UNIVERSITÀ
DI TRENTO
Department of
Cellular, Computational and Integrative Biology - CIBIO

Master's Degree Thesis in
Quantitative and Computational Biology

IDENTIFICATION OF CHROMATIN ORGANIZATION BACKBONE THROUGH NETWORK SPARSIFICATION

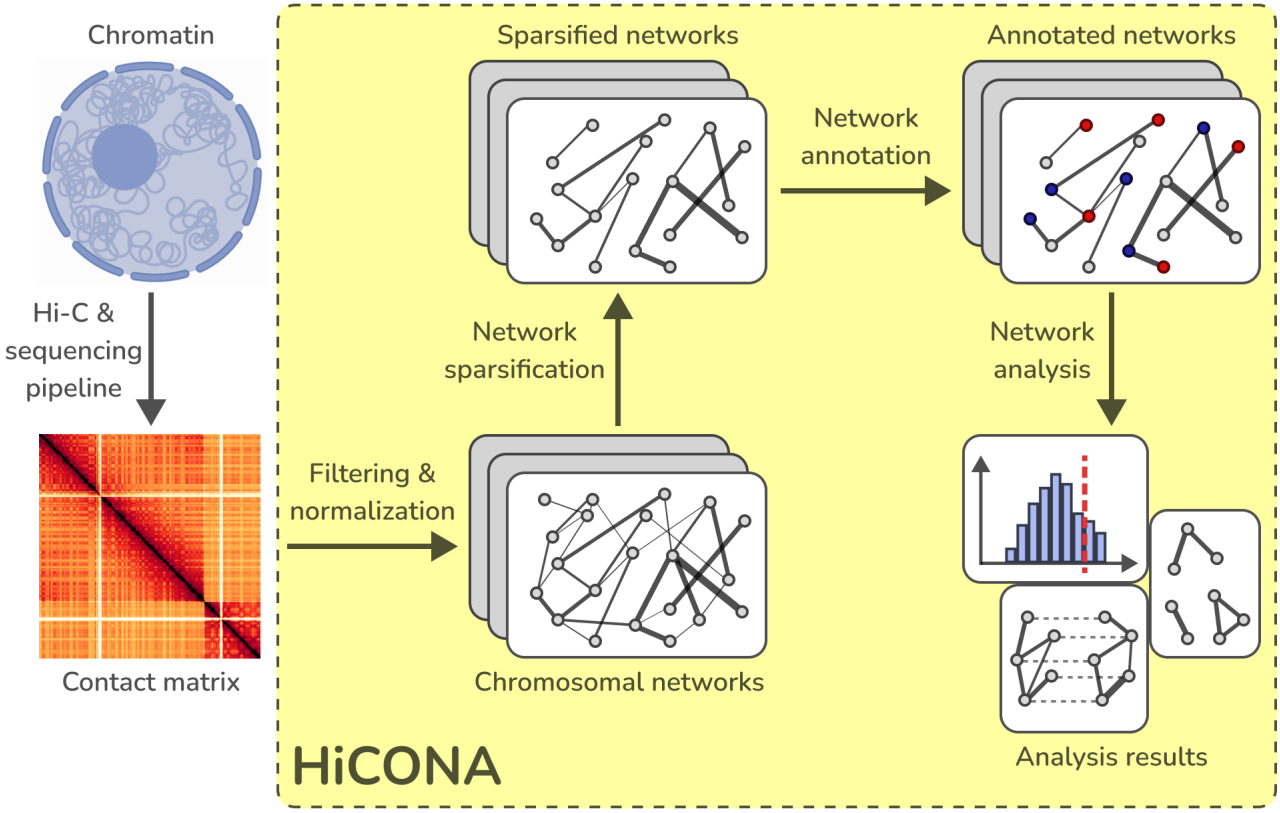
Supervisor:	Co-supervisor:	Graduand:
Alessio Zippo	Leonardo Morelli	Stefano Cretti

Date of discussion: 18/10/2023

Academic year: 2022/2023

Contents

Abstract



Elements from BioRender and cooltools documentation were used to create this image.

The usage of network analysis to study chromatin organization is still at a nascent stage; while there exist some network analysis-based tools or procedures which use network analysis algorithms, their applications are generally restricted to a very narrow set of capabilities and can hardly ever be used for purposes different from the main one for which they were designed. One of the main reasons why network analysis in this field is so limited is the fact that networks representing chromatin organization are extremely dense and thus hard to analyze and handle. Still, network analysis could provide major insights into chromatin organization; it is thus of great importance to develop the means necessary to easily conduct this type of analysis. For this reason, during my internship, I took part in the design and implementation process of HiCONA, a Python3 package aimed at providing a flexible framework in order to conduct any type of network analysis on data coming from Hi-C experiments that capture chromosome conformations. The package provides functionalities to preprocess Hi-C data, which entails filtering, normalization and network sparsification; this last step is of particular relevance, since it allows to handle the network density problem. The preprocessed data is then used to create graphs, which can be annotated and analyzed using algorithms from standard network analysis, which were tailored to Hi-C data derived networks. Keeping into account user-friendliness, the package was optimized in order to be able to process almost any Hi-C data file even on less performing systems, such as laptops. The focus of this thesis is about the preprocessing part of the package, how it has been implemented and how it performs, in terms of computational efficiency as well as reproducibility and consistency. Some network analysis operations which can be conducted using the package will also be discussed, though briefly since they are still under development.

Chapter 1: Introduction

With the passing of time, it has become more and more evident that the way DNA is organized in the nucleus is not inconsequential, but rather a fundamental aspect of gene transcription and other genomic functions such as DNA replication and DNA damage repair. The way DNA is compacted into chromatin and folded, for instance, can affect whether genes are accessible to the RNA-polymerase, or whether regulatory sequences are accessible to transcription factors; moreover, chromatin conformation can define which regulatory elements are close enough to their target to exert their action [1;2].

First, how DNA is shaped into chromatin and how this is organized into the nucleus will be discussed. Then, a subset of experimental techniques used to study chromatin conformation, the so-called ligation-based methods, will be described and in particular the in situ Hi-C protocol. The next topic will be the way data coming from these techniques is typically processed and stored. Finally, some graph theory concepts will be provided, since network sparsification, a network analysis algorithm, is the main point of the data preprocessing pipeline proposed in this work.

1.1 Chromatin organization and functions

DNA in its unwound state cannot fit inside the cell nucleus; additionally the way DNA is placed inside the nucleus can be used as a regulatory mechanism to upregulate or downregulate gene expression, by co-localizing or separating actors involved in gene transcription. The compacted DNA, along with the proteins and regulatory RNAs taking part in this process, is called chromatin. Chromatin 3D-organization is rather complex, with different structures which can be identified at different scales [2;3]. Chromatin 3D-organization is summarized in figure 1.1. First, chromatin 3D-organization will be discussed, starting from its most basic element, the nucleosome, up to the highest organization order, the chromosome territories. Then, a more in depth analysis of chromatin functions, and especially of gene transcription regulation, will be performed. Finally, some examples of the role of chromatin organization in pathology will be made.

1.1.1 Chromatin 3D-organization

Starting from the lowest organization level, the elementary structural unit of chromatin is the **nucleosome**, which is composed of the nucleosome core particle and the linker DNA. The nucleosome core particle (NCP) is constituted by a stretch of 147 base pairs of DNA and an octamer. The octamer consists of two copies of each histone: H2A, H2B, H3 and H4. [4]. Each core histone is composed of a globular portion, around which the DNA is wrapped, and an unstructured tail portion, which is 20-40 aminoacids long, exposed to the nucleosol. These tails can undergo covalent modifications, prevalently acetylation, phosphorylation and methylation, which change the properties conferred by the aminoacidic sequence [5]. Linker DNA is a portion of DNA of variable size, usually between 10 and 90 base pairs, which connects two adjacent NCPs; the variably spaced sequence of NCPs and their linker DNA portions is usually referred to as *beads-on-a-string* structure. This structure can be further organized into fibers with different shapes and compaction levels [1].

The chromatin fiber-like structure is then organized into **topologically associated domains** (TADs) and chromatin loops. A TAD is a portion of the genome which forms contacts with itself more frequently than with other regions. Although not fully understood yet, the function of TADs seems to facilitate the interaction of gene regulatory elements with their targets, by creating compartments of the genome containing co-regulated elements, therefore making gene regulation more robust [6]. TADs posses boundary regions which are denoted by the proteins CCCTC-binding factor (CTCF) and cohesin, as well as specific histone modifications [3]. These facts, together with both experimental perturbations and polymer physics simulations, led to the current proposed mechanism that TADs are formed by a competition of loop extrusion, mediated by cohesin, and epigenetically defined compartmentalization, mostly due to CTCF [7]. Within TADs it is thus possible to define **loop domains**,

which are loops in the chromatin structure which promote interactions within TADs, by bringing genes and their regulatory elements into close proximity, while insulating regions across TAD boundaries[3].

At a very broad scale, chromatin is organized into chromosome territories and compartments. Interphasic chromosomes tend to occupy distinct regions of the nucleus, the so-called **chromosome territories**[8]. Chromosome territories usually do not overlap with each other, therefore inter-chromosomal interactions are by far less common than intra-chromosomal ones. Within a chromosome territory, it is possible to distinguish transcriptionally active and inactive regions. Typically, regions interact with others of the same type to form **compartments**. Compartment A is mostly composed of transcriptionally active regions, with high gene density and active histone modifications. On the other hand, compartment B is mostly composed of transcriptionally inactive regions, with lower gene density or gene deserts[3]. Compartment A is frequently found in the interior nuclear space, organized around nuclear speckles, while compartment B is generally found at the periphery of the nucleus, in contact with the nuclear lamina, or close to the nucleolus[2;3].

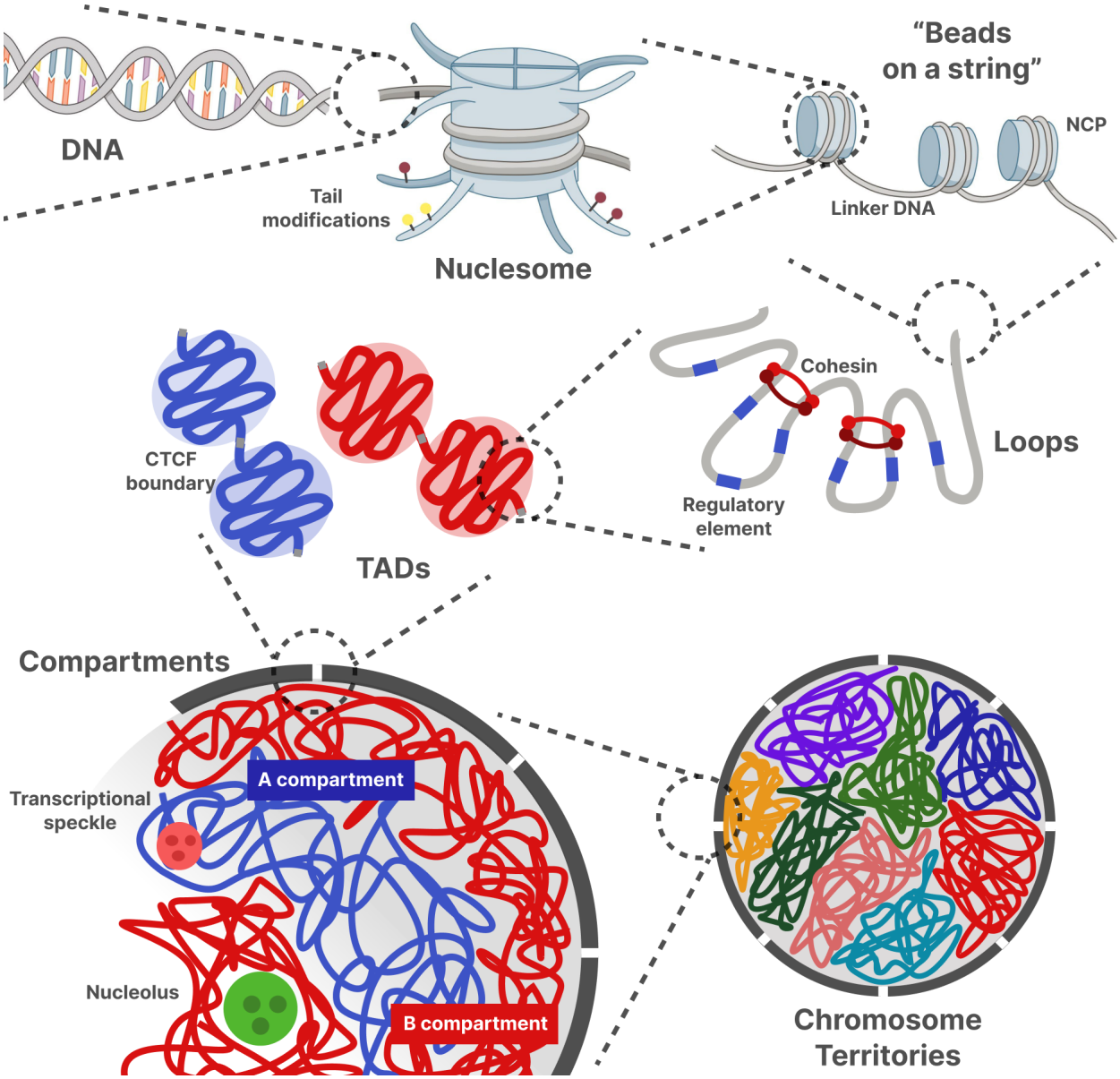


Figure 1.1: Chromatin 3D-organization levels. Double-strand DNA wraps around the octamer in order to obtain the nucleosome core particle (NCP), which together with linker DNA forms the nucleosome. Nucleosomes are placed along chromatin with a variable spacing, defining the so-called "beads-on-a-string" structure. This fiber is then shaped into loops through cohesin complexes; these loops bring into close proximity genes and cis-regulatory elements. The loops are organized into topologically associated domains (TADs) by CTCF, limiting which regions can interact with each other. TADs belong to either chromatin compartment A or B, the former being transcriptionally active, the latter being transcriptionally repressed. Each interphasic chromosome occupies a distinct territory, or area, of the nucleus. *The top row of elements was drawn by Annarita Zanon.*

1.1.2 Chromatin functions

Aside from allowing the genetic material to fit into the nucleus, one of the main functions of chromatin is to regulate gene transcription. Nucleosome positioning, histone tails modifications and many other factors are all fundamental in defining chromatin accessibility by the transcriptional machinery[1;9]. It is in fact possible to define, on a mesoscale level, two **chromatin states**, those being heterochromatin and euchromatin. The former is a more tightly packed state, meaning that genes located within this region are less transcribed since transcriptional machinery binding is impeded; the latter is a more loosely packed state, favoring gene accessibility and transcription[10]. Chromatin states and compartments are not necessarily overlapping, even though compartment A tends to be mostly composed of euchromatin and vice versa. Reflecting the role in gene regulation, the chromatin state of a genomic region is not fixed and it can in fact change overtime, for instance by the action of chromatin remodeling complexes[11]. Moreover, the states are associated with different histone modifications which can be added or removed overtime by many enzymes[5;12].

Furthermore, chromatin regulates transcription through **nuclear condensates**, which are areas of the nucleus with an enrichment of molecules linked to a particular function[13]. Condensate formation seems to be driven by phase-separation, though the mechanism responsible for selecting which molecules will take part in a specific type of condensate is still unclear[14]. Condensates can have different sizes and functions; some examples are the nucleolus (for ribosomal RNA transcription), nuclear speckles (for splicing), Cajal bodies (for spliceosomal RNA maturation) and DNA repair foci[15]. Of particular interest for our discussion are **transcriptional condensates**, which are smaller scale condensates composed of RNA Pol II, transcription factors and other molecules, whose role is to promote the clustering of cis-regulatory elements, favoring enhancer-promoter interactions[16;17]. Transcriptional condensates appear to be involved in defining chromatin structure, while at the same time they seem to depend on CTCF and cohesin for the phase-separation process[18;19]. Condensates are also involved in nongenetic functions, which are still being discovered and studied[13].

Given the importance of chromatin conformation for gene regulation, its role in organism development and cell differentiation has been studied in depth[3], as well as its role in other physiological processes such as X chromosome inactivation[20] and environment response[21]. Then, one must take into account that there exist many other processes which are still affected by chromatin organization, although not directly due to the gene regulation function: for example, DNA replication[22] and repair[23]. Considering this myriad of functions, all of which could be dysregulated leading to pathologies, it becomes evident why improving our understanding of chromatin organization is paramount.

1.1.3 Involvement of chromatin in disease

Considering how complex and highly regulated chromatin conformation is, it is easy to see how its dysregulation can lead to different pathologies. Some examples are reported to show different pathological mechanisms.

The term **chromatinopathies** refers to rare Mendelian diseases caused by haploinsufficiency in some chromatin regulator genes. When the altered gene pertains to transcriptional condensate formation, chromatinopathies manifest mostly with aberrant neurological development and growth abnormalities (either in excess or defect). An example of this type of pathology is Kabuki syndrome[13]. Alterations in chromatin regulator genes, such as regulators of histone post-transcriptional modifications as well as genes involved in chromatin remodeling complexes, are also well known to cause different types of tumor[5;24].

Alterations in genes coding for structural proteins can lead to changes in the nuclear mechanical properties and its ability to withstand stress. Mutations in the LMNA gene, coding for the proteins lamin A and C which are fundamental for chromatin organization, have been shown to lead to cardiomyopathies[25].

Another mechanism which can lead to altered chromatin conformation are **structural variants**, those being defined as DNA sequence differences larger than 50 bp from the reference genome[26]. A common cause of structural variants are inversions and translocations. If an inversion happens in proximity of a TAD boundary, this might disrupt its structure, leading to abnormal gene-enhancer interactions (*enhancer hijacking*); this mechanism has been found in both developmental diseases such as polydactyly and brachydactyly[27] as well as different types of both solid and hematopoietic tumors[28;26].

1.2 Ligation-based methods for the study of chromatin organization

There exist plenty of methods to study chromatin organization, which can be divided into imaging-based, ligation-based and ligation-free methods. Ligation-based methods stem from one original method, chromosome conformation capture[29], which has been modified over time to increase resolution and throughput. From the derived methods, Hi-C was the first which allowed the study of interactions on a genome-wide scale.

1.2.1 Chromosome conformation capture

Chromosome conformation capture (3C), the original ligation-based method, is a technique which allows to study interactions among genomic regions through **chromatin crosslinking** and **proximity ligation**[29]. The main steps of the original protocol are the following:

- formaldehyde fixation of isolated, intact nuclei. Formaldehyde causes a crosslinking reaction which stabilizes protein-protein and protein-DNA interactions; this means that DNA regions will be in proximity to each other thanks to their protein mediated interactions.
- chromatin digestion through a restriction enzyme (EcoRI).
- fragment ligation at very low DNA concentration. In this condition, ligation among DNA fragments crosslinked through proteins is heavily favored, given that they will always be in proximity of each other.
- crosslinking reversion and DNA purification.
- quantitative PCR using locus specific primers. Contact frequency among two loci is given by the ratio between the quantity obtained through quantitative PCR on the sample and that obtained using a control.

It is important to notice that this is a **1 vs 1 technique**, meaning that only one pair of loci can be analyzed at a time; moreover each comparison requires 2 quantitative PCRs (sample and control) and a pair of locus specific primers, thus requiring some prior knowledge of the target. For these reasons the technique is very limited and was modified over time to become 1 vs all (4C[30]), many vs many (5C[31]) and then finally all vs all with Hi-C.

1.2.2 Hi-C protocol

Hi-C was the first ligation-based method to allow a **genome wide** study of chromatin interactions. It differs from the original 3C technique mostly for nucleus lysis using sodium dodecyl sulfate (SDS), the usage of biotin to tag and enrich ligation products and for the usage of **sequencing** for quantification rather than quantitative PCR[32]. An optimization of the original Hi-C protocol is in situ Hi-C, whose main difference is that it does not perform nuclear lysis; this reduces the number of ligation events due to random interaction of DNA fragments in solution, while at the same time reducing the reaction volumes and increasing the number of captured interactions[33]. The standard in-situ Hi-C protocol, represented in figure 1.2 (a), can be summarized as follows:

- DNA crosslinking on intact nuclei
- chromatin digestion through a restriction enzyme
- filling of the sticky ends at the restriction sites with biotinylated nucleotides
- proximity ligation of the now flat and tagged ends
- crosslinking reversion and DNA shearing (to 400 bp size)
- fragment pulldown using streptavidin beads
- elution and pair-ends sequencing

Although in-situ Hi-C is the most common variant of the Hi-C protocol, it is not the only one. Worth of notice is Micro-C, which uses micrococcal nuclease as a restriction enzyme for DNA digestion. This means that all linker DNA is degraded and only interactions among nucleosomes are kept. Given the smaller fragment size, this results in a higher resolution at shorter genomic distances, while long range interactions are poorly captured, making this technique complementary to Hi-C rather than a replacement[34].

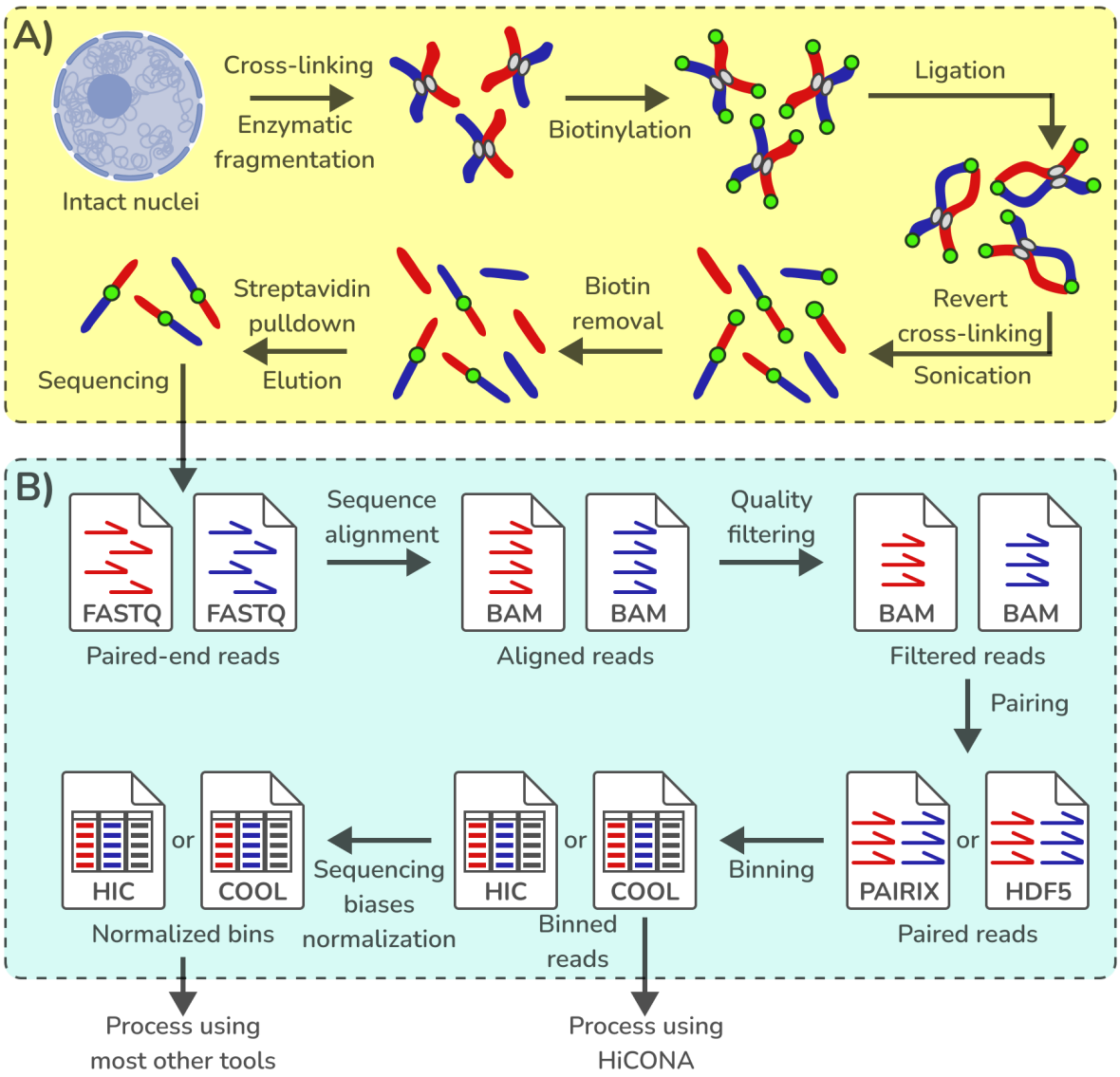


Figure 1.2: Hi-C data generation and analysis workflow. Steps required to generate Hi-C data files using the in situ Hi-C protocol and the basic sequencing data preprocessing. (a) Main steps of an in situ Hi-C experiment. (b) In-silico steps to generate Hi-C data files from raw paired-ends sequencing reads. *Figure created using some elements from BioRender.*

1.3 Hi-C data storage and analysis

Handling Hi-C sequencing data is rather challenging for several reasons, both in terms of storage and analysis itself. This type of data requires different processing steps and file formats with respect to standard genomic DNA sequencing, thus requiring specifically designed tools and algorithms. The following subsections will cover the typical steps used to obtain contact matrices starting from raw sequencing data[35]; these steps are shown in figure 1.2 (b). The most common representations and file formats for Hi-C data are also presented.

