UiO : Department of Informatics
University of Oslo

Finding Topologically Associating Domains

Revealing hidden patterns in the three-dimensional structure of DNA

Ivar Grytten

Master's Thesis Spring 2015

Finding Topologically Associating Domains

Ivar Grytten

4th May 2015

Abstract

In this thesis, we propose a new method for finding *Topologically Associating Domains*, which are contiguous segments of chromatin, ranging in size from thousands to millions of base pairs. These domains, which are apparent throughout most of the genome, have been postulated as being fundamental building blocks of higher-order genome structure, and being linked to the biological function of the DNA.

Our method uses Hi-C interaction matrices that describe the interaction frequency between pairs of loci. The method produces a set of hierarchically nested domains, and a set of non-overlapping consensus domains — both of which can be used in further biological analyses. We made our method and domains accessible by creating three tools in the Genomic HyperBrowser. These tools can be used to create domain sets, to visualize domains with the Hi-C data, and to compare and analyse domain sets. We analyse the association between the domains and CTCF binding sites, and compare domains found in the human genome with those found in the mouse genome. We discuss how these types of analysis have been performed by others, and propose alternative ways of performing them.

Our domains are similar to those found by others, but they are more self-interacting and interact less with their surroundings. Based on the strong self-interacting nature of our domains, and their association with biological features, we argue that we find a preferable set of domains.

Acknowledgements

I would first and foremost like to thank my supervisors *Geir Kjetil Sandve* and *Jonas Paulsen* for patient guidance, thorough feedback, and helpful discussions throughout the work of this thesis. Also, gratitude goes to *Sveinung Gundersen*, who helped me with questions related to the Genomic HyperBrowser, and to *Karin Lagesen*, who gave helpful advice on the structure and disposition of this thesis.

I would also like to thank the staff and the other master students at the Biomedical Informatics Research Group at the Department of Informatics, who have made this period a memorable time.

Some of the computations performed during the work on this thesis were performed on the Abel Cluster, owned by the University of Oslo and the Norwegian Metacenter for Computational Science (NOTUR), and operated by the Department for Research Computing at USIT, the University of Oslo IT-department [1].

Ivar Grytten
University of Oslo
May, 2015

[1] http://www.hpc.uio.no/

Glossary and abbreviations

A: The Hi-C interaction frequency matrix

$\mathbf{A}_{i,j}$: The interaction frequency between bin i and bin j

Average intra-domain interaction frequency: Mean interaction frequency within a domain

Area on the genome: Some closed interval of connected bins on the genome

Bin: A group of contiguous base pairs on the genome. Unless otherwise noted, a bin will consist of 40 000 base pairs.

bp: base pair (of DNA)

Domain: The terms *domain*, *topologically associating domain* and *topological domain* are used interchangeably.

Domain borders: The first and last bin in a domain

Domain boundary: Segment between two domains

Domain density: Mean interaction frequency within a domain

Domain edges: The same as *domain borders*

Downstream and upstream: Directions on the DNA. Related to the Hi-C data matrix, downstream is to the right and upstream is to the left.

hES: Human embryonic stem cell

IMR90: A human cell line derived from lung tissue

Intra-domain interactions: Interactions between loci inside the same domain

Inter-domain interactions: Interactions that loci inside a domain have with loci outside the domain

mES: Mouse embryonic stem cell

Locus (pl. loci): A position on the genome

TAD: Topologically associating domain. The exact definition is discussed throughout the thesis.

Contents

Chapter 1

Introduction

Since Darwin proposed the theory of evolution by natural selection [1], there has been an increasing interest in understanding how characteristics of living species are encoded and expressed. In 1953, Watson and Crick discovered the double helix structure of DNA, solving the mystery about how genetic information is stored in living organisms. In the 1980s, techniques allowing automated genome sequencing were developed, leading to a 90 % complete sequence of the human genome being published in 2001 [2, 3]. With the genetic sequence of the human genome being available online, and an increasing amount of computational power available, it has has been possible for researchers to link characteristics and diseases to the sequence.

In recent years, another aspect of DNA has achieved attention: how it is folded and structured in three-dimensional space. New techniques have made it possible to probe this structure, making it possible to investigate how biological features are connected to the topology of the genome — not only to the genetic sequence. One of the more recent methods is Hi-C [4], which produces a data matrix of interaction frequencies between every part of the genome at a certain resolution.

From the data generated by Hi-C, it has been discovered that DNA groups together in spatially compact clusters, termed *Topologically Associating Domains* (TADs). These domains, ranging in size from thousands to millions of kilobase pairs, cover most of the genome, and are related to biological features as well as being preserved across cell types [5]. Interestingly, the same domains found in the human genome are also found in the mouse genome, meaning that these domains have been preserved throughout evolution and may be fundamental building blocks of the genome.

1.1 Aims for the thesis

The main aim of this thesis is to provide an improved method for finding TADs based on Hi-C data. For a method to be an improvement, it should:

- given an understanding of what TADs are, discover a set of TADs that fits better with these definitions than the sets found using existing methods.
- return TADs that, when using objective quantitative ways of comparing with previous sets of TADs, generally score as well or higher than the TADs found using existing methods.
- not be significantly slower or more complex than existing methods

The subgoals of this aim are to:

Provide a better model for understanding TADs. In order to provide an improved method for finding TADs, one also needs to have a better understanding of what a TAD really is. The term TAD is frequently used in the literature without being fully explained, and no formal definition exists. The goal is to formalize and conceptualize what TADs really are, through developing a method.

Analyse the set of TADs we find and compare it with previously found sets. As part of evaluating a new method, several types of analyses can be performed. Some inspiration can be drawn from previous attempts to find these domains, but since this is a new and emerging field of research, efforts should be made to assess what kinds of analysis are relevant and whether there are different and new types of comparisons and analyses that can be used.

Visualize and share the results. An important part of providing a better understanding of what TADs are, is to make the method, results and analyses reproducible and accessible. Thus, we wish to utilize the power of the Genomic HyperBrowser [6], and make a set of tools that present our results.

The tools should present the results through visualization of the data and domains, and should make it possible to reproduce the results and compare sets of TADs.

1.2 Overview of chapters

Chapter 2 contains background material relevant for the rest of the thesis. Emphasis is placed on previous methods for finding TADs.

In **Chapter 3** we propose methods to solve the main problem in this thesis, finding a new set of TADs. We first briefly present some background work, e.g. how the data are visualized, before presenting and discussing three different approaches for finding TADs. They are presented in the order they were developed throughout the work of this thesis. Since weaknesses and limitations of each approach motivate the next approach, a brief discussion of each is also included. After presenting different methods, we choose one of them, and create a consensus set of domains. At the end of the chapter, we present the main methods we wish to use for analysing the domains.

In **Chapter 4** we present the resulting set of domains, analyse the domain set, and compare the domains to previously found domains. This chapter also includes a discussion of the results.

In **Chapter 5** we summarize and conclude.

The **appendices** contain further details. In **Appendix A** we explain *hidden Markov models*. **Appendix B** provides details about the source code and how the analyses in Chapter 4 were performed.

As part of this thesis, a *galaxy page* and some tools were created in the Genomic HyperBrowser. Some of the Figures presented can also be found as interactive figures on this page: https://hyperbrowser.uio.no/3dml/u/ivar/p/master under the *Figures* history. If a Figure is available online, a HyperBrowser history element number is given below the Figure in this thesis.

Chapter 2

Background

2.1 DNA

All the genetic information of any living species is stored in *deoxyribonucleic acid* (DNA). In humans, the DNA in each cell is about 2 meters long if stretched out, and consists of approximately 6 billion bases. The basic building blocks of DNA are called *nucleotides* — each consisting of two parts, a sugar and a nucleobase. There are four different types of nucleobases: adenine (A), guanine (G), cytosine (C) and thymine (T). These are often referred to as bases, and since DNA consists of two strands of nucleotides, the units are often called *base pairs* (bp) [7].

A series of the abbreviations of the base molecules are commonly used to denote a *DNA sequence*, e.g. *ATCTGCAC*. The DNA string is tightly packed into units called *chromosomes*. In humans there are 46 of them (often referred to as 23 pairs). The order of the base pairs in the DNA sequence differs between individuals of the same species, and contains information that is used to encode proteins — influencing the characteristics of the species it is in. Thus, this sequence is of great interest for biologists trying to figure out how characteristics of an individual are expressed and inherited.

Gene is a term used to describe some stretches of the DNA. These special stretches each influence a particular characteristic of the individual, and are inherited from parent to child.

2.2 The 3D organization of the genome

Since the time when chromosomes were discovered in the 19th century, biologists have been investigating their three-dimensional structure, in order to understand how this relates to gene expression and the transition of genes. For the first time, in 2001, the human genome was sequenced [2, 3], and much of the focus shifted to analysing the genomic sequence. However, during the last decade, advanced methods that are able to probe the chromosome organization have given biologists a huge amount of new data describing the 3D organization of the genome, bringing the focus back to studying the topology of the genome [8].

2.2.1 Chromosome conformation capture

Chromosome conformation capture (3C), invented by Dekker et al. in 2002 [9], made it possible to map interactions between parts of the genome in three-dimensional space. In the wake of this invention, improved versions were developed, one of them being the Hi-C method, introduced by Lieberman-Aiden in 2009 [4].

2.2.2 Hi-C

Hi-C produces data matrices that describe the spatial relationship between parts of the genome in three-dimensional space. It is performed by first *cross-linking* the chromatin using formaldehyde, connecting parts of the DNA that are spatially close together. Then, the DNA is cut into bins[1] at the given resolution, e.g. 40 kb, and the loose ends of the cross-linked DNA are ligated together. We now have a set of pairs of connected DNA fragments, which are identified by *pair-ended sequencing*. See Figure 2.1 for an illustration of the main steps of Hi-C. We refer to [4] for a more detailed explanation of the technique.

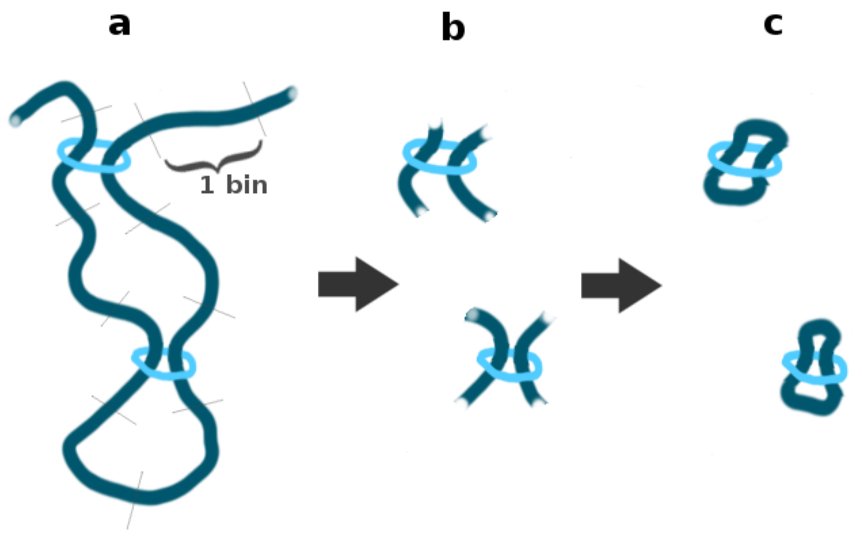


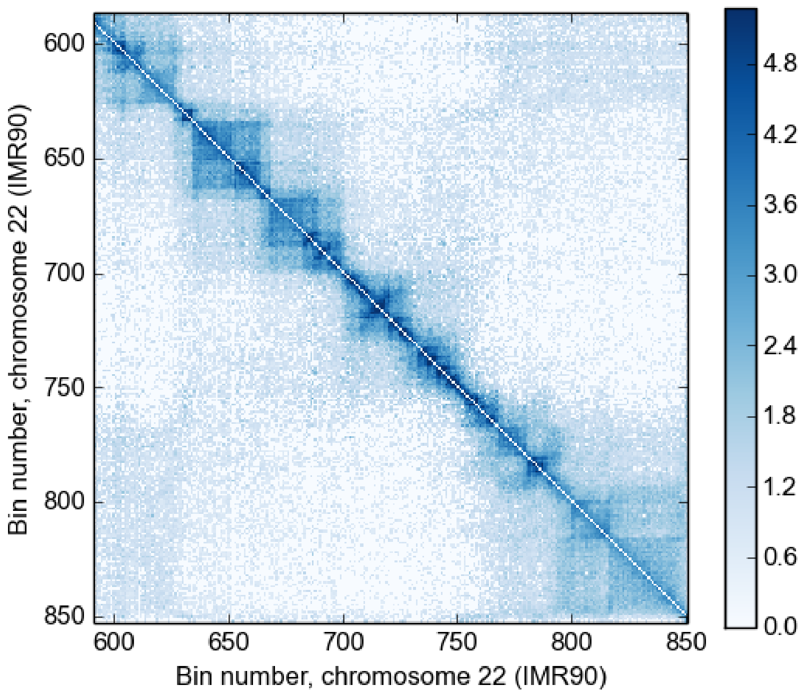
Figure 2.1: The main steps of Hi-C. The DNA is first cross-linked, linking together bins that are spatially close (a). Then, the DNA is cut with a restriction enzyme (b) and the cross-linked parts are ligated (c).

A population of cells is used. The output of the Hi-C method is a matrix A, where $A_{i,j}$ is a count of how many times a part of bin i interacted with a part of bin j

[1] *Bin* is used to denote a contiguous segment of the genome, e.g. 40 kb.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & \cdots & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix}$$

(a)



(b)

Figure 2.2: Illustration of the mathematical notation of the data matrix (a) and a heat map of a submatrix of a real data matrix (b)

in the sample of cells. The number does not explicitly say how close the bins are. Since cross-linking occurs only between loci that are close together, two loci with a low interaction frequency may be pretty close, but not close enough. Neither do we know the variance of the interaction frequency, i.e. two bins with a high interaction frequency may have been far away from each other in some cells, and very close in other cells. Also, since a population of cells is used, underlying bigger structures that are distinct in some cells may be smoothed out over a bigger sample, so that they are not directly visible in the Hi-C data matrix. The matrix A, which is symmetrical with respect to the diagonal, is illustrated in Figure 2.2.

2.2.3 The topology of the genome is related to genomic features

Much work has been done in investigating the relationship between the topology of the genome and genomic features. In the same paper from 2009 in which Lieberman-Aiden et al. proposed Hi-C, they also showed that the genome is organized in large compartments. By using principal component analysis (PCA) on the Hi-C data matrix, they assigned every bin in every chromosome to one of two compartments, A or B. They found that bins inside each compartment tended to interact more with other bins inside the same compartment than with bins in the other compartment. Further, they showed that these compartments were related to genetic and epigenetic features, e.g. compartment A had a higher correlation with the presence of genes than compartment B. These compartments have been extensively studied, see [8] and [10] for detailed reviews.

2.2.4 Topologically associating domains (TADs)

Using higher resolution Hi-C data (1 bin = 40 kb), Dixon et al. (2012) [5] found that these compartments are built up of smaller domains with many *intra-domain*[2] interactions (average size ≈ 900 kb), termed *topologically associating domains* (TADs) or simply topological domains. They showed that pairs of loci within such domains were closer than pairs of loci that were in different domains, concluding that these domains are self-interacting segments of the genome.

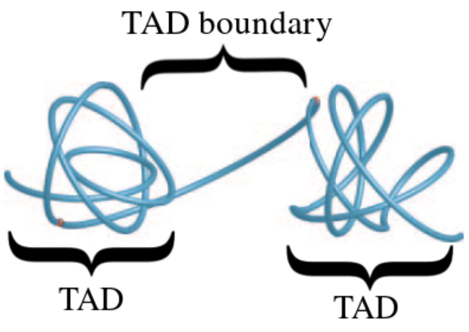


Figure 2.3: Schematic illustration of two TADs and a TAD boundary between them. Image credit: Dixon et al. [5]

Areas between TADs, termed *domain boundaries*, also have important features, including a relation to transcription start sites and *CTCF binding sites*. CTCF is a protein that plays an important role in gene regulation[3]. Dixon et al. concluded that the domain boundaries they found were enriched with CTCF binding sites, indicating that these areas might act as insulators, blocking interactions between enhancers and promoters, i.e. linked to activation of genes.

Nora et al. (2012) [12] performed an experiment in which they deleted one of the domain boundaries in the X chromosome inactivation centre. The result was that the two domains on each side started to interact with each other, and one of the TADs

[2]With *intra-domain* interactions, we mean interactions between loci inside the domain.

[3]See [11] for a detailed overview of the role of CTCF.

got reconfigured. In the same article they showed that TADs play an important role in the inactivation of the extra X chromosome in the female mouse.

2.3 Methods for finding TADs

Since TADs are of great interest for biologists, methods for automatically finding these domains have been proposed. In this section we present the main methods that are described in the literature. A schematic representation of the these methods is given in Table 2.1 on page 13.

2.3.1 Dixon et al. (2012)

In 2012, Dixon et al. published a now widely cited method for finding TADs in the genome. They aimed to find domains by first finding *domain edges*[4] by calculating a *directionality index*, which was fed into a hidden Markov model (HMM) (see Section 2.4.1 for details about HMMs). The directionality index for a given bin is based on the ratio between the number of upstream and downstream[5] interactions for that bin:

$$d(i) = \frac{U(i) - D(i)}{|U(i) - D(i)|} \cdot \left(\frac{(U(i) - T(i))^2}{T(i)} + \frac{(D(i) - T(i))^2}{T(i)} \right) \tag{2.1}$$

where

$$U(i) = \sum_{j=1}^{50} A_{i,i-j} \tag{2.2}$$

$$U(i) = \sum_{j=1}^{50} A_{i,i+j} \tag{2.3}$$

$$T(i) = \frac{U(i) + D(i)}{2} \tag{2.4}$$

$d(i)$ will typically be a low negative number when bin i is at the beginning of a domain (compared to a random bin), because the bin at the start of a domain will have many downstream interactions and fewer upstream interactions (assuming the domain has many intra-domain interactions). At the end of the domain, the directionality index will be higher than expected elsewhere. Dixon et al. treat this directionality as an observation of the *true directionality bias* and feed it into a HMM, assuming that it follows a mixture of Gaussians, returning one out of three

[4]The terms *domain edges* and *domain borders* are used interchangeably to denote the first and last bin in a domain.

[5]*Upstream* and *downstream* are used to denote direction on the genome. In relation to the visualization, downstream will be to the *right* and upstream will be to the *left*.

different states for every bin: downstream biased state, upstream biased state or non-biased state.

TADs are then defined as areas starting at a single downstream biased state, continuing throughout any downstream biased or non-biased states, and ending at the last bin in a series of upstream biased states.

2.3.2 Filippova et al. (2014)

In 2014, Filippova et al. [13] proposed an alternative method to that of Dixon et al., using the same data, but with a different approach. In the paper *Identification of alternative topological domains in chromatin*, they first argue that domains are often nested inside other domains, something that is clearly visible in the interaction matrix (for instance, see Figure 3.8 on page 30). This motivates a method that can find domains on different scales[6], since limiting oneself to one scale means omitting a lot of domains.

The problem is formulated as an optimization problem, aiming to find the optimal set of domains given a resolution parameter γ. A score function $q(i, j, \gamma)$ is defined, returning a score for a domain starting at bin i, ending at bin j when the desired resolution is γ:

$$q(i, j, \gamma) = s(i, j, \gamma) - \mu_s(i - j) \tag{2.5}$$

where $s(i, j, \gamma)$ is a weighted sum of the interactions within the domain:

$$s(i, j, \gamma) = \frac{\sum_{g=i}^{l} \sum_{h=g+1} jA_{g,h}}{(i - j)^{\gamma}} \tag{2.6}$$

The nominator in Equation 2.6, which is proportional to the sum of the intra-domain interactions, will typically grow exponentially when the domain grows. The denominator will also grow exponentially as the domain grows, but the growth can be limited by the choice of γ, so that either smaller or bigger domains are favoured. $\mu_s(l)$ is simply the average of Equation 2.6 over all possible domains of size l.

Using this score function, Filippova et al. found the optimal set of non-overlapping domains by optimizing the sum of domain scores for all possible sets of domains:

$$\underset{D_\gamma}{\operatorname{argmax}} \sum_{[i,j] \in D_\gamma} q(i, j, \gamma) \tag{2.7}$$

where D_γ is a set of domains chosen from all possible sets of domains.

Filippova et al. also found a consensus set of domains. The consensus set contains the domains that persist across sets, and is found by selecting the (non-overlapping) domains that most often occur when varying γ.

[6]The term *scale* loosely denotes size. Two domains on different scales will typically be different in size.

2.3.3 Rao et al. (2014)

Later in 2014, as part of a Hi-C study done on new high resolution Hi-C data, Rao et al. proposed their own method for finding TADs [14]. The new in situ Hi-C data have a much higher resolution (1 bin = 1 kb) than those of Dixon et al. (1 bin = 40 kb).The authors aimed to discover domains by finding the squares along the diagonal of the interaction matrix.

The paper points out that even though finding these easily visible squares may seem straightforward, *finding domains is tricky*, partly because *the interaction frequency is declining as one moves away from the diagonal*. The algorithm, which is called *Arrowhead*, makes these squares easier to find by first performing an "arrowhead" transformation of the initial interaction matrix:

$$T_{i,i+d} = \frac{A_{i,i-d} - A_{i,i+d}}{A_{i,i-d} + A_{i,i+d}} \tag{2.8}$$

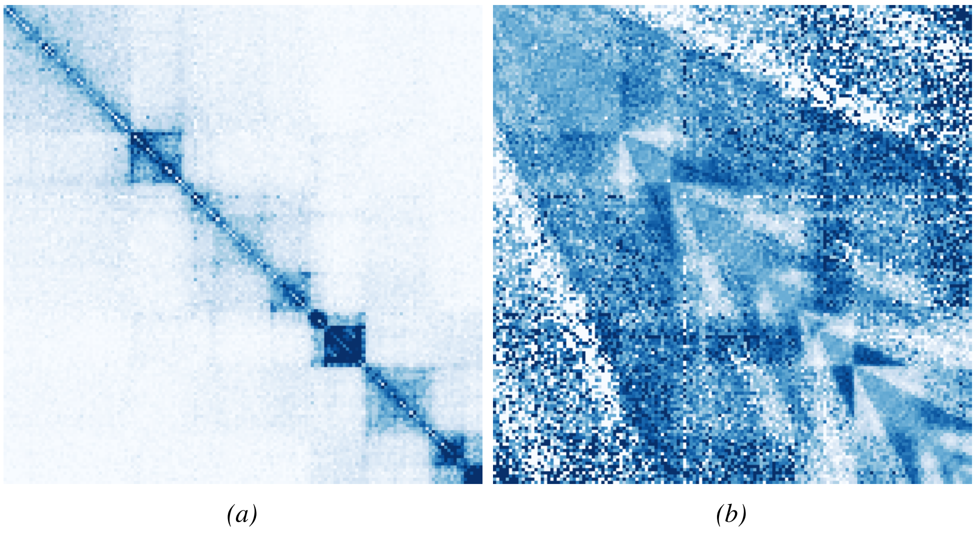


Figure 2.4: Heat map of a submatrix of the raw data matrix A (a) and the same matrix after an arrowhead transformation has been performed (b). Arrowhead-like shapes can be spotted along the diagonal, indicating possible domains.

$T_{i,i+d}$ will be positive if locus $i-d$ is inside a dense domain and locus $i+d$ is outside, and negative in the opposite situation. This leads to arrowhead-like shapes occurring along the diagonal of T, hence the name Arrowhead (Figure 2.4). For a dense domain starting at bin a and ending at bin b, T will take on negative values inside a triangle U with corners $[a,a]$, $[(a+b)/2,b]$ and $[b,b]$, and positive values inside a triangle L with corners $[(a+b)/2,b]$, $[b,b]$ and $[b,2b-a]$. Further, Rao et al. reasoned that if a domain starts at bin a and ends at bin b, then:

- Triangle U will mostly contain negative values and triangle L will mostly contain positive values
- Subtracting the sum of the values in U from the sum of the values in L will result in a high number (relative to doing the same for a non-domain).

- The variance of the values in U and the variance of the values in L are both small (relative to the same for a non-domain).

Domains are then found by looking for pairs of bins a and b that satisfy these three criteria. Based on the three heuristics, a corner score matrix S is calculated. A large value for $S_{a,b}$ means that $[a, b]$ probably is a domain. Domains are chosen by first setting values in S below a threshold to zero, as well as those values representing domains with variance above a certain threshold. This selection is done in two steps: first using thresholds that favour small domains, and second using thresholds that favour bigger domains. In the second step, only domains that do not cover small domains from the first step are selected. After these steps are performed, S is a sparse matrix, with areas of connected non-zero values (representing possible domains). S is then again filtered so that only the highest values in a connected set are kept, representing the domain in the set with the highest score.

2.3.4 Other methods

A few other methods, similar to the ones already listed, are also worth mentioning. Naumova et al. (2013) [15] used a directionality index, similar to the one used by Dixon et al., to find domain edges that demarcate domains. Mizuguchi et al. (Dec 2014) [16] used an *insulation score* in a similar way. The insulation score measures how many interactions there are between bins upstream and downstream of a single bin. A low insulation score indicates that the given bin is likely to be a domain edge. Sexton et al. (2012) [17] defined a *distance-scaling factor* score, smoothed the score and assigned domain edges to the 5 % bins with highest score. The study was done on the fruit fly (Drosophila) genome, and small domains (average size $\approx$ 100 kb) were found.

2.4 Tools and technology

2.4.1 Hidden Markov models

A hidden Markov model (HMM) is a model describing some underlying hidden process that goes through a chain of states at discrete time steps t, and at each state creates output that we are able to observe. HMMs are widely used to model processes in speech recognition and other pattern recognition tasks, but have also been used extensively in the field of bioinformatics [18]. The known parameters are often only some observations at every time step, and the goal is typically to try to find out what states the process has gone through given the observations that have been made.

