UNIVERSITY
OF OSLO

Roberto Rossini

Developing reproducible and efficient computational tools for modeling and analysis of dynamic multi-level 3D genome organization

Thesis submitted for the degree of Philosophiae Doctor

Department of Biosciences
Faculty of Mathematics and Natural Sciences

2023

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 2712

ISSN 1501-7710

Cover: UiO.
Print production: Graphic center, University of Oslo.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of *Philosophiae Doctor* at the University of Oslo. The research presented here was conducted at the University of Oslo under the supervision of Professor Jonas Paulsen.

The thesis is a collection of three papers, presented in chronological order of writing. The papers are preceded by an introductory chapter that relates them to each other and provides background information and motivation for the work.

Acknowledgements

First, I would like to thank my supervisor Jonas for always being supportive, and guiding me through my PhD.

Second, I wish to thank my colleagues at EVOGENE, in particular, I would like to thank my friends and colleagues Andrea H., Andrea R., Dominik, Fabiana, Negar, Oda, and Saleh, for the many scientific discussions, coffee breaks, and beer evenings.
A special thanks to Negar and Saleh for sharing the office with me and listening to my complaints and loud keyboard for the past three years.
Many thanks also to Rein for the numerous, thought-provoking scientific discussions.
Thank you also to my friends and colleagues at the department for the many chats and coffee breaks, especially Chiara, Dabao, Ida, Frida, Renate, and Veslemøy.
Thanks to Yuri and Sissel for being my friends and joining me on many hikes and fishing trips around Oslo.

Lastly, I would like to thank my family in Italy for being supportive and understanding throughout my studies.

Roberto Rossini
Oslo, December 2023

Summary

In English

The genome of eukaryotes is tightly packed into micrometer-sized nuclei.

To achieve this impressive feat, DNA is folded into a number of 3D structures ranging from nucleosomes at the nanometer scale to chromosome territories at the micrometer scale. These structures are not just simply a means to achieve the required level of compaction, but rather structures with functional implications when it comes to gene expression, DNA repair, recombination, and more.

Over the past two decades, Chromosome Conformation Capture (3C) methods have evolved to map the 3D structure of whole genomes, including Hi-C methods, which is widely used to map pairwise 3D genome interactions. These techniques pioneered the discovery of Topologically Associating Domains (TADs), which are kilo- to megabase scale regions that show preferential self-interaction in 3D space. It was later discovered that the vast majority of TADs are a consequence of a process known as DNA loop extrusion.

The first part of this dissertation was dedicated to the development of computational tools for simulation and data analysis of the 3D genome. As part of our software development efforts, we developed a high-performance *in silico* model of DNA loop extrusion, named MoDLE, that is capable of simulating loop extrusion on the entire human genome in a matter of minutes.

While developing MoDLE we realized a lack of interoperability between .hic and cooler, the two most popular file formats used to store Hi-C interactions. This prompted us to develop hictk, a toolkit to transparently perform common operations on .hic and cooler files, such as converting between the two formats and fetching interactions overlapping a region of interest.

The second part of this dissertation focused on the analysis of cancer data with emphasis on 3D genomics. Here we characterized changes in the 3D structure of genomes during breast cancer progression. We report significant changes at the subcompartment and TAD clique level. Furthermore, we identify a structural variation at the *MYC* locus that is associated with the dysregulation of several oncogenes.

På Norsk

Genomet hos Eukaryoter er tett pakket inn i mikrometerstore cellekjerner. For å oppnå dette, er DNAet foldet inn i en rekke 3D-strukturer som strekker seg fra nukleosomer på nanometerskala til kromosomterritorier på mikrometerskala. Disse strukturene er ikke bare nødvendig for å gjøre DNAet kompakt, men har også funksjonell betydning for genuttrykk, DNA-reparasjon og rekombinasjon.

I løpet av de siste årtiene har Chromosome Conformation Capture (3C)-metoder blitt utviklet for å kartlegge hele genomer i tre dimensjoner (3D), inkludert Hi-C-metoder som nå er mye brukt for å kartlegge genomvide parvise 3D-interaksjoner. Disse teknikkene har vært banebrytende for oppdagelsen av topologisk assosierende domener (TAD), som er regioner i kilo- til megabasestørrelse med anrikning av 3D-kontakter innad i domenene. Det har senere vist seg at flertallet av TADene er et resultat av en prosess kjent som DNA-løkkeekstrudering.

Den første delen av denne avhandlingen ble viet til utvikling av beregningsverktøy for simulering og dataanalyse av 3D-genomet. Som en del av dette, utviklet vi en effektiv in silico-modell av DNA-løkkeekstrudering, kalt MoDLE, som er i stand til å simulere løkkeekstrudering på hele det menneskelige genom på få minutter.

Under utviklingen av MoDLE innså vi at det manglet interoperabilitet mellom formatene .hic og cooler, som er er de to mest populære filformatene for lagring av Hi-C-interaksjoner. Dette ledet oss til å utvikle hictk, et verktøy for å utføre vanlige operasjoner på .hic og cooler-filer på en brukervennlig måte, som for eksempel å konvertere mellom de to formatene, eller å hente interaksjoner som overlapper et interesseområde.

Den andre delen av denne avhandlingen fokuserte på analyse av kreftdata med vekt på 3D-genomikk. Her karakteriserte vi endringer i 3D-strukturen av genomer under utvikling og progresjon av brystkreft. Vi fant signifikante endringer på subkompartement- og TAD-klikknivå. Vi identifiserte også strukturell variasjon ved MYC-lokuset som er assosiert med dysregulering av flere onkogener.

List of Papers

Paper I

Rossini R., Kumar V., Mathelier A., Rognes T. and Paulsen J.
“MoDLE: high-performance stochastic modeling of DNA loop extrusion interactions”
Genome Biology 23, 247 (2022) DOI: 10.1186/s13059-022-02815-7.

Paper II

Rossini R. and Paulsen J.
“hictk: blazing fast toolkit to work with .hic and .cool files”
Submitted for publication. DOI: 10.1101/2023.11.26.568707

Paper III

Rossini R., Oshaghi M., Nekrasov M., Bellanger A., Domaschenz R., Dijkwel Y., Abdelhalim M., Collas P., Tremethick D. and Paulsen J.
“Multi-level 3D genome organization deteriorates during breast cancer progression”
Submitted for publication. DOI: 10.1101/2023.11.26.568711

Contents

List of Figures

List of Acronyms

3C	Chromosome Conformation Capture
4C	Circular Chromosome Conformation Capture
5C	Chromosome Conformation Capture Carbon Copy
ACP	Architectural Chromatin Proteins
AID	Auxin Inducible Degron
AML	Acute Myeloid Leukemia
API	Application Programming Interface
ASan	Address Sanitizer
ATAC-seq	Assay for Transposase-Accessible Chromatin using Sequencing
BED	Browser Extensible Data
BEDPE	BED Paired-End
bp	base pairs
CARGO	Chimeric Array of gRNA Oligonucleotides
CHi-C	Capture Hi-C
ChIA-Drop	Chromatin Interaction Analysis with Droplets
ChIP-seq	Chromatin ImmunoPrecipitation Sequencing
CI	Continuous Integration
CIGAR	Compact Idiosyncratic Gapped Alignment Report
CLI	Command Line Interface
CND	Chromatin Nanodomains
CNV	Copy Number Variation
CSR	Compressed Sparse Row
CTCF	CCCTC-binding Factor
CTs	Chromosome Territories

dCas9	catalitically-dead Cas9
DDR	DNA-Damage Response
DI	Directionality Index
DPM	DNA phosphate modified
DSB	Double Strand Break
DSG	disuccinimidyl glutarate
ecDNA	extra chromosomal DNA
ERC	European Research Council
FACS	Fluorescence Activated Cell Sorting
FAIR	Findability, Accessibility, Interoperability, and Reusability
FISH	Fluorescence In Situ Hybridization
FUSE	Filesystem in USErspace
GAM	Genome Architechture Mapping
GEM	Gel bead in Emulsion
GEO	Gene Expression Omnibus
HDF	Hierarchical Data Format
HDF5	Hierarchical Data Format 5
HiPore-C	High-throughput Pore-C
ICE	Iterative Correction and Eigenvector decomposition
ICR	Imprinting Control Region
INM	Inner Nuclear Membrane
IO	Input-Output
IS	Insulation Score
JVM	Java Virtual Machine
LAD	Lamina Associating Domain
LCR	Locus Control Region
LEF	Loop Extruding Factor
LINE	Long Long Interspersed Nuclear Element

MCM Minichromosome Maintenance

MD Molecular Dynamics

MERFISH Multiplexed Error-Robust Fluorescence in situ Hybridization

MINA Multiplexed Imaging of Nucleome Architecture

MLE Maximum Likelihood Estimation

MoDLE Modeling of DNA Loop Extrusion

NCHG NonCentral HyperGeometric

NGS Next-Generation Sequencing

NHS N-hydroxy-succinimide

NPC Nuclear Pore Complex

ONM Outer Nuclear Membrane

ORCA Optical Reconstruction of Chromatin Architecture

OSS Open-Source Software

PC Principal Component

PCA Principal Component Analysis

PCR Polymerase Chain Reaction

PRNG Pseudo-Random Number Generator

PTM Post Translational Modifications

qPCR Quantitative PCR

RCMC Region Capture Micro-C

RD-SPRITE RNA & DNA SPRITE

RNAP RNA Polymerase

rRNA Ribosomal RNA

scHi-C single-cell Hi-C

scNanoHi-C single-cell Nanopore Hi-C

scSPRITE single-cell SPRITE

seqFISH+ Sequential Fluorescence in Situ Hybridization

sgRNA short guide RNA

SMC	Structural Maintenance of Chromosomes
SPRITE	Split-Pool Recognition of Interactions by Tag Extension
SRA	Sequence Read Archive
TAD	Topologically Associating Domain
Trac-looping	Transposases-mediated Analysis of Chromatin Looping
TSS	Transcription Start Site
TTS	Transcription Termination Site
UBSan	Undefined Behavior Sanitizer
UiO	Universitetet i Oslo
URI	Uniform Resource Identifier
WGS	Whole Genome Sequencing

Part I

Introduction

Chapter 1

Genome organization in three dimensions

1.1 Introduction

This chapter introduces the core principles of three-dimensional (3D) genome organization, including nucleosomes, TADs, loops, compartments, and chromosome territories. The chapter also focuses on DNA loop extrusion, which is responsible for the formation of TADs, loops, and a few other 3D features. DNA loop extrusion is covered in depth, as it is a central aspect of Paper I. Finally, the information presented in this chapter mainly pertains to chromatin organization in metazoans, even though variants of most of the processes and features discussed are present in most eukaryotes.

1.2 Chromatin organization

In eukaryotes, genomic DNA is tightly packaged with various types of proteins to form chromatin. This packaging must be efficient, as considerable amounts of DNA must fit in a relatively small nucleus. Yet, packaging must be well organized and allow rapid local chromatin remodeling to facilitate processes such as gene expression, DNA replication, recombination, and repair. To meet all the above requirements, chromatin is organized into a hierarchical structure, with different structures and processes being involved at different levels of the hierarchy [1].

1.2.1 DNA and nucleosomes

At the nanometer scale, chromatin is a complex of DNA and proteins such as histones and transcription factors [2]. Due to its phosphate backbone, DNA is a negatively charged polymer and thus cannot easily fold on itself because of electrostatic repulsive forces. To achieve the high level of compaction observed in eukaryote nuclei, DNA is wrapped around wedge-shaped protein complexes to form nucleosomes.

Each nucleosome consists of 145–147 base pairs (bp) of DNA wrapped around a core histone octamer made of two of each of H2A, H2B, H3 and H4 proteins [2, 3]. Adjacent nucleosomes are connected through a DNA-linker of variable length (20-50 bp), resulting in a bead-on-a-string like structure that constitutes the fundamental structural unit of chromatin [2, 3]. The DNA-linker connecting adjacent nucleosomes is often bound by linker histones, namely histon H1 and its variants, forming chromatosomes. H1 histones have profound effects on chromatin folding and accessibility, as their presence promotes chromatin compaction and limits DNA accessibility to DNA-binding proteins [4].

Besides the canonical histone proteins, nucleosomes can include other histone variants, which contribute to the structural and functional diversity of chromatin [5]. For example, histone variant macro-H2A.1 has been shown to replace H2A in regions subjected to X chromosome inactivation, where its inclusion in nucleosomes inhibits transcription factor binding [5, 6].

Furthermore, histones are susceptible to Post Translational Modifications (PTM), which affect histones' affinity for DNA and other histones or proteins. PTMs are involved in a myriad of functions, among which the recruitment of ATP-dependent chromatin remodellers (see below), transcription promotion and repression by facilitating or obstructing the recruitment of the transcriptional machinery, and DNA repair, by marking regions slated for repair with phosphorylated H2AX (γH2AX) [7–9].

On top of histone modifications, nucleosomes can be remodeled by Architectural Chromatin Proteinss (ACPs) [10], histone chaperones [11], and ATP-dependent chromatin remodellers [12], all of which can affect how histones interact with other molecules, including DNA and other histones. Histone chaperones and ATP-dependent chromatin remodellers are responsible for incorporating specific histone variants into the histone core [3, 5]. ATP-dependent chromatin remodellers are also responsible for chromatin remodeling by sliding, displacing, or removing nucleosomes. This can have profound impacts on transcription activation and repression when remodeling occurs around the Transcription Start Site (TSS) of genes [3, 12]. Polycomb proteins are an example of ACP and play an important role in gene silencing during development and differentiation through methylation of H3 and ubiquitination of histone H2A [10, 13].

Overall, nucleosomes are highly configurable DNA/protein complexes that are crucial to chromatin folding in eukaryotes [3, 7].

Chromatin characterized by the bead-on-a-string structure, also known as 10 nm chromatin, organizes into higher-order structures. Since late 1970, it was believed that 10 nm chromatin folds into 30 nm chromatin, as this is the structure that was observed *in vitro* studies [14–18]. However, over the past 10 years, a large number of studies tried and failed to image 30 nm chromatin *in vivo*, suggesting that if 30 nm chromatin exists in living cells, it is not the most common structure [19–24]. Furthermore, recent studies using live-imaging and single-cell Hi-C (reviewed in Section 2.3.2 and Section 2.2.1.8, respectively) to visualize individual nuclei revealed that chromatin structure is highly dynamic, and its structure varies within and among nuclei, suggesting that chromatin cannot be tightly folded in a single, static structure [25, 26]. In recent years, Ricci et al. [27] showed that a variable number of nucleosomes transiently interact *in vivo*, forming nucleosome clutches, which are nucleosome-rich regions interspersed with nucleosome-poor regions. The existence of nucleosome clutches is also supported by Micro-C data, which identified a large number of short stretches of nucleosomes organized in a zig-zag orientation [28]. Clutches have been proposed to organize in higher order structures named Chromatin Nanodomainss (CNDs) [29], whose formation is likely driven by nucleosome-nucleosome interactions [29].

Overall, while the bead-on-a-string is widely recognized as a model of chromatin, more studies are needed to shed light on how individual nucleosomes fold into higher-order structures.

1.2.2 Topologically Associating Domains

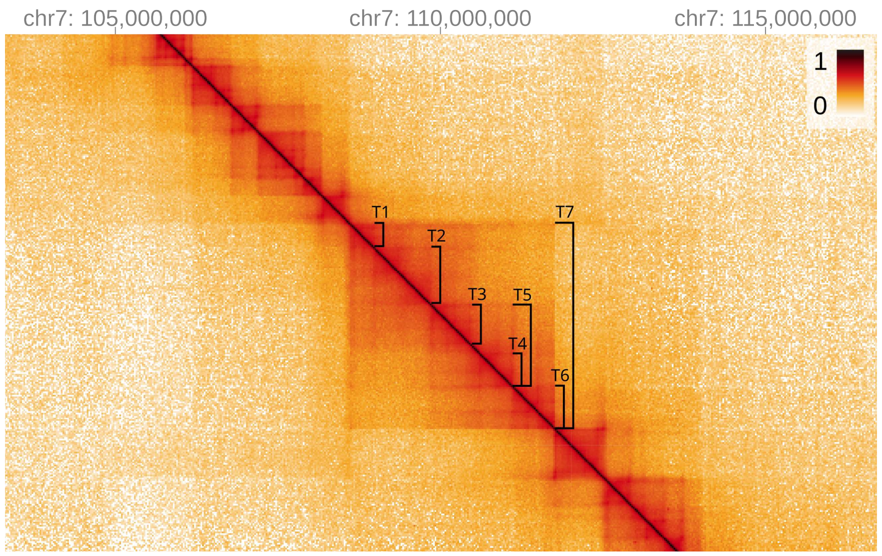


Figure 1.1: Micro-C heatmap showing TADs over a 13 Mbp region from the human genome. TADs within TAD T7 are highlighted with square braces. Individual pixels in the heatmap show the interaction frequency between a pair of genomic intervals (also known as bins). Pixels in dark shades of red represent pairs of bins characterized by high interaction frequencies (see color scale on the top-right).

Heatmap generated from dataset 4DNFI9GMP2J8 at 25 kbp resolution [28, 30].

Topologically Associating Domains (TADs) are genomic regions in the range of 100 kbp to 5 Mbp that exhibit preferential self-interactions [31]. They were first described by Dixon et al. [32] and Nora et al. [33] by analyzing high-resolution Hi-C and 5C maps (see Sections 2.2.1.3 and 2.2.1.4). Since then, TADs have also been observed using other methods, such as GAM, SPRITE and ChIA-Drop (reviewed in Sections 2.2.2.1, 2.2.2.2 and 2.2.2.4). TADs appear like squares along the Hi-C matrix diagonal (often nested within one another, see Figure 1.1) and are observed in the genome of many eukaryotes, including animals, flies, nematodes, and fungi [34, 35]. There is convincing evidence that most TAD structures observed in Hi-C data result from a process known as DNA loop extrusion (reviewed in Section 1.3 and Paper I). Surprisingly, TADs are usually not observable in individual cells [26], but rather become apparent when looking at an ensemble of cells. This can be explained by a preference for loop formation within TADs rather than between TADs that becomes more obvious when aggregating multiple cells.

Despite being one of the most studied chromatin structures, there is no unique formal definition of TAD. Instead, there are many operative definitions of TADs which have historically been introduced as part of the development of new software for TAD-calling [36]. Dixon et al. [32] were among the first to provide an operative definition of TADs through the Directionality Index (DI). Given a genomic region, the DI quantifies the degree of interaction with upstream and downstream regions. Genomic regions inside TADs usually exhibit a DI of approximately zero, as they interact with equal frequencies with up- and downstream regions. When approaching TAD boundaries the DI will be different from zero, as genomic regions inside and close to the beginning of TADs show preferential interactions with downstream regions, while regions inside TADs but close to the end exhibit preferential interactions with upstream regions. A few years later, Crane et al. [37] introduced a new definition of TADs based on Insulation Score (IS) (also known as diamond score). Insulation scores are computed by sliding a diamond-shaped window along the Hi-C matrix diagonal and aggregating interactions. When inside TADs, IS will be large, as the sliding diamond overlaps regions with intra-TAD interactions. Conversely, at TAD boundaries, IS will be small, as in this case the sliding diamond overlaps inter-TAD interactions. Rao et al. [38] proposed a heuristic method for TAD-calling, named Arrowhead. Arrowhead applies a transformation to Hi-C matrices, after which TADs appear as arrowhead-shaped patterns, with the arrow's head pointing to the left corner of TADs. Arrowheads in the transformed matrix, and thus TAD boundaries, are identified using a dynamic programming approach.

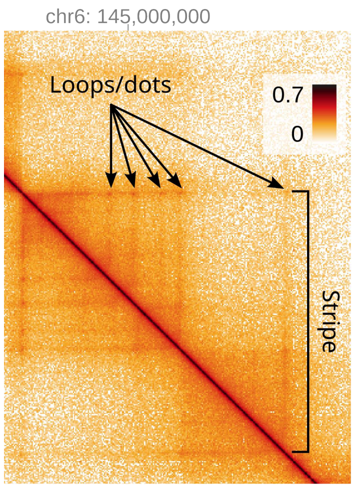
All TAD definitions discussed so far ignore the nested nature of TADs, where smaller sub-TADs are often found inside larger TADs. In 2015, Weinreb et al. [39] published TADtree, the first method capable of identifying hierarchies of TADs. TADtree models TADs from different resolutions as TAD forests and then finds the optimal forest by maximizing an objective function using dynamic programming.

Since the first TAD discovery, a large number of TAD-callers have been developed. As a comprehensive review of all TAD-calling methods is outside the scope of this thesis, for more details, readers are referred to the following reviews on the topic [40–42].

A controversial topic is the functional role of TADs. TADs have been observed across many eukaryotic species, suggesting that they are an important feature of chromatin folding [34, 35]. Initial studies investigating the functional role of TADs reported that enhancers preferentially interacted with promoters located within the same TAD, suggesting that TADs may act as regulatory domains for gene expression, facilitating enhancer-promoter interactions within the same TAD and obstructing enhancer-promoter interactions among different TADs [43]. Thus, TADs have been proposed to be units for DNA replication, V(D)J recombination, and repair (reviewed in Section 1.3.2). This idea was further substantiated by several other studies showing how perturbing specific TADs had a significant and specific impact on tissue and disease development [44]. However, recent studies introducing perturbation to TADs and DNA loop extrusion genome-wide suggested a more nuanced view. Nora et al. [45] show that while rapid CCCTC-binding Factor (CTCF) degradation through Auxin Inducible Degron (AID) in mice leads to a near-complete loss of TADs, it only had relatively mild effects on gene expression regulation, causing the misregulation of 370 genes after 1 day of AID induction and 4996 genes 4 days after induction. Similar results were reported by Hsieh et al. [46], where the authors showed loss that acute loss of CTCF, cohesin, WAPL, or YY1 had minimal effects on promoter-enhancer interactions. Altogether, these results suggest that TADs may not be required for the maintenance of promoter-enhancer interactions, but rather for their establishment right after mitotic exit. Different studies assessed the effects of adding one or more CTCF binding sites between promoter and enhancers that are known to be interacting and showed that the reduction in gene expression was significant but not complete. For a more thorough overview of perturbations to DNA loop extrusion refer to the review by de Wit et al. [47].

In conclusion, despite their prevalence, TADs are poorly defined features of the chromatin 3D structure. Despite this, TADs have been shown to be implicated in various biological processes, such as gene expression regulation, DNA repair, and somatic recombination.