1.3.1 Sequence alignment and binning

The sequencing step of a Hi-C experiment generally yields some billions of **paired-end** reads. The fragments from which the reads are derived are chimeric, meaning that they are composed of sequences from regions which are (typically) not genomically contiguous. For this reason it is assumed that the two paired-ends can be mapped independently using standard tools such as bowtie. Still, it is important to notice that, especially for longer reads, the junction among the two interacting genomic regions might fall within one of the paired reads, making the read itself chimeric. **Chimeric reads** are difficult to map and require specific strategies which are implemented by Hi-C data specific tools in order to avoid losing a large amount of information. The aligned

reads are then filtered using both standard DNA sequencing criteria, such as base quality and PCR artifacts, as well as Hi-C specific ones, such as self-ligation products, compatibility with undigested chromatin[36] or more elaborate and *ad hoc* ones[37].

Binning is the next main step performed; the genome is divided into regions, the bins, and each read is substituted with the bin it falls into. The objective of this procedure is to discretize the signal in order to reduce noise, since it is easier to define the contact frequencies among two regions if they have well defined boundaries (even though this comes with a reduction in resolution). Although using bins of variable size is an option, fixed-size bins are the more common choice since they are easier to work with. The chosen bin size will affect the resolution of the final contact matrix. The bigger the bin size, the lower the resolution but the more robust the signal; conversely, the smaller the bin size, the higher the resolution but the noisier the signal. Virtually any bin size could be used, though common sizes are 10 kb, 20 kb and 40 kb. The choice of bin size usually depends on the chromatin structure of interest (5-10 kb for loops, 10-40 kb for TADs, about 200 kb for compartments) as well as economic reasons (the smaller the bin size, the higher the costs). Bins are assigned a progressive numeric ID, from 0 to the number of bins minus 1, to avoid having to refer to them with the entire genomic coordinates they correspond to; sticking to 10 kb resolution, chromosome 1 from base 1 to 10000 would be bin 0, chromosome 1 from base 10001 to 20000 would be bin 1, and so on. After binning, the aligned paired-end reads are thus substituted by pairs of bin IDs; it is then possible to count the number of occurrences (absolute frequency) of each pair, which corresponds to counting how many times the two genomic regions represented by the bins have been found in contact with each other.

Even though Hi-C has a relatively high throughput, a single experiment, even with very high coverage, rarely yields enough reads for robust analysis in the later steps. For this reason, it is very common to aggregate the reads coming from different biological and technical replicates into a single data file.

1.3.2 Contact matrix and ijv table

The aligned, filtered and binned Hi-C data can be represented either using a contact matrix or using an *ijv* table; these formats are illustrated and compared in figure 1.3 (a).

A **contact matrix** is a symmetric and sparse matrix, with all the bins the genome has been divided into as both rows and columns. This means that each cell of the matrix represents how many times the bin corresponding to its row has been found in contact with the bin corresponding to its column. A contact matrix is usually displayed using a heatmap, where each cell is represented by a colored square with intensity proportional to the cell value. For this reason each cell of the matrix is also referred to as a **pixel**. Using a contact matrix it is possible to visualize many chromatin organization levels, as shown in figure 1.3 (b); actually, TADs and compartments were discovered from these heatmaps. Aside from being convenient for visualization purposes, a contact matrix is also convenient to perform row or column-wise operations. For instance, given a certain bin, one can easily obtain the list of interacting bins by looking at the non-zero cells in the corresponding row, or one can compute the number of interactions for that bin by summing the values of all cells in the corresponding row. On the other hand, a contact matrix is not a convenient way to store data. The size of the contact matrix scales quadratically with the number of bins the genome has been divided into; with an intermediate bin size of 10 kb, a contact matrix corresponding to the entire human genome has around 10^{11} cells. For this reason a contact matrix can become hard to handle and store into memory, especially as the resolution increases.

The ***ijv* table** is a format which allows for efficient Hi-C data storage by taking advantage of the properties of the contact matrix. Since the contact matrix is symmetric (the bin contacts are not directional), only the upper triangular part of it (the cells on the main diagonal and those above it) can be stored without losing information. Moreover, since it is a sparse matrix, meaning that most cells have value equal to zero, only the cells with non-zero value can be stored. From these considerations one can define an *ijv* table, also called coordinated list format (COO), as a tabular format where each row contains the two bin IDs corresponding to the coordinates of the cell and the value associated to it, and only cells with a value greater than zero are stored. This representation drastically reduces the number of pixels that need to be stored, and can be converted at any point to the full contact matrix version for visualization. Notice that this format is not well suited for row or column-wise operations, since not all the pixels involving a bin are likely to be contiguous. The pixels are in fact ordered in the same way one would encounter them by going over the upper triangular part of the contact matrix from left to right, from top to bottom; this means that the row bin ID will always be smaller than or equal to the column bin ID, and that the rows are sorted by row bin ID, then by column bin ID.

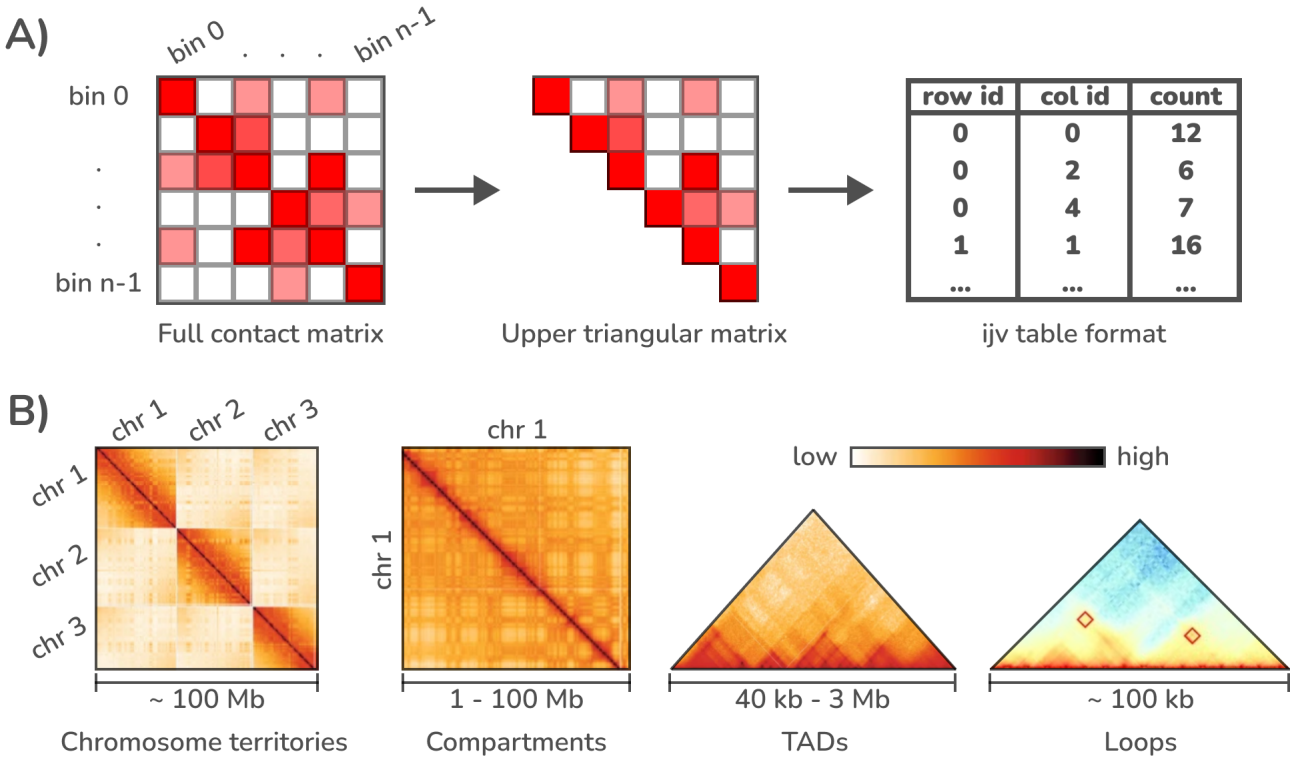


Figure 1.3: Contact matrix and ijv table. (a) Graphical comparison between contact matrix (left) and ijv table (right). (b) Chromatin structures which can be observed through a contact matrix at different scales. From left to right: chromosome territories, compartments, TADs, loops. *Images from Wolff et al., 2022*[38] *and from Kempfer and Pombo, 2019*[2] *were adapted to create the figure.*

1.3.3 HDF5 and cooler file formats

An *ijv* table is a very long one dimensional array, that could theoretically be n^2 rows long, where n is the number of bins the genome has been divided into. Although the full possible size is never reached, a plain text representation of an *ijv* table would be rather limiting in terms of I/O speed. A better choice is to use a compressed, indexed binary file format.

The Hierarchical Data Format version 5 (**HDF5**) is a binary file format maintained by the HDF group[39]. A HDF5 file is organized hierarchically just like a file system, with nested **groups** corresponding to folders and **datasets** corresponding to tables. A dataset is a table with one or more columns, but whose cells all share the same data type; in order to represent tables with heterogeneous column datatypes, a typical solution is to create the columns as individual datasets, then fetch rows from them simultaneously. Each dataset can be compressed using different algorithms, the default one being gzip. Both groups and tables can be associated with some metadata, divided into **attributes**, leading the format to be dubbed as self-descriptive. For all these reasons, the HDF5 file format is ideal for the storage and quick retrieval of big, heterogeneous datasets, which led to its usage in several fields, usually with specialized versions for different applications.

One such specialized version is the **cooler** file format, which was introduced in order to facilitate the storage and retrieval of Hi-C data (or any other type of genomically labeled array)[40]. A cooler file is therefore a HDF5 with a specific structure, represented in figure 1.4 (a). Each file contains 4 main groups:

- *bins*, containing ids and genomic coordinates of the bins the genome was divided into.
- *pixels*, containing the actual contact matrix in *ijv* form.
- *chroms*, containing ids for the chromosomes of the reference genome and their lengths.
- *indexes*, containing indexes for the pixels group, for faster pixel retrieval.

Each of these groups contains several datasets which represent individual columns of a single table, which is a common practice for HDF5 files as previously mentioned; since the group itself merely acts as a container to group the columns, from here on, the term **table** will be used to refer to the table obtained by joining the individual datasets present in the group.

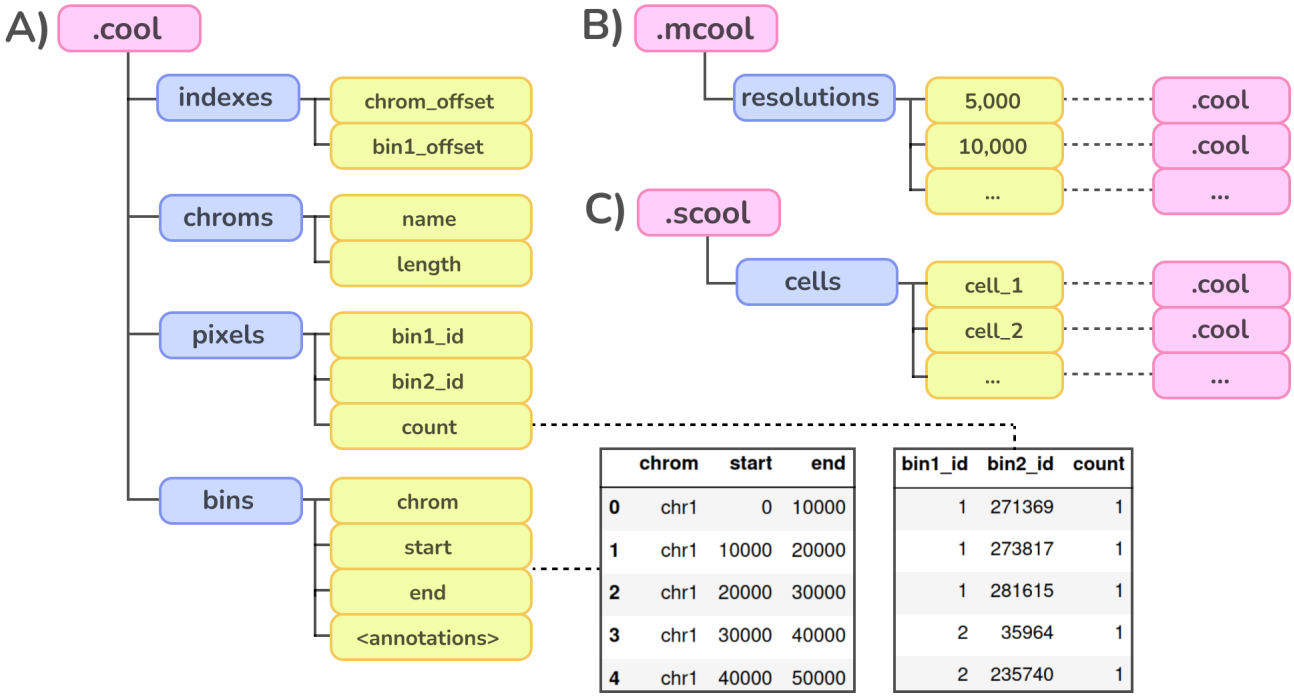


Figure 1.4: Structure of cooler files and derivates. Specifics of .cool, .mcool and .scool file formats. (a) .cool file structure, which includes bins, pixels, chromosomes and indexes groups. (b) .mcool file structure, where each group in the resolution one corresponds to a .cool file. (c) .scool file structure, where each group in the cells one corresponds to either a .cool or an .mcool file.

Each row in the **bins table** represents one of the bins the genome has been divided into. For each bin, `chrom`, `start` and `end` positions are always specified (i.e. "`chr1 10001 20000`"), while any number of columns containing additional bin annotations can be present. In general, the package *cooler*, used to create these files, adds some columns with bin weights for normalization purposes (see next subsection), but any other type of annotation, such as the presence of functional elements (i.e. promoters), can be present. While there is no column reserved for bin ids, the index of the bin in the bin table itself is the bin ID; this way it becomes easy to retrieve a certain bin by simply knowing its ID.

The **pixel table** is the *ijv* representation of the contact matrix. Each row has exactly three values, those being `bin1_id`, `bin2_id` and `count`, where the first two are the ids of the two interacting bins (row and column bin, respectively), while the third is the raw number of contacts found after binning and counting the aligned paired-ends reads. By referring to the bins via their ids, the pixel table can be drastically simplified, since each bin can be identified using just one column rather than the three necessary to specify the genomic coordinates. At any point, a subset of the pixel table can be converted to the extended form, in which bins are identified by their genomic coordinates, via the process called pixel annotation.

A `.cool` file contains only one pixels table at one resolution (one bin size). A multiresolution cooler, or `.mcool` file, is a collection of `.cool` files referring to the same experiment but at different resolutions. It is itself a single HDF5 file, with a root group called `resolutions` in which the individual coolers can be found in groups named after the binning resolution, as shown in figure 1.4 (b). It is important to notice that this format is mostly designed for grouping purposes; the different resolutions are completely independent from each other during processing, therefore working on one does not impact the others. Another specialized form of cooler format is the `.scool` format, which is instead a collection of `.cool` (or `.mcool`) files corresponding to different cells and it is thus used for single cell experiments. Its structure is shown in figure 1.4 (c).

1.3.4 Sequencing biases normalization strategies

Different bins have different genomic sequences and properties, leading to different sequencing biases. Several normalization strategies for systematic biases in Hi-C data exist and none of them seems to be a clear cut winner with respect to the others[41]. These normalization methods can be divided into explicit and implicit methods.

Explicit methods are methods which correct for a clearly defined set of systematic biases, such as GC content and mappability. The biases are assumed to be independent from each other; thus, for each bias, a normalization factor is computed for each bin, then the normalization factors are combined to obtain the overall bin normalization factor. It is important to notice though, that this type of technique generally considers the bins homogeneous, which even at a fairly low bin sizes might be seen as a major approximation. For instance, it is obvious that a 5 kb genomic sequence could be divided into areas with very distinct GC content; yet a single GC content normalization factor is applied to all reads falling into the bin, disregarding the fact that a read will never cover the entirety of the bin and it might fall in areas with significantly higher of lower GC content than the average for the bin.

Implicit methods are methods which do not clearly state the systematic biases to correct for. Their assumption is that all systematic biases are recapitulated by sequencing depth, and therefore, making sequencing depth comparable among bins should correct for all of them, whether they are known or not. This in turn equates to assuming that all genomic regions form the same number of interactions and that there is the same number of copies of each region in the genome (no aneuploidies). This is usually achieved through matrix-balancing methods, a class of mathematical algorithms which try to shape a matrix into some constrained form (in this case, comparable bin coverages) while retaining most of the properties of the original matrix. Among matrix-balancing methods, the most common are iterative correction and eigenvector decomposition (ICE)[42] and Knight-Ruiz (KR)[43], which are also commonly used as weight columns in `.cool` files. Matrix-balancing methods are individual-sample normalization approaches and they are generally outperformed by cross-sample normalization ones, especially in regards to reproducibility. Still, individual-sample normalization approaches have the major advantage of usually being significantly faster, also considering the fact that one file has to be normalized only once regardless of how many analyses and comparisons will be performed on it. For this reason matrix-balancing methods are the more common ones.

After computing bin normalization factors, be it through explicit or implicit methods, the pixel counts can be normalized by applying (usually by multiplying) the normalization factors of the two bins constituting the pixel. This entails a further assumption, that is that systematic biases are independent across bins.

1.4 Network analysis

A network is an object describing a set of entities and how they are connected to each other. Networks are a very convenient way of representing complex interactions among any type of entities, and are therefore adopted by many fields, including biology. The mathematical field responsible for the study of networks is graph theory, and for this reason the terms network and graph will be used interchangeably throughout the discussion. In the following subsections, some basic graph theory concepts useful to understand the algorithms presented in the methods chapter, will be introduced. Then, different methods to represent a graph will be briefly compared.

1.4.1 Graph theory concepts

A graph is a mathematical object G defined over a set of vertices, or nodes, V and a set of edges E, which can therefore be written as $G = (V, E)$. The vertices represent the entities of the graph, while the edges represent the connections among them. Graph order is defined as the number of its nodes ($|V|$), while graph size is defined as the number of its edges ($|E|$).

An edge is a pair of vertices belonging to the graph. If the pairs of vertices are ordered, meaning that (v_1, v_2) and (v_2, v_1) are distinct, the graph is said to be directed; conversely, if the pairs are not ordered, meaning that $(v_1, v_2) \equiv (v_2, v_1)$, the graph is said to be undirected. Moreover, if a weight is assigned to each edge, the graph is called weighted, else it is called unweighted. The different types of graph are shown in figure 1.5 (a).

The degree of a node is the number of incident edges, meaning the number of edges touching it; if the graph is directed, one can distinguish between out-degree and in-degree, those being the number of times the node appears as the first or second element in the ordered pairs, respectively. The neighbors of a node are the nodes directly connected to it by an edge (i.e. both nodes are incident to the same edge); the set of neighbors of a node is called neighborhood. In a weighted graph, node strength is the sum of the weights of all incident edges.

Graph density is a measure of how interconnected the nodes of the graph are on a global scale; graph density is defined as the number of edges present in the graph over the number of possible edges for that graph, which is $|V|(|V| - 1)/2$ in the undirected case, $|V|(|V| - 1)$ in the directed one.

1.4.2 Graph representations

Aside from the standard graphical representation using lines and dots, a network has three other major representation methods: edge list, adjacency matrix and adjacency list. These are shown in figure 1.5 (b).

A network can be represented using an **edge list**, which corresponds to listing all edges which do exist in the network, ideally in an ordered manner. This representation is memory efficient, since all edges are listed only once, but it is not very convenient for searching specific edges.

An **adjacency matrix** is a matrix with all nodes of the graph as both rows and columns. If the edge individuated by a set of coordinates (i.e. a pair of nodes) exists, the corresponding cell will contain the edge value (edge weight in the weighted case, 1 in the unweighted one); if it is not present, the cell will contain zero. Notice that in the case of an undirected graph, the matrix is symmetric with respect to the main diagonal. This representation is not memory efficient, especially for sparse graphs, since it always stores $|V|^2$ values, but it is quite fast to retrieve all node neighbors and in general perform operations.

In the **adjacency list** representation, a list containing all neighbors is associated with each node. This representation is as memory efficient as the edge list when used for directed graphs, since all edges are listed only once; in case of an undirected graph, this representation is slightly less memory efficient, since each edge is present in the adjacency list of both incident nodes, therefore requiring to store $2|V|$ values. This representation is convenient in order to retrieve node neighbors, though not quite as convenient for other types of operations.

