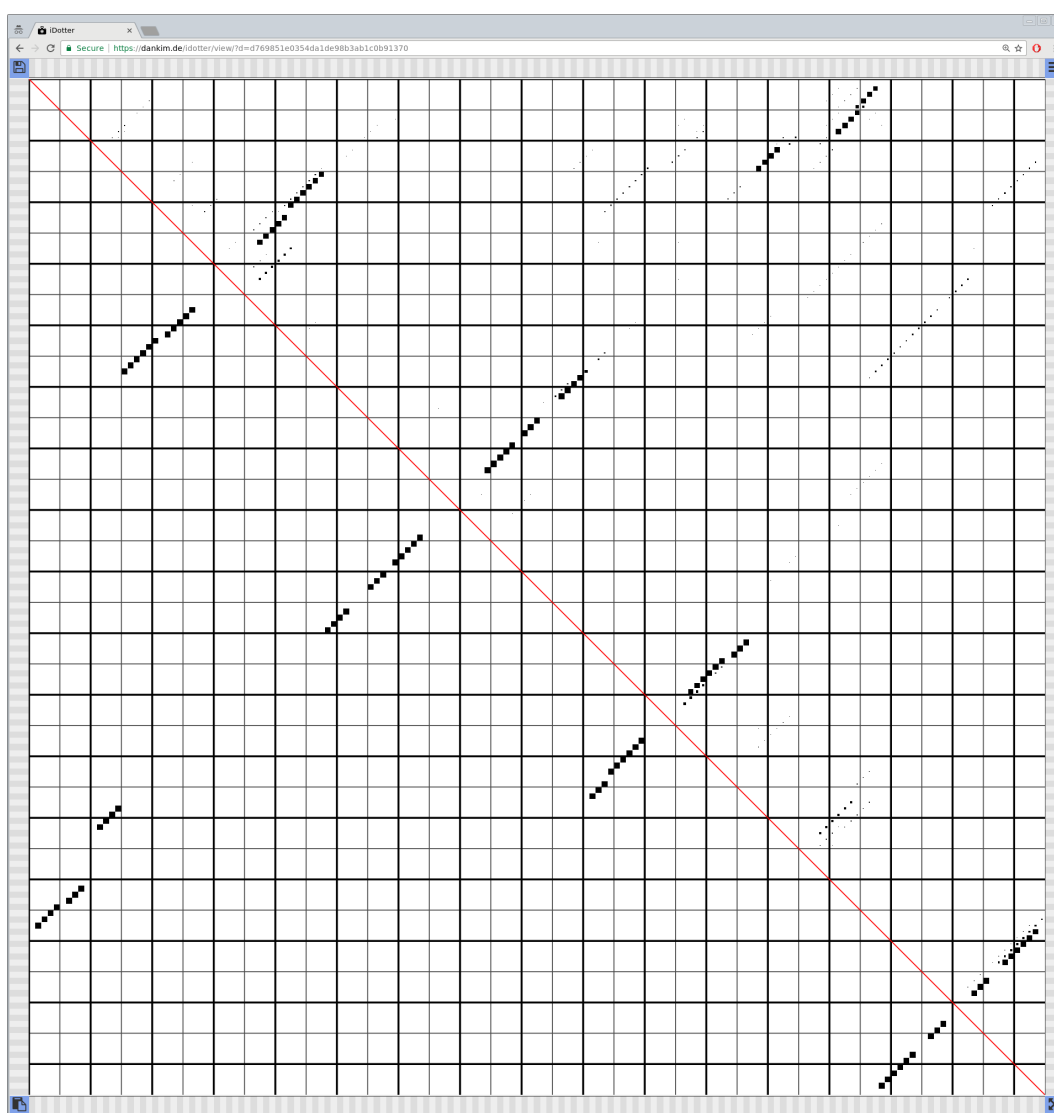
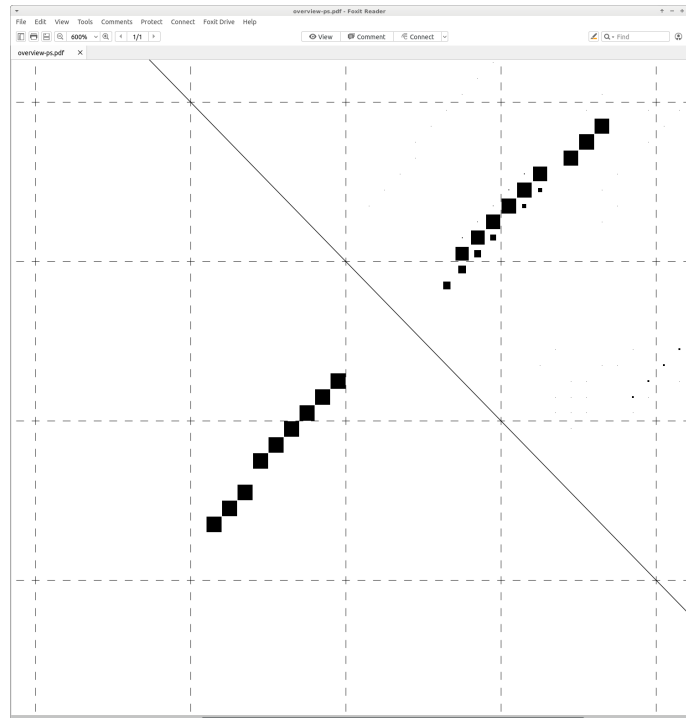
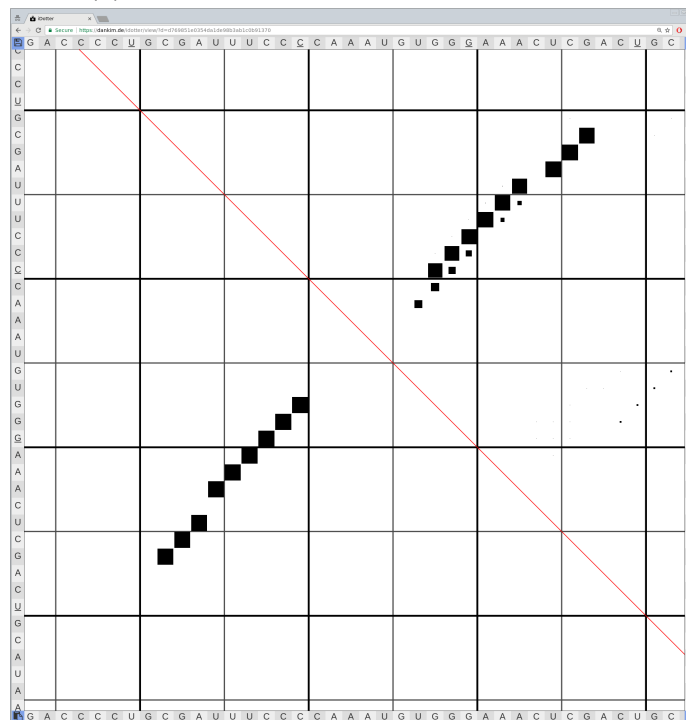


Figure 4.2: Overview of the iDotter interface showing the same dot plot as Figure 4.1. The nucleotide sequence is only shown at the borders, if the nucleotides are readable in the current zoom level.





(a) PostScript-View, zoom-in of Figure 4.1



(b) iDotter view, zoom-in of Figure 4.2

Figure 4.3: Comparison of the dot plot interfaces after zooming into a sub-sequence. In the postscript view (a), the nucleotide sequence is no longer visible, while in iDotter (b) it stays visible at all borders easing the analysis of sub-sequences.

Finally, the highlighted part of the dot plot or the complete dot plot can be exported in postscript-format (Section 4.4.3). A web-based API provides a connection to dot plot generating services (Section 4.4.4).

### 4.4.1 Data Import

#### Parsing Postscript

After starting iDotter, the original ps-file generated by the ViennaRNA package [54] is transformed into a JSON file by iDotter, if the JSON file does not already exist. To do so, the RNA sequence, as well as the ubox and the lbox containers are extracted from the ps-file and stored in a JSON array representing the box plot. Each ubox and lbox container comprises an x- and a y-coordinate designating the cell in the dot plot matrix, the size of the dot, and the color of the dot. The color information is optional. By transforming the input file into a generic JSON file iDotter can easily be extended to other input types by implementing a corresponding import routine.

#### Reformat Data

During the parsing of the original ps-file iDotter converts the color values of the dots, if they are present. Due to numerous different implementations of the color model that can be used in the dot plot, the color value is mapped to a value between 0 and 1. This value is later used as the hue value in the internal color model of iDotter. However, all original values are stored besides the converted value in the JSON file so that the user can export selected regions with the original color model to a ps-file (Section 4.4.3).

### 4.4.2 Visualization

The JSON file is imported by iDotter and the complete dot plot (zoom out) is displayed in the browser (Figure 4.2). This follows the Shneiderman Mantra, presenting an “overview first” [75]. On each border, the nucleotide sequence is displayed. For convenience, the diagonal showing the same nucleotide on both the x- and the y-axis is shown in red. At the same time, this diagonal separates the upper from the lower triangle of the matrix. In the upper and the lower triangle, either the same or two different base pair probabilities (encoded as size in the input file) are shown. The size of each dot is encoded depending on

the zoom level so that the user can compare the probabilities easily on each zooming level. As default, the consensus probabilities are shown in the upper and the energetically best solution probabilities are shown in the lower triangle of the matrix, respectively. The probability of a dot is mapped to its size. The color can be used to represent, e.g., the conservation of the sequence during evolution. An adaptive background grid is displayed to enable an easy counting of the base pairs. Matching the zoom-level of the dot plot, the different grid levels can be shown or faded out. This view corresponds to the zoomed out standard dot plots, except that the nucleotide sequence is always shown, except if the text becomes unreadable.

### **4.4.3 Interaction**

#### **Zooming and Panning**

The second step in Shneiderman's Mantra is "zoom and filter" [75]. The user can use the semantic zoom to more closely analyze a sub-sequence (Figure 4.3b). The user benefits from the sequence labels staying visible at all borders of the dot plot all the time. This improves the scalability with respect to the size of the data that can be analyzed conveniently, which is an improvement over the state of the art (Figure 4.3a) where the nucleotide sequence might disappear during zooming. Moreover, the individual nucleotides of the nucleotide sequence are only shown, if the zoom level allows displaying them in a readable manner. Otherwise, they are hidden (Figure 4.2). The semantic zoom is triggered by mouse wheel motion. Moreover, the user can pan the viewport by holding the left mouse button and moving the mouse. Also during panning, all sequence labels are staying visible all the time as long they are readable.

Filtering of the original data is not provided, because it would not be useful in this context. However, parts of the dot plot can be selected and this selection can then be exported (see below). This corresponds to a filtering step its primary applications being reporting and collaborating.

#### **Details on Demand**

The third step in Shneiderman's Mantra is "details on demand" [75]. While working with the dot plot, information about individual dots can be displayed on demand as a tool tip by mouse over. Then, all available information is shown (Figure 4.4). Thus, the user can get the exact information about the nucleotides

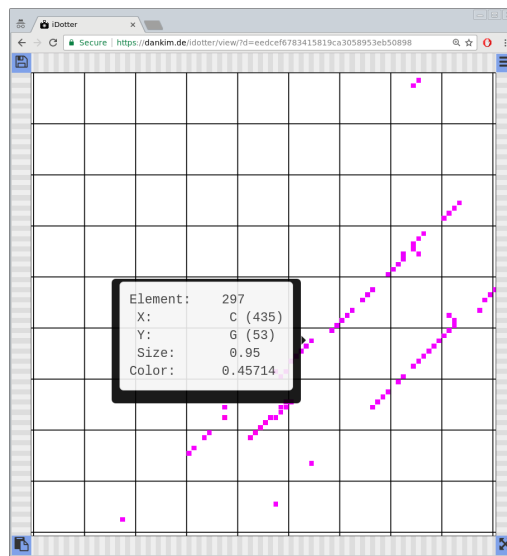


Figure 4.4: Each dot provides details on demand by mouse-over showing a tool tip: element ID, X showing the nucleotide of the column and its position, and Y showing the nucleotide of the row and its position. The (biological) attributes mapped onto 'Size' and 'Color' are application dependent.

(names and positions) involved in a base pair even though the respective names are no longer visible at the corners because they would be too small to be read. Moreover, the values for the size (here: representing the base pair probability) and the color are shown.

## Highlighting

Following the taxonomy of Yi et al. [88], selecting dots is provided by iDotter. The user can mark a dot by left clicking on it (Figure 4.5a). Then, the selected dot is highlighted with the 'Selected Dot Color' (Figure 4.7). Moreover, the user can select multiple dots by left clicking into the viewing area and dragging the mouse while holding the 'Shift' key pressed. This creates a rectangular region. Within this region, all columns and rows that contain dots are highlighted with the 'Dotmarker Color' (Figure 4.7). Additionally, the selected dots are highlighted using the 'Selected Dot Color' (currently yellow) in both cases. Deselecting a region of dots is achieved by pressing the 'Ctrl' key while using the mouse. Besides marking dots, the user can mark single columns by left clicking on them (Figure 4.5b). Then, the selected column is highlighted with the 'Linemarker Color' (see Figure 4.7). In the same way, the user left clicks on a row to select it (Figure 4.5b). Both—marking dots as well as marking columns and rows—can be combined (Figure 4.6) to mark those parts of the dot plot

that are of interest to the user. Finally, the selection can be reset by pressing the button having the 'three horizontal bars' icon in the upper right corner of the dot plot invoking the settings dialog (Figure 4.7), and then pressing the 'Remove Marker' button there.

### Data Export

After working with the data, the user can export the *highlighted parts* of the dot plot into a new ps-file for publication or other purposes by pressing the disc icon in the *upper left* corner and selecting the menu item 'export only selection'. Further, the user can export the *complete* dot plot into a new ps-file by pressing the disc icon and selecting the menu item 'export all'.

### URL Export

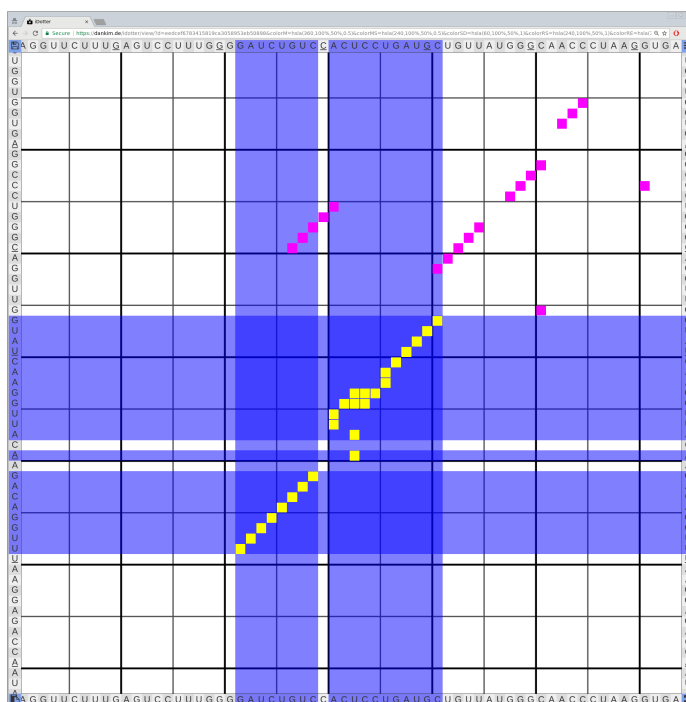
The URL export is triggered by pressing the clipboard icon in the *lower left* corner. The URL contains all necessary parameters and is copied into the clipboard of the operating system. The user can copy it afterwards to any application.

### 4.4.4 API

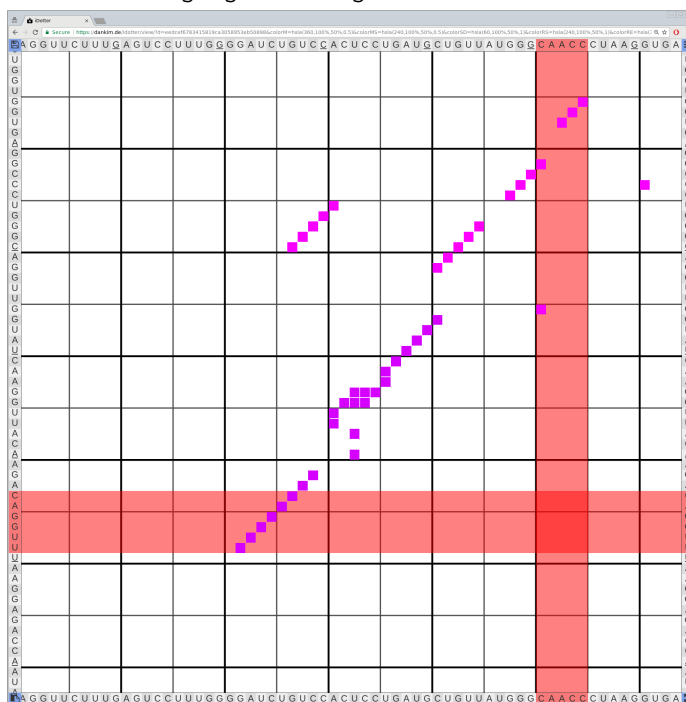
iDotter comes with a web-based API that provides a connection with dot plot generating services like the ViennaRNA Web Services [39]. This API supports direct import of ps-files into the view, pre-selecting highlighted regions, and exporting the highlighted regions for automatic workflows. The API is controlled by URL parameters. This type of control provides iDotter with additional possibilities for collaboration between users. The user can export his current settings, like zoom level, position, and color settings, and share these with his collaborators or save them for documentation purposes.

### 4.4.5 Implementation

iDotter provides an interactive web interface that is implemented using the current state of the art web-programming languages HTML5, PHP, and JavaScript. The upload functionality and the API are implemented in PHP, because the web server executes them on the server hardware. All other functions are executed on the local machine of the user to reduce the computational overhead of the server. Therefore, this application consumes only a low amount of run time



(a) The 'mark dot' interaction allows selecting single dots by clicking on them. In this case, the selected dot is highlighted with the 'Selected Dot Color' (see Figure 4.7). Moreover, multiple dots can be selected by marking a rectangular region. (Clicking into the viewing area and dragging the mouse while holding the 'Shift' key pressed. For deselection, the 'Ctrl' key should be pressed instead.) All columns and rows that contain dots in the selected region are highlighted with the 'Dotmarker Color' (Figure 4.7), while the selected dots are highlighted using the 'Selected Dot Color' (Figure 4.7).



(b) The 'mark row' interaction allows selecting single rows by clicking on them. In this case, the selected row will be highlighted with the 'Linemarker Color' (see Figure 4.7). In the same way, columns can be selected.

Figure 4.5: Highlighting dots 4.5a as well as rows and columns 4.5b.

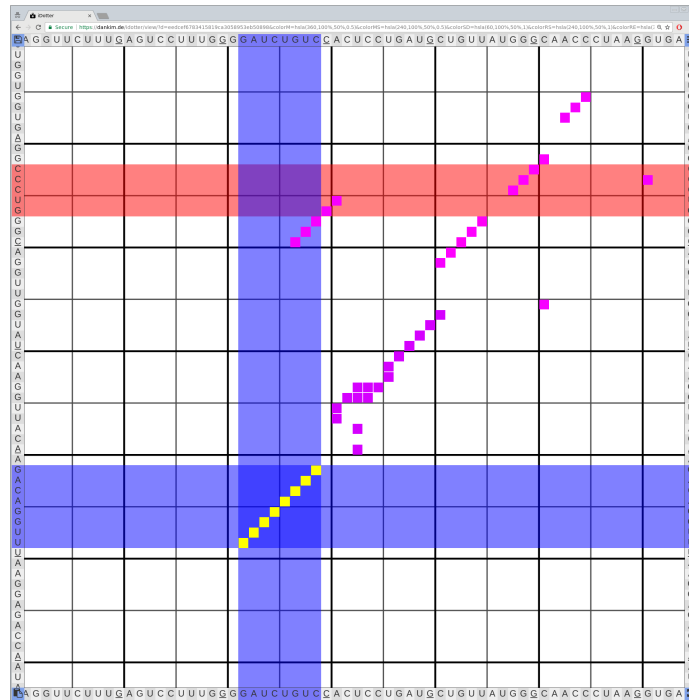


Figure 4.6: Marking dots and regions of dots (Figure 4.5a) and marking rows and columns (Figure 4.5b) can be combined.

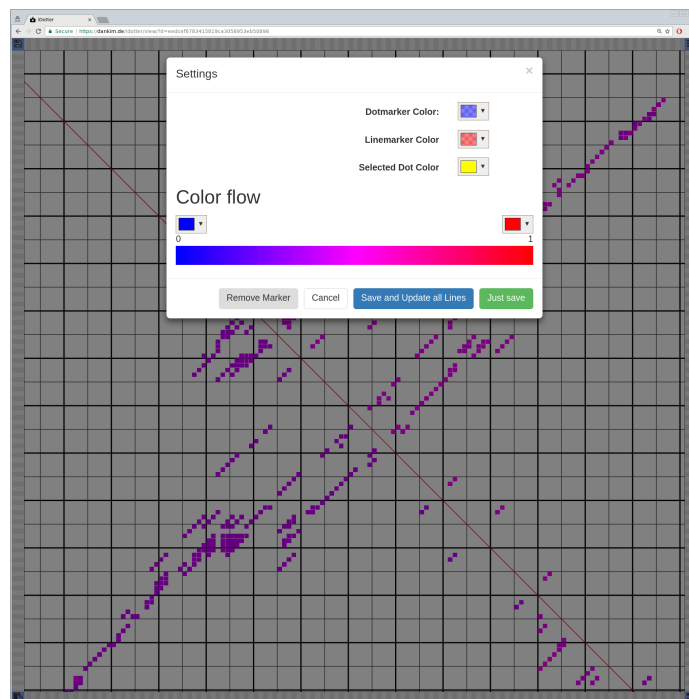


Figure 4.7: In contrast to the postscript visualization, iDotter provides choosing the color gradient. Additionally, choosing the highlighting colors for dots ('Selected Dot Color', Figure 4.5a) and columns/rows ('Linemarker Color', Figure 4.5b) is possible. Moreover, it is possible to reset highlighting in the dot plot by pressing the 'Remove Marker' button.

and memory on the server. In addition, the server stores the dot plots that have already been used locally so that the user can select them from a list when navigation to the start page of the web service in order to facilitate its usage of the web service.

## 4.5 Results

In a small testcase study, a biological collaborator used iDotter for analyzing the evolution of long non coding RNAs (lncRNA). Since these RNAs are longer than 200nt, it is challenging to analyze the generated dot plots in ps-format due the lack of interactivity. Furthermore, it is hard to compare specific regions between different dot plots. For that reason, the expert used the interactivity features for selecting regions of interests. By exporting these regions with the API from all investigated RNA samples, it was possible to detect evolutionary changes between several species. According to the biological collaborator that regularly uses dot plot viewer, iDotter outperforms previous approaches with respect to facilitating dot plot based analysis of RNA secondary structures.

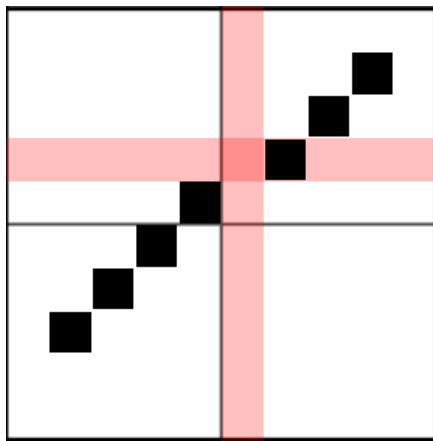
## 4.6 Use Case

Each dot of the dot plot shown in Figure 4.2 represents the probability how likely a base pair between the row and the column nucleotide is. The energetically best solution is presented in the lower triangle of the dot plot. Since only one prediction is shown here, exactly one probability per nucleotide is shown. This predicted folding is also shown in Figure 4.9 as a graph layout drawn by RNApuzzler.

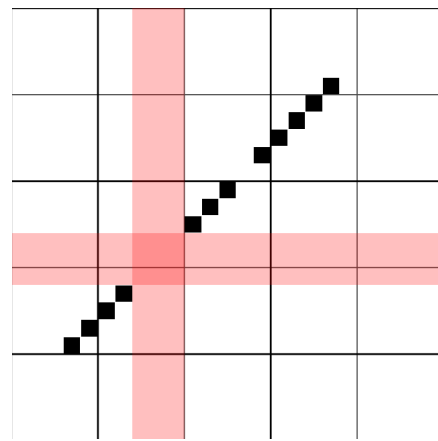
A diagonal sequence of directly consecutive dots represents a stem in the RNA secondary structure. If this sequence is interrupted by a single column, the stem contains a bulge at this position (Figure 4.8a). If a sequence of dots is interrupted by more than one column, the stem contains an internal loop (Figure 4.8b). A hairpin of the RNA secondary structure is encoded as the free columns between a stem and the main diagonal of the dot plot (Figure 4.8c). Multi-loops are harder to detect in the dot plot but they usually occur with larger gaps between the stems (Figure 4.8d).

The upper triangle of the dot plot represents the probabilities of the consensus folding of several suboptimal folding predictions. Therefore, it is possible that a

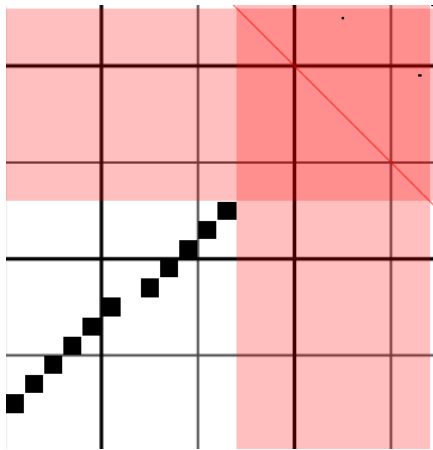




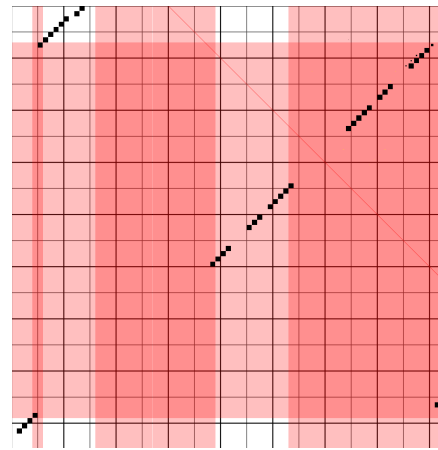
(a) The bulge annotated with '1' in Figure 4.9.



(b) The internal loop annotated with '2' in Figure 4.9.



(c) The hairpin loop annotated with '3' in Figure 4.9.