1.2.3 Extrusion loops, stripes, and fountains



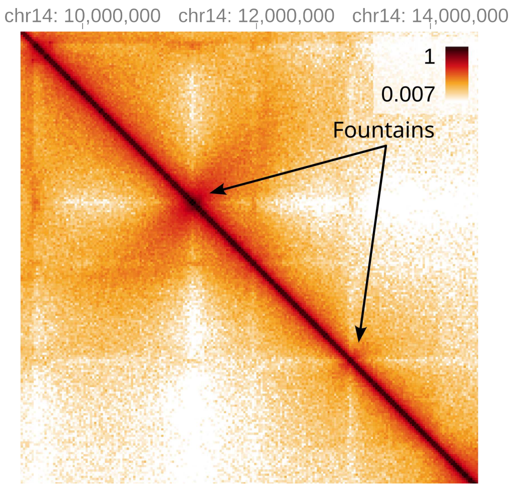


Figure 1.2: Micro-C heatmap showing stripes and loops over a 1 Mbp region from the human genome (left). Hi-C heatmap showing fountains over a 5 Mbp region from the mouse genome (right).

Left heatmap was generated from dataset 4DNFI9GMP2J8 at 5 kbp resolution [28, 30].

Right heatmap was generated from dataset GSE199059 at 25 kbp resolution [48, 49].

TADs are not the only feature visible at the kilo- to mega-base scale in Hi-C data, as at this scale we also find extrusion loops, stripes, and fountains.

Loops, often also known as dots, represent a pair of loci interacting at a high frequency. Most of the loops observed in Hi-C data arise as a consequence of loop extrusion bridging a pair of convergent CTCF binding sites (see Section 1.3 for more details). However, thanks to the advent of Micro-C and more recently Region Capture Micro-C (RCMC), we now know that loops can also arise from stable enhancer-promoter and promoter-promoter interactions [28, 50, 51]. Stipes or lines, are another pattern that arises due to cohesin interacting with CTCF during loop extrusion.

More specifically, stripes form when cohesin extrusion is stalled only on one side due to collisions with a specific CTCF site that is bound by CTCF, but is free to extrude on the other side. When this is the case, genomic regions that are adjacent in 1D space are progressively brought together with the CTCF site that is stalling cohesin, thus forming a stripe enriched of interactions. Similarly to what has been described for loops, Hsieh et al. [50] show that a large number of small-scale stripes (10–50 kbp) link promoter-promoter and enhancer-promoter sites, with weak loops forming at intersections between stripes.

Another pattern that has recently been described in *C. elegans*, *Zebrafish* and mouse embryonic stem cells are extrusion fountains (also known as plumes or jets) [49, 52–55]. Fountains appear as "patterns of contacts that emanate from a single genomic locus and broaden with distance from the diagonal", and have been hypothesized to be sites of preferential cohesin load [55].

In conclusion, DNA loop extrusion (reviewed in Section 1.3) seems involved in the formation of several 3D structures characterized by distinct appearances in the Hi-C matrix, namely loops, stripes, and fountains. These structures form as a consequence of either cohesin stalling or preferential loading at given genomic loci.

1.2.4 Compartments

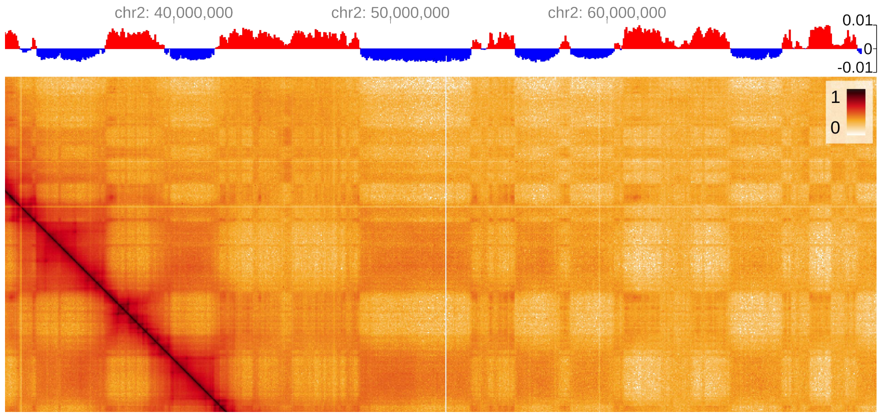


Figure 1.3: Hi-C heatmap showing A/B compartments over a 35 Mbp region from the human genome. Alternating between A/B compartments is highlighted by the barplot on top of the heatmap, showing the first Principal Component (PC) computed as outlined in Section 1.2.4. The barplot shows A and B compartment domains in red and blue, respectively. Regions in the heatmap corresponding to A/B compartments exhibit the characteristic checkerboard pattern, showing preferential interactions between compartment domains of the same type.
Heatmap generated from dataset ENCFF447ERX at 50 kbp resolution [56].

Chromatin compartments were initially described by Lieberman-Aiden et al. [57] as genomic domains forming a checkerboard pattern on the Hi-C map.

Compartments are identified using Principal Component Analysis (PCA) of normalized Hi-C contact matrices. First, the Hi-C matrix (i.e., the observed matrix) is normalized by computing the ratio with the expected matrix. Next, the Pearson correlation of all pairs of rows and columns of the observed/expected matrix is computed. Finally, the first Principal Component (PC) is computed by PCA, and regions with positive and negative signs are assigned to A and B compartments, respectively [57–59]. Note that it is often required to re-orient the PC, such that the PC vector is positively correlated with GC content, gene content, or any other suitable metric that is associated with open chromatin [59].

The method described above is the original method to call A/B compartments introduced by Lieberman-Aiden et al. [57], however, over the years this method has been modified to improve accuracy and computational performance to allow computing A/B compartments at finer resolutions [60–63]. However, almost every modern approach still relies on the computation of PCA to assign genomic bins to the A or B compartment, the main differences reside in the numerical method used to compute the PCA and in the preprocessing and normalization steps that preceded PCA computation [59]. One exception is CscoreTool, which instead frames the problem as a Maximum Likelihood Estimation (MLE) optimization problem, computing the probability that a genomic bin belongs to the A compartment [64].

Compartments are associated with different chromatin states, with A and B compartments being associated with open and closed chromatin, respectively. The first studies reported that compartment intervals, that is stretches of bins annotated with the same compartment, were in the order of 1 to several megabasepairs [65]. It later became clear that these numbers were for the most part incorrect due to resolution and sequencing depth limits.

A recent study by Harris et al. [66] reported that "chromatin alternates between A and B compartments at kilobase scale". These findings were made possible by algorithmic advances in PCA computation as well as an ultra-deep Hi-C map, with over 33 billion interactions. The authors report that the median compartment interval is only 12.5 kbp, and that virtually all active enhancers and promoters localize in the A compartment. Furthermore, several genes exhibit discordant compartmentalization, whereby gene TSS and Transcription Termination Site (TTS) belong to different compartments. This behavior becomes more common the longer a gene is. Goel et al. [51] decided to take a different approach, and instead of generating an ultra-deep, genome-wide Hi-C map, they generated an ultra-deep Micro-C map over three small regions of interests using RCMC (reviewed in Section 2.2.1.7). The authors describe localized regions of highly focal and nested interactions that give rise to a checkerboard pattern at a very fine scale. They name these domains microcompartments, as they exhibit properties similar to A/B compartments, but at a much finer scale. Microcompartments have been found to often connect enhancers and promoters, and their presence is independent of loop extrusion and transcription.

One of the critiques to the two-state A/B compartment model is that two states may be inadequate to recapitulate chromatin's compartmental behavior. For this reason, several studies introduced the concept of subcompartments. The first of such studies was published in 2011 by Yaffe et al. [67], where the authors used k-means clustering to assign genomic bins to three different states.

In 2014, thanks to the improvements introduced by *in-situ* Hi-C, Rao et al. [38] used unsupervised clustering to classify chromatin across 6 different subcompartments. Over the last decade, several other approaches have been proposed to identify subcompartments. These tools can be broadly classified into three categories: tools that annotate subcompartments de-novo from inter-chromosomal interactions, such as GHMM, SCI and SNIPER [38, 68, 69]; tools that do de-novo annotation from intrachromosomal interactions, namely Calder, dcHiC, and MOSAIC [70–72]; tools that predict subcompartment labels from existing annotation based on epigenetic marks, for example, PyMEGABASE and SCI-DNN [68, 73]. Overall, most tools identify between 4 and 9 distinct subcompartment types [59]. Calder's authors find "that eight subcompartments were sufficient to capture significant changes of [these] histone modifications in all cell lines". The authors also report that subcompartments A.1.1, A.1.2, and A.2.1 are associated with active transcription marks, while repressive marks are more common in B.1.1 and B.1.2 subcompartments [70].

In conclusion, compartments are a layer of chromatin organization distinct from TADs whose mechanism of formation possibly counteracts that of TADs [74]. Compartment domains were originally thought to be much larger than TADs, however, recent studies using ultra-deep Hi-C maps revealed that the median compartment domain size is in reality much smaller than TADs [51, 66]. Both compartments and subcompartments types show distinct enrichments and depletions of epigenetic markers associated with active, inactive, and heterochromatic chromatin, thus making compartments and subcompartments a good proxy for chromatin states.

1.2.5 TAD cliques

TAD cliques are defined as a "subset of k TADs (with $k \geq 3$) which are fully connected - that is, which all interact pair-wise in the Hi-C data" [75, 76].

To identify TAD cliques, Paulsen et al. [76] start by identifying pairs of TADs whose interactions in 3D space are statistically significant using the NonCentral HyperGeometric (NCHG) distribution. Next, TADs are represented as a uni-directed graph, where TADs with statistically significant interactions are connected by an edge. TAD cliques are then identified using the Bron–Kerbosch algorithm [76, 77] as subsets of fully connected nodes in the TAD graph. The largest clique in which a TAD participates is known as the maximal clique size for the given TAD, or more colloquially as the TAD clique size.

TAD cliques have been shown to be enriched in B compartments and to exhibit overall low levels of gene expression. Furthermore, the larger a TAD clique is, the greater the overlap with B compartments and Lamina Associating Domains (LADs), suggesting that TAD cliques may stabilize heterochromatin at the nuclear periphery [76, 78] and Paper III.

Paulsen et al. followed TAD cliques during adipose cell differentiation, and found that TAD cliques sizes grow and shrink during differentiation. The authors also show that gene expression changes as expected following changes in clique sizes. That is clique growth is associated with transcription repression while clique shrinkage is associated with an increase in transcription. Ali et al. [78] found an enrichment of Long Long Interspersed Nuclear Element (LINE) transposons in TADs taking part to cliques compared to TADs not part of cliques. In the same study, the authors reported a depletion in convergent CTCF binding sites in TADs part of TAD cliques. CTCF depletion was positively correlated with TAD clique size.

TAD cliques have initially been described in human and mouse [76, 78, 79] and Paper III, but have later been identified in several other species, such as *Drosophila* [80], cotton [81] and giant panda [82], suggesting that TAD cliques are an emerging, wide-spread level of chromatin organization.

1.2.6 Chromosome territories

Chromosome Territories (CTs) were the first 3D feature of the genome to be described, with their first description by Carl Rabl [83] dating back to 1885.

After being abandoned for several decades, the concept of CTs was brought back in the late 1980s and early 1990s, when CTs were directly visualized through chromosome painting [84].

CTs are defined as discrete territories in the cell nucleus that are occupied by individual chromosomes during interphase. In 2000, Sun et al. [85] showed that chromosomes assume nonrandom radial nuclear positions in approximately spherical nuclei, with larger chromosomes preferentially located at the nuclear periphery and smaller chromosomes preferentially located at the nuclear core. For flat-ellipsoid nuclei, chromosome radial positioning seems to instead be driven by gene content, with gene-rich chromosomes located towards the nuclear core while gene-poor chromosomes are preferentially located closer to the nuclear periphery [86] Furthermore, different tissues have been shown to exhibit different CTs [87].

Neighbor chromosome territories have been shown to intermingle, even though the extent of intermingling varies among studies, with 1–20% of the nuclear volume consisting of intermingled sequences from different chromosomes [88, 89]. Strikingly, chromosomal rearrangements appear to correlate with the degree of chromosome intermingling [89]. Chromosomal regions of intermingling have also been found to be enriched with RNA Polymerase (RNAP)II. This enrichment was shown to be resistant to temporary perturbations to transcription. However, intermingling regions were found to be enriched in both active and repressive marks [90]. For a thorough review of the current state of CTs, readers are referred to the review by Szczepińska et al. [91]

Taken together, these findings describe CTs as important features of the 3D genome that contribute to chromatin folding at a level above TADs and chromatin compartments. CTs have an important role in determining the radial positioning of chromosomes and genes in the nucleus. Furthermore, chromosome intermingling has been described to have profound effects on the frequency of translocations and gene expression regulation.

1.2.7 Closing remarks

In this section, I introduced some of the most prominent structures involved in chromatin folding as viewed using Hi-C.

At the 150–200 bp scale, chromatin is folded into nucleosomes. Recent studies [27, 29] suggest that groups of nucleosomes co-localize in 3D space forming nucleosome clutches and CND. Next, at the 100 kbp–5 Mbp scale, chromatin folds into TADs [32, 33], which are for the most part a result of a process known as DNA loop extrusion. TADs are often found nested within one another. Furthermore, in Hi-C data, TAD boundaries are often enriched by other patterns, such as stripes and dots, which are a consequence of the partial or complete stall of DNA loop extrusion [28, 50]. Another level of chromatin folding are A/B compartments and subcompartments [38, 57]. For a long time, it was believed that chromatin compartments occurred at a scale significantly larger than TADs. However, recent evidence suggests that A/B compartments are alternating at a much finer scale than what was previously thought [66]. Paulsen et al. [76] showed that groups of TADs can organize into TAD cliques, forming domains that may help stabilize heterochromatin at the nuclear periphery. At the chromosome level, chromosomes are organized into chromosome territories, which are defined as discrete territories in the cell nucleus that are occupied by individual chromosomes during interphase [84].

In this thesis, I am focusing on DNA-DNA interactions, as these kinds of interactions have been at the core of my research. However, as it is well known, the nucleus is not only filled with DNA but also with complexes and structures made of proteins and lipids, some of which play a crucial role in chromatin folding.

The nucleus is delimited by the nuclear envelope, which is a double layer membrane separating the nucleus from the cytoplasm in eukaryotes. The nuclear envelope consists of two membranes, an Inner Nuclear Membrane (INM) and Outer Nuclear Membrane (ONM). The INM and ONM are fused in correspondence of Nuclear Pore Complexes (NPCs), which are protein complexes forming channels across the nuclear envelope, and are responsible for regulating the nucleocytoplasmic transport. Metazoans have an additional structure connecting the nuclear envelope to the peripheral chromatin named nuclear lamina. The nuclear lamina is a meshwork of A- and B-type lamins, and is involved in numerous processes, including DNA replication, transcription, and chromatin organization. As dissecting the role and function of these structures is outside the scope of this thesis, readers are referred to the following articles reviewing the nuclear envelope [92, 93], nuclear lamina [94–96], and NPC [97, 98].

Moving to the interior of the nucleus, besides DNA, RNA, and proteins, the nucleus contains several different membrane-less structures known as nuclear bodies. Nuclear bodies cover numerous different functions. For example, the nucleolus is the main responsible for Ribosomal RNA (rRNA) synthesis and ribosome assembly, while Cajal bodies are involved in several processes related to RNA processing. While several of the nuclear bodies have profound effects on chromatin folding, their description as well as the description of the resulting protein-DNA interactions is outside the scope of this thesis. For this reason, readers are referred to the following reviews on the topic [99–106].

1.3 DNA loop extrusion

DNA loop extrusion is established as one of the important processes involved in chromatin folding in interphase nuclei of metazoans and other eukaryotes. In metazoans, loop extrusion mediated by cohesin is responsible for the formation of TADs (reviewed in Section 1.2.2), which have been implied to play a role in several biological processes, such as gene expression, DNA repair, and replication. In this section, I will review the current state of the art on the process of DNA loop extrusion and relate loop extrusions to other biological processes. The section ends with a review of available in-silico models simulating loop extrusion. I will focus on DNA loop extrusion by cohesin in metazoans during interphase, even though loop extrusion is also carried out by condensin I and condensin II on mitotic chromosomes in other organisms.

1.3.1 Mechanism and key players

The process of DNA loop extrusion, as it is currently understood, involves two key players, namely cohesin and CCCTC-binding Factor (CTCF). Cohesin is a protein complex belonging to the Structural Maintenance of Chromosomes (SMC) protein family and is the main responsible for the formation of DNA loops in interphase nuclei. Cohesin's core complex consists of proteins SMC1, SMC3 and RAD21 (SCC1) and STAG1/2 (SCC3) in humans (see Figure 1.4). Furthermore, loop-extruding cohesin associates with NIPBL/MAU2 (SCC2/SCC4) complex or the PDS5A/B protein. CTCF is a transcription factor implicated in numerous processes, such as transcriptional regulation, insulation, V(D)J recombination, and chromatin folding. The CTCF protein consists of three domains: an N-terminal domain, which is used to interact with cohesin, a C-terminal domain and a central domain with 11 zinc fingers, which is used to recognize CTCF's binding site [107, 108].

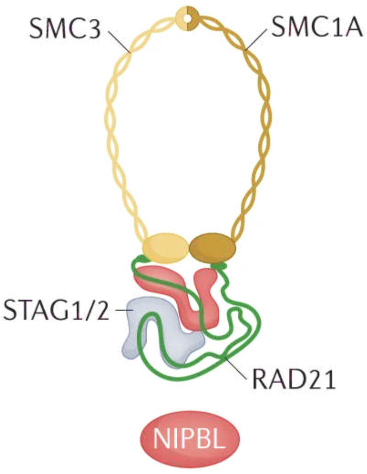


Figure 1.4: Diagram of cohesin's structure.
Taken from Figure 2 from *New insights into genome folding by loop extrusion from inducible degron technologies* [47].
Reproduced with permission from Springer Nature.

At its core, loop extrusion by cohesin consists of cohesin being loaded onto DNA and growing a loop of DNA until cohesin is released, or until one or more extrusion barriers (such as CTCF) are met, in which case loop extrusion is stalled. This process of progressive loop extrusion continues throughout interphase to continuously fold chromatin into TADs.

The mechanism through which cohesin extrudes DNA has been debated for a long time. First, the "tethered-inchworm" model [109] was proposed, where cohesin was hypothesized to walk along the DNA using its ATPase heads. This model was followed by the "pumping" model [110], which proposed that DNA is initially bound to cohesin's hinge, then translocated towards SCC1 by conformation changes induced by ATPase activity. Next, the "scrunching" model [111] was proposed, where cohesin extrudes DNA by alternating between straight and bent coiled-coil conformation. More recently the "swing-and-clamp" model was proposed by Bauer et al. [112]. This model, which is depicted in Figure 1.5, starts with DNA bound to the hinge of cohesin-NIPBL complex, where NIPBL is also associated with cohesin's hinge. Next, cohesin's coils spontaneously align and fold, translocating DNA bound to the hinge to the SMC3 head through clamping by NIPBL. In this conformation, when ATP is present at cohesin's ATPase heads, ATPase engagement forces the hinge away from SMC3, while DNA is kept in proximity of SMC3 by NIPBL clamping. Finally, ATP hydrolysis causes the ATPase heads to disengage, and the cohesin-NIPBL complex is brought back to its initial state, with a new DNA segment bound to cohesin's head.

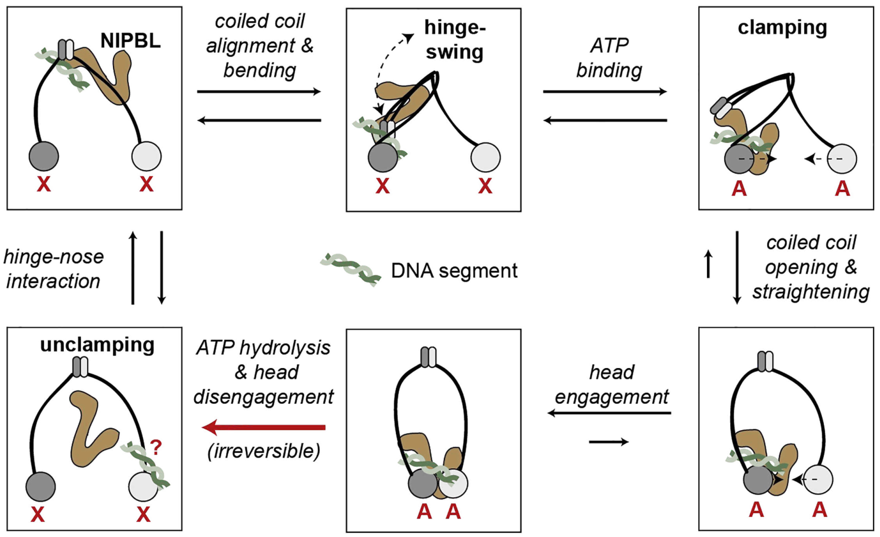


Figure 1.5: Diagram of the "swing-and-clamp" loop extrusion model. Figure 7F from *Cohesin mediates DNA loop extrusion by a "swing and clamp" mechanism* - 10.1016/j.cell.2021.09.016 [112].
Figure licensed under the CC BY-NC-ND 4.0 license.

The process of DNA loop extrusion, as it is currently understood, develops as follows. Loop extrusion begins with cohesin being loaded onto DNA in a non-sequence-specific way that is not currently understood, with preferential loading onto accessible chromatin, especially at enhancer sites [113]. This requires cohesin to be associated with NIPBL [114, 115]. Cohesin is unloaded from DNA through interactions with WAPL and WAPL-PDS5 [116–118]. NIPBL is not only required for cohesin loading onto DNA but also for active loop extrusion, as cohesin's ATPase activity is observed only in the presence of NIPBL [119, 120]. Cohesin processivity can be affected by several factors. On the one hand, cohesin can be acetylated by ESCO1/2 [121] and deacetylated by HDAC8 [122]. NIPBL competes with PDS5A/B for cohesin binding, and when cohesin is acetylated, binding to PDS5A/B is favored. On the other hand, when cohesin is in its deacetylated form, binding with NIPBL is favored [123]. Given that cohesin ATPase activity is observed only when cohesin is bound to NIPBL, acetylated cohesin bound to PDS5A/B cannot actively extrude DNA, and thus loops are stabilized. Furthermore, acetylated cohesin appears to be protected from unloading by WAPL-PDS5, thus increasing the lifetime of already stable loops [123].