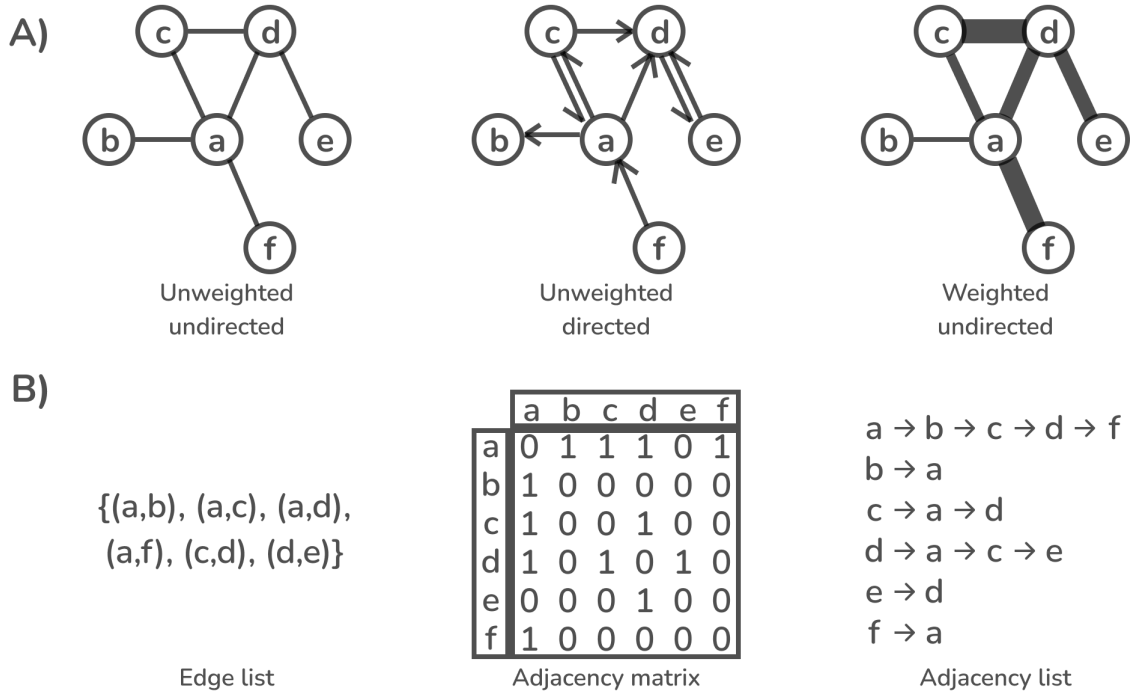


Figure 1.5: Graph features and representations. (a) Different types of networks. Considering the unweighted undirected graph, we can say that it has both order and size equal to 6. Taking as an example node a, in the undirected cases both in-degree and out-degree are equal to 4, while in the directed case its in-degree is 2 while its out-degree is 3. The neighborhood of node a in the unweighted undirected graph is the set of nodes $\{b, c, d, f\}$. In the weighted case, the strength of node a is given by the sum of its incident edges, which equates to 17. (b) Different ways of representing a network. From left to right, edge list, adjacency matrix and adjacency list representation of the unweighted undirected graph from point a of this image.

Chapter 2: Objectives

Hi-C data is intrinsically hard to process due to several factors, among which the fact that it is represented by a huge, extremely sparse, upper triangular matrix; this leads to challenges both in terms of memory management and computational runtime, as well as processing in general. There exist some tools which allow to work with this type of data, notably those grouped under the Open Chromosome Collective[44]. Among these, cooler[40] is used to handle data storage, while cooltools[45] can be used for general preprocessing and certain specific types of analysis (compartment and boundaries definition, feature and pattern pileup).

Nevertheless, at the time of writing, it seems that very few tools for network analysis of Hi-C data exist yet[46]. This is probably because Hi-C data can be converted into very dense and high order networks, which are memory-demanding and difficult to handle. For this reason only standard analyses which have been optimized for memory efficiency can be currently conducted on them. Still, this is quite unexpected, given the fact that Hi-C contact matrices are examples of adjacency matrices, which are natural representations of graphs. Rephrasing for clarity, Hi-C contact matrices can be easily converted into weighted, undirected graphs, with nodes representing the bins in which the genome has been divided into, and edges representing the contacts among those regions. The weights of the edges can be defined as the raw numbers of contacts obtained from sequencing or (better) some normalized measure derived from them. Network analysis of Hi-C data would provide information complementary to the one obtained through current analysis techniques, thus allowing to better study and characterize chromatin conformation and its role.

HiCONA is a package which aims to address this lack of tools for Hi-C data network analysis as well as the technical challenges in working with these networks; in order to do so, it provides some general functionalities which are useful regardless of the network analysis task of interest, as well as specialized version of network analysis algorithms already used in data science or other fields. Since the package was developed in Python3 using a fully object-oriented approach, it is easy to extend with any new functionalities or needs which may arise. Package functionalities can be grouped into three main tasks:

- Pixel preprocessing, which includes all steps required to prepare the pixels contained in the original data file, encompassing filtering, normalization and sparsification. These steps are aimed at achieving networks with fairly consistent properties in order to facilitate the comparison of different experiments, which is currently quite difficult due to the high variability tied to the Hi-C technique itself.
- Bin annotation manipulation, meaning some utility functions which can be used to add, or manipulate, bin annotations starting from bed-like files. This exploits the capability of networks to integrate data, in order to allow for more in depth and robust analysis.
- Network generation and analysis. While the package itself provides some algorithms, another objective is to be able to integrate this tool with others, and for this reason import and export to standard graph file formats (such as .xml) is also supported.

Hi-C data files can be quite large and cumbersome to work with; moreover network analysis itself is not trivial especially regarding memory usage. HiCONA strives to make possible to process almost any Hi-C data file even on laptops; for this reason, particularly demanding operations, such as the preprocessing, have been optimized keeping into account the specific properties of Hi-C data, in order to reduce memory usage and computational runtime by chunking, vectorizing and parallelizing operations.

As previously mentioned, the main focus of discussion will be pixel preprocessing.

Chapter 3: Materials and methods

This chapter presents the HiCONA package, how it was implemented and what it can be used for. The main focus is on the steps leading to the creation of a network which can be then analyzed using standard or custom network analysis algorithms. One such algorithm, node-labels permutations, is then illustrated. The chapter does not explain how to actually use the package; for that, ample documentation and notebooks will be available as soon as the package is made public.

3.1 Implementation

Hi-C Organization Network Analysis (HiCONA) is a **Python3 package** compatible with Python version 3.8 and later. The package strives to provide a fast, memory-efficient, flexible and user-friendly framework for network analysis of Hi-C data. In order to do so, it is implemented taking into account some key aspects:

- A fully **object-oriented programming** (OOP) approach is used to keep the functions organized and make it easy to extend with further functionalities.
- Well established file formats are used for both input and output, in order to facilitate package **integration** into already existing analysis pipelines.
- All expensive operations are **optimized** both in terms of memory usage and computational runtime, in such a way that almost any file can be quickly processed even on less performing systems such as laptops. This is mostly achieved by chunking and vectorizing operations.

HiCONA is mostly built on top of two main packages, those being *cooler* and *graph-tool*. ***Cooler*** is a library designed to work with `.cool` files[40]; from its Python API, HiCONA extends the main object class `Cooler` into `HiconaCooler`, which is responsible for data I/O, preprocessing and bin annotation. ***Graph-tool*** is a Python package for network analysis[47]; it is highly optimized and fast since all algorithms and data structures are implemented in C++. The `Graph` object class from *graph-tool* is extended into `HiconaGraph`, which is responsible for network manipulation and analysis. Other major packages HiCONA relies on are *scipy*[48] for integral computation, *numpy*[49] for algorithm implementation, *pandas*[50] for data manipulation, *pybedtools*[51] for bin annotation, *seaborn*[52] and *matplotlib*[53] for plotting. Aside from the previously mentioned `HiconaCooler` and `HiconaGraph` classes, HiCONA also implements a `ChromTable` class to facilitate conversion among the two and some utility classes such as custom iterators for these object types.

3.2 Package functionalities

HiCONA implements everything which is needed to perform network analysis starting from a `.cool` file. As shown in the usage flowchart in figure 3.1, these functionalities can be roughly divided into three groups:

- **Pixel preprocessing**, which encompasses pixel filtering, normalization and sparsification, which allow to obtain chromosome-level tables ready to be converted into graphs, starting from the full contact matrix.
- **Bin annotation manipulation**, which includes adding bin annotations from bed-like files, deleting bin annotations or converting categorical bin annotations into one-hot encoding form.
- **Network generation and analysis**, which comprehends the creation of the graph object from the preprocessed chromosome-level pixel table and all the network analyses conducted on it.

3.3 Pixel preprocessing

Pixel preprocessing includes all steps required to go from the full pixel table contained in a `.cool` file to one or more chromosome-level pixel tables, which are normalized subsets of the full table stored in the `.cool` file

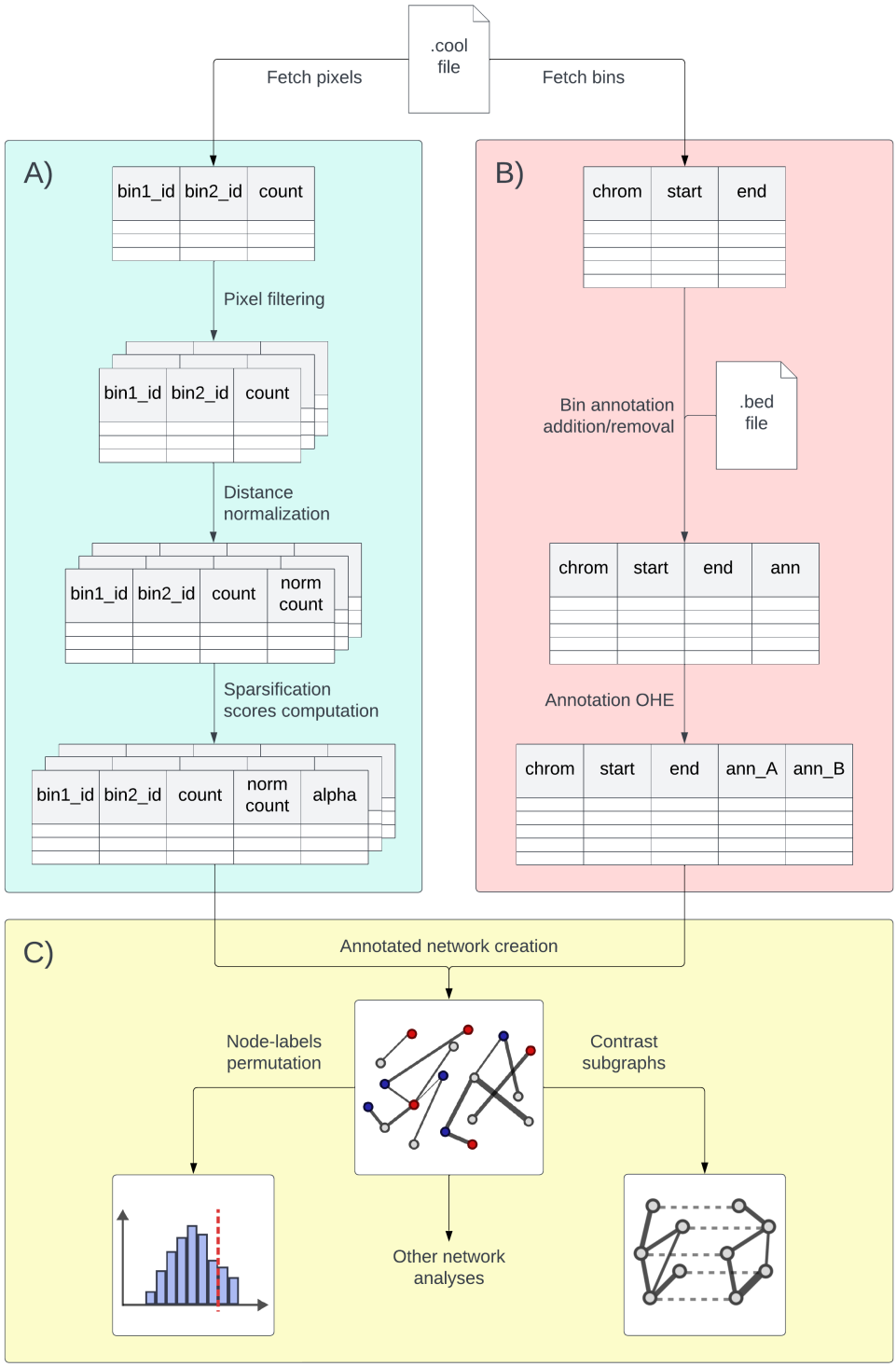


Figure 3.1: HiCONA package flowchart. Typical steps required for Hi-C data analysis using HiCONA. (a) Pixel preprocessing steps. Starting from the complete contact matrix, filter, normalize and compute the sparsification scores of the pixels. The end result is a reduced pixel table ready for conversion to network. (b) Bin annotation manipulation steps. All steps needed to add, remove or modify the optional bin annotation columns present in the file. (c) Network generation and analysis steps. Combine the prepared pixel table subsets with the bin annotations to create networks which can be analyzed with a multitude of algorithms.

for later usage in network creation. Pixel preprocessing is a key step in order to be able to conduct downstream analyses, especially those requiring the comparison of different experiments and/or replicates. In order to guarantee consistent preprocessing, the procedure is fairly streamlined; all steps are handled by a single public method of the `HiconaCooler` class (`create_tables`), to which the user can pass arguments if they want to supersede the default parameter values. For most applications, the default values should already be the optimal ones. If no default value is superseded, one chromosome-level pixel table for each chromosome will be created using those defaults. The preprocessing steps, described in the following subsections, are shown in the context of the usage flowchart in figure 3.1 (a).

3.3.1 Pixel filtering

The initial preprocessing step is the retrieval and filtering of subsets of the pixels from the whole pixel table. For each specified chromosome (or all of them if no specific one is selected), a table containing all pixels whose first bin falls into that chromosome is created. Each of these tables is then filtered in order to remove:

- **Inter-chromosomal pixels**, meaning pixels whose bins do not fall into the same chromosome. This is the main simplifying step which allows to break the entire contact matrix into tables of manageable size; moreover, inter-chromosomal pixels are removed since they also constitute a problem for genomic distance normalization (see the following subsection).
- **Self-looping pixels**, meaning pixels whose bins are identical (main diagonal of the contact matrix). Self-looping pixels can exist since the two sequences constituting the sequencing read in the Hi-C experiment are typically way shorter than the bin size, and can therefore fall into the same bin. These pixels are removed since the contact frequency is too variable, leading to a very noisy signal.
- Pixels whose bins have a **genomic distance** among them **above a certain threshold**; this corresponds to removing the pixels where the bins are so far apart that is likely that the contact is random and not biologically significant. Currently, the default value for this distance is 200 Mb.
- Optionally, pixels whose count value is lower than a threshold; this can be used to speed up computation in less performing systems, but it should be avoided if possible since it introduces instability during distance normalization and most pixels with very low counts are removed during network sparsification anyway.

Notice that, the way the initial subsetting is performed, not all inter-chromosomal pixels are fetched, specifically not those where the bin belonging to the chromosome of interest is the second in the pixel; this does not matter currently, since inter-chromosomal pixels are filtered out anyway.

As of now, subsetting is supported only at chromosome-level and this is because subsetting on arbitrary genomic coordinates would lead to some issues. Even though the preprocessing procedure would work without issues (given that it is chunked), subsetting on a region larger than a chromosome could lead to extremely big networks downstream which would be difficult to handle; moreover, unless the inter-chromosomal filter is removed, the two chromosomes would be disconnected anyway. On the other hand, subsetting on a region smaller than a chromosome would skew the sparsification procedure, since it would arbitrarily remove some connections of the bins. Arbitrary subsetting might be added in the future if a compelling reason is found.

3.3.2 Pixel distance normalization

A very important notion to keep into account, while preprocessing this type of data, is pixel genomic distance, which is the **genomic distance** among the two bins composing the pixel. Formally, this can be defined as

$$d_i = r(b_{2i} - b_{1i})$$

where d_i is the genomic distance for pixel i, b_{2i} and b_{1i} are the bin ids for the second and first bins of that pixel, r is the working resolution, meaning a positive integer representing bin size in bp. It is known that the probability of random contact among two bins exponentially decays with the genomic distance among them[54]; this means that pixels whose bins are closer to each other will tend to have significantly higher counts with respect to pixels whose bins are further away just due to random chance. This in turn translates into the fact that, without some correction for this bias, potentially meaningful long range interactions will be outshadowed

by random close range interactions. The most natural way to perform this type of normalization is to rescale the counts using some summary statistic which is a function of genomic distance ($P(d)$).

Although not directly for normalization purposes, the `cooltools` package[45] can compute one such summary statistic for intra-chromosomal pixels, that being the expected counts given a genomic distance. Formally speaking, it can be defined as

$$P(d) = \frac{\sum_{i=1}^{n} c_i \cdot I(d = d_i)}{\sum_{i=1}^{n} I(d = d_i)}$$

where c_i is the count value of the i-th pixel, n is the total number of pixels, and I is an indicator function (1 if the genomic distance of the i-th pixel is the same as the one of interest, 0 otherwise). From a graphical point of view it corresponds to averaging the count values of all pixels on the same diagonal of the contact matrix. In reality the algorithm is slightly more complex, working on individual chromosome arms and removing some regions such as telomeres and centromeres. Since at high genomic distances the number of averaged pixels becomes progressively smaller, the expected counts estimates become more and more unstable; for this reason `cooltools` can perform a smoothing of the expected counts by distance curve by averaging the values in a sliding window of increasing size (basically, use the fact that very similar distances should have only a slight difference in expected counts). This method is straightforward and immediate to understand, yet it has a major downside when used for normalization purposes: all pixels at a given distance are considered, even those with count value equal to zero, and given that these zero-count pixels are the majority of pixels, especially at high genomic distances, the expected counts quickly tend to zero. Extremely small values are difficult to use for normalization purposes, since they cannot be used as fraction denominators, otherwise the values would explode.

For this reason, HiCONA proposes a slightly different approach to compute the **summary statistic** for normalization purposes: only non-zero count pixels are considered and the median value is taken for each distance rather than the mean. Formally, this can be written as:

$$P(d) = median(\{c_i | (d_i = d) \wedge (c_i \neq 0) \, \forall i \in [1, n]\})$$

where $\{\}$ represents a multiset (set with multiplicity). Notice that this means that the normalization factors will always be positive integers, therefore solving the denominator issue. Once the $P(d)$ curve has been computed, the **normalized count value** for the i-th pixel can be computed as

$$m_i = \log_2 \left(\frac{c_i}{P(d_i)} + 1 \right)$$

where m_i is the normalized count value, c_i is the count value for that pixel and $P(d_i)$ is the normalization factor corresponding to the genomic distance of that pixel. Even though the normalized count value is a floating point number, it is derived from a fraction of two integers which can take a fairly limited number of values (especially the denominator); for this reason, the number or actual values that the normalized counts can take is relatively small, allowing the usage of memoization in later steps to speed up computations.

Optionally, the user can specify a percentile as filter; all pixels whose normalized counts fall below the specified percentile are removed. This is done to both remove some noise as well as reduce dimensionality.

All considerations up till this point are applicable only to **intra-chromosomal pixels**. This is because it is not possible to define a notion of genomic distance for inter-chromosomal ones. Still, `cooltools` does give the option of computing the expected counts for inter-chromosomal pixels; given a pair of chromosomes, the expected counts are given by the average of all pixels counts where one bin belongs to one chromosome and one bin belongs to the other. HiCONA could implement a similar method for inter-chromosomal pixel normalization but it currently does not. This is because, ultimately, the normalized values are used to compute the edge weights; if both inter and intra-chromosomal pixels were to be kept, the network would contain edges whose weights were computed using different procedures, and therefore there would not be a way to guarantee that the comparison is fair. For this reason, and those listed previously, all inter-chromosomal pixels are removed.

3.3.3 Pixel sparsification score computation

The normalized chromosome-level pixel tables could already be converted into networks; that being said, the resulting networks would be rather big in size (about $7 \cdot 10^7$ edges) and noisy (many edges with very low weight). In order to mitigate these problems, a network sparsification algorithm[55] is used; the objective of the procedure is to sparsify the network, i.e. reducing graph density (the number of edges in the graph), while retaining most of the overall topological properties and structure of the network.

While standard top-n edge filtering keeps only the top n scoring edges according to their weight, network sparsification keeps the edges which are contributing the most in defining the first order neighborhoods of the nodes in the graph. This means that edges with relatively low weight but very high contribution in defining the local structure of the network are kept using network sparsification, while they would not be using top-n edge filtering. This is important when trying not to belittle nodes with lower strength.

The algorithm, shown fully in Alg.1, can be summarized as follows. For each edge ij in the graph, tangent to the nodes i and j, two sparsification alpha scores are computed, one for each tangent node ($\alpha_{ij,i}$ and $\alpha_{ij,j}$ respectively); using node i as an example, the sparsification score would be computed as

$$\alpha_{ij,i} = 1 - (k_i - 1) \int_0^{p_{ij,i}} (1 - x)^{k_i - 2} dx$$

where k_i is the number of neighbors of node i and $p_{ij,i}$ is the normalized edge weight for edge ij considering the neighborhood of i, which means

$$p_{ij,i} = \frac{w_{ij}}{s_i}$$

with w_{ij} being the weight of edge ij and s_i being the strength of node i (both of which distance normalized). The value $\alpha_{ij,i}$ represents the probability (or p-value) of edge ij having weight w_{ij} assuming random distribution of edge weight among all neighbors of node i. The same procedure is repeated for node j. After computing both $\alpha_{ij,i}$ and $\alpha_{ij,j}$, only one value is kept for each edge, generally the minimum of the two

$$\alpha_{ij} = \min(\alpha_{ij,i}, \alpha_{ij,j})$$

After having computed α_{ij} for every edge, the graph can be sparsified by removing all edges whose sparsification score is below a certain threshold α

$$\text{keep edge } ij \iff \alpha_{ij} < \alpha$$

Some considerations to be made regarding this algorithm:

- If the number of neighbors is 1, then the alpha score will always be 1. This means that, for an edge tangent to a terminal node, the sparsification score derived from the non-terminal node will always be chosen, since it will always be less than, or equal to, 1. Moreover, if both nodes tangent to an edge are terminal, meaning that they form a disconnected component of two nodes (*doublet*), the edge will always have a sparsification score equal to 1 and thus the edge will always be removed regardless of the threshold used. This is fine since a disconnected doublet is not very informative for network analysis purposes.
- Given that the sparsification alpha values are p-values, they always take a value in the range $(0, 1]$. Moreover, given that they are p-values, one might expect multiple correction testing to be needed; this is not the case, given the fact that the main objective is not to filter out the significant edges, but rather to denoise and try to remodel the network to a somewhat standard and reproducible form usable for comparison. For the same reason, the alpha cutoff is not necessarily the standard 0.05 generally used for p-values, and even in the original sparsification paper other values are suggested and used.
- HiCONA also allows the selection of the maximum of the two sparsification scores, since they have different meanings. Choosing the minimum means that the edge will be kept if it is significant for at least one node; on the contrary, choosing the maximum would mean keeping the edge only if significant for both nodes. Using the minimum can be seen as the more conservative option, while using the maximum yields a less noisy graph though at the expense of some potentially relevant weak edges.