HMMs play an important role in some of the methods described in this thesis, and understanding HMMs helps us to understand these methods better. Thus, we have devoted Appendix A to describe the mathematics behind HMMs in more detail.

	Dixon et al.	**Filippova et al.**	**Rao et al.**
Input	Hi-C data, resolution 1 bin $= 40kb$	Hi-C data, resolution 1 bin $= 40kb$	Hi-C data, resolution 1 bin $= 1kb$
Assumptions and model	A TAD begins with a bias of downstream interactions, contains bins with non-biased or downstream biased states, and ends on an upstream biased state.	TADs are areas with many intra-domain interactions. They can be nested inside each other and appear at different scales.	TADs are visible as squares in the interaction matrix, and consist of many intra-domain interactions.
Algorithm	The true directionality bias is found by feeding the directionality index into a HMM, which assigns a state (upstream, downstream or non-biased state) to every bin. TADs are inferred from these states.	A score function rates each possible domain (given a scale preference, chosen through a parameter γ), and the optimal set of domains is the set that maximises the sum of the score for every domain in the set. A consensus set is chosen to contain the domains that occur the most when γ is varied.	The squares in the interaction matrix are first made easier to find by doing an arrowhead transformation, emphasizing the corners of the squares. A threshold criterion is used to select the most intense corners, suggesting the optimal domains.
Output	A consensus set of domains with median size $800kb$ covering approximately 90% of the genome.	Domains at a chosen scale, as well as a consensus set of domains with median size of $120kb$ covering approximately 65% of the genome.	Domains with a median size of $180kb$.
Key features	The first proposed method.	Finds domains at different scales.	Using data with higher resolution.

Table 2.1: A schematic representation of different methods for finding TADs.

2.4.2 Machine learning with Python and Scikit-learn

Machine learning is the technique of *teaching* the machine to find patterns in data or to predict an outcome based on observations. When talking about machine learning, the following notation is commonly used:

- Outcome variables, often denoted as Y_i. These variables are what one wants the machine to predict.
- Predictors or explanatory variables, also called *features*. Every outcome variable Y_i has an associated explanatory variable X_i, a vector that is said to *explain* Y_i.

Machine learning can be *supervised* or *unsupervised*. In supervised machine learning, a set of Y_i with associated explanatory variables is known, constituting a *training set*. Based on these known samples, an unknown Y_i can be predicted from a new explanatory variable[7]. In unsupervised machine learning, there is no training set or known outcome variables. Instead of learning from known samples, the goal is to discover patterns in the data or to separate the data only based on the predictor variables. A common way of finding patterns in the data is *clustering*, grouping the data into a set of clusters where the data in each cluster have similar features.

Python is a general-purpose high-level programming language, widely used for scientific computing and machine learning. Its simple syntax usually means that programs written in Python contain fewer lines than equivalent programs written in other widely used programming languages, such as Java and C++.

Scikit-learn [19] is a *machine learning* library for Python, containing tools for performing many common machine learning tasks. A great benefit of using Scikit-learn is that one very quickly can test and play around with different machine learning algorithms, which is often necessary when solving a machine learning problem, since it is not always obvious which algorithm to use. Implementing all algorithms from scratch is time-consuming and error prone. Algorithms implemented in Scikit-learn are used by thousands of scientist every day, well tested and optimized for speed.

2.4.3 The Genomic HyperBrowser

The Genomic HyperBrowser [6] is a web server for analysing genomic data, built on the Galaxy system [20, 21, 22], and freely available at http://hyperbrowser.uio.no.

The Genomic Hyperbrowser consists of several easy-to-use *tools*, available through a web interface, making it easy for users to perform statistical analyses on the genome without the need for programming skills. The whole project is open source,

[7]The most simple machine learning algorithm, a *nearest neighbour classifier*, classifies Y_i to be the same as Y_j where Y_j is the outcome variable from the training set where X_j is most similar to X_i

and the source code is freely available for download, making it easy to extend the HyperBrowser with custom tools. With access, it is also straightforward to implement new tools in one of the existing installations of the HyperBrowser, that run on the Abel Computer Cluster at the University of Oslo.

Custom tools in the Genomic HyperBrowser are implemented in Python, and can print *HTML*[8] that is sent to the browser and displayed as a web page to the user. This makes it possible to create user-friendly figures and graphical and textual representations of the results.

2.4.4 Visualizing data with Javascript and the HighCharts library

A web browser is not only capable of displaying static text and images, but also interactive content with the use of Javascript. Javascript is a programming language that is interpreted by all major web browsers, making it possible to change the content of a web page without needing to refresh the whole page. This makes the language ideal for generating interactive charts and figures.

An ideal Javascript library for generating interactive charts is *HighCharts*[9]. The library is capable of producing a wide range of charts, including heat maps and histograms. The advantage of using HighCharts instead of producing charts server-side as static images and then feeding them to the browser, is that HighCharts makes it possible for the user to interact with the chart, e.g. zooming and panning.

[8]HTML is a mark-up language used to create web pages

[9]http://www.highcharts.com/

Chapter 3

Methods

In this chapter, we develop a new method for finding a set of TADs in a genome, based on the Hi-C data. After presenting different approaches, we choose one and create a consensus set of domains. At the end of the chapter, we present methods for analysing this domain set.

This chapter also includes a brief discussion of the methods as they are proposed. The discussion is focused around how the methods fit with the principles about TADs, and is necessary in order to develop and improve the methods.

When referring to *the data*, we refer to the Hi-C data generated by Ren Lab[1], which is in the form of an interaction matrix A where $A_{i,j}$ is the interaction frequency between bin i and bin j. Most of the examples are based on the human *IMR90*[2] cell line, but mES (mouse embryonic stem cell) and hES (human embryonic stem cell) are also used. The notation $[i,j]$ is used to denote a domain that begins at bin i and ends at bin j.

We begin this chapter by presenting some background work, such as how the data are visualized and how we can select features from the data.

3.1 Visualizing the Hi-C data

We need to visualize the data matrix A. The visualization should:

1. make us able to visually spot possible TADs
2. include as much interesting and relevant information as possible
3. make us able to evaluate easily whether a bin has a bias towards upstream or downstream interactions, the strength of these interactions, and approximately which bins it is interacting with

The data matrix A is square, and since bins generally interact most with bins closer than a fraction of the length of a chromosome, most of the interesting information

[1] http://chromosome.sdsc.edu/mouse/hi-c/download.html

[2] IMR90 is derived from a lung tissue cell in the human body

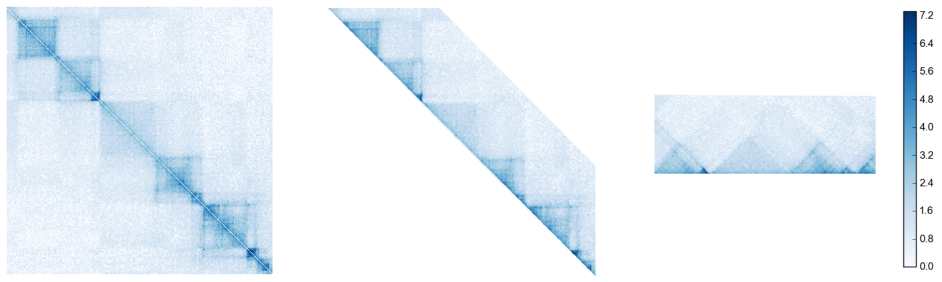


Figure 3.1: Illustration of the visualization technique used by Dixon et al., Filippova et al. and others. To the left: a part of the original data matrix A. In the middle: a sliced part of the upper diagonal. To the right: a rotation of the sliced part, which is the actual visualization used. The area visualized is chromosome 18, bin 1560 to 1830.

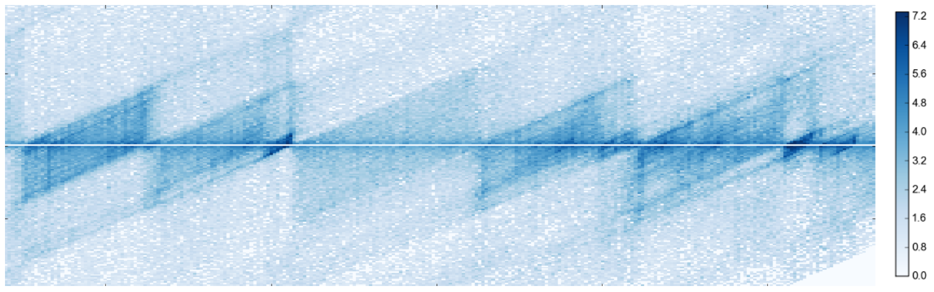


Figure 3.2: Approximately the same area as in Figure 3.1 visualized by the method we propose

in A is situated close to the diagonal. Thus, somehow extracting and visualizing the diagonal is a good way to start. The general way of doing this[3] is to extract the diagonal of A and to visualize it as a heat map. Since A is symmetrical around its diagonal, often only the upper triangle of the matrix is used. The result is a 45 degree rotated segment of the upper (or lower) triangle of A (Figure 3.1).

This method of visualizing the data satisfies the first two criteria listed above, but not the third. Which bins a certain bin is interacting with, and whether it has upstream or downstream interactions are not presented intuitively, and have to be read by following the diagonal from the bin in question.

We propose an improved version of this technique, where we generate a visualization matrix $V \in R^{2n+1,m}$ where m is the number of bins in the area being visualized and n is the range of interactions we are interested in (maximum distance in the data matrix). Each column of V is an interaction vector, where the element in the centre of the vector represents the interaction frequency between the bin and itself[4], the first part of the vector contains upstream interaction frequencies and the last part contains downstream interaction frequencies (Figure 3.2). Mathematically we have $V_{i,j} = A_{i-n+2,j}$.

[3]We refer here to how Dixon et al., Filippova et al, and Rao et al., among others, visualize the data.

[4]The interaction frequency between a bin and itself is not of interest (see Section B.4 for an explanation). We set this frequency to 0 when visualizing.

To accentuate low interaction frequencies, we also do a histogram transformation of the visualization matrix. Every value is raised to α (where $0 < \alpha < 1$).We use $\alpha = 0.4$ since this gives nice plots, as in Figure 3.2. This transformation increases low values more than high values. Outliers, defined as values higher than the mean plus 4 standard deviations, are removed before the histogram transformation.

Our proposed method of visualizing the data is the one used throughout this thesis. When the area of interest is not too large, we often simply visualize the data matrix as a heat map instead. Because of the normalization, colorbars next to figures should not be interpreted directly as the interaction frequency, but should only be used to compare interaction frequencies in a figure.

3.2 Reducing the dimensionality of the data: choosing features

For some of our proposed methods, we need features for bins. Recall matrix A where $A_{i,j}$ is the interaction frequency between bin i and j. There is a huge amount of data in matrix A, and we wish to extract the most interesting data. There are mainly two types of interaction we are interested in:

- *Relative interactions* are interactions measured in relative distance to a bin. For instance, bin i and j will have the same relative interactions if they both only interact with one bin each that is positioned 50 bins upstream of them. The position of the bins they interact with is measured in relative distance.
- *Exact interactions* are interactions with bins measured in exact position. Bin i and bin j will have the same exact interactions if they only interact with bin k. Bin k may be closer to bin i than to bin j. The relative distance is not important.

Exact interactions are interesting to study when we want to find bins that interact with the same areas somewhere else in the genome. Relative interactions are interesting when we want to find bins that have the same interaction patterns, for instance bins that only interact with bins nearby. We will mainly consider relative interactions.

When continuing to analyse the data, we would like to not have too many features, but still have representative features that give us relevant information. A key point here is that few features are wanted if we later want to use a HMM to find patterns or cluster bins with similar features[5]. A look at the heat map of A shows that a lot of the information is located close to the diagonal. Since we are interested in relative interactions, that most frequently occur close to the diagonal at the matrix, we do not need a very high level of detail for the interactions far away from the diagonal. We can group some of them together without loosing too much information. The same can be done close to the diagonal, but a grouping here will result in a greater loss of information.

[5]Few features are generally wanted when using machine learning techniques.

With this argument in mind, we create a new matrix $B \in \mathbb{R}^{NxM}$, where M is the number of bins, and N is the number of intervals we want to group interactions into. For instance, we can create three intervals, the first being the sum of all interactions up to 50 bins upstream, the second being the sum of interactions between 50 bins downstream and 50 bins upstream, and the third being the sum of interactions with all the bins more than 50 bins downstream.

To keep more information (and get more features), even more intervals may be used. Let $a_{l:h,i}$ denote the sum of interaction frequencies in A between bin i and all the bins that are between l and h bins away from bin i, so $a_{l:h,i} = \sum_{j=l}^{h} A_{i+j,i}$ [6]. Using this notation, we can represent B as in Figure 3.3.

$$B = \begin{pmatrix} a_{-50:-30,1} & a_{-50:-30,2} & \cdots & a_{-50:-30,M} \\ a_{-30:-15,1} & a_{-30:-15,2} & \cdots & a_{-30:-15,M} \\ a_{-15:-5,1} & a_{-15:-5,2} & \cdots & a_{-15:-5,M} \\ a_{-5:0,1} & a_{-5:0,2} & \cdots & a_{-5:-0,M} \\ a_{0:5,1} & a_{0:5,2} & \cdots & a_{0:5,M} \\ a_{5:15,1} & a_{5:15,2} & \cdots & a_{5:15,M} \\ a_{15:30,1} & a_{15:30,2} & \cdots & a_{15:30,M} \\ a_{30:50,1} & a_{30:50,2} & \cdots & a_{30:50,M} \end{pmatrix}$$

Figure 3.3: Notation of an example of a feature matrix.

3.3 Developing new methods for finding TADs

We begin by discussing different perspectives that can be used to examine the problem.

Much of the knowledge about TADs concerns the role these domains play in the genome, e.g. their relationship to CTCF and histone modifications, that they are consistent across different cells, and that they appear as a *plaid pattern* in the Hi-C data matrix. We emphasize that this knowledge does not *explicitly* define TADs, it only *implicitly* does so[7]. These results are based on the analyses done on TADs *after* they are found, and should not be used to find TADs. Other characteristics define TADs more explicitly, e.g. that they are self-interacting areas.

Figure 3.4 illustrates the general problem as a flow chart. We know the *input* and the *output*, and need to develop the *method*. In order to develop the method, *rules and knowledge* about the domains have to be used.

From here, having input and some knowledge about TADs, we see two possible approaches:

[6]This equation will give negative indices and indices greater than the number of bins, something that should be kept in mind when implementing the algorithm using a programming language.

[7]This important distinction between explicit and implicit TAD features has been discussed by Lajoie et al. [23].

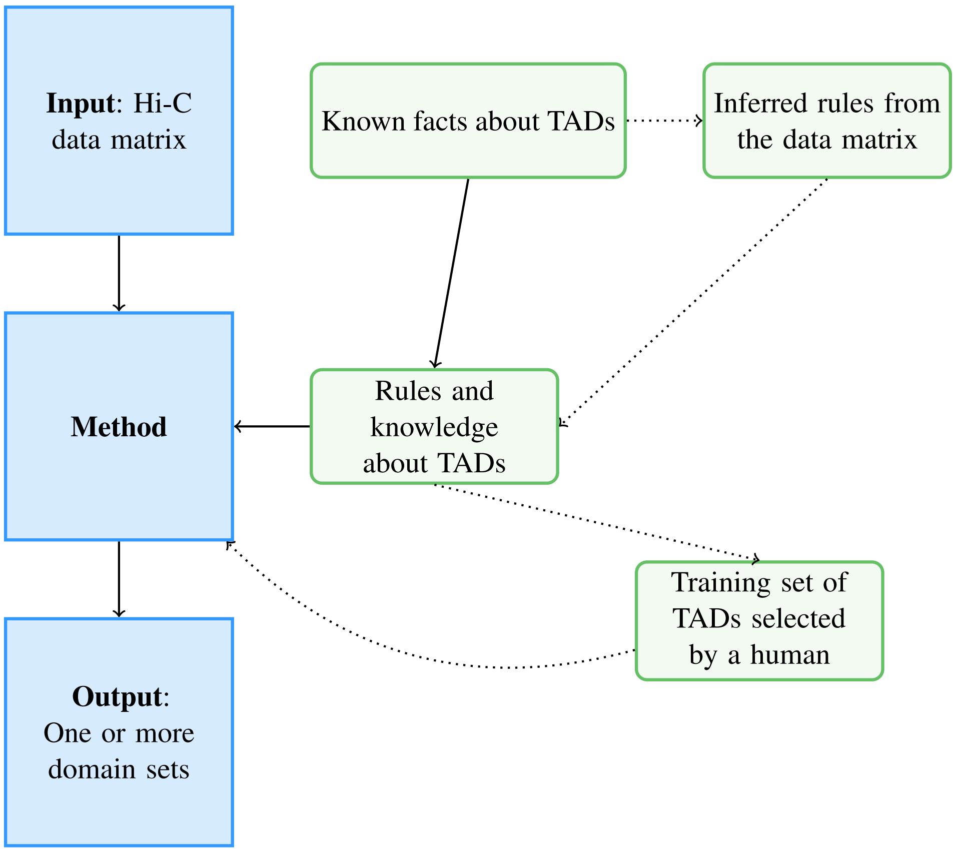

Input: Hi-C data matrix
Known facts about TADs
Inferred rules from the data matrix
Method
Rules and knowledge about TADs
Training set of TADs selected by a human
Output: One or more domain sets

Figure 3.4: Diagram illustrating the problem of finding TADs. Dotted lines represent paths *we decided not to follow.*

1. Use the knowledge directly to create a rule-based method. This is what previous methods have done (see Section 2.3).
2. Use the knowledge to manually pick TADs in order to create a *training set*. Use supervised machine learning to let the machine infer rules about what TADs are, and pick TADs from these rules.

Supervised machine learning is a common way of solving data science problems, more specifically problems where one wishes to classify instances, and the true class labels of a set of instances (a *training set*) are known. For instance, assume we are interested in solving a simplified version of the problem: for every bin in a chromosome, decide whether it is a domain edge or not. Based on knowledge about domain edges, we could select a representative set of bins that are believed to be domain edges and a set of bins that are believed not to be domain edges. These two sets together would constitute a training set. Based on some predefined features for each bin and a training set, a machine learning algorithm could classify bins with unknown class labels to one of the two classes: edge or non-edge.

The strength of this approach is that we can let the machine choose the optimal set of rules that fit with the training set, instead of trying to decide these rules ourselves. For this to work in practice, the training set needs to be reasonably large and sufficient features need to be picked.

However, there are a couple of reasons why supervised machine learning may not be appropriate in this case:

- The few rules we know about TADs are very simple. Using these simple rules to manually create a training set can seem to be a detour if the machine learning algorithm ends up choosing very similar rules.
- If we already have a set of simple rules about TADs, it should be possible to create an *algorithm* that finds these TADs directly instead of choosing them manually.
- We have more knowledge about the basic rules about TADs than about what TADs really look like in the data matrix. Visually selecting TADs from the data matrix is prone to error, and there might be interesting patterns and partly hidden TADs in the data matrix that follow the basic rules, and that at the same time are not easily visible. How TADs appear in a visualized data matrix is dependent on the visualization, e.g. the choice of colours and the visualization technique.

These arguments are related to what we discussed at the beginning of this section. How TADs visually appear in the data matrix does not explicitly define them, and should not be used as a basis for finding new domains. Instead, the problem is about using the basic principles of TADs in a direct and correct way. Thus, we choose (as others before us) not to use machine learning, and instead to develop a rule-based method.

3.3.1 Model

Before discussing new methods for finding TADs, a model and a better understanding of TADs is required. Section 2.3 briefly presents what already is known about TADs, which mainly is that they are areas in the genome where bins:

1. interact frequently with other bins in the same area (many intra-domain interactions)
2. interact less frequently with bins outside the area (few inter-domain interactions)

These two criteria are easily justified from a biological point of view. We know that TADs are spatially compact clusters of chromatin. However, this is not a very precise definition of what TADs are. In order to develop methods and algorithms to find TADs, one would like a definition that is as precise as possible. Note that the terms *interact frequently* and *interact less frequently* are relative terms. They cannot directly be used to assess whether an area on the genome is a TAD or not. However, these measures can be used to rank TADs, i.e. determine to what degree an area is a TAD. Also, we are not specifically interested in evaluating whether a certain domain is a TAD or not. Instead, given an area, we would like to find the optimal set of TADs within it.

Seen from this perspective, it seems that a TAD is defined by being in the optimal set of TADs. So what is the optimal set of TADs? Previous attempts to find TADs have tried to answer this question. Dixon et al. [5] were the first to discuss this, however they never explicitly defined the most optimal set, they only implicitly did so through the algorithm they used to find the set. A more to-the-point definition is the one proposed by Filippova et al. [13], in which they define a function for scoring a TAD, and optimize the sum of all TAD scores over all possible sets of TADs.

One aspect of these attempts, is that they implicitly define a minimum criterion for a domain to be regarded as being a TAD. Every bin in the genome is assigned either to be in a TAD or not:

- In the method of Dixon et al., a bin needs to be between a *downstream biased* and an *upstream biased* state to be regarded as a TAD.
- In the method of Filippoca et al., a bin needs to be in an area that has a higher score than the average score for all possible areas of the same size to be regarded as a TAD.

Having these constraints, huge parts of the genome are left out. In the consensus domains of Filippova et al., large self-interacting areas are classified as being non-TADs because the average interaction count within the area is less than the average interaction count within areas of the same size.

The idea seems to be that some areas of the genome are not of interest, because there are simply too few interactions between bins in these areas. This argument requires some kind of threshold, some minimum criterion for accepting an area as being a TAD. The threshold used by Filippova et al. is the average score for all possible domains of the same size, i.e. a TAD is required to have a higher score than the average. Does it make sense to have such a minimum criterion? If we want to find self-interacting domains, why discard the ones that have fewer interactions than the average?

We believe it is of greater interest not to limit our scope at this point, and not to discard any potentially interesting areas without having a good reason. Thus, instead of classifying bins into the classes TAD or non-TAD, we would wish to assign every bin a domain, so that the whole genome is divided into domains. Some of the domains will have many intra-domain interactions, and will be similar to what Dixon and Filippova define as TADs. Others will have fewer interactions, typically fill up the space between TADs, but may still be of interest. Note that this approach does not stop us from filtering out TADs later by using a threshold criterion in a similar way as Filippova et al.

Thus, the problem becomes: Given an area on the genome (e.g. a chromosome), assign a domain to every bin. These domains should:

1. contain bins that interact frequently with other bins in the same area (many intra-domain interactions)
2. contain bins that interact less frequently with bins outside the area (few inter-domain interactions)

This not-very-strict model gives us the opportunity to explore many different approaches, not limiting the scope of potential algorithms.

Even though these principles are simple, it is not obvious how the Hi-C data can be operationalized in order to find these domains. The following sections are about this.

3.3.2 Approach I: using a hidden Markov model

Motivation

This first method was developed before the work of Filippova et al. (2014) was published. It does not directly seek to solve the problem we have defined, but is included as a preliminary method, partly to see how a HMM could be used, and because it motivates later methods. While mainly being inspired by the work done by Dixon et al., it tries to do things slightly differently in the following ways:

- Find different types of domain. Instead of only classifying every bin into the category TAD or non-TAD, we wish to find different types of domain, e.g. domains with many or few intra-domain interactions.
- Utilize the use of a feature vector for every bin. Adjusting the feature vector makes us able to decide which interactions are most interesting, thus giving us an easy way to focus on finding small domains, big domains or domains with a certain type of interaction.
- Classify every bin into one domain, not omitting any bins. This way, non-TADs are also kept as separate domains, so that further analysis can be done on them.

Method

We begin by assuming a set of features, i.e. a feature vector, for every bin, as presented in Section 3.2. If we define the feature vectors to be *symmetrical*, i.e each feature consists of interactions in the same interval range upstream and downstream, feature vectors for bins that are inside the same domain will be similar (because they interact with the same other bins). Further, assume that the whole area we are looking at consists of a given number of different types of domain (e.g. three types: domains with many intra-domain interactions, domains with fewer intra-domain interactions and domains with few intra-domain interactions), and that we know the probability distribution type that features within each of these domain types follow. These assumptions fit with those of a hidden Markov model (see Section 2.4.1).

To test the method, a simple Python script was created that uses a HMM library from *Scikit-learn* [19], which uses the Baum-Welch method and the features defined in Section 3.2. Different choices for number of states and intervals were used. Examples of results when applied to chromosome 3 are shown in Figures 3.5 and 3.6. In Figure 3.5, two states are shown, which can be interpreted as areas with many interactions (marked with a blue line at the bottom), and areas with weaker interactions (marked with a green line at the bottom). In Figure 3.6, more states are shown.

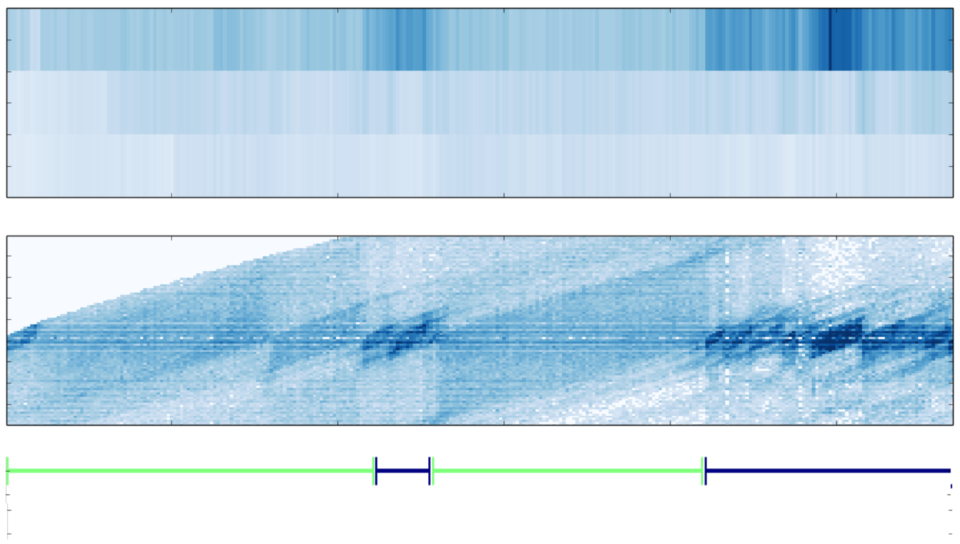


Figure 3.5: Illustration of states/domains found by using a HMM at the beginning of chromosome 3 (IMR90). Heat map of feature vectors on the top. The feature vector is as follows: The top row (first feature) represents interactions with bins less than 10 bins downstream, the middle row (second feature) interactions with bins more than 10 and less than 40 bins downstream, and the third row (third feature) interactions with bins more than 40 bins and less than 100 bins downstream. The data is visualized in the middle, and the states are shown on the bottom.