Loop extrusion can be stalled by interactions with several molecules acting as molecular barriers. First and foremost, loop extrusion can be stalled by collisions with the CTCF protein [124, 125]. CTCF recognizes and binds a non-palindromic DNA motif. Thus, motif direction determines the positioning of CTCF's *N-terminus*. Crucially, cohesin is unable to extrude through CTCF when its *N-terminus* is facing against cohesin's extrusion direction [126] It has been shown that this is due to molecular interactions between cohesin and RAD21/STAG2 and CTCF's *N-terminus* [108, 126].

In recent years, several studies showed that loop extrusion can be stalled, or at least slowed down by Minichromosome Maintenance (MCM) complex and by transcribing RNAP [113, 127, 128]. This behavior is at odds with the observation that loop extruding cohesin can bypass obstacles as large as 200 nm [129]. The two findings can be reconciled by hypothesizing that what stalls cohesin is not molecule size, but rather specific interactions between cohesin and molecules acting as loop extrusion barriers. Finally, a recent study challenged the view that CTCF is a static barrier to loop extrusion by showing that CTCF's blocking ability depends on DNA tension, and is greater when DNA tension is high [130].

Overall, loop extrusion by cohesin establishes itself as a highly dynamic process that continuously compacts DNA of interphase nuclei in metazoans. The fact that current simulations show good agreement with Hi-C experiments only in selected regions suggests that loop extrusion, as is currently understood, is not the only process involved in the formation of loops, stripes and TADs [131, 132].

1.3.2 How does loop extrusion relate to other processes?

This section focuses on the functional role of DNA loop extrusion and how this relates to other biological processes.

Shortly after the discovery of TADs [32, 33], it was hypothesized that their main function, and thus the main function of interphase loop extrusion, was that of constraining enhancer-promoter interactions, facilitating interactions of elements within the same TAD and obstructing all other interactions. This hypothesis was supported by evidence that enhancer-promoter interactions are indeed much more frequent within TADs than between TADs [133]. However, this model was quickly put into discussion by several studies showing how depleting cohesin or CTCF had minimal impact on gene expression [45, 125, 134, 135]. Based on this evidence, the model was revised such that loop extrusion is mostly important to establish enhancer-promoter interactions during differentiation or in response to stimuli, but that once these interactions are established, loop extrusion is no longer necessary [136]. It has also been proposed that the impact of loop extrusion on expression levels may depend on several other factors, such as individual gene susceptibility to enhancer-mediated activation, the linear distance between enhancers and the genes they regulate, and the proximity of gene promoters to TAD borders [136].

In conclusion, our current understanding of DNA loop extrusion in relation to gene expression is that loop extrusion may not be necessary for the short-term maintenance of enhancer-promoter interactions, but it is likely crucial to establish and maintain them in the long term. The relationship between DNA loop extrusion and gene expression is analyzed in more detail in this review by Karpinska et al. [136].

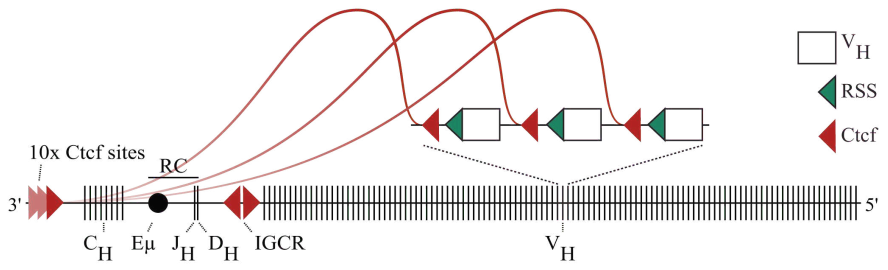


Figure 1.6: Diagram of V(D)J recombination aided by DNA loop extrusion. Figure 2b from *How DNA loop extrusion mediated by cohesin enables V(D)J recombination* [137].
Figure licensed under the CC BY NC ND 4.0 DEED license.

In recent years, loop extrusion was found to be implicated in V(D)J recombination in pro-B cells.

V(D)J recombination is a type of somatic recombination that occurs during B cell differentiation and is responsible for the highly variable repertoire of antibodies produced by the vertebrate immune system [138]. In this context, loop extrusion has been thoroughly studied at the mouse immunoglobulin heavy-chain locus Igh [139–144].

Figure 1.6 shows a diagram of how loop extrusion is hypothesized to aid V(D)J recombination. The *Igh* locus is fairly large, measuring 2.75 Mbp, and consists of 13 D_H segments, 4 J_H segments, and 113 V_H segments, with over 90% of its size due to V_H segments. As shown in Figure 1.6, this region harbors 11 CTCF sites at the 3′-end, all pointing towards the 5′-end, as well as 125 sites interspersed in the *Igh* locus, all of which are pointing towards the 3′-end. Due to the considerable length of the *Igh* locus, recombination involving different segments from this locus requires bridging over long distances. Several studies showed that cohesin and loop extrusion are likely at play here [139–144].

To generate a highly diverse repertoire of antibodies, it is crucial that all V_H segments are allowed to participate in the V(D)J recombination. Loop extrusion mediated by cohesin can facilitate this, as it can be loaded anywhere on the *Igh* locus, and can bridge one of the CTCF sites near the recombination center (RC) to any of the CTCF sites interspersed in the *Igh* locus, depending on where cohesin is loaded and on which CTCF sites are bound by CTCF at a given time [137].

Together, these observations suggest that DNA loop extrusion is one of the crucial processes required for the proper functioning of the vertebrate immune system.

Another process where DNA loop extrusion is proving to be crucial is that of DNA Double Strand Break (DSB) repair.

DNA DSBs can be caused by several endogenous and exogenous factors, such as radiation, drugs, transcription and replication stress [145–148]. DNA DSBs formation induces the assembly of DNA-Damage Response (DDR) foci in correspondence with damaged sites, which are responsible for detecting and repairing DNA DSBs [149]. Part of DDR's damage response involves recruiting the ATM kinase to mark the damaged region with γH2AX, which eventually leads to the recruitment of other proteins involved in the DDR mechanism, such as 53BP1 and MDC1 [150]. It is crucial that DDR foci are assembled as rapidly as possible to reduce the chance that the DNA ends resulting from a DSB event drift away, which could render the DSB repair impossible.

Arnould et al. [151] show that TADs produced by DNA loop extrusion mediated by cohesin are functional units of the DNA damage response, and play an important role in establishing γH2AX-53BP1 domains around the DSB site. The authors show that upon DSB damage, cohesin is recruited by the ATM kinase on both sides of the DSB site to carry out divergent, one-sided loop extrusion, reeling DNA towards the damage site. This enables the ATM kinase to phosphorylate H2AX nucleosomes over megabase-sized domains within 10–30 minutes. Crucially, phosphorylation spread is limited to the TAD where the DSB occurred. Finally, ATM can also phosphorylate cohesin, leading to an increase in loop strength, suggesting that cohesin phosphorylation may modify properties of the cohesin complex, such as extrusion speed and DNA binding stability.

In summary, DNA loop extrusion facilitates DNA DSB repair by promoting rapid phosphorylation of H2AX-containing nucleosomes surrounding the DSB site. This phosphorylation is crucial for the timely establishment of the DDR foci.

1.3.3 In-silico models of DNA loop extrusion

Given that Paper I presents MoDLE (Modeling of DNA Loop Extrusion), a high-performance stochastic model of DNA loop extrusion, it felt appropriate to provide an overview of efforts preceding and following MoDLE. One key distinction between MoDLE and most of the models introduced in this section, is that MoDLE simulates loop extrusion as a stochastic process, while most other models utilize different types of Molecular Dynamics (MD) simulations.

A mechanism for chromatin organization by DNA loop extrusion was initially proposed by Alipour et al.[152] in 2012. Alipour et al. used lattice simulations modeling a single loop of DNA, where a bi-directional loop extruder was repeatedly loaded onto the DNA. Their simulation showed that this configuration was sufficient to generate large loop domains as long as Loop Extruding Factor (LEF) processivity was sufficiently high (i.e., as long as LEFs were given enough time to traverse loop domains before being released). In the same paper, the authors propose that the formation of several loop domains along whole chromosomes may be regulated by molecular barriers that act as boundaries to loop domains. These barriers could either halt loop extrusion or promote the dissociation of LEFs. It should be noted that Alipour et al. proposed this mechanism for the compaction of mitotic chromosomes, however many of the properties of the proposed mechanism have later been shown to also apply to DNA loop extrusion by cohesin on interphase chromosomes.

Between 2013 and 2015, two studies simulated loop extrusion in the context of mitotic chromosomes [153, 154].

However, it was not until late fall 2015 that the first MD simulations of interphase DNA loop extrusion became available. On July 27, 2015, Sanborn et al. [155] submitted an article titled “Chromatin extrusion explains key features of loop and domain formation in wild-type and engineered genomes” to PNAS. The article was published on September 17, 2015. Shortly after, on August 14, 2015, Fudenberg et al. [131] published a preprint titled “Formation of Chromosomal Domains by Loop Extrusion”, which was later published on Cell reports on May 31, 2016. The two studies introduce similar models for DNA loop extrusion (described below) and conclude that a model based on DNA loop extrusion can recapitulate known features of Hi-C matrices, such as the observed contact probability function ($P(s) \propto s^{-\gamma}$), interaction enrichment within TADs, and loop formation at convergent CTCF binding sites. However, while Sanborn et al. find that the loop extrusion model can recapitulate experimental interaction probability curves $P(s)$, regardless of simulation parameters, Fudenberg et al. report that the level of agreement with experimental evidence does depend on the parameter choice. Another point of contrast between the two studies is the level of agreement between simulated and experimental Hi-C maps: Sanborn et al. report that the outcome of all 13 genomic engineering experiments were successfully predicted by simulations, while Fudenberg et al. find that the level of agreement between simulated and experimental Hi-C matrices varies depending on the region that is being compared.

Fudenberg et al. [131] simulate loop extrusion over a 10 Mbp region subject to Langevin dynamics. The simulation involves binding and release of LEFs to and from DNA. LEFs are simulated as particles that symmetrically translocate along the genome, forming bonds between the two monomers at the base of DNA loops. As LEFs translocate along the DNA, bonds between monomers at the base of the loop are broken and re-formed such that the bond always connects the two monomers at the base of DNA loops. Extrusion barriers are placed at fixed locations and can halt loop extrusion when correctly oriented, similar to CTCF's behavior *in vivo*. Furthermore, LEFs are not able to bypass each other, resulting in stalling of loop extrusion.

Sanborn et al. [155] model DNA molecules using Langevin dynamics as homopolymers where each monomer represents 1 kbp of DNA. The loop extrusion model proposed by the authors is very similar to that of Fudenberg et al. The main difference between the two models is that in Sanborn's model, collisions between LEFs cause one of the LEFs to dissociate from DNA, while in Fudenberg's model LEF collision simply cause loop extrusion to stall until one of the LEFs is released. Fudenberg et al. use their model to predict the effects of cohesin and CTCF depletion and find good agreement between in-silico and experimental heatmaps. In contrast, Sanborn et al. applied their model to predict the effect of local perturbation of CTCF binding sites by genome editing and found that their model could recapitulate the impact of the genome edits on the resulting Hi-C matrix.

The models proposed by Fudenberg et al. [131] and Sanborn et al. [155] were used to simulate loop extrusion by several other studies. For example, in September 2017, Schwarzer et al. [135] used Fudenberg's model to simulate NIPBL depletion and find good agreement with experimental data. Shortly after, on October 2017, Rao et al. [134] extended Sanborn's model by using a block copolymer instead of a homopolymer to simulate phase-separation by A/B compartments and loop extrusion using a single model. The authors use their model to predict the effects of cohesin depletion on the resulting Hi-C matrix and found good agreement between experimental and simulated data. Crucially they show that compartment strength is weakened by the process of DNA loop extrusion, suggesting that the two processes antagonize each other.

In 2018, Nuebler et al. [156] extended Fudenberg's model to use a block copolymer to simulate phase-separation by A/B compartments (similarly to what was done by Rao et al. [134]). Neubler et al. confirmed that phase-separation induced by A/B compartments is weakened by DNA loop extrusion. Furthermore, the authors use their model to predict the effect of NIPBL, CTCF, and WAPL depletion and find good agreement with experimental evidence. Finally, Neubler et al. showed that the active nature of loop extrusion is required to contrast compartmentalization, as simulations with static loops show no effect on compartment strength, and that compartment weakening is proportional to LEF extrusion speed.

In the same year, Pereira et al. [157] combined the loop extrusion model with a model based on multivalent interactions between DNA and transcription factors. The authors find that neither model can on their own accurately predict features of Hi-C data, as the loop extrusion model fails to recapitulate large-scale interactions while the transcription factor model fails to model local domain patterns. However, when the two models are combined, the resulting model is capable of reproducing features observed in Hi-C data at all scales.

Later in the same year, Pereira's model was integrated into the Hip-Hop model developed by Buckle et al. [158]. The Hip-Hop model integrates several different types of data, including epigenetic marks (H3K27ac), chromatin accessibility (ATAC-seq), and extrusion barrier location and direction (CTCF). This model is capable of simulating interactions driven by A/B compartmentalization as well as DNA loop extrusion.

In 2020, another paper using simulations based on Fudenberg's model was published by Banigan et al. [159]. In this paper the authors dissect different regimes of loop extrusion: two-sided, one-sided, semi-diffusive, and switching extrusion. Overall the authors find that one-sided and semi-diffusive loop extrusion cannot recapitulate dots found in Hi-C matrices unless LEFs are almost exclusively loaded at CTCF sites. In contrast, two-sided and switching loop extrusion can recapitulate all features of interphase Hi-C matrices. For the switching model, this is the case only as long as the switching rate is sufficiently fast. The authors also show that a mix of approximately 50% one- and two-sided extrusion can recapitulate interphase Hi-C matrices.

In early 2021, Xi et al. [160] developed a mathematical model to predict DNA loops observed in ChIA-PET CTCF. The model combines the probability of CTCF binding a given site with the distance-dependent decay of contact frequencies and loop competition around CTCF pairs to predict the probability of CTCF interaction. The authors show that the model can predict the effects of ΔWAPL on ChIA-PET data.

Later in the same year, Zhang et al. [127] adapted the original Hip-Hop model to preferentially load cohesin at TSS occupied by RNAPII. Results from these simulations complemented the main findings of this study, where the authors present data suggesting that RNAPII recruits NIPBL and WAPL following mitotic exit.

In early 2022 Gabriele et al. [161] simulated loop extrusion using a model based on Fudenberg's. The authors simulated loop extrusion around the *Fbn2* locus under different regimes, including ΔCTCF, ΔRAD21, and ΔWAPL. Consistent with microscopy and Micro-C data, Gabriele et al. simulations find that the *Fbn2* TAD is rarely in a state where it is fully extruded. Later in the same year, Dequeker et al. [162] used MD simulations to predict the effects of MCM on loop extrusion during G1 phase. The authors report that MCMs act as semi-permeable barriers to DNA loop extrusion by cohesin.

In Nov 2022, we published Paper I to introduce MoDLE, a high-performance stochastic model of DNA loop extrusion interactions (reviewed in Section 4.1). MoDLE was the first model capable of simulating DNA loop extrusion interactions genome-wide in a matter of minutes. MoDLE models CTCF barriers using a two-state (bound and unbound) Markov process where transition probabilities are computed from the average occupancy of a given CTCF binding site. MoDLE provides a wide range of parameters that can be used to tweak numerous aspects of DNA loop extrusion simulations, such as extrusion barrier permeability, reciprocal LEF permeability, extrusion speeds, and much more. Finally, despite not simulating the dynamics of the DNA polymer, MoDLE can generate interaction matrices resembling Micro-C and Hi-C matrices, suggesting that full-fledged MD simulations may not be necessary to simulate loop extrusion interactions. Similarly to what was reported by Fudenberg et al. [131], the level of agreement between simulated and experimental heatmaps depends on the regions that are being compared. When comparing regions where loop extrusion appears to be the main process driving chromatin conformation, interaction matrices generated by MoDLE and Micro-C exhibit high levels of agreement.

In late 2022, a paper by Banigan et al. [128] describing the interplay between loop extrusion and transcription was published. In this paper, the authors simulate loop extrusion using a model based on Fudenberg's, where RNAPs act as semi-permeable, moving barriers to loop extruding cohesin. More specifically, in simulations, head-on collisions between cohesin and RNAP result in cohesin being slowly pushed back by RNAP. This is justified by RNAP's stall force, which is much greater than that of cohesin. In case of collisions between cohesin and RNAP moving in the same direction, cohesin continues extruding as fast as RNAP allows. This is justified by the fact that cohesin's translocation speed is much greater than the RNAP's one. Crucially, in both cases, cohesin is allowed to bypass RNAP with a relatively small probability. Simulations confirm experimental evidence showing that RNAP acts as semi-permeable, moving barriers. Furthermore, simulations were used to test whether preferential cohesin loading at TSS could lead to better predictions. However, the authors find that this does not improve simulation accuracy, and suggest that cohesin's enrichment at TSS is likely due to collisions with RNAP paused at TSS before transcription initiation.

In 2023, Zhang et al. [113] used a model based on the Hip-Hop model [158] to simulate the effect of RNAPII on loop extrusion similarly to what was done by Banigan et al. [128]. In this study, the authors use simulations to support two findings, namely that RNAPII antagonizes loop extrusion and that accumulation of interactions between enhancers and promoters requires RNAPII.

1.3.4 Closing remarks

In this section, I reviewed the state of the art on the process of DNA loop extrusion from a biological and computational perspective.

While some aspects of DNA loop extrusion are clear, such as the involvement of cohesin and CTCF, some of the details are not yet resolved (e.g., whether two cohesins are able to traverse one another *in vivo*) and others are still being debated (e.g., whether cohesin preferentially loads at TSS).

Overall, the models introduced by Fudenberg et al. [131] and Sanborn et al. [155] have stood the test of time, and have proven fundamentally correct. However, recent developments such as Davidson et al. [130], Banigan et al. [128], Zhang et al. [113], and Dequeker et al. [162] provide a more nuanced view of this process, showing how loop extrusion is affected by biophysical properties of chromatin such as DNA tension, as well as by other processes such as transcription and DNA replication. It is reassuring to see that in-silico simulations are slowly becoming just another tool in the researcher's toolbox, as exemplified by some of the studies reviewed in this section [113, 127, 128, 134, 135, 155, 161, 162] where *in vitro* and *in vivo* experiments have been supported or guided by insights gained through in-silico modeling.

Given the level of agreement between experiments and simulations [131, 155, 156, 158, 159], Paper I, it is clear that our understanding of DNA loop extrusion is substantially correct. That being said, a few studies [50, 131], including Paper I, reported how the level of agreement is not uniform along the entire genome. This suggests that DNA loop extrusion, in its current form, is likely not the only process responsible for chromatin folding at the kilobase and megabase scale. It will be interesting to witness how the field evolves to answer these and other questions, in the quest of unraveling the process of DNA loop extrusion.

Chapter 2

Experimental methods to study the 3D Genome

2.1 Introduction

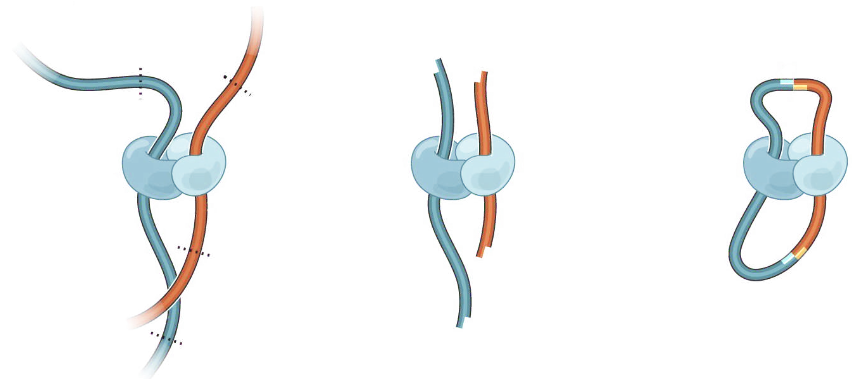
This chapter introduces the most important techniques that are being used to characterize the chromatin 3D structure. Even though I have not directly used most of these techniques as part of my research, many of these methods have been used to first describe hallmark 3D structures in individual or ensemble of cells, such as chromosome territories (Section 1.2.6), A/B compartments (Section 1.2.4), TADs (Section 1.2.2) and loops (Section 1.2.3). For this reason, I consider it important to acknowledge and introduce at least the most important of these techniques, as well as some very recently developed techniques. Methods are classified into imaging and sequencing methods, even though some of the sequencing methods use PCR or microarrays instead of DNA sequencing. Methods are introduced in rough historical order such that methods coming later in the chapter build on methods that have already been introduced. At the end of the chapter, I will briefly summarize and contrast the most relevant techniques and outline some of the considerations that came to mind while writing this chapter.

2.2 Sequencing-based methods

Sequencing methods listed in this section rely on PCR, microarrays, or Next-Generation Sequencing (NGS) to characterize the 3D structure of genomes. These methods do not measure the location of one or more loci in 3D space, but rather whether two or more loci co-localize in 3D space, i.e., whether they are within a certain distance of each other. The first method to study the 3D structure of genomes using a sequencing approach was 3C, which was developed in 2002 by Dekker et al. and allowed detecting whether a pair of loci interact in 3D space [163].

2.2.1 Chromosome Conformation Capture methods

Chromosome Conformation Capture methods all descend from the original 3C method and have several steps in common. First, cells are cross-linked (usually by treatment with formaldehyde), Figure 2.1a. This has the effect of freezing chromatin in its current 3D structure by forming protein-protein and DNA-protein covalent bonds. Next, DNA is digested using one or more restriction enzymes (or sometimes other types of enzymes such as DNase I and MNase), Figure 2.1b. Crucially, DNA digestion is followed by dilution and a proximity-based ligation step where T4 DNA ligase is added to cell nuclei, Figure 2.1c. This favors ligation of cross-linked DNA fragments, meaning that there is a greater chance that fragments that are cross-linked (i.e., that were originally close in 3D space) will be ligated. Finally, cross-linking is reversed and DNA is either isolated and quantified, or subjected to more processing before eventually being quantified.



(a) Nuclei are cross-linked. (b) Digestion with restriction enzymes. (c) Proximity ligation with T4 DNA ligase.