HiCONA does not directly sparsify the network, but rather stores the sparsification scores for a later use; by deferring network filtering, one can generate networks using different cutoffs without having to recompute the scores. HiCONA does allow filtering according to any arbitrary threshold; for this reason one might want to use

Algorithm 1 Network sparsification algorithm, Serrano et. al, 2009

Input: G = a weighted, undirected graph defined over vertices V and edges E, α = sparsification cutoff
Output: G' = the sparsified graph

Step 1: sparsification score computation
Set $\alpha_{ij,curr} = 1, \forall (i,j) \in V, j \in$ neighbors of i (Note: $\alpha_{ij} = \alpha_{ji}$ since G is undirected)
for $i \in V$ **do**
    w_{sum} = sum of the weights of all edges incident to i
    k_i = degree of node i
    for $j \in$ neighbors of i **do**
        $w_{norm} = w_{ij}/w_{sum}$
        $\alpha_{ij,new} = 1 - (k_i - 1)\int_0^{w_{norm}} (1-x)^{k_i - 2} dx$
        $\alpha_{ij,curr} = \min(\alpha_{ij,curr}, \alpha_{ij,new})$
    end for
end for

Step 2: network filtering
G' = empty graph
for $e_{ij} \in E$ **do**
    if $\alpha_{ij,curr} < \alpha$ **then**
        Add edge e_{ij} to G'
    end if
end for

a fixed value as cutoff, for instance 0.05. However, one such value is, indeed, arbitrary and does not necessarily meet the need for comparable graphs. For this reason the package can compute a graph-dependent threshold, defined as the value which maximizes the fraction of retained nodes (nodes that are still tangent to at least one edge after the filtering) while also minimizing the fraction of edges in the graph. To do so, multiple thresholds are tested and the one with the smallest Euclidean distance from the point $(1, 0)$, in the plane with the fraction of retained nodes on the x-axis and the fraction of retained edges on the y-axis, is chosen.

The network sparsification procedure is quite simple to apply, assuming that the entirety of the data can be loaded into memory either in a dense adjacency matrix or in a graph-like structure. While the former is not an option due to the previously discussed nature of Hi-C data, a graph can be reasonably generated, though the amount of memory required might be quite prohibitive if a server is not being used. The algorithm could be thus modified to work with chunks of the graph rather than the entirety of it, though this is non-trivial considering that the data is in ijv form. For this reason, HiCONA implements a modified version of the algorithm which works on chunk of the ijv table; though the modified algorithm has a slightly higher complexity, requiring more passes of the pixels, it allows to work at a fixed RAM-footprint specified by the user, making it possible to be used even on laptops. Moreover, the modified version is easy to vectorize and it is therefore quite fast (and it could be sped up even more using parallelization).

3.4 Bin annotation manipulation

The cooler format is structured in order to accommodate for any number and type of bin annotations, and in fact most `.cool` files have a bin normalization weight column by default. Although this is the case, the *cooler* package does not provide a very convenient facility to manipulate these annotations; for this reason HiCONA does implement methods to add, remove and manipulate bin annotations, which are shown in figure 3.1 (b).

3.4.1 Adding and removing bin annotations

HiCONA allows the inclusion of annotations to the bin table of the `.cool` file starting from a bed-like file. To be more precise, the package uses *pybedtools* to perform a left outer join of the bin table and the bed-like file. The user can then decide to keep any number of columns from the bed-like file to use as annotations; moreover, a binary annotation column can be created, containing 1 if the bin has any overlap with any interval in the bed-like file. Currently any number of overlapping bases greater than zero is considered enough for bin annotation.

HiCONA does allow removing bin annotations by specifying the name of the annotation to be removed; however, currently, removing an annotation does not reduce file size. This is due to how `.hdf5` files work since, rather than removing a dataset, they remove its memory address pointer so that it cannot be reached anymore (this is to avoid having to shift the position of the other bytes in memory and reindexing). To reduce file size after removing an annotation, an external tool such as *h5repack* can be used, though this functionality might be added natively in the future.

3.4.2 Bin annotation OHE

Working with a categorical bin annotation with multiple modalities is not convenient, especially for algorithms such as node-labels permutations. For this reason, HiCONA allows the conversion of any categorical bin annotation to One Hot Encoding (OHE). This means that, given a categorical annotation, for each modality of that annotation a new binary annotation is created, where 1 indicates that the bin was annotated as that modality, 0 otherwise. For example, the annotation $ann = [A, B, A, A]$ would be split into $ann\_A = [1, 0, 1, 1]$ and $ann\_B = [0, 1, 0, 0]$. A hard cap is placed on the number of modalities an annotation can have and be split, in order to prevent splitting annotations such as gene names into thousands of columns.

3.5 Network generation and analysis

After the chromosome-label pixel tables have been created, and any required annotation has been added to the bin table, the networks are ready to be generated and analyzed. The minimal network is created by selecting a pixel table to use as edge list; unless otherwise specified, the edges are then annotated with raw counts, normalized counts and sparsification score, while the nodes are annotated with the corresponding bin id, chromosome, start and end. Any number of other node annotations can be also added, provided that it is present in the bin table. Moreover, a sparsification score cutoff can be specified to filter the edges prior to building the network.

Once a network, which is an instance of the `HiconaGraph` class, has been created, it can be analyzed either with one of the many generic network analysis algorithms inherited from the *graph-tool* package, or using one of the more specific algorithms implemented by HiCONA. Currently HiCONA only implements node-labels permutation, while contrast subgraphs extraction is being developed (both shown in figure 3.1 (c)), but the package is structured in such a way that it can be easily extended to accommodate other analysis algorithms.

3.5.1 Node-labels permutation

Node-labels permutation, or simply node permutation, is a widely used algorithm in network science; its main objective is to test whether the average (or distribution) of some node-level statistic differs among nodes with different labels (bin annotations). As an example on Hi-C data, one might want to test whether bins containing a promoter region tend to form more connections, on average, with respect to bins containing a boundary region.

The algorithm, displayed in Alg.2, can be summarized as follows. A network, whose nodes are annotated as either A, B, none or both, is given; A and B are the two labels which are to be compared. Once some node-level statistic has been defined, that statistic is computed for all nodes annotated as A and then averaged. After this has been repeated for the nodes annotated as B, the $\log_2$ fold change among the two is computed; this is the *original* fold change. Then, an empirical distribution is created by randomly shuffling node annotations and recomputing the fold change for N times, where N is the number of specified permutations; notice that the actual structure of the network never changes. The fraction of *permuted* fold changes which are more extreme (higher) than the *original* fold change is the p-value for the test $H_0 : S(A) \leq S(B)$, $H_1 : S(A) > S(B)$, where A and B represent the sets of nodes annotated as A and B, respectively, and S is the average statistic for a node set.

Algorithm 2 Node-labels permutation algorithm

Input: G = a graph defined over vertices V and edges E; each vertex can be annotated as A, B, both or neither. S = the average of a node-level statistic over a set of nodes. N = the number of permutations to perform.

Output: The p-value for the test $\begin{cases} H_0 : S(A) \leq S(B) \\ H_1 : S(A) > S(B) \end{cases}$

Define a vector of node labels $L = \left[l_1, l_2, \ldots, l_{|V|}\right]$, such that each element is a tuple $l_i = (l_{Ai}, l_{Bi})$ with $l_{Ai} = 1$ if v_i is annotated as A, 0 otherwise (define l_{Bi} analogously)
Define the sets $A = \{v_i | v_i \in V, l_{Ai} = 1\}$, $B = \{v_i | v_i \in V, l_{Bi} = 1\}$
Compute $s_{original} = \log_2(S(A)/S(B))$
Initialize $counter = 0$
for $n \in [1, N]$ **do**
    $L_n = [l_{\pi_n(1)}, l_{\pi_n(2)}, \ldots, l_{\pi_n(|V|)}]$, where π_n is some permutation function
    $A_n = \{v_i | v_i \in V, l_{n,Ai} = 1\}$, $B_n = \{v_i | v_i \in V, l_{n,Bi} = 1\}$
    $s_{permutation} = \log_2(S(A_n)/S(B_n))$
    if $s_{permutation} > s_{original}$ **then**
        $counter = counter + 1$
    end if
end for
$p_{val} = (counter + 1)/(N + 1)$

The annotations which can be used for this algorithm are all those added to the bin table, as long as they were converted to OHE. Moreover, HiCONA can assign the mock label *universe* to all nodes in the network, which allows to compare nodes with a specific label with respect to the entire network.

Like any statistical test which has to be applied multiple times, some p-value correction should be applied, such as the *Benjamini-Hochberg* correction. This is because the procedure is likely to be repeated for all chromosomes, for multiple pairs of annotations and possibly for multiple resolutions, therefore leading to some hundreds of tests. Currently HiCONA does not natively apply any correction.

Different node level statistics can be used, with different meanings at the network level. Some statistics that can be used, and which are implemented in HiCONA, are degree, betweenness centrality and local clustering coefficient. For an in depth explanation of their definition and meaning, refer to Appendix I.

3.6 Benchmarking

In this section, both files and methods used to benchmark memory and runtime of the tool are presented. Benchmarking was performed only on the preprocessing steps, given that they are the limiting factors in term of performance.

3.6.1 Memory and runtime benchmarking

Given the focus of the package on being fast and usable even on less performing systems, benchmarking on both RAM usage and computational runtime are needed. This is especially true for the preprocessing part, when data dimensionality is still very high.

Memory benchmarking is performed through the *memory_profiler*[56] Python package. To display RAM usage line-by-line, the functions to profile must be preceded by the *@profile* decorator, then the code can be run to generate a log-like file. The package also allows to display RAM usage overtime, however the default package plotting functions have been substituted with custom ones for better clarity.

Since runtime benchmarking was performed only on the preprocessing steps, none of which are parallelized yet, wall time was used rather than CPU time, in order to have an estimate from a more practical point of view. Computational runtime benchmarking was performed simply using the built-in *timeit* module.

3.6.2 Test datasets

Several datasets have been used for benchmarking purposes, all of which are available on the 4D Nucleome data portal[57]. Although those datasets are available as `.mcool` files, due to a bug during file generation (confirmed by the database maintainers), the `.cool` files had to be re-generated starting from the `.pairs.gz` files using *pairix*[58] and *cooler*. The datasets were analyzed at the 5 kb and 10 kb resolutions; this is because these bin sizes are a good compromise in terms of having high enough detail while still working with contact matrices which are not exceedingly sparse. For each dataset, rather than file size, the number of pixels is reported; this is because the actual number of pixels is a better proxy for how the algorithms will scale, considering that:

- The pixel table will always be some orders of magnitude bigger than the bin table; for instance, the 10 kb bin table for a human cell line file will have around 3×10^5 rows (constant number depending on genome size), while an average pixel table will have around 1.5×10^9 rows (variable number depending mostly on experiment coverage and number or replicates merged to create the file).
- Total file size takes into account bin annotations, which can take a huge amount of space (especially when categorical with many modalities, such as names, since they are stored as strings) and do not impact preprocessing run time.

The procedure was tested on data coming from different cell lines processed with the same restriction enzyme, as well as on data coming from the same cell line processed with different restriction enzymes. This is to start testing whether cell type and restriction enzyme introduce noticeable bias; still, a more in depth analysis, with multiple files per combination of cell type and restriction enzyme, would be needed for a definite answer to this question. For two cell types, GM12878 and IMR90, the biological and technical replicates, which were merged to obtain the full-size files, are also analyzed individually; this is to test how reproducible the procedure is on replicates, as well as how similar the processed full-size files are to the processed individual replicates. This point is of particular interest since a high concordance would mean that the same results could be obtained by performing less experiments and sequencing, reducing costs for a technique which is rather expensive.

The package is designed to work with `.cool` files as input, since the main objective is the analysis of Hi-C data; still, any type of data representable in `.cool` format can be analyzed, therefore the package has also been tested on a data file coming from another technique, namely Micro-C.

The characteristics of the analyzed datasets are summarized in Table 3.1.

4DN ID	Publication	Experiment	Enzyme	Cell Type	Res (kb)	Pixels
4DNFIYECESRC	Rao, 2014	In situ Hi-C	MboI	GM12878	5	$1.89 \cdot 10^9$
					10	$1.57 \cdot 10^9$
4DNFIIG4IWKW	Rao, 2014	In situ Hi-C	MboI	IMR90	5	$4.95 \cdot 10^8$
					10	$4.03 \cdot 10^8$
4DNFIIFAUT24	Rao, 2014	In situ Hi-C	MboI	HMEC	5	$2.15 \cdot 10^8$
					10	$1.83 \cdot 10^8$
4DNFIYL35EHL	Rao, 2014	In situ Hi-C	MboI	HUVEC	5	$2.00 \cdot 10^8$
					10	$1.77 \cdot 10^8$
4DNFIUTE4F4B	Oksuz, 2021	In situ Hi-C	HindIII	H1-hESC	5	$1.41 \cdot 10^8$
					10	$1.26 \cdot 10^8$
4DNFIUPGJLFO	Oksuz, 2021	In situ Hi-C	DpnII	H1-hESC	5	$8.91 \cdot 10^7$
					10	$7.41 \cdot 10^7$
4DNFIPVA6VYB	Oksuz, 2021	In situ Hi-C	DdeI	H1-hESC	5	$8.16 \cdot 10^7$
					10	$7.41 \cdot 10^7$
4DNFIR1FK55F	Oksuz, 2021	Micro-c	MNase	H1-hESC	5	$5.73 \cdot 10^7$
					10	$4.39 \cdot 10^7$

Table 3.1: Summary of the datasets used for benchmarking. Table summarizing the characteristics of the analyzed datasets. From left to right: 4DNucelome Data Portal id of the .pairs file converted to .cool, reference publication, experiment type, restriction enzyme used for DNA fragmentation, cell type, resolution (bin size) in kilobases, number of pixels in the file.

Chapter 4: Results

This chapter will focus on benchmarking the preprocessing procedure implemented by HiCONA, computing various statistics at different steps, measuring time and memory performance, as well as performing some biological validation via replicate correlation. For some steps, HiCONA will also be compared to loop-callers, meaning tools which try to identify pixels corresponding to loop domains using various approaches. Though loop-callers do not have the exact same objective as HiCONA, they will still be used as reference since they are the only tools found in literature which could reasonably fulfill this role. No proper network analysis will be presented, since the algorithms are still being implemented and tested, though some preliminary results will be mentioned in the discussion.

Unless specified otherwise, all the data and analyses shown in this chapter were performed by my tutor, Leonardo Morelli, using HiCONA, which I implemented, as well as other custom scripts. The comparison with cooltools and the benchmarking were performed by myself; moreover, I adapted all the analyses and plots to highlight aspects relevant to the way HiCONA was implemented.

4.1 Pixel preprocessing steps analysis

The objective of preprocessing is to create chromosome-level networks and reduce their density using network sparsification, that is, extracting the backbone composed of the most significant interactions while also preserving topology. Moreover, since the network has a biological meaning, there are other characteristics that could be kept into account; one such characteristic is the distribution of the genomic distances of the edges, i.e. the pixels. For instance, one could try and preserve this distribution, meaning that the preprocessing procedure should remove, for each genomic distance, an amount of pixels proportional to the initial fraction of pixels with that distance; this would guarantee that the algorithm is not biased in favor of some genomic distance. This objective cannot be achieved with a method such as network sparsification, since genomic distance is not taken into account during the sparsification step; still, it is worth keeping track of mean and median genomic distance at each step to guide in the choice of parameters to use and, in general, to analyze algorithm behavior.

With this in mind, the individual processing steps will be analyzed. Firstly, distance normalization will be discussed, then pixel filtering. This is because, though some filtering steps do happen prior to normalization, the last filter is applied on normalized counts; thus, it seems clearer to tackle filtering all at once. Then, network sparsification will be discussed.

As another remark, each preprocessing function works with exactly one chromosome at a time. The chromosomes are fully independent from each other throughout the entirety of the preprocessing (and, in the future, of the analysis); this can make it quite difficult to aggregate the data coming from all the chromosomes of an individual processed file. For this reason, data will be aggregated when possible; when no convenient way is found, one chromosome from one of the files will be used as an example, and the same analysis conducted on at least another random chromosome of the same file, as well as on a random chromosome from another file, will also be included in the Appendix II as replication data.

4.1.1 Normalization factor computation

Pixels whose bins are close in the genome tend to have significantly higher counts than pixels whose bins are far apart, simply due to random interactions; it is thus needed to penalize short-range interactions to allow for a fair pixel comparison during filtering and sparsification. In order to normalize for genomic distance, it is necessary to compute some summary statistic $P(d)$, with d being genomic distance, to use as normalization factor (i.e. denominator of a fold change); here we compare the statistics obtained using *cooltools* and HiCONA (for the formulas refer to subsection 3.3.2).

As previously mentioned, the probability of two genomic regions interacting by chance decays exponentially with their distance from each other; for this reason, we also expect the function describing probability of contact

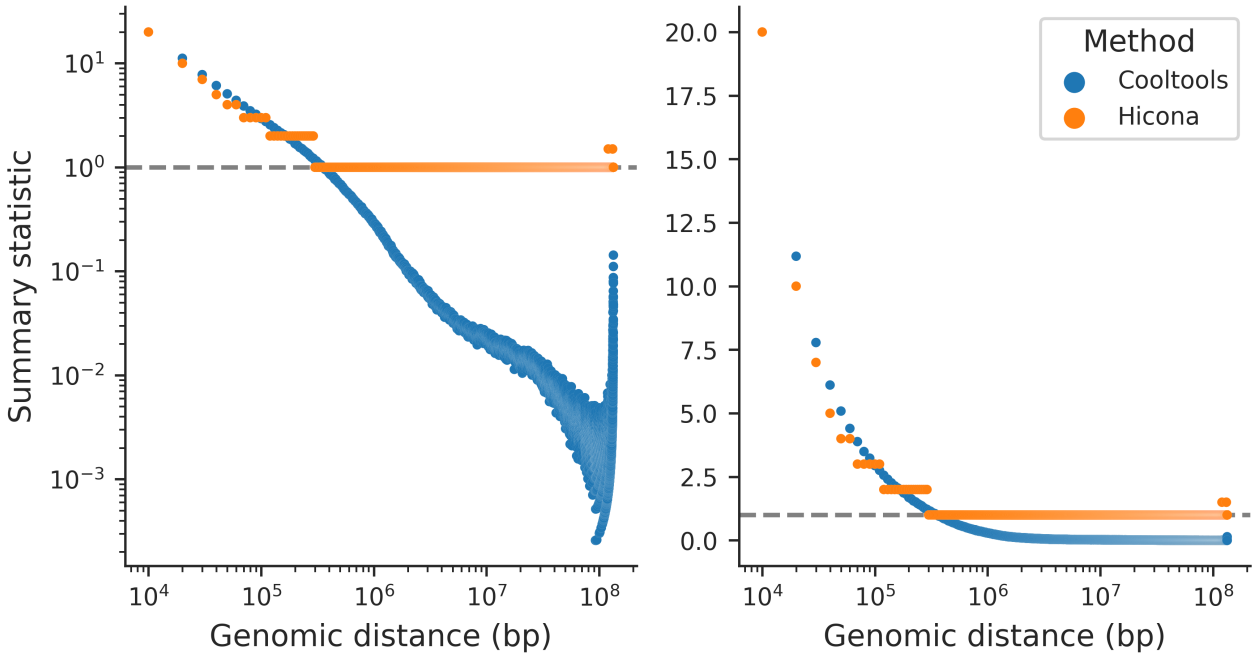


Figure 4.1: Comparison of summary statistics from HiCONA and cooltools. Comparison of the genomic distance normalization factors obtained using Hicona and cooltools on chromosome 1 of one replicate of IMR90 cells (4DNFIMU9T2QI), at 10 kb resolution, filtered using 200 Mb as genomic distance threshold. On the left, normalization factor with respect to distance in log-log scale, on the right the same plot but without log-transform of the y-axis.

given distance to be linear in a log-log plot. As shown in the left panel of figure 4.1, the summary statistic provided by *cooltools* can indeed be considered linear for a big range of genomic distances (up to 10^7 bp); for very high distances though, we start observing some fanning which represents instability due to the low number of pixels averaged. This happens not only because the number of interactions decays with distance, but also because the number of theoretically possible pixels decreases. Visually, increasing distance means moving from the main diagonal of the contact matrix to diagonals closer to the upper right corner, which become shorter and shorter. *Cooltools* allows to address this instability with a smoothing procedure, though it currently requires matrix balancing, which introduces the assumption that all genomic loci form the same number of contacts. In synthesis, the procedure adopted by *cooltools* is biologically sensible, though it has some drawbacks such as very low counts and instability at very high distances.

The summary statistic proposed by HiCONA, instead, is not linear in the log-log plot. At low distances it behaves similarly to *cooltools*, but then it plateaus at 1 (or if a minimum count threshold is imposed, the cutoff itself becomes the plateau value). Though at first glance the two methods seem rather different, this is mostly due to the logarithmic scale of the y-axis. The right panel of figure 4.1 represents the same plot but with the y-axis in linear scale; from this we notice that the values provided by the two methods are actually quite close, since *cooltools* tends asymptotically to 0, while HiCONA is stable at 1 (with some rare exceptions at extremely high distances). Moreover, it must be considered that non-normalized pixel counts are integers strictly greater than zero, therefore the minimum value which they can take is 1; by using a number close to zero as summary statistic, and thus to compute a fold change, a pixel with count 1 at high distance will seem extremely significant, though it might be due to a single random ligation event. The statistic provided by HiCONA is therefore similar to the biologically motivated one provided by *cooltools*, though with the advantage of being more stable and yielding integer values.

4.1.2 Genomic distance normalization

After computing the normalization factors, the pixels are normalized (see subsection 3.3.2 for the exact formula). This means that pixels whose count is equal to the expected one will have normalized value equal to 1; any value above 1 means higher count than the expected, any value below 1 means lower count than the expected.