Limitations of Approach I

Using a HMM to define states along the genome is useful, because the HMM takes into account the relationship between bins next to each other. This has enabled us to cluster bins close to each other with similar features into a domain. However, this method does not take into account the important fact that the domains we seek to find are usually separated by distinct edges. For now, let us loosely define a distinct edge as a bin where the interaction vector for the bin next to it to some degree differs from itself.

Since we defined our features to be symmetrical, these edges will not be seen in these features. For instance, take two domains of the same type next to each other, with a distinct edge between them. These two bins will have feature vectors that are similar. Thus, the HMM we trained will classify each bin in both domains to the same domain, resulting in one domain that actually consists of two.

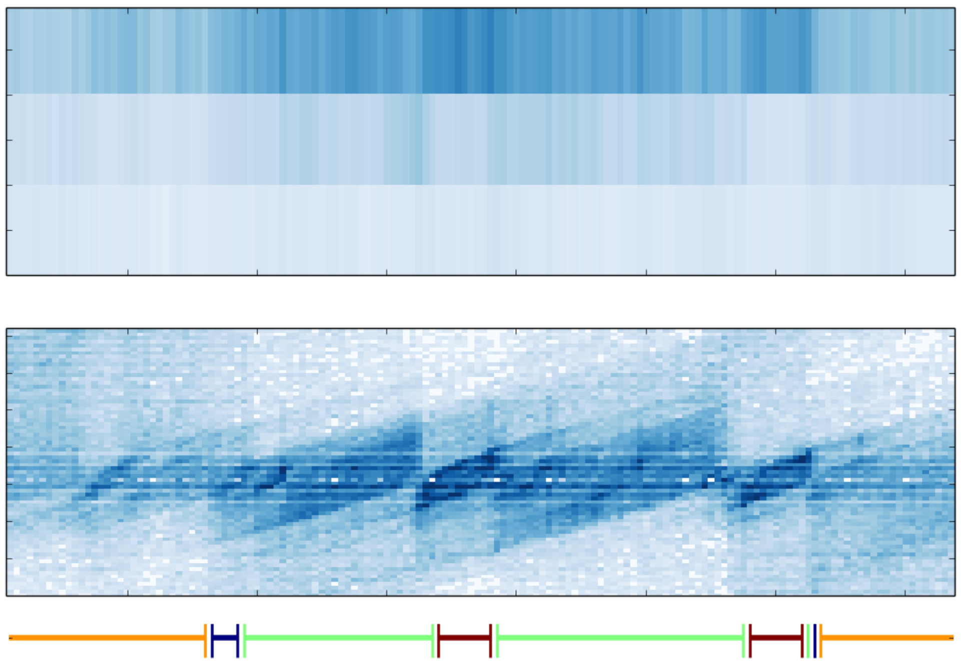


Figure 3.6: Similar illustration as in Figure 3.5, but more states have been found

3.3.3 Approach II: edge-based approach

Motivation

The problems with Method I motivate a new approach, where we first attempt to find the edges in order to make sure that separate TADs are not merged. This method is based on the following concept:

- Domains start and end with an edge.
- Finding these edges divides the area of interest into domains.
- After finding domains, they can be clustered into different types of domain, which gives us a set of domain types as in the first method.

Method

To find domain edges, we start with a very simple idea: A domain edge should have a feature vector that is significantly different from the feature vector next to it. Note that this requires non-symmetrical feature vectors, since the last symmetrical feature vector in one domain can be similar to the first symmetrical feature vector in the next domain.

The simplest rule that can be formulated from this idea is: Bin i is an edge if the sum of the squared pointwise difference between its feature vector and the next feature vector is larger than some threshold, i.e.:

$$d(i) \geq t \Leftrightarrow \text{Bin } i \text{ is an edge} \tag{3.1}$$

where t is some predefined threshold constant and

$$d(i) = \sum_j (B_{j,i} - B_{j,i+1})^2 \tag{3.2}$$

is the euclidean distance between the feature vector of bin i and the next.

Choosing a suitable t will give us a set of edges. These edges will be the "sharpest" edges, in terms of being the bins with features that differ most from their respective neighbour bins. However, we might also be interested in finding domain edges in parts of the genome where edges in general are not that sharp ($d < t$). These are typically parts of the genome with bigger and less dense domains that are not as clearly separated as in other parts of the genome. To discover these edges, we can look for local maxima of d. Let us define the threshold t as a function:

$$t(i) = \max(\tau, \max(d(i-m), \cdots, d(i), \cdots, d(i+m))) \tag{3.3}$$

Here the threshold value for a bin i, $t(i)$, is the largest number of some constant τ and all values of d in some window with size $2m + 1$ around bin i. Thus, by defining the rule

$$d(i) \geq t(i) \Leftrightarrow \text{Bin } i \text{ is an edge} \tag{3.4}$$

bin i is an edge if and only if:

- $d(i)$ is greater than some threshold τ
- $d(i)$ is a local maximum, i.e. the greatest value of d in some window with size $m + 1$ around bin i

Having these two constraints, we make sure that an edge will never be next to another edge. Also, in areas with lots of potential edges, only the most distinct ones are chosen. In areas with few and weak edges, some edges are still chosen, as long as they satisfy the first criterion.

There are several variations of the method that might give better results. Some of them are:

- The distance between two vectors can be defined in different ways (e.g. cosine distance, correlation distance etc.). We tested cosine distance without noticing any huge difference.
- We also made a small adjustment to the threshold function $t(i)$. Normal distribution of edge scores was assumed, and the threshold was set so that an edge needed to be significant in its local surroundings in order to be selected, i.e. have a small enough p-value. We did not observe any big differences to the results with this change.

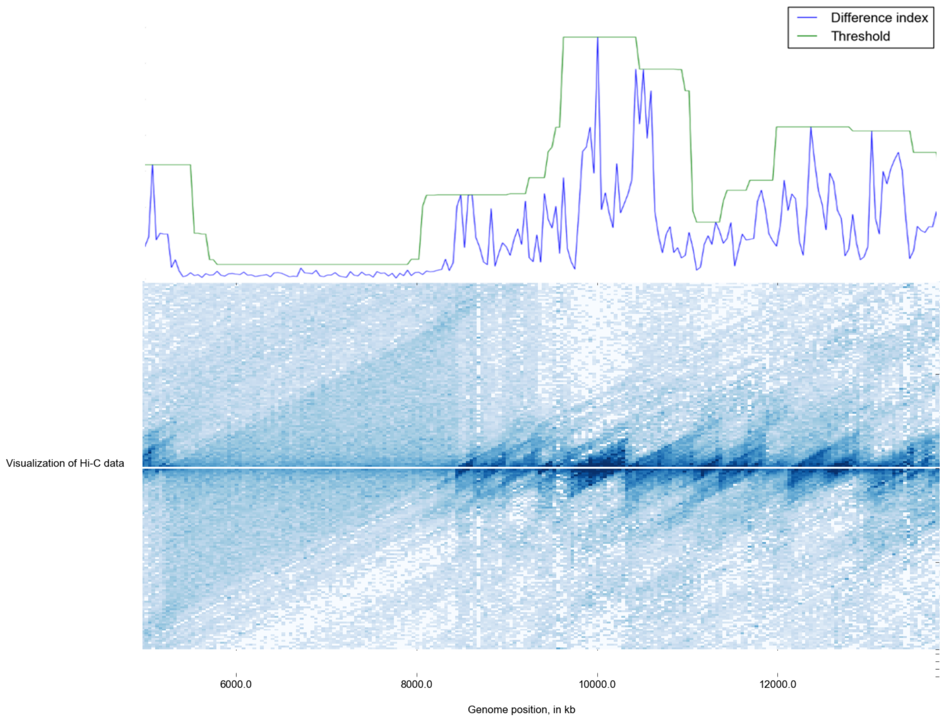


Figure 3.7: Plot of the difference index $d(i)$ and threshold function $t(i)$ at the beginning of chromosome 3 (IMR90). Where the threshold function meets the difference index, there is an edge of a domain.

Clustering domains

Now that edges have been defined, we implicitly have TADs between the edges. Some have many intra-domain interactions, others few. It is now possible to label each TAD with a class. For instance, we could choose two classes: one representing dense domains (like the set of Filippova et al.) and one representing less dense domains (similar to the boundary areas of Dixon et al.). Other classes could also be of interest, e.g. bigger domains with weak interactions. Grouping into such classes can be done by calculating *one* feature vector for every domain, and clustering these feature vectors. The feature vector for each domain should represent the domain by information we are interested in. One choice is to let it be the average of all the feature vectors inside the domain.

Limitations of the method

There are a few obvious shortcomings with this method:

- The threshold function t tries to adjust itself according to the surroundings of the bin it evaluates. However, this does not always work very well. Areas consisting of bins with low edge scores next to areas with high edge scores will be ignored.

- The method is dependent on some crucial parameters — window size and a minimum threshold value. There is no general rule about what these parameters should be set to. TADs smaller than the window size will not be detected.
- The edge score is measured between neighbouring bins. However, the edge of a TAD might not appear as one single bin, e.g. the amount of intra-domain interactions increases over multiple bins. Edges that are not very "sharp" are sometimes not discovered by the method, and a very dense domain does not necessarily have sharp edges. A domain should not alone be judged by how sharp its edges are.

Finding domains by first finding edges is an intuitive and simple approach. However, it is clear that domains are more than edges. What lies between the edges is just as important. This motivates the next approach.

3.3.4 Domain size and hierarchy of domains

Method II improved on Method I by looking for edges and avoiding that domains that should be separated were merged. But Method II also had its problems. Instead of trying to solve these problems by adjusting the method, let us continue by discussing an important feature of domains, which should be considered when developing an improved method:

Domains seem to be nested hierarchically. It is obvious that domains come in different sizes, typically ranging from a few to several hundred bins, but visual inspection of the data matrix A also shows that smaller domains are nested inside bigger domains (Figure 3.8).

This fact raises the fundamental question: In which size range are we interested in finding domains? We have previously been searching for "the optimal" set of domains, but it is hard to define optimality when not considering a certain size range.

Filippova et al. were the first to propose a TAD-finding algorithm that considered this question. The algorithm takes a parameter that will affect the size of the returned domains.

Let us return to the somewhat loose description of an optimal set of domains. An optimal set of domains should consist of domains where bins in each domain frequently interact with other bins inside the same domain, and less frequently interact with bins outside the domain. Does this definition limit the size of optimal domains in any way? Certainly not. A whole chromosome as one single domain satisfies the definition, as does having a single bin alone in its own domain.

From this it follows that every domain found can possibly consist of smaller domains, or can be a part of a bigger domain. This points towards the fact that what the optimal domain set is, depends on the domain size that is preferred. It leaves us with a few possibilities when searching for an optimal set of domains:

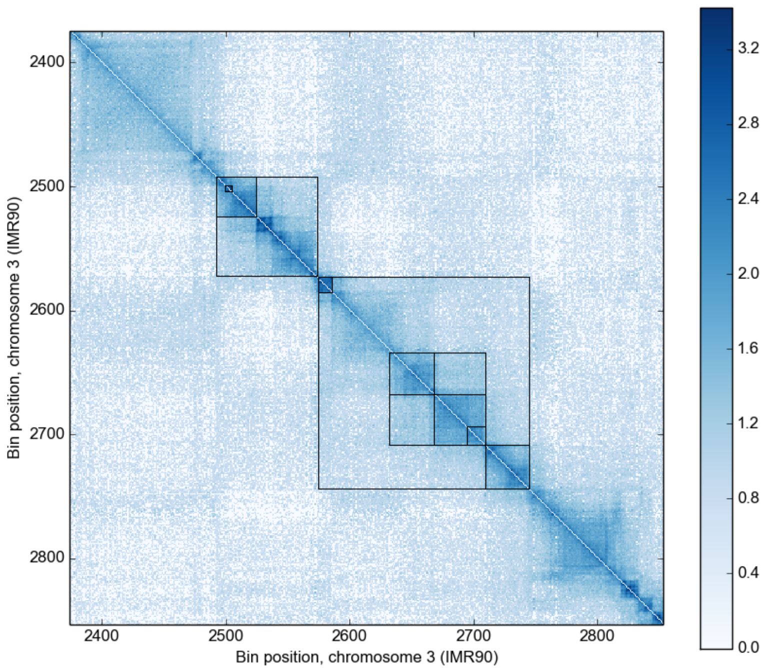


Figure 3.8: Heat map of a part of the data matrix with some visually spotted TADs marked, showing that smaller TADs are positioned inside bigger TADs.

1. Implement a size parameter into the method, like Filippova et al. did. Let the user decide which size is preferred, and return domains close to that size.

2. Try to choose an optimal set of domains across sizes. This is what Filippova et al. did with their consensus domains.

3. Return a hierarchy of domains, where smaller domains can be nested inside bigger domains, i.e. return a set of domains where domains may overlap. Filippova et al. also made a hierarchy of domains from the first solution. The goal would be to assign every bin to multiple domains that are nested in a hierarchical way.

Note that the third solution does not exclude the first and second solutions, i.e. one can first find a hierarchy of domains of different sizes and later filter out domains in a certain size range, or choose a consensus set of domains. We still do not want to exclude any possibilities or limit the method. Thus, we aim for solution 3, developing a method that returns domains nested hierarchically.

3.3.5 Approach III: hierarchical domains

Motivation

We now have a slightly different view of domains: Because domains may be nested in other domains, there is no way to define an optimal set of domains within an area when optimal size is not defined. We do not (yet) want to define optimal size, instead we will find domains of different sizes, nested in a hierarchy.

In order to describe a hierarchy of domains, let us first list some terms and definitions that will be useful:

- *Subdomain*: A domain that is a part of (nested inside) a bigger domain. Domain $A \in [a,b]$ is a subdomain of domain $B \in [c,d]$ if $a \geq c$ and $b \leq d$ and either $a > c$ or $b < d$.
- *Mother domain*: If domain A is a subdomain of domain B, then domain B is a mother domain of domain A.
- *Direct subdomain of a mother domain*: Let A be a subdomain of B. Then A is a direct subdomain of B if and only if there is no mother domain of A that is also a subdomain of B.
- *Brother domain*: Domain A is a brother domain of domain B if A and B share the same direct mother domain.

We assume that every subdomain has only one direct mother domain. This can be argued for through the following contradiction: Assume a domain is in two direct mother domains. Let us denote the two parts placed in each direct mother domain as a and b. Part a will be in one mother domain with some other domain (or a part of it), as will part b. Since a is with other bins in a different domain than b, it means that a interacts more with these bins than with other parts of the genome, which contradicts the fact that a originally was in a domain with b.

For simplicity, for now we will assume that domains consist of contiguous bins, i.e. bins that are connected on the genomic line. See Section 3.3.8 for a more detailed discussion of this.

The goal is to find a hierarchy of domains within an area, i.e. for every bin inside the area, assign it to a set of domains so that all the domains are direct subdomains of each other, except the biggest domain, which is the whole area we are considering (e.g. a chromosome).

Method

The above assumptions form a model. To fit a genomic area into this model, we propose that we first can find the direct subdomains of the area, since all other subdomains will be subdomains of these direct subdomains. There are two problems to solve:

1. How can we find all the direct subdomains inside a mother domain?

2. How do we know whether there are any subdomains at all in the mother domain? Always assuming there is implies that every bin is its own domain.

We will begin by proposing a solution to the first problem. Assuming that there are one or more direct subdomains of a mother domain, how do we find them? We will need to divide the mother domain into subdomains. If we know how many direct subdomains there are in the domain, we can find the optimal set of that number of domains inside the mother domain[8]. Not knowing the exact number of subdomains, we could assume an amount, but assuming more than there is will result in direct subdomains not being discovered. The simple solution is to divide the mother domain into no more than two new domains. If we divide it into three domains, two of the domains together may be a direct mother domain to themselves and are thus not direct subdomains of the one we are dividing. Thus, for every bin in the mother domain, assign it to one of two domains, such that each of the domains consists of that interact frequently inside the domain and less frequently outside the domain.

There is a consequence of dividing in two. The argument was that by dividing in two at every level, we make sure that direct subdomains will be found (at some point) and not omitted. However, by doing this, we might find domains that by previous definitions we would not actually call domains. Imagine a mother domain consisting of three direct subdomains. By dividing in two, we will first find one of them as a separate domain, and two of them grouped together in a separate domain. Dividing this last domain in two again gives us the two remaining direct subdomains. However, we have now included a domain in our results, which is the grouping of two of the direct subdomains. In the origin of the argument, we did not think of this as a domain. Since our aim at this stage is to process these domains further, e.g. to filter out some of them or group them into different domain types, we do not need to worry about these extra domains now.

Thus, the solution to the first problem is: Start with the whole area of interest as one domain. Divide recursively in two subdomains, following some rule about where the optimal place to divide is. The question then obviously becomes, when should we stop dividing? This question is related to the second question stated above: How do we know whether there are any subdomains at all in the mother domain? There is obviously no point in continuing to divide if there are no more subdomains to find.

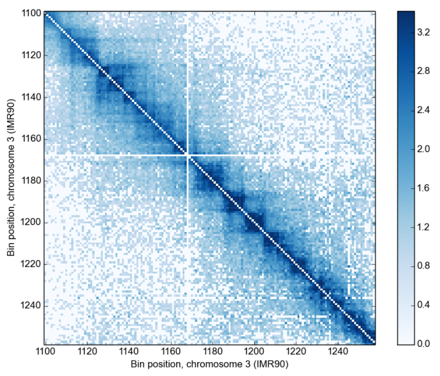


Figure 3.10: By only looking at the data matrix, it is not always obvious how far one should divide.

[8]There are several ways this can be done. A general approach is to define a score function for a given domain, and the set of domains within the area that maximizes the sum of scores.

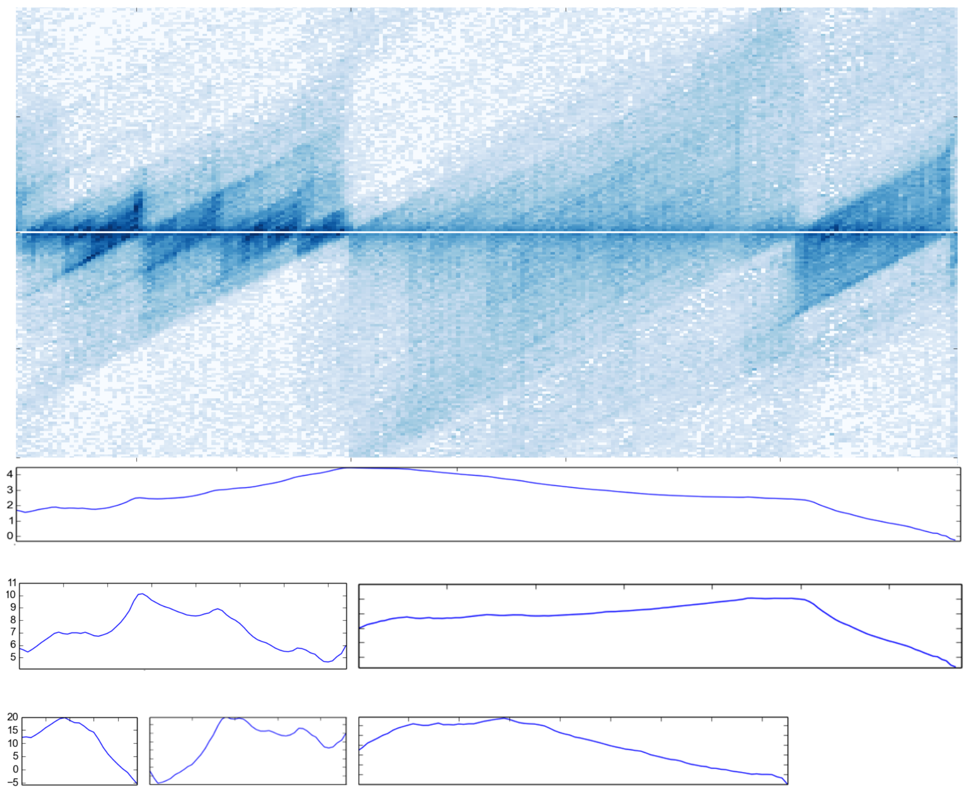


Figure 3.9: Illustration of Approach III. Visualization of Hi-C data at the top, and score functions below (Equation 3.5). Each line of figures represents a level in the recursion, e.g. the first division is done at the maximum of the score function plotted on the top. A given area gets divided where the score function has its maximum. The two new areas are further divided in the same way. The area visualized is bin 4273 to bin 4490 on chromosome 3 (IMR90).

Assume that we value a domain by a score function, e.g. the average intra-domain interaction frequency minus the average intra-domain interaction frequency in the domain's direct mother domain. Other score functions are also possible, as will be discussed in the next section. Areas that are actual TADs should have a significantly high score compared to a random area of the same size within a similarly sized mother domain. Thus, we reckon an area to be a TAD if its score is significantly high compared to all possible scores of pairs of subdomains and direct mother domains of the same sizes across the genome.

We assume that the scores are normally distributed, thus we can perform a simple t-test to evaluate whether a domain is significant. We decide to stop dividing when there is no significant subdomain anywhere in the domain we are evaluating, using the definition of significance just stated.

This method produces a hierarchy of domains, where each domain is one of the following:

- a mother domain that is a grouping of two direct subdomains

- a subdomain with no more subdomains, but which is significant by definition
- a subdomain that is not significant by definition, but that shares a direct mother domain with a significant brother domain

There is one more issue that should be briefly discussed. Originally, before introducing the idea of dividing in two, we assume an unknown number of direct subdomains of a domain. Are we guaranteed to find all these domains at some point by recursively dividing into two domains? Imagine again a domain consisting of three subdomains. We find them by first dividing in two, then dividing one of the two new domains again. For the three domains not to be found, the first division has to divide one of the three domains into two parts, which means that we have a domain with two direct mother domains. This contradicts what we have shown earlier, that every domain is only part of one direct mother domain. Thus, we conclude that we will find all domains by recursively dividing in two.

An important part of the algorithm is of course what the rule for dividing should be, i.e. given a domain, at what bin inside the domain should one divide. The rule has to follow the basic principle: The two new domains should consist of bins that interact frequently with other bins inside their own domain and less frequently with the other domain. We suggest the following rule that follows this principle: Divide where the average interaction frequency within the two domains minus the average interaction frequency between the two domains is maximized, i.e. divide at the bin $\hat{\imath}$ inside the domain $[a,b]$ that maximizes the following score function:

$$S(\hat{\imath}) = \frac{\sum_{i=a}^{\hat{\imath}-1}\sum_{j=a}^{\hat{\imath}-1} A_{i,j} + \sum_{i=\hat{\imath}}^{b}\sum_{j=\hat{\imath}}^{b} A_{i,j}}{e(a,\hat{\imath}-1)+e(\hat{\imath},b)} - \frac{\sum_{i=a}^{b}\sum_{j=a}^{b} A_{i,j}}{(\hat{\imath}-a)(b-\hat{\imath}+1)} \tag{3.5}$$

where $e(a,b)$ is the number of bins that we count interactions for inside domain $[a,b]$. Since we set the diagonal to be zero, we do not count the elements on the diagonal (see Section B.4):

$$e(a,b) = (a-b+1)^2 - (a-b+1) \tag{3.6}$$

A visualization of how this score function behaves with the data is shown in Figure 3.9. A further discussion of possible score functions is given in the next section.

Algorithm

The algorithm was implemented using Python. We will briefly present the main part of the algorithm using Python-like pseudocode. Code for the score function from Equation 3.5 is shown in Listing 3.1 together with code for two simple helper functions that compute the sum of intra-domain and inter-domain interactions.

Using these functions, a simple recursive function is used to divide the genome, shown in Figure 3.2. The implementation of the stopping criterion is not shown.

Listing 3.1: Score function

```
import numpy as np

# Returns the sum of intra-domain interactions
def intra(start, end):
  return np.sum(A[start:end, start:end])

# Returns the sum of inter-domain interactions
# between two consecutive domains
def inter(domain1_start, domain2_start, domain2_end):
  return np.sum(A[domain1_start:domain2_start, \
                  domain2_start, domain2_end])

def score(start, divide, end):
  size_a = divide - start
  size_b = end - divide

  # Compute number of interactions
  # (Subtract number of bins on diagonal in intra,
  # since these are always zero)
  int_a = size_a**2 - size_a
  int_b = size_b**2 - size_b
  int_inter = size_a * size_b

  # Compute the score
  avg_intra = (intra(start, divide) \
             + intra(divide, end)) / (int_a + int_b)
  avg_inter = inter(start, divide, end) / int_inter
  score = avg_intra - avg_inter

  return score
```

Reducing the complexity by using dynamic programming

The algorithm as presented in Listing 3.2 is complex. For every domain, the score is calculated for every bin inside that domain, which means that every interaction frequency is iterated over several times.