(d) The multi-loop annotated with '4' in Figure 4.9.

Figure 4.8: Representation of a bulge, an internal loop, a hairpin loop, and a multi-loop in iDotter. The highlighted regions annotate the structural elements.

single nucleotide has multiple base pairing options represented by several dots in the same row or column. If the size of these dots deviates considerably (as shown in Figure 4.4), the alternative folding suggestions are rather unlikely. If the dot sizes are of similar size, then the consensus probabilities show other similar base pairings for the sequence and the RNA secondary structure might have additional possible foldings that can occur, e.g., depending on the temperature or on the interaction with other molecules.

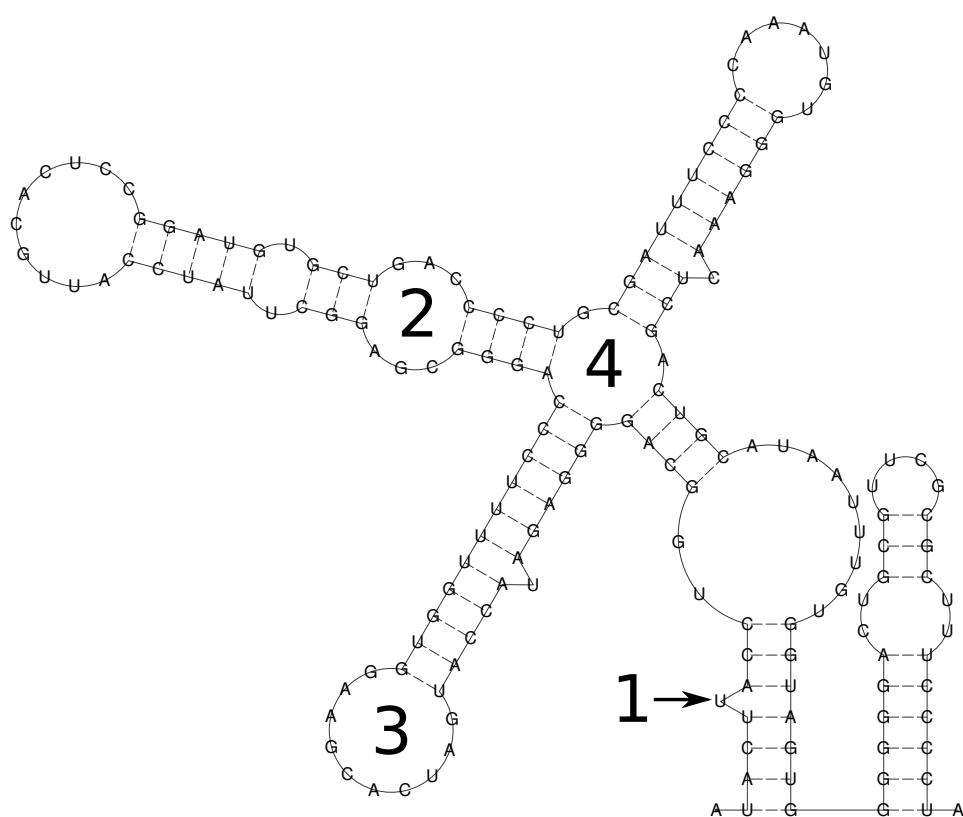
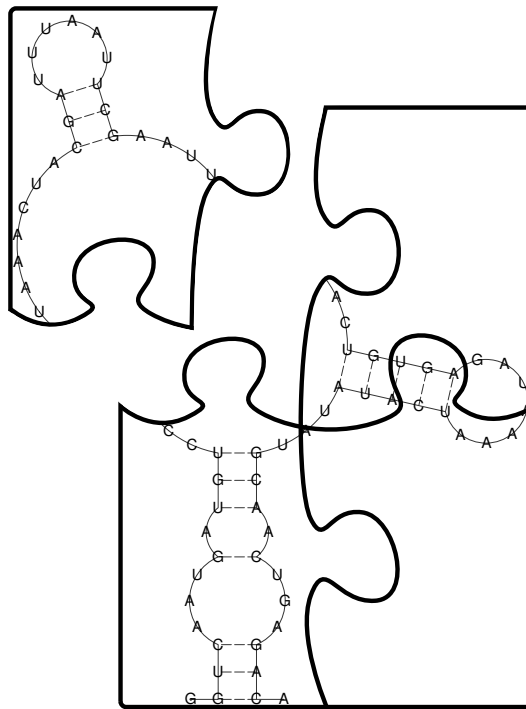


Figure 4.9: The predicted energetically best solution for the RNA shown in Figure 4.2 lower left visualized as graph using RNApuzzler. '1' annotates a bulge, '2' annotates an internal loop, '3' annotates a hairpin loop, and '4' annotates a multi-loop.

# Chapter 5

## RNApuzzler



## 5.1 Introduction

Besides the dot plots described in Chapter 4, RNA secondary structures can be visualized as graphs. Using this representation, the expert can more easily identify structural elements such as loops and stems and analyze them for determining possible binding sites or other interactions. Therefore, it is very common to visualize RNA secondary structures using a graph layout.

RNA secondary structures are outerplanar graphs, hence they can be drawn in the plane without intersection following the contour of a tree. In order to facilitate the interpretation by the expert, a meaningful drawing has to conform—at least approximately—to a series of constraints, such as parallel arrangement of stacked base pairs, or circular layout of loops. These additional requirements make RNA drawing difficult.

The current implementation of ViennaRNA uses a default layout algorithm proposed three decades ago by Brucoleri and Heinrich [22], which does not always produce a planar (i.e., self-intersection-free) drawing. Many other RNA drawing algorithms suffer from the same shortcoming [64, 74] because the desirable properties of the RNA secondary structure layout are incompatible with a planar drawing. We refer to the recent review by Ponty and Leclerc [64] for an overview of available programs, their capabilities, and their output.

RNApuzzler approaches the problem of drawing RNA secondary structure in two steps. First, a drawing method was devised that fulfills drawing constraints with respect to nucleotide distances (RNAturtle). Then, relaxations of these drawing constraints in a manner that guarantees an intersection-free layout were investigated. RNApuzzler is designed along the guidelines of Muller et al. [59]:

- *Simplicity*: The elements of the RNA secondary structure as well as the start and the end of the sequence should be clearly and easily recognizable.
- *Robustness*: The RNA drawing should be robust against small changes within the folding. This supports the experts comparing RNAs by identifying similar regions as well as differences between the RNAs.
- *Automation*: The (planar) RNA drawing should be created automatically.
- *Aesthetics*: The RNA drawing should have an aesthetic character.

## 5.2 Theory

This section describes the graph-theoretical basis of the algorithm (Section 5.2.1), the formal definition of the RNA secondary structure elements (Section 5.2.2), and the combination of both for defining *RNA-Trees* (Section 5.2.3).

### 5.2.1 Directed Rooted Trees

The following paragraphs define directed rooted trees and all related concepts that are needed subsequently. A general overview over graphs, trees, and networks as well as over drawing algorithms for them is given in the Handbook of Graph Drawing and Visualization [78].

**Definition 1** (Directed Graph). A directed graph  $G = (V, E)$  consists of a finite set of vertices  $V$  and a finite set of edges  $E \subseteq V \times V$  such that each edge  $e \in E$  is a tuple of vertices, i.e.,  $E \subseteq V \times V$ . One can say that  $e = (u, v)$  is an outgoing edge of  $u$  and an incoming edge of  $v$ .

**Definition 2** (In-degree and out-degree). Let  $v \in V$  be a vertex. The in-degree of  $v$  is the number of incoming edges of  $v$   $|\{u | (u, v) \in E\}|$ , while the out-degree of  $v$  is the number of outgoing edges of  $v$   $|\{w | (v, w) \in E\}|$ .

**Definition 3** (Directed Path). A directed path  $p$  is a sequence of  $l$  vertices  $(v_1, v_2, \dots, v_l)$ ,  $l \geq 2$  such that

- (1)  $\forall 1 \leq i \leq l : v_i \in V$  and  $\forall 1 \leq i \leq l - 1 : (v_i, v_{i+1}) \in E$  and
- (2) the  $v_i$  are pairwise distinct with the possible exception of the end points  $v_1$  and  $v_l$ . The length of the path is  $l - 1$ .

**Definition 4** (Cycle). A cycle is a directed path with  $v_l = v_1$ .

**Definition 5** (Directed Rooted Tree). A directed rooted tree  $T = (V, E)$  is a directed graph with the following properties:

- $T$  is weakly connected.
- $T$  does not contain cycles.
- There is a dedicated vertex named root vertex with in-degree zero. It is the only vertex with in-degree zero.
- There are dedicated vertices named leafs with out-degree zero.

- Let  $|V| \geq 2$ . For each leaf, there is exactly one path from the root vertex to the leaf.
- All other vertices are called internal vertices.

This definition implies that for each internal vertex  $v$ , there is a directed path from the root to  $v$  and from  $v$  to at least one leaf.

**Corollary 1** (Directed Rooted Tree: Properties). *Let  $T = (V, E)$  be a directed rooted tree.*

- All vertices of  $T$  except the root vertex have in-degree 1.
- $T$  has at least one leaf.
- If  $|V| = 1$  then the root vertex is also a leaf vertex.

**Definition 6** (Subtree). *Let  $T = (V, E)$  be a directed rooted tree and  $v \in V$  be a vertex. Then, the subtree  $T' = (V', E')$  rooted at  $v$  is defined as*

- $V' \subseteq V$
- $v \in V'$
- $E' \subseteq E \cap (V' \times V')$
- For all paths  $p = (v = v_1, \dots, v_l)$ :  
 $(v_i \in V' \forall 1 \leq i \leq l) \wedge$   
 $((v_i, v_{i+1}) \in E' \forall 1 \leq i \leq l - 1)$

**Definition 7** (Relationships). *Let  $T = (V, E)$  be a directed rooted tree and  $v \in V$  be a vertex.*

- If an edge  $(u, v) \in E$  exists, then  $u$  is called the parent of  $v$ .
- For all vertices  $w$  such that  $(v, w) \in E$ ,  $w$  is called a child of  $v$ .
- If  $w', w'' \in V$ ,  $w' \neq w''$  are children of the same vertex  $v$ , then they are called siblings.
- For all paths  $p = (u, \dots, v)$ ,  $u \in V$ ,  $u$  is called ancestor of  $v$ .
- For all paths  $p = (v, \dots, w)$ ,  $w \in V$ ,  $w$  is called descendant of  $v$ .

**Corollary 2** (Relationships: Properties). *Let  $T = (V, E)$  be a directed rooted tree.*

- *Each vertex  $v \in V$  except the root has exactly one parent.*
- *All vertices  $v \in V$  except the leaves have at least one child.*

**Definition 8** (Lowest Common Ancestor). *Let  $u, v \in V$ ,  $u \neq v$ . Further,  $u$  is neither an ancestor nor a descendant of  $v$ .*

- *A common ancestor  $ca(u, v)$  is defined as being a vertex that is an ancestor of both  $u$  and  $v$ .*
- *The lowest common ancestor  $lca(u, v)$  is defined as being the common ancestor of  $u$  and  $v$  such that the path  $p_u = (lca(u, v), \dots, u)$  is the shortest path for all common ancestors of  $u$  and  $v$ .*

**Corollary 3** (Lowest Common Ancestor). *Let  $p_u = (lca(u, v), \dots, u)$  and  $p_v = (lca(u, v), \dots, v)$ . Then their lengths  $|p_u|$  and  $|p_v|$  are the shortest ones among all paths from a common ancestor to  $u$  and  $v$ , respectively.*

## 5.2.2 RNA Secondary Structure

An *RNA sequence* is a string over the alphabet of nucleotides  $\mathcal{A} := \{A, U, G, C\}$ .

**Definition 9** (Secondary Structure). *A secondary structure on the vertex set  $X = \{1, 2, \dots, n\}$  is the disjoint union of the backbone  $B$ , i.e., the path  $1 - 2 - \dots - n$  and a set of base pairs  $\Omega$  with the following properties:*

- (1)  *$\Omega$  is a matching, i.e., for every  $x \in X$  there is at most one base pair  $b \in \Omega$  with  $x \in b$ .*
- (2)  *$\Omega$  is non-crossing, i.e.,  $i < k < j$  implies  $i < l < j$  for all  $\{i, j\}, \{k, l\} \in \Omega$ .*
- (3)  *$|j - i| > 3$  for all  $\{i, j\} \in \Omega$ .*

An secondary structure is compatible with an RNA sequence if  $\{i, j\} \in \Omega$  implies that the nucleotides  $x_i$  and  $x_j$  form one of the six allowed base pairs  $A - U$ ,  $U - A$ ,  $G - C$ ,  $C - G$ ,  $G - U$ , or  $U - G$ .

The graph  $(X, B \cup \Omega)$  of a secondary structure is outerplanar, i.e., it can be drawn in the plane in such a way that all vertices are incident to the infinite outer face and boundaries between finite faces are formed by base pairs. This outerplanar embedding is unique [52]. It gives rise to a unique tree representation of the secondary structure graph as follows (Figure 5.1b):

1. The vertices of the fully resolved tree are the faces of the outerplanar embedding.
2. The infinite outer face corresponds to the root of the tree.
3. An edge connects two vertices of this tree if and only if the corresponding faces share a base pair. Please note, that this construction differs from the usual notion of the *dual graph* by omitting edges between the infinite face and any finite face that does not share a base pair with the infinite face. It follows immediately from outerplanarity that this restricted dual graph is indeed a tree.

For the purpose of drawing the RNA secondary structure, faces are categorized (Figure 5.1a). The corresponding geometric representation of each face or each set of faces is shown in Figure 5.1d.

The first characteristic used is the number of outgoing edges. If the number of outgoing edges is 0, then the face is bounded by a single base pair and a sequence of consecutive unpaired bases belonging to the backbone. This face is called a *hairpin-loop* (H). Hairpin-loops will be drawn using a circle whose radius is determined by number of unpaired nucleotides and the backbone distance together with the base pair distance.

If the number of outgoing edges is  $\geq 2$ , then the face is called a *multi-loop* (M). Multi-loops will also be drawn as circles. Multi-loops will be characterized by their radii and by the angles of the outgoing base pairs compared to the incoming base pair. Changing these parameters allows to create outerplanar drawings.

If the number of outgoing edges is exactly 1, an additional characteristic for categorizing the corresponding faces is used. If the boundary of the face does not contain any unpaired nucleotides, two base pairs are directly connected to each other. All adjacent faces having these properties are connected and called *stem*. The faces will be represented by a rectangle with the two parallel paired nucleotides lying on edges of this rectangle. The stem is then formed by stacking this rectangle on top of each other. If the boundary of the face contains exactly one unpaired nucleotide, then this face is called a *bulge-loop* or simply *bulge* (B). Bulges and stems will be connected into one structure such that all lines connecting base pairs are parallel to each other. Bulges are represented by triangles formed by a sequence of three nucleotides (paired–unpaired–paired) attached to one side of the stem. Two edges are formed by



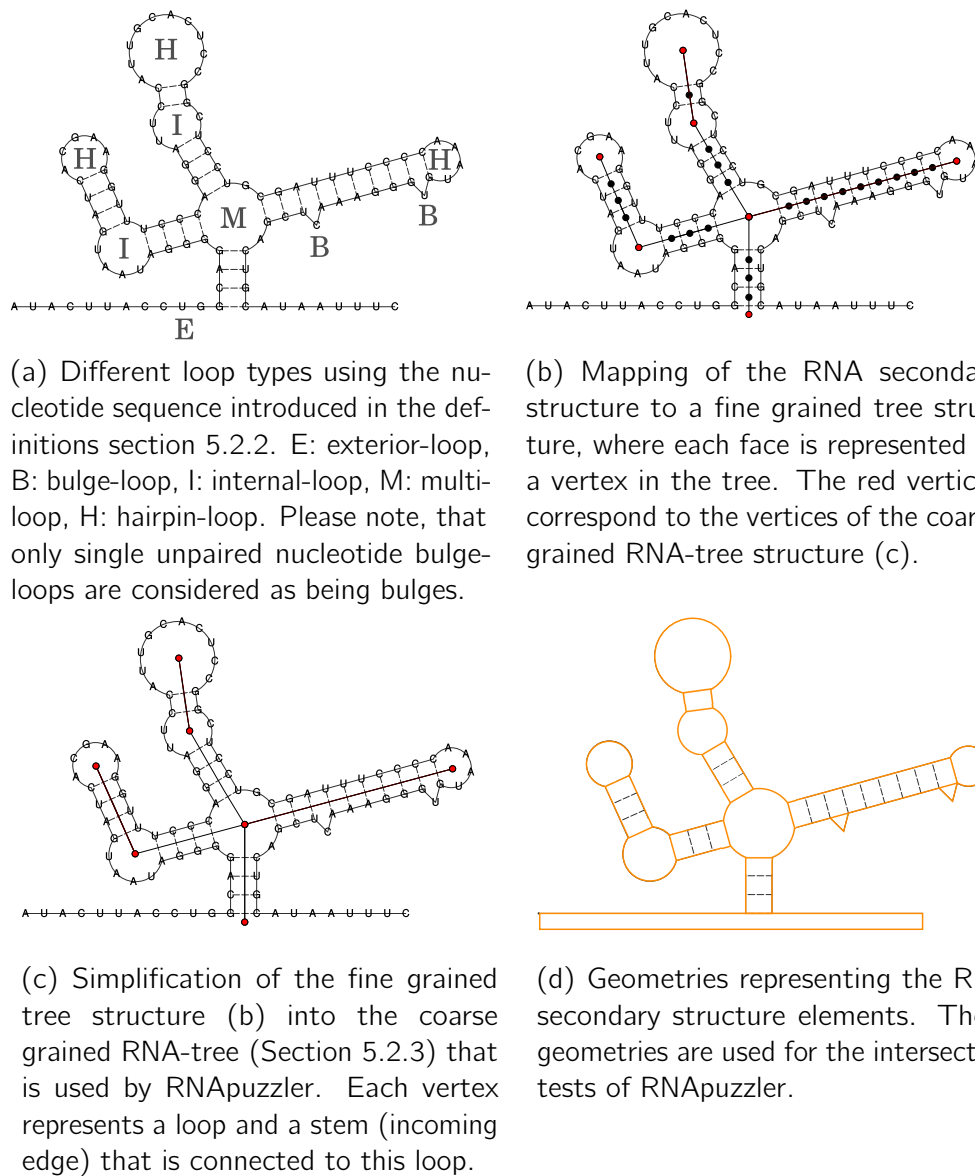


Figure 5.1: Different elements of the RNA secondary structure (a) and the corresponding fine grained (b) and coarse grained (c) trees as well as the corresponding geometries (d).

two base pairs connecting the stem to a loop. Stems, including intervening bulges, are considered as rigid objects for the purpose of drawing.

All other faces with exactly 1 outgoing edge are called *internal-loop* (l). Their boundary is formed by the two base pairs and at least 2 unpaired nucleotides. In the following no distinction is made between internal-loops with one or two sequences of consecutive nucleotides. It should be noted that the distinction of bulge and internal-loop is different from the one in the standard energy model [79], where all internal-loops with a single contiguous interval of unpaired positions are termed “bulge”.

Finally, all nucleotides not bounding faces form the *exterior-loop* together with the base pairs that are neighbors to some of them with respect to the backbone structure. The root of the tree is associated with the exterior-loop.

Representing each face by a vertex of the tree, a fine grained tree representation is obtained (Figure 5.1b). Similar to coarse tree models used in other contexts [43], it is useful to retain only those vertices that belong to hairpin-loops, internal-loops, and multi-loops as well as to the root of the tree. Altogether, they determine the topological structure of the tree (Figure 5.1c).

All remaining bounded faces are coalesced into the stem structure. As each of the loops described before has exactly one incoming base pair, the stem containing this base pair is associated with the respective loop. This stem–loop pair is called a *node* and is associated with its tree vertex.

### 5.2.3 RNA-Trees

**Definition 10** (RNA-Tree). *The RNA secondary structure can be represented by a rooted directed tree  $T = (V, E)$  called RNA-tree having the following properties:*

- *The exterior-loop is represented by a node. This node has no incoming edges and is the root vertex of the tree.*
- *All nodes constructed by combining stems and the associated loops are vertices of the tree.*
- *There is an edge  $(u, v) \in E$  between two vertices  $u, v \in V$  iff the stem of  $v$  is connected to the loop of  $u$  in the RNA secondary structure.*

**Corollary 4** (RNA-Tree). *Let  $T = (V, E)$  be an RNA-tree.*

- *Each leaf of the tree contains exactly one hairpin-loop.*
- *Each node containing an internal-loop has exactly one child.*
- *Each node containing a multi-loop has two or more children.*

Every RNA secondary structure allows for a outerplanar drawing [64]. In the following the RNApuzzler algorithm (Section 5.4) is used to construct such a planar drawing that builds upon the RNAturtle algorithm (Section 5.3) and resolves intersections in the layout generated by the latter. Therefore, it necessary to characterize intersections in the RNA-tree. Let  $T = (V, E)$  be an RNA-tree.

**Definition 11** (Intersection between two nodes). *Let  $u, v \in V$  be two nodes of the RNA-tree.  $u$  and  $v$  intersect each other, if the geometries represented by  $u$  and  $v$  intersect.*

**Definition 12** (Ancestor Intersection). *Let  $u, v \in V$  be two nodes of the RNA-tree such that  $u$  is an ancestor of  $v$ . If  $u$  and  $v$  intersect each other, then*

- *the intersection is called an ancestor intersection*
- *$u$  is called intersected node*
- *$v$  is called intersecting node*

*The expression used in the following is:  $v$  causes an ancestor intersection.*

**Definition 13** (Exterior Intersection). *Let  $r, v \in V$ ,  $r$  being the root node of  $T$ . An ancestor intersection between  $r$  and  $v$  is called exterior intersection.*

**Definition 14** (Sibling Intersection). *Let  $u, v', v'' \in V$ ,  $v' \neq v''$  being children of  $u$ . Let further  $T_{v'} = (V_{v'}, E_{v'})$  be the subtree of  $T$  with root  $v'$  and  $T_{v''} = (V_{v''}, E_{v''})$  be the subtree of  $T$  with root  $v''$ . If there exist nodes  $w' \in V_{v'}$  and  $w'' \in V_{v''}$  such that  $w'$  and  $w''$  intersect each other, the intersection is called a sibling intersection and one can say that “ $u$  has a sibling intersection”.*

**Corollary 5** (Sibling Intersection). *If  $u$  has a sibling intersection and  $w', w''$  are the intersecting nodes given in the previous definition, then  $u = \text{lca}(w', w'')$ .*

**Definition 15** (Exterior Subtree Intersection). *Let  $r \in V$  be the root node of  $T$ . If  $r$  has a sibling intersection, then the intersection is called an exterior subtree intersection.*

Based on these definitions of the different intersection types that will be considered, the absence of intersections is defined as:

**Definition 16** (A-planar). *A node  $v \in V$  is called A-planar iff the parent node  $u$  of  $v$  is A-planar and  $v$  does not cause any ancestor intersection.*

**Definition 17** (S-planar). *A node  $v \in V$  is called S-planar if  $v$  does not have a sibling intersection.*

**Definition 18** (T-planar). *Let  $v \in V$  be the root of the subtree  $T_v = (V_v, E_v)$  of  $T$ .  $v$  is called T-planar iff  $\forall w \in V_v : w$  is A-planar and S-planar.*

**Definition 19** (Planar Tree). *Let  $r \in V$  be the root of  $T$ .  $T$  is called a planar tree iff  $r$  is T-planar.*

These definitions are the foundation of the subsequent conclusions. Altogether, the definitions and the conclusions build the theoretical background for the RNApuzzler algorithm and for showing that the RNApuzzler algorithm computes a planar layout of the RNA-Tree and thus of the RNA secondary structure.

Next, conclusions will be drawn based on the definition of A-planar (Definition 16).

**Lemma 1.** *The root  $r \in V$  of  $T$  is A-planar.*

*Proof.* The statement follows trivially since  $r$  has no ancestor.  $\square$

**Lemma 2.** *If  $v \in V$  is A-planar and  $u \in V$  is an ancestor of  $v$  then  $u$  is A-planar.*

*Proof.* Every ancestor  $u$  of  $v$  lies on the (unique) directed path from the root to  $v$ . Proceed by induction on the length  $l$  of the path from  $r$  to  $v$ . For  $l = 1$ , the only ancestor is  $r$ , which is A-planar by Lemma 1. In the general case assume that the statement is true for all ancestors of  $u$  of  $v$  with a path of length  $l$  between  $r$  and  $v$ . Now, let  $w \in V$  be a child of  $v$ . Thus, the path from  $r$  to  $w$  has length  $l + 1$ . The ancestors of  $w$  are its parent  $v$  as well as all ancestors of  $v$ . The induction hypothesis implies that all ancestors of  $v$  are A-planar. By definition, the fact that  $w$  is A-planar implies that its parent  $v$  is also A-planar. Therefore all ancestors of  $w$  are A-planar.  $\square$

**Lemma 3.** *If  $v \in V$  is A-planar and  $u \in V$  is an ancestor of  $v$  then  $u$  does not cause any ancestor intersection.*

*Proof.* This follows directly from Lemma 2 and the definition of A-planar.  $\square$

**Lemma 4.** *If neither  $v \in V$  nor any ancestor  $u \in V$  of  $v$  causes an ancestor intersection, then  $v$  is A-planar.*

*Proof.* Every ancestor  $u$  of  $v$  lies on the (unique) directed path from the root to  $v$ . Proceed by induction on the length  $l$  of the path from  $r$  to  $v$ . For  $l = 1$ , the only ancestor is  $r$ , which is A-planar by Lemma 1. By assumption,  $v$  does not cause an ancestor intersection and thus by definition  $v$  is A-planar. In the general case assume that the statement is true for all nodes  $v$  with a path of length  $l$  between  $r$  and  $v$ . Now, let  $w \in V$  be a child of  $v$ . Then, the path from  $r$  to  $w$  has length  $l + 1$ . The ancestors of  $w$  are its parent  $v$  as well as all ancestors of  $v$ . The induction hypothesis implies that  $v$  and all its ancestors are A-planar. As by assumption  $w$  does not cause an ancestor intersection, it is A-planar by definition.  $\square$

**Theorem 1** (A-planar). *A node  $v \in V$  of  $T$  is A-planar if and only if neither  $v$  nor any ancestor  $u \in V$  of  $v$  causes an ancestor intersection.*

*Proof.* One can show the equivalence by showing each implication:

“ $\Rightarrow$ ”: Let  $v$  be A-planar. Then neither  $v$  nor any ancestor  $u \in V$  of  $v$  causes an ancestor intersection by Lemma 3.