In figure 4.2 we can observe how the counts distribution changes with normalization. In the top panel, the

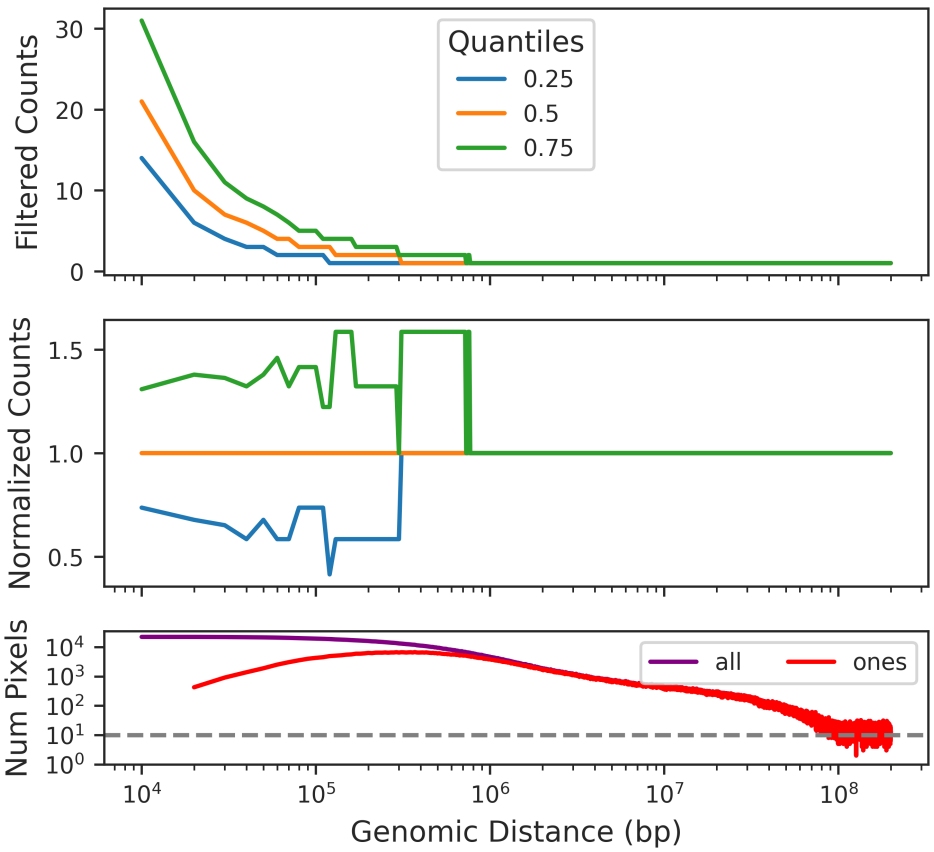


Figure 4.2: Comparison of pixels counts distribution prior and post genomic distance normalization. Comparison of the distribution of raw and normalized counts at each genomic distance for chromosome 1 of one replicate of IMR90 cells (4DNFIMU9T2QI), at 10 kb resolution, filtered using 200 Mb as genomic distance threshold. At the top, quantiles of the raw counts distribution for each genomic distance. In the middle, quantiles of the normalized counts distribution for each genomic distance. At the bottom, total number of pixels, and number of pixels with raw count equal to 1, for each genomic distance.

quantiles of the raw counts distribution for each genomic distance are shown. Though the median of the raw counts distribution can start at different values, in this case 22, it always quickly converges to 1, regardless of chromosome and file. This behavior can also be justified by looking at the bottom panel, which shows the total number of pixels, and the number of pixels with raw count equal to 1, as a function of genomic distance. While the number of pixels quickly decreases with distance, to the point where there are less than 100 at 10^8 bp, the fraction of them with raw count equal to 1 rapidly increases; at 10^6 bp, basically all pixels have raw count equal to 1, thus motivating the median being equal to 1. In the middle panel, the quantiles of the normalized counts distribution for each genomic distance are shown. As one might expect from a normalization process, the value of the median is always 1. Even though the high fraction of raw counts equal to 1 makes it less obvious, on closer inspection it can be seen that the regularization of the spread actually works quite well. Up to $2 \cdot 10^5$ bp, there is a spread in the normalized counts distributions of up to 0.6 units in either direction of the median. Then, from $2 \cdot 10^5$ to $8 \cdot 10^5$ bp, the same spread of 0.6 points is present, though only in the positive direction. This it is due to the fact that, at these distances, more than half of the raw counts are equal to 1, as highlighted by the bottom panel and the median in the top one being 1; since 1 is the minimal value that raw counts can take, it means that there are no pixels who can have a raw count below the expected one, and thus a normalized value lower than 1. After $8 \cdot 10^5$ bp, almost all the pixels have raw count equal to 1, thus all quantiles are equal to 1; for this reason it is still impossible to have values lower than 1, and while normalized values above 1 are possible, they are so rare that they do not alter the position of the quantiles except for very high distances. This is to say that, when there is a sufficient amount on pixels with raw count different from 1 to allow for non overlapping quantiles, the amplitude of the spread falls within a very consistent range of one point in either direction, with few distances exceeding this range (regardless of chromosome, cell type, filtering parameters or file size).

4.1.3 Pixel filtering

The preliminary step of pixel filtering is to remove inter-chromosomal pixels. As explained in subsection 3.3.1, this is because it is not possible to define genomic distance across chromosomes, and therefore these pixels cannot be processed in a comparable way with respect to the intra-chromosomal ones. This step removed, on average, 53.27% of pixels from the files at 10 kb resolution, 45.62% of pixels from those at 5 kb resolution.

All the intra-chromosomal pixels for a single chromosome are then filtered. First, self-looping pixels are removed. Subsequently, pixels with genomic distance above a specified threshold are removed. The partially filtered chromosome-level table is then normalized as discussed in the previous subsection. Finally, the normalized table is filtered using a quantile threshold: the value corresponding to a specified quantile of the normalized counts is computed, then all pixels with normalized counts equal to or below it are removed.

In figure 4.3, some chromosome filtering statistics, aggregated by file, are shown. In each of the panels, the boxes are divided into groups, each one corresponding to a genomic distance threshold, which were sorted from left to right by increasing strictness. Within each group of boxes, each box corresponds to a quantile threshold, once again sorted from left to right in each group by increasing strictness.

In the top panel, the fraction of retained intra-chromosomal pixels after filtering is reported. As expected, both decreasing the distance threshold and increasing the quantile threshold, therefore making the filtering conditions more strict, lead to a decrease in the fraction of retained pixels. While changing distance threshold yields gradual changes in the fraction of retained pixels, changing quantile threshold can cause a quite abrupt drop. This is due to the fact that many pixels, if not most of them, have normalized counts equal to 1. To exemplify this, consider the hypothetical case in which two very distant quantiles, say 0.25 and 0.75, both correspond to a normalized value of 1 due to the sheer number of pixels with that normalized counts value; trying to filter using quantile threshold 0.25 would mean removing all pixels with normalized counts equal to 1, which in turn results in removing all pixels up to the quantile threshold 0.75 since those have normalized counts equal to 1 too, thus causing a drop similar to the one observed in the plot.

In the middle panel, the fold change between the mean genomic distance of the filtered pixels and the mean genomic distance of the non filtered ones is shown. In the bottom panel, the same fold change is represented but using the median instead of the mean. As one might expect, a reduction in both mean and median is observed, which gets progressively bigger as the distance threshold gets smaller. Quantile threshold does not seem to have a big impact on mean and median distance reduction, aside from the drop between 0.05 and 0.1 which is due to the previously mentioned drastic reduction in pixel number. The spread of the mean fold change for a single combination of parameter is quite small, while the spread of the median fold change is quite big; although the spread in median fold change might be due to actual differences in the distribution of genomic distances in the different samples, it is more likely that it is simply due to the way the measure is computed, since it is the ratio among two individual data points (while in the case of the mean, both numerator and denominator are the average of millions of data points).

Since at this step the edges are still being removed ignoring network properties, it makes sense to apply a filter only if the distance metrics are not heavily altered, and if the filter is consistent in its effect. Given these premises, the only parameter values which seem appropriate are 0.0 and 0.01 for quantile threshold and 100 Mb and 50 Mb for the distance threshold. It is worth noting that 0.0 and 0.01 quantile threshold behave almost identically and that 0.0 corresponds to not applying the filter at all; moreover both 100 Mb and 50 Mb are still very permissive thresholds in terms of genomic distance. This indicates that a filtering step which uses these parameters is not doing much, and it could be removed (aside from the removal of self looping pixels and inter-chromsomal ones).

4.1.4 Optimal alpha computation

After pixel filtering, the sparsification scores (or alpha values) are computed for each pixel as discussed in subsection 3.3.3. HiCONA computes and stores both alpha values for each edge, in order to be able to use either the minimum or the maximum of the two. To perform the actual sparsification, a sparsification threshold (or alpha cutoff) must be defined. Although an arbitrary one can be chosen, HiCONA gives the option of computing the so-called optimal alpha value, that is the chromosome-level threshold which minimizes the number of edges while maximizing the number of nodes. This value is computed using a local search algorithm; one example of the grid of values tested by a run of the local search is shown in figure 4.4.

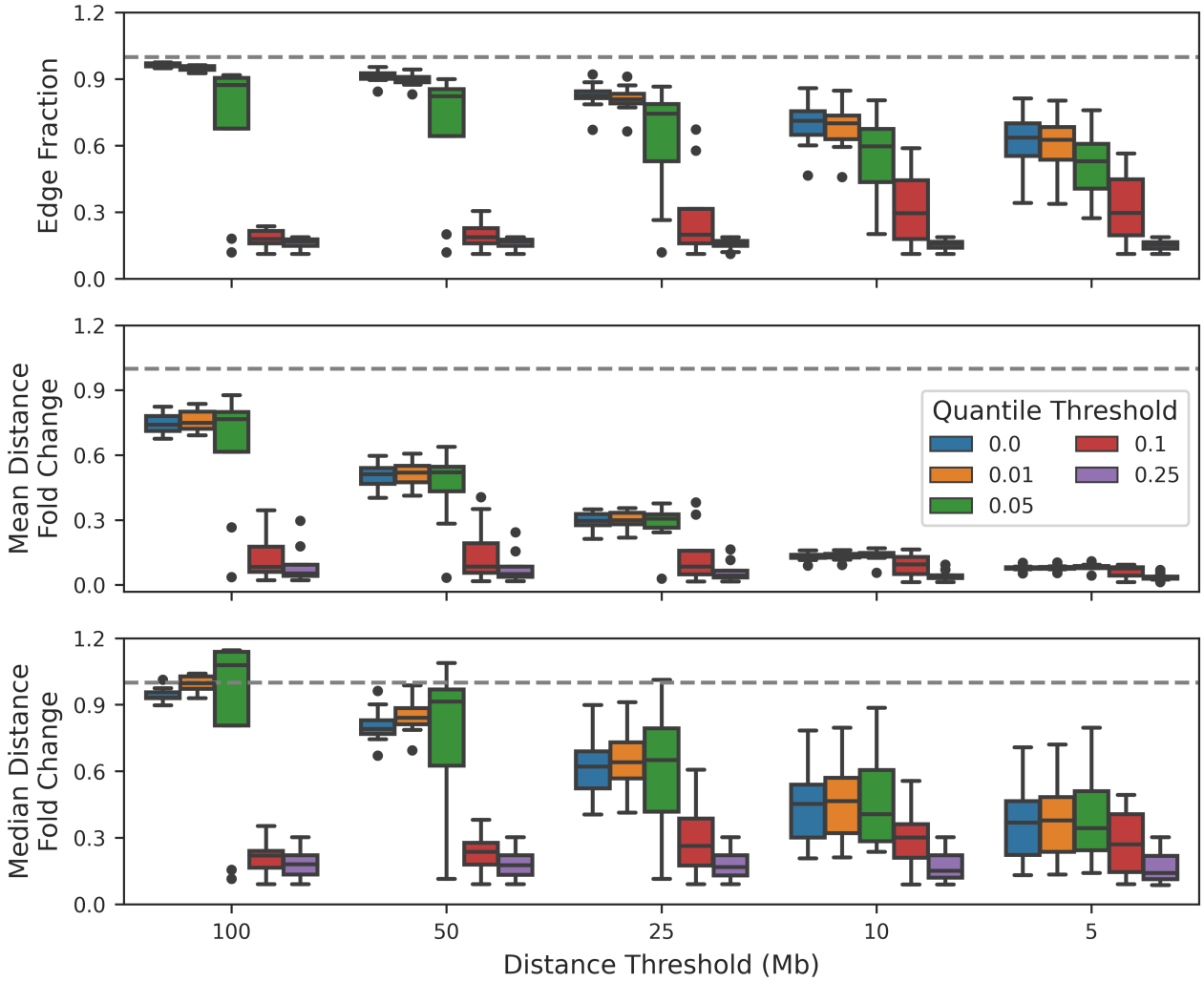


Figure 4.3: Changes in pixel number and distance statistics after filtering. Comparison of the changes in edge number and distance statistics using different filtering parameters on all cool files at 5 kb resolution. Each group of boxes corresponds to one distance threshold, which are sorted from left to right, from least stringent to most stringent. Each box in a group corresponds to one quantile threshold, and within a group, boxes are sorted from left to right, from least stringent to most stringent. At the top, the fraction of intra-chromosomal pixels remaining after filtering. In the middle, fold change between filtered pixels mean genomic distance and intra-chromosomal pixels mean genomic distance. At the bottom, fold change between filtered pixels median genomic distance and intra-chromosomal pixels median genomic distance. The dashed gray line represents the 1 on the y-axis for ease of visualization.

For each chromosome the local search is actually run twice, once using the minimal alphas, once using the maximal ones. For each modality of alpha, a curve is generated from which the corresponding optimal value can be obtained. Typically, the two alpha thresholds for a chromosome are different, though they still tend to be very close to each other (±0.03). The typical range for the optimal alpha threshold is between 0.15 and 0.30. Chromosome Y is an exception; either due to the smaller size with respect to the other chromosomes, or possibly due to some difference in biological properties, the curve is too irregular for a reliable estimate.

4.1.5 Pixel sparsification

After computing the sparsification scores for the filtered pixels and the appropriate alpha thresholds, the actual sparsification step can be conducted. In figure 4.5, some statistics regarding pixel number and genomic distance for files which were sparsified using the optimal alpha values are reported. The only combinations of parameters which are shown are those which were considered appropriate in the filtering step; as was previously noted, those combinations yielded very similar filtering results, therefore it is not surprising that the same holds true for the sparsification step.

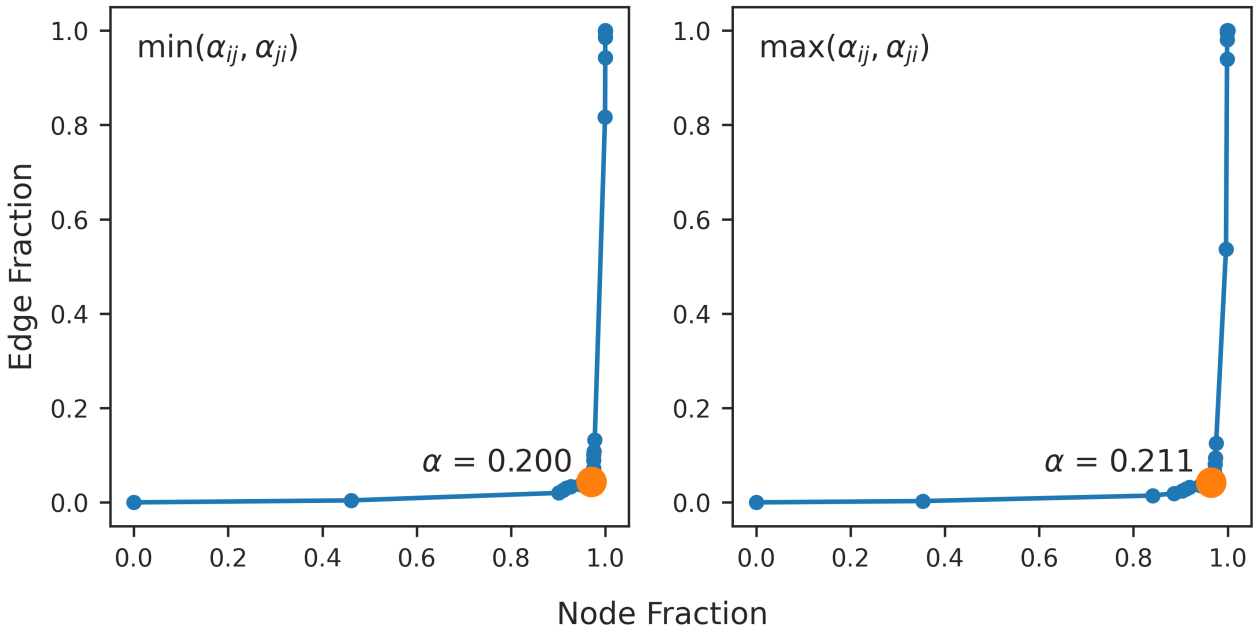


Figure 4.4: Optimal alpha selection process. To define the optimal alpha value to sparsify the chromosome-level network, local search is performed. In the panels, sets of points tested to find the optimal alpha value for chromosome 1 of one replicate of IMR90 cells (4DNFIMU9T2QI) at 10 kb resolution, processed using 200 Mb as distance threshold, 0 as quantile threshold. On the left, local search for the optimal value using maximal alphas. On the right, local search for the optimal value using minimal alphas.

In the top panel, it can be seen that sparsification leads to a drastic reduction in the number of pixels retained; on average, only 5% of the pixels are kept. This holds true for both the minimal alpha and the maximal alpha case, which is partially to be expected considering how similar the thresholds tend to be. In the central panel, the mean genomic distance fold change is shown; a very strong decrease can be observed, which corresponds to the average genomic distance being reduced by 95%, which is constant for both alpha modalities as well as all parameter combinations. Finally, in the bottom panel, the median genomic distance is shown; once again, there is a strong reduction, though less severe, averaging at 80% of the filtered measure, regardless of alpha modality and parameter combination.

It is worth noting that, even though in terms of fold change there is a drastic reduction in the distance statistics, in terms of absolute values, those statistics are still quite high; the median value for the mean genomic distance after sparsification is 590 kb, while for the median it is 400 kb. Moreover, the distances obtained after sparsification, even for the worst file (95 kb), are compatible with the range in which most enhancer-promoter interactions are expected to be found, which are, for most applications, the interactions of interest.

4.2 Pixel preprocessing performance

Most of the operations performed by the package could probably be applied on an entire chromosome at once, given that enough Random Access Memory (RAM) is available to store it and the results in memory; though this might be true, it is a rather brute force method which relies on having access to a fairly high-end machine or a cluster. Given that the package aims to make network analysis of Hi-C data accessible, this is definitely not an option. In order to make the processing manageable even on lower-end laptops, the entirety of it, from filtering to sparsification, is conducted in chunks. By chunking the process, the RAM footprint decreases, allowing it to adapt it to system specifics. While chunking itself does not speed up the computation, it must be noted that a chunked procedure is more amenable to parallelization, therefore it could result in significantly faster runtimes with respect to a non-chunked procedure. In the next subsections, benchmarking on both computational runtime as well as RAM usage are required.

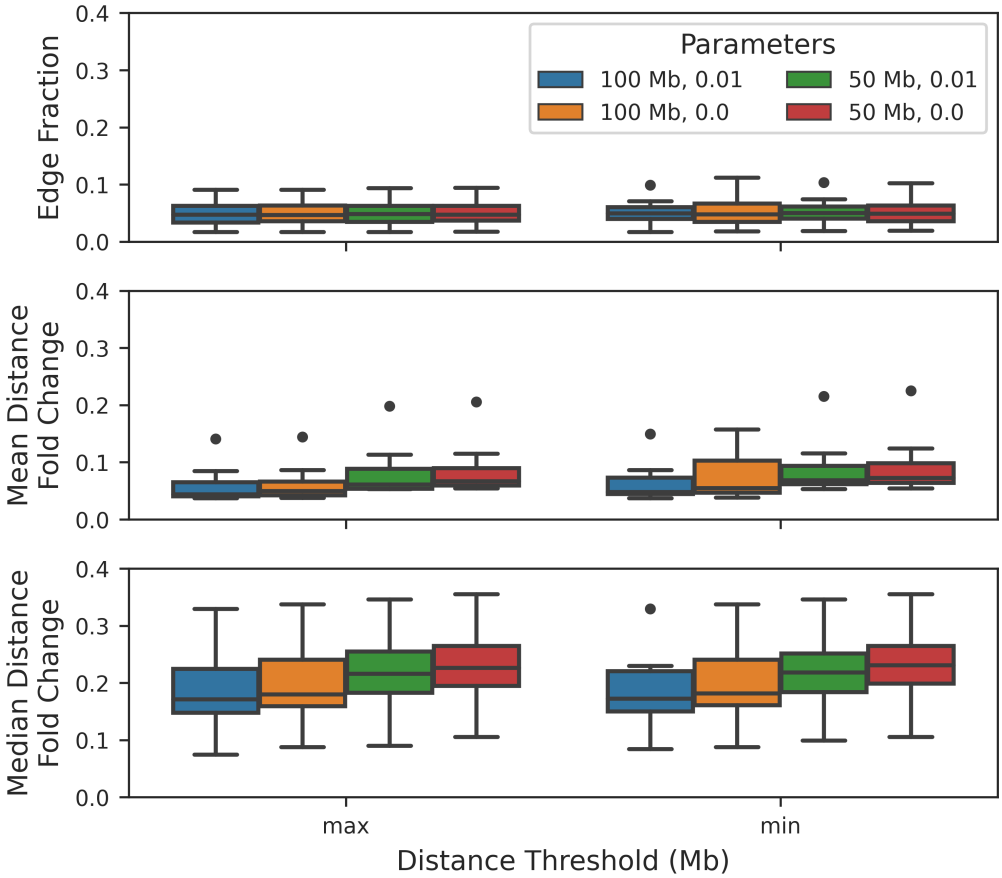


Figure 4.5: Changes in pixel number and distance statistics after sparsification. Comparison of the changes in edge number and distance statistics after sparsification using optimal alpha value threshold on all files at 5 kb resolution. In each panel, the boxplots are divided into two groups, respectively using minimal and maximal alphas. Each group is composed of the boxplots corresponding to the combinations of parameters which were previously deemed appropriate in the filtering subsection. At the top, fraction of chromosomal pixels remaining after sparsification. In the middle, fold change between sparsified pixels mean genomic distance and filtered pixels mean genomic distance. At the bottom, fold change between sparsified pixels median genomic distance and filtered pixels median genomic distance.

4.2.1 Time benchmark

Time benchmarking was performed by analyzing the wall-time required to process each file using a certain set of parameters. The sparsification scores computation step is by far the most computationally intensive step, to the point where the time required by the other operations is basically negligible; in fact, as shown in figure 4.6A, the time required for the file to be processed correlates with the number of pixels remaining after the filtering step. From the sparsification algorithm we expect the relation to be linear (which is indeed the case), since the number of integrals to be computed is twice the number of pixels remaining after filtering. Still, it must be considered that, although the normalized counts are floating point numbers, they are obtained by taking the log2 of a fraction whose terms are integers. Moreover, the values which the denominator can take are rather limited, very rarely exceeding 100. For this reason, the pool of normalized values is smaller than it might seem which, combined with the fact that most integrals are repeated, makes it possible to drastically reduce the number of integrals computed. Since each integral only depends on the number of neighbors and the normalized edge weight (see 3.3.3), a `pandas.DataFrame.groupby()` on those two columns was performed in order to obtain the set of unique combinations; then, after computing the integral for each combination, the values are associated back to the pixels by using a *left-outer-join* merge. Though the same integral is computed multiple times if it appears in multiple chunks of the table being processed, this way it is possible to compute, on average, only 65,000 integrals per million of filtered pixels, maintaining linearity while reducing the slope.