Figure 2.1: Diagram of common steps shared by methods from the 3C family. Adjacent DNA strands are depicted in blue and orange. Dotted lines in Figure 2.1a represent restriction sites. Crosslinked proteins are shown in light blue. Figure based on Figure 1a from *A 3D Map of the Human Genome at Kilobase Resolution Reveals Principles of Chromatin Looping* [38].
Reproduced with permission from Cell Press.

2.2.1.1 3C

3C was developed in 2002 and was the first technique to use PCR to characterize 3D genomic interactions. The first steps of the 3C protocol are the same as those outlined at the beginning of the previous section. In the original protocol, after cross-link reversal, the cross-linking frequency (and thus, interaction frequency) of a pair of loci is determined by Quantitative PCR (qPCR) using locus-specific primers [163].

Dekker et al. [163] demonstrated the capabilities of the 3C method by applying it to the genome of *Saccharomyces cerevisiae*. Using 3C the authors were able to recapitulate known properties of centromere clustering in the yeast genome throughout the cell cycle. 3C also allowed the authors to characterize chromatin flexibility in yeast as well as generating a 3D model of chromosome III. The model showed that chromosome III has a contorted ring structure [163].

Besides the low throughput, the main disadvantage of 3C is that it requires prior knowledge of the sequence of interacting loci.

2.2.1.2 4C

Zhao et al. [164] modify the original 3C protocol to enable detection of interactions between a target locus and the rest of the genome by developing a new technique named Circular Chromosome Conformation Capture (4C). The authors used 4C to characterize interactions of *H19* Imprinting Control Region (ICR) in mouse, identifying 114 sequences interacting with the maternally inherited *H19* ICR [164].

After the first ligation step (see Figure 2.1c), the 4C protocol involves a second ligation step to produce self-circularized DNA fragments. These fragments are then amplified by inverse PCR. The amplified library is then quantified using DNA microarrays or sequencing.

While 4C is a significant improvement upon 3C, throughput is still extremely limited, and knowledge of the sequence of a locus of interest is still required.

2.2.1.3 5C

Chromosome Conformation Capture Carbon Copy (5C) is another variant of 3C that allows the characterization of all pairwise interactions occurring among loci in a genomic region of interest (usually not larger than 1 Mbp). The first steps of the 5C protocol are identical to that of the original 3C protocol. After generating a 3C library, 5C primers are annealed to the 3C library. 5C primers consist of two sequences: the sequence corresponding to the $3'$-end of restriction fragments (sense sequence in forward primers and antisense in reverse primers); T7 and T3c universal tails for amplification ($5'$-end of forward primers and $3'$-end of reverse primers, respectively). The result of the above primer design strategy is that forward and reverse primers will often anneal across ligation junctions. Next, primes that anneal next to one another are ligated using Taq ligase. Finally, 5C libraries can be amplified using T7 and T3c universal primers. The amplified library can then be quantified using microarrays or DNA sequencing [165].

Dostie et al. [165] showcased 5C by characterizing a 400 kbp region surrounding the β-globin region in K562 and GM06990 human cell lines. The authors reported that the Locus Control Region (LCR) strongly interacts with the γ-globin genes in K562 cells, where γ-globin is known to be expressed. Furthermore, the authors identified a novel chromatin loop between the β-globin LCR and a pseudogene located in the intra-genic region between γ- and δ-globin genes. This pseudogene is weakly expressed in K562 cells. Finally, the authors applied 5C on a gene-desert region, and found, as expected, that the chromatin structure in this region is similar to that of a random coiled polymer [165].

Overall, 5C is another significant improvement to 3C. Improvements in 5C are orthogonal to those of 4C: 5C allows to characterize all-versus-all interactions over a limited genomic region, while 4C allows characterizing one-vs-all interactions genome-wide. That being said, the throughput of the technique is still very limited. Furthermore, 5C primer design is a fairly laborious step, which limits the extent of the genomic regions that can be characterized.

2.2.1.4 Hi-C

Hi-C is the latest iteration of 3C techniques and is by far the most widely used method from the 3C family, largely superseding 3C, 4C, and 5C. Hi-C allows for characterizing chromatin interactions at very fine resolutions (up to 500 bp [66]) and genome-wide.

The original Hi-C protocol, dilution Hi-C [57], shares the initial steps of cross-linking and digestion with restriction enzymes from the original 3C protocol. However, instead of re-ligating the sticky ends left by the digestion step, the overhangs are filled with biotinylated nucleotides. The blunt ends are then ligated, and DNA fragments labeled with biotin are selected by streptavidin enrichment. The resulting library is then fragmented and finally sequenced using NGS [57].

The original Hi-C protocol was updated and refined several times to improve resolution and reduce the frequency of uninformative re-ligation events. The first improvement was *in-situ* Hi-C [38], which used a 4-cutter instead of a 6-cutter enzyme and where the re-ligation step is performed in intact nuclei instead of lysed and pooled nuclei. Using a restriction enzyme that cuts more frequently increases the number of possible restriction fragments and thus the resolution of the resulting Hi-C matrix. Performing the re-ligation step in intact nuclei poses two main advantages: a reduction in spurious ligation events (i.e. ligation between DNA fragments from different cells); and a reduction in sample processing time. The progress made with *in-situ* Hi-C was further refined and formalized with the release of the Hi-C 2.0 protocol [166], which makes a distinction between adherent and suspension cells and includes a step to remove biotin from un-ligated ends to reduce the number of uninformative reads. This is crucial not only to improve sequencing efficiency but also to remove invalid interactions that cannot be detected computationally, such as partially digested DNA fragments whose ends map onto adjacent restriction fragments. Another variation of the *in-situ* Hi-C protocol consists of using 2 or more restriction enzymes for DNA digestion, thus improving the theoretically achievable resolution and partially mitigating issues due to restriction site mappability biases. This and other improvements are now part of the Hi-C 3.0 protocol [167].

Using Hi-C, Aiden et al. [57] showed that the human genome segregates into two separate compartments, named A and B compartments (Section 1.2.4). A compartments correlate with gene-rich regions, high expression levels, and accessible chromatin, while B compartments correlate with gene-poor, highly condensed chromatin, and overall lower expression levels.

A few years later, Dixon et al. and Nora et al. [32, 33] identified Topologically Associating Domains (Section 1.2.2) using Hi-C on human and mice cell lines. In 2014, Rao et al. [38] developed *in-situ* Hi-C and identified many loops connecting enhancers and promoters. The authors also refine the notion of A/B compartments by identifying 6 subcompartments, each with distinct histone marks.

In conclusion, Hi-C is a significant improvement over 3C, which allows profiling chromatin interactions genome-wide across a broad range of resolutions.

However, this comes with an increase in sequencing costs, which can be quite substantial when generating deep Hi-C libraries.

2.2.1.5 Capture Hi-C

Capture Hi-C (CHi-C) extends the Hi-C protocol to include a sequence capture step to enrich Hi-C libraries for a set of loci of interest [168]. This greatly reduces sequencing costs, enabling studying a subset of chromatin interactions at very high resolutions.

2.2.1.6 DNase Hi-C

One of the limitations of Hi-C is the cut frequency imposed by digestion with restriction enzyme, which limits the maximum interaction map resolution that can theoretically be achieved.

To work around this issue, Ma et al. developed DNase Hi-C [169, 170] (also known as *in-situ* DNase Hi-C), a variation of the *in-situ* Hi-C protocol that uses DNase I instead of restriction enzymes for DNA digestion. Besides an increase in cut frequency, the main advantage of using DNase I instead of restriction enzymes is the reduction of several sources of bias that affect digestion by restriction enzymes, such as GC content, restriction site mappability, and genomic coverage. Due to an additional immobilization step, which reduces nuclei loss during library preparation, DNase Hi-C is also more suitable to be used in low-input experiments. DNase Hi-C can be combined with hybridization capture kits to reduce sequencing costs required to profile interactions for a set of loci of interest at very high resolutions. Finally, while DNase I DNA cleavage is nonspecific with respect to DNA sequence, DNaseI has been shown to exhibit mild GC bias at cleavage sites. This should be taken into account when applying DNAse Hi-C to organisms with low GC content.

The main limitation of DNase Hi-C is the high sequencing cost required to achieve high-resolution maps. This becomes less of an issue when DNase Hi-C is coupled with a capture step to enrich interactions involving one or more regions of interest.

2.2.1.7 Micro-C

Micro-C [171, 172] is another improvement on dilution Hi-C that seeks to improve interaction map resolution and genome coverage by using MNase instead of restriction enzymes for DNA digestion. MNase is a nuclease capable of digesting DNA not bound to histones and is thus suitable to generate maps reaching the nucleosome unit level (200 bp). Similarly to DNase Hi-C, the main limitation of Micro-C is the sequencing cost required to generate high-resolution maps.

A recent variation of Micro-C is Region Capture Micro-C (RCMC) [51], which combines Micro-C with tiling region capture to enrich interactions falling on a genomic region of interest. The main advantage of RCMC is a significant reduction in sequencing costs, which allows profiling a region of interest at unprecedented resolutions and sequencing depths.

2.2.1.8 Single-cell Hi-C

While bulk sequencing methods can provide precious insights into the chromatin 3D structure, they tend to show an average picture of a population of cells. This can be an issue when attempting to characterize highly dynamic processes, where different cells may exhibit radically different structures.

It is with this motivation that Nagano et al. [26] developed single-cell Hi-C, a technique that seeks to characterize chromatin structure in individual nuclei by modifying the original Hi-C protocol to perform the ligation step in intact nuclei (similarly to what is done in *in-situ* Hi-C). After cross-linking and ligation (see Figure 2.1), nuclei are manually selected under the microscope. Next, DNA is fragmented with an additional round of digestion with a different restriction enzyme. Finally, the sequencing library is prepared by inserting a barcode that uniquely identifies each nucleus.

Using scHi-C, Nagano et al. described that TADs observed in bulk HiC are generally conserved in individual nuclei. Inter-TAD interactions, however, show a much greater degree of variability, suggesting that higher-order chromosome folding may not be as conserved as previously thought. Intriguingly, with the aid of in-silico 3D modeling, the authors showed how variability in inter-TAD interactions does not affect chromatin folding of chromosome X of mouse at the level of chromosome territories.

2.2.1.9 Pore-C

Pore-C [173] is a variation of Hi-C that uses Nanopore long-read sequencing instead of paired-end Illumina sequencing to sequence ligation products.

Pore-C is the first method from the 3C family that enables the detection of multiway interactions (that is, interactions involving more than a pair of loci). Besides the sequencing platform, the main difference between the *in-situ* Hi-C and Pore-C protocol is that the Pore-C protocol does not require ligation products to be fragmented and amplified (see Figure 2.2). The main limitations of Pore-C are sequencing costs and throughput when compared to similar techniques, such as Hi-C.

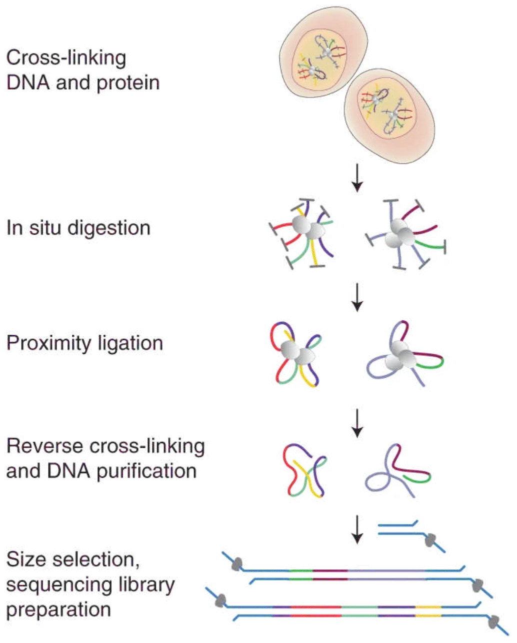


Figure 2.2: Diagram of the Pore-C protocol showing the main steps from nuclei crosslinking to Nanopore sequencing. DNA fragments are represented as lines drawn using different colors. Crosslinked proteins are depicted as gray globules.
Figure 1a from *Pore-C concatemers yield high-fidelity maps of proximal 3D contacts* [173]. *Reproduced with permission from Springer Nature.*

To at least partially address the throughput issue, High-throughput Pore-C (HiPore-C) was developed [174]. The sequencing pores used in nanopore sequencing are sensitive to DNA-bound proteins, which can result in pore clogging, thus affecting sequencing throughput. The HiPore-C protocol modifies the original Pore-C protocol to address the pore clogging issue by using pronase instead of protease K for protein degradation. Pronase is a cocktail of nonspecific proteases that can completely degrade native as well as denatured proteins. This resulted in an 80% increase in throughput compared to Pore-C, allowing to capture significantly more pairwise and multiway interactions

Deshpande et al. [173] introduced the concept of synergies, which are loci with more multiway interactions than expected. The authors found that synergies are enriched in enhancers and promoters in open chromatin.

2.2.1.10 scNanoHi-C

Single-cell Nanopore Hi-C (scNanoHi-C) [175] is a variant of Pore-C capable of recording chromatin interaction in single cells. The first steps of the protocol are similar to those of *in-situ* Hi-C. After proximity ligation, individual cells were selected using Fluorescence Activated Cell Sorting (FACS). Each cell is then deposited into a 96-well plate where DNA fragments are barcoded using a Tn5 transposase. Next, cells are pooled, cross-linking is reversed and DNA is purified for sequencing.

The main advantage of scNanoHi-C compared to scHi-C, is the ability to detect multiway interactions. Besides obvious advantages, this allows to generate many more pair-wise interactions, as a k-way interaction can be decomposed into k^2 pair-wise interactions.

2.2.2 Ligation-free methods

While 3C methods have proven to be extremely powerful, they have inherent limitations due to their reliance on cross-linking, DNA digestion, and ligation to detect interacting regions. These limitations manifest as several sources of bias, namely protein occupancy, restriction site density, and GC content. Ligation-free methods seek to overcome some of these limitations by taking a different approach, one that does not involve ligation of restriction fragments.

2.2.2.1 GAM

With the intention of developing an experimental technique orthogonal to Hi-C, Beagrie et al. [177] developed GAM, the first method that enabled capturing genome-wide 3D interactions without requiring proximity ligation.

GAM protocol is radically different from any of the 3C protocols (see Figure 2.3). First, 400–1200 cells are fixed using sucrose and then frozen. Next, cells are cryosectioned, and nuclear profiles are extracted by laser microdissection. Finally, the DNA content of each nuclear profile is extracted, amplified, and sequenced.

The underlying assumption of GAM is that neighbor loci in 3D space will be found in the same slice more often than loci that are farther apart. The computational analysis of GAM sequence data involves detecting the set of loci that are present in each slice. This information is then used to infer several chromatin parameters using the SLICE mathematical model, such as contact frequencies, radial distributions, and chromatin compaction.

Compared to Hi-C, the GAM protocol is much more laborious, requiring the collection and amplification of hundreds of nuclear profiles to achieve resolutions on par with shallow *in-situ* Hi-C maps. To at least partially mitigate this issue, Beagrie et al. [178] developed multiplex-GAM, a variation of GAM that allows collecting up to 10 nuclear profiles in a single PCR tube, thus greatly reducing the time required for library preparation.

Beagrie et al. [177, 178] compared interaction maps produced by GAM with those produced by Hi-C, and find them to be highly correlated. The authors also reported that multiway interactions detected by GAM in mice are enriched with super-enhancers and highly expressed re-

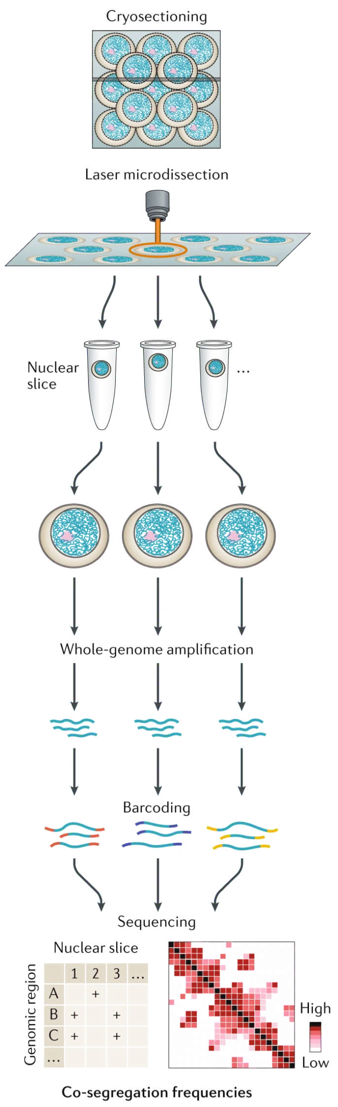


Figure 2.3: Schematic of the GAM protocol showing the main steps from cryosectioning to generation of co-segregation frequencies rendered as contact maps. Figure based on Figure 4a from *Methods for mapping 3D chromosome architecture* [176]. *Reproduced with permission from Springer Nature.*

gions. Furthermore, when comparing GAM with Hi-C, it appears that Hi-C tends to underestimate interaction frequencies for regions involved in multiway interactions.

Overall, GAM is a technique for characterizing the 3D structure that is orthogonal to FISH (introduced in section Section 2.3.1) and 3C, which validates previous findings regarding chromosome territories, A/B compartments, TADs and loops. GAM comes with several advantages, namely requiring low DNA input, independence from DNA cross-linking, digestion, and proximity ligation, and finally, the ability to capture multiway interactions. The main downside of GAM is the complexity of library preparation. However, this issue was partially mitigated by the introduction of multiplexed-GAM.

2.2.2.2 SPRITE

SPRITE [179, 180] is a ligation-free method to detect genome-wide chromatin interactions.

The core idea behind SPRITE is a split-pool barcoding scheme, where DNA fragments are repeatedly pooled in a single well, and then split across 96 wells with distinct barcodes (see Figure 2.4). The first step in SPRITE's protocol is cross-linking by formaldehyde and disuccinimidyl glutarate (DSG) followed by cell lysis using sonication and DNase digestion. Next, crosslinked DNA is bound to N-hydroxy-succinimide (NHS)-ester magnetic beads to facilitate DNA pooling in subsequent steps. At this point, DNA ends are repaired using dA-tailing enzymes, leading to DNA molecules with a single dA overhang. The next step is split-pool barcoding, which is repeated 5 times. For the first iteration, a DNA phosphate modified (DPM) adaptor with a dT overhang is added. For the remaining iterations, complexes bound to magnetic beads are split across 96 wells, each of which contains a unique tag, which is ligated to the crosslinked DNA molecules through the DPM adaptor. Next, the sample is pooled into a single well. This process will create barcodes with over 8 billion possible combinations. The last steps consist of cross-linking reversal, PCR amplification, and library preparation followed by high-throughput sequencing. Chromatin interactions can then be detected computationally by clustering reads sharing the same barcode. Each cluster represents DNA molecules that originated from the same cross-linked complex.

Using SPRITE, Quinodoz et al. [179] were able to recapitulate many of the known features and structures of the 3D genomes, such as chromosome territories, A/B compartments, TADs and loops. The

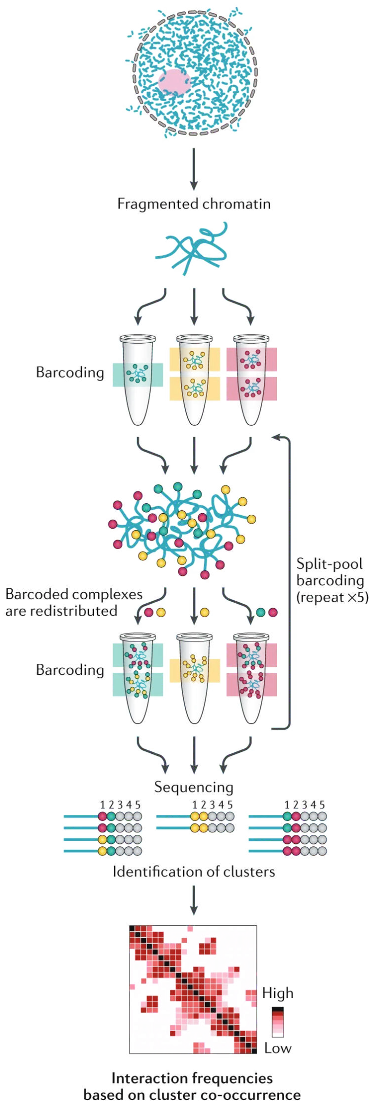

Fragmented chromatin
Barcoding
Barcoded complexes are redistributed
Barcoding
Split-pool barcoding (repeat ×5)
Sequencing
1 2 3 4 5
Identification of clusters
High
Low
Interaction frequencies based on cluster co-occurrence

Figure 2.4: Schematic of the SPRITE protocol showing the main steps from crosslinking to generation of contact maps from SPRITE clusters. Figure based on Figure 4b from *Methods for mapping 3D chromosome architecture* [176]. *Reproduced with permission from Springer Nature.*

authors also identified two interaction hubs consisting of mostly long-range interactions. One hub is enriched for gene-poor regions and regions surrounding genes encoding rRNAs and is shown to co-localize with the nucleolus. The other hub is enriched for gene-rich regions as well as several genes known to associate with the nuclear speckles. Using FISH (introduced in section Section 2.3.1) the authors showed that this hub indeed co-localizes with the nuclear speckles.

The main strength of SPRITE is the ability to characterize multiway interactions, even long-range interactions that cannot be detected by traditional 3C methods. Another feature of SPRITE is the ability to be extended to simultaneously track DNA and RNA interactions, as was recently shown with RNA & DNA SPRITE (RD-SPRITE) [181]. The main disadvantage of SPRITE is the laboriousness of its protocol, as while SPRITE's protocol is simpler and faster than that of GAM, it is more than twice as long as the typical Hi-C protocol, taking 5 days instead of 2.

2.2.2.3 scSPRITE

Single-cell SPRITE (scSPRITE) [182] is a variation of SPRITE to detect multiway chromatin interactions at the single-cell level.

The main difference between scSPRITE and the original SPRITE protocol, is that scSPRITE performs an additional round of barcoding, where DNA fragments in intact nuclei are labeled with barcodes that almost always uniquely identify individual cells. Single-cell barcoding is performed with a split-and-pool protocol analogous to that used in normal SPRITE. After cell barcoding, nuclei are lysed and the normal SPRITE split-and-pool barcoding is performed to label individual DNA-protein aggregates.