Since the score function is based on sums of rectangular submatrices of the interaction matrix, we can reuse these sums to reduce computation time. The most obvious way of reusing these sums is to reuse them when the score is calculated for every bin inside a given domain. For instance, when the score for a bin i is calculated inside some domain $[a,b]$, the sum of the inter-domain interactions is the sum of the submatrix $A_{a:i,i:b}$. When the next score is calculated, for $i+1$, the sum of the submatrix $A_{a:i+1,i+1:b}$ is calculated. These submatrices overlap, so when the score for bin $i+1$ is calculated, we only need to compute the parts not already contained in the previous submatrix. See Figure 3.11 for a more detailed explanation.

Listing 3.2: Code that divides the genome

```
domains = []

# Return the optimal bin in some area
def optimal_bin(start, end):
  optimal_bin = -1
  optimal_score = -1000

  for i in range(start, end):
    s = score(start, i, end)

    if s > optimal_score:
      optimal_bin = i
      optimal_score = s

  return optimal_bin


def divide_genome(start, end, level = 0):
  if level == max_level:
    return

  divide = optimal_bin(start, end)

  # Store two new domains, and continue recursion
  domains.append([start, divide])
  divide_genome(start, divide, level + 1)
  domains.append([divide, end])
  divide_genome(divide, end, level + 1)
```

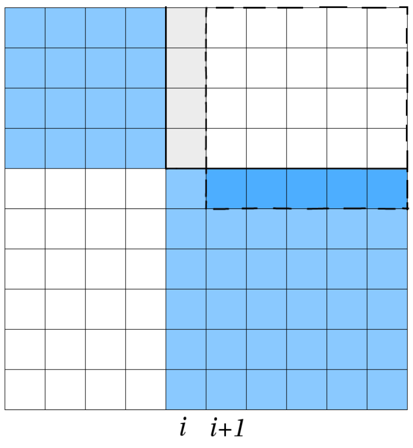


Figure 3.11: Illustration of the score calculation for bin i inside some domain. The two parts on each side of bin i are coloured blue. The sum of the inter-domain interaction frequencies is the sum of all the frequencies inside the rectangle marked by a solid black line. When the equivalent sum is calculated for the next bin $i + 1$, we can reuse the last sum. All we need to do in order to get the new sum, is to subtract the interactions in the light grey area and add the interactions in the dark blue area.

3.3.6 Improvements to Approach III

We believe that Approach III solves the original problem we formulated at the beginning of this chapter. It uses the basic known principles of TADs and deals with the problem of hierarchically nested domains.

So far, the method has been presented without discussing important details further, such as the score function and stopping criteria. In this section, we give a more thorough discussion of these aspects.

Score function

The score function (Equation 3.5) does a trade-off between high intra-domain interactions within and weak inter-domain interactions between the two new parts it aims to divide between, tending to divide at a bin where the two new parts have many intra-domain interactions and few interactions between each other. This approach assumes that the two new parts on each side behave like individual TADs. However, as previously stated, these areas may not be TADs in the sense that instead they can be areas containing many smaller domains, having few intra-domain interactions on average as individual domains. Also, as previously stated, the approach aimed first to divide the genome into “interesting” areas, then to choose TADs from this set by favouring the most dense areas. Thus, it might be incorrect to give weight to intra-domain interactions, i.e. density, when dividing the genome, since a lot of interesting areas may have weak intra-domain interactions.

We propose an alternative score function. Since the intra-domain interaction frequency within interesting parts may vary, and is not a key characteristic of an interesting part, the only characteristic we have left is that the parts of interest have few interactions with each other. Thus, a score function only measuring inter-domain interactions seems reasonable. The simplest one returns the average inter-domain interaction frequency between two parts:

$$S_{i,j}(k) = -\frac{\sum_{m=i}^{k-1} \sum_{n=k}^{j} A_{m,n}}{(k-i) \cdot (j-k)} \tag{3.7}$$

This is the same as the negative of the mean of the interaction frequencies between the two domains $[i, k-1]$ and $[k, j]$. A division will be done at bin k that maximizes this function, i.e. where the average interaction frequency between the two new domains is minimized.

This score function makes sense, because we actually want to divide the genome into parts that interact as little as possible with each other, and avoid dividing domains unless they contain smaller domains inside. By dividing where the average inter-domain interaction frequency is at its lowest, we lower the risk of dividing a valid domain. Note that the intra-domain interactions within the two areas implicitly affect which bin to divide at, since minimizing the interactions between the two areas is connected to maximizing the interactions within the areas.

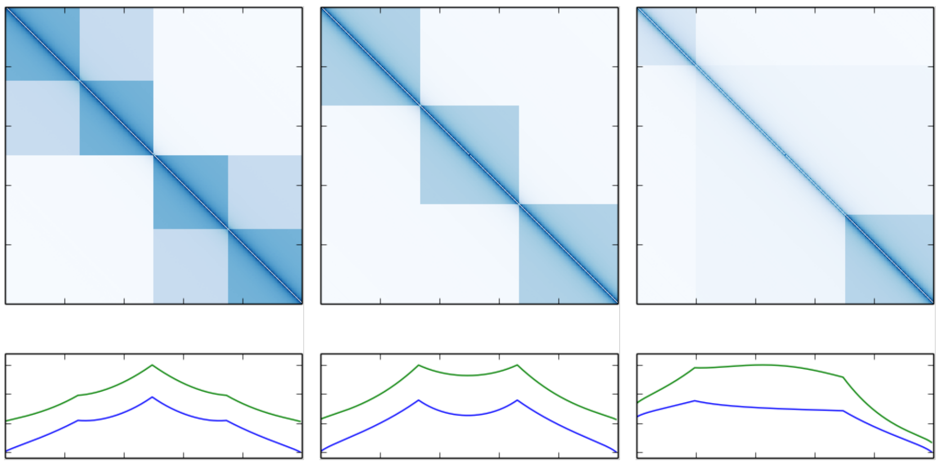


Figure 3.12: Simulated data matrices (on top) and line plots of the two score functions (bottom): The original score function (green) (Equation 3.5), taking intra-domain interactions into account, and the new alternative (blue) (Equation 3.7), only considering inter-domain interactions.

Figure 3.12 shows the two score functions over three simulated data matrices. Both score functions behave as wanted over the two simple cases, shown in Figure 3.12 *a* and *b*. A more complicated example is shown in subfigure *c*. The data matrix contains three domains, one small domain to the left and one big domain that contains a smaller dense domain. We would expect any valid method to first divide at the bin between the first small domain and the big domain, since the big domain contains the last small domain. The original score function (green line plot) fails to do this. It has a local maximum between the small domain and big domain, where we would expect it to have a global maximum, and increases to a global maximum somewhere inside the big domain. At the point where it reaches its maximum, the average intra-domain interaction frequency is bigger than between the small domain and the big domain, because a greater portion of the right part is filled by the dense subdomain inside the big domain. However, when getting closer to this subdomain, the score function decreases because the inter-domain interaction frequency increases.

This example illustrates the problem of making a trade-off between intra-domain and inter-domain interaction frequencies, which here leads to the selected bin being a “trade-off” between two candidates:

- The *correct* bin, between the left small domain and the big domain, which separates two parts having on average medium intra-domain frequency and low inter-domain frequency
- The other potential bin, inside the big domain, at the beginning of the smaller dense domain, which separates two parts having on average high intra-domain frequency and medium inter-domain frequency.

The result is that the optimal score occurs somewhere between these two candidates. The new score function (blue line plot) correctly reaches its maximum between the small domain and the big domain, separating the two parts that interact the least with each other.

We conclude that it is better to use the new score function, only giving weight to inter-domain interactions frequencies.

Selection criterion

Using this new score function, we aim to uncover all possible domains by recursive division. Some of the areas after this process will be groups of domains, and we will also find areas that are the “leftover” parts of small domains inside bigger domains.

The plan is to pick dense domains from this set to create a consensus set. This consensus set is supposed to contain domains with many intra-domain interactions and few inter-domain interactions. However, this set will also contain non-domains in that sense. The reason is that the hierarchical set will contain “leftover” parts, as illustrated in Figure 3.13. If a “leftover” part is in a dense area, it will be included in the consensus set, even though it is only a part of a bigger domain and has many interactions with the rest of the domain it is a part of.

We also used a stopping criterion, to avoid dividing too far and introducing non-domains. Division ended when the score at the optimal bin was not significantly high compared to a random score. It came to our attention that an even simpler stopping criterion could be used, not relying on doing a statistical hypothesis test. This new stopping criterion was derived from our original definition of domains. If a domain interacts more with some other neighbour domain than it does with itself,

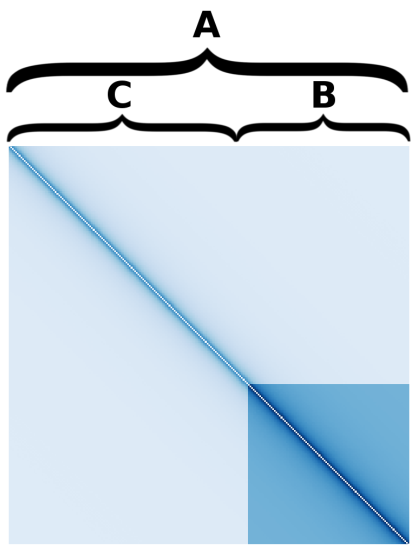


Figure 3.13: Heat map of simulated data matrix. One big domain (A) having one subdomain inside itself (B). The first version of Approach III will find the area (C) as a separate domain in the hierarchical set, and it may be chosen to be in the consensus set.

it is probably a part of that domain, and is not itself a domain. Thus, we changed the criterion to be: Do not include a domain if it has higher average interactions with the other domain in the same mother domain than it has to itself. This criterion could not be used as a stopping criterion, since domains that do not fit with the rule might have subdomains that fit with the rule. Thus, instead we used it as *selection criterion*: After domain candidates were found, each domain candidate was either removed or kept depending on whether it interacted more with itself than with its brother domain.

To measure whether a domain interacts more with itself than with its brother domain, we normalized the interaction matrix so that interactions between bins far away were given more weight. See Section B.5 for details about how this normalization was performed and a discussion about global vs. local normalization.

Using this selection criterion, we ensure that all selected domains have higher average intra-domain interactions than average inter-domain interactions. Thus, also “leftover” parts that interact more with their brother domains than with themselves will be filtered out.

Bias in the score function

While developing the second score function, we were aware that the first score function had a bias towards giving higher scores in the middle of a domain. When dividing in the middle of a domain, the bins inside the two new domains (upstream and downstream of the bin we are dividing at) are on average closer to the diagonal than they would be when dividing further away from the middle. The expected value is higher closer to the diagonal, leading to higher expected average intra-domain interaction frequency when dividing close to the centre of a domain. For the same reason, the average inter-domain interaction frequency is biased towards a lower value when dividing close to the centre of a domain.

Even though this may seem to be a serious problem, it is debatable whether it is more a natural consequence of the score function. Let us first go a bit further into understanding this bias, using the second score function as an example.

Imagine the average inter-domain interaction frequency between two consecutive domains (a and b) of the same size. We now move the bin inside b that lies closest to a over to a, increasing the size of a and decreasing the size of b. On average, this bin will interact more with the domain it was in, b, than with domain a, since on average it is closer to bins in b than it is to bins in a. Thus, the inter-domain interaction between the two domains is expected to increase when we move the bin.

The question becomes whether this is a weakness of the score function or a feature of the data. Recall that on every level in the recursion, we wish to find the two parts where the average inter-domain interaction is as small as possible. It turns out that these two parts naturally tend to be of equal size, because bins inside equally sized domains are on average closer to each other. Since we also do not observe any problems with the score functions in practice, we do not consider this to be a problem.

3.3.7 Missing data

We will briefly discuss how we handle missing data in the Hi-C data we use.

We have identified three types of missing data:

- A contiguous set of bins that have no interactions with any other bins, appearing as columns and rows with only values of zero in the data matrix (Figure 3.14)
- Bins that have no interactions with many, but not all, other bins (Figure 3.15)
- Pairs of bins that have no interactions (zeroes in the data matrix)

The first type includes the centromere in every chromosome, a long series of bins (typically 100 to 200 bins in the human genome) that have no interactions with any other bins. To avoid these areas appearing as domains, we exclude any domains found inside these areas. However, we do not prohibit such bins from being in domains, since that would lead to domains being split whenever a single bin with no interactions appears.

Both the first and second type of missing data will affect statistics for the domains they appear in. Since we assume these are missing data, domains covering such bins will have a lower measured intra-domain interaction frequency than they actually have. This could be adjusted for by replacing the columns and rows that sum to zero in the data matrix with numbers that are more likely to have been there, for instance by interpolating the interaction numbers from bins nearby. Another alternative would be to ignore these columns and rows when statistics are calculated, e.g. when calculating the mean intra-domain interaction frequency, exclude bins that have no interactions with any other bins. This correction would be most important in cases where several contiguous bins have zero interactions, since a single bin does not affect the score a lot. However, several contiguous bins of that kind are not very common in the data, except for in the centromere. Thus, we choose to not use any of the proposed solutions and to avoid changing the data.

However, we see that when calculating statistics for a chromosome, it is important to not include big parts of the data matrix that represent missing data, e.g. the centromere. In our method, we normalize every interaction in the data matrix based on the average interaction with the same distance from the diagonal. If we include huge parts of the matrix containing only zeroes when calculating these averages, it will lead to a bias in the normalized version of the data. Thus, we make sure that columns and rows that sum to zero are not included when calculating average interaction frequencies used for normalizing the data.

The third type of missing data is harder to deal with. First, it is difficult to know whether single zeroes in the data matrix really represent bins that have no interactions, or whether they are missing data. Thus, we do not address this problem.

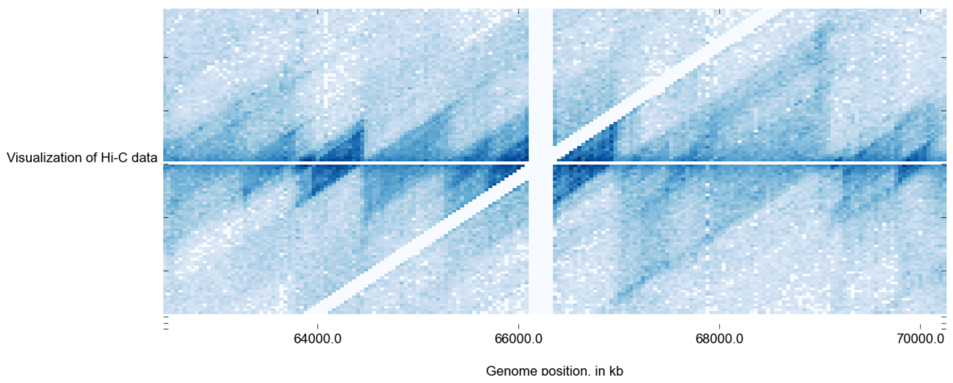


Figure 3.14: An example of missing data on chromosome 3. A set of contiguous bins have zero interactions with all other bins.

3.3.8 Contiguous or non-contiguous domains?

Until now we have treated domains as sets of contiguous bins in the genome, i.e. bins that are connected in the genomic sequence. This assumption has made it easier to develop suitable algorithms for discovering these domains. This is also how previous methods have treated TADs.

However, from our basic definition of a domain, that it consists of bins that interact frequently with each other and less frequently with bins outside the domain, a domain does not necessarily need to consist of contiguous bins on the genomic sequence. An example of two non-contiguous domains that interact more with each other than with their surroundings is shown in Figure 3.16. If we wish to separate

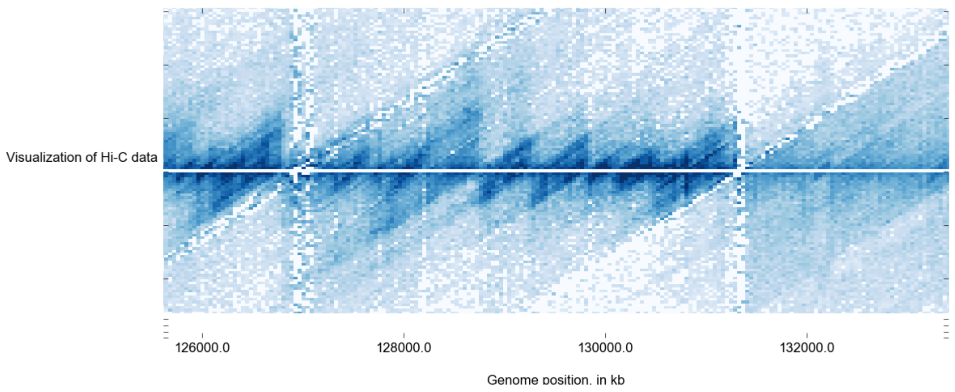


Figure 3.15: An example of possibly missing data on chromosome 3. Some bins have zero interactions with many, but not all, other bins.

the genome into self-interacting parts, not requiring that these parts have to consist of contiguous bins, the two domains shown in Figure 3.16 would preferably be together.

We will still search for contiguous domains, since that is the original problem, but we keep in mind that dense domains might be non-contiguous. For future study, it would be interesting to develop efficient algorithms for finding non-contiguous domains and investigating possible biological features related to such domains.

3.3.9 Summary of methods

Method I treats the genome as a sequence of bins belonging to different states/domains, defines a feature for every bin and feeds this into a HMM to find the states/domains. The method does not directly aim to find topological domains, rather it discovers different states along the genome.

Method II is based on the simple idea that a topological domain starts or ends where there is an abrupt change in the interaction vector for that bin to the next bin. Instead of the interaction vector, a simplified version of it is used where interactions over an interval are grouped together, into features. The difference between one bin and the next is calculated as the sum of the squared pointwise differences between two feature vectors. Edges of domains are then decided to be at bins where this function has local maxima. A minimum threshold is used to avoid too many edges in areas with small change in the feature vectors. After the edges are chosen, every sequence of bins between two edges is chosen to be a topological domain. One feature vector is chosen to represent each domain (e.g. the average or pointwise median of all feature vectors in that domain), and the domains can then be clustered into a predefined number of domain types.

Method III tries to capture the hierarchical nature of domains, by treating the whole chromosome as a potential domain and then recursively dividing it into hierarchically nested domains. A score function is used to find the optimal bin to divide at in every step in the recursion, and a selection criterion is used to select valid TADs from all domain candidates.

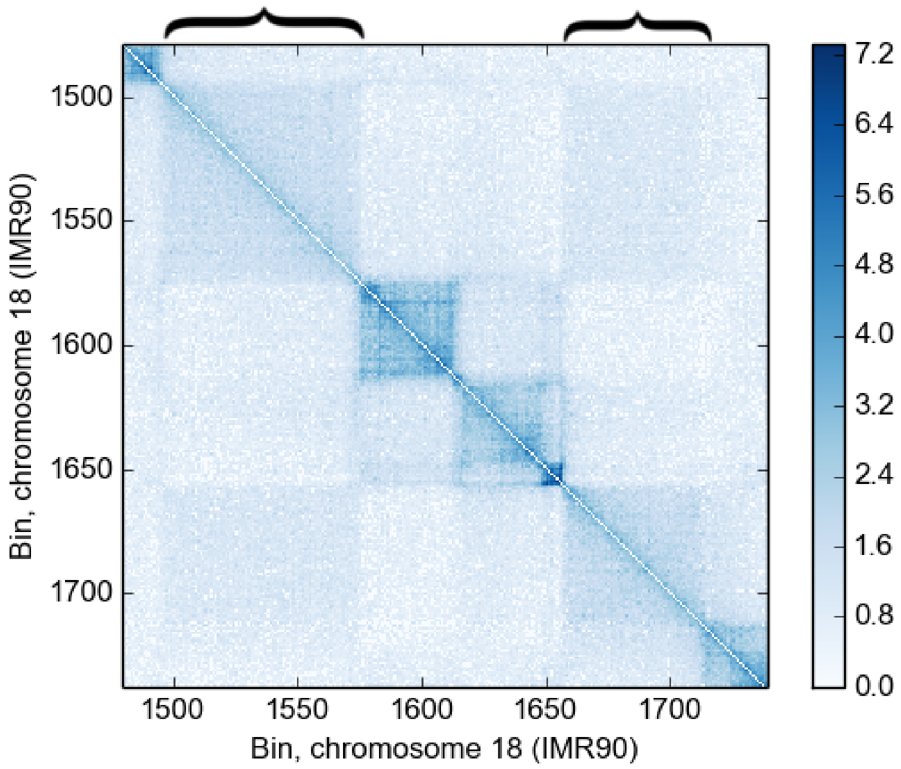


Figure 3.16: Heat map of the data matrix, showing two domains that are not connected on the genomic line, but they still interact more with each other than with their surroundings.

3.4 Deciding on a method and creating domain sets

We have presented three different approaches. As pointed out at the beginning of this chapter, we wish to decide on a method based on how it fits with the initial basic rules about TADs, not selecting a method by measuring implicit features, like how the domains relate to genomic features. As has been clear in the discussion done throughout this chapter, the third method with the improved score function and selection criteria fits with all the criteria we have discussed, whereas the first and second methods only partially solve the problem of finding TADs.

3.4.1 Consensus set of domains

The proposed method maps every bin into one or more hierarchically nested domains. However, it is useful to provide a set of non-overlapping domains, for several reasons:

- A set of non-overlapping domains can easily be compared with the domains found by Dixon et al, Filippova et al. and others.
- When analysing domain borders, the non-filtered hierarchical set is not very suitable, since there are very many domain borders representing domains on

several different scales.

- A set of non-overlapping domains is a genomic track than can easily be used in further analysis.

The consensus set should consist of domains that are the most interesting. We already have a hierarchical set, consisting of overlapping domains. For instance, a consensus set could be selected by choosing all domains found at a certain *level*[9] in the recursion, or by selecting all the smallest domains above a certain size threshold. However, as we have seen, interesting domains appear in different sizes at different levels, so only restricting according to either size or level does not make sense. Instead, we should choose domains that are interesting across scales and levels.

In order to measure how interesting a domain is, independently of its size or level, we calculate a score for every domain. We let the score be the density of the domain relative to its size, i.e. the average intra-domain interaction frequency calculated from the normalized data set[10,11].

Based on these domain scores, we want to select a set of non-overlapping domains with as high scores as possible. One *greedy* approach is as follows: Sort all domains in descending order based on score, and select domains from the top of the list iteratively. Do not select domains that overlap with an already selected domain. This approach is greedy in the sense that it includes the domain with the highest score at every step. If we define the most interesting set as the set where the average domain score is maximized, this approach will not find the most interesting set. For instance, assume a mother domain with a high score, containing only two subdomains — one of them with a higher score than the mother domain and one with a very low score. The subdomain with the high score will be selected first. The consequence is that the mother domain cannot be selected because it overlaps with its own subdomain. The other subdomain with a very low score will also be selected, since it does not overlap with any other selected domain. So even though the mother domain has a higher score than the average score of its subdomains, it will not be selected.

A less greedy approach finds the set with the highest possible average domain score among all sets of non-overlapping domains: Select domains so that no domains overlap and every selected domain has a higher score than the average for all possible sets of non-overlapping subdomains[12]. This will generate the set with the highest average domain score, since no selected domain has any alternative

[9]*Level* is used to denote the step or depth at which the domain is found in the recursion. *Deep* level means far down in the recursion, i.e. where small domains are nested inside bigger domains.

[10]See Section B.4.1 for details about the normalized data set.

[11]We choose density relative to size as a criterion, since density is one of the main features of a TAD. There are obviously other criteria that could be used, e.g. how dense the domain is compared to its surroundings. We choose density relative to size since this is a simple criterion and bears a resemblance to the way Filippova et al. do the selection. Note that even though inter-domain interactions are not directly a factor in the criterion, they are so implicitly. This is because if a domain has high inter-domain interaction frequency, a mother domain of the domain is more likely to have high intra-domain interaction frequency, making it more likely that the mother domain is chosen instead.

[12]Note that *possible sets* of non-overlapping subdomains also need to fit with the definition, and domains cannot be omitted from a set unless they do not fit with the definition.

set of subdomains with an average score higher than the domain's score. Also, no selected domain is part of a set with an average score lower than a mother domain of the set. Details about the algorithm and a proof showing why this set has the maximum average domain score is included in Section B.2.

A consensus set will always be a trade-off between different goals, since not all domains can be included. With the chosen approach, some domains with very high score might be omitted because they have a mother domain with higher score than the average of the subdomains that the domain is a part of.

3.4.2 Simplified set of hierarchically nested domains

We also provide a simplified set containing overlapping domains on different levels that are hierarchically nested. The original hierarchical set contains domains on many levels, covering the whole of the genome. We wish to make a smaller hierarchical set, covering fewer levels. To do this, we use a simple approach: All domains are sorted descending based on score, and domains are iteratively included from the beginning of the list. A domain is not included if it overlaps with more than three other domains (i.e. on three levels). The result is a set of partially overlapping domains at three levels.

3.5 Methods for analysing domains

In this section, we briefly present *how* some of the domain analyses will be performed. We use the analyses performed by others as a starting point, and propose improved ways of performing them.

We will often use the term *domain size being adjusted for*. By this, we mean that the statistic is calculated on the normalized interaction matrix[13], meaning that the statistic is somewhat invariant of domain size. We use the terms *average* and *mean* interchangeably, always referring to the *arithmetic mean*.

3.5.1 Intra- and inter-domain interactions

An important analysis to do is measuring to what degree domains are self-interacting and interact less with their surroundings, since this is a feature directly connected with the principles about TADs.

Filippova et al. measure the mean intra-domain interaction frequency within domains. We propose that mean interaction frequencies normalized for domain size should also be measured, since we are interested in how dense domains are compared to their size.

If domains are selected based on having high density, a smaller domain set with more dense[14] domains will on average have higher density than a bigger set. Thus,

[13]Normalized for distance from the diagonal, as explained in Section B.4.1.