“ $\Leftarrow$ ”: Neither  $v$  nor any ancestor  $u \in V$  of  $v$  causes an ancestor intersection. Then  $v$  is A-planar by Lemma 4.  $\square$

In the next step, it is shown that leafs are always S-planar (Definition 17).

**Lemma 5.** *Every leaf  $v \in V$  of  $T$  is S-planar.*

*Proof.* This follows trivially from the definition since leaves have no children.  $\square$

Further, if a node is T-planar, all descendants are T-planar, too. Moreover, if all children are T-planar and the node itself is both A-planar and S-planar, the node is T-planar, too.

**Lemma 6.** *Let  $v \in V$  be the root of the subtree  $T_v = (V_v, E_v)$  of  $T$ . If  $v$  is T-planar, then  $w$  is T-planar for all  $w \in V_v$ .*

*Proof.* Let  $w \in V_v$  be a descendant of  $v$ . By Definition 18,  $w$  is A-planar and S-planar. Moreover, all descendants of  $w$  are A-planar and S-planar as they are descendants of  $v$ , too. Let  $T_w = (V_w, E_w)$  be the subtree rooted at  $w$ . As  $V_w$  contains  $w$  and all its descendants, all  $w' \in V_w$  are A-planar and S-planar. Therefore,  $w$  is T-planar by definition.  $\square$

**Lemma 7.** *Let  $v \in V$  be the root of the subtree  $T_v = (V_v, E_v)$  of  $T$ .  $v$  is  $T$ -planar, if all children  $w \in V_v$  of  $v$  are  $T$ -planar and  $v$  is  $A$ -planar and  $S$ -planar.*

*Proof.*  $V_v$  contains  $v$  and all descendants of  $v$ . By assumption, all children  $w$  of  $v$  are  $T$ -planar. Therefore, all children  $w$  and their descendants are  $A$ -planar and  $S$ -planar by Definition 18. Together, these are all descendants of  $v$ . Thus, all descendants of  $v$  are  $A$ -planar and  $S$ -planar. As by assumption,  $v$  is  $A$ -planar and  $S$ -planar, too,  $v$  is  $T$ -planar by Definition 18.  $\square$

Some important consequences are given next.

**Corollary 6.** *Let  $v \in V$  be a leaf. If  $v$  is  $A$ -planar, then it is  $T$ -planar.*

*Proof.*  $v$  is  $A$ -planar by assumption and  $S$ -planar according to Lemma 5. Moreover,  $T_v = (N_v, E_v) = (\{v\}, \emptyset)$  is the subtree of  $T$  rooted at  $v$  as  $v$  is a leaf and thus has no descendants. Thus, all  $v \in N_v$  are  $A$ -planar and  $S$ -planar and thus  $v$  is  $T$ -planar by Lemma 7.  $\square$

**Corollary 7.** *Let  $r \in V$  be the root of  $T$ . Then,  $r$  is  $T$ -planar, if  $r$  is  $S$ -planar and  $\forall w \in V : w \text{ is a child of } r \rightarrow w \text{ is } T\text{-planar}$ .*

*Proof.*  $r$  is  $A$ -planar according to Lemma 1 and  $S$ -planar by assumption. Moreover, all children  $w$  of  $r$  are  $T$ -planar by assumption. Thus,  $r$  is  $T$ -planar by Lemma 7.  $\square$

## 5.3 Method RNaTurtle

### 5.3.1 Drawing Constraints

The initial drawing is created satisfying the following constraints:

1. The drawing starts at a defined position.
2. The exterior nucleotides are drawn on a straight horizontal line (increasing x-value).
3. Stems attached to the exterior loop point upwards only (increasing y-value). These stems are perpendicular to the horizontal straight line defined by the exterior nucleotides forming the exterior loop.
4. The distance between two neighboring exterior nucleotides is constant (backbone distance) [64].
5. The distance between two neighboring nucleotides of a stem is constant [64]. It is the same as that of two neighboring exterior nucleotides (backbone distance).
6. The distance between two neighboring nucleotides of a loop is constant [64]. However, the arc-length between two neighboring nucleotides of a loop depends on the loop radius and thus is constant for a specific loop, but might differ for different loops.
7. The distance between two paired nucleotides of a stem is constant (base pair distance) [64].
8. Stems should be drawn as rectangles [19].
9. Loops should be drawn as circles [19].
10. Loops should be drawn as compact as possible [59].
11. Bulges should not change stem direction [84].

All of these constraints can be satisfied simultaneously. However, the resulting drawing will in general not be intersection-free.

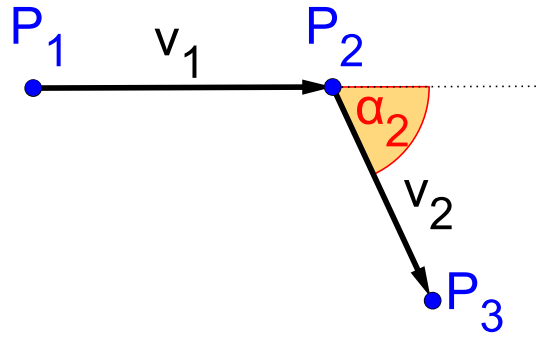


Figure 5.2: The turtle starts at  $P_1$  and walks a fixed distance to  $P_2$ . There, it changes its direction by an angle  $\alpha_2$  and walks a fixed distance to  $P_3$ .

### 5.3.2 Turtle Algorithm

The initial drawing is created using a so-called turtle graphics algorithm (Algorithm 6) imagine a turtle walking and leaving a trail. First, the starting point of the turtle is specified and the first nucleotide of the RNA-sequence is positioned at this point. Then, two actions are possible by the turtle: *change direction* and *walk a fixed distance*. Change direction instructs the turtle to change its direction by an angle between  $-180^\circ$  and  $180^\circ$ , exclusive. Walk a fixed distance instructs the turtle to walk a fixed distance in its current direction.

The turtle is initially put onto a pre-defined position (Constraint 1). The standard angle is  $0^\circ$  and the standard distance is equal to the backbone distance. In the beginning, the turtle with  $0^\circ$  would walk along the increasing x-axis (Constraint 2). Thus, the turtle moves forward the distance between two neighboring nucleotides in the RNA-sequence satisfying Constraints 4 and 5 (Figure 5.2).

At each point reached by the turtle, the direction might be changed once before walking. Here, different cases have to be distinguished. In the case that the turtle is on the exterior loop and reaches a nucleotide belonging to a stem, the angle is set to  $-90^\circ$  changing the direction from going to the right to going up (Constraint 3). In the case that the turtle is on a stem and reaches a nucleotide belonging to the exterior loop, the angle is set to  $-90^\circ$  changing the direction from going down to going to the right (Constraint 3). If the current and the next nucleotide belong to the same stem, the angle is set to the default angle of  $0^\circ$ . Bulges are also easily drawn by changing the angle from stem to unpaired nucleotide to  $-120^\circ$  and making one step forward, followed by setting the angle to  $60^\circ$  and making another step forward, followed by setting the angle to  $-120^\circ$  and making a final step forward. Thus, bulges neither bend the



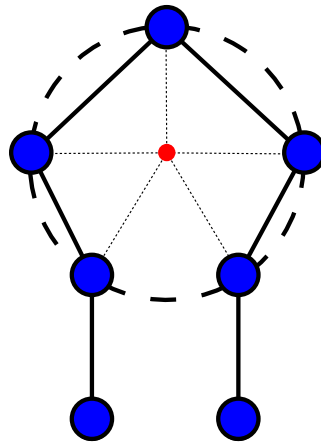


Figure 5.3: The turtle walks through a loop: First, the radius of the loop is approximated. Then, the turtle changes its direction based on the arc segment that the current and the next position form in the loop. Please note, that the turtle walks on the chords of the loop. The arcs are drawn during a post-processing step.

corresponding stem nor do they cause the backbone distances of the involved nucleotides to be modified (Constraint 11). Thus far, the computation is simple and straightforward.

The situation becomes slightly more involved whenever a loop other than the exterior loop is connected to a stem. Then, the calculation of the angle depends on the radius of the loop and thus on its structure. A loop consists of multiple free nucleotides whose distance from each other and to neighboring paired nucleotides equals the backbone distance. Further, two stems of a loop that are next to each other, i.e., that are not separated by an unpaired nucleotide, are separated by the backbone distance, too. On the other hand, the distance between two paired nucleotides is equal to the base pair distance. As it is not possible to calculate the exact radius if the backbone and the base pair distance are different, the radius of the loop is approximated using the *Newton Raphson* method. The upper and lower bounds of the radius are set to the radius of a loop having the same number of loop elements but using only the backbone or the base pair distance since the real radius of the loop lies within this interval. After calculating an appropriate radius for the loop, the nucleotides are positioned by the turtle algorithm onto the loop by calculating the correct angles (Figure 5.3).

Lonely base pairs have to be treated as a special case and require the computation of the sum of consecutive loop-stem and the stem-loop angles. They can also occur when such a lonely base pair connects the exterior nucleotides to

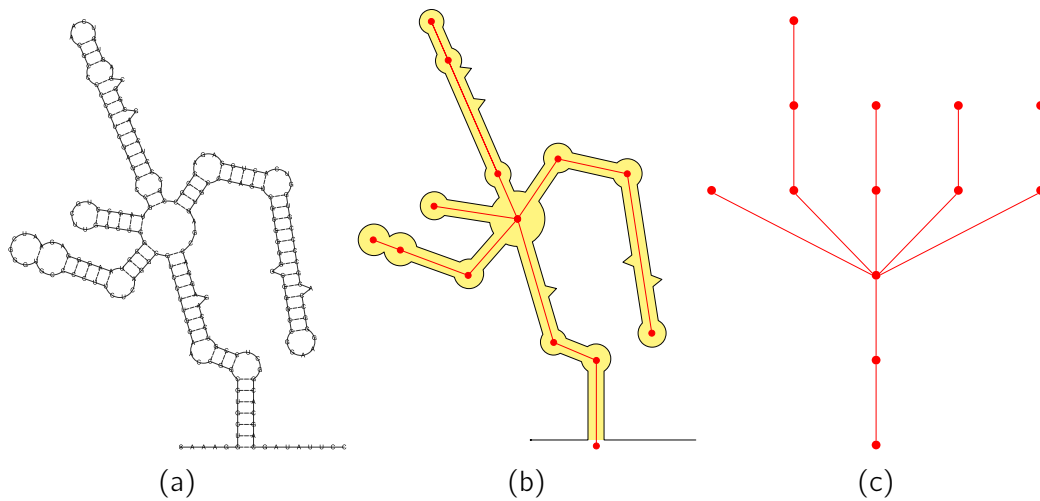


Figure 5.4: The RNAfold-predicted minimum free energy structure of the Magn\_3109 RNA of *Magnetospirillum magnetotacticum* sequence, retrieved from RFAM [38]. (a) Drawing generated by RNAturtle. (b) Tree structure superimposed on the RNAturtle drawing. (c) Extracted tree only.

a loop, i.e., the two nucleotides belong to all three structures at the same time. The resulting sum is taken as the angle for pointing the turtle to the correct direction.

This approach has the property that it fulfills all constraints. Drawings created by RNAturtle are shown in Figures 5.4a (planar) and 5.11a (not planar).

The turtle graphics approach makes it possible to reposition complete subgraphs by changing just the originating point or direction. This property will be used extensively in the next stage. During the initial drawing stage, the algorithm furthermore detects all structural segments of the RNA and stores it for the subsequent processing steps.

---

#### Algorithm 6 RNAturtle

---

```

for each nucleotide do
  check structural type of the nucleotide
  if start of a new loop then
    approximate radius
  end if
  calculate distance and angle to its predecessor based on the type
end for

```

---

## 5.4 Method RNApuzzler

### 5.4.1 Drawing Constraints

Drawings generated by RNAturtle may already be intersection-free (planar, Figure 5.4a). In particular for larger RNAs, however, this is usually not the case (Figure 5.11a). RNApuzzler starts from the RNAturtle output. The key idea is to replace Constraints 4 and 6 and add two new constraints:

- 4'. The distance between two neighboring exterior nucleotides is constant *between two stems* attached to the exterior loop only. It can be larger than the backbone distance.
- 6'. The distance between two unpaired neighboring nucleotides of a loop *segment* is constant. It can be larger than the backbone distance.
- 12'. The resulting drawing is intersection-free, i.e., planar [59].
- 13'. Between any pair of bases, there is a *minimum* distance [59].

Constraint 6' relaxes the requirement of a minimum loop radius with constant distance between paired and unpaired nucleotides of the loop, respectively. Thus, it makes it possible to change the directions of sub-sequences starting at a loop so that Constraint 12' can be satisfied for each subtree attached to the exterior loop. Allowing larger distances between the exterior nucleotides (Constraint 4') allows the algorithm to handle each subtree attached to the exterior loop individually and then to place the subtrees next to each other without intersection (Constraint 12').

Moreover, the drawing can be made more compact by relaxing Constraint 3.

- 3'. Stems attached to an exterior loop point upwards or downwards only (increasing or decreasing y-value). These stems are perpendicular to the horizontal straight line defined by the exterior loop.

Constraint 3' provides more flexibility for placing subtrees attached to the exterior loop. However, this might make it harder to locate the exterior loop in certain cases. In the same way, Constraints 6', 4', and 3' enable fulfilling Constraint 13' which prevents bases from overlapping.

**Algorithm 7** RNApuzzler Main Algorithm

---

```

for all nodes from the exterior to the leaves do
  detect ancestor intersections {Algorithm 8}
  if current node is a leaf then
    detect sibling intersections {Algorithm 11)}
  end if
end for

```

---

**5.4.2 Coordinate transformation and data structures**

Consider the secondary structure of the *Magnetospirillum magnetotacticum* Magn\_3109 RNA as computed with ViennaRNA, which is shown in Figure 5.4a. First, a tree structure is extracted from the secondary structure as follows: the tree nodes are the hairpin, internal, multi-branch, and external loops of the secondary structure; the tree edges correspond to stems (Figure 5.4b). This tree is rooted at the exterior loop. Hairpin loops therefore correspond to leaves (Figure 5.4c). Each stem is followed by exactly one loop, which will be associated with this stem. It should be noted that the base pair that joins the stem with its loop is the closing pair of the loop, which plays an important role in the RNA folding algorithms. One can then re-interpret the tree vertices as representing a stem and its associated loop. The edges thus no longer represent geometry. To distinguish between the tree structure and its associated geometry, the elements of a tree structure are called *vertices* and the combination of stem and loop geometry associated with a vertex is called the corresponding *node*.

**5.4.3 Main Algorithm**

The RNApuzzler uses the output of RNAturtle as input for producing the final layout. The major requirement for this final layout is planarity. This necessitates the removal of any intersection present in the layout produced by RNAturtle. The first step uses this layout for building the data structures, and especially the RNA-tree. Now, the theoretical foundations (Section 5.2) are used as the basis for the main algorithm (Algorithm 7).

Let  $T = (V, E)$  be an RNA-tree. By Definition 19,  $T$  is planar, if the root node  $r \in V$  is T-planar. By Corollary 7, it is sufficient that  $r$  is S-planar and all children of  $r$  are T-planar. T-planarity of a node  $v \in V$  requires A-planarity and S-planarity. As the root  $r$  is A-planar (Lemma 1) it is a good starting point for checking A-planarity. If a node  $v \in V$  is A-planar, then a child  $w \in V$  is either A-planar, too, or it intersects an ancestor. In the latter case, this intersection is

resolved by Algorithm 9. Thus, A-planarity is constructed on the way from the root node of the RNA-tree to its leaves for all nodes on that path recursively.

On the other hand, each leaf is S-planar (Lemma 5). Thus, on the way back from the leaves to the root, S-planarity can be checked and established for all nodes  $v \in V$ . Therefore, consider a node  $v \in V$  such that all its descendants are S-planar. Now,  $v$  is either S-planar itself in which case it is also T-planar by construction, or  $v$  has a sibling intersection that then is resolved.

While resolving both types of intersections, new intersections might be generated. However, the nodes that have to be reconsidered are restricted. In case of A-planarity, the algorithm continues with the node  $v \in V$  used for resolving the intersection and checks all nodes in its subtree. As the ancestors of  $v$  are not changed, none of them can cause new ancestor intersections. In case of S-planarity, the algorithm re-evaluates the current node  $v \in V$  and all its children. In this case, S-planarity for disjoint subtrees has been or will be established independently, while S-planarity of ancestor nodes will be established at a later step. Due to these repetitions in case of intersection removal, the complexity of the algorithm is difficult to assess.

However, it is easy to see that the depth-first traversal proposed is better suited than a breadth-first traversal. In the latter case, all nodes would be checked for A-planarity first. After all nodes are A-planar, S-planarity is checked and achieved. In both cases, if removing an intersection necessitates checking a subtree again, all nodes of the subtree will be considered. This leads to all nodes of these subtrees being checked for A-planarity several times—as often as they are re-assessed. During depth-first traversal, however, only those nodes already checked in the subtree are re-assessed while those not yet handled by the algorithm are treated at a later point, only. This reduces the overall amount of nodes to be checked.

Even if the complexity is difficult to estimate, the algorithm is *correct* and *terminating*:

Correctness refers to the fact that the resulting drawing satisfies the Constraints 1-3, 4', 5, 6', 7-11, 12', and 13'. Constraints 1-3 are met by RNAturtle. RNApuzzler also satisfies these constraints because it leaves the starting position, the orientation of the exterior loop, and the orientation of the stems attached to the exterior loop unchanged. Constraint 4' is trivially met as distances between exterior loop subtrees and thus distances between exterior nucleotides are only increased compared to the RNAturtle drawing. As RNApuzzler does not change the geometry of stems, Constraints 5 and 7 are still

met. RNApuzzler constructs the drawing such that Constraint 6' is met. The algorithm draws stems as rectangles (satisfying Constraint 8) and loops as circles (satisfying Constraint 9). Further, bulges do not change stem direction (satisfying Constraint 11). Constraint 12' is equivalent to the drawing being intersection-free. Section 5.2 provides a proof that this is always the case. Moreover, the algorithm guarantees minimal distances between any two bases satisfying Constraint 13'. The optimization performed (see Section 5.5) reduces the loop sizes as much as possible. Thus, Constraint 10 is met.

It is not obvious that RNApuzzler indeed terminates. The recursion terminates at the leaves and Algorithm 8 moves from the root to all leaves. Algorithm 9, however, involve repetitions. Algorithm 11 might cause all three steps to be applied repeatedly to the node including a depth-first traversal of the subtree. Algorithm 8 might even cause to repeat the whole process at any node between the root node and the current node. The reason why the recursion nevertheless terminates is due to the particular way in which intersections are solved.

This is achieved by changing the angles between the stems attached to a loop and by increasing the radius of the loop. Due to Constraint 6', it is necessary to increase the radius of a loop for changing the angle of certain loop segments, since the minimal distance between two unpaired nucleotides of a loop segment can not be smaller than the backbone distance. Increasing the loop radius automatically leads to a state where all siblings can be spaced such that they do not intersect any more. As illustrated in Figure 5.5, it is always possible to resolve the intersection by increasing the radius only. Since this would lead to vast radii, the angle between the two intersecting siblings is also increased. One might choose to increase only the angle between the siblings. This would make it necessary to compress the angles of all other siblings of the loop, which in turn would also compress the distances between the unpaired bases of these segments. Reducing them below the minimal distance, however, would violate Constraint 6'.

A transformation from polar to Cartesian coordinates shows that enlarging the radii is sufficient. In Cartesian coordinates, each subtree of the node has a certain height and a certain width. The sum of the widths of all subtrees plus some space between them is an upper bound on the width required to place all subtrees without intersections. A similar approach was used in [19], albeit without proof. The width in Cartesian coordinates required for intersection free placement of subtrees translates to a circumference in polar coordinates, from

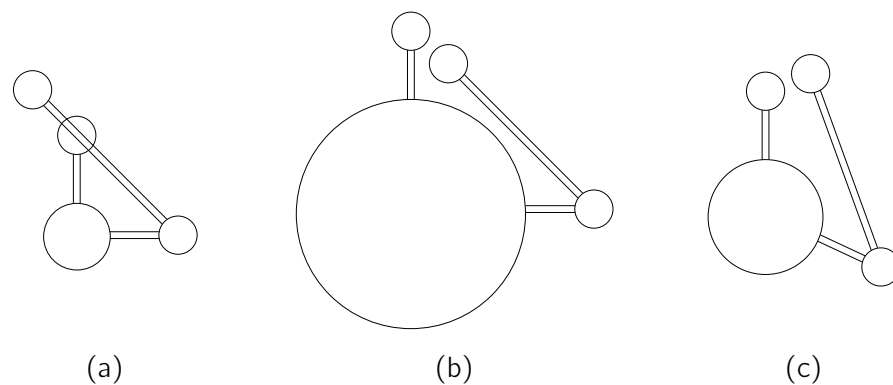


Figure 5.5: Any sibling intersection can be resolved (inefficiently) by only increasing the radius of the common multi-loop. Changing the angles of outgoing stems up to a certain degree to resolve intersections in addition to increasing the loop's radius improves the efficiency of the approach. (a) A sibling intersection between the two stems of the multi-loop. (b) The sibling intersection is resolved by only increasing the radius of the multi-loop. (c) The sibling intersection is resolved by changing the angles of the outgoing stems in addition to increasing the radius of the multi-loop.

which the appropriate radius can be computed. The same idea can be used to resolve all ancestor intersections.

### A-Planarity

The first step of the main algorithm checks for ancestor intersections (Detect Ancestor Intersections, Algorithm 8) and if necessary resolves them. If the ancestor found is the root node, then the special case of an exterior intersection is resolved (Resolve Exterior Intersections, Algorithm 10). Otherwise, the general algorithm for resolving ancestor intersections is used (Resolve Ancestor Intersections, Algorithm 9).

**Detect Ancestor Intersections** In principle there can be zero, one, or several intersections between the current node and one of its ancestors. If there are no intersections, then the current node is A-planar and no further action has to be performed. If there is exactly one intersection, this intersection is resolved. Otherwise, the algorithm resolves the intersection with the closest ancestor.

For this, the current node is checked recursively against its ancestors starting with its parent and ending with the root node (Figure 5.6). If there is an ancestor intersection (Figure 5.6a), the current node is the intersecting node (red) and the ancestor is the intersected node (blue).

**Algorithm 8** Detect ancestor intersection

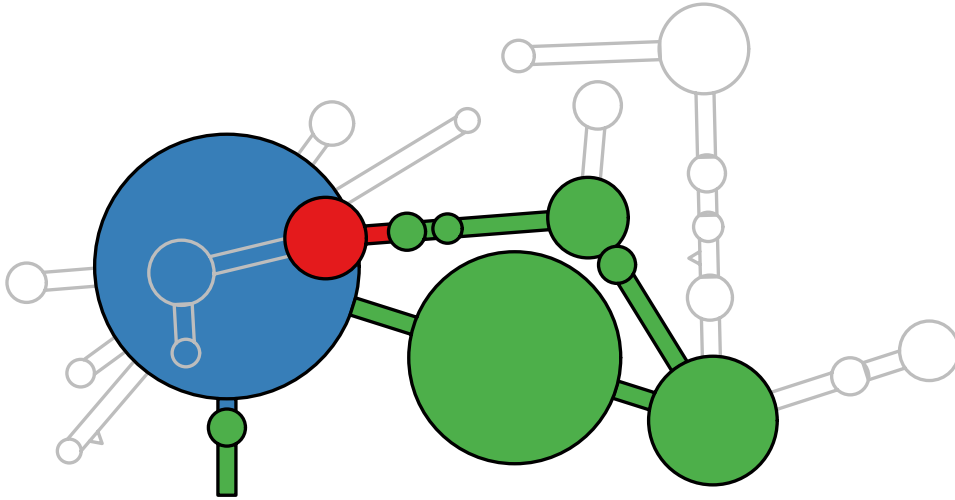
---

```

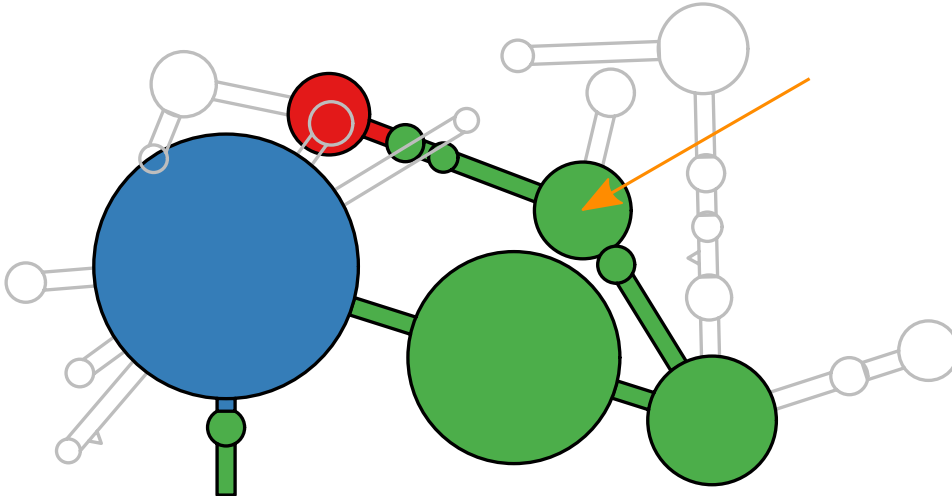
if node intersects with an ancestor then
  solve ancestor intersection
end if

```

---



(a) Intersection between the current node (red) and one of its ancestors (blue). The blue node and all green nodes do not intersect any of their ancestors.



(b) After resolving the intersection shown in (a), the algorithm proceeds with the green node marked with an orange arrow used for resolving the intersection.

Figure 5.6: Ancestor intersection (a) and its resolution (b).

**Resolve Ancestor Intersections** The ancestor intersection between the red intersecting node  $v_i$  and the blue intersected node  $v_a$  can be resolved using the loop marked with an orange arrow (Figure 5.6b). The latter is called *rotation loop* and belongs to the node  $v_r$  being used to rotate the intersecting node away from the intersected node. The principal solution consists of selecting a loop between the intersected and the intersecting node, and changing the



angle of the child  $v_c$  of the rotation loop  $v_r$  that is either the intersecting node  $v_i$  itself or the ancestor of the intersecting node. Thus, there exists a path  $p = (v_a, \dots, v_r, v_c, \dots, v_i)$  from  $v_a$  to  $v_i$  such that  $v_r \neq v_c$ . The nodes  $v_a$  and  $v_r$  as well as  $v_c$  and  $v_i$  might be identical. Therefore, the length of the path  $|p| \geq 2$ .