Using scSPRITE, Arrastia et al. [182] show that while most cells contain TADs, these TADs show high variability between cells. The authors also show that variable TADs are not uniformly spread along the genome, but rather cluster in regions that they term variable TAD regions. Next, the authors focus on a single variable TAD region on chromosome 4 of mouse, where they identify two populations of cells that exhibit distinct chromatin organization. The two populations of cells not only have distinct TAD configurations, but their compartmentalization is also different.

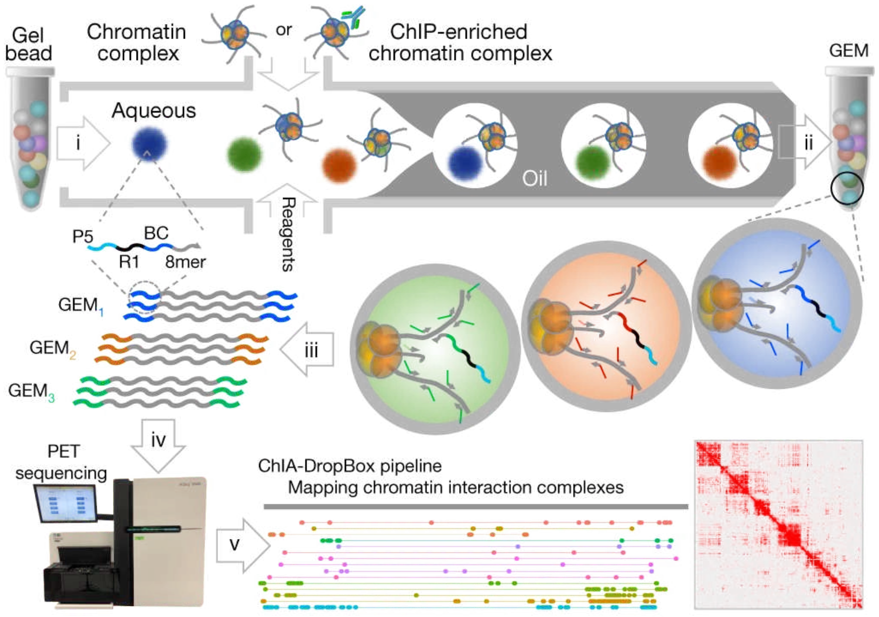
2.2.2.4 ChIA-Drop

ChIA-Drop [183] is a technique to perform multiplex analysis of chromatin interactions.

In ChIA-Drop, cross-linked and fragmented chromatin is loaded into a Chromium microfluidics system, where each DNA-protein complex is assigned to a Gel bead in Emulsion (GEM) (see Figure 2.5). GEMs are droplets containing the reagents required for DNA amplification and barcoding as well as a unique barcode that identifies the GEM (see Figure 2.5). After barcoding and amplification, DNA from droplets is pooled, purified, and sequenced (see Figure 2.5).

ChIA-Drop interactions are highly similar to Hi-C interactions, however with ChIA-Drop it is also possible to detect multiway interactions. Furthermore, ChIA-Drop can be modified to also track interactions mediated by a protein of interest such as RNAPII.

Figure 2.5: Diagram of the ChIA-Drop protocol showing the main steps from crosslinling to contact map generation.
Figure based on Figure 1a from *Multiplex chromatin interactions with single-molecule precision* [183].
Reproduced with permission from Springer Nature.



2.2.2.5 Trac-looping

Transposases-mediated Analysis of Chromatin Looping (Trac-looping) [184] employs a hyperactive Tn5 transposase to simultaneously measure chromatin accessibility and interactions.

DNA transposition mediated by Tn5 relies on the presence of a pair of 19-bp mosaic end (ME) sequences. Accordingly, Trac-looping uses a bivalent ME linker consisting of a pair of ME sequences separated by a 30 bp oligonucleotide spacer. Tn5 transposase forms a tetramer complex with the bivalent ME. The hyperactive Tn5 and bivalent ME are added to cross-linked cells, where Tn5 will insert the 4 MEs in accessible regions of the genome in cis, or trans. Next, DNA is digested using restriction enzymes and overhangs are filled with biotinylated nucleotides. Finally, DNA fragments are circularized, enriched by streptavidin pulldown, and sequenced. Insertions in cis can then be used as a proxy for DNA accessibility, while interactions in trans can be used as a proxy for long-range chromatin interactions.

The main limitation of Trac-looping is that the technique is limited to detecting interactions involving regions located in open chromatin. This is due to Tn5's preference to cut open chromatin.

2.3 Imaging methods

Methods involving visualizing labeled nuclei through microscopes were one of the first methods used to characterize the 3D structure of genomes. These methods can be classified in FISH-based and live-imaging methods. The first requires fixation, while the latter allows for the visualization of living cells.

2.3.1 FISH-based methods

Since the inception of Fluorescence In Situ Hybridization (FISH) in 1982 [185], a great variety of methods based on this technique have been developed.

FISH utilizes fluorescent DNA probes that are complementary to one or more genomic regions of interest. DNA probes bind to complementary regions so that regions of interest can be visualized with fluorescence microscopy (see Figure 2.6). The main features of FISH are: (1) enabling direct visualization of one or more regions of interest (thus allowing taking direct distance measurements between two or more regions), and (2) collecting information from individual nuclei. However, FISH also comes with two important drawbacks: namely requiring cell fixation (potentially perturbing the original 3D structure of nuclei), and being restricted to visualize only a small number of loci at the same time.

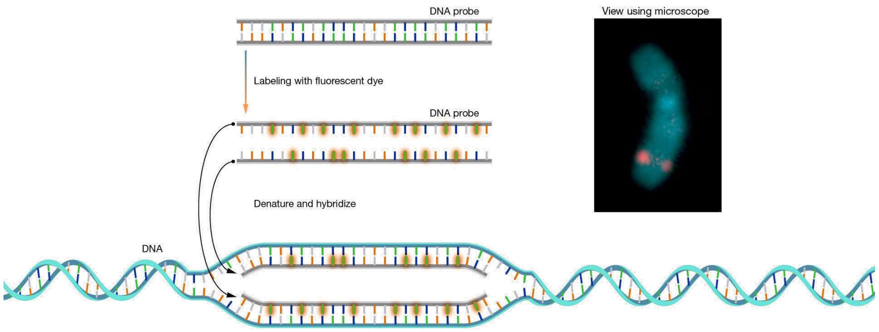
2.3.1.1 2D and 3D-FISH

2D-FISH is performed on flattened cells, where cells have been flattened to the point of exhibiting close to a 2D geometry [186]. This simplifies visualizing cells by ensuring that nuclei are on the same focal plane [187]. FISH on 3D-preserved nuclei (3D-FISH) is instead performed on nuclei in their original geometry [188]. Both 2D and 3D-FISH enable the visualization of 2–52 regions of interest at once [176]. This number is constrained by the limited number of available independent fluorescence channels [189].

One of the most important steps of FISH experiments, is probe design. Probes are usually produced by synthesis or by nick translation and are designed to be 150–500 bp long and to target regions of at least 30 kbp [176]. Probes are either directly labeled with fluorescent nucleotides or with biotinilated nucleotides which will be then recognized and highlighted by fluorescent avidin.

After fixing cells or nuclei to a glass plate, cells are permeabilized by treatment with a detergent or organic solvent. Next, complementary probes are introduced in the nucleus. To increase probe binding rates, DNA is denatured through heath and formamide treatment. Finally, labeled nuclei are imaged through a fluorescence microscope [176].

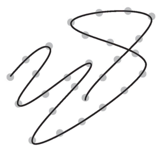
Figure 2.6: Diagram of DNA-FISH. The diagram shows a fluorescently labeled DNA probe annealing with denatured DNA. Regions labeled with fluorescent probes appear as red dots when visualized under the microscope.
Source: NCBI NHGRI



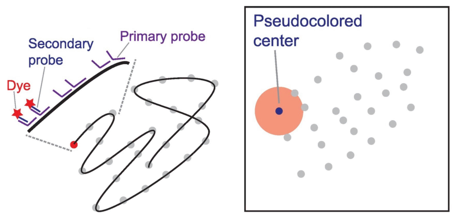
2.3.1.2 Cryo-FISH

Cryo-FISH is a variation of 3D-FISH where probes are hybridized to cryo-sections of fixed cells. Labeled cryo-sections are then visualized using a fluorescence or electron microscope. Compared to 3D-FISH, cryo-FISH enables visualization at higher resolution due to better spatial resolution, sensitivity, and image contrast [176, 190].

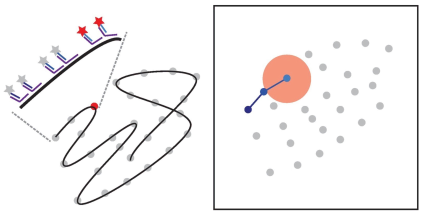
2.3.1.3 Multiplexed FISH



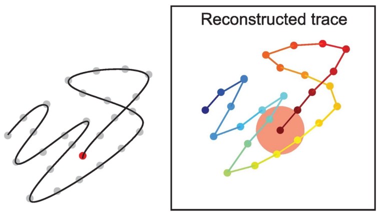
(a) Bind primary probes.



(b) Pinpoint locus 1.



(c) Pinpoint locus 3.



(d) Pinpoint locus N.

Figure 2.7: Diagram of chromatin tracing by FISH showing the protocol steps from the binding of primary probes to loci pinpointing.
DNA is represented as a gray line while loci complementary to primary probes are depicted as gray circles. Figure generated from Figure 1 from *Chromatin Tracing: Imaging 3D Genome and Nucleome* [191].

Multiplexed FISH, also known as chromatin tracing, is a variation of 3D-FISH that combines FISH with oligo probes (such as Oligopaints) and microfluidic technologies to resolve chromosome-scale chromatin folding at higher resolution.

Oligopaints is a method to generate libraries of labeled oligonucleotides suitable for FISH experiments [192]. Compared to traditional probes, probes produced by Oligopaints are shorter (60–100 bp instead of 150–200 bp) and exhibit higher precision in probe placement. Because of this, Oligopaints FISH allows targeting of loci in the order of 5 kbp when using super-resolution microscopy [176].

Chromatin tracing involves labeling a large number of genomic regions across hundreds or thousands of cells by using multistep labeling followed by imaging, where both steps are automated by using microfluidics. The first step involves the hybridization of primary probes to all regions of interest along the genome (Figure 2.7a). Primary probe sequences consist of two portions: the first portion is complementary to a region of interest, while the second portion is a barcode sequence that does not hybridize with any genomic sequence. The first hybridization round is followed by a second round of probe hybridization where fluorescent-labeled secondary probes are added one by one and imaged (Figures 2.7b to 2.7d). Secondary probes hybridize with the barcode sequence of primary probes, thus enabling imaging of one of the genomic regions of interest. Next, following the imaging step, the fluorescent probes are removed by e.g., photobleaching before proceeding with the next round of secondary probe hybridization. Once all loci marked by primary probes have been imaged, the centers of the genomic regions of interest are linked to trace the chromatin structure [191, 193].

Overall, compared to traditional 3D-FISH, multiplexed FISH can be used to visualize smaller regions at high resolution (1.2–2.5 Mbp at 30 kbp resolution [194]) or larger regions at intermediate resolution (such as entire chromosomes at TAD-level resolution [195]) [176, 191, 193].

Earlier iterations of chromatin tracing date back to 2016, and were used to characterize the spatial organization of TADs in human interphase autosomes and X chromosomes, reaching resolutions in the order of 1–4 Mbp [193, 195]. Combining chromatin tracing with super-resolution microscopy allowed for the characterization of smaller regions (1.2–8 Mbp) at up to 10 kbp resolution[193, 194, 196]. Wang et al. [195] report that based on their findings, the fractal globule model does not recapitulate the structure of chromosomes for genomic distances greater than 7 Mbp. Consistent with existing knowledge about TADs segregating into A/B compartments, the authors find that at coarse resolutions TADs segregate into two distinct compartments. Lastly, the authors characterize the active and inactive copies of chromosome X, and find that their topologies and locations are profoundly different.

2.3.1.4 Multimodal FISH imaging

One of the capabilities of chromatin profiling is the possibility of complementing it with other profiling methods, thus providing not only insight into chromatin structure but also into its functional state. The first attempts at multimodal FISH imaging were Hi-M and Optical Reconstruction of Chromatin Architecture (ORCA) which combined chromatin profiling with RNA-FISH, allowing simultaneous imaging of up to 122 loci at 2–10 kbp resolutions and up to 29 RNA targets [193, 197, 198]. Hi-M and ORCA were followed by several other methods improving resolution, throughput, and the number and type of molecules profiled, namely Multiplexed Imaging of Nucleome Architecture (MINA) [199, 200], DNA Multiplexed Error-Robust Fluorescence in situ Hybridization (MERFISH) [201] and DNA Sequential Fluorescence in Situ Hybridization (seqFISH+) [202, 203].

Liu et al. applied MINA to chromosome 19 of mouse fetal liver tissue. They identify 7 different cell types and characterize chromatin folding around *Scd2*, a gene that is expected to be highly expressed in hepatocytes only. Thanks to the fine resolution achieved with MINA, the authors were able to pinpoint a single promoter-enhancer interaction likely responsible for the cell-specific expression of *Scd2*. The authors were also able to recapitulate several results from previous findings, such as the positive correlation between compartment score and gene expression, as well as the polarized organization of A/B compartments [199].

2.3.2 Live-imaging

While FISH techniques are very powerful, they only allow taking still snapshots of chromatin structure from fixed cells. In contrast, methods for live cell imaging allow tracking chromatin structure over time, which is very important, as chromatin folding is known to be a highly dynamic process [204].

2.3.2.1 CRISPR-Tag

CRISPR-Tag uses catalitically-dead Cas9 (dCas9) fused with a fluorescent protein to recognize and bind a region of interest [25]. Similarly to what is done in the CRISPR-Cas9 method for genome editing, a short guide RNA (sgRNA) is engineered to have its guide sequence complementary to a region of interest. The fluorescently labeled dCas9 protein will then be guided to the region of interest and mark it. The main limitation of CRISPR-Tag is that the technique is limited to tagging repetitive regions, as to achieve adequate signal-to-noise ratios, multiple dCas9 proteins have to be recruited to the same locus.

2.3.2.2 CARGO

Chimeric Array of gRNA Oligonucleotides (CARGO) is a method based on CRISPR-TAG that can deliver up to 12 different sgRNAs using a single vector, thus overcoming the main limitation of CRISPR-Tag [205]. Using CARGO, Gu et al. were able to live image a non-repetitive 2 kbp region.

2.4 Differences and similarities between experimental techniques and closing remarks

In the previous sections, I presented four families of experimental techniques that can be used to characterize the 3D structure of genomes. Many techniques tackle the problem using orthogonal approaches, each with their strengths and weaknesses. Despite this, all presented methods can identify the hallmarks of chromatin 3D organizations, namely chromosome territories, open/close chromatin domains, and DNA loops. Most of the sequencing methods can also recapitulate TADs. This gives great confidence that the observed structures are not technical artifacts, but rather real structures that are present in ensemble or individual cells.

FISH and live-imaging methods have the unique capability of directly imaging nuclei, enabling direct measurement of distance between a set of regions of interest. Distance measures can also be converted to interaction maps by setting a distance threshold below which two genomic regions are considered as co-locating. Furthermore, both FISH and CRISPR-Tag-like methods allow visualizing 3D structures at the single-cell level. However, all imaging approaches have several limitations in common. FISH methods require cell fixation, a treatment that can potentially alter the genome 3D structure. Furthermore, all imaging techniques depend on the recruitment of a large number of fluorophores on the region to be visualized, potentially disrupting the local chromatin 3D structure. Another limitation of imaging methods is the low throughput, which even in the case of highly multiplexed imaging cannot match that of other non-imaging approaches.

Chromosome Conformation Capture methods in contrast, especially those based on Hi-C, do not suffer from throughput issues and can be used to characterize 3D interactions genome-wide and at considerably high resolutions. The only exception is single-cell variants of 3C, whose throughput is quite limited when looking at interactions for individual cells. One important limitation of many 3C techniques, is the inability to detect multiway interactions. This has recently changed with the development of Pore-C and related methods (including Tri-C [206], 4C-walks [207] and multi-contact 4C [208], all of which predate Pore-C). The main issue with 3C methods is the compound effect of several different sources of bias introduced by cross-linking, restriction enzyme digestion, and proximity ligation. Another issue is the high sequencing costs required to generate deep Hi-C maps, as sequencing costs scale with the square of the density of interactions in Hi-C maps.

To sidestep some of the issues associated with Hi-C and 3C methods in general, several ligation-free methods have been developed. GAM uses a radically new approach, using the frequency of co-occurrence of genomic loci in thin slices of cryosections of nuclei to infer chromatin interactions. Being orthogonal to Hi-C, GAM is not affected by the source of biases that are associated with Hi-C. Unfortunately even GAM is not a silver bullet, as it is a fairly laborious technique and cannot yet reach the level of details achieved with deeply sequenced Hi-C libraries. SPRITE and ChIA-Drop are less demanding in terms of labor required for library preparation compared to GAM, however, their protocols are still more complex than Hi-C. SPRITE's ability to capture longer range interactions than 3C methods is both an advantage and disadvantage. It is advantageous because it allows the detection of long-range interactions that are invisible to 3C methods, which can only detect interactions between DNA molecules that are less than 200 nm apart. However, there is no clear way to put a hard limit on the maximum distance of interactions detected by SPRITE, making the definition of interacting regions a bit vague. Finally, applications of Trac-looping are limited by its inability to detect interactions taking place in close chromatin.

We are now at a point where we have several techniques using orthogonal approaches to characterize the genome 3D structure. Several of these techniques have been developed with the aim of avoiding biases associated with the proximity ligation and crosslinking steps of 3C methods. However, 3D-FISH and Hi-C are still the most commonly used techniques and are considered as golden standards to compare to when new techniques are developed. This together with the high level of agreement between FISH, Hi-C, and other techniques suggest that the sources of bias affecting FISH and Hi-C, likely have limited effect on detected interactions. Likewise, the high level of agreement between Hi-C and ligation-free methods such as GAM and SPRITE suggest that these methods are valid alternatives to 3C methods that enable the characterization of multiway interactions.

To conclude, we do not yet have an experimental technique to rule them all.

At present, it seems unlikely that any of the available methods will obsolete all others in the near future, especially considering that some of the techniques, such as Hi-C and FISH nicely complement each other. Instead, researchers will have to continue picking the most appropriate technique based on their research questions, available expertise, and experiment/sequencing budgets. This suggests that FISH and Hi-C will continue to thrive for the foreseeable future, as the availability of well-established protocols and commercial preparation kits make them more broadly accessible to research groups who do not yet have extensive experience working with these techniques. The availability of computational pipelines for pre-processing and analysis of new types of data is also a current limitation to the diffusion of novel experimental techniques, as processing and analyzing data produced by said techniques will not only require expertise in the analysis of biological data but also experience in software development.

Chapter 3

File formats used to represent interaction frequencies

3.1 Introduction

This chapter aims to provide a comprehensive overview of textual and binary formats used to represent interaction frequencies, such as those generated by Hi-C experiments. For textual formats, I will focus on relatively popular formats, as describing all formats ever invented would require too much space, given that in the early days virtually every group developed their format. Finally, I will focus on the .hic and cooler formats, as these two are the de-facto format standard to represent pairwise interactions in a highly compressed, indexed file format. The purpose of this chapter is to provide the necessary background for Paper II, which introuces a toolkit to transparently operate on .hic and cooler files.

3.2 From reads to interactions

The journey from raw 3C data to an interaction matrix begins with one or more pairs of files in FASTQ format.

Throughout this section, I will assume to be processing the paired-end FASTQ files produced by a Hi-C experiment, even though most of the steps are independent of which method from the 3C family generated the data. FASTQ is a text format used to store nucleotide sequences together with their quality scores. Most Hi-C experiments use Illumina paired-end sequencing with read lengths ranging between 2x36 and 2x150 bp [209, 210], (with some protocols requiring up to 2x250 bp HIFI reads [211]) Reads pairs are usually mapped individually to a reference genome via iterative mapping [212], two-step mapping [213] or by using specialized read aligners, such as Chromap [214], taking care to properly map chimeric reads produced by the proximity-ligation step in the Hi-C library preparation. Mapped reads are stored in SAM or BAM format [215]. At this point, read pairs have to be filtered to identify informative pairs (i.e., reads mapping to two different restriction fragments) from uninformative pairs (e.g., self-circle and dangling-end pairs). This produces a list of valid pairs of interactions. Finally, valid interactions can then be binned and aggregated to generate interaction matrices.

Over the last decade, several pipelines have been developed to automate some or all of the above steps. To name a few: HiC-Pro [213], Juicer [60], 4DN Hi-C processing pipeline [216], Distiller [217], ENCODE HiC uniform processing pipeline [218] and nf-core/hic [219]. While the file formats used for the mapping step are standardized and well-defined, formats used in other steps, and especially to represent pairs of interactions and interaction matrices are less standardized.

3.3 Formats to store pairwise interactions

This section introduces and describes the most common formats used to represent lists of pairwise interactions. This family of file formats is crucial, as these formats act as intermediate representations of 3C data between the read mapping and interaction binning steps. Pairwise interactions are processed by tools such as Cooler [220], JuicerTools [60], HiCExplorer [221, 222], and hictk (Paper II) to generate highly-compressed and indexed representations of binned interaction matrices.

Pairs of interactions are usually represented as long lists of records, where each record contains a pair of genomic coordinates. Despite the simplicity of the data being stored, lists of pairwise interactions come in a variety of formats, some of which are outlined in the current section.

3.3.1 validPairs

validPairs is the file format used by HiC-Pro [213] as well as one of the formats used by nf-core/hic [219] to represent pairwise interactions. It was one of the first formats introduced to store pairwise interactions of Hi-C experiments.

The format is a tab-delimited textual format without a header and with the following columns:

- `read name` - The read name.
- `chr_reads1` - Chromosome name for the first mate.
- `pos_reads1` - Mapping position of the first mate.
- `strand_reads1` - Strand for the first mate.
- `chr_reads2` - Chromosome name for the second mate.
- `pos_reads2` - Mapping position of the second mate.
- `strand_reads2` - Strand for the second mate.
- `fragment_size` - Fragment size in bp.
- `res frag name R1` - Restriction fragment for the first mate.
- `res frag R2` - Restriction fragment for the second mate.
- `mapping qual R1` - Mapping quality for the first mate.
- `mapping qual R2` - Mapping quality for the second mate.
- `allele_specific_tag` - Allele tag (optional).