From figure 4.6A, it is also possible to observe the time it took for the each file to be processed (using the 4 combinations of parameters which were considered viable); standard-sized files, such as H1hESC, HUVEC

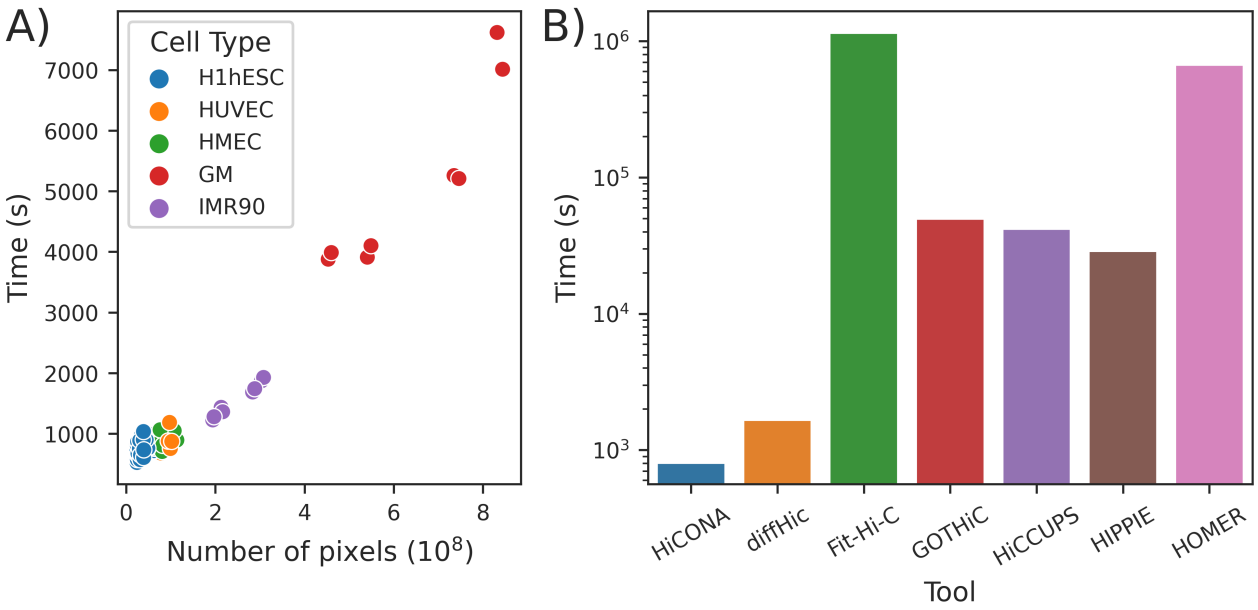


Figure 4.6: Pixel preprocessing time benchmark. (a) Time required to compute the sparsification scores for each file, with different parameter combinations, as a function of the number of pixels remaining after filtering. As one might expect, there appears to be a linear correlation between number of pixels to process and time required. (b) Comparison of the time required by HiCONA and common loop-callers to process an IGM90 cells replicate file (4DN data portal id: 4DNFIAAUZ7EP). Runtimes for the loop-callers were obtained from Forcato et al., 2017. The file was processed by HiCONA using a distance threshold of 100 Mb and a quantile threshold equal to 0.01. HiCONA is much faster than all the loop-callers.

and HMEC, are processed in roughly 30 minutes, while GM and IMR90, which are usually used as examples of very big files for Hi-C benchmarking, are still processed in little over two hours at most.

In figure 4.6B, the speed of HiCONA was compared to the most common loop-callers. One of the replicates of IMR90 cells (4DN data portal id: 4DNFIAAUZ7EP), was processed with the standard pipeline for each tool. For the loop-callers, the time to perform normalization and analysis was obtained from Forcato et al., 2017[59]. The file was processed by HiCONA using a distance threshold of 100 Mb and a quantile threshold equal to 0.01. For HiCONA, the initialization and filtering steps are included too since they have negligible runtime with respect to sparsification. HiCONA resulted being much faster than all loop-callers, with diffHic being the only one with a comparable runtime.

4.2.2 Memory benchmark

RAM usage was tested only on a subset of the files since `memory_profiler`, the package which was used, computes memory usage by pausing script execution every 0.1 seconds. In figure 4.7, RAM usage overtime for the preprocessing of all chromosomes of a single file, using one set of parameters, is shown. In the figure, every pair of dashed red lines corresponds to the interval in which the sparsification scores for a chromosome are computed. It is thus worth noting that the time spent for operations other than sparsification scores computation corresponds exclusively to the interval before the first red line and the interval after the last one; as mentioned in the previous subsection, this amount of time is basically negligible compared to the one used for sparsification.

It can be seen that the program initially loads some objects into memory, reaching around 400 MiB of RAM, then sparsification causes fluctuations of around 200 MiB, for a maximum RAM usage of less than 600 MiB. The memory load prior to sparsification can vary slightly depending on file properties, while the amplitude of the fluctuations during sparsification is constant, since it uniquely depends on the chunk size used during processing (by default, 1 million pixels per chunk). In fact, each peak corresponds to a chunk being loaded into memory. Therefore, HiCONA does indeed allow to process any file using a very low, fixed, amount of RAM.

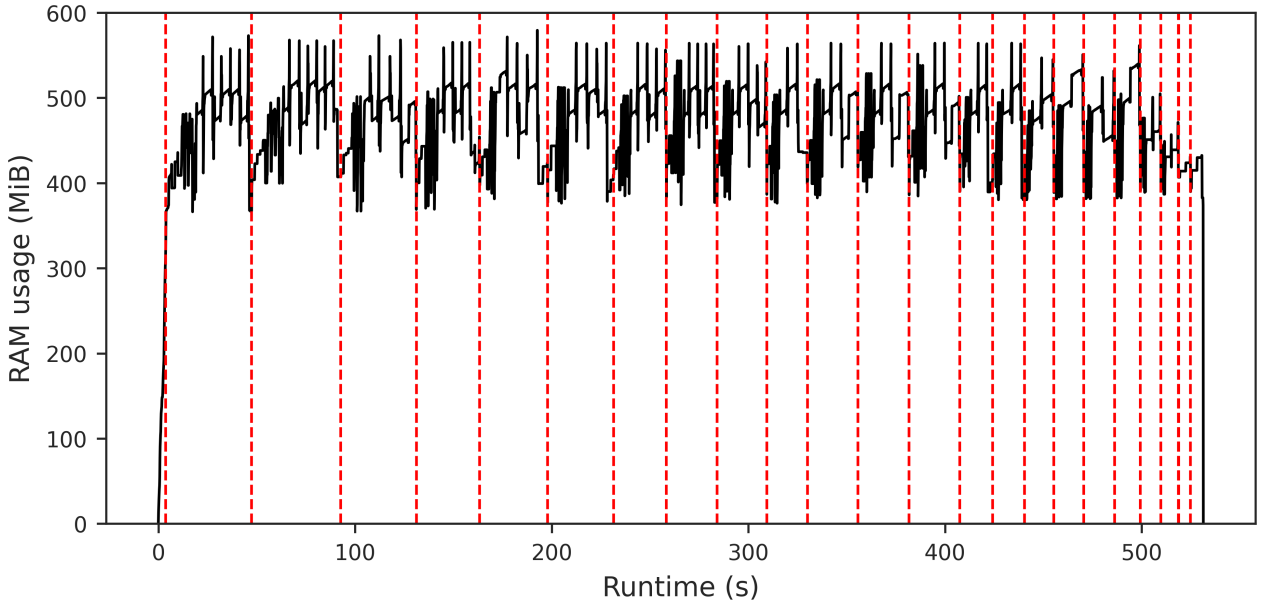


Figure 4.7: Preprocessing RAM usage overtime. Plot of Random Access Memory usage during the preprocessing procedure. Runtime is expressed in seconds, while memory usage is expressed in MebiBytes (1 MiB = 2^{20} bytes). The area included between two red dashed lines corresponds to the sparsification scores computation for a chromosome. In the figure, the RAM usage overtime for IMR90 cells at 10 kb resolution, processed using 200 Mb as distance threshold, 0 as quantile threshold, is shown.

4.3 Pixel preprocessing biological validation

Even though the procedure might work for network analysis purposes, that does not mean that the procedure is able to yield significant biological results. Some form of validation is needed, to show that the networks have features which are amenable to perform biologically meaningful analyses. One feature was already discussed, that being the average pixel genomic distance, which was shown to be compatible with the analysis of promoter-enhancer interactions. In the following subsections another form of validation will be discussed, which is the consistency of the procedure on biological and technical replicates.

4.3.1 Consistency on replicates

Ideally, data generated from replicates should have a high degree of concordance if the biological procedure is consistent and stable enough; this concordance should be maintained, if not increased, by the following processing steps in order to obtain robust and generalizable results. To analyze this concordance, a measure of similarity must be introduced; considering the different sizes of the networks, a good choice is the Jaccard index, which is defined as the cardinality of the intersection of the two networks over the cardinality of their union (i.e. the number of shared edges over the number of all edges, regardless of their weights). All technical and biological replicates from both IMR90 and GM cells at 5 kb resolution were processed using different combinations of parameters; the Jaccard indexes were computed just after the filtering step and after the sparsification procedure.

For GM cells, the Jaccard indexes among filtered replicates spanned the range 0.07-0.17, while those for the sparsified replicates (regardless of minimal or maximal optimal alpha), spanned the range 0.055-0.15. No pattern was found in the distribution of the Jaccard indexes, even considering different filtering parameters. For IMR90 cells, the Jaccard indexes among filtered replicates spanned the range 0.055-0.17, while for the sparsified replicates (again, regardless of alpha) the range was 0.035-0.13. In this case a loose bimodal distribution was visible, with Jaccard indexes being close to either of the range limits. Though there is no real metric to define what a good Jaccard index among replicates should be in this situation, the values seem fairly low. As previously described though, the used thresholds are very loose, to the point were the datasets are very similar to the raw contact matrices without self-looping pixels and inter-chromosomal pixels; this means that, for what concerns intra-chromosomal pixels, the Jaccard indexes of the raw file themselves are likely low, suggesting that the consistency is low due to either the biological procedure or the steps to create the contact matrix.

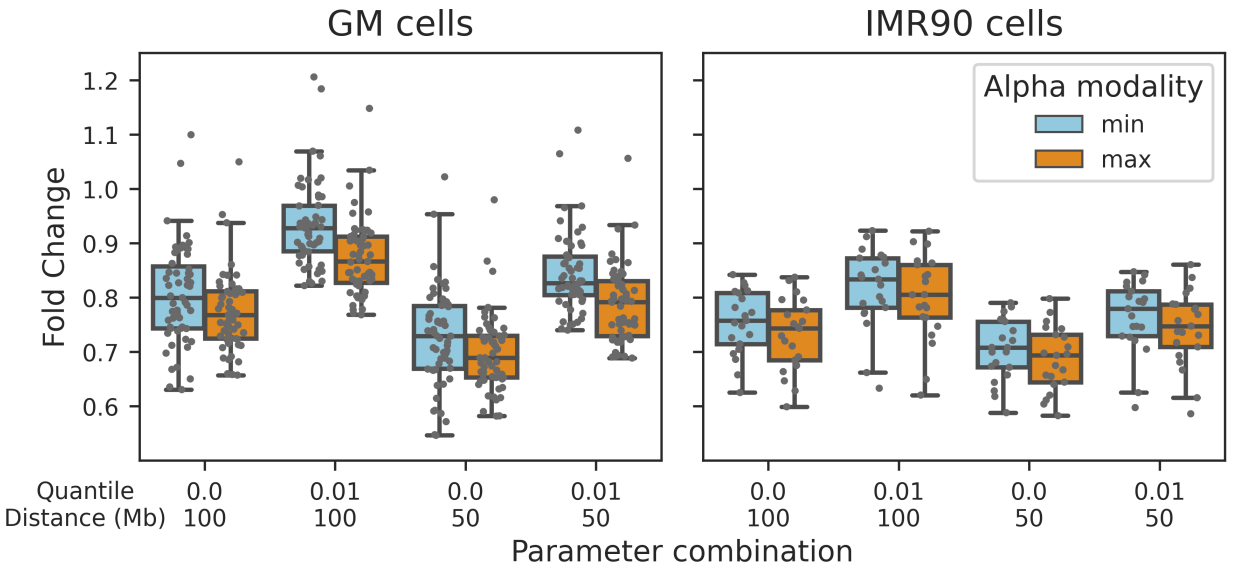


Figure 4.8: Jaccard index fold change. Fold changes between Jaccard index after sparsification and Jaccard index after normalization. The distribution of the fold changes is shown for each combination of parameters used during processing, and for both alpha modalities. In the left panel the fold changes for GM cells, in the right panel those for IMR90 cells. The combination of parameters with distance threshold equal to 100 Mb and percentile threshold equal to 0.01 is the one consistently yielding higher fold changes (and thus higher similarity).

Having asserted that the starting Jaccard indexes are fairly low, the pressing point for the sparsification procedure becomes to try and preserve, or ideally increase, these Jaccard values. To analyze this aspect, the fold changes between Jaccard indexes computed on the sparsified files and those computed on the filtered files are shown in figure 4.8. To test whether any of the combinations of parameters yields, on average, a different fold change with respect to the others, Friedman test was performed. Friedman test is a non-parametric test for the comparison of multiple groups of paired data. For IMR90 cells, the p-value of the test is in the order of 10^{-12} for both minimal and maximal alphas; for GM cells, the p-values for minimal and maximal alphas were in the order of 10^{-33} and 10^{-32}, respectively. Considering that all p-values were significant by a long margin, post-hoc analysis was performed to define which sets of parameters actually have different means. To do so, Wilcoxon Signed-Rank test, a non-parametric test to compare two groups of paired data, was used; the p-values were then corrected using Bonferroni correction (which in this case meant multiplying by 24, since 6 pairs of groups are tested for each of the 4 conditions). For all conditions, all pairs of combinations of parameters resulted significantly different, aside from the pair composed of the combinations "distance threshold equal to 100 Mb, quantile threshold equal to 0" and "distance threshold 50 Mb, quantile threshold equal to 0.01", which were not significant for IMR90 cells.

From these results we derive that the combination of distance threshold equal to 100 Mb and quantile threshold equal to 0.1 is the one leading to the smallest reductions in Jaccard index during the sparsification procedure. In fact, in IMR90 cells, this threshold yielded an average fold change of 0.805 using maximal alphas, 0.834 using minimal alphas; for GM cells the thresholds for maximal and minimal alphas were 0.866 and 0.928, respectively. While there is still a fairly wide spread of points even for the best combination of parameters, it is worth noting that some replicates can have up to two orders of magnitude of pixels less than others, in which case, it seems hard for any set of parameters to close the gap. Overall, it seems that the sparsification procedure, given the right parameters, is able to process replicates with a high degree of similarity.

In figure 4.9, the consistency of HiCONA was compared to the consistency of some common loop-callers. This was done by comparing their Jaccard indexes on replicates of IMR90 cells and GM12878 cells. All replicates were processed using the suggested procedure for each loop-caller; the data was obtained from Forcato et al., 2017. The files were processed by HiCONA using a distance threshold of 100 Mb and a quantile threshold equal to 0.01, since it is the combination of parameters which seems to perform the best in terms of Jaccard fold change. It can be seen that the various loop-callers have quite different Jaccard indexes, though all of them are quite low. Although HiCONA did not produce the highest values, it is the tool which is consistently

higher than all others, regardless of whether minimal or maximal alpha was used.

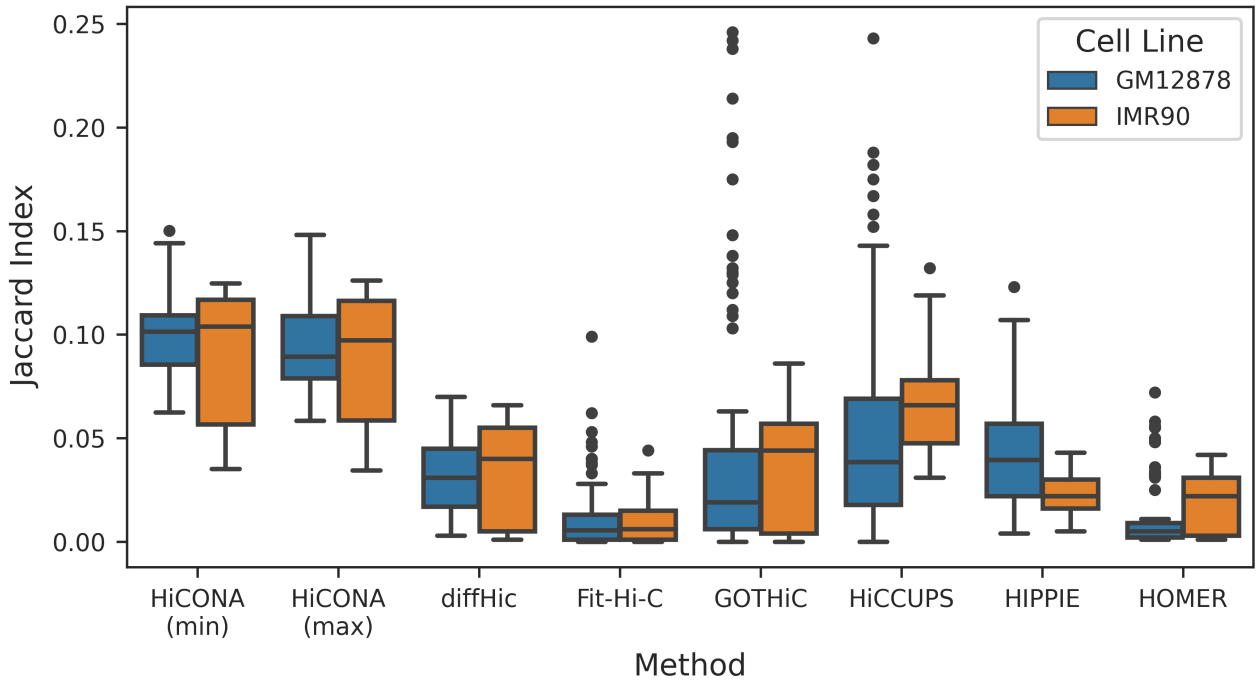


Figure 4.9: Comparison of Jaccard indexes from HiCONA and loop-callers. Jaccard indexes on replicates of IMR90 cells and GM12878 cells were computed for both HiCONA and common loop-callers. All files were processed using the suggested procedure for each loop-caller. For HiCONA, distance threshold of 100 Mb and quantile threshold equal to 0.01 were used. HiCONA seems to be consistently higher than all other tools. Data for the loop-callers from Forcato et al., 2017.

Chapter 5: Discussion

In this chapter, the results shown in the previous section are discussed, including possible solutions to improve performance of the package. Some other preliminary results, which were not included due to them being unrelated to the development of the preprocessing section of HiCONA, are also discussed. Again, these analyses were conducted by my tutor, Leonardo Morelli, and are not part of the functionalities implemented by HiCONA; they are though useful for the validation of the performance of HiCONA, or they are exploratory analyses to define what could be implemented into HiCONA. Finally the current state of the package, what is currently available and what could be done in the near future, are all presented.

5.1 Pixel preprocessing

In this section, the main steps of the pixel preprocessing procedure are discussed individually. Particular attention is given to the reason why certain choices were made, which are the limitations of these steps and how these steps could be improved upon.

5.1.1 Sequencing bias normalization

One concern that might be raised regarding the preprocessing pipeline adopted by HiCONA is the lack of a sequencing bias normalization step. There are two main reasons for that. First, sequencing biases are expected to affect the pixel counts way less than genomic distance, to the point where it is not unreasonable to assume that the networks obtained with or without sequencing bias normalization should behave almost identically. As mentioned in subsection 1.3.4, sequencing bias normalization methods assume the bins to be homogeneous in regards to the feature they are normalizing for. This is quite an approximation, considering that even at a 10 kb resolution, which is on the lower end of viable resolutions, the bins are still quite big and heterogeneous, to the point where it is arguable whether the entire normalization actually makes sense or not. These considerations, coupled with the fact that it is unclear which bias normalization algorithms are actually reliable and the fact that integer counts have substantially better properties for computational purposes, led to the choice of not including sequencing bias normalization in the standard package pipeline. Nevertheless, HiCONA is fully capable of working with sequencing bias normalized counts, though in a less efficient manner considering it was optimized for non-normalized ones.

5.1.2 Genomic distance normalization

The way HiCONA normalizes for genomic distance was shown to be reliable and consistent, being unstable only at extremely high genomic distances. It must be remarked that, using the current algorithm, if a pixel with high count is the only one for that genomic distance, it is assigned the normalized value of 1. This can reasonably happen only at very high genomic distances (80 Mb or more), where the number of pixels per genomic distance is very low, and one individual pixel might have a very high count value causing a spike in the expected counts curve. Though it might seem strange to normalize to 1 a pixel with such a high value with respect to its neighboring distances, at genomic distances that high, the actual biological significance of those spikes starts to become debatable, with them being random ligation products or PCR artifacts becoming the more likely option. Losing a few potentially meaningful pixels at these genomic distances does not therefore seem like a major issue.

In regards to normalization factors computation, it should be noted that, regardless of the usage of *cooltools* or HiCONA, one summary statistic function is computed for each individual chromosome. *Cooltools* does, in fact, go even beyond this, allowing to remove telomeres and centromere, then computing one normalization curve for each arm of the chromosome. The use of one curve per chromosome is to avoid relying on the assumption that random contact probability decays at the same rate in all chromosomes. One might argue that

considering all pixels jointly would yield a more robust and less noisy function; while that would indeed be the case, each chromosome has enough pixels to obtain a robust result on its own (aside from chromosome Y due to its significantly smaller size and probably other biological motivations).

5.1.3 Pixel filtering

Pixel filtering led to unexpected results. The step was introduced to both remove pixels which are definitely to be discarded for sparsification to work properly (self-looping and inter-chromosomal ones), as well as to reduce a bit noise and dimensionality prior to sparsification. This last aspect was especially relevant when the tool did not work in chunks and thus table size was still an issue. Though raw counts threshold is probably not needed anymore and was, in fact, not included in the analyses presented, different combinations of genomic distance threshold and normalized counts quantile threshold were tested. Ideally, the distance threshold would be set to the maximal distance for an interaction to be considered biologically relevant, but such threshold has not been clearly defined yet. For the quantile threshold, defining a value is even more complicated, since it is not tied to any biological meaning. It follows that the only real way to choose these parameters is trial and error using educated guesses.

In terms of filtering itself, the results seem middling; any filtering threshold which provides a significant reduction in pixel number also causes a massive reduction in genomic distance which does not seem to be justifiable prior to network sparsification, making very loose thresholds the only viable option. The need for a filtering step becomes even more debatable when considering that the combination of filtering parameters used does not appear to significantly affect sparsification, if at all. Yet, the choice of filtering parameters affects the consistency of replicates quite a lot, with fold changes which are higher and statistically significant. Further and more systematic testing of parameter combinations could probably lead to even better consistency, though it would be quite demanding in terms of time.

5.1.4 Network sparsification

Regarding network sparsification, a standardized procedure to decide an alpha cutoff was proposed and used throughout all the analyses, removing the need to define arbitrary values, which becomes especially complex when considering that different thresholds are required for different chromosomes. As mentioned, being sparsification scores actual p-values, the usual p-value cutoff of 0.05 seems like a pretty natural choice and it was, in fact, tested as a sparsification score threshold. This value proved far too restrictive with any combination of filtering parameters, to the point where just a couple hundreds of pixels are retained, at most, for each file. This value is therefore not appropriate for this application, and in fact, even the original paper which introduced the algorithm proposed the use of looser thresholds. It must be remarked once more that the objective of this procedure is not to determine the definitely significant pixels themselves, but rather sparsify the network by removing the least significant ones to achieve a network of a manageable size (while still maintaining disconnected components of a reasonable size).

Both minimal alphas, for a more permissive network, as well as maximal alphas, for a more theoretically robust one, were tested. The choice does not change sparsification results in an appreciable manner, while minimal alphas seem to marginally outperform maximal ones in regards of consistency. For this reason, minimal alphas will probably be the standard option for the package going forward. It must be also stated that the use of minimal alphas seems more appropriate even on a purely theoretical level, since it is more in line with the original network sparsification algorithm in trying not to belittle smaller nodes.

The aspect which could be considered the big criticality of the sparsification algorithm is how much the median genomic distance is reduced. While it is fair to assume that there are more significant interactions at closer ranges, and thus that sparsification will inevitably reduce median genomic distance, the reduction feels quite heavy. Again, this is not unexpected, considering that the algorithm does not keep genomic distance into account while defining edge importance; one option could be to try and include the distance information in the algorithm, though it seems rather complex and demanding.