Algorithmically, three steps are performed. First, the direction of rotation is determined. Therefore, the path from the intersected node to the intersecting node is closed by adding an edge from the intersecting node to the intersected node. This sequence of nodes is intersection free except for the intersection found between the intersecting and the intersected node. Thus, it can be determined if the enclosed area lies to the left or to the right of the path. In the former case, the path is counter-clockwise and the rotation should be clockwise. In the latter case, the path is clockwise and the rotation should be counter-clockwise.

Second, the rotation node  $v_r$  is computed by searching backwards from the parent of the intersecting node  $v_i$  to the intersected node  $v_a$ . Thereby, the direction of the rotation is taken into account. Moreover, heuristics restrict the selection process. It should be noted that bulge-loops are fixed and hairpin-loops have no children. Moreover, the exterior-loop is handled separately. Therefore, only internal-loops and multi-loops are candidates for  $v_r$ .

The basis of the first set of heuristics is that internal-loops can be treated differently from multi-loops. Internal-loops are supposed to be optimal if they do not change the stem direction from incoming to outgoing stem [84]. Several algorithms even start by making all internal-loops straight [23]. Thus, the algorithm prefers internal-loops over multi-loops (*Heuristic 1*). Moreover, internal-loops can only be changed towards being straight (*Heuristic 2*). If an internal-loop is straight, it is optimal and thus can be safely ignored (*Heuristic 3*). These heuristics can be safely applied, as in case of an ancestor intersection and all internal-loops being straight, at least on multi-loop exists that can be chosen as rotation loop.

*Heuristic 4* chooses loops as close as possible to the intersecting node. This is due to the fact, that the rotation node is the starting point of the depth-first traversal after resolving the intersection. Thus, choosing it closer to the intersecting node reduces the number of nodes that need to be checked for ancestor intersections again. Moreover, the position of less nodes is changed and thus the probability of introducing new intersections is reduced.

Due to *Heuristic 1*, the algorithm for determining the rotation node performs two passes. During the first pass, only internal nodes are considered as possible rotation nodes. Hereby, *Heuristic 3* and *Heuristic 2* are applied. During the second pass, only multi-loops are considered as possible rotation nodes. To implement *Heuristic 4*, both passes start searching for the rotation node at the parent of the intersecting node towards the intersected node. The first suitable rotation node is taken.

Finally, the rotation angle  $\varphi_r$  is determined such that after rotating  $v_c$  by this angle, the intersection is resolved and the minimal distance constraint is satisfied. Rotating  $v_c$  at the loop of  $v_r$  by  $\varphi_r$  might necessitate increasing the radius of the rotation loop. As this can cause an ancestor intersection of  $v_r$ , the tree traversal algorithm goes back to  $v_r$ . As shown in Figure 5.6, the distance between the rotation loop and the loop of the green child of  $v_a$  is smaller after resolving the intersection (Figure 5.6b) than before (Figure 5.6a) due to the rotation loop's radius increase.

It is possible that  $v_c$  can not be rotated by the complete amount of  $\varphi_r$ . This might be the case, if the rotation loop  $v_r$  is an internal-loop. Then, it is necessary to change two or more different loops until the original intersection is resolved even if no new intersections are introduced. Therefore, RNApuzzler applies the heuristics again to resolve the intersections completely. This can lead to the case, where all internal-loops in the region of interest get straightened. After straightening the internal-loops, the remaining rotation angle  $\varphi_r$  is applied on a multi-loop. Since the algorithm can modify multi-loops with any rotation angle  $\varphi_r$ , it is possible to resolve any intersection by rotating the children of multi-loops.

---

**Algorithm 9** Resolve ancestor intersection
 

---

```

for each node from intersecting to intersected node do
  if node is an interior loop & bending possible then
    bend loop
    restart ancestor intersection at current node
  end if
end for
for each node from intersecting to intersected node do
  if node is a multi-loop then
    bend loop
    restart ancestor intersection at current node
  end if
end for

```

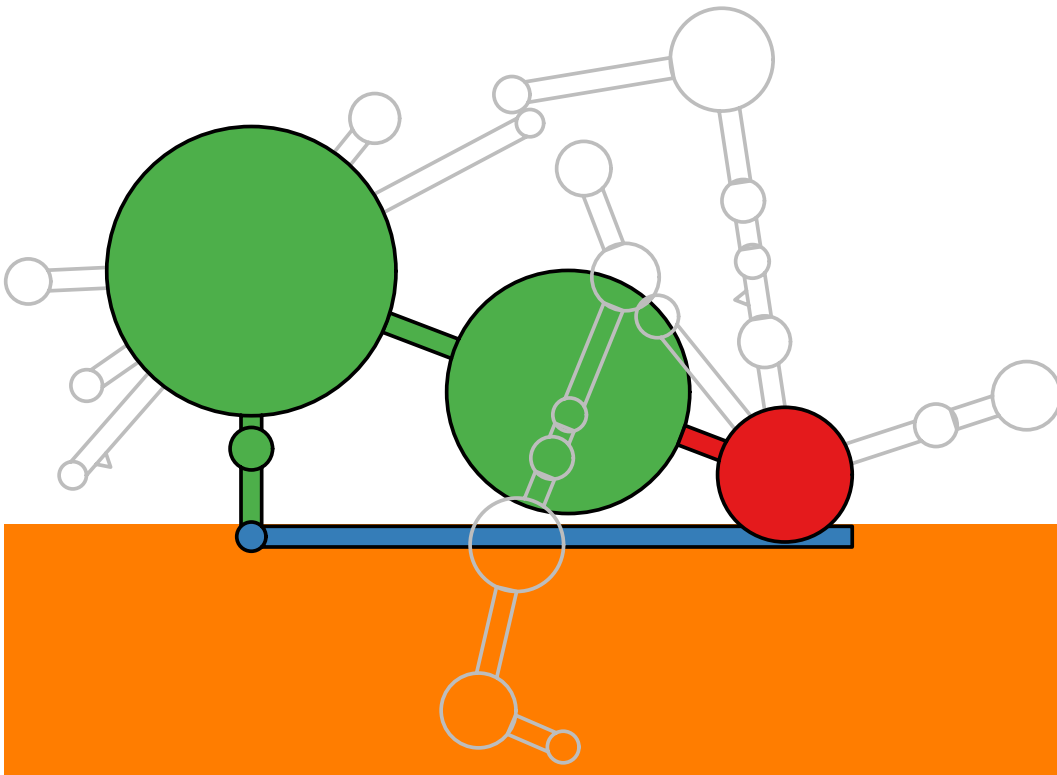
---

**Resolve Exterior Intersections** The root node is special as its geometry is that of a horizontal line. This implies that this line spans the interval  $[-\infty, \infty]$ . However, no additional special algorithms based on this line were developed for resolving exterior intersections. Instead, the algorithms for ancestor intersection resolution were reused. Therefore, a special node consisting of an artificial stem and an artificial loop is created. This node is called *root ancestor* (Figure 5.7, blue node). The loop of this node is constructed such that its center is directly below the center of the child stem. Its radius is computed from the base pair distance. The stem of the root ancestor has as width the base pair distance. Its orientation is parallel to the line formed by the exterior nucleotides. If possible, a single stem to the left or to the right of the loop is constructed. Its length is determined by the horizontal width of the axis-aligned bounding box of the intersecting node. If it is not possible to use a single stem only, two stems to the left and to the right of the loop are constructed and the one intersected first is used for resolving the exterior intersection. In fact, the intersecting node will always intersect the stem of the root ancestor before intersecting the loop of the root ancestor.

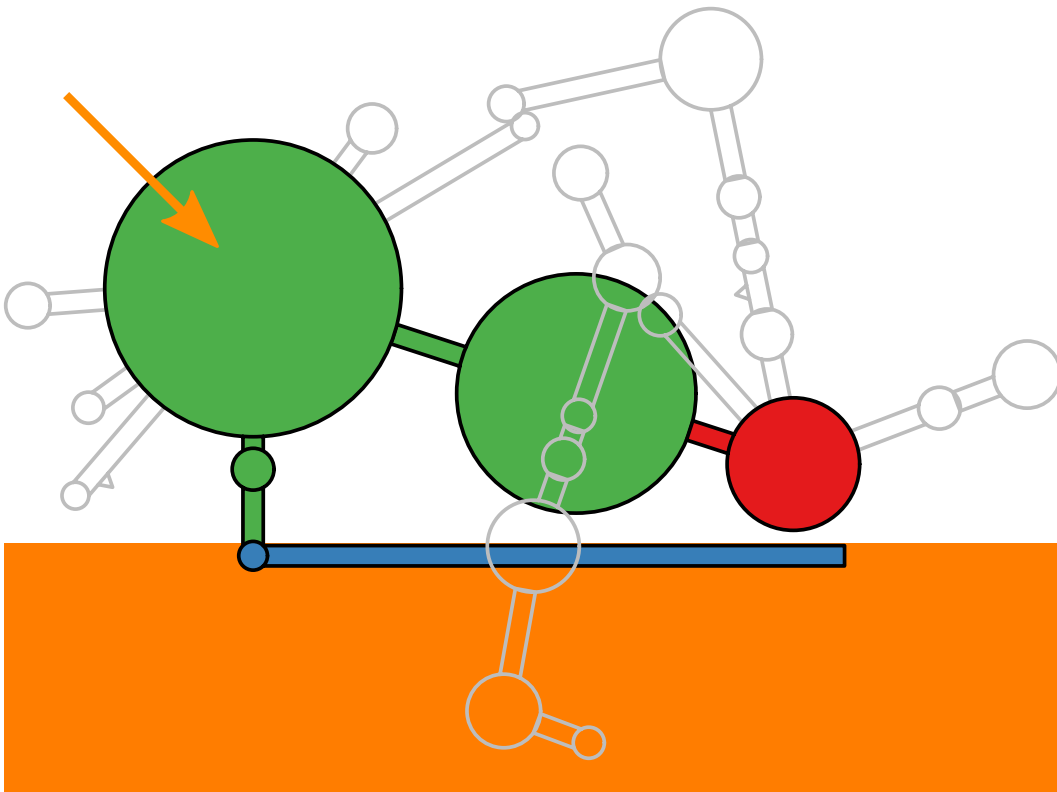
After creating the geometry of the root ancestor, it is assigned to be the intersected node. Then, the same algorithm that is used for resolving standard ancestor intersections is applied, whereby the *root ancestor* can not be used for resolving the intersection since this would violate the constraints concerning the exterior-loop.

Due to the structure of the exterior-loop, no child can intersect it. This also holds for the root ancestor: no child of the root ancestor can intersect the root ancestor. The reason for this is that the angle between root ancestor and any child is either  $90^\circ$  or  $270^\circ$ .

An exterior intersection before and after resolving the intersection is shown in Figure 5.7. The color coding is the same as for the ancestor intersection example: the red node is the intersecting node, the blue node is the intersected node, and the green nodes are the intermediate nodes on the path from the intersected node to the intersecting node. The green node marked with an orange arrow is the rotation node used for resolving the exterior intersection. As can be seen, the length of the root ancestor stem and thus of the stem of the intersected node (blue) matches horizontally the axis-aligned bounding box of the intersecting node.



(a) Intersection between the current node (red) and the root ancestor node (blue). All green nodes do not intersect the root ancestor node.



(b) After resolving the intersection shown in Figure 5.7a, the algorithm proceeds with the green node marked with an orange arrow used for resolving the intersection.

Figure 5.7: Exterior intersection (a) and its resolution (b).

**Algorithm 10** Resolve exterior intersection

---

```

construct geometry for the exterior loop
assign exterior node to the be intersected node
Solve ancestor intersection {Algorithm 9}

```

---

**Algorithm 11** Detect sibling intersection

---

```

for each child of the current node do
  if pair of subtrees then
    Resolve sibling intersection
  end if
  start sibling intersection at the ancestor of the current node
end for

```

---

**S-Planarity**

Detecting and resolving sibling intersections are performed by the third step of the algorithm. If the detection algorithm (Detect Sibling Intersections, Algorithm 11) does not find any sibling intersection for the current node, then the current node is S-planar and no further actions have to be performed. If a sibling intersection is detected and the current node is the root node, the algorithm for resolving exterior subtree intersections is invoked (Resolve Exterior Subtree Intersections, Algorithm 13). Otherwise, the algorithm for resolving sibling intersections is used (Resolve Sibling Intersections, Algorithm 12).

**Detect Sibling Intersections** In order to detect a sibling intersection of a node  $v \in V$ , all the subtrees  $T_i = (V_i, E_i)$  of its children  $w_i \in V, (v, w_i) \in E$  are checked. If  $v$  has no children, then it is S-planar. The same holds if  $v$  has only one child. If  $v$  has  $n$  children, the sibling intersection is checked as follows.

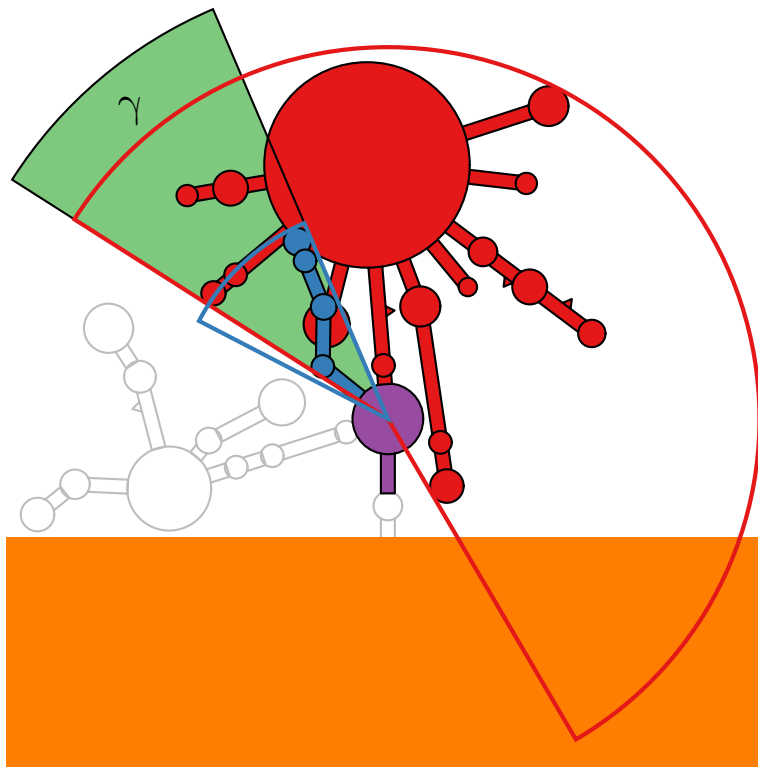
$$\forall 1 \leq i \leq n - 1$$

$$\forall i < j \leq n$$

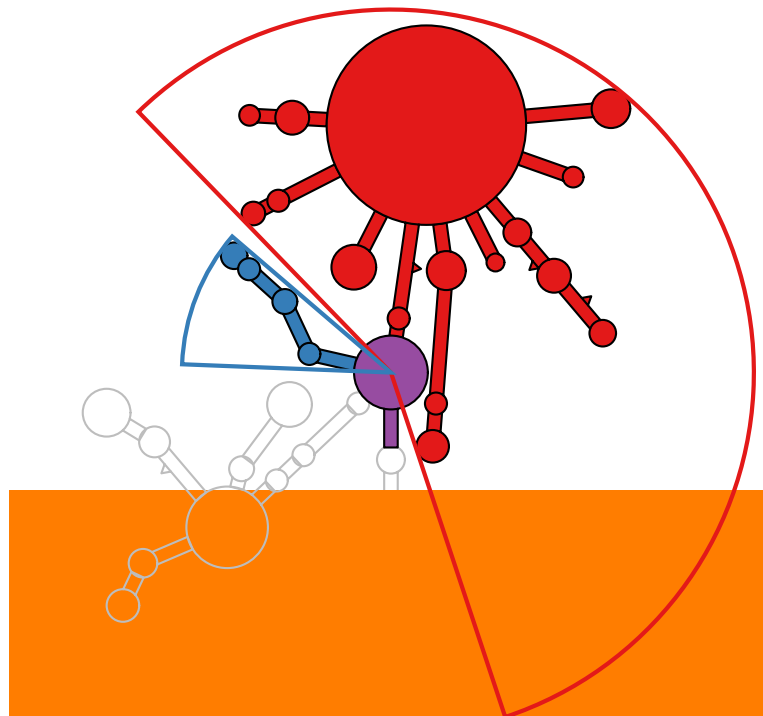
$$\forall w_i \in V_i \forall w_j \in V_j : \text{check if } w_i \text{ and } w_j \text{ intersect}$$

If no intersection is found, then  $v$  is S-planar. Otherwise, the first intersection found is reported and resolved.

**Resolve Sibling Intersections** The violet node in Figure 5.8a has a sibling intersection between the blue subtree and the red subtree. The intersection angle  $\gamma$  (green sector) is computed by first finding the smallest enclosing sector of each subtree with respect to the center of the violet node (blue and red circle



(a) Intersection between the red child tree and the blue child tree of the violet node.



(b) After resolving the intersection shown in (a), an ancestor intersection of the gray child tree with the exterior-loop occurs and needs to be resolved.

Figure 5.8: Siblings intersection (a) and its resolution (b).

---

**Algorithm 12** Resolve sibling intersection

---

```

identify segments for bending
calculate overlap angle
distribute overlap angle equally over identified segments
subtract overlap angle from the remaining segments

```

---



---

**Algorithm 13** Resolve intersection of two subtrees of the exterior

---

```

calculate overlap of the siblings with bounding boxes
if flipping then
    if check if second sibling can be flipped without intersections then
        flip the second sibling
    return
    end if
end if
calculate minimal overlap configuration
stretch exterior loop segments between the siblings

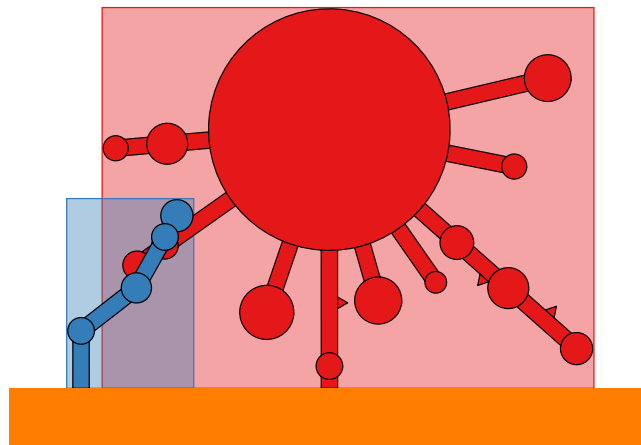
```

---

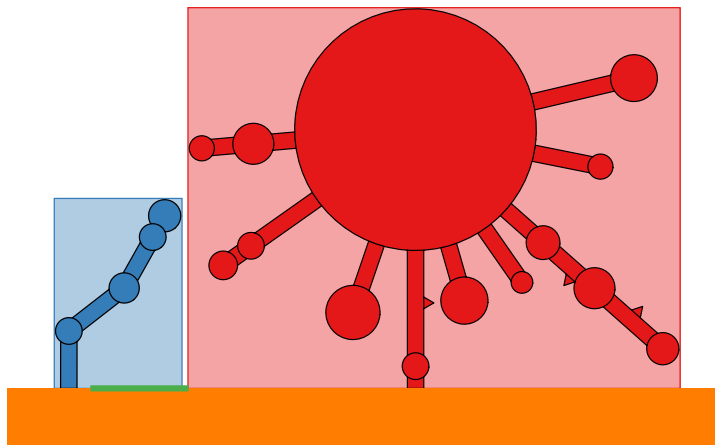
sectors, respectively). Then, the smaller angle of the red sector is subtracted from the larger angle of the blue sector. Hereby, the angle is computed clockwise. To resolve this intersection, the blue subtree is rotated counter-clockwise by  $(\gamma + \delta)/2$ , while the red subtree is rotated clockwise by  $(\gamma + \delta)/2$ . Here, the angle  $\delta$  is the minimal angle between two subtree sectors. To allow for this rotation while maintaining a minimum distance between unpaired nucleotides as well as between unpaired and paired nucleotides of the violet loop, the radius of this loop might have to be increased.

**Resolve Exterior Subtree Intersections** The blue subtree and the red subtree of the exterior-loop intersect in Figure 5.9a. The overlap between the blue bounding box and the red bounding box is computed. Onto this overlap, a minimal distance is added. The resulting increase necessary to separate the blue and the red subtree is distributed equally among the distances between the exterior nucleotides connecting both subtrees (Figure 5.9b).

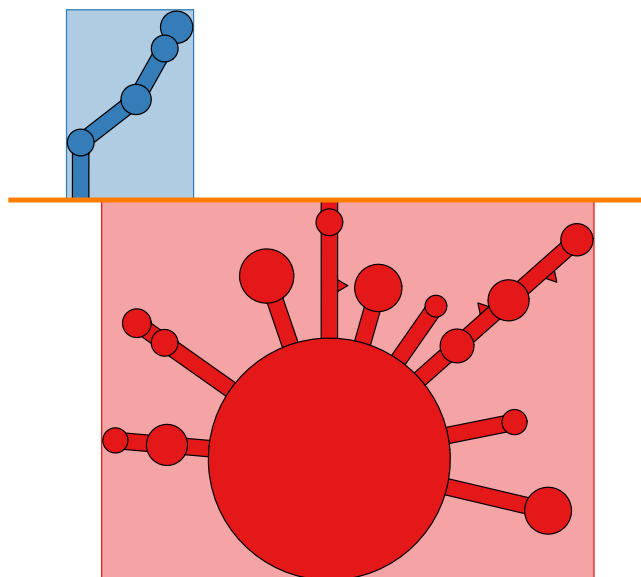
Alternatively, the red subtree might be reflected at the line of the exterior-loop (Figure 5.9c). This does not cause any new intersection between the red subtree and the exterior-loop, or within the red subtree. The resulting drawing is more compact. Allowing this type of resolving exterior subtree intersections might make it harder locating the exterior-loop.



(a) Intersection between the red child tree and the blue child tree of the orange exterior-loop.



(b) The intersection is resolved by increasing the distance between the nucleotides of the exterior-loop between the two child trees.



(c) The intersection is resolved by changing the direction of the second child tree now pointing downwards.

Figure 5.9: Exterior subtree intersection (a) and its resolution by stretching (b) or flipping (c).



### **AABB Box Intersection**

The detection of intersections is performed using standard algorithms. Efficient intersection tests for pairs of simple geometric objects can be found in books such as the one by Ericson [34]. As accelerating structure, axis-aligned bounding boxes (AABB) are used. These are updated immediately after the position or the orientation of a node is changed. Overall, this is sufficient for obtaining a high performance of the algorithm while using minimal space and minimal computational overhead.

## **5.5 Optimization**

Creating intersection-free drawings as described before leads to inner and multi-loops having a very large radius. The radius of hairpin loops is not changed by the algorithm and bulge loops are fixed. Overly large inner and multi-loops are shrunk by an optimization step that reduces the radius of these loops. This optimization step is performed if all of the following conditions are met:

1. the subtree at the current node has no sibling intersections
2. the subtree and the ancestors of the current node do not intersect
3. one of the following conditions is met
  - (a) the radius of the current node has increased by at least a factor of ten compared to its minimum radius
  - (b) the parent of the current node is the exterior

Conditions 1 and 2 hold whenever the handling of a node finished during depth-first traversal. If Condition 3a is met, loops that grew too much (by a factor of ten in relation to their minimum radius) are shrunk. This avoids a too large increase of the loops' radii during planarization. If the parent of the current node is the exterior (Condition 3b), the complete subtree attached to the exterior loop is intersection-free (planar) and will be optimized.

Conceptually, optimization starts at the leaves of the tree constructed—the hairpin loops—and proceeds towards the root of the tree—the exterior loop. The algorithm starts at the current node and optimizes all its children recursively, using depth-first traversal. As long as the radius of at least one child of the current node has been reduced, all children of this node are optimized again. If

no child could be optimized further, the algorithm attempts to reduce the radius of the current node. If the current node can be optimized, the optimization procedure is started again because it is possible that the radii of the children can be improved further.

Optimization proceeds in two alternating steps. The first step reduces the radius of a loop by performing a binary or a linear search for the smallest radius such that the drawing remains intersection-free. Here, the minimum radius is the one that can be reached while respecting the minimum distance between consecutive nucleotides and the current configuration of the loop. Moreover, the maximal radius is the one created by the intersection-free drawing. The search starts with the maximal radius that is then iteratively decreased by a fixed amount until the minimum radius is reached. The second step changes the angles between each pair of adjacent stems of a loop. This allows attaining a smaller radius, which is closer the optimal minimum radius. During this step, the algorithm calculates the angles between the unpaired bases for each segment of a loop. Afterwards, the segment with the largest angle is chosen for optimization. Therefore, the free angle between the stems of the segment is calculated based on the minimal backbone distance. Next, similar to the sibling intersection in Section 5.4.3, the algorithm constructs two wedges to present the neighboring stems. Using these wedges, the free angle between each pair of stems is calculated. Afterwards, the algorithm evaluates the results these calculations and picks the smallest one for the angle reduction. By using this minimal free angle, the algorithm decreases the angle of the selected segment by 50% of the chosen free angle. Finally, the algorithm checks, if the reduction introduced a new intersection and discards the change if necessary. If an angle is changed by this step, both steps are repeated.

All optimization steps are only performed if the corresponding subtree is planar and does not intersect any ancestor, i.e., if Constraints 12' and 13' are met. These constraints are conserved by the optimization process.