Listing 1 shows an example of a validPair file.

```
A00548:315:H77LWDMXY:2:2405:30436:31031 chr1  16145 + chr1  187139  - 2538  HIC_chr1_30 HIC_chr1_951  31  35
A00548:315:H77LWDMXY:2:2231:8603:2550 chr1  16167 + chr1  186953  - 2330  HIC_chr1_30 HIC_chr1_951  31  31
A00548:315:H77LWDMXY:1:1167:16920:11913 chr1  16195 + chr1  186981  - 2330  HIC_chr1_30 HIC_chr1_951  31  31
A00548:315:H77LWDMXY:2:2165:4020:8390 chr1  16265 - chr1  185829  - 512 HIC_chr1_30 HIC_chr1_947  31  11
A00548:315:H77LWDMXY:1:1417:8187:10504  chr1  16750 + chr1  187560  - 2354  HIC_chr1_30 HIC_chr1_951  31  31
A00548:315:H77LWDMXY:2:1401:4996:12680  chr1  16771 + chr1  187566  - 2339  HIC_chr1_30 HIC_chr1_951  11  31
A00548:315:H77LWDMXY:1:2287:16712:16908 chr1  16845 - chr1  186898  + 2536  HIC_chr1_30 HIC_chr1_951  31  31
A00548:315:H77LWDMXY:1:2161:29541:2300  chr1  16868 - chr1  186992  + 2465  HIC_chr1_30 HIC_chr1_951  31  34
A00548:315:H77LWDMXY:2:2117:11351:11913 chr1  17405 + chr1  188136  - 2275  HIC_chr1_30 HIC_chr1_951  37  31
```

Listing 1: validPairs example file.

3.3.2 Juicer

The Juicer suite of tools [60] defines its file format to represent pairwise interactions in text format. This format does not have a header and consists of a space-separated file where each record contains the following fields for each of the two mates:

- `str` - Mate strand (0 for forward, anything else for reverse).
- `chr` - Chromosome name.
- `pos` - Mapping position.
- `frag` - Restriction site fragment.
- `map` - Mapping quality score.
- `cigar` - Compact Idiosyncratic Gapped Alignment Report (CIGAR) string.
- `sequence` - Read sequence.

There are two variations of the original Juicer format:

- A "medium format", with an additional field storing the `readname` and without the `cigar` and `sequence` fields.
- A "short format", which is based on the "medium format" and lacks the `readname` and `map` fields.

```
C24LCACXX130513:3:2315:8962:99350 0 1 13469 7 16 1 52724972 146510 2 178
C24LCACXX130513:3:2216:17074:2449 0 1 18111 15 0 1 235989131 542545 0 178
D258GACXX130605:3:2101:13335:2917 0 1 23820 26 16 1 20850 24 0 0
D258GACXX130605:3:2211:8449:53007 0 1 30819 42 16 1 31292 43 5 0
D258GACXX130605:3:1316:8258:97833 0 1 38027 59 0 1 39931 70 0 0
D258GACXX130605:3:2212:1989:10981 0 1 46778 86 0 1 150327905 328353 0 178
C23LRACXX130511:8:2209:8430:42384 0 1 55470 107 0 1 101038308 262310 0 91
D258GACXX130605:3:1215:9842:28962 0 1 67704 140 16 1 79804 170 0 0
D258GACXX130605:3:1105:6204:52382 0 1 82209 179 0 1 7072188 16245 0 178
C24LCACXX130513:3:2116:12420:51114 16 1 91634 204 16 1 31296715 86648 0 176
```

Listing 2: Juicer "medium" example file.

Interactions in Juicer format can be used by JuicerTools to generate multi-resolution interaction matrices in .hic format. See Listing 2 for an example of the Juicer "medium" format.

3.3.3 DCIC Pairs

The DCIC Pairs file format [223], colloquially known as the "Pairs" format, is a file format that stores pairwise interactions following a header in textual format.

The file header contains metadata regarding the file format, whether the data is sorted by genomic coordinates, the data shape (i.e., whether the file contains interactions only spanning the upper triangle of a Hi-C matrix), and the list of chromosomes with their sizes. After the header, each line represents an interaction expressed as (readID, chr1, pos1, chr2, pos2). See Listing 3 for a sample snippet of the file format. readID is an optional field and when not used should be set to ".". The Pairs format is extensible, meaning that in addition to the reserved columns (readID, chr1, pos1, chr2, pos2, strand1, strand2), additional columns can be added by specifying them through the `#column` header field.

```
## pairs format v1.0
#sorted: chr1-chr2-pos1-pos2
#shape: upper triangle
#genome_assembly: hg38
#chromsize: chr1 249250621
#chromsize: chr2 243199373
#chromsize: chr3 198022430
...
#columns: readID chr1 pos1 chr2 pos2 strand1 strand2
EAS139:136:FC706VJ:2:2104:23462:197393 chr1 10000 chr1 20000 + +
EAS139:136:FC706VJ:2:8762:23765:128766 chr1 50000 chr1 70000 + +
EAS139:136:FC706VJ:2:2342:15343:9863 chr1 60000 chr2 10000 + +
EAS139:136:FC706VJ:2:1286:25:275154 chr1 30000 chr3 40000 + -
```

Listing 3: DCIC Pairs example file.

The Pairs format was standardized in 2016 by the 4D Nucleome Omics Data Standards Working Group, and aims to be a tool-agnostic, textual format to represent lists of pairwise interactions. The format was developed to enable random queries when Pairs files are compressed using bgzip [224] and indexed using Pairix (introduced in Section 3.3.3.2).

3.3.3.1 Pairsam

An example of an extension of the Pairs format is the Pairsam format defined by Pairtools [225], which introduces three additional fields in addition to the 7 reserved fields, namely `pair_type`, `sam1` and `sam2`. `pair_type` is used to store the type of Hi-C pair defined by a record, while `sam1` and `sam2` are used to store the alignment in SAM format for the first and second mate respectively. In addition, Pairtools embeds the SAM file header directly in the Pairs header through the `#samheader` header field.

3.3.3.2 Pairix

Pairix is an indexing strategy and index format over files in Pairs format that enables efficient fetching of pairs overlapping queries consisting of a pair of genomic coordinates. Pairix is an extension of the Tabix indexing format, that is used to index tab-delimited, compressed text files over one set of genomic coordinates [224]. To make a Pairs file indexable by Pairix, interactions in a Pairs file needs to be sorted by chr1, chr2, pos1, and pos2. The sorted Pairs file then needs to be compressed using bgzip [224]. Finally, the file can be indexed using the pairix utility [226], resulting in an index file with extension `.px2`. Lee et al. [223] show that running a query using a Pairix indexed file consisting of 300 million records takes approximately 1 second while satisfying the same query by traversing a Pairs file takes up to 30 minutes.

3.4 Binned interactions

It is often the case we are not interested in the exact mapping position of each mate pair, but rather in the number of read pairs mapping over a given pair of genomic intervals. There are several formats that can store this kind of information as sparse 2D matrices in textual or binary format.

3.4.1 COO

The COO format is the simplest textual format that can be used to represent sparse matrices. It consists of a list of records, where each record stores the matrix row, column, and associated value (see Listing 4 for an example). Files in this format are often compressed and sorted by row and column.

```
0 0 33188
0 1 13454
0 2 2560
0 3 911
0 4 753
0 5 846
0 6 530
0 7 378
0 8 630
0 9 756
```

Listing 4: COO example file.

This format is commonly used by linear algebra applications as intermediate representation of 2D sparse matrices.

3.4.2 Matrix format

The .matrix format was introduced by the Dekker lab and stores interaction matrices as dense matrices in tab-delimited textual format. The first row and column in the matrix contain the list of genomic intervals represented by each column/row in the interaction matrix. Furthermore, the first row begins with the shape of the matrix being represented. Because this format uses a dense matrix representation, it is not suitable for representing the high-resolution interaction matrices produced by modern Hi-C experiments. To reduce storage space requirements, files in this format are often compressed.

Refer to Listing 5 for an example of a file in .matrix format.

```
3x3 0|mm9|chr1:1-500000 1|mm9|chr1:500001-1000000   2|mm9|chr1:1000001-1500000
0|mm9|chr1:1-500000 1  nan 12.6330 17.0892
1|mm9|chr1:500001-1000000  14.7847 18.9111 13.5268
2|mm9|chr1:1000001-1500000 15.1814 9.5573  6.9416
```

Listing 5: .matrix example file.

3.4.3 Bedgraph2

Bedgraph2, also known as 2Dbedgraph, Ginteractions [227], and sometimes incorrectly referred to as BED Paired-End (BEDPE), is a file format derived from the widely popular BED format, where each line contains a pair of genomic coordinates in BED3 format followed by the number of interactions corresponding to the pair of coordinates.

- `chrom1` - Chromosome #1.
- `start1` - Start position of interval #1.
- `end1` - End position of interval #1.
- `chrom2` - Chromosome #2.
- `start2` - Start position of interval #2.
- `end2` - End position of interval #2.
- `value` - Number of interactions.

See Listing 6 for a snippet taken from a file in bedgraph2 format.

```
chr2L 0 50000 chr2L 0 50000 33188
chr2L 0 50000 chr2L 50000 100000  13454
chr2L 0 50000 chr2L 100000  150000  2560
chr2L 0 50000 chr2L 150000  200000  911
chr2L 0 50000 chr2L 200000  250000  753
chr2L 0 50000 chr2L 250000  300000  846
chr2L 0 50000 chr2L 300000  350000  530
chr2L 0 50000 chr2L 350000  400000  378
chr2L 0 50000 chr2L 400000  450000  630
chr2L 0 50000 chr2L 450000  500000  756
```

Listing 6: Bedgraph2 example file.

Files in this format are often sorted by genomic coordinates and compressed.

3.4.4 Juicer

The file format used to store binned interactions in Juicer format is based on the “short” format introduced in Section 3.3.2. On top of the standard columns, this format includes a `score` column storing the number of interactions corresponding to the given bin.

Files in this format can be used by JuicerTools to generate matrices from already binned interactions.

3.4.5 HiC-Pro

Interactions stored in HiC-Pro [213] format consist of two files. A Bedgraph file with the list of genomic bins (see Listing 7) and their corresponding index and a file in COO format with the list of binned interactions (see Listing 4).

```
chr1  0 1000000 1
chr1  1000000 2000000 2
chr1  2000000 3000000 3
chr1  3000000 4000000 4
chr1  4000000 5000000 5
chr1  5000000 6000000 6
chr1  6000000 7000000 7
chr1  7000000 8000000 8
chr1  8000000 9000000 9
```

Listing 7: HiC-Pro bin table example.

Genomic coordinates can then be mapped to interactions using the index specified in the fourth column of the Bedgraph file.

3.4.6 HiCExplorer

The HiCExplorer file format [221, 222], also known as h5, is a format based on Hierarchical Data Format 5 (HDF5) [228].

HDF5 is the latest version of Hierarchical Data Format (HDF) and is used to store data using a hierarchy of objects, such as groups, datasets, and attributes. The resulting hierarchy is similar to that of files and folders in a filesystem. Groups serve the purpose of organizing datasets and attributes using a hierarchical structure. Datasets are the most common way to store large amounts of data as typed multidimensional arrays. Finally, attributes can be used to store small amounts of data associated with groups and datasets, such as metadata about the object they refer to. Datasets can store data using a contiguous or chunked layout. Furthermore, chunked datasets support several filters that can be used e.g., to shuffle or compress the underlying data.

Matrices in h5 format use two groups, `intervals`, and `matrix` and several datasets to represent Hi-C matrices.

- `intervals` - Group used to store the list of intervals represented by the h5 file (i.e., the bin table).

  - `chr_list` - Dataset used to store chromosome names associated with intervals.
  - `start_list` - Dataset used to store the beginning of intervals.
  - `end_list` - Dataset used to store the end of intervals.
  - `extra_list` - Dataset to use additional information associated with each interval.

- `matrix` - Group used to store the list of pairwise interactions.
  - `data` - Dataset used to store the number of interactions in Compressed Sparse Row (CSR) format.
  - `indices` - Dataset used to store column indices in the CSR matrix.
  - `indptr` - Dataset used to store the indptr vector, mapping elements in data and indices to the rows in the CSR matrix.
  - `shape` - Tuple storing the shape of the matrix.

Files in HiCExplorer format can be generated using the hicBuildMatrix and hicConvertFormat subcommands from HiCExplorer. Furthermore, files in h5 format can be used as input by most commands from the HiCExplorer suite of tools. However, the use of this format outside the HiCExplorer suite is quite limited.

3.4.7 The .hic format

The .hic file format was developed by the Aiden group and is today one of the most popular formats used to store Hi-C interactions [60]. The format supports representing pairwise interactions at multiple resolutions, with and without normalization vectors. Files in .hic format are generated from pairwise or pre-binned interactions stored in Juicer format using the `pre` subcommand of JuicerTools or HiCTools [60]. Files in .hic format can be read by many applications, including straw [229] and hictk (see Paper II). The .hic format specification defines how Hi-C interactions are stored in .hic files [230]. The format specification went through several iterations, with the latest iteration being .hic v9.

3.4.7.1 Overview of the .hic v9 specification

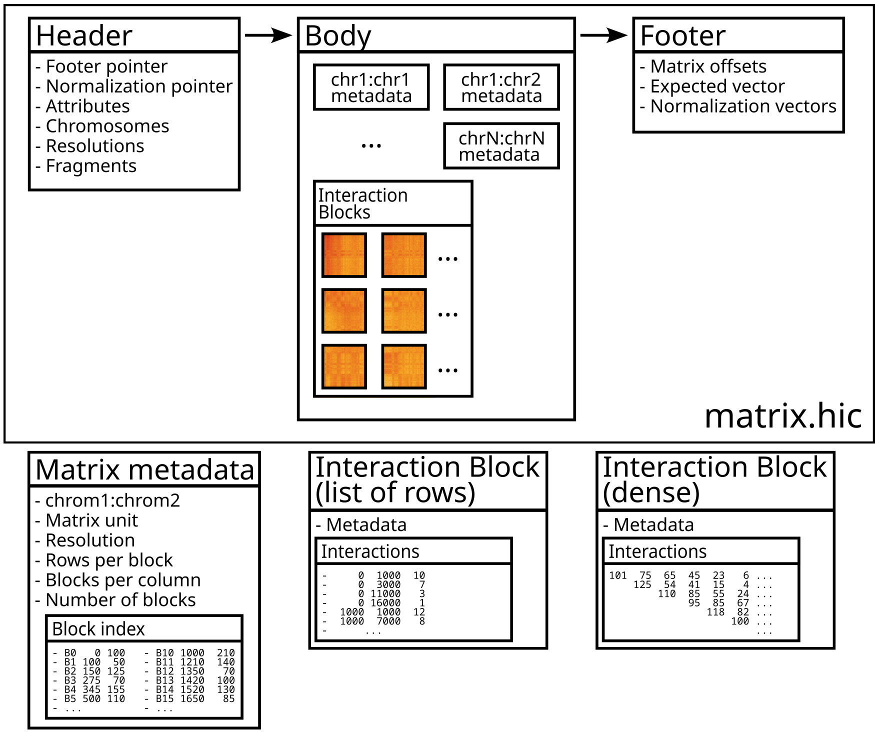


Figure 3.1: Simplified diagram of the .hic specification.
The first half of the diagram shows the file layout of a matrix.hic file. The diagram shows the three main sections of a .hic file. Sections are listed in the same order as they appear in .hic files, as exemplified by the arrows. For each section, the diagram reports the most important data types stored. The body section is by far the largest section in .hic file. This section contains one metadata block for each chromosome-chromosome matrix. The structure of a matrix metadata block is shown in the second half of the diagram. Crucially, the metadata block contains an index mapping interaction block IDs to offsets in the .hic files. These offsets point to blocks stored in the interaction blocks section of the file body. Here, interactions can be represented as sparse or dense sub-matrices (see second half of the diagram).

The format specification for .hic v9 was released in late 2020. Compared to .hic v8, .hic v9 uses a new and improved indexing strategy for intra-chromosomal interactions that results in a more compact index as well as more dense interaction blocks, making block decompression more efficient. Figure 3.1 shows a simplified diagram of the .hic v9 format specification.

Files in .hic format consist of a header, body, footer, and block table. The first three bytes in .hic files contain the “HIC” magic string, which is used to identify .hic files.

After the magic string, .hic files store the file header, which contains metadata stored in binary format. Among the metadata fields, there are the format version, the list of chromosomes, and the list of available resolutions for matrices using bp and fragment units. Crucially, the header stores the offset to the footer section as well as the section where the normalization index is stored.

The header section is immediately followed by the body section, which stores the actual interaction matrices for each pair of chromosomes.

The last section in the file is the footer. The footer contains a list of keys associated with file offsets, where keys correspond to chromosome pairs and the offset points to the relative position in the file where the matrix for the given pair of chromosomes is stored. The footer also contains the list of vectors for the expected values as well as normalization vectors.

Each matrix in the body section consists of some metadata (e.g., the matrix unit and resolution) as well as an index enumerating interaction blocks and mapping them to their offset and size in the .hic file. Interaction blocks are stored in compressed form in the block section, which is part of the body section. Each block contains interactions spanning a rectangular region from the Hi-C matrix. As of .hic v9, there are two types of blocks: blocks storing data as lists of sparse rows, and blocks storing data in dense format.

This section described a simplified version of the .hic v9 specification, for more details readers are referred to the format specification [230].

3.4.7.2 How do queries on .hic files work?

Given a query such as [`chr1:10,000-50,000`, `chr2:25,000-55,000`) at 5 kbp resolution, to satisfy the query, applications such as straw [229] and hictk (see Paper II) need to proceed as follows.

The first step is reading the file header to ensure the .hic file contains interactions for the given chromosome and resolution.

Next, applications seek to the footer offset found in the header section to then read the list of chromosome pairs and the offset to their matrix in the .hic file. Once the appropriate chromosome pair has been found, applications seek to the corresponding file offset, which marks the beginning of a matrix stored in the body section.

At this point, applications read the necessary parts of the metadata, and then read the index portion for the given matrix. Once read, the index can be used to map each block ID to the beginning and end of a block of interactions stored in the block section of the file. Next, applications need to compute the list of block IDs that may contain interactions overlapping the given query. This is achieved in different ways depending on the version of the file that is being read and whether the given query spans an intra- or inter-chromosomal region of the Hi-C map. To answer the example query, first genomic coordinates are converted to bin IDs by dividing them by the bin size. Next, the first and last bins of the query are mapped to blocks by dividing them by the block size (i.e., the number of rows stored in an interaction block). The resulting block IDs correspond to the first and last blocks containing interactions overlapping the query. Block IDs can be mapped to rows and columns in the block grid by dividing block IDs by the number of blocks in a row of the Hi-C matrix (the result of the division will be the row number while the remainder of the division will be the column number). The row and columns are then used to generate the list of blocks containing interactions overlapping the query. The last step consists of reading and decompressing interaction blocks and filtering interactions such that only interactions overlapping the query are returned.

Satisfying intra-chromosomal queries on .hic v9 files is a bit more complex, as in this version of the .hic format specification, blocks are rotated by 45° and their shape is no longer symmetric. Furthermore, in case the given query refers to normalized matrices, raw interactions need to be multiplied by the appropriate normalization coefficients.

Following the above logic, and thanks to the implementation of several optimizations, hictk is capable of running approximately 10 TAD-sized queries per second at 1 kbp resolution on a ultra-deep .hic file (ENCFF447ERX [56], see Paper II).

3.4.8 The cooler format

The cooler file format was developed by the Mirny group in mid-2019 [220] and is a format based on HDF5 [228]. Cooler represents Hi-C interactions using a Compressed Sparse Row (CSR) storage scheme adapted to have genomically labeled axes. CSR matrices are represented using three 1D arrays. A "data" array that stores the list of nonzero values. An "indices" array mapping each element in the "data" array to its column. An "indptr" array mapping elements of the "data" and "indices" vector to rows. HDF5 supports efficient storing of dense matrices but does not natively support sparse matrices. To work around this issue, cooler uses HDF5's dense 1D vector to represent the data and indices required to represent a matrix in CSR format.

On top of single-resolution cooler files (.cool), the cooler specification defines multi-resolution (.mcool) and single-cell cooler files (.scool). Similarly to .hic files, cooler files store raw interactions and compute normalized interactions by dividing raw interactions by a pair of weights from the normalization vector. The latest revision of the single-resolution cooler specification is v3, while the latest specification for .mcool and .scool files are v2 and v1, respectively.

3.4.8.1 Overview of the cooler v3 specification

As outlined in Section 3.4.6, HDF5 uses groups, datasets, and attributes to represent data using a hierarchical format.

The specification for single-resolution cooler files defines 4 mandatory groups, `chroms`, `bins`, `pixels`, and `indexes`, which are used to represent a genomically-labeled CSR matrix. The `chrom` group contains the `name` and `length` datasets with the list of chromosome names and their sizes, respectively.

The `bins` group contains the bin table, which is represented using 3 mandatory datasets, `chrom`, `start` and `end`, defining the chromosome, leftmost and rightmost positions of an interval, respectively. In addition to the 3 mandatory datasets, the `bins` group can contain one or more additional datasets storing the normalization vectors used for balancing interactions. Normalization datasets should have the same shape as the other datasets in the bin table.

The `pixels` group consists of three datasets: `bin1_id`, `bin2_id` and `count`. `bin1_id` and `bin2_id` contain the index corresponding to two entries in the bin table mapping the pixel to a pair of genomic bins, while `count` stores the non-zero count associated with the corresponding pair of bins.

Finally, the `index` group contains two vectors: `chrom_offset` and `bin1_offset`. `bin_offset`, stores the index of the first pixel mapping to each row in the sparse matrix. This dataset will thus consist of $n + 1$ entries, where n is the number of bins stored in the `bins` table. `chrom_offset`, on the other hand, contains $m + 1$ entries, where m is the number of chromosomes and where each entry corresponds to the index of the first bin in the `bins` table mapping to a given chromosome. The last entries in the `chrom_offset` and `bin_offset` tables correspond to the size of the bin table.

To improve the performance of random access queries and reduce file size, datasets in cooler files are chunked and compressed using gzip.

3.4.8.2 How do queries on .cool files work?

Queries on .cool files are carried out in a radically different way than queries on .hic files, as .cool files store interaction genome-wide, in a single table, while .hic files store one interaction matrix for each pair of chromosomes.