5.2 Computational performance

The current implementation of the sparsification algorithm is already quite fast, though many aspects could be tweaked for an even greater speed-up. Regarding integral computation, for instance, the cache of integrals could persist among chunks; this is currently not done since there is a fairly big amount of integrals that are still used only once throughout the entire chromosome sparsification, and keeping all of them in cache needlessly would increase its size drastically. Another option would be to precompute the most common integrals and store them in a tabular file, ready to be loaded as cache when needed. This would lead to a fairly big overhead during the first usage of the package, but could speed-up drastically successive runs. Another aspect which could be optimized are operations on *pandas* dataframes; though vectorized operations are fast, they could be improved by relying on faster structures, such as *polars*[60] dataframes, which work similarly but they are implemented in Rust, making them substantially faster. Dataframe operations could also be sped up using *swifter*[61], which allows to split a dataframe in chunks and work in parallel on them. On this topic, it should be mentioned that HiCONA, as of now, does not perform any form of parallelization explicitly, though some operations are implicitly parallelized by *cooler* using *dask*[62]. A pretty substantial speed-up could be obtained by processing the chromosomes in parallel, considering the fact that they are independent from each other. That being said, the library *h5py*[63], used to work with `.hdf5` files and on which *cooler* is based, is implemented with several file locks to prevent corrupting the files while working on them, making parallel writing rather difficult. Overall, considering that the preprocessing times are already quite fast, speeding up these aspects is not currently marked as a top priority.

In terms of RAM usage, the preprocessing was shown to be very efficient, requiring very low amounts of memory. It might be debated that chunk size could be increased, to reduce the number of I/O operations therefore speeding up the procedure while still keeping a relatively low memory footprint. Surprisingly, this is not the case; by increasing chunk size the time per million of pixels to process actually increases. Though no definitive reason was found, this is most likely due to a poor scaling of the `merge` function of *pandas*, which is used to associate the computed integral values back to the pixels. In any case, a better choice would be to keep the amount of memory of the individual process low, then run multiple of them in parallel on different chromosomes as discussed in the previous paragraph.

5.3 Biological validation

In the results, biological validation through replicate consistency was analyzed. Overall, the pipeline does not seem to have the highest Jaccard indexes in terms of absolute values, though the sparsification step is very good at retaining the correlation present in the filtered pixels, especially given the right filtering parameters. When considering that the correlation of the filtered files is already very low, it does appear that the issue with low correlation might lie in the biological experiments themselves rather than in the processing pipeline. This would be in line with the fact that the matrix obtained through a single Hi-C experiment is very sparse, therefore requiring multiple experiments to be merged to obtain a denser matrix. The ideal pipeline would be able to increase the correlation among replicates, since this would mean that similar information could be obtained by sequencing less, thus reducing experimental costs. HiCONA does not seem to be at that stage yet.

Another form of biological validation which was performed, but not shown in the results since not strictly tied to the actual implementation, regards functional annotation enrichment. For each bin, it is possible to define which functional element (promoter, enhancer, heterochromatin and others) is the most enriched with respect to the abundance of these functional elements in the genome. Then, it is possible to define the fraction of interactions which occur among any pair of functional elements. By comparing these fractions among different processing steps, such as between total pixels and sparsified pixels, it is possible to define which type of interactions are enriched by the procedure. Ideally, one would like for biologically relevant annotations, such as promoters and enhancers, to be enriched by the step. The annotation enrichment was computed on the individual replicates of IMR90 cells and GM12878 cells at 5 kb resolution, both on the total pixels as well as on the pixels sparsified using distance threshold equal to 100 Mb and quantile threshold equal to 0.01. The annotations used were obtained from the paper of Forcato et al., 2017[59], which grouped annotations generated using the tool ChromHMM[64]. For each cell line, the enrichments at each step were averaged, then the $\log_2$ fold change among sparsified and total pixels was determined. This analysis yielded promising results, with a substantial

enrichment of promoter, enhancer and zinc finger binding annotations in the sparsified files. This suggests that the sparsification step is able to enrich for pixels which are more likely to be biologically meaningful.

5.4 Network analysis algorithms

In HiCONA, a `HiconaGraph` object is already implemented. Since it inherits from the main *graph-tool* class, it is able to already perform quite a vast set of operations, such as computing network statistics and centrality measures, performing clustering and complex plotting (but the list is much longer). To these functionalities, an *ad hoc* version of node label permutations was added. The algorithm does work properly on mock networks, while it does not seem to work on real Hi-C derived ones. Though this could simply be that there are, in fact, no differences in properties for nodes annotated with different biological functions, this could also be due to how the node annotation is performed. Currently, bins are annotated by intersecting them with a bed-like file and assigning to them the label of any annotation which has any amount of overlap with them. It must be considered though that bins of 5 kb in size are quite big and thus many bins end up being annotated with a lot of labels. This huge overlap of annotations is what could be currently preventing the detection of any significant differences.

Contrast subgraphs extraction is another *ad hoc* analysis algorithm which was planned to be included into HiCONA. Contrast subgraphs extraction is a recently introduced network analysis algorithm for the comparison of networks. Since it has found some success in different biological applications, such as brain network classification as well as comparison of omics data-derived networks[65;66], it seems plausible that it could yield interesting results when applied to the analysis of Hi-C data-derived networks. The algorithm can be roughly summarized as follows. The inputs are two weighted undirected networks defined over the same set of nodes and some tuning parameter C. The edge weights are scaled to the range $(0, 1]$, then one graph is subtracted edgewise from the other. The heuristic peeling process follows, which is a greedy solution for the densest subgraph problem on networks with negatively weighted edges. Basically, at each step of this iterative procedure, the edge contributing the least to overall graph sum of weights is removed; the parameter C determines how much positive weights are favored with respect to negative ones. In synthesis, the results are one or more contrast subgraphs, which are subgraphs recapitulating the major differences among the two input networks (in a non-symmetric manner). This procedure was tested on sparsified networks obtained using HiCONA, though it does not appear to provide any insightful results. The procedure seems to simply return the difference of the two graphs, and this is likely due to the fact that the sparsified graphs do not have enough overlap. Still, contrast subgraphs might still be considered provided that some major modification of the algorithm is performed.

5.5 Package development

The package was implemented for Python version 3.8 and later, but it should work even for previous versions provided that there is a release of the dependencies for it (though it was not checked since it does not seem particularly relevant). The code is styled using *Black*[67], in order for the code formatting to be consistent and compliant with PEP style guidelines[68]. Linting is performed using *Pyflakes*[69]. All public functions and methods of the package are fully documented and type hinted using the NumPy docstring conventions. For this reason, documentation using *Sphinx*[70] is being created and will become the *Read The Docs* page once the package is made available on *PyPI* and *Conda*. To further support package documentation, *Jupyter*[71] notebooks will be created to illustrate standard pipelines. Moreover, now that the package has a partially stable structure, tests are being implemented using *pytest*[72] to simplify the addition of new features and collaborative work, especially with an open-source model in mind.

As of the time of writing, the code for the preprocessing procedure, therefore the steps from the .cool file to the generation of annotated networks, is almost deployment-ready. Some minor optimizations and refactoring are still required, though no functionality is missing. For what concerns network analysis, the node-labels permutation algorithm is fully implemented and working, though whether it is able to produce meaningful results is still to be verified. Contrast subgraph extraction is currently not included, though it might be in the future if a change in the algorithm or procedure is found to make it work with sparsified networks. Other functionalities will be added overtime when the need for them arises.

In figure 5.1, a roadmap summarizing the current state of HiCONA and its future, at least for a few months to come, is presented.

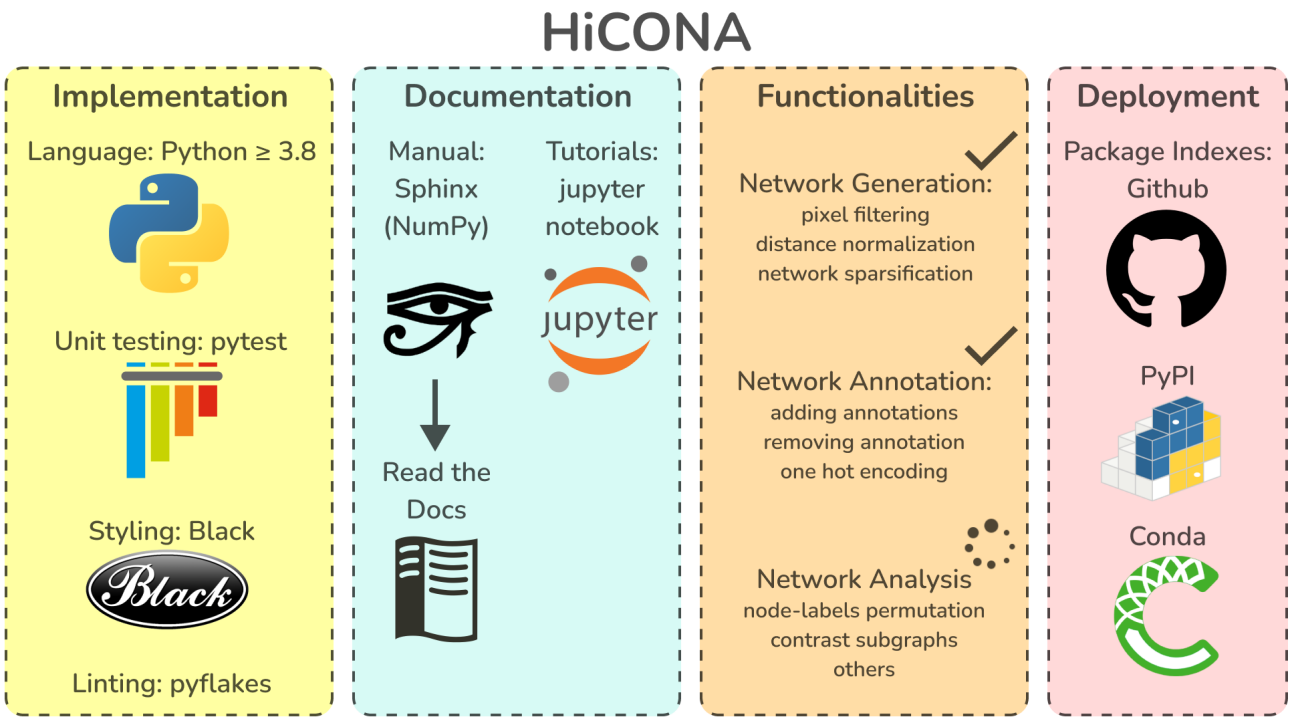


Figure 5.1: Current state and future of HiCONA. Main aspects regarding implementation, documentation, deployment and functionalities of HiCONA. Package documentation and deployment are still being worked on, though they are the next major step in the package development.

Chapter 6: Conclusions

This thesis introduces HiCONA which is a Python3 package aimed at providing a flexible, user-friendly and efficient framework for the network analysis of Hi-C data. This package was created because, despite the potential insights that this type of analysis could provide, there are currently no options to perform versatile and comprehensive network analysis on Hi-C data. The main reason for this lack of tools is the nature of the data obtained from Hi-C experiments; though the conversion of this data into graphs is almost trivial, the results are high-order and very dense networks, sometimes too big to even fit into memory, which is rather difficult to work with.

To address this need, the package provides multiple functionalities, which can be divided into network generation, network annotation and network analysis. Network generation includes all of those steps allowing to convert a .cool file, a standard format for the storage of Hi-C data, into multiple chromosome-level networks, with nodes representing genomic regions and edges representing the contacts among them. Network annotation comprehends functions to add, remove and manipulate bin annotations. Network analysis provides both general network analysis algorithms, by interfacing with the *graph-tool* library, as well as custom implementations of more specific algorithms.

This work focused on network generation and all its sub-steps, those being pixel filtering, genomic distance normalization and network sparsification. Pixel filtering was not found to be effective in reducing pixel number, since only loose thresholds can be applied; still, while pixel filtering is almost irrelevant in terms of sparsification results, it seems to be quite impactful for the consistency of replicates. In regards to genomic distance normalization, a very simple yet stable method was introduced which provides normalized distance values that are within a very consistent range and amenable to optimization via memoization. Network sparsification is the crucial step in dealing with the issue of graph density, allowing to reduce the number of edges while trying to preserve the topological properties of the graph, therefore extracting the backbone of the network. First, a consistent method to determine the sparsification score threshold to use was defined. Then, it was shown that the technique is able to reduce substantially the number of pixels, while maintaining most of the correlation among replicates. Moreover, despite the reduction of the average pixel genomic distance after sparsification, its value remains high enough to suggest that the distribution of the pixel genomic distances continues to cover the range of distances in which promoter-enhancer interactions are expected to be found. Finally, sparsification was found to enrich for pixels whose bins are annotated as functional elements.

From a computational efficiency point of view, it was shown that HiCONA does indeed provide very fast processing with minimal RAM footprint, so that even lower end systems can analyze basically any .cool file. This is achieved through efficient chunking and vectorization of the operations. Another major speed up could be attained using parallelization, which is one of the main objectives for the future of the package.

HiCONA is still currently in the development stage. While network generation and annotation are basically deployment ready, some work is needed for the network analysis part. The node-labels permutation algorithm is already implemented, though some tweaking is required to extract biologically relevant information from it. Contrast subgraphs extraction is another candidate algorithm for implementation, though it would need some substantial changes to fit Hi-C data.

The objective is for HiCONA to become a full-fledged package available on all major package indexes (Github, PyPI and Conda), with extensive documentation and tutorials. HiCONA was made to be easy to slot in any already existing pipeline, providing new analysis options, as well as to be easily extended with new functionalities, in order to accommodate for any new need which might arise. Hopefully HiCONA will provide an opportunity to learn more from the still very active research field of chromatin biology.

References

[1] Guohong Li and Ping Zhu. Structure and organization of chromatin fiber in the nucleus. *FEBS Letters*, 589 (20PartA):2893–2904, apr 2015. doi: 10.1016/j.febslet.2015.04.023. URL https://doi.org/10.1016%2Fj.febslet.2015.04.023.

[2] Rieke Kempfer and Ana Pombo. Methods for mapping 3d chromosome architecture. *Nature Reviews Genetics*, 21(4):207–226, dec 2019. doi: 10.1038/s41576-019-0195-2. URL https://doi.org/10.1038%2Fs41576-019-0195-2.

[3] Hui Zheng and Wei Xie. The role of 3d genome organization in development and cell differentiation. *Nature Reviews Molecular Cell Biology*, 20(9):535–550, jun 2019. doi: 10.1038/s41580-019-0132-4. URL https://doi.org/10.1038%2Fs41580-019-0132-4.

[4] Karolin Luger, Armin W. Mäder, Robin K. Richmond, David F. Sargent, and Timothy J. Richmond. Crystal structure of the nucleosome core particle at 2.8 Å resolution. *Nature*, 389(6648):251–260, sep 1997. doi: 10.1038/38444. URL https://doi.org/10.1038%2F38444.

[5] Yanjun Zhang, Zhongxing Sun, Junqi Jia, Tianjiao Du, Nachuan Zhang, Yin Tang, Yuan Fang, and Dong Fang. Overview of histone modification. In *Histone Mutations and Cancer*, pages 1–16. Springer Singapore, nov 2020. doi: 10.1007/978-981-15-8104-5_1. URL https://doi.org/10.1007%2F978-981-15-8104-5_1.

[6] Eileen E. M. Furlong and Michael Levine. Developmental enhancers and chromosome topology. *Science*, 361 (6409):1341–1345, 2018. doi: 10.1126/science.aau0320. URL https://www.science.org/doi/abs/10.1126/science.aau0320.

[7] Johannes Nuebler, Geoffrey Fudenberg, Maxim Imakaev, Nezar Abdennur, and Leonid Mirny. Chromatin organization by an interplay of loop extrusion and compartmental segregation. *Biophysical Journal*, 114(3):30a, feb 2018. doi: 10.1016/j.bpj.2017.11.211. URL https://doi.org/10.1016%2Fj.bpj.2017.11.211.

[8] T. Cremer and M. Cremer. Chromosome territories. *Cold Spring Harbor Perspectives in Biology*, 2(3):a003889–a003889, feb 2010. doi: 10.1101/cshperspect.a003889. URL https://doi.org/10.1101%2Fcshperspect.a003889.

[9] Tony Kouzarides. Chromatin modifications and their function. *Cell*, 128(4):693–705, 2007. ISSN 0092-8674. doi: https://doi.org/10.1016/j.cell.2007.02.005. URL https://www.sciencedirect.com/science/article/pii/S0092867407001845.

[10] Jing Liu, Mujahid Ali, and Qi Zhou. Establishment and evolution of heterochromatin. *Annals of the New York Academy of Sciences*, 1476(1):59–77, feb 2020. doi: 10.1111/nyas.14303. URL https://doi.org/10.1111%2Fnyas.14303.

[11] Alexis A. Reyes, Ryan D. Marcum, and Yuan He. Structure and function of chromatin remodelers. *Journal of Molecular Biology*, 433(14):166929, jul 2021. doi: 10.1016/j.jmb.2021.166929. URL https://doi.org/10.1016%2Fj.jmb.2021.166929.

[12] Lian Zhang, Qianjin Lu, and Christopher Chang. *Epigenetics in Health and Disease*, pages 3–55. Springer Singapore, Singapore, 2020. ISBN 978-981-15-3449-2. doi: 10.1007/978-981-15-3449-2_1. URL https://doi.org/10.1007/978-981-15-3449-2_1.

[13] Maria Luce Negri, Sarah D'Annunzio, Giulia Vitali, and Alessio Zippo. May the force be with you: Nuclear condensates function beyond transcription control. *BioEssays*, 45(10), aug 2023. doi: 10.1002/bies.202300075. URL https://doi.org/10.1002%2Fbies.202300075.

[14] Tanja Mittag and Rohit V. Pappu. A conceptual framework for understanding phase separation and addressing open questions and challenges. *Molecular Cell*, 82(12):2201–2214, jun 2022. doi: 10.1016/j.molcel.2022.05.018. URL https://doi.org/10.1016%2Fj.molcel.2022.05.018.

[15] Benjamin R Sabari, Alessandra Dall'Agnese, and Richard A Young. Biomolecular condensates in the nucleus. *Trends in biochemical sciences*, 45(11):961–977, 2020.

[16] Won-Ki Cho, Jan-Hendrik Spille, Micca Hecht, Choongman Lee, Charles Li, Valentin Grube, and Ibrahim I Cisse. Mediator and rna polymerase ii clusters associate in transcription-dependent condensates. *Science*, 361(6400): 412–415, 2018.

[17] Benjamin R Sabari, Alessandra Dall'Agnese, Ann Boija, Isaac A Klein, Eliot L Coffey, Krishna Shrinivas, Brian J Abraham, Nancy M Hannett, Alicia V Zamudio, John C Manteiga, et al. Coactivator condensation at super-enhancers links phase separation and gene control. *Science*, 361(6400):eaar3958, 2018.

[18] Ryanggeun Lee, Moo-Koo Kang, Yong-Jin Kim, Bobae Yang, Hwanyong Shim, Sugyung Kim, Kyungwoo Kim, Chul Min Yang, Byeong-gyu Min, Woong-Jae Jung, et al. Ctcf-mediated chromatin looping provides a topological framework for the formation of phase-separated transcriptional condensates. *Nucleic Acids Research*, 50(1):207–226, 2022.

[19] Je-Kyung Ryu, Céline Bouchoux, Hon Wing Liu, Eugene Kim, Masashi Minamino, Ralph de Groot, Allard J Katan, Andrea Bonato, Davide Marenduzzo, Davide Michieletto, et al. Bridging-induced phase separation induced by cohesin smc protein complexes. *Science advances*, 7(7):eabe5905, 2021.

[20] Teddy Jégu, Eric Aeby, and Jeannie T. Lee. The x chromosome in space. *Nature Reviews Genetics*, 18(6):377–389, may 2017. doi: 10.1038/nrg.2017.17. URL https://doi.org/10.1038%2Fnrg.2017.17.

[21] Giacomo Cavalli and Edith Heard. Advances in epigenetics link genetics to the environment and disease. *Nature*, 571(7766):489–499, jul 2019. doi: 10.1038/s41586-019-1411-0. URL https://doi.org/10.1038%2Fs41586-019-1411-0.

[22] Christoph F. Kurat, Joseph T.P. Yeeles, Harshil Patel, Anne Early, and John F.X. Diffley. Chromatin controls DNA replication origin selection, lagging-strand synthesis, and replication fork rates. *Molecular Cell*, 65(1):117–130, jan 2017. doi: 10.1016/j.molcel.2016.11.016. URL https://doi.org/10.1016%2Fj.molcel.2016.11.016.

[23] Jens Stadler and Holger Richly. Regulation of DNA repair mechanisms: How the chromatin environment regulates the DNA damage response. *International Journal of Molecular Sciences*, 18(8):1715, aug 2017. doi: 10.3390/ijms18081715. URL https://doi.org/10.3390%2Fijms18081715.

[24] Fang-Lin Zhang and Da-Qiang Li. Targeting chromatin-remodeling factors in cancer cells: Promising molecules in cancer therapy. *International Journal of Molecular Sciences*, 23(21), 2022. ISSN 1422-0067. URL https://www.mdpi.com/1422-0067/23/21/12815.

[25] Alessandro Bertero and Manuel Rosa-Garrido. Three-dimensional chromatin organization in cardiac development and disease. *Journal of Molecular and Cellular Cardiology*, 151:89–105, feb 2021. doi: 10.1016/j.yjmcc.2020.11.008. URL https://doi.org/10.1016%2Fj.yjmcc.2020.11.008.

[26] Giovanni Botten, Yuannyu Zhang, Kseniia Dudnyk, Yoon Jung Kim, Xin Liu, Jacob T Sanders, Aygun Imanci, Nathalie M. Droin, Hui Cao, Pranita Kaphle, Kathryn E Dickerson, Kirthi R Kumar, Mingyi Chen, Weina Chen, Eric Solary, Peter Ly, Jian Zhou, and Jian Xu. Structural variation cooperates with permissive chromatin to control enhancer hijacking-mediated oncogenic transcription. *Blood*, mar 2023. doi: 10.1182/blood.2022017555. URL https://doi.org/10.1182%2Fblood.2022017555.

[27] Darío G Lupiáñez, Katerina Kraft, Verena Heinrich, Peter Krawitz, Francesco Brancati, Eva Klopocki, Denise Horn, Hülya Kayserili, John M Opitz, Renata Laxova, et al. Disruptions of topological chromatin domains cause pathogenic rewiring of gene-enhancer interactions. *Cell*, 161(5):1012–1025, 2015.

[28] Meng Wang, Benjamin D Sunkel, William C Ray, and Benjamin Z Stanton. Chromatin structure in cancer. *BMC Molecular and Cell Biology*, 23(1):1–10, 2022.

[29] Job Dekker, Karsten Rippe, Martijn Dekker, and Nancy Kleckner. Capturing chromosome conformation. *Science*, 295(5558):1306–1311, feb 2002. doi: 10.1126/science.1067799. URL https://doi.org/10.1126%2Fscience.1067799.