In order to demonstrate the effects of the optimization, we have drawn our example RNAs using RNApuzzler with the optimization being deactivated. The results are shown in Figure 5.10. To ease comparison with the final results of RNApuzzler, we added the drawings generated using RNApuzzler with optimizations being activated below the figures. It is clearly visible that the optimizations reduce the amount of drawing space needed considerably and that they avoid unnecessary radii enlargements. Due to the fact that all optimizations are applied locally to a single loop, it is only possible to reduce

---

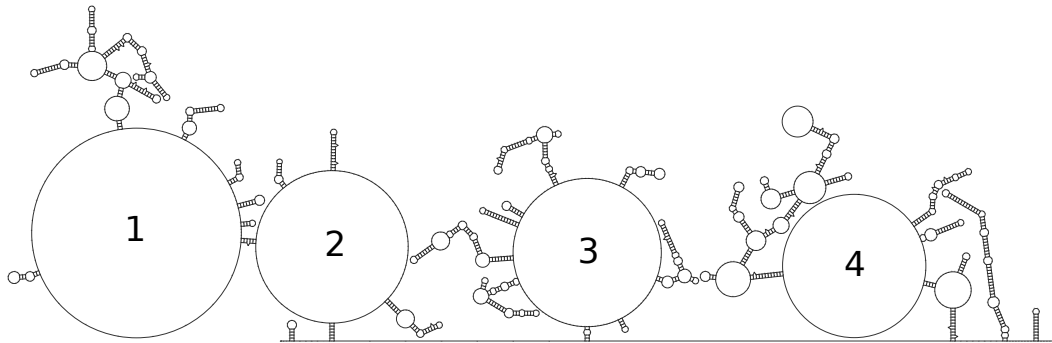
**Algorithm 14** Optimize Loop

---

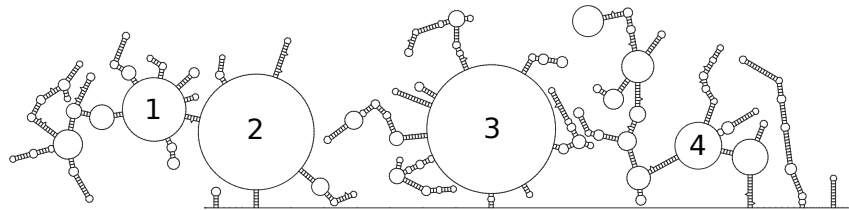
```
repeat
  calculate a new radius for the loop with no new intersection
  calculate a new loop configuration:
    for all stems of the loop do
      create a wedge for the stem
    end for
  calculate free angle between the wedges
  select segment with largest free angle
  reduce free angle of the segment by 50%
  if new intersection was introduced then
    revoke angle change
  end if
until (radius or configuration did not change)
```

---

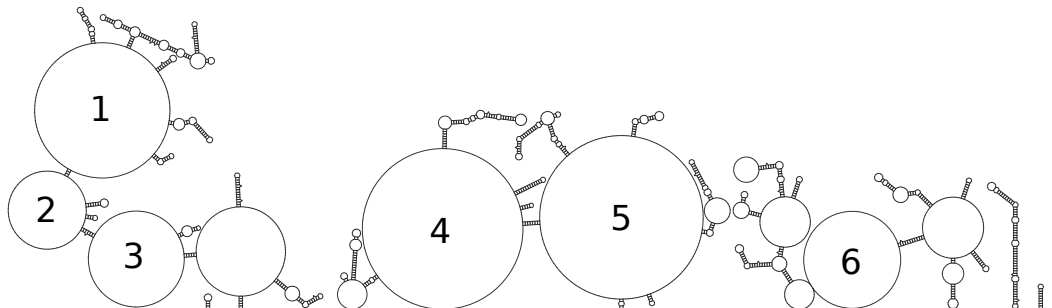
the radius of the targeted loop as long as no new intersections are introduced by the optimization.



(a) Secondary structure of AADC01167538.1/2439-571 Homo sapiens without optimization.



(b) Secondary structure of AADC01167538.1/2439-571 Homo sapiens with optimization.



(c) Secondary structure of CABD02136541.1/885-2750 Gorilla gorilla without optimization.



(d) Secondary structure of CABD02136541.1/885-2750 Gorilla gorilla with optimization.

Figure 5.10: Comparison of the human and the gorilla SSU rRNA drawn using RNApuzzler without and with optimization.

## 5.6 Benchmarks

A main goal in the development of RNApuzzler was to keep the overall runtime of the drawing algorithm smaller than the actual folding time. To verify this goal, RNAfold from the ViennaRNA Package version 2.35 [54] was used to fold 5000 RNAs from the family RF00012 that contains smaller RNAs (around 215nt) and 50 large sequences from family RF01960 (around 1860nt) from the RFAM [38]. The benchmark was performed on a machine with an Intel i7-7700HQ CPU and a Samsung Evo 850 EVO M.2 SSD. The results are shown in Table 5.1. RNApuzzler draws the secondary structure in less than 10% of the folding time. Compared to NAView [22], the standard drawing algorithm of the ViennaRNA Package [54], which draws the RNA in nearly linear time, it takes considerably more computing time. Drawing larger RNAs, most of the time is needed by the optimization algorithm reducing the size of the multi-loops. This behavior is related to the increase of the number of structures in general and the number of multi-loops in particular with increasing length of the RNA. Other algorithms can only be compared using this benchmark to a limited extent, as they are usually not directly manageable via a batch job. Additionally, most of these tools are designed as GUI applications. Therefore, they were excluded from this benchmark. In general, the force directed layout based algorithms shown in Section 5.7 are at least one order of magnitude slower than RNApuzzler.

As a stress test for the algorithm, it was applied to all RNAs provided by the RFAM Version 12.2 [60]. All sequences were folded using RNAfold [54] drawn using RNApuzzler, and checked for the absence of intersections. All RNAs collected in this database were drawn without intersections. The complete process took approximately 2 days on a 48 core workstation running up to 46 processes in parallel.

	RNAfold	NAVview	RNApuzzler	
			simple	optimized
5000 sequences RF0012	76.14s	1.66s	2.761s	5.585s
50 sequences RF01960	129.48s	0.138s	0.262s	10.907s

Table 5.1: Benchmark results comparing folding and drawing time for different RFAM families.

Table 5.2: Constraints met by different algorithms: ✓: met, \*: not fully met, ?: unclear, ✗: not met

Algorithm	Constraint							
	2,3	4-7	8	9	10	11	12'	13'
<i>RNApuzzler</i>	✓	*	✓	✓	*	✓	✓	✓
<i>RNAturtle</i>	✓	✓	✓	✓	✓	✓	✗	✗
NAView [22]	✗	*	✓	✗	✗	✗	✗	✗
forna [48]	✗	✗	✗	✗	✓	✗	✗	✗
jViz.RNA 2.0 [85]	✗	✗	✗	✗	✓	✗	✗	✗
jViz.RNA 4.0 [73]	✗	✗	✓	✓	?	✗	✗	✗
RNAfdl [41]	✗	✗	✗	*	✓	✗	✓	✗
VARNA (radial) [31]	✓	*	✓	✓	✓	✗	✗	✗
RNAstructure [66]	✗	*	✓	✗	✗	✓	✗	✗
PseudoViewer 3 [24]	✗	*	✓	✓	✗	✗	✗	✗
Auber et al. [19]	✗	✗	✓	✗	✗	✓	?	?

## 5.7 Results

Table 5.2 shows which constraints introduced in Section 5.4 are met by different RNA drawing tools. By design, RNAturtle meets Constraints 1-11. RNApuzzler meets all constraints with Constraints 4, 6, and 10 being met in their relaxed forms, respectively. Constraint 10 is still satisfied, however, planarity has precedence: after finding a planar solution, all loops are made as compact as possible by the optimization step. A detailed comparison is provided in Section 5.8.

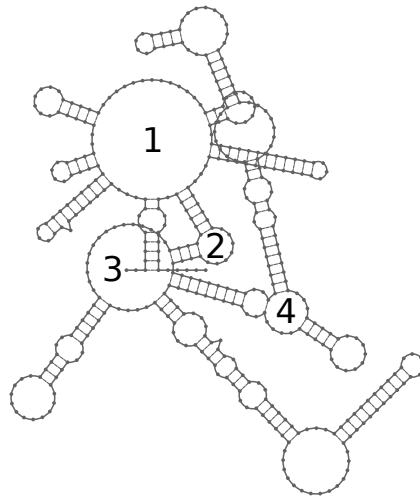
All other tools only meet the constraints to a lesser extent. In the following, the layouts of these tools for the human and the gorilla SSU rRNAs are presented together with additional information about the different algorithms. Further, the tools' advantages and issues are discussed in detail.

## 5.8 Comparison to other algorithms

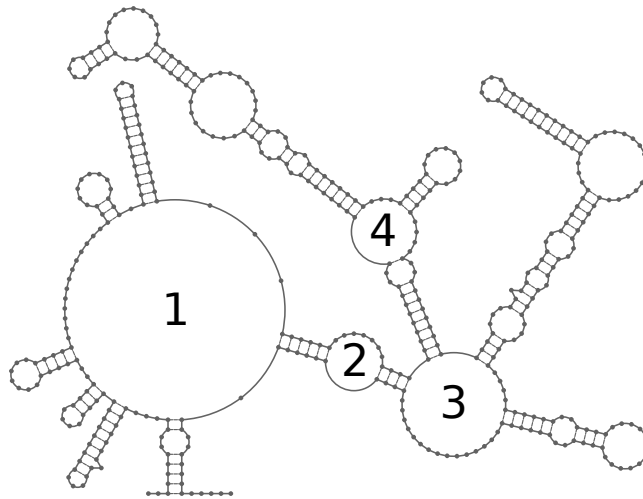
### 5.8.1 Comparison to NAView

A planar drawing generated by RNAturtle is shown in Figure 5.4a. However, not all RNAs can be drawn planar by this algorithm, as demonstrated by the more complex example in Figure 5.11a. This RNA contains 447 nucleotides forming 11 internal loops, 3 multi-loops, and 9 hairpin loops. The same structure drawn by RNApuzzler is shown in Figure 5.11b. Here, the loops marked 1, 2, 3, and 4 were enlarged and changed. These four changes suffice to obtain a planar drawing.

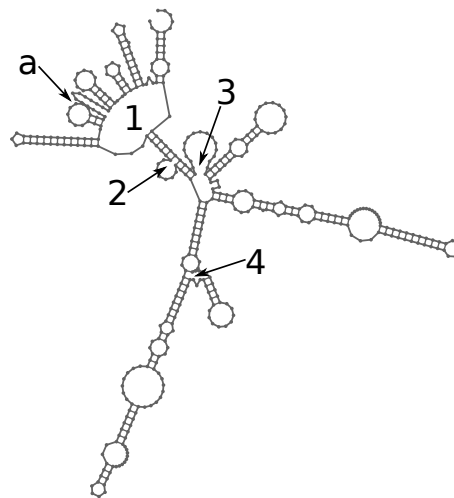
For comparison, the drawing generated by NAView, the current default algorithm of the ViennaRNA package [54] is shown in Figure 5.11c. As can be seen, there are intersections and overlaps (a) as well as dense packings of nucleotides (4). Furthermore, it is not easy to determine where the exterior loop is located and it is unclear why loop (1) forms a 'hand'-like structure, where the upper stems form the 'fingers' and the lower stem looks like the 'arm'. Furthermore, one 'finger' of the hand (marked (a)) is not a stem! In fact, it is a sequence of unpaired bases of the loop that looks like a stem and that is very hard to differentiate from a real stem. RNApuzzler was tested to find out whether similar drawings can be obtained for similar RNAs. For this comparison the SSU rRNA from human and gorilla were used as examples from the RFAM family RF01960 [38]. Both sequences were first aligned to the RFAM SSU model with *cmalign* [61]. The base pairs of the consensus model were used as folding constraints by RNAfold [54]. The drawings generated using RNApuzzler and NAView are shown in Figure 5.12. It is noticeable that RNApuzzler requires more drawing space since the exterior loop is drawn as a linear structure. By doing so, it is very easy to detect the exterior loop, whereas with the NAView algorithm the exterior loop is difficult to locate. Furthermore, it is clearly observable that RNApuzzler creates planar drawings for both structures whereas NAView creates non-planar drawings with huge intersections. Although it is possible to detect larger structural changes between the two NAView layouts, the details remain unclear. Since the drawings are intersecting massively in some regions, it is nearly impossible to differentiate between the structural elements within these regions. On the other hand, with RNApuzzler it is feasible following the path of each structural element to detect structural changes between two different species.



(a) Drawing generated by RNAturtle; not planar.



(b) Drawing generated by RNApuzzler; planar.



(c) Drawing generated by the NAView algorithm using the ViennaRNA package [54].

Figure 5.11: Drawings of the secondary structure of the tRNA-Leu (trnL) gene and trnL-trnF intergenic spacer from the chloroplast of *Streptocarpus papangae* isolate S106 (FJ501444.1/1-447 retrieved from RF00028 [38]).



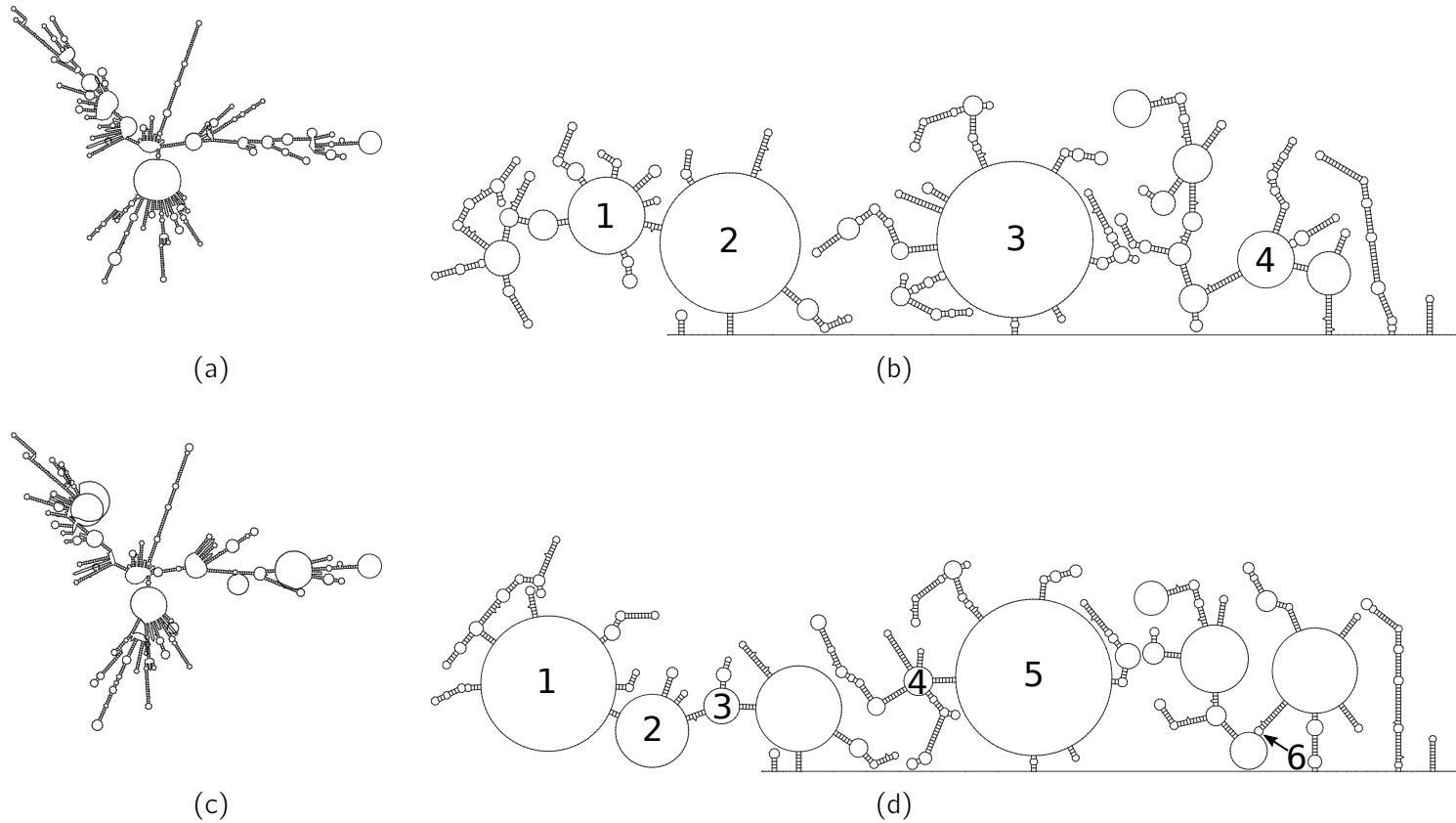


Figure 5.12: Comparison of the human and gorilla SSU rRNA. (a) Secondary structure of AADC01167538.1/2439-571 Homo sapiens drawn with NAView. (b) Secondary structure of AADC01167538.1/2439-571 Homo sapiens drawn with RNApuzzler. (c) Secondary structure of CABD02136541.1/885-2750 Gorilla gorilla drawn with NAView. (d) Secondary structure of CABD02136541.1/885-2750 Gorilla gorilla drawn with RNApuzzler.

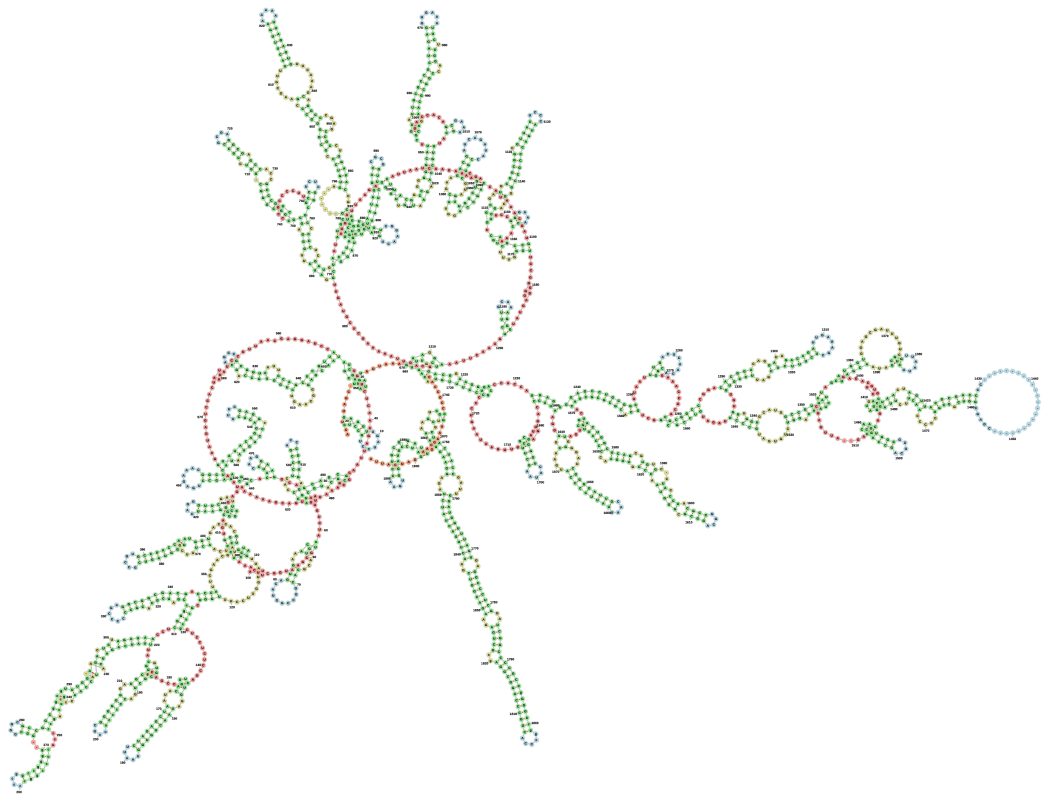
## 5.8.2 Comparison to other tools

The additional algorithms described in Table 5.2 were used to compare the results of RNApuzzler with state of the art tools. To facilitate the comparison, the tools have been classified into two algorithm categories: force-directed layouts (Section 5.8.2) and tree-based layouts (Section 5.8.2).

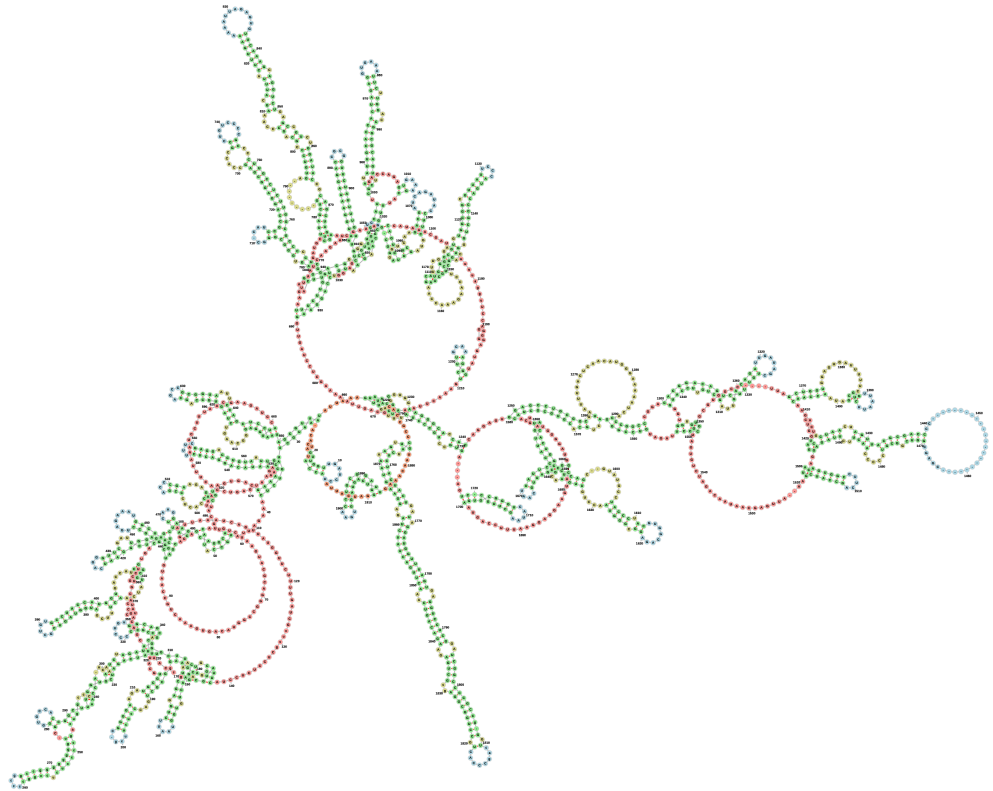
### Comparison to FDL-based Layouts

One group of tools for drawing RNA secondary structures relies on force-directed layouts (FDL). The simplest approach for calculating FDLs is based on the structure of the graph. The basic idea is to have repelling forces between vertices that are not connected and attracting forces between the vertices connected by edges [78]. These forces are used to iteratively adapt the positions of the vertices. Practical applications of this paradigm require several improvements, including parameters and heuristics for assuring termination of the process, additional forces for keeping the drawing centered, or simplifications for computing approximations of the forces [78]. One possibility for speeding up the calculation of the layout is based on providing an initial layout of the graph that is then optimized by the FDL algorithm. The complexity of the FDL algorithm varies widely based on the implementation. Simpler algorithms provide a complexity of  $O(n \cdot (|E| + |V|^2))$ , where  $n$  is the number of iterations of the algorithm [32]. By using more sophisticated implementations, it is possible to reduce the complexity towards  $O(n \cdot (E + |V| \cdot \log|V|))$  [65].

**Forna** Forna [48] is implemented as a web service in JavaScript and uses a preprocessed layout created by the NAView algorithm as the initial layout for the FDL algorithm. To avoid crossings of the edges connecting the nucleotides of two adjacent base pairs (backbone edges), support edges between these base pairs are added such that these base pairs form a complete  $k_4$ . As shown in the Figures 5.13a and 5.13b, Forna does not always produce planar drawings because stems attached to a multi-loop may be flipped to the inner side of this loop. This behavior can impede the analysis and the comparison of RNA structures. In Figure 5.13, for example, it is hard to detect the changes between the two species.



(a) Secondary structure of AADC01167538.1/2439-571 Homo sapiens.



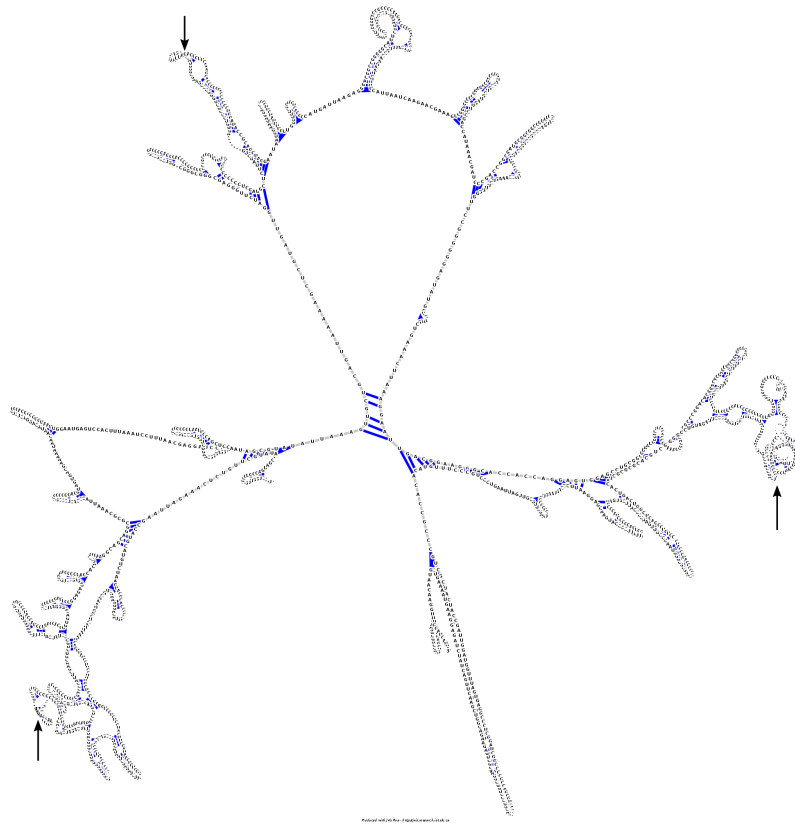
(b) Secondary structure of CABD02136541.1/885-2750 Gorilla gorilla.

Figure 5.13: Comparison of the human (a) and the gorilla (b) SSU rRNA drawn using Forna [48].

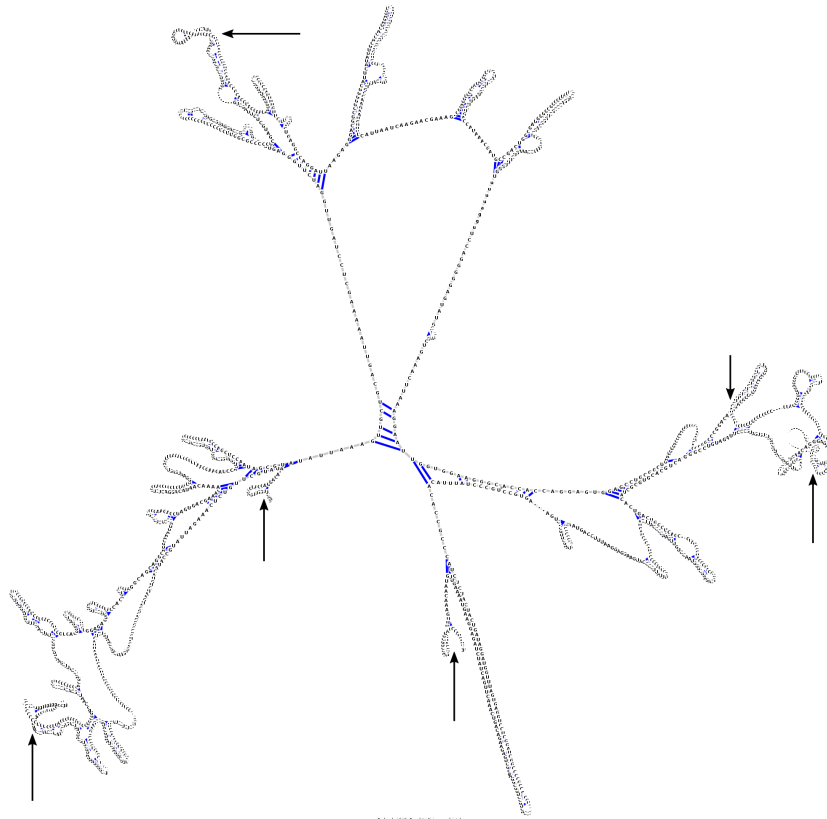
**jViz.RNA 2.0** jViz.RNA 2.0 [85] is implemented in Java and uses the simple circle layout of the RNA as its initial layout, which is planar in the absence of pseudoknots. Nevertheless, the results of jViz.RNA 2.0 are often non-planar, as shown, e.g., in Figures 5.14a and 5.14b. Furthermore, the layouts are not stable because the results differ for several runs with the same input. This makes it even harder to compare different sequences.