Given a query such as [`chr1:10,000-50,000`], the first step undertaken by applications satisfying the query consists of identifying the first row containing pixels overlapping the given query. This is accomplished in two steps. First applications identify the index i_C corresponding to the queried chromosome. i_C is then used to find the first and last bin overlapping the given chromosome by indexing vector `chrom_offset` using i_C and $i_C + 1$, respectively.

Next, these two indices are used to slice dataset `bin_offset`, yielding slice BO. This slice contains the indices of the first pixel overlapping each row between the first and last bins.

Finally, the first row overlapping the given query can be identified through a binary search over BO using $j_0 = i_C + \frac{10{,}000}{bin\ size}$ as key. Similarly, the last row overlapping the query can be computed as $j_1 = i_C + \frac{50{,}000}{bin\ size}$.

At this point, the search for pixels overlapping the given queries can be narrowed down to pixels between j_0 and j_1. There are several optimization opportunities to efficiently traverse this interval of pixels and only return pixels overlapping the query. However, a naive implementation can simply iterate over pixels between j_0 and pixel j_1 and only return pixels overlapping the given query.

Following the above logic, and thanks to the implementation of several optimizations, hictk is capable of running approximately 10 TAD-sized queries per second at 1 kbp resolution on an ultra-deep .cool file (ENCFF447ERX [56], see Paper II).

3.5 Comparing .hic and cooler formats

While the .hic and cooler formats represent the same kind of data, they do so in radically different ways. Cooler takes advantage of the HDF5 format to abstract away low-level Input-Output (IO) operations, greatly simplifying the implementation of libraries reading cooler files. However, this comes with some drawbacks:

- Limited freedom in the choice of how to serialize data to disk.
- Inability to effectively take advantage of multi-threading, as HDF5 achieves thread-safety by wrapping most operations behind a global lock.

The .hic format, on the other hand, does not depend on any library to do low-level IO, making implementing libraries reading this format more complex. However, this gives format designers full control over the layout of serialized data, and there are no limitations imposed by the format itself when it comes to multi-threading.

Another important difference between .hic and cooler is how matrices are stored: cooler uses a single matrix to store all interactions, while .hic stores matrices for each pair of chromosomes separately. This makes cooler inherently better at satisfying queries that require traversing all interactions, while .hic is more efficient at satisfying queries spanning a pair of chromosomes. Furthermore, cooler stores genomic coordinates in a bin table once, and individual pixels refer back to the bin table as appropriate, while .hic stores genomic coordinates directly in interaction blocks. This has important consequences on file sizes, with cooler usually being able to achieve significantly smaller file sizes. However, this comes with a performance penalty when satisfying queries requesting the full genomic coordinates associated with interactions, as this requires an extra lookup in the bin table.

One of the features that is currently only available in .hic files, is the ability to efficiently store expected interactions. Doing the same with cooler requires creating a new file only storing expected interactions, while .hic files can achieve the same purpose by storing one vector (with one entry per bin) for each chromosome.

One flaw of the .hic specification, is the lack of a section in the specification dictating the endianness that should be used when serializing data to disk. In practice, this is rarely an issue, as most modern CPU architecture used in desktop and server environments are little-endian, and so applications reading and writing .hic files happen by chance to run on machines using the same endianness. However, it would be desirable for the .hic specification to be updated to include a section on how to handle endianness. Cooler does not have to dictate this aspect in its specification, as data endianness is transparently handled by the HDF5 library.

3.6 Closing remarks and future perspectives

In this chapter, I summarized available formats that can be used to represent pairwise and binned interactions produced by 3C experiments.

As the field of 3D genomics and 3C matures, there is a pressing need for stable, portable, efficient, and standard file formats. On the front of pairwise interactions, the Pairs standard and Pairix indexing are becoming more and more popular. As for representing binned interactions, the field is currently split across .hic and cooler standards. Given that both formats have their strengths, I see no reason for one to prevail over the other in the near future. In an ideal world, every tool should accept files in both .hic and cooler formats. However, this is in practice rarely the case, as this requires tool developers to duplicate all code to perform IO on interactions matrices, and so we are left with tools that either understand one of the textual formats, .hic or cooler. To make things worse, converting .hic files to cooler and vice versa has historically been difficult, as no tool was able to convert any .hic file to any cooler file and vice versa (see Paper II).

In Appendix B I suggest some ways in which .hic and cooler file formats can be improved. Furthermore, in Paper II and Section 4.8 I present and discuss hictk, a blazing fast toolkit to work with .hic and .cool files. Among other things, hictk enables easy conversion of files in .hic to cooler format and vice versa.

Part II

Aims of the study

The human 3D genome is structured in a dense, yet well-organized manner, packing over 2 meters of DNA into nuclei with diameters in the order of a few micrometers. In the last two decades, we began to understand many of the details that underpin this intricate structure and discovered several processes involved in shaping the 3D genome structure at different levels. 3C and Hi-C experiments have been crucial in this process.

Despite this, the tools to analyze Hi-C data and simulate the recently uncovered processes, such as DNA loop extrusion, are often suboptimal. At the beginning of my PhD, I noticed a lack of user-friendly and high-performance tools to simulate DNA loop extrusion. Back then, existing tools were difficult to use, and due to their computational complexity, they could be used to simulate loop extrusion on relatively small regions, thus limiting the questions that could be answered with these tools. For this reason, I decided to **develop a user-friendly and high-performance stochastic model of DNA loop extrusion** named MoDLE. Next, using this newly developed model, I sought to **infer global and local parameters of DNA loop extrusion**.

While developing MoDLE, I found myself needing to write simulated interactions to disk and found that there were no C++ libraries enabling this functionality. Upon further inspection, I realized that the lack of libraries to write files in common formats used in the Hi-C field was not only an issue for C++ applications like MoDLE, but rather a more widespread problem. Furthermore, the field is split across two file formats: the .hic and cooler formats, with no easy way to convert between the two formats. For these reasons, I sought to **develop a toolkit to transparently and efficiently operate on .hic and cooler files**. Developing this toolkit is not only useful for my own research but also for the scientific community as well, as a significant amount of time is currently being spent dealing with format incompatibilities and other technical details that should not interest end users.

Finally, understanding the 3D structure of genomes and the processes that concur to determine this structure is not an end in itself. A better understanding of the 3D genome will also help us better understand diseases characterized by disorganization of the 3D genome, such as certain types of cancer. This led me to **develop and use a computational approach to unravel multi-level 3D genome alterations during breast cancer progression**.

Part III

Summary of papers

Paper I

MoDLE: high-performance stochastic modeling of DNA loop extrusion interactions
Rossini R., Kumar V., Mathelier A., Rognes T. and Paulsen J.
2022 *Genome Biology* 23, 247 DOI: 10.1186/s13059-022-02815-7

We present MoDLE, a high-performance stochastic model of DNA loop extrusion interactions. MoDLE takes as inputs a list of chromosomes and extrusion barriers and produces an interaction matrix similar to that produced by Hi-C experiments as output. Thanks to several optimizations to the model design and implementation, MoDLE can simulate interactions mediated by DNA loop extrusion on the entire human genome in a matter of minutes. Loop extrusion has been extensively reviewed in Section 1.3.

In brief, MoDLE's algorithm articulates as follows. First, MoDLE generates a sparse representation of the genome by using the list of chromosomes and extrusion barriers provided as inputs. Extrusion barriers are modeled using a two-state (bound and unbound) Markov process where transition probabilities are computed from the occupancy of a given extrusion barrier. LEFs are modeled as pairs of extrusion units bridging together pairs of genomic regions corresponding to the base of loops.

Crucially, MoDLE tracks extrusion barrier and LEF positions with base-pair resolution, meaning that the model could in principle be used to generate interaction matrices at 1 bp resolution. Next, the model enters a burn-in phase, where LEFs are progressively bound to DNA where loop extrusion is simulated as described below. The only difference between the burn-in phase and the rest of the simulation is that during the burn-in phase, interactions are not tracked. The burn-in phase terminates once the system has reached equilibrium. MoDLE simulation is iterative, meaning that the model iteratively performs a set of predefined operations, such as binding LEFs and detecting collisions.

MoDLE's simulation loop begins with binding and unbinding of LEFs to DNA. LEFs can bind any region on the genome with equal probability, while LEF release is biased such that LEFs stalled by a pair of convergent extrusion barriers are less likely to be released. Next, MoDLE generates a set of candidate moves for each LEF. These correspond to the distance a LEF is set to move assuming no collision will take place during the current iteration.

The next step involves updating the extrusion barrier states by computing the next state in the Markov chain used to model barriers. After that, MoDLE proceeds to compute LEF-LEF and LEF-barrier collisions. Collision information is used to update the candidate moves such that they do not violate the constraints imposed by collision events. For example, if an extrusion unit is set to move 100 bp but a collision will occur 50 bp downstream from the current position, then the candidate move for this extrusion unit will be changed to 50 bp.

Next, LEF positions are updated using the adjusted candidate moves. Finally, a subset of the active LEFs are selected and released from the DNA. These

LEFs will be bound to a new genomic region in the next iteration. Once the burn-in phase is complete, every iteration, before moving LEFs, positions of a subset of active LEFs are used to sample molecular interactions. The above description refers to how simulations of individual chromosomes in distinct nuclei are performed. In practice MoDLE runs many simulation instances in parallel by generating a number of tasks based on the parameters provided as input. This allows MoDLE performance to scale linearly with the number of CPU cores, as simulation instances are for the most part independent from one another.

In MoDLE's paper, we compare the interaction matrices produced by MoDLE with Micro-C matrices as well as matrices generated with MD simulations and show that the matrices produced by MoDLE are very similar to both Micro-C and MD-simulated matrices. The paper next proceeds to show potential applications of MoDLE, including how MoDLE can be used to estimate genome-wide as well as local parameters of loop extrusion. Results of the genome-wide parameter optimization highlight the possibility that CTCF may act as a barrier to DNA loop extrusion using a broad range of binding kinetics. In the local optimization, we show how MoDLE can be used to infer extrusion barrier occupancies from Micro-C data.

Besides comparing simulated heatmaps with Micro-C data, we also compare simulated LEF and extrusion barrier occupancies with choesin and CTCF Chromatin ImmunoPrecipitation Sequencing (ChIP-seq) data, respectively. The comparison shows how peaks and valleys in simulated data often correspond to peaks and valleys in ChIP-seq data. Finally, we showcase MoDLE's predictive capabilities by modeling the HoxD cluster and showing how MoDLE can predict the result of perturbing extrusion barriers at the HoxD locus.

MoDLE is written in C++17 and is released under the MIT license. MoDLE's source code is available on GitHub at github.com/paulsengroup/modle. MoDLE's binaries are available on GitHub and bioconda. A containerized version of MoDLE can be downloaded from GHCR.io and DockerHub.

Paper II

hictk: blazing fast toolkit to work with .hic and .cool files
Rossini R. and Paulsen J.
Submitted for publication.

We present hictk, a blazing-fast toolkit to work with .hic and .cool files. hictk is both a Command Line Interface (CLI) application and a library designed to transparently operate on .hic and .cool files (reviewed in Sections 3.4.7 and 3.4.8). The CLI component is built on top of the library components and allows performing many common operations on .hic and .cool files directly from the shell, including converting files in .hic an .mcool format. hictk provides a C++ library with Python bindings that can be used as base to build more complex applications. The library provides a set of functions and classes to perform simple operations such as fetching interactions overlapping a given query, as well as more complex operations, such as merging multiple streams of pixels. The hictk library makes heavy use of iterators to lazily traverse potentially large collections of pixels using very little memory. From the developer's perspective, there is no difference between querying .hic or .cool files. The following Python snippet shows how to fetch interactions overlapping a region of interest from a .hic file using hictk's Python bindings:

```python
import hictkpy

import sys


file = hictkpy.File("interactions.mcool", 10_000)
selector = file.fetch("chr1:10,000,000-20,000,000")
df = selector.to_df()

# Write pixels to stdout
df.to_csv(sys.stdout, sep="\t")
```

To fetch interactions from a .cool file, the only change required would be to specify a different path to `hictkpy.File()`.

In hictk's paper, we compare hictk performance with that of `cooler`, `juicer_tools`, `straw` and `hic2cool`, and find that hictk consistently outperforms other tools across a broad selection of benchmarks. As an example, one of our benchmarks shows that `hictk convert` is up to 42 faster than `hic2cool` when converting a large .hic file to .mcool format. Another benchmark shows that hictk is up to 1.5 times faster than `straw` and up to 20 faster than `cooler` at reading interactions for chromosome 1 and writing them to standard output. In the paper, we also show how trivial it is to modify existing applications to use hictk instead of cooler to read interactions from .hic and .cool files.

hictk is written in C++17 and is released under the MIT license. hictk's source code is available on GitHub at github.com/paulsengroup/hictk. hictk's binaries are available on bioconda. A containerized version of hictk can be downloaded from GHCR.io and DockerHub.

Python bindings for the hictk library are available on GitHub at github.com/paulsengroup/hictkpy, bioconda and PyPI.

Paper III

Multi-level 3D genome organization deteriorates during breast cancer progression
Rossini R., Oshaghi M., Nekrasov M., Bellanger A., Domaschenz R., Dijkwel Y., Abdelhalim M., Collas P., Tremethick D. and Paulsen J.
Submitted for publication.

We report here our study on the multi-level alteration of genome organization and expression during breast cancer progression. Using an *in vitro* model of breast cancer (MCF10A, MCF10AT1 and MCF10Ca1a) we investigated changes in 3D genome organization and expression associated with cancer progression. To this aim, we generated Hi-C, RNA-seq, and Lamin B1 ChIP-seq data and used publicly available datasets for several epigenetic marks. Using Hi-C, we generate annotations of TADs, (sub)compartments, and TAD cliques (reviewed in Sections 1.2.2, 1.2.4 and 1.2.5). Overall we find that TADs remain stable during cancer progression, however, significant changes occur at the level of TAD-TAD interactions, (sub)compartments, and gene expression.

First, we report major (sub)compartment switching throughout cancer progression, with weak subcompartments, such as B0 and A0 subcompartments showing a greater tendency to switch compared to stronger subcompartments such as B3 and A3. Overlapping our subcompartment annotation with 3D *in silico* models of the genome shows how the well-behaving radial positioning of subcompartments in MCF10A is degraded in MCF10AT1 and MCF10Ca1a, showing that radial positioning becomes compromised as cancer progresses to later stages.

After describing the association between cancer progression and 3D structure deterioration at the (sub)compartment level, we investigated the relation between subcompartment switching and gene expression. We find that subcompartment switches are often associated with changes in expression levels, where switches towards open chromatin correspond to expression upregulation, while switches towards closed chromatin correspond to downregulation of gene expression.

Next, we characterize changes in TAD-TAD interactions by analyzing TAD cliques across the three stages. We find that TAD clique composition throughout cancer progression is highly dynamic, exhibiting pervasive changes in maximal clique sizes. By overlapping our subcompartment annotation with TAD cliques, we find that larger cliques mostly consist of heterochromatic TADs, with cliques of size 6 or greater almost exclusively consisting of regions annotated as B3 subcompartments. Strikingly, in MCF10AT1 and MCF10Ca1a, large cliques become more associated with regions annotated as B2 subcompartments. Next, we cluster TAD cliques based on their subcompartment composition. Clustering shows that TAD cliques often contain TADs belonging to different subcompartments. Furthermore, we find that TAD cliques consisting of different types of A-subcompartments are depleted in MCF10AT1 and MCF10Ca1a compared to MCF10A.

Part IV

Discussion

Chapter 4

Main projects

During my PhD I have spent a significant amount of time designing, implementing and using bioinformatics tools. The first year of my PhD was almost entirely devoted to designing, implementing and testing MoDLE (Paper I).

While developing MoDLE, I found myself needing to write interactions mediated by DNA loop extrusion to disk, which led me to develop a library to read and write cooler files. This library was initially part of MoDLE, but was later extracted and overhauled into a short-lived library, named coolerpp. Coolerpp source code is now part of the hictk code base (Paper II). hictk itself was developed while working on Paper III.

While working on this paper, at one point we wanted to assess the impact of different matrix balancing algorithms on TAD calling. At that point, we were mostly working with interactions in cooler format, however some balancing algorithms can only be computed when working with .hic files. Unfortunately, extracting balancing vectors from .hic file turned out to not be trivial. This, and other issues faced when working with .hic and cooler files prompted us to develop hictk.

After carefully studying the source code of straw, the only available library capable of reading .hic files, we decided to write our own library to read .hic files (see Paper II), as straw is affected by several important flaws discussed in Section 4.8. This later turned out to be beneficial for hictk performance, as having full control over code reading .hic files and writing cooler files opened up possibilities for several optimizations. As Paper I (and later Paper II) involved a significant amount of software development and relatively little data analysis, towards the end of the first year of my PhD, I decided to embark in a project involving data analysis of cancer progression, which eventually resulted in Paper III.

The rest of this thesis is dedicated to a discussion of the three projects that resulted in Paper I, Paper II, and Paper III. The discussion also includes a section on the reproducibility of bioinformatic analyses and a section about other projects I took part during the course of my PhD.

4.1 MoDLE: Modeling of DNA Loop Extrusion

MoDLE was the first project I began working on right after moving to Norway and starting my PhD. Back then I was vaguely aware of the process of DNA loop extrusion and had no experience working with Hi-C data. However, after spending the first 1–2 months familiarizing myself with the field, I felt ready to start developing an *in silico* model of DNA loop extrusion.

4.2 Why develop MoDLE?

We decided to develop MoDLE for several reasons.

First, we saw a lack fo tools to simulate DNA loop extrusion, especially a lack of efficient and easy to use tools. Back in 2020, virtually all attempts at modeling DNA loop extrusion relied on MD simulations. While MD simulations offer great flexibility in how a process like loop extrusion can be modeled, they are extremely compute-intensive. Furthermore, MD simulations are not trivial to design and implement, often requiring the involvement or at least the supervision of a MD expert. Finally, it was not clear to us whether full-fledged MD simulations were required to simulate loop extrusion interactions.

Second, the few available tools, were difficult to use. Reproducing the results of MD simulations was often not possible due to the low quality or complete lack of simulation code. This was preventing us, and likely many others, from reproducing the results from scientific publications, as well as using existing simulations as a starting point for our own simulations. Another issue with MD simulations of DNA loop extrusion, is that they often require access to one or more NVIDIA GPUs for the simulation to run in a matter of days instead of weeks or months. Unfortunately, many compute clusters as well as individual research groups do not have access to such GPUs.

Third, the computational demands of MD simulations do not support chromosome-level or genome-wide simulations of loop extrusion, limiting our ability to better understand the process. For example, without genome-wide simulations, systematic identification of regions whose 3D structure is not primarily determined by loop extrusion is not possible. Furthermore, there is often no Hi-C data available for less commonly studied species and tissues. A tool enabling genome-wide simulation of loop extrusion would allow us to predict what the Hi-C data would look like at the TAD-scale.

Given the above points, we saw a need for a user-friendly tool for *in silico* simulation of DNA loop extrusion at the genome-scale. It was originally thought that the main function of TADs produced by DNA loop extrusion was that of facilitating enhancer-promoter interactions of elements within the same TAD and obstructing other types of interactions. Accordingly, our vision was to apply MoDLE to predict the effect of perturbations at the TAD level on enhancer-promoter interactions, and thus potentially gene expression.

4.3 A brief history of MoDLE

The current version of MoDLE (MoDLE v1.1.0) is a relatively large and complex C++17 project, counting close to 30 thousand lines of code. However, MoDLE started as a much simpler project, counting less than 10 thousand lines of code.

The initial version of MoDLE, v0.1.0 (09/03/2021), represented the genome as a dense vector of bins, where each bin represented a fixed amount of DNA. Bins were also used to track which LEFs and extrusion barriers were bound to the DNA represented by the bin in a given simulation epoch. Upon binding,

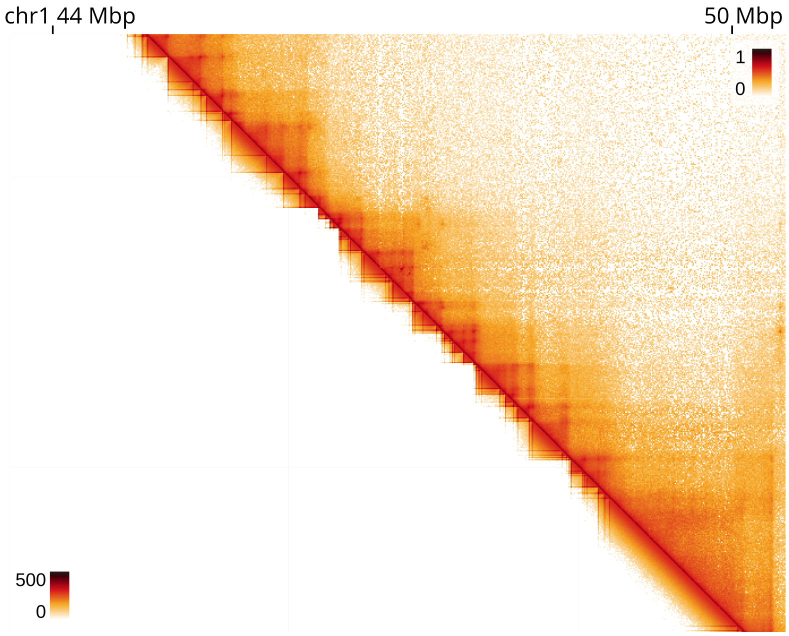


Figure 4.1: Comparison between heatmaps generated by MoDLE v0.1.0, (below the diagonal) and Micro-C (4DNFI9GMP2J8, above the diagonal) at 10 kbp resolution.

each LEF was assigned a lifetime, that is the number of epochs the LEF-DNA binding was supposed to last. Upon collision events, a stochastic process was used to determine the number of epochs a given LEF was not allowed to extrude in the direction affected by the collision.

This design was simple to understand and implement, however, it was affected by several issues (see Appendix A for more details). One of the issues was performance. After profiling the execution time of this version of MoDLE, it was clear that most of the CPU time was spent breaking and forming DNA bonds between adjacent bins. Another factor affecting performance was the low level of parallelism achievable with this implementation, as concurrency was limited by the number of chromosomes to be simulated. This design was also affected by a logic bug that caused extrusion barriers to appear as blocking (i.e., bound by CTCF) and non-blocking (i.e., not bound by CTCF) in the same epoch, depending on which LEFs queried barriers for their occupancy. This bug was due to the stochastic process used to simulate LEF-barrier collision events.