[14]With *density of a domain*, we refer to the average interaction frequency within the domain.

we believe it is interesting to analyse domain sets that cover the same portion of the genome, to exclude the possibility that a set contains denser domains only because it covers a smaller and denser portion of the genome. Also, it is interesting to see how average domain density changes as a function of genome coverage. This can be evaluated by plotting the average domain density as the domain set is shrunk by iteratively removing the least dense domain. This should be analysed when removing the domain with the lowest average intra-domain interaction frequency, both calculated on the raw and the normalized interaction matrix.

We also propose that the inter-domain interaction frequencies should be evaluated, since a key feature of TADs is that they have few interactions with their surroundings. This can be done by calculating the average frequency that domains have to their close surroundings, e.g. to all bins that are less than 50 bins away. This calculation should also be performed on the normalized interaction matrix, so that the numbers are not affected by domain size. Also, the difference between average intra-domain and average inter-domain interaction frequencies should be computed.

3.5.2 Agreement in position of domains

When proposing an alternative set of domains, it is of interest to see how different it is from existing sets. This can be done by counting the number of *matches* between two domain sets. We defined a match between set S_1 and S_2 to be when a domain $[i, j]$ in set S_1 also exist as a domain $[k, l]$ in set S_2 where $|k - i| \leq 2$ and $|l - j| \leq 2$, i.e. when a domain exists in the other set with domain boundaries not further than two bins away. In this way, we get a number of approximately how many domains that exist in one set also exist in the other set.

We also propose another way of measuring the similarity between two methods. As described in Section 3.4.1, consensus set selection can be a trade-off between preferring different domain features, and a consensus set does not necessarily reflect the method completely. For instance, two methods can behave similarly, but choose domains on different scales to be in the consensus set. To provide a similarity measure that somewhat takes this into account, one can calculate the ratio of domains from one set that overlap a domain border in the other set. Two methods can have a similar "nature", but find domains that are on different scales, meaning that if many domains from one set overlap a domain border in another set, the sets, seen from this perspective, are not very similar.

3.5.3 CTCF analysis

Dixon et al. and Filippova et al. investigated the association between domain boundary[15] segments and CTCF and different histone modifications, concluding much the same for both. Instead of analysing both CTCF and histone modifications, we choose to do a more thorough analysis of CTCF. This requires a quick review of how it has been done previously, before discussing how we wish to do it.

[15] *Domain boundaries* are the segments between domains.

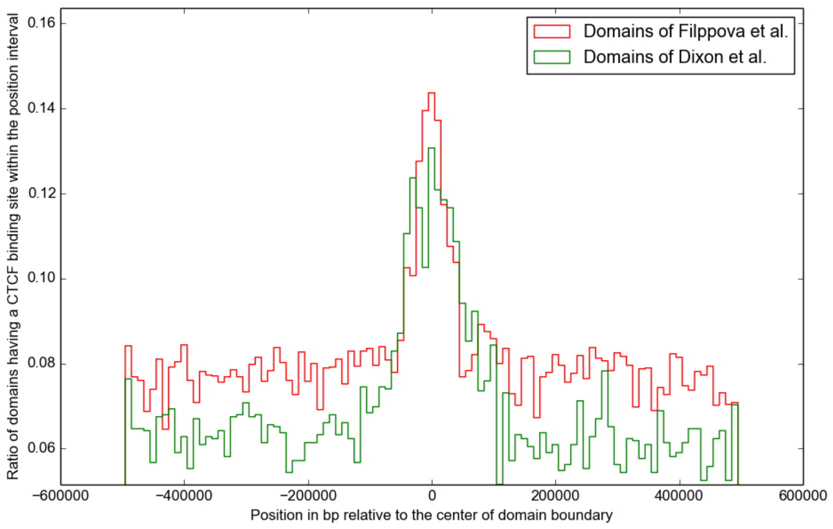


Figure 3.17: Aggregation plot of the position of CTCF binding sites relative to domain boundaries in IMR90 for CTCF data set 1 from Table 4.2. The intervals used in the histogram have length 10 kb.

Dixon et al. and Filippova et al. generate aggregation plots[16] of CTCF binding sites over domain boundaries, showing where CTCF binding sites are positioned relative to the *centre* of each domain boundary (Figure 3.17). Based on these plots, Filippova et al. point out in [13] that *the CTCF binding signals peak more sharply in the boundaries between the domains we (Filippova et al.) discover than in the boundaries between the domains of Dixon et al.* This is correct, but it is more interesting that the aggregation plot for the domain boundaries of Dixon et. al have a broader peak and seem to have two local maxima (not only one) — indicating that enrichment of CTCF binding sites is higher in the edges of the domain boundaries, or, equivalently, in the edges of the domains. Also, if the assumption is that CTCF binding sites are positioned in domain boundaries, the aggregation plots over the boundaries of Dixon et al., which on average are 50 kb, will have a broader peak than the aggregation plot over the narrower boundaries of Filippova et al. (average size 29 kb). In the supplementary material of [5], Dixon et al. show that the highest enrichment of CTCF binding sites is not in general in the domain boundaries, but more specifically in the domain edges. This has also been pointed out in recent studies [24].

The more distinct peak in Figure 3.17 for the boundaries of Filippova et al. may also be due to the fact that most of these boundaries (3766 of 4789) are actually zero bins in length, positioned between domains directly following each other. These domain boundaries are actually domain edges. We generated the same plots, omitting boundaries with zero length. These new plots, shown in Figure 3.18, show no distinct peak for the boundaries of Filippova et al. For the boundaries of Dixon

[16]Aggregation plots are histograms where the x-axis is the relative distance to one or more elements (e.g domain edges). The y-axis represents the frequency at which a second type of element (e.g. CTCF binding sites) are positioned within the intervals on the x-axis.

et al., we see two peaks at each side of zero — indicating CTCF binding site enrichment towards the edge of domain boundaries, i.e. near domain edges.

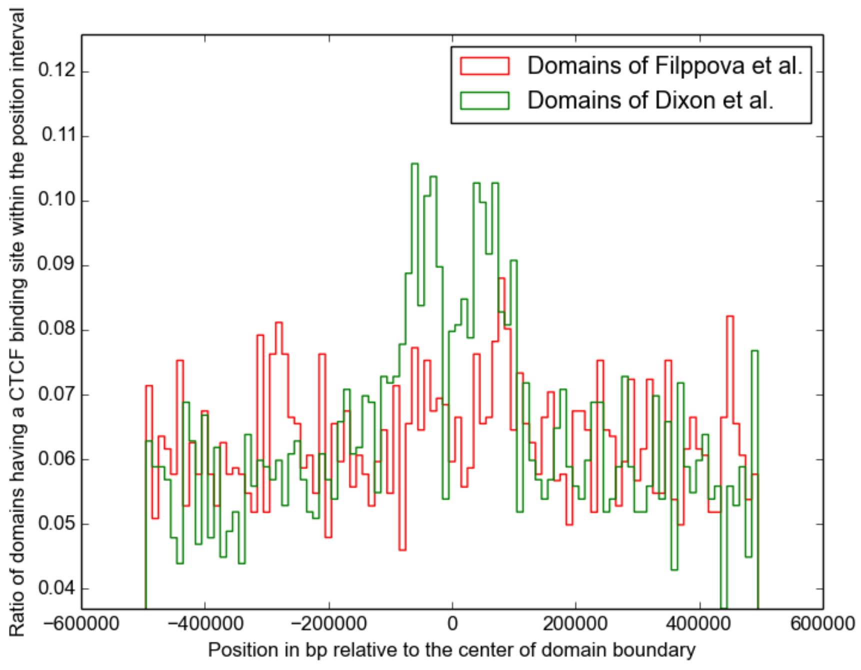


Figure 3.18: Same as Figure 3.17, but only boundaries with length $>$ 0 bins are included.

All this indicates that the interesting area for analysis is around domain edges, not around the centre of domain boundary segments. Also, when generating aggregation plots over domains or boundaries, the size of the areas should be normalized, since we are interested in spotting peaks relative to areas that vary in size.

Taking all this into account, we propose that CTCF association can be visualized best by creating aggregation plots over segments that each represent a domain and an area outside the domain. All domains should be normalized, e.g. by dividing them into 100 equally sized parts. Flanks on each side of the domains with lengths 50 % of domain sizes can be included, so that one can compare the CTCF frequency outside domains against the frequency inside domains.

3.5.4 Similarities between TADs in the mouse genome and the human genome

Dixon et al. made an interesting discovery, finding that a significant portion of domain boundaries in the human genome exist in the mouse[17]genome and vice versa, further implying that a lot of TAD boundaries have been preserved throughout evolution.

[17]When talking about human and mouse in this section, we refer to the human ES and the mouse ES cell.

This made us interested in doing the same kind of analysis. There are few details in [5] about exactly how the analysis was done. However, we contacted the first author, Jesse R. Dixon, and obtained the details. Here is a brief summary:

- Domain boundaries get defined as every area between domains. Domain boundaries less than 400 kb in size were omitted.
- All domains are lifted from genome A to genome B (either hES to mES or vice versa), using the UCSC liftover tool [25]. The parameter *Minimum ratio of bases that must remap* was set to 0.1. Usually, for different reasons, not all domain boundaries will get lifted, but in this case most of them got lifted.
- Domain boundaries originally found in genome B were compared to those lifted from genome A to B. The requirement for a match between two domain boundaries was that the two areas either overlapped, or were less than 40 kb away from each other.
- The number of matches was counted and compared to the number of domain boundaries that got lifted. Dixon et al. found that 53.8 % of the boundaries lifted from hES to mES had a match, and 75.9 % of the boundaries lifted from mES to hES had a match.

These match percentages seem high, but the criterion for a match is not very strict. Since many of these domain boundaries are large areas (up to 400 kb), getting an overlap with any of the lifted areas is quite likely. Even though Dixon et al. showed the results to be statistically significant, we believe that the match percentage, using their definition of match, does not really reflect to what degree there is a similarity between domains in the mouse and human genome. It would be more interesting to know whether there are exact domains that have survived evolution. A slight overlap between two large areas does not really indicate that the area has survived evolution.

Since whole domains are likely to not get lifted over (because a too large portion of the domain does not exist in the other genome), or change the start or end position (because a portion of the domain does not get lifted), we believe it is better to investigate match between borders (not boundary segments). We propose the following analysis:

- Define the starting bin of every domain to be a domain border. We only include the start bin because also including the end bin would result in many borders being close to each other.
- Before lifting the borders, we represent them as genomic intervals. Lifting the whole domains first, and then creating border intervals, would lead to inaccurate positions of the borders, for the following reason. When an interval is lifted from one genome to another, some part of the interval might not exist in the other genome, resulting in only a part of the interval being lifted. The user can choose a minimum ratio of the interval that has to exist in the other genome for the interval to be lifted. A minimum ratio of 0.1 (which is the default) could lead to only a portion of 0.1 of the interval being lifted, resulting in the border being moved 0.9 of the interval length away from what

we would expect[18].

- We chose the intervals to be 40 kb before being lifted over. It makes sense to make the intervals as small as possible, as explained in the previous point. However, making them them too small leads to very few intervals complying with the minimum ratio of bases that must map. For instance, if the intervals are one base pair long, only about 50 % of the intervals get lifted.
- We define a match between genome A and genome B to be when a border lifted from A to B was close to a border already found in B. With *being close*, we assumed being closer than 20 kb away, since if two elements are inside the same bin of size 40 kb, they could be a maximum of 20 kb away from each other.
- We compute the percentage match between the two genomes as the ratio between the number of matches and the number of borders that got lifted [19].

This method indicates to what degree there is a similarity between domains in two genomes.

[18]A higher mean ratio could of course be chosen to avoid this problem, but then another problem would occur: very few domains would get lifted. For instance, when the *minimum ratio of bases that must remap* is set to 0.5, only 269 out of 3127 of the domains of Dixon et al. get successfully lifted.

[19]It can be debated whether this is the correct way of doing it, since there will be some borders that cannot be mapped, and the reason for failed mapping could be that the borders are not really a part of a topological domain. However, we will simplify things by not addressing this problem any further.

Chapter 4

Results and discussion

In Chapter 3 we developed a method for finding topologically associating domains (TADs), and created a consensus set of domains. In this chapter we analyse our domain set and those found by others, and compare the results. We do this partly by using the types of analyses presented in Section 3.5.

4.1 The model and method we have developed

As part of the results, we summarize the model and the method we have developed for finding TADs. We refer to Section 3.3 for details about the method, and an explanation of the terminology used here.

Model. TADs consist of contiguous bins on the genome, and contain loci that frequently interact with each other and less frequently interact with loci outside the TAD. They occur in different sizes, and can be nested inside each other. Areas that interact less with themselves than with their surroundings, after interaction frequencies are normalized[1], are not TADs.

Method. We find all TADs within a chromosome by recursively dividing the chromosome into areas that have few interactions with each other. At each step in the recursion, a division is made so that the two new parts have the lowest possible mean inter-domain interaction frequency (Equation 3.7). This procedure creates TAD candidates. TADs are chosen by selecting only those domains that interact more with themselves than with their brother domain, measured on the locally normalized interaction matrix. A consensus set is selected by first calculating a score for every domain. The score is the mean intra-domain interaction frequency measured on the normalized interaction matrix. The consensus set is found by choosing the set of non-overlapping domains that have maximum average domain score (Section 3.4.1). A hierarchical set is created by selecting the domains with highest score, allowing overlap at a maximum of three levels.

[1]See Section B.4.1

4.2 Selection of domain sets to compare with

The domain set produced by Dixon et al. is one of the sets we compare our set with, since their method is the most cited. We also include the method of Filippova et al., since they find hierarchical domains and argue that their method is better than the method of Dixon et al. Since the method of Rao et al. was published at the end of the period of this thesis, and since they use a different data set and find much smaller domains, we do not include their method. We choose not to include the method by Naumova et al., since their method is very similar to the method of Dixon et al. We also omit the method of Sexton et al., since they find domains in the fruit fly genome. Therefore, their analyses are not easily comparable to the analyses performed by Dixon et al. and Filippova et al. We do not include the method of Mizuguchi et al., which is based on an insulator score, because their study was published when our work was nearly finished. We believe that since this method finds domains by first finding domain edges, it is not very different from the method of Dixon et al.

Thus, we compare our domains to the domains found by Dixon et al. and Filippova et al. When we talk about *our domains*, we refer to the consensus set (Section 3.4.1).

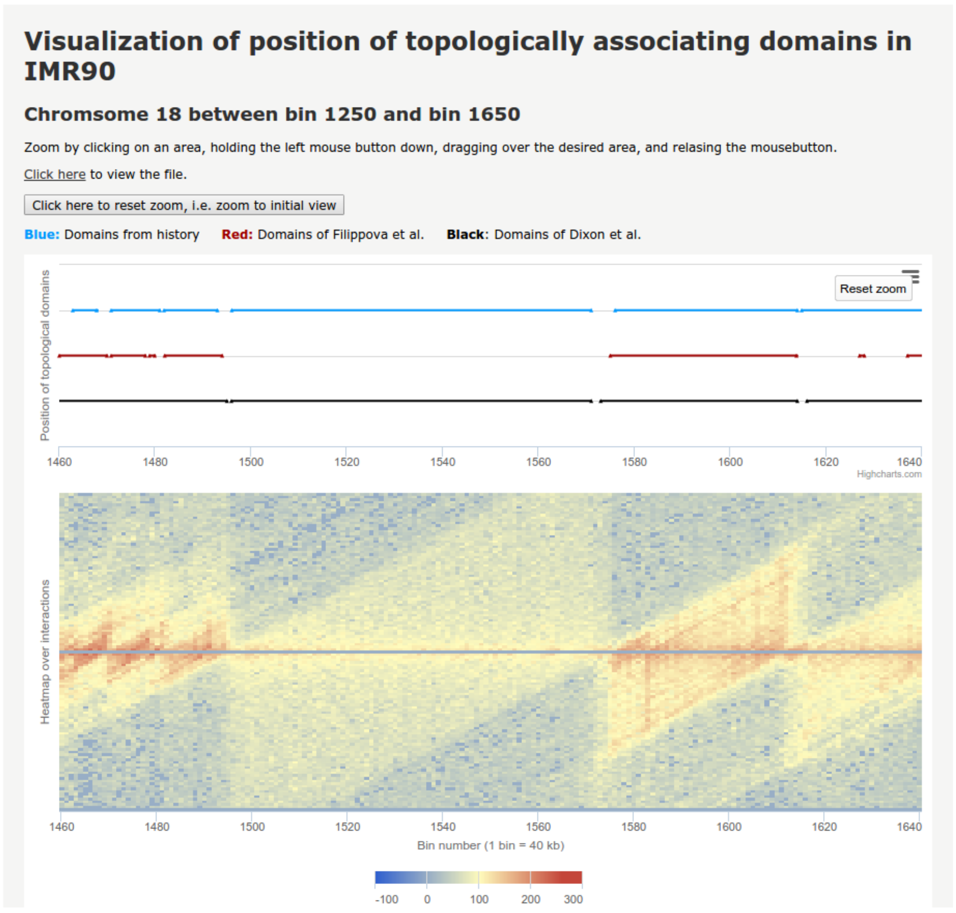
4.3 Tools in the Genomic HyperBrowser

We developed three tools in the Genomic Hyperbrowser [6]. These tools are on a separate installation from the main HyperBrowser, and can be found by following the url https://hyperbrowser.uio.no/3dml/. The tools are available from the main menu under *3DML tools*:

- *Find TADs*: The tool finds TADs in a chosen genome. The output is a simple web page with links that, when clicked on, will generate history elements that contain the consensus sets. A few other choices can be made that affect the algorithm. If no input parameters are changed, the tool will find TADs according to our selected method.

- *Visualize TADs*: The tool takes a history element that contains domains represented in the *bed-format*, e.g. a history element returned by the previous tool. It then visualizes the domains in a chosen part of a chromosome with the domains found by Dixon et al. and Filippova et al.

- *Analyse TADs*: The tool takes a history element that contains domains, and analyses these domains with the domains of Dixon et al. and Filippova et al. The analyses that are performed are mainly the same as those presented and discussed in this chapter.


Visualization of position of topologically associating domains in IMR90

Chromsome 18 between bin 1250 and bin 1650

Zoom by clicking on an area, holding the left mouse button down, dragging over the desired area, and relasing the mousebutton.

Click here to view the file.

Click here to reset zoom, i.e. zoom to initial view

Blue: Domains from history Red: Domains of Filippova et al. Black: Domains of Dixon et al.

Reset zoom
Position of topological domains
1460
1480
1500
1520
1540
1560
1580
1600
1620
1640
Highcharts.com
Heatmap over interactions
Bin number (1 bin = 40 kb)
-100
0
100
200
300

Figure 4.1: Screen shot of the tool in the Genomic HyperBrowser.

4.4 Comparing and analysing domain sets

A page in the Genomic Hyperbrowser lists history elements that show the analyses that are performed in this section (url: https://hyperbrowser.uio.no/3dml/u/ivar/p/master). The history elements on the page can be exported and one can easily re-run the analyses from these history elements, choosing different domain sets if wanted. Some of the figures in this chapter can be found as history elements on this galaxy page. When a figure is available online, the text “HyperBrowser history: X” is added in the Figure caption, where X is the number of the history element under the *Figures* history on the galaxy page.

The following table lists key characteristics of the three domain sets from IMR90.

	Our	**Filippova et al.**	**Dixon et al.**
Part of genome covered[2]	86.4 %	61.5 %	87.3 %
Number of domains	3875	5408	2349
Median domain size	240 kb	160 kb	840 kb

[2]Number of bins covered by domains divided by number of bins in the genome

Size standard deviation	1154 kb	482 kb	892 kb
Mean density[3]	35.4	35.2	19.3
Mean density (normalized data[4])	4.94	4.39	-0.708
Mean inter-domain int. freq.[5]	3.57	4.00	2.34
Mean inter-domain int. freq. (normalized data)	-9.85	-8.76	-9.96
Mean of mean density minus mean inter-domain int. freq.[6]	31.8	31.2	17.0
Mean of mean density minus mean inter-domain int. freq. (normalized data)	14.8	13.1	9.25

4.4.1 Domain size

On average, the domains of Dixon et al. are notably larger than the domains from the two other sets. The *directionality index* used by Dixon et al. includes interactions 50 bins upstream/downstream, meaning that when measuring interactions for a bin at the edge of a domain containing less than 50 bins, bins outside the domain will affect the score. For many small domains, the score can be affected so that the domain edge will not be marked as an edge by the algorithm. This is probably one of the reasons why few small domains are found by the method of Dixon et al. Also, since the method does not use domain density as a criterion for domains to be selected, domains with low densities are found. These are generally large and will increase the average domain size.

We find bigger domains than Filippova et al. One reason might be that Filippova et al. omit all domains that have lower intra-domain interaction frequency than the average for domains of the same size. This results in a smaller set with dense domains, which are usually small. Also, the method of Filippova et al. seems to split domains into several smaller domains. We will discuss the reason for this at the end of this chapter.

When selecting our consensus set, we chose to maximize the average domain score over all domains, as discussed in Section 3.4.1. We also presented a more greedy algorithm, where the domains with the highest scores were chosen first. This greedy selection results in a median domain size of 120 kb[7], half of the median size of domains in our consensus set. This seems to be because the most dense domains, independent of size, are small domains. These are selected first, making room for

[3]*Mean density* is the mean of the mean intra-domain interaction frequencies for all domains.

[4]See Section B.4.1 for details about the *normalized data*.

[5]The mean interaction frequency that bins in selected domains have with bins that are outside the domains and closer than 50 bins away, averaged over all domains.

[6]This is calculated by first calculating the mean density of every domain and subtracting the mean inter-domain interaction frequency, limiting interactions to 50 bins away. Then the mean of these numbers is calculated over all domains.

[7]See history element 12 under the history *Analysis presented in Chapter 4* in the Genomic HyperBrowser.

selection of other small domains that do not overlap with the first selected domains (big domains are more likely to overlap with already selected domains).

4.4.2 Agreement in position of domains

We were interested to know how much the three sets *agree*, i.e. to what degree domains in one set appear at approximately the same position in the other sets. We use the definition of match proposed in Section 3.5.2, and count the number of times that domains from one set are found in the other sets. The result is illustrated as a Venn-diagram in Figure 4.2, indicating that a higher proportion of our domains match the domains of Filippova et al. than the domains of Dixon et al.

Differences and similarities can also be studied visually. Figure 4.3 shows an example of a part of the genome where the three sets mostly agree, and Figure 4.4 shows an example where the three sets agree less. This figure is also an example of how we and Filippova et al. find more small domains, whereas Dixon et al. find fewer bigger domains.

Further, we measure the number of domains in one set that overlap a domain border in another set, as presented in Section 3.5.2. We find that 34 % of our domains overlap a domain border of Dixon et al. and 47 % overlap a domain border of Filippova et al. Among the domains of Filippova et al., 27 % overlap borders of Dixon et al. and 55 % overlap our domain borders. For the domains of Dixon et al., 55 % overlap our borders and as many as 65 % overlap borders of Filippova et al. This may indicate that our method in some way is more similar to the method of Dixon et al. than the method of Filippova et al. The reason can also be that Dixon et al. find larger domains, making it less likely for our domains to overlap their domain borders than the borders of the smaller domains of Filippova et al.

4.4.3 Density of domains

We compute the average of the mean intra-domain interaction frequencies, both using the raw interaction matrix and the normalized interaction matrix, as proposed in Section 3.5.1. Our domains have approximately the same average interaction frequencies as the domains of Filippova et al ($\approx$ 35), when calculated on the raw interaction matrix. The domains of Dixon et al. have lower average interaction frequency ($\approx$ 19). Calculated on the normalized data matrix, our domains are denser than those of Filippova et al. (4.9 vs. 4.4).

We also do the calculations on a smaller domain set that covers the same portion of the genome as the set of Filippova et al. (where we have removed the least dense domains from our set). The mean density of this smaller set is 39.4 calculated on the raw data and 7.19 calculated on the normalized data matrix — both noticeably larger than the numbers for the domains of Filippova et al.

As proposed in Section 3.5.1, we measure the average density while shrinking the domain sets by removing the least dense domains, to see how the mean density changes according to how much of the genome is covered. Figure 4.6 shows the

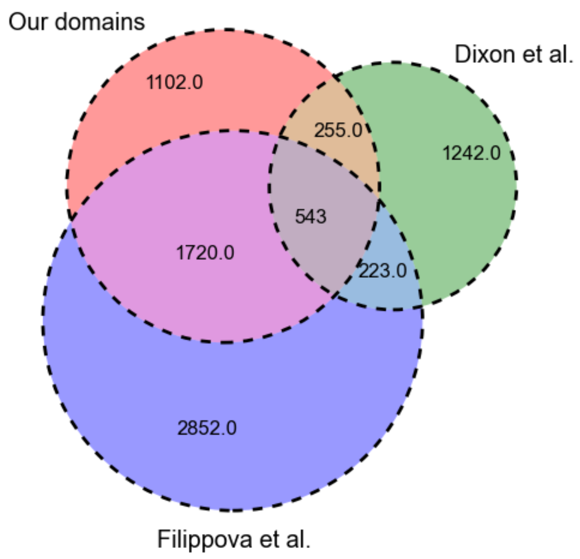


Figure 4.2: Match between us, Dixon et al. and Filippova et al. 543 domains are found by all three.