**jViz.RNA 4.0** jViz.RNA 4.0 [73] improves the runtime of the FDL algorithm by applying numerical integration methods. The developers improved also the layout of stems and loops by using what they call a “compressed graph” of the RNA structure. It reduces stems and loops similar to our RNA-Tree approach. After calculating the layout, they apply a static template for each loop and each stem. The results obtained are shown in Figures 5.15a and 5.15b. Compared to jViz.RNA 2.0 this approach may produce additional intersections. It is also possible to create swinging stems that circulate endlessly around a loop. The clutter in Figure 5.15b is a result of this effect.

**RNAfdl** RNAfdl [41] is implemented in C. Like jViz, the circle layout is used as an initial layout but RNAfdl takes much more time to compute the layouts shown in Figures 5.16a and 5.16b ( $\approx 4$  hours). In contrast to the other FDL-based tools, it produces planar drawings. However, whose drawings are very densely packed with curved stems and deformed loops. Therefore, these drawings are hard to compare. Another drawback of this tool is its excessive computation time.

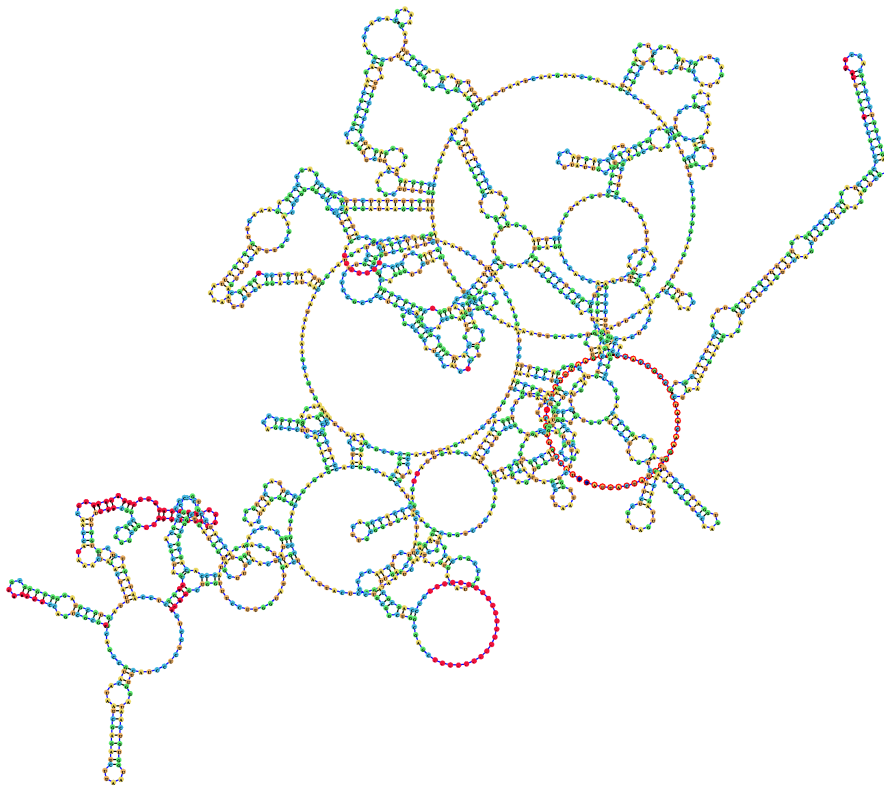


(a) Secondary structure of AADC01167538.1/2439-571 Homo sapiens.

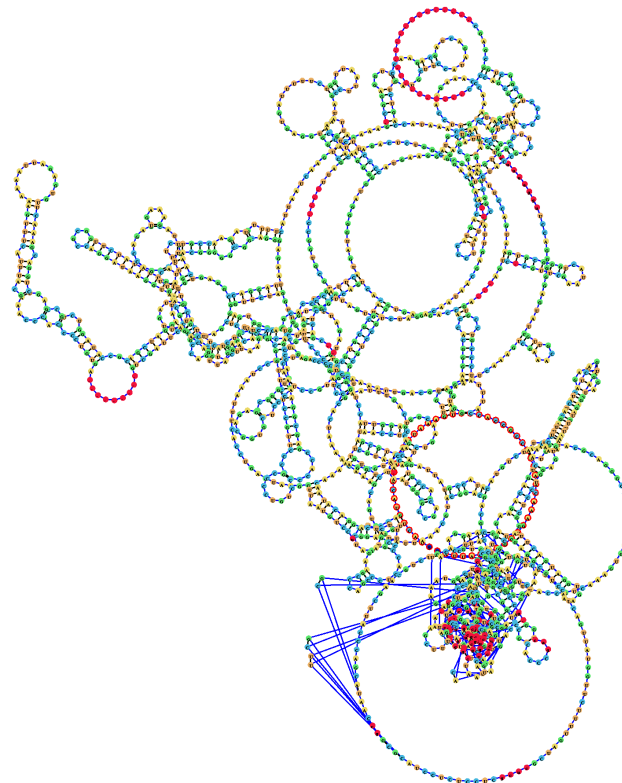


(b) Secondary structure of CABD02136541.1/885-2750 Gorilla gorilla.

Figure 5.14: Comparison of the human (a) and the gorilla (b) SSU rRNA drawn using jViz.RNA 2.0 [85]. Edge crossings are annotated with arrows.

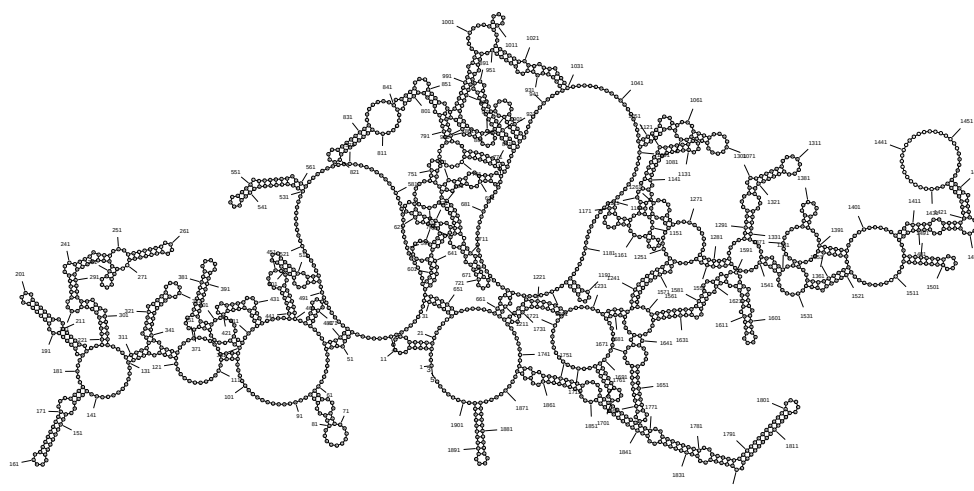


(a) Secondary structure of AADC01167538.1/2439-571 Homo sapiens.

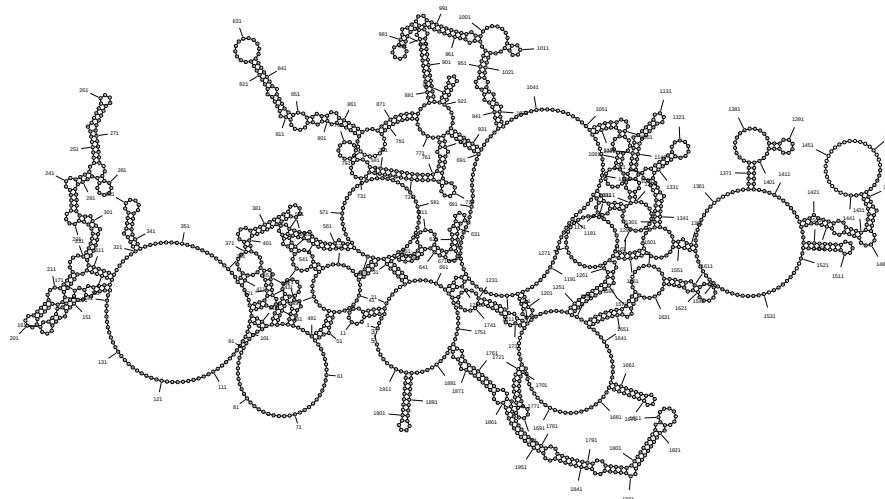


(b) Secondary structure of CABD02136541.1/885-2750 Gorilla gorilla. Please note, that this is a snapshot after 10 minutes of calculation. The overlapping clutter at the bottom is rotating in every step, as jViz 4.0 fails to resolve the intersection.

Figure 5.15: Comparison of the human (a) and the gorilla (b) SSU rRNA drawn using jViz.RNA 4.0 [73].



(a) Secondary structure of AADC01167538.1/2439-571 Homo sapiens.



(b) Secondary structure of CABD02136541.1/885-2750 Gorilla gorilla.

Figure 5.16: Comparison of the human (a) and the gorilla (b) SSU rRNA drawn using RNAfdl [41].

## Comparison to Tree-based Layouts

Several layouting algorithms for RNA secondary structure use a tree-based layout. These algorithms are based on the tree structure of the RNA folding and most of them visualize the RNA starting from the exterior-loop. Since tree layouts can run in  $O(n)$  time, most of these algorithms have this complexity. Postprocessing steps might increase the complexity, however. The NAView algorithm [22] is also a tree-based algorithm. However, since it was compared in detail in Section 5.8.1, it will not be discussed further here.

**VARNA Radial Layout** VARNA [64] is a Java-based tool and provides multiple drawing algorithms for RNA secondary structures. Beside the basic circle and arc layouts [83], it provides the so-called “radial layout”. The algorithm for drawing this radial layout is similar to our RNAturtle algorithm, since it draws the RNA secondary structures using a linear approach. VARNA comes with a GUI and provides several interaction methods so that the user can resolve intersections manually. It is notable that VARNA draws the exterior-loop as a straight line. As shown in Figure 5.17a and 5.17b, the drawings of larger RNAs contain many overlaps and require tedious manual postprocessing.

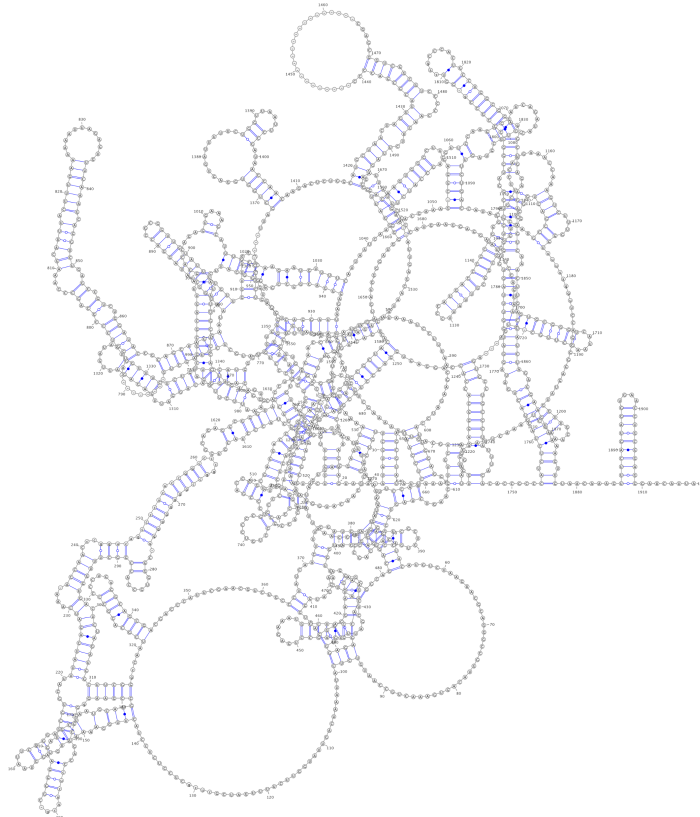
**VARNA NAView Layout** The VARNA NAView layout is based on the original NAView layout [22] but has slight modifications. It still tends to overlap in complex regions but reduces the ‘hand-like’ structures. By using different colors for the backbone and the base pair connections, the ‘stem-like’ regions within complex multi-loops are also easier to distinguish. However, due to the overlaps created by this algorithm, it is hard to compare the visualizations as shown in Figure 5.18a and 5.18b.

**RNAstructure** RNAstructure [66] is written in C++ and is the basic layout algorithm of the RNA folding package RNAstructure. It tries to arrange the outgoing stems of multi-loops on a circular arc, which produces “hand-shaped” structures similar to the ones generated by NAView. Furthermore, it arranges the stems connected to the exterior-loop on a circle. It remains unclear, why the calculated radii of the exterior-loops can become very large as shown in Figure 5.19a and Figure 5.19b. This algorithm is not suitable for larger RNAs since it produces several intersections and uses unnecessary much drawing space.





(a) Secondary structure of AADC01167538.1/2439-571 Homo sapiens.

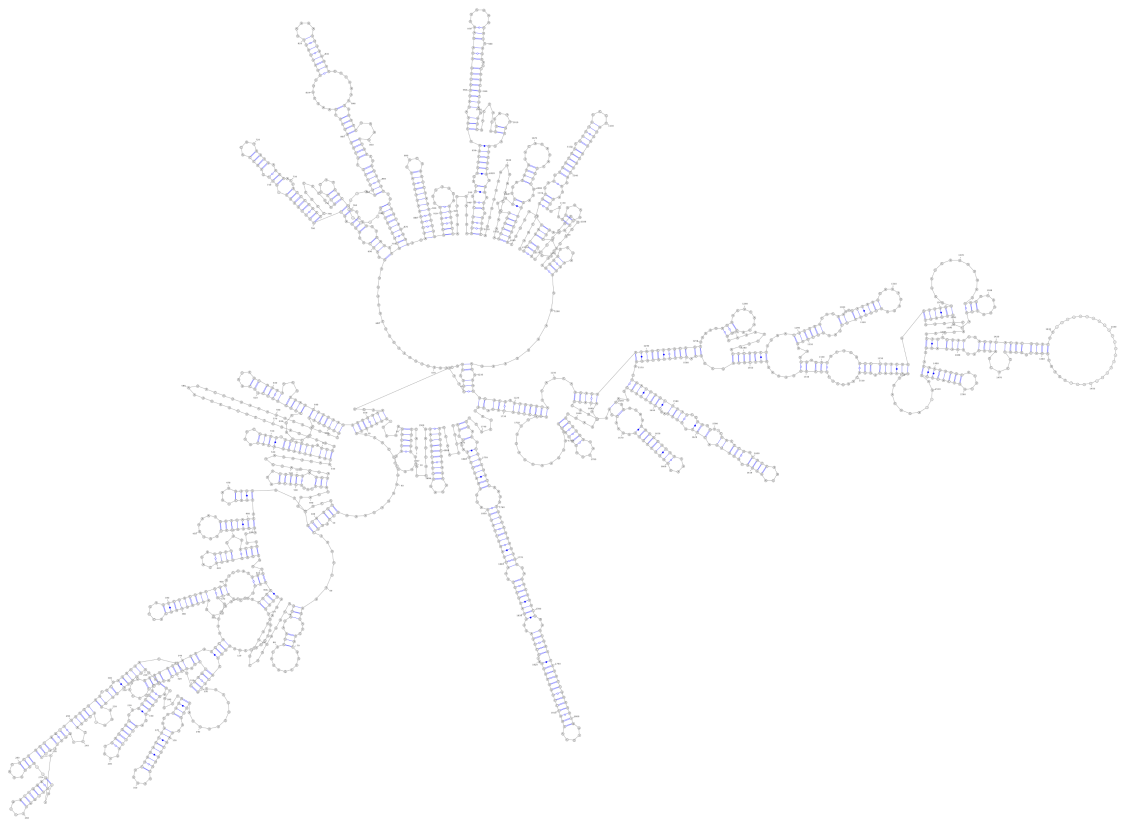


(b) Secondary structure of CABD02136541.1/885-2750 Gorilla gorilla.

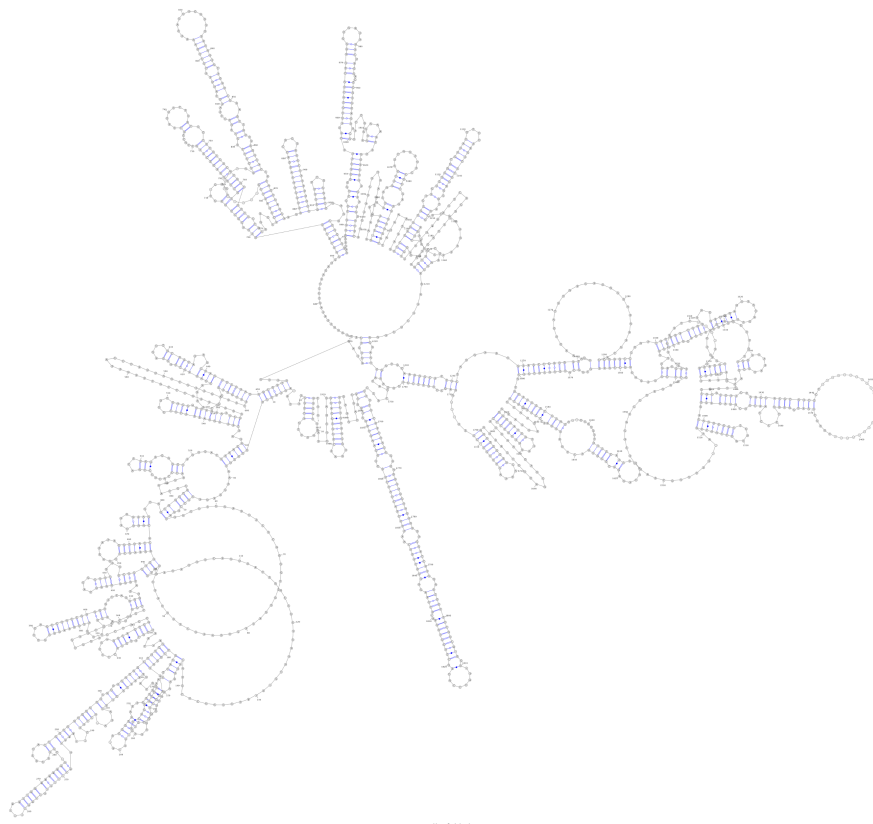
Figure 5.17: Comparison of the human (a) and the gorilla (b) SSU rRNA drawn using the VARNAL radial layout [64].

**PseudoViewer 3** PseudeViewer 3 [23] is a sophisticated RNA secondary structure layout algorithm implemented in C# that additionally layouts all possible types of pseudoknots. The layouts generated using PseudoViewer 3 are useful as shown in Figures 5.21a and 5.21b. However, while the authors claim that all layouts are planar, this is only true for a majority of cases. A non-planar example is shown in Figure 5.20. Unfortunately, details of the algorithm remain unclear since the description of the drawing mechanism for the pseudoknot-free regions is kept very brief [24]. The latest version of the software was published in 2012, and no open source version is available. The standalone client is not stable and crashes while loading specific RNA sequences. PseudoViewer 3 is also hosted as a web service, which loads the same sequences reliably. The complexity of the algorithm remains unclear and a detailed benchmarking is impossible due to the instability of the standalone client. Finally, the implementation in C# makes it cumbersome to use PseudoViewer 3 on operating systems other than Windows.

**ARNA** Auber et al. [19] presented an algorithm for creating RNA secondary structure layouts. They included the tool “ARNA” implementing this algorithm in their graph drawing framework named “Tulip” [35]. Unfortunately, the algorithm does not seem to be maintained any more and could not be found in the current Tulip version. Additionally, only a high level overview of the algorithm is given in the publications. The available information there does neither permit to assess whether or not the algorithm always produces planar layouts as claimed by the authors, nor to assess its time complexity.

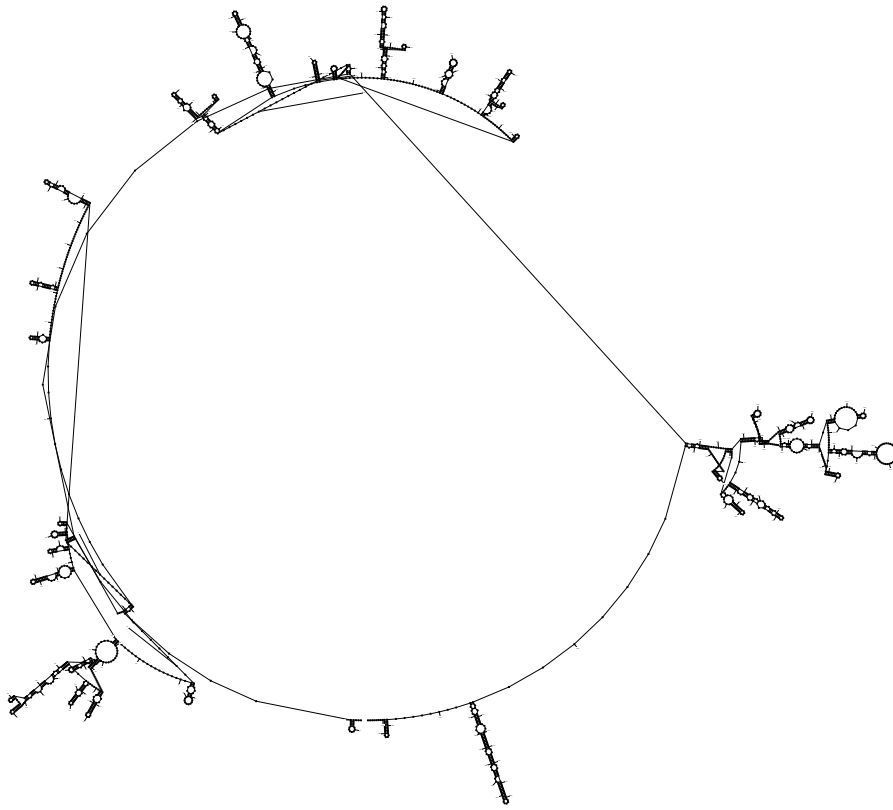


(a) Secondary structure of AADC01167538.1/2439-571 Homo sapiens.

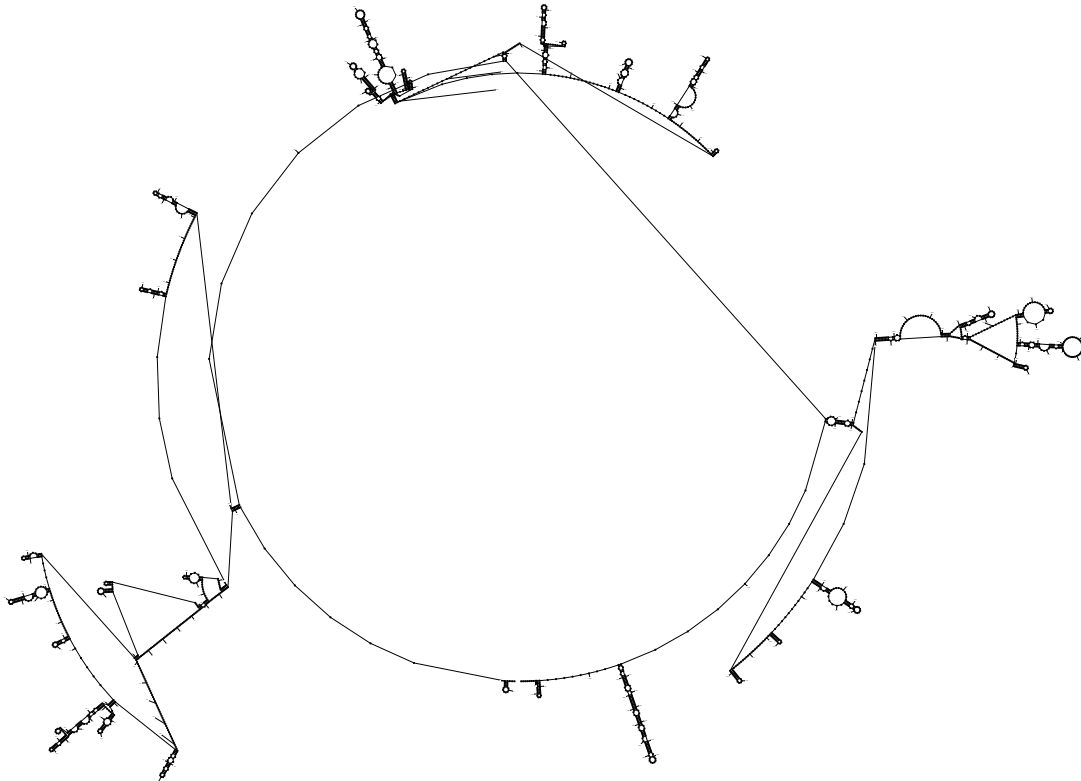


(b) Secondary structure of CABD02136541.1/885-2750 Gorilla gorilla.

Figure 5.18: Comparison of the human (a) and the gorilla (b) SSU rRNA drawn using the VARNAs NAView layout (see also Brucoleri and Heinrich [22]).



(a) Secondary structure of AADC01167538.1/2439-571 Homo sapiens.



(b) Secondary structure of CABD02136541.1/885-2750 Gorilla gorilla.

Figure 5.19: Comparison of the human (a) and the gorilla (b) SSU rRNA drawn using RNAstructure [66]).