[30] Zhihu Zhao, Gholamreza Tavoosidana, Mikael Sjölinder, Anita Göndör, Piero Mariano, Sha Wang, Chandrasekhar Kanduri, Magda Lezcano, Kuljeet Singh Sandhu, Umashankar Singh, Vinod Pant, Vijay Tiwari, Sreenivasulu Kurukuti, and Rolf Ohlsson. Circular chromosome conformation capture (4c) uncovers extensive networks of epigenetically regulated intra- and interchromosomal interactions. *Nature Genetics*, 38(11):1341–1347, oct 2006. doi: 10.1038/ng1891. URL https://doi.org/10.1038%2Fng1891.

[31] Josée Dostie, Todd A. Richmond, Ramy A. Arnaout, Rebecca R. Selzer, William L. Lee, Tracey A. Honan, Eric D. Rubio, Anton Krumm, Justin Lamb, Chad Nusbaum, Roland D. Green, and Job Dekker. Chromosome conformation capture carbon copy (5c): A massively parallel solution for mapping interactions between genomic elements. *Genome Research*, 16(10):1299–1309, sep 2006. doi: 10.1101/gr.5571506. URL https://doi.org/10.1101%2Fgr.5571506.

[32] 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, Richard Sandstrom, Bradley Bernstein, M. A. Bender, Mark Groudine, Andreas Gnirke, John Stamatoyannopoulos, Leonid A. Mirny, Eric S. Lander, and Job Dekker. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science*, 326(5950):289–293, oct 2009. doi: 10.1126/science.1181369. URL https://doi.org/10.1126%2Fscience.1181369.

[33] Suhas S.P. Rao, Miriam H. Huntley, Neva C. Durand, Elena K. Stamenova, Ivan D. Bochkov, James T. Robinson, Adrian L. Sanborn, Ido Machol, Arina D. Omer, Eric S. Lander, and Erez Lieberman Aiden. A 3d map of the human genome at kilobase resolution reveals principles of chromatin looping. *Cell*, 159(7):1665–1680, dec 2014. doi: 10.1016/j.cell.2014.11.021. URL https://doi.org/10.1016%2Fj.cell.2014.11.021.

[34] Tsung-Han S. Hsieh, Assaf Weiner, Bryan Lajoie, Job Dekker, Nir Friedman, and Oliver J. Rando. Mapping nucleosome resolution chromosome folding in yeast by micro-c. *Cell*, 162(1):108–119, jul 2015. doi: 10.1016/j.cell.2015.05.048. URL https://doi.org/10.1016%2Fj.cell.2015.05.048.

[35] Koustav Pal, Mattia Forcato, and Francesco Ferrari. Hi-c analysis: from data generation to integration. *Biophysical Reviews*, 11(1):67–78, dec 2018. doi: 10.1007/s12551-018-0489-1. URL https://doi.org/10.1007%2Fs12551-018-0489-1.

[36] Fulai Jin, Yan Li, Jesse R. Dixon, Siddarth Selvaraj, Zhen Ye, Ah Young Lee, Chia-An Yen, Anthony D. Schmitt, Celso A. Espinoza, and Bing Ren. A high-resolution map of the three-dimensional chromatin interactome in human cells. *Nature*, 503(7475):290–294, oct 2013. doi: 10.1038/nature12644. URL https://doi.org/10.1038%2Fnature12644.

[37] Wibke Schwarzer, Nezar Abdennur, Anton Goloborodko, Aleksandra Pekowska, Geoffrey Fudenberg, Yann Loe-Mie, Nuno A Fonseca, Wolfgang Huber, Christian H. Haering, Leonid Mirny, and Francois Spitz. Two independent modes of chromatin organization revealed by cohesin removal. *Nature*, 551(7678):51–56, sep 2017. doi: 10.1038/nature24281. URL https://doi.org/10.1038%2Fnature24281.

[38] Joachim Wolff, Rolf Backofen, and Björn Grüning. Loop detection using Hi-C data with HiCExplorer. *GigaScience*, 11:giac061, 07 2022. ISSN 2047-217X. doi: 10.1093/gigascience/giac061. URL https://doi.org/10.1093/gigascience/giac061.

[39] The HDF Group. The hdf5® library and file format, Accessed on 2023-09-14. URL https://www.hdfgroup.org/solutions/hdf5/.

[40] Nezar Abdennur and Leonid A Mirny. Cooler: scalable storage for Hi-C data and other genomically labeled arrays. *Bioinformatics*, 36(1):311–316, 2020. doi: 10.1093/bioinformatics/btz540. URL https://doi.org/10.1093/bioinformatics/btz540.

[41] Hongqiang Lyu, Erhu Liu, and Zhifang Wu. Comparison of normalization methods for hi-c data. *BioTechniques*, 68 (2):56–64, feb 2020. doi: 10.2144/btn-2019-0105. URL https://doi.org/10.2144%2Fbtn-2019-0105.

[42] Maxim Imakaev, Geoffrey Fudenberg, Rachel Patton McCord, Natalia Naumova, Anton Goloborodko, Bryan R Lajoie, Job Dekker, and Leonid A Mirny. Iterative correction of hi-c data reveals hallmarks of chromosome organization. *Nature Methods*, 9(10):999–1003, sep 2012. doi: 10.1038/nmeth.2148. URL https://doi.org/10.1038%2Fnmeth.2148.

[43] P. A. Knight and D. Ruiz. A fast algorithm for matrix balancing. *IMA Journal of Numerical Analysis*, 33(3): 1029–1047, oct 2012. doi: 10.1093/imanum/drs019. URL https://doi.org/10.1093%2Fimanum%2Fdrs019.

[44] Open Chromosome Collective. Open chromosome collective webpage, Accessed on 2023-09-25. URL https://open2c.github.io/.

[45] Open2C, Nezar Abdennur, Sameer Abraham, Geoffrey Fudenberg, Ilya M. Flyamer, Aleksandra A. Galitsyna, Anton Goloborodko, Maxim Imakaev, Betul A. Oksuz, and Sergey V. Venev. Cooltools: enabling high-resolution hi-c analysis in python. Pre-print, 2022. URL https://www.biorxiv.org/content/early/2022/11/01/2022.10.31.514564.

[46] Vera Pancaldi. Network models of chromatin structure. *Current Opinion in Genetics & Development*, 80:102051, jun 2023. doi: 10.1016/j.gde.2023.102051. URL https://doi.org/10.1016%2Fj.gde.2023.102051.

[47] Tiago P. Peixoto. The graph-tool python library. *figshare*, 2014. doi: 10.6084/m9.figshare.1164194. URL http://figshare.com/articles/graph_tool/1164194.

[48] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[49] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

[50] The pandas development team. pandas-dev/pandas: Pandas, February 2020. URL https://doi.org/10.5281/zenodo.3509134.

[51] Ryan K. Dale, Brent S. Pedersen, and Aaron R. Quinlan. Pybedtools: a flexible python library for manipulating genomic datasets and annotations. *Bioinformatics*, 27(24):3423–3424, 09 2011. ISSN 1367-4803. doi: 10.1093/bioinformatics/btr539. URL https://doi.org/10.1093/bioinformatics/btr539.

[52] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021. doi: 10.21105/joss.03021. URL https://doi.org/10.21105/joss.03021.

[53] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[54] 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, Richard Sandstrom, Bradley Bernstein, M. A. Bender, Mark Groudine, Andreas Gnirke, John Stamatoyannopoulos, Leonid A. Mirny, Eric S. Lander, and Job Dekker. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science*, 326(5950):289–293, oct 2009. doi: 10.1126/science.1181369. URL https://doi.org/10.1126%2Fscience.1181369.

[55] M. Ángeles Serrano, Marián Boguñá, and Alessandro Vespignani. Extracting the multiscale backbone of complex weighted networks. *Proceedings of the National Academy of Sciences*, 106(16):6483–6488, apr 2009. doi: 10.1073/pnas.0808904106. URL https://doi.org/10.1073%2Fpnas.0808904106.

[56] memory-profiler, 2022. URL https://pypi.org/project/memory-profiler/. Version 0.61.0.

[57] Sarah B. Reiff, Andrew J. Schroeder, Koray Kırlı, Andrea Cosolo, Clara Bakker, Luisa Mercado, Soohyun Lee, Alexander D. Veit, Alexander K. Balashov, Carl Vitzthum, William Ronchetti, Kent M. Pitman, Jeremy Johnson, Shannon R. Ehmsen, Peter Kerpedjiev, Nezar Abdennur, Maxim Imakaev, Serkan Utku Öztürk, Uğur Çamoğlu, Leonid A. Mirny, Nils Gehlenborg, Burak H. Alver, and Peter J. Park. The 4d nucleome data portal as a resource for searching and visualizing curated nucleomics data. *Nature Communications*, 13(1), may 2022. doi: 10.1038/s41467-022-29697-4. URL https://doi.org/10.1038%2Fs41467-022-29697-4.

[58] Soohyun Lee, Clara R Bakker, Carl Vitzthum, Burak H Alver, and Peter J Park. Pairs and Pairix: a file format and a tool for efficient storage and retrieval for Hi-C read pairs. *Bioinformatics*, 38(6):1729–1731, 01 2022. ISSN 1367-4803. doi: 10.1093/bioinformatics/btab870. URL https://doi.org/10.1093/bioinformatics/btab870.

[59] Mattia Forcato, Chiara Nicoletti, Koustav Pal, Carmen Maria Livi, Francesco Ferrari, and Silvio Bicciato. Comparison of computational methods for hi-c data analysis. *Nature Methods*, 14(7):679–685, jun 2017. doi: 10.1038/nmeth.4325. URL https://doi.org/10.1038%2Fnmeth.4325.

[60] Polars: Blazingly fast dataframes in rust, python, node.js, r and sql, 2023. URL https://github.com/pola-rs/polars/.

[61] Swifter: A package which efficiently applies any function to a pandas dataframe or series in the fastest available manner., 2023. URL https://github.com/jmcarpenter2/swifter.

[62] Dask: a flexible parallel computing library for analytics., 2023. URL https://github.com/dask/dask.

[63] H5py: a thin, pythonic wrapper around hdf5., 2023. URL https://github.com/h5py/h5py.

[64] Suhas S.P. Rao, Miriam H. Huntley, Neva C. Durand, Elena K. Stamenova, Ivan D. Bochkov, James T. Robinson, Adrian L. Sanborn, Ido Machol, Arina D. Omer, Eric S. Lander, and Erez Lieberman Aiden. A 3d map of the human genome at kilobase resolution reveals principles of chromatin looping. *Cell*, 162(3):687–688, jul 2015. doi: 10.1016/j.cell.2015.07.024. URL https://doi.org/10.1016%2Fj.cell.2015.07.024.

[65] Tommaso Lanciano, Francesco Bonchi, and Aristides Gionis. Explainable classification of brain networks via contrast subgraphs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3308–3318, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403383. URL https://doi.org/10.1145/3394486.3403383.

[66] Tommaso Lanciano, Aurora Savino, Francesca Porcu, Davide Cittaro, Francesco Bonchi, and Paolo Provero. Contrast subgraphs allow comparing homogeneous and heterogeneous networks derived from omics data. *GigaScience*, 12: giad010, 02 2023. ISSN 2047-217X. doi: 10.1093/gigascience/giad010. URL https://doi.org/10.1093/gigascience/giad010.

[67] Black: The uncompromising code formatter., 2023. URL https://github.com/psf/black.

[68] Python enhancement proposals (peps)., 2023. URL https://peps.python.org/pep-0000/.

[69] Pyflakes: A simple program which checks python source files for errors., 2023. URL https://pypi.org/project/pyflakes/.

[70] Sphinx makes it easy to create intelligent and beautiful documentation., 2023. URL https://github.com/sphinx-doc/sphinx.

[71] The jupyter notebook is a web-based notebook environment for interactive computing., 2023. URL https://github.com/jupyter/notebook.

[72] The pytest framework makes it easy to write small tests, yet scales to support complex functional testing for applications and libraries., 2023. URL https://github.com/pytest-dev/pytest/.

Appendix I

This section contains information which could be useful to understand the package, though not deemed relevant enough to be placed in the materials and methods section.

Node-level statistics

Mathematical formulations for some node-level statistics which can be used for node-labels permutation.

- **Betweenness centrality**: a measure of the importance (centrality) of the node for the flow of information through the network. Mathematically defined as:

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

  with s,v and t being three vertices of the graph, σ_{st} being the number of shortest paths from s to t and $\sigma_{st}(v)$ being the number of shortest paths from s to t which do include v.
- **Local clustering coefficient**: a measure of how connected the neighborhood of a node is. Defined as:

$$c_i = \frac{\{e_{jk}\}}{k_i(k_i - 1)} : v_j, v_k \in N_i, e_j k \in E$$

  where $\{e_{jk}\}$ is the number of edges among the neighbors of v_i (N_i) present in the graph, while $k_i(k_i - 1)$ represents the total number of possible edges among those neighbors, being k_i the degree of vertex v_i.

Appendix II

In this section, replication data is provided for all the analyses for which aggregating the results coming from multiple files, or even chromosomes, is very difficult.

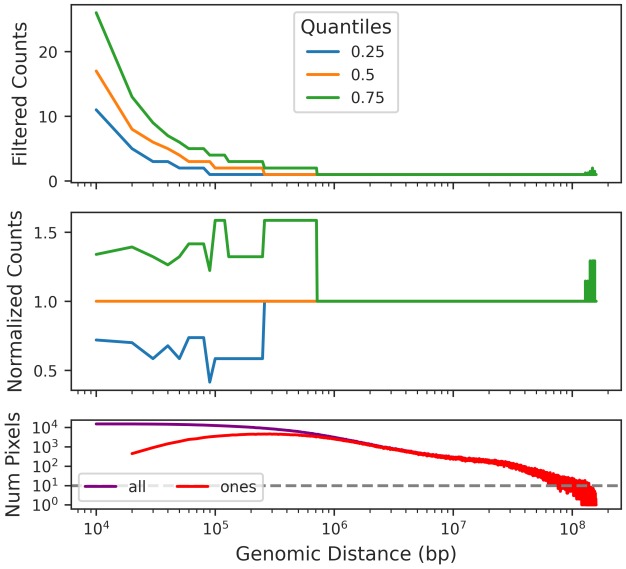


Figure 6.1: Replication data for figure 4.2. Comparison of the distribution of raw and normalized counts at each genomic distance for chromosome 7 of one replicate of IMR90 cells (4DNFIMU9T2QI), at 10 kb resolution, filtered using 200 Mb as genomic distance threshold.

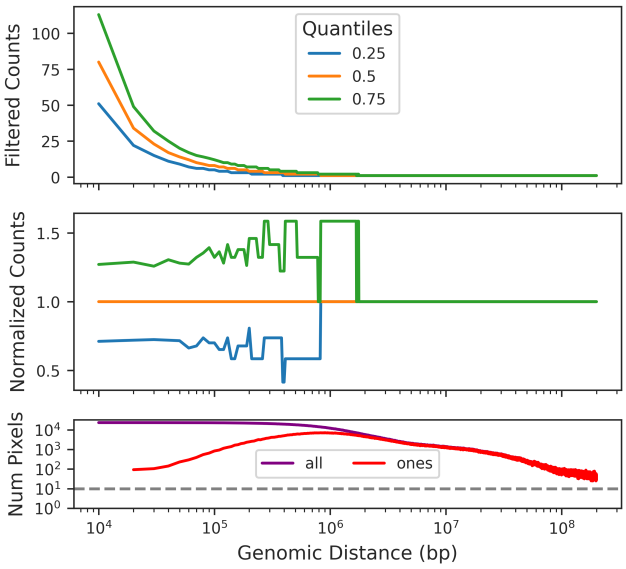


Figure 6.2: Replication data for figure 4.2. Comparison of the distribution of raw and normalized counts at each genomic distance for chromosome 2 of HMEC cells (4DNFIIFAUT24), at 10 kb resolution, filtered using 200 Mb as genomic distance threshold.

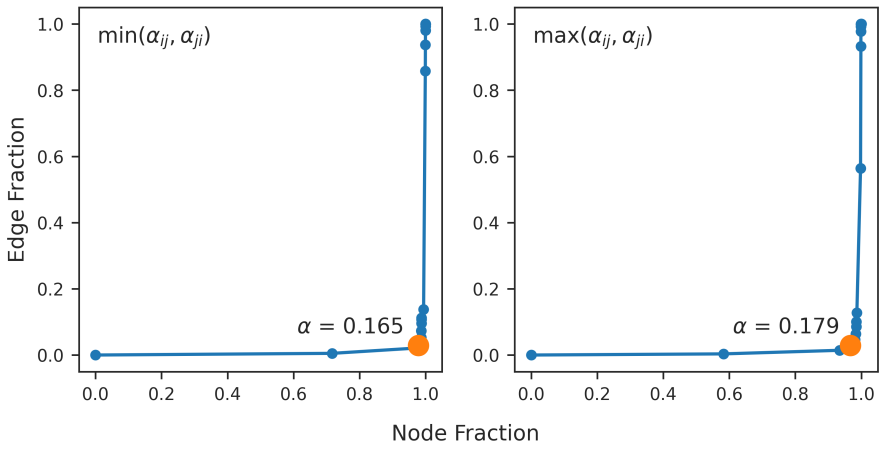


Figure 6.3: Replication data for figure 4.4. In the panels, sets of points tested to find the optimal alpha value for chromosome 1 of HMEC cells (4DNFIIFAUT24) at 10 kb resolution, processed using 200 Mb as distance threshold, 0 as quantile threshold.

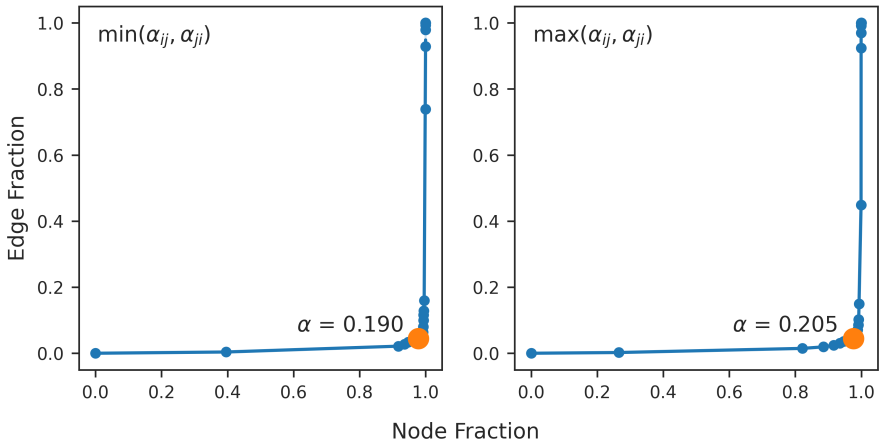


Figure 6.4: Replication data for figure 4.4. In the panels, sets of points tested to find the optimal alpha value for chromosome 19 of one replicate of IMR90 cells (4DNFIMU9T2QI) at 10 kb resolution, processed using 200 Mb as distance threshold, 0 as quantile threshold.

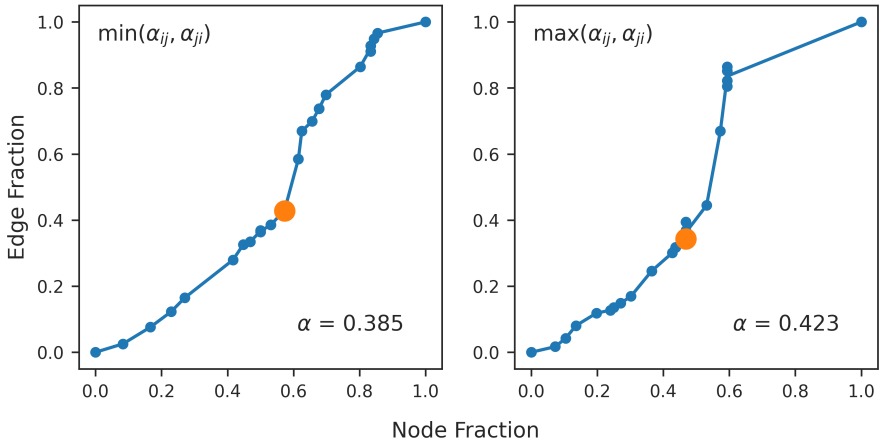


Figure 6.5: Replication data for figure 4.4. In the panels, sets of points tested to find the optimal alpha value for chromosome Y of one replicate of IMR90 cells (4DNFIMU9T2QI) at 10 kb resolution, processed using 200 Mb as distance threshold, 0 as quantile threshold. Reported to show the instability of the procedure on chromosome Y.

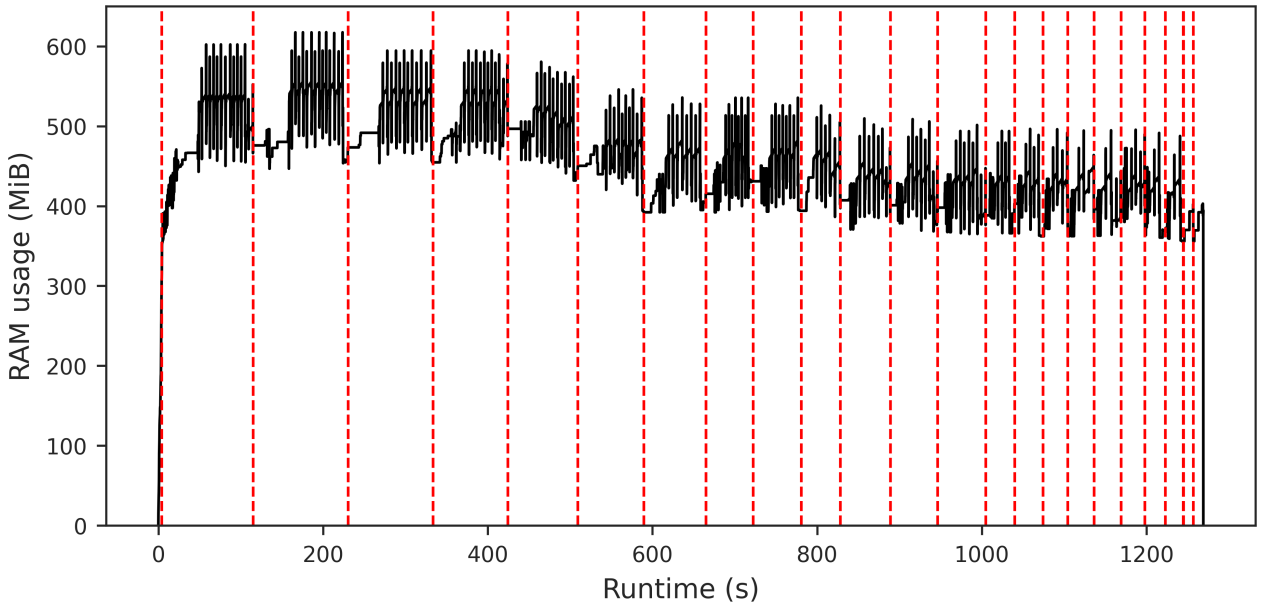


Figure 6.6: Replication data for figure 4.7. Plot of Random Access Memory usage during the preprocessing procedure. Runtime is expressed in seconds, while memory usage is expressed in MebiBytes (1 MiB = 2^{20} bytes). The area included between two red dashed lines corresponds to the sparsification scores computation for a chromosome. In the figure, the RAM usage overtime for HMEC cells (4DNFIIFAUT24) at 10 kb resolution, processed using 100 Mb as distance threshold, 0.01 as quantile threshold, is shown.