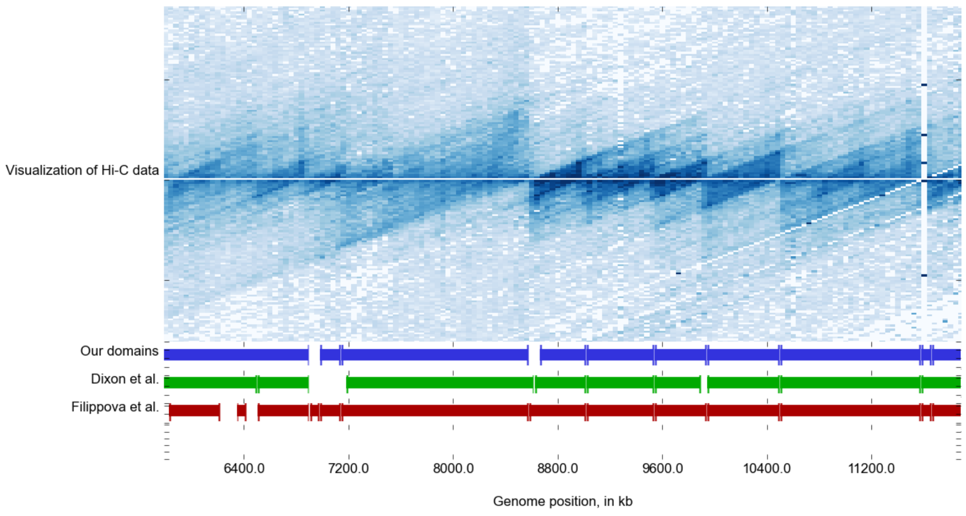


Figure 4.3: An example of agreement between us, Dixon et al. and Filippova et al. Most of the domains visualized here appear in all three sets, which may be because the area consists of what visually seems to be well-defined TADs. The area visualized is a part of chromosome 18 (IMR90). HyperBrowser history: 8

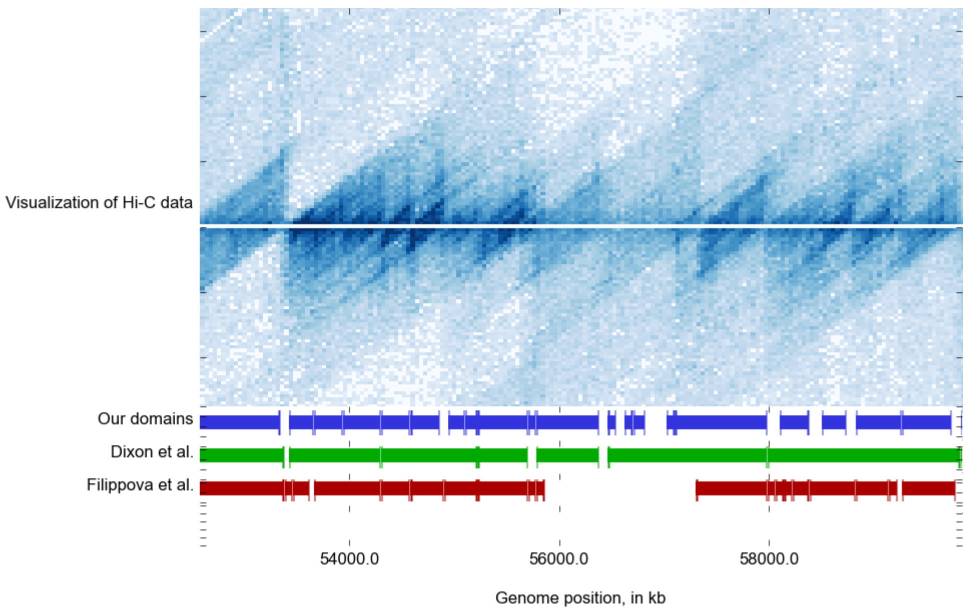
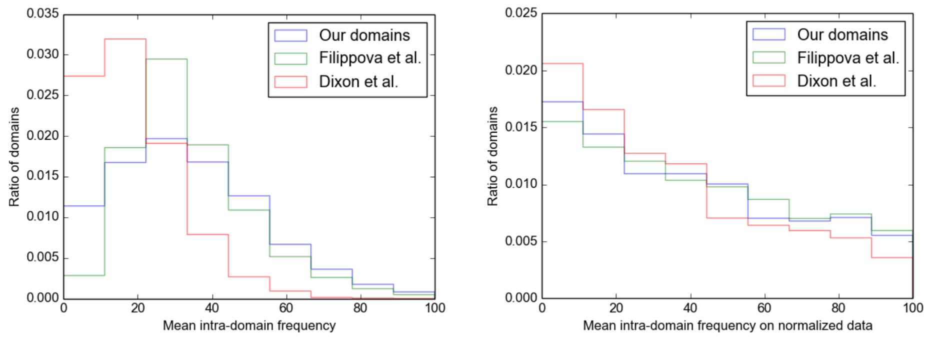


Figure 4.4: An example of disagreement between us, Dixon et al. and Filippova et al. The area visualized contains what visually seem to be TADs nested inside TADs. Some domain edges are selected by all three methods, but few domains appear in all three sets. The area visualized is a part of chromosome 18 (IMR90). HyperBrowser history: 6



(a) *(b)*

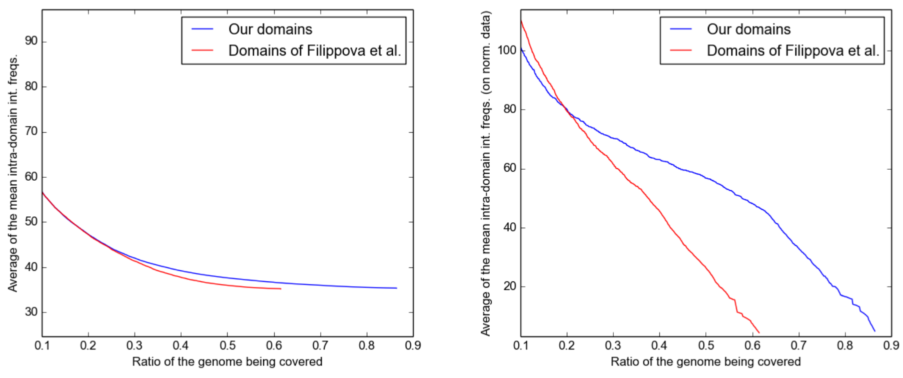
Figure 4.5: Histograms of domain density calculated over the raw data matrix (a) and the normalized data matrix (b) for the three domain sets. The most visible trend is that our domains and the domains of Filippova et al. are denser than those of Dixon et al.

result of this process, indicating that our domains are generally denser than those of Filippova et al. when covering the same portion of the genome.

These results show that our domains are generally denser than the domains of Filippova et al. and Dixon et al.

4.4.4 Inter-domain interaction frequency

We have described TADs as areas with many intra-domain interactions and fewer interactions with their surroundings. Of the three sets, the domains of Dixon et al.



(a) *(b)*

Figure 4.6: Average domain density plotted against the portion of the genome that the domains cover, calculated on raw data matrix (a) and normalized data matrix (b). The curve for the domains of Dixon et al., which is much lower than the plotted curves, is not included in the plots.

have lowest average inter-domain interaction frequency, i.e. on average interacting the least with their surroundings. This is probably due to the fact that the domains themselves are not very dense and are often positioned in parts of the genome where there are generally fewer interactions between bins. However, when calculated on the normalized data matrix, our domains have approximately the same average inter-domain interaction frequency as the domains of Dixon et al. Our domains interact less with their surroundings than the domains of Filippova et al, also when calculations are done on the normalized data matrix.

We also calculate the *mean of mean density minus mean intra-domain interaction frequency*, as suggested in Section 3.5.1. This measures the average of how dense the domains are compared to their surroundings, and further illustrates to what degree the domains interact more with themselves than with their surroundings. Our domains come out better than the domains of Filippova et al. and Dixon et al., both when calculated on the raw and the normalized interaction matrix (see the table at the beginning of this section).

4.4.5 CTCF analysis

Dixon et al. showed that the areas between TADs are enriched with the insulator protein CTCF (see Section 2.2.4). We were interested in whether this is also true for our domains, and whether there are any remarkable differences in the degree of association with CTCF binding sites.

The CTCF data set that Dixon et al. and Filippova et al. used in their analyses is the one of Kim et al. (2007) [26]. There are many publicly available CTCF data sets from the same cell-line, and we wanted to include more than one set in our analysis, to make it more robust, and so that we could identify potential differences between the sets. Also, the set of Kim et al. (2007) is relatively old, and contains fewer CTCF binding sites than there are in other data sets. Thus, we collected four additional CTCF data sets from the ENCODE Project [27], one from each of the four labs represented in the database. We also generated a random CTCF data set, containing sites at random positions. To ease notation, we assign a number to each dataset, shown in the table below.

#	**Lab / origin**	**Number of sites**[8]	**GEO accession number**[9]
1	Kim et al. (2007) [26]	13 720	GSE5559
2	Bradley Bernstein, Broad	50 031	GSM1003558
3	Richard Myers, HAIB	40 557	GSM803333
4	Vishwanath Iyer, UTA	55 295	GSM822307
5	John Stamatoyannopoulos, UW	45 697	GSM945243
6	Randomly generated set	50 000	-

Table 4.2: CTCF data sets

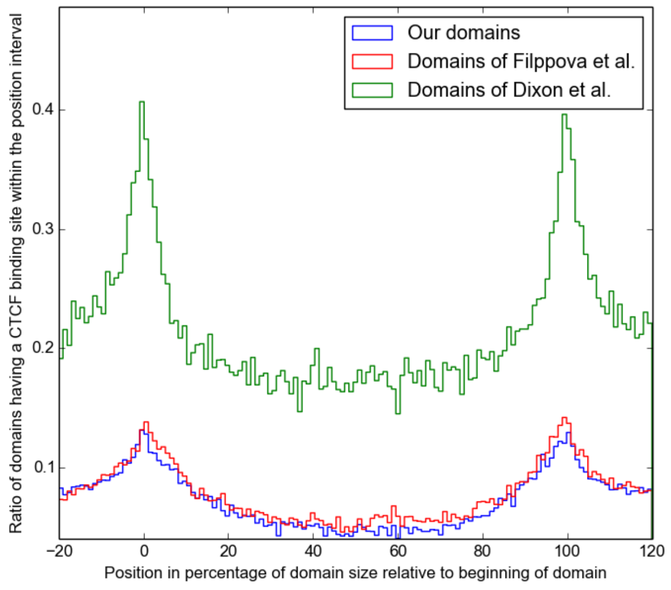
We create the normalized aggregation plots, proposed in Section 3.5.3, showing where CTCF binding sites frequently occur relative to the domains. Figure 4.7 a shows the results, with the ratio of domains on the y-axis, leading to a higher count for the domains of Dixon et al. that are generally larger — thus containing more domain bins for every histogram bin. In the second plot (Figure 4.7 b), the count is normalized so that the histogram sums to one. Both figures clearly show that the enrichment of CTCF binding sites greatly peaks near domain edges and decreases inside domains and outside domains. The histogram representing the domains of Dixon et al. shows a slightly higher ratio within domains, compared to the curve for the other domain sets. This may be because these domains are larger, and may contain smaller domains. Another interesting observation is that CTCF generally appears less often inside domains compared to outside domains, for all domain sets. It is clear that CTCF is more associated with *domain edges* than with the *domain boundary segments*.

We also performed the same analyses for all other CTCF data files. The result is shown in Figure 4.8. All data sets indicate the same. An interesting trend is that the domains of Dixon et al. seem to peak slightly more at the beginning of domains compared to the other sets, whereas all three domain sets seem to peak about the same at the edges of domains.

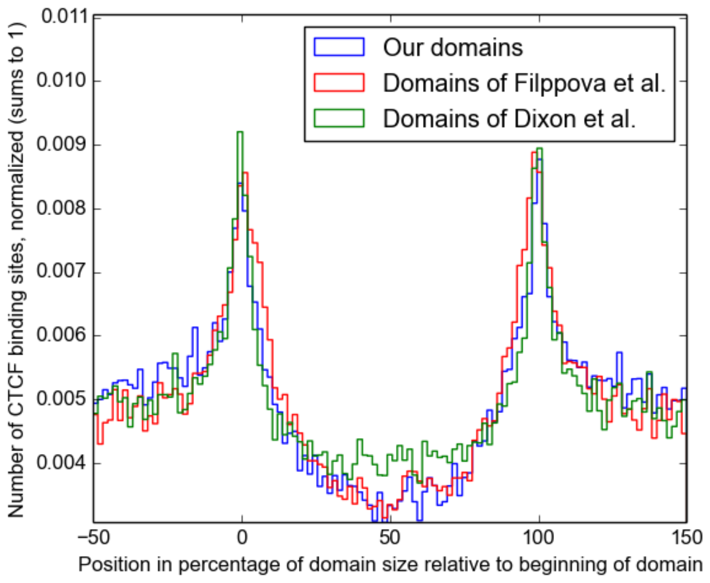
Even though CTCF is positioned more often close to domain boundaries than elsewhere, only 10 to 20 % of domain boundaries are located less than 10 kb away from a CTCF binding site. We conclude, as Dixon et al., that domain edges and boundaries are enriched with CTCF binding sites, but *CTCF binding alone is insufficient to demarcate domain boundaries* [5]. Figure 4.9 shows the position of CTCF binding sites from CTCF data set 1 together with the data. The figure supports this conclusion — CTCF binding sites are spread across the genome, not only positioned near domain edges.

[8]If the intervals were originally referenced on hg19, this is the number of CTCF sites after intervals are lifted to hg18

[9]Data sets can be found by using the GEO accession number in the search tool on http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi.



(a)



(b)

Figure 4.7: (a): Aggregation plots of the position of CTCF binding sites (data set 1) relative to the starting position of normalized domains. (b): The same as in (a), but the ratios are normalized so that they sum to one.

4.4.6 Similarities between TADs in the mouse genome and the human genome

We performed the analysis proposed in Section 3.5.4, counting how many domains that are found in the human genome also exist in the mouse genome (and vice versa).

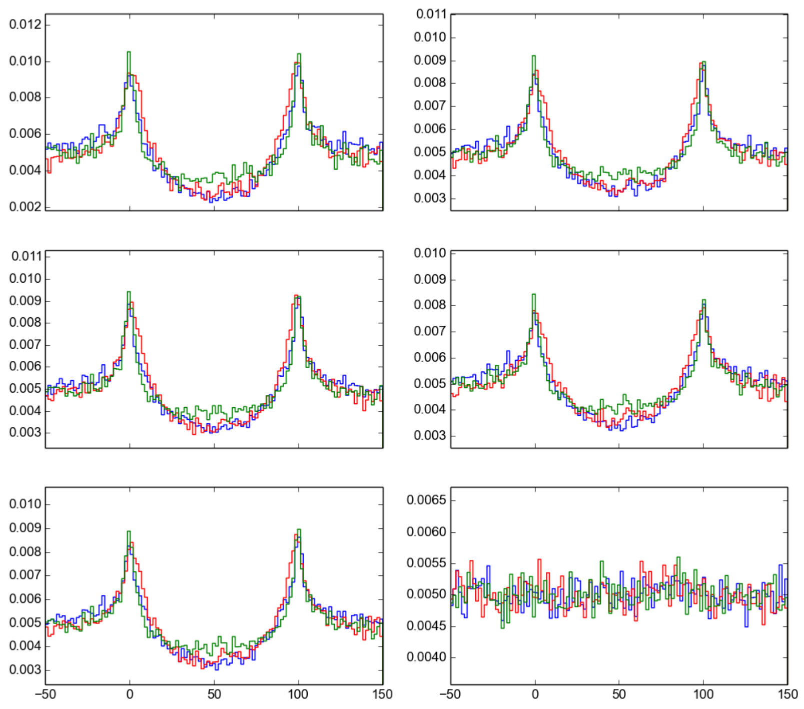


Figure 4.8: *Same as Figure 4.7 (b), one plot for each CTCF data set from Table 4.2. Domains of Filippova et al. (red), Dixon et al. (green) and our domains (blue).*

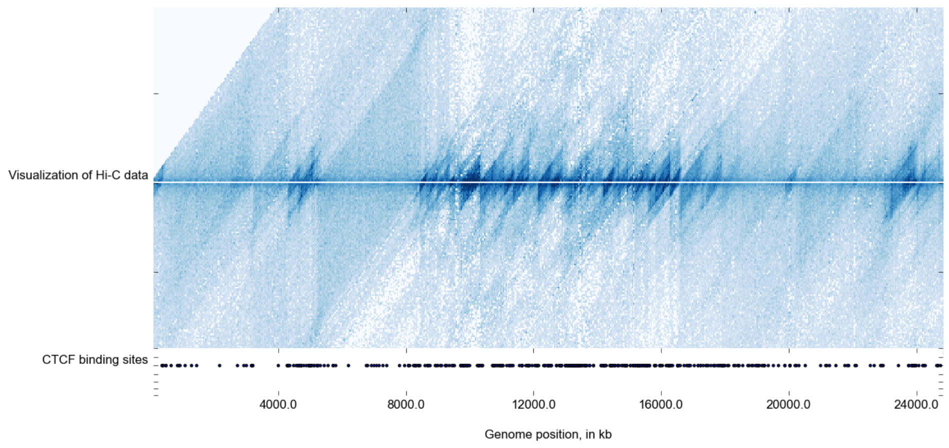


Figure 4.9: *Position of CTCF binding sites with heat map of the data matrix (chromosome 3, IMR90), illustrated here with a point at the centre of the interval of the binding site.*

We compared our domains with the domains of Dixon et al., making the comparison between the mES and hES cell lines. Since Filippova et al. have not published domains from the hES cell line, we carried out an analysis between IMR90 and mES with their domains. We summarize the results by calculating the ratio between the number of matches and the number of domain borders that were lifted, which are listed in Table 4.3 and Table 4.4. See Section B.3 for more details.

	From mES to hES	**From hES to mES**
Domains of Dixon et al.	17.4 %	14.4 %
Our domains	21.7%	18.8 %

Table 4.3

	From mES to IMR90	**From IMR90 to mES**
Domains of Filippova et al.	16.4 %	23.7 %
Our domains	15.2 %	19.7 %

Table 4.4

The analyses were performed in the Genomic HyperBrowser. A hypothesis test was run, concluding that the match was significant for all domain sets with p-value 0.0. Based on the numbers listed in Table 4.3 and Table 4.4, the domains of Filippova et al. seem to be the most consistent between the mouse and the human genome.

4.4.7 Consistency between human cells

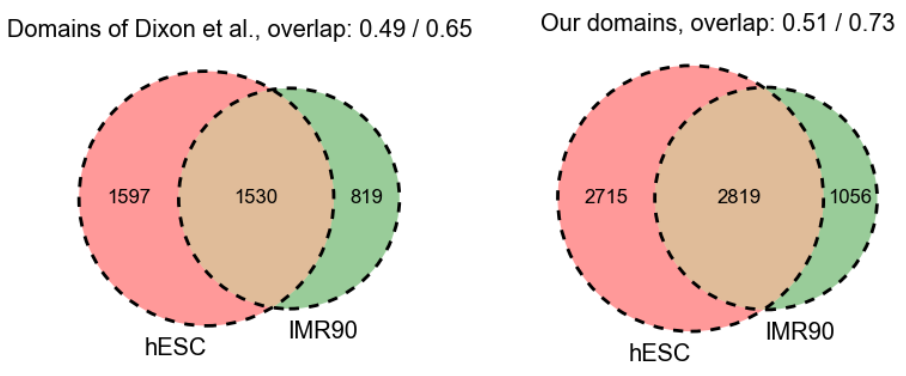
An algorithm for finding TADs should be stable, i.e. not perform differently or give very different output if the input changes slightly. One way of measuring the consistency of an algorithm, is to see how the results differ between two cell types of the same species. Thus, we compared the domains found in hES with those found in IMR90.

We define a match to be when a domain is found at the same position in the other cell line, only allowing one bin change in domain boundaries. The results for the human genome are shown in Figure 4.10, which indicates that our method is slightly more consistent than the method of Dixon et al.

4.4.8 Summary

Our domains are similar to those found by Dixon et al. and Filippova et al. Based on median size and average domain density, our domains are most similar to those of Filippova et al. We argue that our domains fit better with the fundamental principles about TADs. They are generally more self-interacting and interact less with their surroundings than the domains of Dixon et al. and Filippova et al. Our smaller set, covering the same portion of the genome as the set of Filippova et al., has even denser domains.

We see a significant consistency between the mouse and the human genome for all three sets. The domains of Filippova et al. seem to be slightly more consistent than our domains. We see a distinct enrichment of CTCF binding sites near the domain borders for all three sets. CTCF binding sites seem to be less associated with the domain boundary segments than with the domain borders, and occur even less often inside domains. We note that even though CTCF binding sites commonly occur near



(a) *(b)*

Figure 4.10: Consistency between cell replicates (hES and IMR90) in the human genome. Our domains (b) seem to be more consistent than those of Dixon et al.

domain borders, domain borders are not dependent on or cannot be demarcated from these.

Our domains are more consistent between the IMR90 and hES cell line than the domains of Dixon et al., possibly indicating that our method is more robust and less affected by small changes in the input data.

In this section, we have analysed our consensus set in IMR90. Using our tool in the Genomic HyperBrowser, analyses on other cell lines with other domain sets can be performed. We also performed similar analyses in the mouse cell line mES. The results can be found under the history *Analyses of mES* on the galaxy page. The results for mES indicate the same as for IMR90, that our domains are more self-interacting and interact less with their surroundings.

4.5 Understanding weaknesses of the methods

The previous section presented analyses of the domains that are produced by the different methods. We now discuss possible weaknesses of the methods. Some of these weaknesses are reflected in the analyses from the previous section, and some are inherent in the methods.

4.5.1 Dixon et al.

To understand what really happens in the algorithm created by Dixon et al., we need to briefly describe what the Baum-Welch algorithm does (see Appendix A for details). Given a predefined number of states, three in this case, it finds the suboptimal underlying model (transaction probabilities, a priori state probabilities, and the probability distribution for the observation for every state) given the observations that are made. In this case, this is the directionality index (DI) for every bin. An interpretation of this process is that all bins are classified into one out of three possible states, where the state chosen for a bin is dependent on the state for the previous bin and the observation at the current bin. Simply put, the method aims to discover domain edges in a rather complicated way by using a HMM.

The method does not take into account that TADs are nested hierarchically, and can fail in cases like the following: Assume that a small domain is nested somewhere in the middle of a larger domain. A downward biased state marks the beginning of the big domain. Non-biased states will be dominant until the small domain begins. The problem occurs at the end of the small domain, before the large domain has ended. If the small domain is dense enough, the end of it will consist of an upstream biased state, and the method will fail to capture any of the domains as a single domain. An example of this behaviour is seen in Figure 4.11. The domain marked in the figure is split into two domains by Dixon et al. These two domains do not seem visually to be TADs.

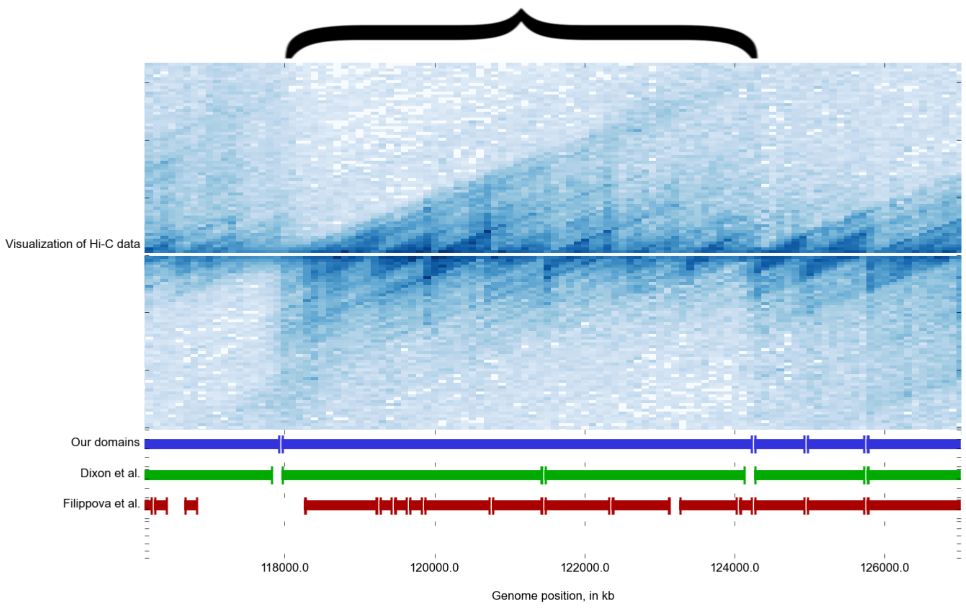


Figure 4.11: Visualization of domains on chromosome 8 (IMR90). What visually looks like a big domain is marked with a bracket. Dixon et al. do not find this domain. HyperBrowser history: 5

A point is that the DI is always computed from interactions that are 50 bins upstream and downstream[10]. When the DI is computed inside the domain marked in Figure 4.11 (the size of the domain is approximately 150 bins), it does not include information from the beginning or end of the domain. The choice of starting a new domain in the middle of the marked domain, is not based on the information from all bins in the domain.

Limiting the range of bins included in computing the DI is a problem, since this will

[10]Dixon et al. say that they choose 50 bins *because these parameters maximize the reproducibility of the DI and the domain calls while retaining a sufficiently high resolution to identify domains and boundary regions* [5].

favour domains of a certain size, and behave differently for domains of different sizes. The decision of the state on a domain edge will always be made based on 50 bins upstream/downstream. This means that domains bigger than 50 bins will have only a part of their domain included in the calculations, whereas the DI for the edge of a domain smaller than 50 bins will be affected by the whole domain as well as a part outside it. This problem, deciding the range of interactions to include, is a general problem of edge based approaches, as we saw when proposing our first approach (Section 3.3.2).

4.5.2 Filippova et al.

Filippova et al. avoid the pitfall of only looking at domain edges, and instead find domains by looking at the intra-domain interaction frequency. They handle the problem of nested hierarchical domains by discovering domains on different scales. Thus, they do not limit the method or define TADs to be on only one scale, the way Dixon et al. implicitly do. Filippova et al. also have a much more direct way of defining the optimal set of TADs. They actually solve an optimization problem — thus finding the optimal set from their definition of optimal.

Let us recall how Filippova et al. rank domains, in order to discuss a possible problem. The score for a domain is the sum of interactions within the domain divided by the size of the domain to the power of γ, minus the average score of all domains of the same size. The definition makes sense in some way, because a TAD is characterized by having many intra-domain interactions.