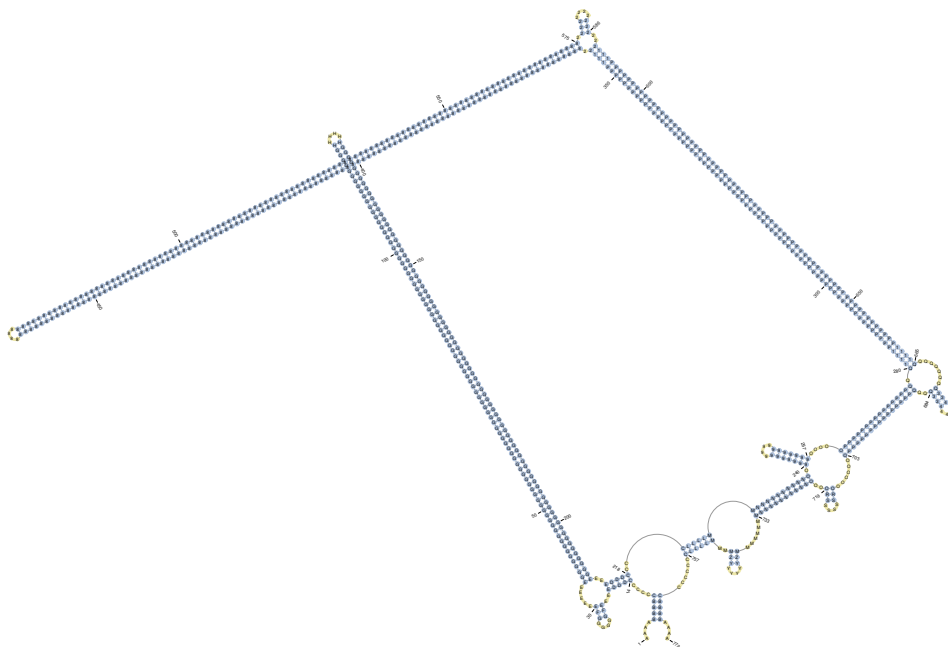
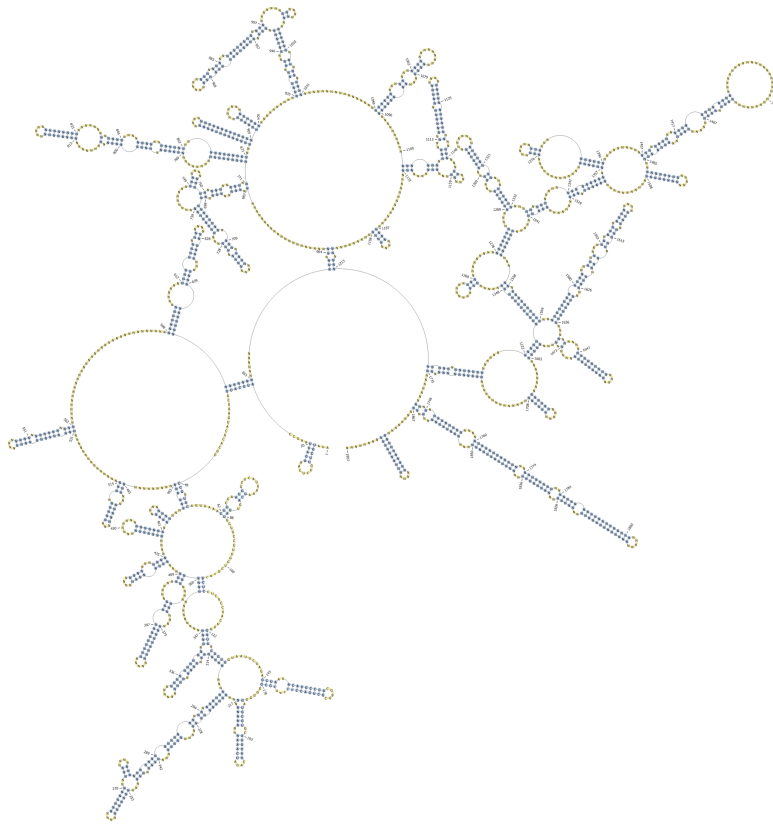
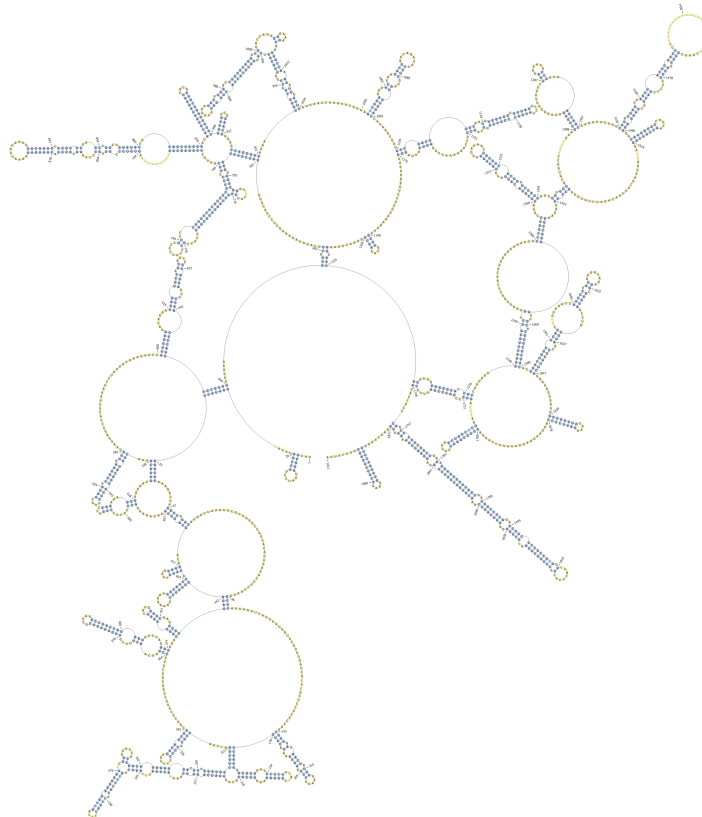


Figure 5.20: Example of a non-planar drawing produced by PseudoViewer 3



(a) Secondary structure of AADC01167538.1/2439-571 Homo sapiens.



(b) Secondary structure of CABD02136541.1/885-2750 Gorilla gorilla.

Figure 5.21: Comparison of the human (a) and the gorilla (b) SSU rRNA drawn using PseudoViewer 3 [23]).

# Chapter 6

## Conclusion

### In Detail

#### Sierra Platinum

Sierra Platinum is a fast and robust multiple-replicate peak-caller. So far, it is the only peak-caller allowing visual quality control and -steering. Its sophisticated statistical computation leads to provably better peak-calls compared to current approaches and tools. The procedure and parameters are chosen to produce an optimal result with respect to recall and FDR. Sierra Platinum is robust against noise and thus allows multiple-replicate peak-calling even for replicates not produced by the same lab or study. Alongside with Sierra Platinum, a benchmark data set was provided which allows to compare the performance of peak-callers with respect to specificity and sensitivity. The implementation of the method is optimized such that only as much memory as required to ensure a fast computation of the peak-calls is consumed.

#### iDotter

Dot plots are one of the default visualizations for the analysis of RNA secondary structure predictions. iDotter enhances and extends this visualization with state of the art interaction techniques. It is implemented using modern web programming languages and is provided as freely available web service. Based on its API, it is possible to integrate iDotter easily in analysis workflows. The expert can highlight regions of interest for documenting insights gained during the exploration of the dot plot. Furthermore, the exporting feature offers functions for collaboration.

## **RNApuzzler**

RNApuzzler provides a new planar layout algorithm to draw RNA secondary structures taking into account aesthetic constraints. The algorithms for drawing RNAs—RNAturtle and RNApuzzler—are implemented in the ViennaRNA package [54]. Due to the modularity of the implementations, they support testing alternatives for each of the individual steps. Furthermore, RNApuzzler creates comparable and deterministic drawings of RNA secondary structures. Compared to the folding time of an RNA by RNAfold [54], this is achieved with a fraction of the workload. Moreover, RNApuzzler outperforms most of the available layout algorithms since they do not produce planar drawings and/or require much more computation time. Finally, extensions and modifications of both RNAturtle and RNApuzzler can easily be included into the current implementation since the algorithms were implemented modularly in ViennaRNA.

## **In General**

In visualization, one usually needs a cooperation partner from the fields of applied research in order to generate questions and to obtain the necessary data. Especially in the field of computational biology it is necessary to involve the domain expert into the design process of the visualizations to create meaningful representations that are then also used by this community. Based on the size and the age of this community, several visualization were already developed by this community. However, they are highly specialized and can not easily be generalized from a visualization point of view. The majority of these representations are only well-known in this community, which is why an extensive literature search was necessary at the beginning of this thesis. A further characteristic is that the visualizations are directly integrated into the software packages of the bioinformaticians so that it is possible to interact directly with the processed data. Furthermore, it is possible to visualize results of intermediate steps of a workflow without exporting a huge amount of data for a post-processing step.







# Bibliography

- [1] Model of the fine structure of a chromosome. Last accessed at 10.11.2014 and modified (changed the telomere sequence): <http://upload.wikimedia.org/wikipedia/commons/1/1a/Chromosom.svg>.
- [2] A Java API for high-throughput sequencing data (HTS) formats. <http://samtools.github.io/htsjdk/>.
- [3] A set of tools (in Java) for working with next generation sequencing data in the BAM (<http://samtools.sourceforge.net>) format. <http://broadinstitute.github.io/picard/>.
- [4] Apache Commons IO. <https://commons.apache.org/proper/commons-io/>.
- [5] Apache Commons Logging. <https://commons.apache.org/proper/commons-logging/>.
- [6] Apache Commons Math. <https://commons.apache.org/proper/commons-math/>.
- [7] Apache Commons Net. <https://commons.apache.org/proper/commons-net/>.
- [8] Apache Commons VFS. <https://commons.apache.org/proper/commons-vfs/>.
- [9] DNA to protein or ncRNA. Last accessed at 28.11.2018: [https://commons.wikimedia.org/wiki/File:DNA\\_to\\_protein\\_or\\_ncRNA.svg](https://commons.wikimedia.org/wiki/File:DNA_to_protein_or_ncRNA.svg).
- [10] Google Gson. <https://github.com/google/gson/>.
- [11] JFreeChart. <http://www.jfree.org/jfreechart/>.

- 
- [12] JSch - Java Secure Channel. <http://www.jcraft.com/jsch/>.
- [13] Nucleosome organization schema. Last accessed at 27.11.2018: [https://commons.wikimedia.org/wiki/File:Nucleosome\\_organization.png](https://commons.wikimedia.org/wiki/File:Nucleosome_organization.png).
- [14] Schematic representation of histone modifications. Last accessed at 20.11.2018: [https://commons.wikimedia.org/wiki/File:Histone\\_modifications.png](https://commons.wikimedia.org/wiki/File:Histone_modifications.png).
- [15] A. Abdul-Rahman, G. Roe, M. Olsen, C. Gladstone, R. Whaling, N. Cronk, R. Morrissey, and M. Chen. Constructive Visual Analytics for Text Similarity Detection. *Computer Graphics Forum*, 36(1):237–248, 2016.
- [16] A. Ambrogelly, S. Palioura, and D. Söll. Natural expansion of the genetic code. *Nature chemical biology*, 3(1):29, 2007.
- [17] A. R. Amândio, A. Necsulea, E. Joye, B. Mascrez, and D. Duboule. Hotair Is Dispensable for Mouse Development. *PLOS Genetics*, 12(12):1–27, 12 2016.
- [18] S. Andrews. FastQC - A Quality Control tool for High Throughput Sequence Data.
- [19] D. Auber, M. Delest, J.-P. Domenger, and S. Dulucq. Efficient drawing of RNA secondary structure. *J. Graph Algorithms Appl.*, 10(2):329–351, 2006.
- [20] T. Barrett, S. E. Wilhite, P. Ledoux, C. Evangelista, I. F. Kim, M. Tomashevsky, K. A. Marshall, K. H. Phillippy, P. M. Sherman, M. Holko, A. Yefanov, H. Lee, N. Zhang, C. L. Robertson, N. Serova, S. Davis, and A. Soboleva. NCBI GEO: archive for functional genomics data sets—update. *Nucleic Acids Res*, 41(Database issue):D991–5, Jan 2013.
- [21] R. Berezney and K. Jeon. *Nuclear Matrix: Structural and Functional Organization*. A Single Volume Reprint of Volumes 162 a and B in International Review of Cytology Series. Academic Press, 1995.
- [22] R. E. Brucoleri and G. Heinrich. An improved algorithm for nucleic acid secondary structure display. *Computer applications in the biosciences: CABIOS*, 4(1):167–173, 1988.

- [23] Y. Byun and K. Han. PseudoViewer3: generating planar drawings of large-scale RNA structures with pseudoknots. *Bioinformatics*, 25(11):1435–1437, 2009.
- [24] Y. Byun and K. Han. An efficient algorithm for planar drawing of RNA structures with pseudoknots of any type. *Journal of bioinformatics and computational biology*, 14(03):1650009, 2016.
- [25] R. Cao, L. Wang, H. Wang, L. Xia, H. Erdjument-Bromage, P. Tempst, R. S. Jones, and Y. Zhang. Role of histone H3 lysine 27 methylation in Polycomb-group silencing. *Science*, 298(5595):1039–43, Nov 2002.
- [26] D. Charif and J. Lobry. SeqinR 1.0-2: a contributed package to the R project for statistical computing devoted to biological sequences retrieval and analysis. In U. Bastolla, M. Porto, H. Roman, and M. Vendruscolo, editors, *Structural approaches to sequence evolution: Molecules, networks, populations*, Biological and Medical Physics, Biomedical Engineering, pages 207–232, New York, 2007. Springer Verlag.
- [27] P. Cheung and P. Lau. Epigenetic regulation by histone methylation and histone variants. *Molecular endocrinology (Baltimore, Md.)*, 19(3):563–573, Mar. 2005.
- [28] F. Crick. Codon-anticodon pairing: the wobble hypothesis. 1966.
- [29] F. Crick. Central dogma of molecular biology. *Nature*, 227(5258):561, 1970.
- [30] F. H. Crick. On protein synthesis. In *Symp Soc Exp Biol*, volume 12, page 8, 1958.
- [31] K. Darty, A. Denise, and Y. Ponty. VARNA: Interactive drawing and editing of the RNA secondary structure. *Bioinformatics*, 25(15):1974, 2009.
- [32] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.
- [33] R. Edgar, M. Domrachev, and A. E. Lash. Gene Expression Omnibus: NCBI gene expression and hybridization array data repository. *Nucleic Acids Res*, 30(1):207–10, Jan 2002.
- [34] C. Ericson. *Real-Time Collision Detection*. Elsevier, 2004.

- 
- [35] G. Gainant and D. Auber. ARNA: Interactive Comparison and Alignment of RNA Secondary Structure. In *10th IEEE Symposium on Information Visualization (InfoVis 2004), 10-12 October 2004, Austin, TX, USA, 2004*.
- [36] A. J. Gibbs and G. A. McIntyre. The Diagram, a Method for Comparing Sequences. *European Journal of Biochemistry*, 16(1):1–11, 1970.
- [37] E. Goren, P. Liu, C. Wang, and C. Wang. BinQuasi: a peak detection method for ChIP-seq data with biological replicates. *Bioinformatics*, 1:9, 2018.
- [38] S. Griffiths-Jones, A. Bateman, M. Marshall, A. Khanna, and S. R. Eddy. Rfam: an RNA family database. *Nucleic Acids Research*, 31(1):439–441, 2003.
- [39] A. R. Gruber, R. Lorenz, S. H. Bernhart, R. Neuböck, and I. L. Hofacker. The vienna RNA websuite. *Nucleic acids research*, 36(suppl 2):W70–W74, 2008.
- [40] J. Hartung. A Note on Combining Dependent Tests of Significance. *Biometrical Journal*, 41(7):849–855, 1999.
- [41] N. Hecker, T. Wiegels, and A. E. Torda. RNA secondary structure diagrams for very large molecules: RNAfdl. *Bioinformatics*, 29(22):2941–2942, 2013.
- [42] L. V. Hedges and I. Olkin. *Statistical methods for meta-analysis*. Academic Press, 1985.
- [43] I. L. Hofacker, W. Fontana, P. F. Stadler, L. S. Bonhoeffer, M. Tacker, and P. Schuster. Fast folding and comparison of RNA secondary structures. *Monatshefte für Chemie/Chemical Monthly*, 125(2):167–188, 1994.
- [44] S. Hoffmann, C. Otto, S. Kurtz, C. M. Sharma, P. Khaitovich, J. Vogel, P. F. Stadler, and J. Hackermüller. Fast mapping of short sequences with mismatches, insertions and deletions using index structures. *PLoS Computational Biology*, 5(9):e1000502, Sep 2009.
- [45] S. Holm. A Simple Sequentially Rejective Multiple Test Procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.
- [46] P. Humburg. *ChIPsim: Simulation of ChIP-seq experiments*, 2011. R package version 1.18.0.

- [47] W. J. Kent, C. W. Sugnet, T. S. Furey, K. M. Roskin, T. H. Pringle, A. M. Zahler, and D. Haussler. The human genome browser at UCSC. *Genome research*, 12(6):996–1006, 2002.
- [48] P. Kerpedjiev, S. Hammer, and I. L. Hofacker. Forna (force-directed RNA): Simple and effective online RNA secondary structure diagrams. *Bioinformatics*, 31(20):3377–3379, 2015.
- [49] H. Koohy, T. A. Down, M. Spivakov, and T. Hubbard. A Comparison of Peak Callers Used for DNase-Seq Data. *PLoS ONE*, 9(5):e96303, 05 2014.
- [50] D. Lai, J. R. Proctor, J. Y. A. Zhu, and I. M. Meyer. R-CHIE: a web server and R package for visualizing RNA secondary structures. *Nucleic acids research*, page gks241, 2012.
- [51] E. S. Lander, L. M. Linton, B. Birren, and et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, Feb 2001.
- [52] J. Leydold and P. F. Stadler. Minimal Cycle Basis of Outerplanar Graphs. *Elec. J. Comb.*, 5:209–222 [R16: 14 p.], 1998.
- [53] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, and 1000 Genome Project Data Processing Subgroup. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–9, Aug 2009.
- [54] R. Lorenz, S. H. Bernhart, C. H. Zu Siederdissen, H. Tafer, C. Flamm, P. F. Stadler, and I. L. Hofacker. ViennaRNA Package 2.0. *Algorithms for Molecular Biology*, 6(1):26, 2011.
- [55] K. Luger, A. W. Mäder, R. K. Richmond, D. F. Sargent, and T. J. Richmond. Crystal structure of the nucleosome core particle at 2.8 Å resolution. *Nature*, 389(6648):251–260, 1997.
- [56] A. T. Lun and G. K. Smyth. csaw: a Bioconductor package for differential binding analysis of ChIP-seq data using sliding windows. *Nucleic Acids Research*, 44(5):e45, 2016.
- [57] N. R. Markham and M. Zuker. UNAFold. *Bioinformatics: Structure, Function and Applications*, pages 3–31, 2008.

- 
- [58] T. S. Mikkelsen, M. Ku, D. B. Jaffe, B. Issac, E. Lieberman, G. Giannoukos, P. Alvarez, W. Brockman, T.-K. Kim, R. P. Koche, et al. Genome-wide maps of chromatin state in pluripotent and lineage-committed cells. *Nature*, 448(7153):553–560, 2007.
- [59] G. Muller, C. Gaspin, A. Etienne, and E. Westhof. Automatic display of rna secondary structures. *Bioinformatics*, 9(5):551–561, 1993.
- [60] E. P. Nawrocki, S. W. Burge, A. Bateman, J. Daub, R. Y. Eberhardt, S. R. Eddy, E. W. Floden, P. P. Gardner, T. A. Jones, J. Tate, and R. D. Finn. Rfam 12.0: updates to the RNA families database. *Nucleic Acids Research*, 43(D1):D130–D137, 2015.
- [61] E. P. Nawrocki and S. R. Eddy. Infernal 1.1: 100-fold faster RNA homology searches. *Bioinformatics*, 29(22):2933–2935, 2013.
- [62] H. Nishida, T. Suzuki, S. Kondo, H. Miura, Y.-i. Fujimura, and Y. Hayashizaki. Histone H3 acetylated at lysine 9 in promoter is associated with low nucleosome density in the vicinity of transcription start site in human cell. *Chromosome research*, 14(2):203–211, 2006.
- [63] R. Nussinov, G. Pieczenik, J. R. Griggs, and D. J. Kleitman. Algorithms for loop matchings. *SIAM Journal on Applied mathematics*, 35(1):68–82, 1978.
- [64] Y. Ponty and F. Leclerc. Drawing and Editing the Secondary Structure(s) of RNA. In E. Picardi, editor, *RNA Bioinformatics*, pages 63–100. Springer New York, New York, NY, 2015.
- [65] A. Quigley and P. Eades. Fade: Graph drawing, clustering, and visual abstraction. In *International Symposium on Graph Drawing*, pages 197–210. Springer, 2000.
- [66] J. S. Reuter and D. H. Mathews. RNAstructure: software for RNA secondary structure prediction and analysis. *BMC bioinformatics*, 11(1):129, 2010.
- [67] J. L. Rinn, M. Kertesz, J. K. Wang, S. L. Squazzo, X. Xu, S. A. Brugmann, L. H. Goodnough, J. A. Helms, P. J. Farnham, E. Segal, and H. Y. Chang. Functional Demarcation of Active and Silent Chromatin Domains in Human HOX Loci by Noncoding RNAs. *Cell*, 129(7):1311 – 1323, 2007.



- [68] Roadmap Epigenomics Consortium. Integrative analysis of 111 reference human epigenomes. *Nature*, 518(7539):317–30, Feb 2015.
- [69] M. D. Robinson, D. J. McCarthy, and G. K. Smyth. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26(1):139–140, 2010.
- [70] C. S. Ross-Innes, R. Stark, A. E. Teschendorff, K. A. Holmes, H. R. Ali, M. J. Dunning, G. D. Brown, O. Gojis, I. O. Ellis, A. R. Green, S. Ali, S.-F. Chin, C. Palmieri, C. Caldas, and J. S. Carroll. Differential oestrogen receptor binding is associated with clinical outcome in breast cancer. *Nature*, 481(7381):389–393, Jan. 2012.
- [71] D. Schübeler, D. M. MacAlpine, D. Scalzo, C. Wirbelauer, C. Kooperberg, F. Van Leeuwen, D. E. Gottschling, L. P. O’Neill, B. M. Turner, J. Delrow, et al. The histone modification pattern of active genes revealed through genome-wide chromatin analysis of a higher eukaryote. *Genes & development*, 18(11):1263–1271, 2004.
- [72] M. J. Serra and D. H. Turner. Predicting thermodynamic properties of RNA. In *Methods in enzymology*, volume 259, pages 242–261. Elsevier, 1995.
- [73] B. Shabash and K. C. Wiese. Numerical integration methods and layout improvements in the context of dynamic RNA visualization. *BMC bioinformatics*, 18(1):282, 2017.
- [74] B. Shabash and K. C. Wiese. RNA Visualization: Relevance and the Current State-of-the-Art Focusing on Pseudoknots. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 14(3):696–712, May 2017.
- [75] B. Shneiderman. *The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations*. VL ’96. IEEE Computer Society, Washington, DC, USA, 1996.
- [76] E. L. Sonnhammer and R. Durbin. A dot-matrix program with dynamic threshold control suited for genomic DNA and protein sequence analysis. *Gene*, 167(1):GC1–GC10, 1995.
- [77] J. D. Storey and R. Tibshirani. Statistical significance for genomewide studies. *Proceedings of the National Academy of Sciences*, 100(16):9440–9445, 2003.

- 
- [78] R. Tamassia. *Handbook of Graph Drawing and Visualization*. Discrete Mathematics and Its Applications. CRC Press, 2013.
- [79] D. H. Turner and D. H. Mathews. NNDB: the nearest neighbor parameter database for predicting stability of nucleic acid secondary structure. *Nucleic Acids Res*, 38:D280–D282, 2010.
- [80] Z. Wang, C. Zang, J. A. Rosenfeld, D. E. Schones, A. Barski, S. Cuddapah, K. Cui, T.-Y. Roh, W. Peng, M. Q. Zhang, et al. Combinatorial patterns of histone acetylations and methylations in the human genome. *Nature genetics*, 40(7):897, 2008.
- [81] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2 edition, 2004.
- [82] J. D. Watson and F. H. Crick. The structure of DNA. In *Cold Spring Harbor Symposia on Quantitative Biology*, volume 18, pages 123–131. Cold Spring Harbor Laboratory Press, 1953.
- [83] M. Wattenberg. Arc diagrams: visualizing structure in strings. In *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002.*, pages 110–116, 2002.
- [84] Z. Weinberg and R. R. Breaker. R2R-software to speed the depiction of aesthetic consensus RNA secondary structures. *BMC bioinformatics*, 12(1):3, 2011.
- [85] K. C. Wiese, E. Glen, and A. Vasudevan. jViz.RNA - A Java tool for RNA secondary structure visualization. *IEEE transactions on nanobioscience*, 4(3):212–218, 2005.
- [86] E. G. Wilbanks and M. T. Facciotti. Evaluation of Algorithm Performance in ChIP-Seq Peak Detection. *PLoS ONE*, 5(7):e11471, 07 2010.
- [87] P. R. Wright, A. S. Richter, K. Papenfort, M. Mann, J. Vogel, W. R. Hessa, R. Backofen, and J. Georg. Comparative genomics boosts target prediction for bacterial small RNAs. *Proceedings of the National Academy of Science of the United States of America*, 110(37):E3487–96, Sep 2013.
- [88] J. S. Yi, Y. ah Kang, J. Stasko, and J. Jacko. Toward a Deeper Understanding of the Role of Interaction in Information Visualization. *IEEE*

*Transactions on Visualization and Computer Graphics*, 13(6):1224–1231, Nov 2007.

- [89] Y. Zhang, Y.-H. Lin, T. D. Johnson, L. S. Rozek, and M. A. Sartor. PePr: a peak-calling prioritization pipeline to identify consistent or differential peaks from replicated ChIP-Seq data. *Bioinformatics*, 30(18):2568–75, Sep 2014.
- [90] Y. Zhang, T. Liu, C. A. Meyer, J. Eeckhoute, D. S. Johnson, B. E. Bernstein, C. Nusbaum, R. M. Myers, M. Brown, W. Li, et al. Model-based analysis of ChIP-Seq (MACS). *Genome biology*, 9(9):R137, 2008.
- [91] M. Zuker and D. Sankoff. RNA secondary structures and their prediction. *Bulletin of mathematical biology*, 46(4):591–621, 1984.
- [92] M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic acids research*, 9(1):133–148, 1981.



# List of Figures

2.1	DNA and its structural organization in the nucleus of eukaryotic cells [1]. . . . .	8
2.2	Single stranded pieces of DNA and RNA. . . . .	9
2.3	Watson Crick base pairs . . . . .	9
2.4	Additional possible base pairs occurring in RNA molecules. . . .	10
2.5	From a gene to function: either a region of the DNA is transcribed to an mRNA, which is translated into a functional protein or the DNA is transcribed into a functional ncRNA [9]. . . . .	11
2.6	Overview of all amino acids that are encoded by codons. The visualization is read from the center of the circle outwards so that each sequence of nucleotides results in an amino acid or a start/stop codon. . . . .	12
2.7	The <i>Central Dogma of Molecular Biology</i> proposed by Crick [30, 29]. The solid arrows show the common information flow and the dashed arrows specialized transfer directions. . . . .	13
2.8	Schematic structure of a nucleosome [13]. . . . .	15
2.9	A nucleosome divided into its components. This model was generated from the crystal structure published by Luger et al. [55]. . . . .	16
2.10	Overview of known histone modifications for the histones H4, H2A, H3, and H2B [14]. . . . .	17
2.11	A model of the histone H3 generated from the crystal structure published by Luger et al. [55]. . . . .	18
2.12	Overview of the substructures of an RNA secondary structure. E: exterior loop, I & B: internal loop, M: multi-loop, H: hairpin loop. . . . .	22

2.13	Decomposing an RNA secondary structure into substructures. The folding space $F$ of the nucleotides $i$ and $j$ can be decomposed into a subfolding space of $i + 1$ and $j$ , or into a bifurcation from $i + 1$ to $k - 1$ and from $k + 1$ to $j$ , where $i < k \leq j$ , and $i$ and $k$ form a base pair. . . . .	22
2.14	Graphical representation of the structural decomposition by the Zuker algorithm. Base pairs are shown as arcs, dotted lines represent unpaired subregions. A not fully calculated multi-loop region is visualized as mountain ridge. . . . .	25
3.1	Overview of the multiple-replicate peak-calling process. Phase I: Windows are constructed and single replicate $p$ -values for each window are computed (pentagons) Phase II: From the single $p$ -values, combined $p$ -values are computed by combining windows using the inverse normal method (large pentagon). Phase III: Suitable narrow and broad peaks (white triangle) are computed based on the windows' combined $p$ -values. The magnifying glass symbolizes all points, where a visualization-based quality control is included in the peak-calling process. . . . .	29
3.2	Overview of the multiple-replicate peak-calling process showing the basic steps of multiple-replicate peak-calling for Sierra Platinum, PePr, BinQuasi, MACS-CR (combine replicates approach using MACS as single-experiment peak-caller), and MACS-SA (combine peaks approach using MACS as single-experiment peak-caller). The MACS approaches and Sierra Platinum extract the parameters of the underlying model (squares) from the background data (white circles) and use the model and the experimental data to calculate $p$ -values (pentagons) indicating how significantly enriched the experiment (black circles) is. PePr and BinQuasi generate also a model for the experiment and use both models to calculate $p$ -values. Based on the $p$ -values, peaks (triangles) are calculated. Quality control (magnifier) is provided usually alongside with the peaks. Only Sierra Platinum allows to examine the quality during the peak-calling process, while all other methods only allow to examine the quality of the peaks obtained. . . . .	34

3.3	Example of a boxplot with lower and upper whisker representing the minimum and the maximum in the data, respectively. . . .	43
3.4	Example boxplot figure created by Sierra Platinum with the arranged background. . . . .	44
3.5	Distribution of tags . . . . .	47
3.6	Example of the $p$ -value distribution generated during the single peak-calling step. Red: $p$ -values of one replicate. Orange: median of $p$ -values over all replicates. Please note, that the x-axis shows only the exponent to the basis 10 and that both axes are scaled logarithmically. . . . .	54
3.7	Example of a significant window distribution over all chromosomes. Red: significant windows of one replicate. Orange: median of significant windows over all replicates. . . . .	59
3.8	Top: heatmap of the Pearson's correlations between the replicates. Bottom: checkbox for enabling or disabling the correlation correction while computing the combined $p$ -value. If the checkbox is checked, $\hat{\rho}^*$ is computed as described in Section 3.3.13, otherwise $\hat{\rho}^*$ is set to 0. . . . .	62
3.9	Example of a combined $p$ -value distribution. . . . .	66
3.10	Overview of the options to weight replicates provided by the GUI of Sierra Platinum. The upper part contains one row per replicate showing the replicate identifier (left column), the assigned weight (middle column), and whether the replicate is used (ON) or not (OFF, right column). The weight checkbox on top of the middle column allows for disabling weights altogether (if unchecked). The lower part contains a checkbox that allows to enable or disable the computation of the quality of the peaks ('Enable quality counting', Section 3.3.17) and a drop-down-box that allows to select the $q$ -value correction method (Sections 3.3.10 and 3.3.13). Pressing the button ('Recalculate', bottom right) starts the recomputation. . . . .	67

3.11	Comparison of the final $p$ -value distributions with different weight settings. Each run is assigned a different color according to the color legend below the figure. The $p$ -values of the bins are mapped to the x-axis while the amount of windows having the respective $p$ -value are mapped to the y-axis. A logarithmic scale is used in both cases. Here, three different runs are shown assigned to the colors red (run 1), orange (run 2), and green (run 3). From the $p$ -value distribution alone, run 1 would be preferred over run 3 over run 2. . . . .	68
3.12	Comparison of the overlap of the peaks between the replicates.	71
3.13	Final peak quality boxplots for experiment and background of 6 replicates. . . . .	73
3.14	The server connection window of Sierra Platinum. . . . .	76
3.15	The Server Menu . . . . .	77
3.16	The File Menu showing the sub menu for storing and loading a configuration. . . . .	78
3.17	The settings-tab in the GUI of Sierra Platinum. In the large, middle window, the replicate information is shown. For each replicate, the file names for the experiment and the background are given. In the lower part, the 'Add Replicate' button allows adding additional replicates. Below this button, the parameters for the 'window size' and the 'window offset' can be changed. Further, the 'Job name', the ' $p$ -value cutoff', and the 'number of threads' can be assigned. Finally, to the right, the 'Start' button allows starting a computation and the progress bar to the right of this button shows the progress of the computation. A context sensitive menu in the main window allows to change the files associated with the replicates' experiment and background and to delete replicates. . . . .	79



3.18	A replicate tab in the GUI of Sierra Platinum. Top left: the estimated Poisson distribution for experiment and background (Figure 3.5a). Top middle: the count distribution for experiment and background (Figure 3.5b). Top right: the mapping quality (Figure 3.4). Bottom left: the estimated Poisson distribution for adjusted experiment and background (Figure 3.5c). Bottom middle: the $p$ -value distribution over all windows (Figure 3.6). Bottom right: the distribution of significant windows per chromosome (Figure 3.7). . . . .	81
3.19	The 'Summary' tab in the GUI of Sierra Platinum. Left top: the Pearson Correlation of the replicates. Left bottom: checkbox allowing for enabling or disabling the correlation based correction (Figure 3.8). Right top, middle column: weights affecting the combination of the single replicates for creating the combined peaks. Right top, right column: enabling (ON) or disabling (OFF) a replicate. Right bottom row: two parameters and restarting the computation (Figure 3.10). . . . .	82
3.20	The 'Peak information' tab in the GUI of Sierra Platinum. Left (see also Figure 3.11): the final $p$ -value histogram showing the combined $p$ -values for all windows. Right (see also Figure 3.12b): the percentage of agreement of each replicate with the multiple-replicate result. Three runs with different setting for weights and different replicate combinations are shown. . . . .	83
3.21	The File Menu showing the sub menu for storing and loading the data mapper. . . . .	83
3.22	Overview of the Sierra Platinum Service container. The web server handles the user registrations and starts the Sierra Platinum Server after a valid activation. All necessary user information are stored in an SQLite database. After a successful registration the user can upload his data with the Sierra Platinum Client and start the computation. Afterwards, it is possible to export all results and visualizations within the client. . . . .	87
3.23	Quality measurements for a noise free replicate. . . . .	100
3.24	Quality measurements for a replicate with low sequencing quality	101
3.25	Quality measurements for a replicate with low enrichment . . .	102
3.26	Quality measurements for an under-sequenced replicate . . . .	104
3.27	Quality measurements for an over-sequenced replicate . . . .	105

3.28	Quality measurements for a replicate having the wrong signal .	106
3.29	Evaluation of the $p$ -value cutoff on the result quality for the noise-free data set. . . . .	108
3.30	Recall of the peak-calls using different combinations of window size, window offset, and cutoff. . . . .	110
3.31	Positive predictive value of the peak-calls using different combinations of window size, window offset, and cutoff. . . . .	111
3.32	False discovery rate of the peak-calls using different combinations of window size, window offset, and cutoff. . . . .	112
3.33	Number of peaks found using different combinations of window size, window offset, and cutoff. . . . .	113
3.34	Evaluation of the $q$ -value methods on the noisy, the bad, and the noise-free data set (Section 3.5.7). . . . .	116
3.35	Evaluation results for the noise-free data set (6 replicates). .	118
3.36	Evaluation results for data sets with noise. First column: low sequencing quality. Second column: low enrichment. Third column: too low sequencing depth. Fourth column: too high sequencing depth. First four rows: one bad replicate; three replicates in total. Second four rows: two bad replicates; four replicates in total. . . . .	120
3.37	Evaluation results for quality deficits in some of the data sets. First four rows, left to right: one under-sequenced replicate with low enrichment and low quality, two under-sequenced replicates with low enrichment and low quality, and a mixture of quality inspired by real data for H3K4me3 in embryonic stem cells. Second four rows, left to right: one, two, and three noisy replicates. . . . .	122
3.38	Overview of the possible peak overlaps with three replicates. Only the first overlap (marked with green) is a valid peak overlap and should be counted, the other three (marked with red) are discarded. Similarly, the merging process overlaps the peaks of all four peak-calling methods. . . . .	126
3.39	Agreement of the peak predictions: The overlap of the peaks predicted by Sierra Platinum (blue), MACS-SA (green), MACS-CR (orange), and PePr (red) is shown. . . . .	127

3.40	Genomic location of HOTAIR locus: The peak-calls are shown below the transcript annotation of HOTAIR. Peak-calls for the different modifications and data sets are color coded as shown in Table 3.13. . . . .	134
3.41	Genomic location of Hox-C locus containing the HOTAIR gene: The peak-calls are shown below the transcript annotation of Hox-C. Peak-calls for the different modifications and data sets are color coded as shown in Table 3.13. . . . .	135
3.42	Genomic location of Hox-D locus: The peak-calls are shown below the transcript annotation of Hox-D. Peak-calls for the different modifications and data sets are color coded as shown in Table 3.13. . . . .	136
3.43	Peak coverage for each replicate: For each peak, the number of supporting reads is calculated. Peaks are counted in the categories no supporting reads (red), 1-200 supporting reads (blue), and more than 200 supporting reads (green). The peaks are the H3K4me3 peaks predicted by the data set ESC. Replicate numbers refer to those in Table 3.11. . . . .	139
4.1	Overview of the postscript dot plot generated by ViennaRNA [54] showing base pair probabilities of an RNA. Black squares are used for showing the possibility of two nucleotides forming a base pair. The probability for forming this base pair is encoded in the size of these squares: large squares imply a high probability, while small squares imply a low probability. The diagonal is used as a landmark only. It divides the upper-right triangle showing the consensus probabilities from the lower-left triangle showing the probabilities according to the energetically best solution. .	144
4.2	Overview of the iDotter interface showing the same dot plot as Figure 4.1. The nucleotide sequence is only shown at the borders, if the nucleotides are readable in the current zoom level.	146
4.3	Comparison of the dot plot interfaces after zooming into a sub-sequence. In the postscript view (a), the nucleotide sequence is no longer visible, while in iDotter (b) it stays visible at all borders easing the analysis of sub-sequences. . . . .	147

4.4	Each dot provides details on demand by mouse-over showing a tool tip: element ID, X showing the nucleotide of the column and its position, and Y showing the nucleotide of the row and its position. The (biological) attributes mapped onto 'Size' and 'Color' are application dependent. . . . .	150
4.5	Highlighting dots 4.5a as well as rows and columns 4.5b. . . .	152
4.6	Marking dots and regions of dots (Figure 4.5a) and marking rows and columns (Figure 4.5b) can be combined. . . . .	153
4.7	In contrast to the postscript visualization, iDotter provides choosing the color gradient. Additionally, choosing the highlighting colors for dots ('Selected Dot Color', Figure 4.5a) and columns/rows ('Linemarker Color', Figure 4.5b) is possible. Moreover, it is possible to reset highlighting in the dot plot by pressing the 'Remove Marker' button. . . . .	153
4.8	Representation of a bulge, an internal loop, a hairpin loop, and a multi-loop in iDotter. The highlighted regions annotate the structural elements. . . . .	155
4.9	The predicted energetically best solution for the RNA shown in Figure 4.2 lower left visualized as graph using RNApuzzler. '1' annotates a bulge, '2' annotates an internal loop, '3' annotates a hairpin loop, and '4' annotates a multi-loop. . . . .	156
5.1	Different elements of the RNA secondary structure (a) and the corresponding fine grained (b) and coarse grained (c) trees as well as the corresponding geometries (d). . . . .	163
5.2	The turtle starts at $P_1$ and walks a fixed distance to $P_2$ . There, it changes its direction by an angle $\alpha_2$ and walks a fixed distance to $P_3$ . . . . .	170
5.3	The turtle walks through a loop: First, the radius of the loop is approximated. Then, the turtle changes its direction based on the arc segment that the current and the next position form in the loop. Please note, that the turtle walks on the chords of the loop. The arcs are drawn during a post-processing step. . .	171

5.4	The RNAfold-predicted minimum free energy structure of the Magn_3109 RNA of <i>Magnetospirillum magnetotacticum</i> sequence, retrieved from RFAM [38]. (a) Drawing generated by RNAturtle. (b) Tree structure superimposed on the RNAturtle drawing. (c) Extracted tree only. . . . .	172
5.5	Any sibling intersection can be resolved (inefficiently) by only increasing the radius of the common multi-loop. Changing the angles of outgoing stems up to a certain degree to resolve intersections in addition to increasing the loop's radius improves the efficiency of the approach. (a) A sibling intersection between the two stems of the multi-loop. (b) The sibling intersection is resolved by only increasing the radius of the multi-loop. (c) The sibling intersection is resolved by changing the angles of the outgoing stems in addition to increasing the radius of the multi-loop. . . . .	177
5.6	Ancestor intersection (a) and its resolution (b). . . . .	178
5.7	Exterior intersection (a) and its resolution (b). . . . .	182
5.8	Siblings intersection (a) and its resolution (b). . . . .	184
5.9	Exterior subtree intersection (a) and its resolution by stretching (b) or flipping (c). . . . .	186
5.10	Comparison of the human and the gorilla SSU rRNA drawn using RNApuzzler without and with optimization. . . . .	190
5.11	Drawings of the secondary structure of the tRNA-Leu (trnL) gene and trnL-trnF intergenic spacer from the chloroplast of <i>Streptocarpus papangae</i> isolate S106 (FJ501444.1/1-447 retrieved from RF00028 [38]). . . . .	194
5.12	Comparison of the human and gorilla SSU rRNA. (a) Secondary structure of AADC01167538.1/2439-571 Homo sapiens drawn with NAView. (b) Secondary structure of AADC01167538.1/2439-571 Homo sapiens drawn with RNApuzzler. (c) Secondary structure of CABD02136541.1/885-2750 Gorilla gorilla drawn with NAView. (d) Secondary structure of CABD02136541.1/885-2750 Gorilla gorilla drawn with RNApuzzler. . . . .	195
5.13	Comparison of the human (a) and the gorilla (b) SSU rRNA drawn using Forna [48]. . . . .	197

---

5.14	Comparison of the human (a) and the gorilla (b) SSU rRNA drawn using jViz.RNA 2.0 [85]. Edge crossings are annotated with arrows. . . . .	199
5.15	Comparison of the human (a) and the gorilla (b) SSU rRNA drawn using jViz.RNA 4.0 [73]. . . . .	200
5.16	Comparison of the human (a) and the gorilla (b) SSU rRNA drawn using RNAfdl [41]. . . . .	201
5.17	Comparison of the human (a) and the gorilla (b) SSU rRNA drawn using the VARNA radial layout [64]. . . . .	203
5.18	Comparison of the human (a) and the gorilla (b) SSU rRNA drawn using the VARNA NAView layout (see also Bruccoleri and Heinrich [22]). . . . .	205
5.19	Comparison of the human (a) and the gorilla (b) SSU rRNA drawn using RNAstructure [66]). . . . .	206
5.20	Example of a non-planar drawing produced by PseudoViewer 3	207
5.21	Comparison of the human (a) and the gorilla (b) SSU rRNA drawn using PseudoViewer 3 [23]). . . . .	208

# List of Tables

3.1	The steps performed by Sierra Platinum: section and name of the step, and whether it is part of the method, a quality measure, or a visualization, respectively. . . . .	30
3.2	Constructing all windows for 6 replicates: time for different strategies . . . . .	38
3.3	The time needed for constructing all windows for 6 replicates using the strategy Chunk Parallel Coherent with different thread pool sizes . . . . .	42
3.4	Computing window neighborhoods and $p$ -values for 6 replicates and 3 different neighborhood sizes: time for different strategies. . . . .	52
3.5	Time needed for transforming $p$ - to $q$ -values using different strategies . . . . .	57
3.6	Computing Pearson's correlation for 6 replicates: time for different strategies. . . . .	63
3.7	Chromosome names and lengths . . . . .	92
3.8	Parameter settings for the different levels of quality. See Section 3.5.5 for a description of the parameters. . . . .	94
3.9	Overview of the data sets used for H1. IDs are only used internally to distinguish the different replicates available for H1. For each replicate, the sequencing center, which produced the data, and the GEO identifications are provided for the data sets H3K4me3, H3K27me3, H3K9me3, H3K27ac, H3K9ac, and ChIP-input. . . . .	129
3.10	Decisions based on visual inspection of the quality of the replicates listed in Table 3.9. <i>ID</i> : ID given Table 3.9, <i>Weight</i> : weight used, <i>off</i> : replicate excluded, <i>empty cell</i> : replicate not available. . . . .	130

---

3.11	Overview of the data sets used for ESC: IDs are only used internally to distinguish between the different replicates available for embryonic stem cell lines. For each replicate, the sequencing center, which produced the data, and the GEO identifications are provided for the data sets H3K4me3, H3K27me3, H3K9me3, H3K9ac, and the ChIP-input. . . . .	131
3.12	Decisions based on visual inspection of the quality of the replicates listed in Table 3.11. <i>ID</i> : ID given Table 3.11, <i>Weight</i> : weight used, <i>off</i> : replicate excluded, <i>empty cell</i> : replicate not available. . . . .	132
3.13	Color coding used in the figures showing the peaks as UCSC tracks at selected genomic positions. . . . .	133
5.1	Benchmark results comparing folding and drawing time for different RFAM families. . . . .	191
5.2	Constraints met by different algorithms: ✓: met, ✱: not fully met, ? : unclear, ✕: not met . . . . .	192





# Daniel Wiegreffe

## *Curriculum Scientiae*

---

### Personal Information

Fullname Daniel Wiegreffe  
Born Name Gerighausen  
Date of Birth 14.10.1989  
Place of Birth Lingen (Ems)

---

### Education

03/2015 **Master of Science, Computer Science, Universität Leipzig**,  
Thesis Title: TiBi-3D - a Guide through the World of Epigenetics,  
Supervisor: Prof. Scheuermann, Prof. Prohaska.  
10/2013 – **Study of Master of Science, Computer Science, Universität**  
03/2015 *Leipzig*.  
09/2013 **Bachelor of Science, Computer Science, Universität Leipzig**,  
Thesis Title: Kombination von K-means++ Clustering und  
PCA zur Analyse von Chromatin-Daten, Supervisor: Prof.  
Scheuermann.  
10/2009 – **Study of Bachelor of Science, Computer Science, Univer-**  
09/2013 *sität Leipzig*.

---

### Academic Appointments

since 08/2018 **Project Coordinator, Universität Leipzig**,  
Coordination of the EFRE project "Data Mining and Value  
Added".  
4/2015 – **PhD student, Universität Leipzig**,  
07/2018 Developing new visualizations for Big Data from biological  
sources.

- 4/2013 – **Student Assistant**, *Universität Leipzig*,  
 12/2014 Assistance in the 'ChromatinVis' project, Supervisor: Prof. Scheuermann, Prof. J. Prohaska, Dr. Zeckzer.

### Publications related to the Dissertation

- 2018 **Method Paper**, *RNApuzzler: Efficient Outerplanar Drawing of RNA-Secondary Structures*, Wiegrefe, Alexander, Stadler, Zeckzer, Oxford Bioinformatics.
- 2018 **Research Note**, *The Sierra Platinum Service for generating peak-calls for replicated ChIP-seq experiments*, Wiegrefe, Müller, Steuck, Zeckzer, Stadler, BMC Research Notes.
- 2017 **Application Paper**, *iDotter - an interactive dot plot viewer*, Gerighausen, Hausdorf, Zänker, Zeckzer, WSCG 2017 - Pilsen.
- 2016 **Method Paper**, *Sierra platinum: a fast and robust peak-caller for replicated ChIP-seq experiments with visual quality-control and -steering*, Müller, Gerighausen, Farman, Zeckzer, BMC Bioinformatics.

### Publications

- 2019 **Summary Paper**, *Big Data Competence Center ScaDS Dresden/Leipzig: Overview and selected research activities*, Rahm, Nagel, Peukert, Jäkel, Gärtner, Stadler, Wiegrefe, Zeckzer, Lehner, Datenbanken-Spektrum (accepted).
- 2018 **Application Paper**, *Analyzing Histone Modifications Using Tiled Binned Clustering and 3D Scatter Plots*, Zeckzer, Wiegrefe, Müller, WSCG 2018 - Pilsen.
- 2016 **Method Paper**, *Analyzing Histone Modifications in iPS Cells Using Tiled Binned 3D Scatter Plots*, Zeckzer, Gerighausen, Müller, BDVA 2016 - Sydney.
- 2016 **Research Paper**, *A consensus network of gene regulatory factors in the human frontal lobe*, Berto, Perdomo-Sebogal, Gerighausen, Qin, Nowick, Frontiers in Genetics.
- 2014 **Short Paper**, *Using Significant Word Co-occurrences for the Lexical Access Problem*, Feist, Gerighausen, Konrad, Richter, Eckart, Goldhahn, Quasthoff, CogAlex Workshop 2014.
- 2014 **Method Paper**, *Analyzing Chromatin Using Tiled Binned Scatterplot Matrices*, Zeckzer, Gerighausen, Steiner, J. Prohaska, BioVis 2014 – Boston.

## Conferences

- 2018 **Conference Talk**, *Introduction of the Supergenome Browser*, 33rd TBI Winterseminar in Bled (Slovenia).
- 2017 **Conference Talk**, *RNApuzzler IV*, 15th Bioinf Herbstseminar in Doubice (Czech republic).
- 2017 **Poster**, *A fast and robust peak-caller for replicated ChIP-seq experiments with visual quality-control and -steering*, Müller, Gerighausen, Farman, Zeckzer, at BiVi 2017 - Edinburgh (United Kingdom).
- 2016 **Conference Talk**, *RNApuzzler II*, 31st TBI Winterseminar in Bled (Slovenia).
- 2015 **Conference Talk**, *RNApuzzler*, 13th Bioinf Herbstseminar in Doubice (Czech republic).
- 2014 **Conference Talk**, *iDotter - An interactive dotplot viewer*, 12th Bioinf Herbstseminar in Doubice (Czech republic).
- 2014 **Poster**, *ChromatinVis: a tool for analyzing epigenetic data*, Gerighausen, Zeckzer, Steiner, J. Prohaska, at Vizbi 2014 - Heidelberg.
- 2014 **Conference Talk**, *Clustering to the power of 2*, Application of clustering algorithms on epigenetic data, 29th TBI Winterseminar in Bled (Slovenia).
- 2013 **Conference Talk**, *New visualizations of chromatin data*, 11th Bioinf Herbstseminar in Doubice (Czech republic).

## Teaching Experience

- 04/2016 – **Teaching Assistant**, *Universität Leipzig*,
- 07/2016 'Praktikum Objektorientierte Programmierung', Lecturer: Dr. Zeckzer.
- 10/2015 – **Teaching Assistant**, *Universität Leipzig*,
- 02/2016 'Modellierung und Programmierung I', Lecturer: Prof. Prohaska.
- 10/2012 – **Student Assistant**, *Universität Leipzig*,
- 02/2013 'Grundlagen der Informatik und Numerik', Lecturer: Dr. Meiler.
- 09/2012 – **Student Assistant**, *Universität Leipzig*,
- 09/2012 Remedial course 'c programming', Lecturer: Dr. Meiler.
- 05/2012 – **Student Assistant**, *Universität Leipzig*,
- 07/2012 'Praktikum Objektorientierte Programmierung', Lecturer: Dr. Meiler.

## Co-Supervised Bachelor- and Masterthesis

- 2018 **Bachelorthesis**, *Visualization of 3D Tiled Binned Scatter Plots Using a Virtual Reality Enviroment*, Michelle Kampfrath.
- 2018 **Masterthesis**, *segemehlQC - A quality control tool for mapped NGS data*, Marcel Winter.
- 2018 **Bachelorthesis**, *Sierra Platinum as a Webservice*, Matthias Haeßner.
- 2018 **Masterthesis**, *Visualization of synteny-based orthology data*, Yuan Peng.
- 2017 **Masterthesis**, *Planares Zeichnen von RNA-Sekundärstrukturen*, Daniel Alexander.
- 2017 **Bachelorthesis**, *Visualisierung von Chromatindaten in Virtual Reality*, Dominik Michael.
- 2017 **Masterthesis**, *Masakari - Algorithms for Segmenting Chromatin*, Alrik Hausdorf & Nicole Hinzmann.
- 2016 **Bachelorthesis**, *Tiled-Binned Clustering of Multi-Variate Data*, Karl Kaiser.
- 2016 **Bachelorthesis**, *Zeichnen von RNA-Sekundärstrukturen mittels Konfigurationen im ViennaRNA Package*, Daniel Alexander.
- 2016 **Bachelorthesis**, *iDotter - a RNA Dot Plot Viewer*, Sebastian Zänker.
- 2014 **Bachelorthesis**, *Analyse von Chromatin durch den k-Median und das Consensus Clustering*, Daniel Abitz.

## Programming skills

Java, R, C, C++, and Perl  
advanced knowledge of Linux and Bash  
advanced knowledge of LaTeX and BibTeX

## Languages

German  
English





# Eidesstattliche Erklärung

Hiermit erkläre ich, die vorliegende Dissertation selbstständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialien oder erbrachten Dienstleistungen als solche gekennzeichnet.

Leipzig, den 20. Dezember 2018

Daniel Wiegrefe