But what about inter-domain interactions? The number of inter-domain interactions is not directly included in the score function. This means that a dense domain A placed inside a bigger domain B with the same density will have exactly the same score as an independent isolated domain with the same density and size as A somewhere else on the genome. The fact that A interacts frequently with its surrounding bins and is indeed just a part of a bigger TAD, will not affect its score. Will domain A be preferred in favour of domain B using this method? Not necessarily, because the method maximizes the sum of domain scores. If domain B has a higher score than A, it will be correctly chosen. But if γ is set to favour smaller domains, can the smaller domain A be wrongly selected as a TAD?

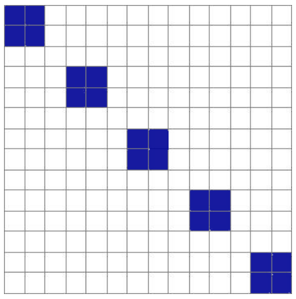


Figure 4.12: Heat map of the interaction matrix for a test scenario. Coloured pixels indicate interactions with frequency 1 *and white pixels indicate interactions with frequency* 0.

It is difficult to provide an answer to this question directly without having all the necessary information: interaction frequencies within the domains, and the average score of domains of different sizes across the genome. However, we can generate a test scenario to check whether the method fails in a similar case: Assume our area of interest consists only of TADs of size r separated by an area with few interactions, of size $r/2$ (Figure 4.12). The interesting question is whether there is a choice of γ that will divide these TADs of size r into two separate TADs of size $r/2$. We can check this by calculating the score functions q_A and q_B (Equation 2.5, page 10) for the small domains and big domains respectively. For simplicity, we assume that all interactions between bins within the domains are 1, and all other interactions are 0 (in reality, interactions far from the diagonal will be weaker, however that will favour smaller domains, so we are not favouring a division of the domains when assuming these interaction counts).

First, we calculate μ_A and μ_B, the averages of all possible domains of size $r/2$ and r:

$$\mu_A = \frac{(\frac{r}{2})^2 \cdot 2 + 0}{3} = \frac{r^2}{6}$$

$$\mu_B = \frac{r^2 + 2 \cdot (\frac{r}{2})^2}{3} = \frac{r^2}{2}$$

We then calculate the score for the small domains (A) and big domains (B):

$$q_A = \frac{(\frac{r}{2})^2 - \frac{r^2}{6}}{\left(\frac{r}{2}\right)^\gamma} = \frac{2^{\gamma-2} \cdot r^{2-\gamma}}{3} \tag{4.1}$$

$$q_B = \frac{r^2 - \frac{r^2}{2}}{r^\gamma} = \frac{r^{2-\gamma}}{2} \tag{4.2}$$

The question is whether a γ exists so that two domains of size $r/2$ will be preferred instead of one domain of size r. For this to happen, the inequality $2q_A > q_B$ has to hold:

$$2q_A > q_B \tag{4.3}$$

$$\implies 2^\gamma > 3 \implies \gamma > \log_2 3 \approx 1.58 \tag{4.4}$$

It turns out that choosing $\gamma \gtrsim 1.58$ will result in non-TADs being classified as TADs in this case. One can argue that $\gamma > 1.58$ is not used in practice, but the example illustrates an important problem with the method: By not including inter-domain interactions in the formula, large domains that should not be divided can be broken into pieces. By considering only intra-domain interactions, TADs are not valued by their ability to be independent pieces of DNA that less frequently interact with other parts of the genome. This fault seems to be visible, see Figure 4.14.

In general, it is problematic that inter-domain interactions are not directly included in the algorithm, resulting in domains being selected that interact as much with their surroundings as with themselves.

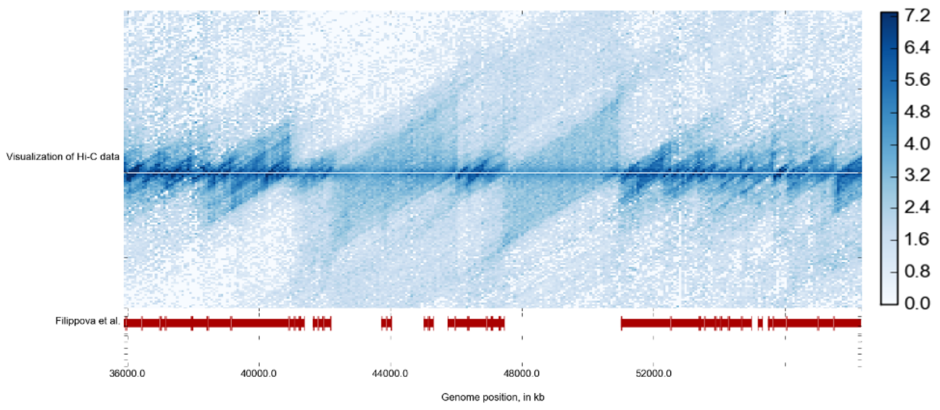


Figure 4.13: Visualization of interactions for bin 984 to 1367 on chromosome 20 with the domains of Filippova et al. Two visible big domains that could be interesting are not chosen by the algorithm to be TADs. HyperBrowser history: 7

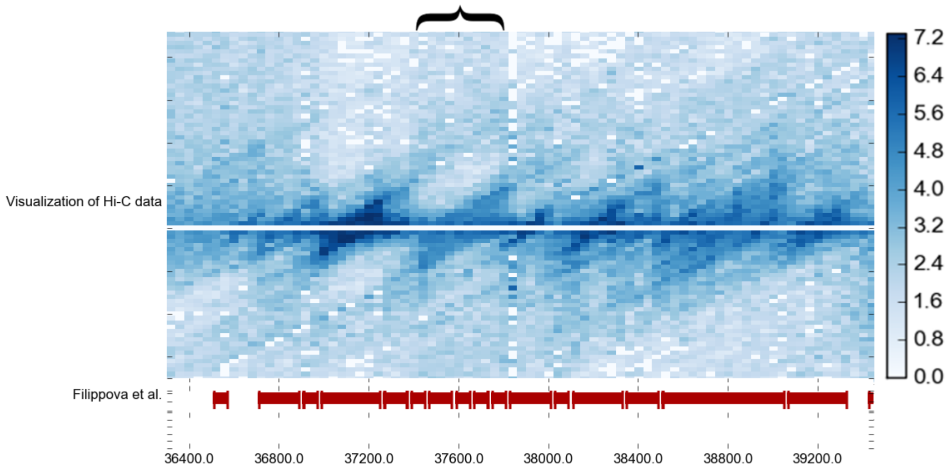


Figure 4.14: Bin 900 to 1000 on chromosome 3 with the domains of Filippova et al. In the centre of the area visualized, what seems like a TAD has been divided into four smaller domains. HyperBrowser history: 3

Domain density. Another questionable part of the method is the minimum density criterion: An area needs to have more interactions than the expected amount of interactions for areas of the same size, in order to be included in the domain set. There is no obvious answer to the question of how dense a domain should be in order to be regarded as a TAD, but the criterion used by Filippova et al. is not rooted in any definition or general agreement about TADs. A lot of potential domains are left out because of this criterion (Figure 4.13).

It is interesting that our domains are denser, when the algorithm of Filippova et al. directly aims to find domains that are dense compared to domains of the same size, whereas our algorithm in a less greedy manner first divides the genome into parts that have few interaction with each other, and then filters out dense domains from that set, without having any minimum density criteria. We believe the reason must have to do with how the consensus set is selected from multiple sets of domains on different scales. If γ in the score function they use (Equation 2.6) is set so that the denominator in the equation is proportional to the expected sum of intra-domain interactions for domains of the given size, then the set chosen by maximizing Equation 2.7 will have the maximum possible size-adjusted mean density. This would require $\gamma > 1$ (a simulation shows $\gamma \approx 1.5$). However, the consensus set of Filippova et al. is chosen by selecting the most consistent (most frequently occuring) domains when γ is varied stepwise between 0 and 1. Even though Filippova et al. point out that they seek dense domains, γ is not chosen so that the most dense domains are found. As shown, a γ around 1.6 will result in homogeneous domains being split into several domains. Paradoxically, the algorithm that initially tries to find the set of the most dense domains (adjusted for size) fails to do this if γ is set accordingly, and instead ends up using a set of γs that do not fulfil the original purpose. Thus, the consensus set of Filippova et al. will not contain the most dense domains, which is reflected in the analysis done in Section 4.4.3.

4.5.3 Our method

We have criticized the minimum criterion used by Filippova et al. Finding a suitable criterion roots in the more general problem of considering what a TAD really is. This is a question we need to ask when we choose the selection criterion in our method, deciding whether an area really is a domain or not.

We decide to call an area a TAD as long as it interacts more with itself than with the other area inside the same mother domain, after normalizing the interactions, as explained in Section 3.3.6. We already assume it interacts more with itself than with any domain outside the same mother domain, since the mother domain has been selected by minimizing inter-domain interactions.

We see that this selection criterion sometimes results in domains being found that we visually do not spot by looking at the visualization of the data. These domains have slightly higher average intra-domain interaction frequency than the average interaction frequency with their brother domain. It can be debated whether our criterion is too “slack”, allowing the division and shrinking of domains to go too far. The challenge is the lack of a definite answer of what a TAD really is. Maybe future knowledge and research will result in a stricter criterion.

We also notice that our method might fail in cases where non-contiguous domains have frequent interactions. In Figure 4.15 we have marked three domains — A, B and C — with the data matrix from a part of chromosome 3. The three domains we have marked are what we believe to be all the direct subdomains of this area. Thus, we would assume our method to first divide either between A and B or between B and C. However, since A seems to interact more with C than with B, there might

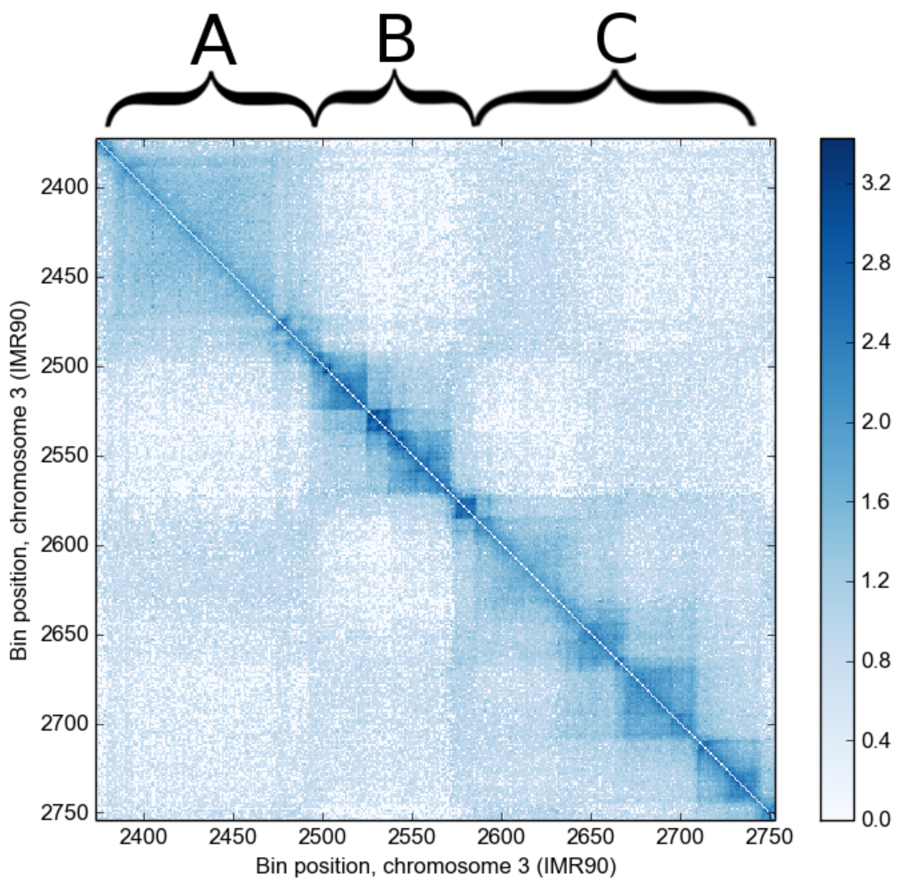


Figure 4.15: Three domains in a part of chromosome 3 (IMR90). Domain A interacts more with domain C than with B.

be a bin somewhere inside B where the score function (the minus of the average inter-domain interaction frequency) is larger, resulting in domain B being split. Our method does not actually end up splitting the middle domain in this case. This is only an example to illustrate the problem. We have not come up with any solution to this problem, which is rooted in the more general problem of non-contiguous domains (Section 3.3.8).

4.6 Algorithm and method complexity

When finding domains from a Hi-C matrix of size $m \times m$, our method (in the first step of the recursion) iterates over m bins. For every bin, it counts at maximum m interaction frequencies (assuming dynamic programming is used, see Section 3.3.5). At the next step in the recursion, $m/2$ bins are iterated over for every bin m, and there is a maximum of $\log_2(m)$ recursions. At step k in the recursion, m/k bins are iterated over for every bin m. Thus, the worst case complexity is of the order $O(\log_2(m) \cdot m^2)$.

The complexity of the algorithm of Filippova et al. is of the order $O(m^2)$ as stated in [13].

The complexity of the Baum-Welch algorithm on each iteration is a linear function of the length of the sequence, which is m. We assume a finite number of iterations, independently of m. Thus, the complexity of the method of Dixon et al. is of the order $O(m)$.

Our method is the most complex in terms of computations required, but it is only slightly more complex than the method of Filippova et al. Run on an ordinary desktop machine, it only takes a few seconds to find domains in a chromosome using our method.

We argue that the method of Dixon et al. is most complex in terms of the *model* it uses, by relying on the somewhat complicated hidden Markov model to find domain edges. The method of Filippova et al. and our method use simpler techniques that are easier to understand.

Chapter 5

Conclusion

Finding topologically associating domains (TADs) is only one way of exploring interaction patterns in the vast amount of data generated by Hi-C. While TADs seem to be fundamental building blocks of the three-dimensional structure of the genome, identifying a set of TADs is also a way of extracting features from the Hi-C data, and processing it into a format that easily can be analysed further.

We have presented a new method for finding TADs, which takes Hi-C data as input and returns a consensus set of domains and a set of hierarchically nested domains. The method is based on the principle that a TAD is a spatially compact cluster of chromatin, where loci inside the area frequently interact with each other and less frequently interact with loci outside the area.

As pointed out by Filippova et al. [13], it is apparent that domains occur in different sizes and are nested inside each other. Thus, we conclude that pure edge based methods, like the one created by Dixon et al. and the one created in our first approach, are not appropriate for identifying these domains.

Compared to the domains found by Dixon et al. and Filippova et al., our method returns domains that on average have a higher amount of intra-domain interactions and fewer inter-domain interactions. We also investigated how CTCF binding sites are associated with these domains. We conclude that our domain borders are about as much associated with CTCF binding sites as the domain borders of Dixon et al. and Filippova et al. By creating aggregation plots over domains that are normalized in size, we see that CTCF binding sites most often occur at the domain borders, occur less often outside domains and even less often inside domains. We propose that comparison between domains in the mouse and the human genome should be done by comparing domain borders, not domain boundary segments, like Dixon et al. did. Our domain borders are more consistent between the mouse and the human genome than the borders of Dixon et al. However, the borders of Filippova et al. seem slightly more consistent than our borders. Based on the fact that our method finds domains that are more self-interacting, and because of the weaknesses of the methods of Dixon et al. and Filippova et al., we argue that our method is a preferable method.

For future work, it would be interesting to use our method on even higher resolution

Hi-C data, like the recently published data by Rao et al. [14]. It would also be interesting to look at genome topology “the other way around”: Instead of finding TADs and investigating biological features linked to them, it would be interesting to see how the genome topology relates to predefined biological features. For instance, it would be interesting first to find areas that are consistent between the mouse and the human genome, and to see how these are related to the topology of the genome. This kind of analysis could discover new potentially interesting interaction patterns, not necessarily TADs. It would also be interesting to investigate further another interaction pattern, which is similar to TADs, that is non-contiguous domains.

Bibliography

[1] Charles Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life.* London: John Murray, 1859.

[2] John C Venter, Mark D Adams, Eugene W Myers, Peter W Li, Richard J Mural, Granger G Sutton, Hamilton O Smith, Mark Yandell, Cheryl A Evans, Robert A Holt et al. 'The sequence of the human genome'. *Science* 291.5507 (2001), pp. 1304–1351.

[3] Eric S Lander, Lauren M Linton, Bruce Birren, Chad Nusbaum, Michael C Zody, Jennifer Baldwin, Keri Devon, Ken Dewar, Michael Doyle, William FitzHugh et al. 'Initial sequencing and analysis of the human genome'. *Nature* 409.6822 (2001), pp. 860–921.

[4] Erez Lieberman-Aiden, Nynke L van Berkum, Louise Williams, Maxim Imakaev, Tobias Ragoczy, Agnes Telling, Ido Amit, Bryan R Lajoie, Peter J Sabo, Michael O Dorschner et al. 'Comprehensive mapping of long-range interactions reveals folding principles of the human genome'. *Science* 326.5950 (2009), pp. 289–293.

[5] Jesse R Dixon, Siddarth Selvaraj, Feng Yue, Audrey Kim, Yan Li, Yin Shen, Ming Hu, Jun S Liu and Bing Ren. 'Topological domains in mammalian genomes identified by analysis of chromatin interactions'. *Nature* 485.7398 (2012), pp. 376–380.

[6] Geir K Sandve, Sveinung Gundersen, Morten Johansen, Ingrid K Glad, Krishanthi Gunathasan, Lars Holden, Marit Holden, Knut Liestøl, Ståle Nygård, Vegard Nygaard et al. 'The Genomic HyperBrowser: an analysis web server for genome-scale data'. *Nucleic Acids Research* 41.W1 (2013), W133–W141.

[7] Lawrence Hunter. 'Molecular biology for computer scientists'. *Artificial Intelligence and Molecular Biology* (1993), pp. 1–46.

[8] Johan H Gibcus and Job Dekker. 'The hierarchy of the 3D genome'. *Molecular Cell* 49.5 (2013), pp. 773–782.

[9] Job Dekker, Karsten Rippe, Martijn Dekker and Nancy Kleckner. 'Capturing chromosome conformation'. *Science* 295.5558 (2002), pp. 1306–1311.

[10] Wendy A Bickmore and Bas van Steensel. 'Genome architecture: domain organization of interphase chromosomes'. *Cell* 152.6 (2013), pp. 1270–1284.

[11] Jennifer E Phillips and Victor G Corces. 'CTCF: master weaver of the genome'. *Cell* 137.7 (2009), pp. 1194–1211.

[12] Elphège P Nora, Bryan R Lajoie, Edda G Schulz, Luca Giorgetti, Ikuhiro Okamoto, Nicolas Servant, Tristan Piolot, Nynke L van Berkum, Johannes Meisig, John Sedat et al. 'Spatial partitioning of the regulatory landscape of the X-inactivation centre'. *Nature* 485.7398 (2012), pp. 381–385.

[13] Darya Filippova, Rob Patro, Geet Duggal and Carl Kingsford. 'Identification of alternative topological domains in chromatin'. *Algorithms for Molecular Biology* 9.1 (2014), p. 14.

[14] Suhas S.P. Rao et al. 'A 3D map of the human genome at kilobase resolution reveals principles of chromatin looping'. *Cell* 159.7 (2014), pp. 1665–1680.

[15] Natalia Naumova, Maxim Imakaev, Geoffrey Fudenberg, Ye Zhan, Bryan R Lajoie, Leonid A Mirny and Job Dekker. 'Organization of the mitotic chromosome'. *Science* 342.6161 (2013), pp. 948–953.

[16] Takeshi Mizuguchi, Geoffrey Fudenberg, Sameet Mehta, Jon-Matthew Belton, Nitika Taneja, Hernan Diego Folco, Peter FitzGerald, Job Dekker, Leonid Mirny, Jemima Barrowman et al. 'Cohesin-dependent globules and heterochromatin shape 3D genome architecture in S. pombe'. *Nature* 516.7531 (2014), pp. 432–435.

[17] Tom Sexton, Eitan Yaffe, Ephraim Kenigsberg, Frédéric Bantignies, Benjamin Leblanc, Michael Hoichman, Hugues Parrinello, Amos Tanay and Giacomo Cavalli. 'Three-dimensional folding and functional organization principles of the Drosophila genome'. *Cell* 148.3 (2012), pp. 458–472.

[18] Byung-Jun Yoon. 'Hidden Markov models and their applications in biological sequence analysis'. *Current Genomics* 10.6 (2009), pp. 402–415.

[19] F. Pedregosa et al. 'Scikit-learn: Machine Learning in Python'. *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[20] Jeremy Goecks, Anton Nekrutenko, James Taylor et al. 'Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences'. *Genome Biology* 11.8 (2010), R86.

[21] Daniel Blankenberg, Gregory Von Kuster, Nathaniel Coraor, Guruprasad Ananda, Ross Lazarus, Mary Mangan, Anton Nekrutenko and James Taylor. 'Galaxy: a web-based genome analysis tool for experimentalists'. *Current Protocols in Molecular Biology* (2010), pp. 1–21.

[22] Belinda Giardine, Cathy Riemer, Ross C Hardison, Richard Burhans, Laura Elnitski, Prachi Shah, Yi Zhang, Daniel Blankenberg, Istvan Albert, James Taylor et al. 'Galaxy: a platform for interactive large-scale genome analysis'. *Genome Research* 15.10 (2005), pp. 1451–1455.

[23] Bryan R Lajoie, Job Dekker and Noam Kaplan. 'The hitchhiker's guide to Hi-C analysis: practical guidelines'. *Methods* (2014).

[24] Yuanyuan Li, Weichun Huang, Liang Niu, David M Umbach, Shay Covo and Leping Li. 'Characterization of constitutive CTCF/cohesin loci: a possible role in establishing topological domains in mammalian genomes'. *BMC Genomics* 14.1 (2013), p. 553.

[25] W James Kent, Charles W Sugnet, Terrence S Furey, Krishna M Roskin, Tom H Pringle, Alan M Zahler and David Haussler. 'The human genome browser at UCSC'. *Genome Research* 12.6 (2002), pp. 996–1006.

[26] Tae Hoon Kim, Ziedulla K Abdullaev, Andrew D Smith, Keith A Ching, Dmitri I Loukinov, Roland D Green, Michael Q Zhang, Victor V Lobanenkov and Bing Ren. 'Analysis of the vertebrate insulator protein CTCF-binding sites in the human genome'. *Cell* 128.6 (2007), pp. 1231–1245.

[27] ENCODE Project Consortium. 'An integrated encyclopedia of DNA elements in the human genome'. *Nature* 489.7414 (2012), pp. 57–74.

[28] Lin Liu, Yiqian Zhang, Jianxing Feng, Ning Zheng, Junfeng Yin and Yong Zhang. 'GeSICA: Genome segmentation from intra-chromosomal associations'. *BMC Genomics* 13.1 (2012), p. 164.

[29] Jason Ernst and Manolis Kellis. 'Discovery and characterization of chromatin states for systematic annotation of the human genome'. *Nature Biotechnology* 28.8 (2010), pp. 817–825.

[30] Christopher M Bishop et al. *Pattern recognition and machine learning*. Vol. 1. New York: Springer, 2006. Chap. 13.2.

[31] Lawrence Rabiner. 'A tutorial on hidden Markov models and selected applications in speech recognition'. *Proceedings of the IEEE* 77.2 (1989), pp. 257–286.

[32] Leonard E Baum and George R Sell. 'Growth transformations for functions on manifolds'. *Pacific Journal of Mathematics* 27.2 (1968), pp. 211–227.

[33] Scott Schwartz, W James Kent, Arian Smit, Zheng Zhang, Robert Baertsch, Ross C Hardison, David Haussler and Webb Miller. 'Human–mouse alignments with BLASTZ'. *Genome Research* 13.1 (2003), pp. 103–107.

[34] Walther Flemming. 'Contributions to the knowledge of the cell and its vital processes'. *The Journal of Cell Biology* 25.1 (1965), pp. 3–69.

[35] Eitan Yaffe and Amos Tanay. 'Probabilistic modeling of Hi-C contact maps eliminates systematic biases to characterize global chromosomal architecture'. *Nature Genetics* 43.11 (2011), pp. 1059–1065.

[36] James D Watson and Francis HC Crick. 'Molecular structure of nucleic acids'. *Nature* 171.4356 (1953), pp. 737–738.

Appendices

Appendix A

Hidden Markov Models: details and the mathematics behind

A hidden Markov model (HMM) is a model describing a process that goes through a set of states at discrete time steps.

The following notation is used:

- A HMM has N states, which we will denote $s_1, s_2, \ldots, s_N$.
- The model goes through discrete time steps, t_0, t_1, t_2, ..., t_T. At each time step, the model is in one of the possible states.
- Between each time step, the model may change state with a given probability. The probability depends on the current state only. We will denote the probability of going from state i to state j as a_{ij}, which for $1 \le i, j \le N$ can be represented in the matrix A.
- The current state at time step t is q_t. The probability of being in state s_i at time t only depends on the state at time $t - 1$.
- At time step t, we make the observation o_t. We will denote the set of all observations made as O. The notation $b_i(k) = P(o_t = k|q_t = s_i)$ will be used to denote the probability that an observation o_t is k when the process is in state s_i. B will be used to denote the set of all these probabilities. Note that this probability is independent of the time t.
- π_i is the probability of beginning the whole process at state s_i .
- We will denote the set of all the parameters as $\lambda = (A, B, \pi)$.

There are three main questions to answer:

1. Given all the model parameters λ, what is the probability of a given sequence of observations occurring?
2. Given the observations and the model parameters λ, what is the most probable sequence of states that has produced these observations?

3. Given *none* of the model parameters λ, only the observations O, what are the most probable model parameters and the most probable sequence of states?

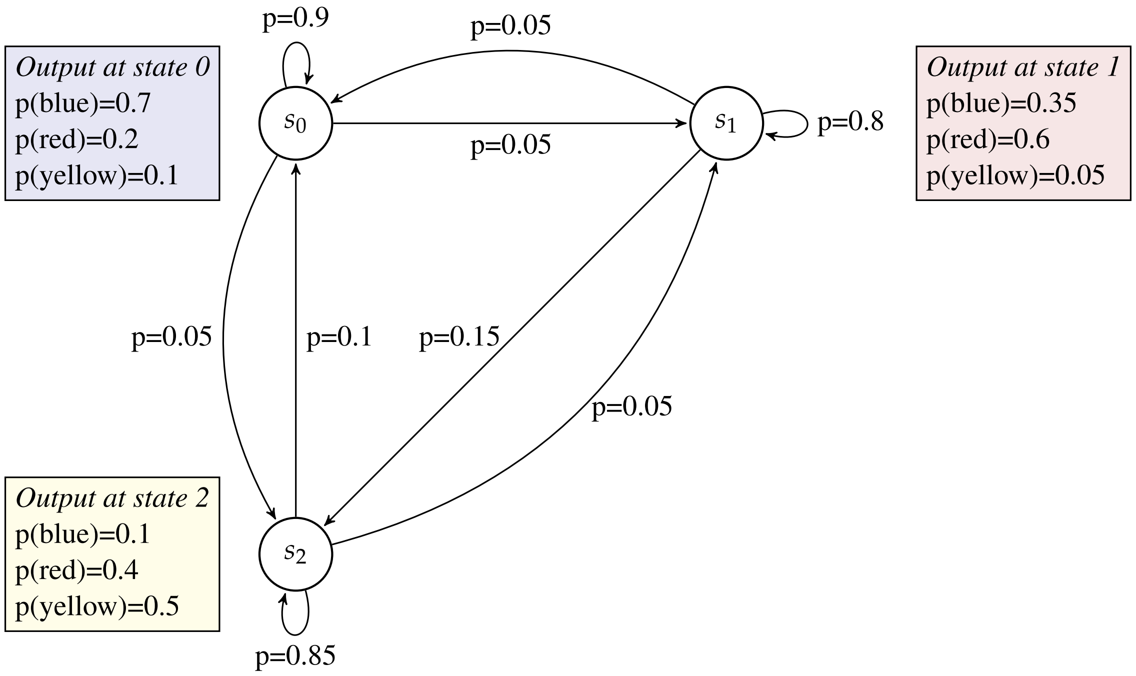


Figure A.1: An illustration of a hidden Markov model with three states, each state producing a colour. At each time step, there is a probability of changing state or staying at the same state. At each time step, an "output", which we can observe, is produced. In this case, we observe one of the colors blue, red or green.

Problem 1 The first problem concerns finding the probability that a series of observations, $o_1, o_2, \ldots, o_T$, occurs. Remember that the true sequence of states is unknown, so one approach we can use to find this probability is to look at all the different possible state sequences that produce the set of observations. Since the probabilities of different state sequences occurring are independent, we can sum all these probabilities:

$$P(o_1, o_2, ..., o_T|\lambda) = \sum_{Q \in \text{all possible state sequences}} p(o_1, o_2, ..., o_T|Q, \lambda) \tag{A.1}$$

As T grows, the number of possible paths (N^T) grows exponentially, making the computations impossible for big T. A more efficient method is obviously necessary to solve this problem. To find such a method, the following notation is useful, given in Rabiner's tutorial on HMMs [31]. Let us denote $\alpha_t(i)$ as the probability of having made the observations $o_1, o_2, ..., o_t$ (a set of observations up to time step t) and ending up in state i at time t.

$$\alpha_t(i) = P(o_1, o_2, ..., o_t \wedge q_t = s_i|\lambda) \tag{A.2}$$

α can easily be found for $t = 1$:

$$\alpha_1(i) = \pi_i \cdot P(o_1|s_1 = i) \tag{A.3}$$

We can now define $\alpha_2(j)$ by using $\alpha_1(i)$. All we need to do is to sum over the possible states when $t = 1$ and multiplying by the probability of changing to state j:

$$\alpha_2(i) = \sum_{j=1}^{N} \alpha_1(j) \cdot a_{ji} \cdot p(o_2|q_2 = s_i) \tag{A.4}$$

By now, we see that α can be defined recursively, because to be in state j after time t, the process needs to be in some state i at the previous step, and go to state j with probability a_{ij}. Being in state i at the previous step is $\alpha_{t-1}(i)$ and the probability of going from state i to j is a_{ij}. Thus, the recursive formula is:

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) \cdot a_{ij} \cdot p(o_t|q_t = s_j) \tag{A.5}$$

With α defined, we can now proceed and find a more efficient solution to the first question. Remember that, at time step t, we want to find the probability that a given set of observations has occurred. We also know that the probabilities of being in some state at time step t are all independent of each other. Thus, the probability of having made the observations $o_1, o_2, ..., o_t$ is simply a sum over α_t for all possible values of i (all different states we can be in at time step t):

$$P(o_1, o_2, ..., o_t|\lambda) = \sum_{i=1}^{N} \alpha_t(i) \tag{A.6}$$

Problem 2 This problem requires us to define an optimality criterion. A simple criterion is that each state in the sequence has to be the one with the highest probability of occurring given the observation, but we will see that this criterion does not make perfect sense.

First, let us define function β, in a similar way as α, as the probability of the observations $o_{t+1}, ...o_T$ occurring and being in state i at time t.

$$\beta_t(i) = P(q_t = i \wedge o_{t+1}, ..., o_T|\lambda) \tag{A.7}$$

Let us also define $\gamma_t(i)$ as the probability of being in state s_i at time step $t \leq T$ given a set of observations $o_1, o_2, ..., o_T$ and the model parameters:

$$\gamma_t(i) = P(q_t = i|o_1, o_2, ..., o_T, \lambda) \tag{A.8}$$

We can express γ in terms of α and β:

$$\gamma_t(i) = P(\text{Being in state } s_i \text{ at time } t \text{ given } o_1, o_2, ..., o_T)$$

$$= \frac{P(\text{Being in state } s_i \text{ at time } t \text{ and having observed } o_1, o_2, ..., o_t, o_{t+1}, ..., o_T)}{P(\text{Having observed } o_1, o_2, ..., o_T)}$$

$$= \frac{\alpha_t(i)\beta_t(i)}{P(o_1, o_2, ..., o_T)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{A.9}$$

Thus, we can find the most probable state at time t by finding the i that gives the highest $\gamma_t(i)$. The most probable path can be found by doing this for every t.

However, this method makes the mistake of forgetting that we are dealing with a sequence of states. Finding the most optimal state at each time step individually does not make sense, since there is a probability of going from one state to another at a given time step. A better criterion is to find the single best state sequence Q by maximizing $P(Q|o_1, o_2, ..., o_T, \lambda)$. A method that does this is the Viterbi algorithm.

To explain the Viterbi algorithm, we need one more definition:

$$\delta_t(i) = \max_{q_1, q_2, ..., q_t} P(q_1, q_2, ..., q_t = i \wedge o_1, o_2, ..., o_t|\lambda) \tag{A.10}$$

$\delta_t(i)$ is the state sequence $q_1, q_2, ...$ that ends in state $q_t = i$ that has the highest probability of occurring when taking a series of observations into account. In the same way as when defining α, it is easy to find $\delta_1(i)$:

$$\delta_1(i) = \max \pi_i P(o_1|q_1 = i) \tag{A.11}$$

By recursion, we find

$$\delta_t(i) = \max_j \left[\delta_{t-1}(j)a_{ij}\right] P(q_t = i|o_t, \lambda) \tag{A.12}$$

We now have a formula that gives us the probability of the most likely path up to some time step t. The formula is recursive, i.e. to calculate for some t, we need the calculations done at time $t-1$. The Viterbi algorithm simply starts at $t = 1$, finds the state i that maximizes the expression, then continues to $t = 2, 3, ...$. At each step, it keeps track of the states found. At termination, when $t = T$, it has the most optimal path. More precisely:

1. Init: Find i that maximizes $\delta_1(i)$. Store the i as q_1.
2. For $t = 2, ..., T$, find the i that maximizes $\delta_t(i)$. Store the i as q_t.
3. When $t = T$ the algorithm terminates, and the most probable state sequence is stored in $q_1, q_2, ..., q_T$.

This solves problem 2.

Problem 3 This is the most difficult problem to solve. Now, none of the parameters of the HMM are known, all we know is $o_1, o_2, ..., o_T$, and we want to find the parameters that make the set of observations most probable. I.e., we want to find $\lambda = (A, B, \pi)$ that maximizes $P(O|\lambda)$.

There is no exact analytical solution to this problem, only numerical solutions, and they tend to get stuck at local optima [31]. Thus, finding a method that converges to a local optima is our best chance. Several methods have been developed, some using gradient techniques to iteratively find a local optimum. The Baum-Welch method is an iterative algorithm that uses the Expectation−Maximization (EM) algorithm to iteratively find a local optimum. This method will be described here.

First, we define $\xi_t(i,j)$, which is the probability of being in state s_i at time t and state s_j at time $t+1$:

$$\xi_t(i,j) = P(q_t = s_i \wedge q_{t+1} = s_j | O, \lambda) \tag{A.13}$$

This probability can also be expressed in terms of the α and β functions defined earlier. Recall that $\alpha_t(i)$ gave us the probability of having a series of observations up to time t and then ending in state s_i at time t, while $\beta_t(i)$ gave us the probability of starting in state s_i at time t, then having a series of observations. Thus $\alpha_t(i)a_{ij}\beta_{t+1}(i)b_j(o_{t+1})$ will be the probability of having a set of observations $o_1, \ldots, o_t$, being at state s_i at time t, going to state s_j at time $t+1$ and then having a set of observations the rest of the time left. If we divide by the probability of having the whole set of observations, we get the probability of only being in state s_i at time t and being in state s_j at time $t+1$:

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}\beta_{t+1}(i)b_j(o_{t+1})}{p(O,\alpha)} \tag{A.14}$$

Remember that we defined $\gamma_t(i)$ as the probability of being in state s_i at time t. We can now define $\gamma_t(i)$ in terms of $\xi_t(i,j)$ by summing over all the possible cases where we are in state s_i at time t and then go to state s_j:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{A.15}$$

This might seem to be an unnecessary and complicated definition, but it will turn out to be useful. What we would like to do is to estimate the probability of going from state s_i to state s_j for all possible i and j (these are the a_{ij}s). The definition of ξ is useful for this, all we really need to do is to sum all values of t to get an estimate of the number of times we go from state s_i to s_j. If we divide by the expected total number of times we are in state s_i, we get the probability of going from s_i to s_j. This can simply be found by summing $\gamma_t(i)$ for all values of t. Thus an estimation of a_{ij} is:

$$\bar{a_{ij}} = \frac{\text{expected number of transitions from } s_i \text{ to } s_j}{\text{expected number of times in state } s_i} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{A.16}$$

We would also like to know the expected number of times we are in state s_i for any given i, as this will be an estimation of π_i. This is simply $\gamma_1(i)$, so:

$$\bar{\pi}_i = \gamma_1(i) \tag{A.17}$$

In a similar way, we find an estimation for $b_i(k)$:

$$b_i\bar{(k)} = \frac{\text{expected number of times in state } s_i \text{ when observing the observation } k}{\text{expected number of times in state } s_i}$$

$$= \frac{\sum_{t\in\{1,\dots,T|o_t=k]\}} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} \tag{A.18}$$

We now have a formula for estimating the parameters, but they all require that we already know the parameters, which seems very counter-intuitive. However, Baum et al. [32] showed that when guessing some parameters $\lambda = (A, B, \pi)$ and using them to estimate new parameters $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$ by using the above formulas, the new set of estimated parameters will either give the same or a higher value of $P(O|\lambda)$. Thus, by doing this iteratively, we will converge to an estimated $\bar{\lambda}$ that gives a local optimum of $P(O|\lambda)$. The proof for this is beyond the scope of this text.

Appendix B

Source code and analysis details

B.1 Source code

All the scripts used in this thesis and the source code for the tools created in the Genomic HyperBrowser can be downloaded from the galaxy page found at https://hyperbrowser.uio.no/3dml/u/ivar/p/master. This galaxy page also contains our domain sets and the history elements for most of the analyses that have been performed.

The following is an overview of the main files in the source code:

- *FindTads.py* is the main file for the *Find TADs* tool.
- *VisualizeTads.py* is the main file for the *Visualize TADs* tool.
- *AnalyzeTads.py* is the main file for the *Analyse TADs* tool.
- *FindTadsInChr.py* contains the class *FindTadsInChr* that can be used to load Hi-C data, find TADs, visualize TADs and save TADs to a bed-file. This class is used by the *Find TADs* tool.
- *html_visualization.py* contains methods for printing the html that shows the plots in the *Visualize TADs* tool.
- *Analysis.py* contains the class *Analysis* that performs most of the TAD analyses.
- *ctcf_analysis.py* is the script used for performing the CTCF analysis.
- *generate_domain_borders.py* is a small script that generates domain borders as segments from the beginning of domains.
- *domain_consistency.py* performs the consistency analysis (Section 4.4.7).
- *simulate_data_matrix.py* simulates the Hi-C data matrices shown in Section 3.3.6.

B.2 Algorithm for choosing a consensus set with optimal average domain score

In Section 3.4.1 we select the non-overlapping set of domains from the hierarchical set that has the maximum average domain score among all sets of non-overlapping domains[1].

The following algorithm finds this set:

1. Begin the removal process at the second deepest level that contains domains.
2. For every domain on this level: compute the average score for all non-deleted subdomains.
   (a) If this average score is higher than the domain score: delete the domain.
   (b) If not: delete all the subdomains and keep the domain.
3. Carry out step 2 and 3 again at the next (less deep) level until there are no more levels.

Proof. It can be shown through induction that the process above removes domains so that the remaining domains are non-overlapping and have the optimal average score among all sets of non-overlapping domains. First, we show that the kept domains at the highest and second highest levels are the optimal set after the first iteration of the algorithm. This is true since there are only two possible choices for every domain at the second highest level — either the domain is kept or its subdomains are kept. After the choice has been made, the average domain score is increased among the domains at the two highest levels. Also, no domains in the two highest levels will overlap, since the only possibility for overlap is between domains on the second highest level and their respective subdomains. The next step of the proof is to show that if the criterion holds at a given level n, then it must hold for one level less deep, $n-1$, after step 2 has been run on this level. After the process at level $n-1$, no domains at that level will overlap with domains at higher levels, since the domain can only overlap with subdomains, and either all subdomains of the domain are removed or the domain itself is removed. Also, the average domain score will increase as much as possible after the process, since if subdomains of a domain at the level have a lower average score than the domain itself, they will be removed in favour of the domain (increasing the score as much as possible). If not, the domain will not be included (the score stays the same). It follows from induction that the set of maximum average score will be obtained after running step 2 on all levels, from second highest to lowest.

[1]An implied criterion is that as many domains as possible are chosen for a set to be valid, i.e. a set consisting of one domain is not valid, since more domains can be included.

B.3 Liftover analysis

To create domain borders, the following script (*generate_domain_borders.py*) was run. The script writes domain borders to a bed-file, by taking the beginning position of domains from a bed file containing domains.

```python
for filename in ['somefile']:
    f = open(filename + '.bed')
    f2 = open(filename + '_borders.bed', 'w')
    for line in f.readlines():
        d = line.split()
        end = str(int(d[1]) + 40000)
        f2.write(d[0] + " " + d[1] + " " + end + "\n")

    f.close()
    f2.close()
```

These bed files containing border segments were then lifted using the UCSC liftover tool[2] [25]. Genomes *mm9* and *hg18* were used and *minimum ratio of bases that must remap* was set to 0.1.

The resulting files were all uploaded to the Genomic HyperBrowser, and can be found in the history *Liftover analysis* on the galaxy page for this thesis.

B.4 Hi-C data used

The Hi-C data used in this project are from Ren Lab. The data are available for download from the websites of Ren Lab: http://chromosome.sdsc.edu/mouse/hi-c/download.html, and are the same data used by Dixon et al. and Filippova et al.

We used the normalized data matrices, which have gone through a bias correction process described by Yaffe et al. (2011) [35]. We chose to remove the diagonal, i.e. set all numbers on the diagonal to zero, since we believe that these numbers have high variance and mostly represent one-dimensional interactions as a result of the spatial closeness of loci inside the same bin.

B.4.1 Normalizing Hi-C data

We created a version of the Hi-C data where interactions in the Hi-C data matrix were normalized by the distance from the diagonal. In this normalized interaction matrix, the mean interaction frequency between pairs of bins with the same distance from the diagonal is 0.

The normalization was performed by subtracting μ_d from every interaction and then dividing by s_d, where μ_d is the mean and s_d is the standard deviation of the

[2]https://genome.ucsc.edu/cgi-bin/hgLiftOver

interaction frequency between pairs of bins with distance d, i.e. bins that are d bins away from each other on the genomic sequence:

$$\bar{A_{i,j}} = \frac{A_{i,j} - \mu_{j-i+1}}{s_{j-i+1}} \tag{B.1}$$

As discussed in Section 3.3.7, we do not include rows/columns that sum to zero, since these most likely represent missing data. The following Python function takes an interaction matrix and normalizes it according to the formula above:

```python
def normalize_data_matrix(data_matrix):

    d = data_matrix.copy()
    M = len(d)

    # Compute the average and standard error of
    # interactions between pairs of same distance
    avgs = np.zeros(M)
    stds = np.zeros(M)

    # Do not include colums/rows in the matrix that
    # sum to 0 when computing the mean/standard error
    # Matrix is symmetrical around
    # diagonal, so row_i = col_i
    nz = np.sum(d, 0) > 0 # will contain 1 where
                          # columns/rows are non-zero

    for i in range(1, M):
        sub_diagonal = np.diag(d, i)

        # Remove the elements from the
        # columns that are zero
        # (A zero row hits at the row number)
        # (A zero column hits at the column
        # number minus the diagonal offset)
        nz_column = nz[i:]
        nz_row = nz[0:len(sub_diagonal)]
        # nz_tot contains 1 for bins that represent
        # non-zero cols/rows
        nz_tot = nz_column + nz_row

        avgs[i] = np.mean(sub_diagonal[nz_tot])
        stds[i] = np.std(sub_diagonal[nz_tot]) \
                / np.sqrt(len(sub_diagonal[nz_tot]))

    # Normalize data based on these avgs and stds
    for i in range(0, M):
        for j in range(0, M):
            if j != i:

                # Hack if std is zero (shouldnt happen)
                if stds[abs(i-j)] < 10e-10:
                  d[i, j] = (d[i, j] - avgs[abs(i-j)])
            else:
              d[i, j] = (d[i, j] - avgs[abs(i-j)])\
                      / stds[abs(i-j)]

    # Round to four decimals to save RAM/disk space,
    # this is the precision of the original data
    return np.round(d, 4)
```

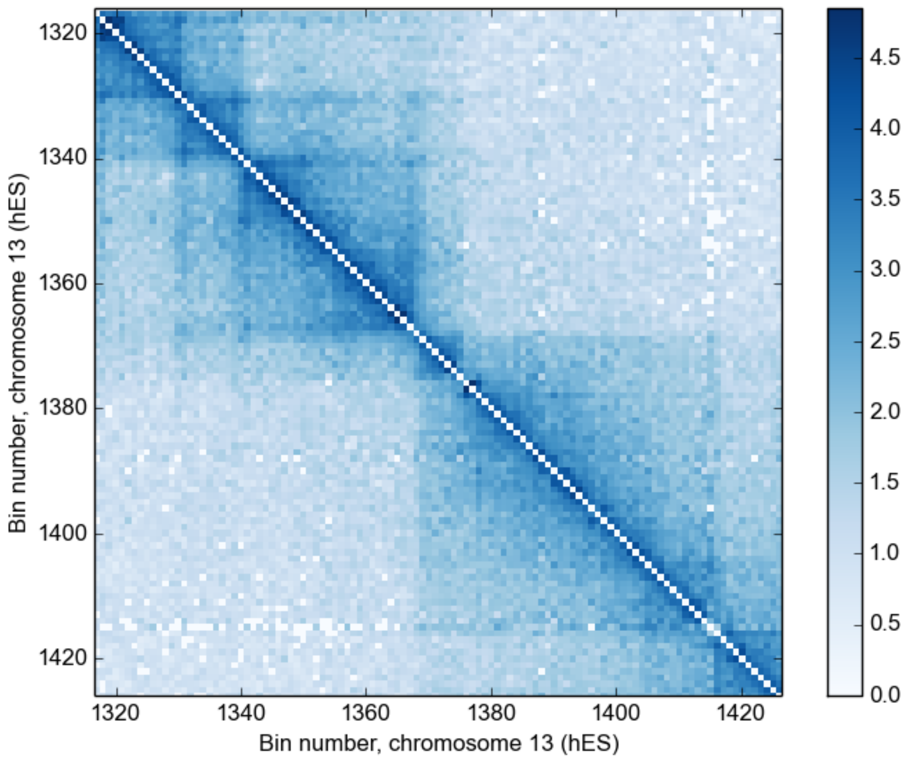
B.5 Normalizing for selection criterion — globally vs. locally

An important part of the selection criterion discussed in Section 3.3.6 is how interactions between two pairs of bins with different *linear distance*[3] are compared. Most areas will interact more with themselves than with neighbouring areas, simply because bins naturally interact more with bins that are closer on the linear genome. Thus, we reasoned that some kind of normalization should be done. What we want to know is whether a given domain interacts more with itself than with its brother domain, and whether this is only because the bins in the domain are closer to each on the linear genome than to the bins in the brother domain. To do this, it seems reasonable to measure the sum of intra-domain and inter-domain from the normalized interaction matrix (Section B.4.1). This would be a *global normalization*, since data across the whole chromosome (or possible genome) are used.

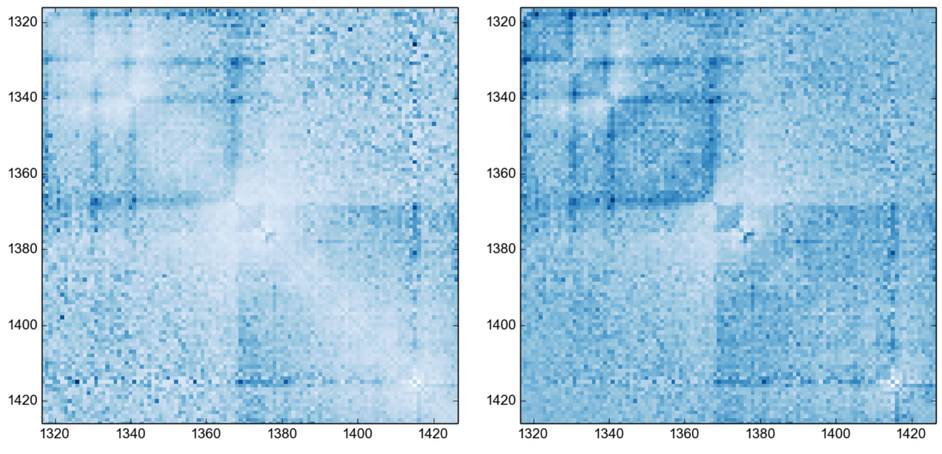
We performed some tests of the selection criterion using this normalized data matrix when computing the sum of intra- and inter-domain interactions. We will use an example from the real data matrix to illustrate one of the problems that occurred when using global normalization. Figure B.1 shows the data matrix of a mother domain selected by the method. One would expect the area from bin 1370 to bin 1415 (or to the end of the matrix) to be a domain that will be selected, since that area seems to interact more with itself than with the other part of the mother domain. However, when normalizing interactions globally, this area has higher average interaction frequency to the other part of the mother domain than it has with itself, and is not selected as a domain. Note from the globally normalized matrix in Figure B.1 (b) that interactions close to the diagonal are weak. This is probably because many small and dense domains are found as dense submatrices close to the diagonal globally. Since this domain does not contain such small and dense domains, the globally normalized intra-domain interaction frequency gets lower, even though lacking such subdomains has nothing to do with the recognition of the domain as an independent domain.

Another problem is that the global average interaction frequency decreases approximately like a negative exponential function of the distance from the diagonal of the data matrix, whereas the average interaction frequency within a domain does

[3]The distance between the bins positions on the genomic sequence



(a)



(b) *(c)*

Figure B.1: Heat map of a part of the data matrix: Raw matrix (a), globally normalized matrix (b) and locally normalized matrix (c).

not decrease as quickly[4]. The negative exponential nature of the global average is a result of averaging over many domains, some dense and some less dense. If

[4]Visual inspection of the data for domains shows that the interaction frequency does not decrease much towards the end of domains. This is also seen in the globally normalized matrix in Figure B.1 (b), where the outer part of the domains have higher interactions than elsewhere.

a domain has a homogeneous set of interactions with itself, then lowering these interactions would *exponentially* increase the difference to what is globally expected. Thus, when lowering all interactions in an area, the new inter-domain interactions would deviate less from what is globally expected than intra-domain, since these are closer to the diagonal. The result can be that in a less dense area of the genome, intra-domain interactions are given too small weight when compared to what is expected globally.

All this indicate that some other kind of normalization should be done when comparing intra-domain interactions with inter-domain interactions in the selection criterion. Another approach is to assume nothing about what is expected globally — instead we only look at the local interactions. Given a mother domain, we want to see if one part of it interacts more with itself than with the other part, only based on the known interactions within the mother domain, i.e. doing *local normalization*. With this approach, every interaction is measured according to all other interactions between pairs of bins having the same distance. The important effect is that interactions far away from the diagonal are given the same weight as interactions close to the diagonal.

We conclude that local normalization is better than global normalization when evaluating whether a domain interacts more with itself than with its brother domain, taking spatial closeness on the genomic line into account.