Despite the outlined issues, heatmaps produced by MoDLE v0.1.0 recapitulated important features of Hi-C data, such as stripes and dots (see Figure 4.1). Not being happy with the computational performance of our model, we went back to the drawing board to develop MoDLE v0.2.0.

The differences between MoDLE v0.1.0 and v0.2.0 (21/03/2021) are too many to exhaustively list here (see Appendix A for more details). In brief, MoDLE v0.2.0 uses a sparse representation of the genome, where bins are no longer explicitly modeled. Instead, MoDLE keeps a sorted list of extrusion barriers and extrusion units (i.e., the reverse- and forward-moving components of LEFs). We developed a family of algorithms based on the merge part of mergesort to detect and process collision events in linear time complexity. These algorithms rely on the fact that extrusion units and extrusion barriers are always sorted by their genomic coordinates. To improve the level of simulation concurrency, we introduced the notion of simulation instances. Instead of running one simulation per chromosome, we now run several simulation instances per chromosome. Simulation instances are largely independent from one another, and can thus run in parallel without requiring extensive synchronization.

The result of the above changes led to a dramatic performance improvement, with MoDLE v0.2.0 taking only 4 minutes 6 seconds to simulate loop extrusion on hg38 using 256 CPU cores versus over 8 hours required by v0.1.0). That being said, the model was still susceptible to falling into an inconsistent state due to the way collisions were handled.

It was in MoDLE v0.3.0 (14/06/2021), that the logic bug affecting previous versions was finally fixed. Addressing this bug required a significant rework of how extrusion barriers were modeled and how collisions were handled (see Appendix A for more details). Extrusion barriers are now modeled as a two-state (bound and unbound) Markov process, such that at any given time, an extrusion barrier instance is either bound or unbound. Changes in this version also include improvements to how LEFs are loaded and released, changes to how collisions are tracked, improvements to MoDLE's test suite, CLI interface, and several changes needed to make MoDLE's output consistent across platforms. The simulation algorithm of MoDLE v0.3.0 was also implemented using CUDA. However, this implementation did not yield the expected performance improvements, and so the GPU implementation was abandoned in favor of the CPU one.

MoDLE v0.3.0 was followed by version 0.4.0 and 0.4.1, which involved the additions of features that are no longer part of MoDLE. After MoDLE v0.4.1 we published 7 release candidates for v1.0.0, where we fixed many bugs, improved the CLI interface, and further optimized performance. Versions 1.0.0-rc.3 and 1.0.0-rc.7 were used to produce the results presented in Paper I.

The latest version of MoDLE is v1.1.0. At its core, the simulation algorithms used by MoDLE v0.3.0 and v1.1.0 are very similar. Figures 4.1 and 4.2 shows a comparison of the heatmaps produced by MoDLE v0.1.0, v1.1.0 with Micro-C heatmaps. Simulating loop extrusion on hg38 using 256 CPU cores with MoDLE v1.1.0 takes 2 minutes 59 seconds.

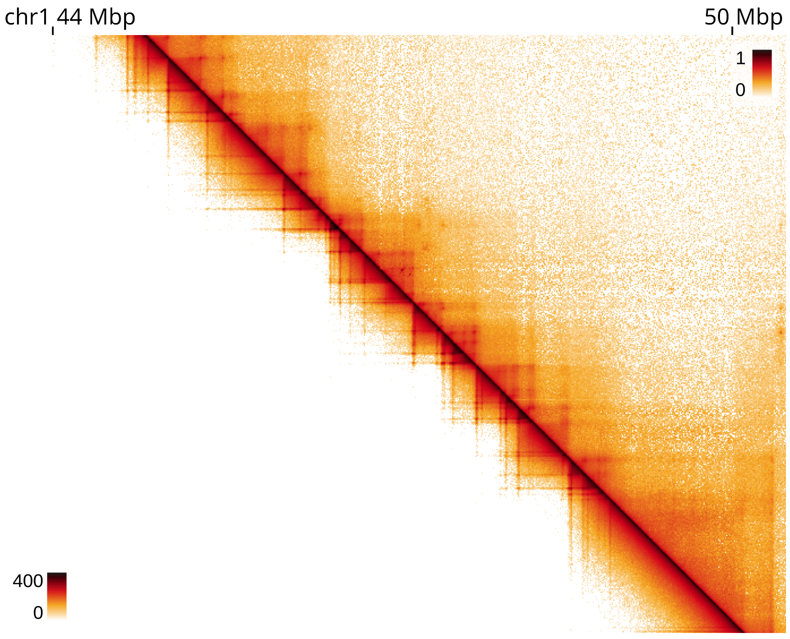


Figure 4.2: Comparison between heatmaps generated by MoDLE v1.1.0, (below the diagonal) and Micro-C (4DNFI9GMP2J8, above the diagonal) at 10 kbp resolution.

4.4 Automating the build process of complex C++ applications

As pointed out in Section 4.3, MoDLE has grown into a fairly large and complex C++ application. The latest version of MoDLE counts close to 30 thousand lines of code, organized into 19 internal libraries and depending on over 25 external libraries.

In languages like Python, Rust, or R, installing and importing third-party libraries is trivial, as all three languages provide standard and mature facilities to express software dependency graphs. Unfortunately C++ does not have a standard build system, nor does it have a standard package manager, making dependency management much more cumbersome. C++ applications with third-party dependencies often require the users to install build, and runtime dependencies manually. This can quickly spiral out of control, as third-party dependencies can themselves depend on other libraries. Furthermore, manually installing C++ dependencies may require compiling dependencies, and injecting the path to libraries and header folders into the build system, which is usually prohibitively difficult for non C++ developers. Finally, there is no guarantee that once all dependencies have been manually installed the build system will be able to find all required files, nor that the version of packages installed are compatible. All of the above makes building bioinformatics software written in C or C++ too complex for the average user.

In the case of MoDLE, we attempt to mitigate the above issues in three ways. First, every time a commit is pushed to MoDLE's main branch, we build and publish MoDLE as a containerized application, so that users can use any version of MoDLE without any setup required. Second, we contributed a recipe for MoDLE to bioconda, so that users can easily install MoDLE using conda. Third, we designed MoDLE's build system to automate as many configuration and build steps as possible, such that building MoDLE doesn't require prior experience building C++ applications.

Configuration and build steps are automated with CMake and Conan, which can be easily installed through the OS package manager or pip, Python's package manager.

Another complication of C++ applications, is that by default applications tend to use dynamic linking, meaning that every time an application is launched, it has to look for all its dependencies. Dynamic linking and shared libraries have their place, however in the context of bioinformatics software, in my opinion, the benefits of dynamic linking do not justify its drawbacks. From a practical perspective, the main drawback of dynamic linking is that updating dependencies can break existing applications. However, this usually will go unnoticed until the next time the application is launched. To avoid this issue, MoDLE is by default built with static linking, meaning that compiled code for third-party libraries is directly embedded in MoDLE's binary. This means that future updates to the OS or other packages are highly unlikely to break MoDLE. The disadvantages of static linking are that it tends to produce large binaries and that applying security updates requires recompiling the entire application. However, given that binary sizes of statically linked bioinformatics tools rarely exceed 10 megabytes, and that most bioinformatics tools are not very exploitable from a security perspective, these disadvantages are fairly limited for most bioinformatics applications and certainly form MoDLE. Static linking is becoming much more popular, to the point where there are now compiled languages such as Go and Rust that build libraries and binaries with static linking by default.

Unfortunately, approaches to build and test automation such as those used by MoDLE are rarely used in C++ bioinformatics applications. This is not without good reason, as developing and maintaining these systems requires a significant amount of work from the developer side, namely (1) knowledge of at least another language (usually CMake or make), (2) familiarity with a package manager such as Conan or vcpkg, and (3) familiarity with a Continuous Integration (CI) pipeline such as GitHub Actions to automate build, test, and deployment. Ultimately, it would be desirable for C++ to standardize a build system and dependency management system. However, as this seems unlikely to happen in the near future, we must resort to other solutions to streamline installing C++ applications.

Conda and more specifically bioconda, provide a way to easily install bioinformatics applications. However, bioconda does not support Windows as a platform. Furthermore, macOS support is limited to the x86 architecture, meaning that the new machines with CPUs based on Apple Silicon are not natively supported. These machines can still install applications through bioconda, however, they will incur a performance penalty, which can be significant for C++ applications, as x86 applications installed through bioconda are executed using the Rosetta 2 compatibility layer. Finally, package managers such as bioconda encourage the use of dynamic linking, which as outlined in previous lines, makes applications more prone to breaking due to dependency updates.

The dynamic linking model poses another issue for applications installed through conda, namely the chance of generating dependency graphs that cannot be solved. Consider two applications, A and B. Both applications depend on library C, however, A depends on C v1 while B requires C v2. In the above case, it will not be possible to install applications A and B in the same environment, as conda does not support installing multiple versions of the same package side-by-side. With static linking, the above issue would not be possible. There are other package managers, such as spack that support installing multiple versions of the same software side-by-side. However, spack requires building all dependencies from source, which takes much longer than installing applications through conda. Furthermore, build issues encountered while using spack are often very difficult to debug even for experienced users, as users have to chase issues that happened in an isolated build environment that is managed by spack.

Another approach to solving package management in bioinformatics is containers. Containers sidestep the dependency management issue by bundling software, libraries, and configuration files into container images, such as Docker or Apptainer images. Using containerized applications does not usually involve any set up step, only downloading the container image from DockerHub or GHCR.io.

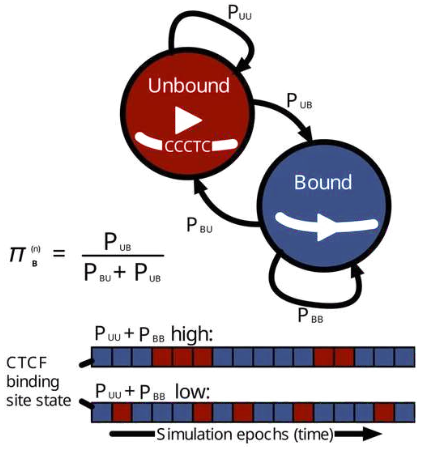
This allows users to effortlessly use applications as setup and configured by the application's developers. However, containers can be quite large for applications with complex dependencies. Furthermore, combining multiple applications into a single container often requires writing a new recipe from scratch. This means that if a user wants to use applications A and B in the same environment, it does not matter that containers for applications A and B are available individually. If container A and B use different base images, or if A and B are not simple applications that can be installed by copying a handful of files, then the two containers cannot be easily combined.

While package managers like conda and distribution systems based on containers are greatly simplifying the distribution of bioinformatics applications, we do not yet have a silver-bullet solution to easily consume any bioinformatics package. Simpler solutions, like bioconda, do not natively support all platforms, while more complex solutions such as containers can in principle support any platform but are less user-friendly to use and combine. An emerging solution to connect multiple containers in a data analysis pipeline are workflow systems such as Nextflow and Snakemake (reviewed in Chapter 6). The resulting workflows are easy to use across platforms, however writing or modifying the workflows requires knowledge of yet another programming language and programming model, thus excluding less technically inclined users.

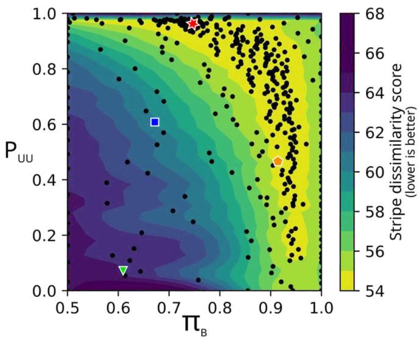
4.5 Applications of MoDLE

The ability to accurately simulate loop extrusion genome-wide and with excellent performance makes MoDLE suitable for several applications that were not previously possible. In Paper I we demonstrate three possible applications of MoDLE.

4.5.1 Genome-wide optimization of loop extrusion parameters



(a)



(b)

Figure 4.3: MoDLE genome-wide parameter optimization. **A** Diagram of the Markov process used to simulate extrusion barriers by MoDLE. The diagram with red/blue boxes shows the chain of states assumed by an extrusion barrier instance given different parameter combinations. **B** Density plot showing the results of the MoDLE's genome-wide parameter optimization. The plot shows MoDLE's performance across the searched parameter space. Red star, orange pentagon, blue square, and green triangle show optimal, near-optimal, suboptimal, and non-optimal parameter combinations, respectively.
Figures 4A and 4C from *MoDLE: high-performance stochastic modeling of DNA loop extrusion interactions* - 10.1186/s13059-022-02815-7.
Figures licensed under the CC BY 4.0 DEED license.

Small-scale *in silico* simulations of DNA loop extrusion have been instrumental to better understanding the core principles and dynamics of DNA loop extrusion. However, the inability to simulate loop extrusion genome-wide limits the extent to which we can understand this process. For example, without genome-wide simulations, it is not possible to generate a genome-wide annotation of regions whose 3D structure is not primarily determined by loop extrusion. Another application that requires performing genome-wide simulation is the unbiased estimation of general loop extrusion parameters.

In Paper I we take advantage of MoDLE's ability to simulate loop extrusion genome-wide to optimize loop extrusion parameters along the entire genome instead of small, selected regions like was previously done. More specifically, we were interested in characterizing the parameters governing the binding kinetics of extrusion barriers modeled as CTCF barriers. This was interesting to us, as it was previously described that CTCF residence time is approximately one minute, while cohesin's residence time is in the order of 30 minutes [231], however, it was not known whether this is the only regime where CTCF can efficiently stall LEFs, or if other regimes are possible. Showing that CTCF can act as a barrier to DNA loop extrusion under different regimes suggests that CTCF binding kinetics may vary along the genome, in different tissues, or even different species.

As outlined in Paper I, Section 4.3 and shown in Figure 4.3a, MoDLE models extrusion barriers using a two-state Markov process (bound and unbound). The dynamics of the Markov process are determined by three variables: (1) P_{BB}, (2) P_{UU}, and (3) π_B. P_{BB} is the probability of self-transition to the "bound" state, while P_{UU} is the self-transition probability to the "unbound" state, and π_B is the probability of binding for a CTCF binding site in a simulation epoch. These transition probabilities can be inferred from cohesin or CTCF ChIP-seq as outlined in Paper I and references [131, 155]. However, in the MoDLE paper we sought to optimize π_B and P_{UU}, genome-wide, to maximize similarities between heatmaps simulated by MoDLE and Micro-C heatmaps. Parameters were optimized with Bayesian optimization using Gaussian processes (see Paper I for more details). The objective function used for optimization was designed to detect similarities in stripe position and length given a pair of interaction matrices (see Paper I for more details). We find that the optimal parameter combination is $\pi_B = 0.747$ and $P_{UU} = 0.963$. This suggests that the dynamics of CTCF barriers are best recapitulated by relatively long intervals where the binding site is occupied, followed by long, but comparatively shorter intervals where the binding site is not occupied. However, as shown in Figure 4.3b, there is a broad range of parameter combinations that can yield near-optimal heatmaps. More specifically, average occupancy π_B can assume values between 0.6 and 0.9 as long as P_{UU} is relatively high (> 0.8). Near-optimal results can also be achieved when occupancy is very high (> 0.9) and P_{UU} is relatively low (< 0.8).

This suggests that CTCF can be an effective barrier to loop extrusion under two regimes: one characterized by lower occupancies and with high binding stability, and the other with high occupancy but low binding stability.

4.5.2 Local optimization of loop extrusion parameters

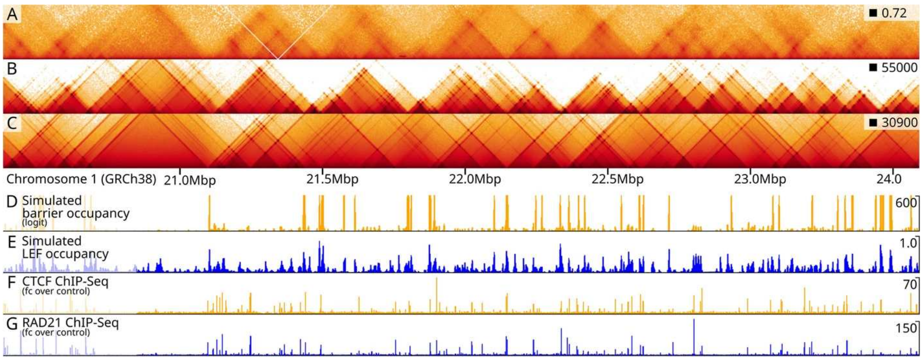


Figure 4.4: Result of the local parameter optimization using MoDLE. **A** Micro-C data from a 5 Mbp region from chromosome 1. **B** Output produced by MoDLE for the same region after the optimization of extrusion barrier parameters. **C** Output produced by MoDLE for the same region using barrier parameters inferred from RAD21 ChIP-seq data. **D-E** Panels showing the average extrusion barrier and LEF occupancy profiles generated by MoDLE. **F-G** Panels showing the fold-change over control for CTCF and RAD21 ChIP-seq data.
Figure 6 from *MoDLE: high-performance stochastic modeling of DNA loop extrusion interactions* - 10.1186/s13059-022-02815-7.
Figures licensed under the CC BY 4.0 DEED license.

Another application of MoDLE presented in Paper I, is the local optimization of loop extrusion parameters. This time we optimized the transition probabilities of individual extrusion barriers in a 5 Mbp region harboring over 2100 candidate CTCF binding sites. We were interested in learning whether MoDLE could be used to infer loop extrusion parameters from Micro-C data alone.

With this aim, we developed an optimization strategy based on genetic algorithms to optimize π_B using a modified version of the objective function employed in the genome-wide parameter optimization (see Paper I for more details).

The results of this optimization show that in the absence of ChIP-seq data, MoDLE can infer individual CTCF barrier occupancies from Micro-C data (see Figure 4.4). We compare the simulated LEF and CTCF occupancy with RAD21 and CTCF ChIP-seq data, and find that peaks and valleys in simulated data often coincide with those found in experimental data (see Figure 4.4), suggesting that the parameters inferred and the process simulated by MoDLE are biologically meaningful.

One extension to this optimization strategy could be to generate candidate extrusion barrier binding sites by e.g., placing a pair of divergent barriers every 100 bp, then optimizing their occupancy. Doing so would yield a profile of extrusion barrier occupancy that includes, but is not limited to, CTCF barriers, thus aiding the characterization of extrusion barriers other than CTCF.

4.5.3 Predicting the effects of TAD border alteration

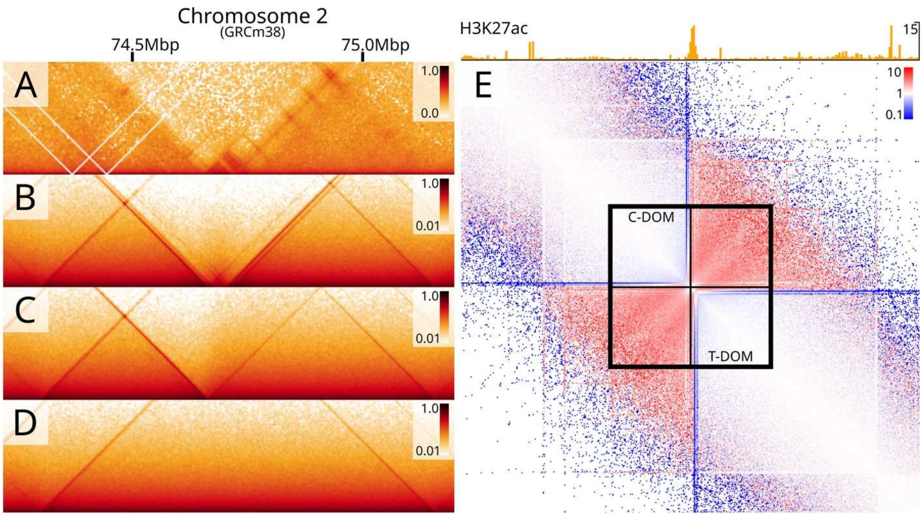


Figure 4.5: MoDLE simulations used to predict the effects of deletions to TAD borders at the HoxD locus.

The last application we display in Paper I, is using MoDLE to predict the effects of extrusion barrier alterations at the HoxD cluster locus. This is an important validation step for our model and will inform us as to whether MoDLE can be used to predict the effects of local perturbations to extrusion barriers.

First, we inferred extrusion barrier occupancies from Micro-C data as outlined in Section 4.5.2. Next, we *in silico* inactivated barriers separating the centromeric and telomeric domains of the HoxD cluster (C- and T-Dom, respectively). Inactivation or deletion of these barriers is known to lead to increased interactions between the two domains [232, 233].

MoDLE's simulations correctly predict that inactivating barriers separating the T- and C-Dom causes a loss of insulation. As shown in Figure 4.5, this leads to the merging of TADs separating the two domains. Thus, we conclude that MoDLE can be used to test the effects of local perturbations to DNA loop extrusion.

4.6 Current limitations of MoDLE

Several factors are limiting MoDLE accuracy, applications, and overall usefulness.

As reported by others and us [131], comparing the result of simulations with real Hi-C or Micro-C data yields mixed results. For some regions, the level of agreement between simulated and observed heatmaps is strikingly high, while for other regions the level of agreement is much poorer. The main issue here is the availability of accurate extrusion barrier annotations, both in terms of barrier positions and strength. It is common practice to generate extrusion barrier annotations to simulate loop extrusion from CTCF or RAD21 ChIP-seq data. However, as outlined above, this often yields mixed results, suggesting two things: (1) loop extrusion as it is currently understood, may not be the only process involved in shaping the 3D genome structure at the TAD level; (2) CTCF, RNAPII, and MCMs may not be the only molecules able to stall loop extrusion. This second point is substantiated by evidence produced by Hsieh et al. [50] as well as by us (unpublished results), showing that DNA accessibility is a better predictor for extrusion barrier occupancy. This is an intriguing possibility, both from a mechanistic point of view, but also for potential applications of MoDLE, as data such as Assay for Transposase-Accessible Chromatin using Sequencing (ATAC-seq) are more widely available than CTCF or RAD21 ChIP-seq. Thus, a combination of CTCF candidate binding sites and ATAC-seq could be used to simulate loop extrusion for tissues or species where CTCF or RAD21 ChIP-seq data is lacking. Candidate binding sites can be easily obtained by mining the reference genome using tools such as MAST [234], while DNA accessibility can be inferred from DNase-seq or ATAC-seq data.

In the case of MoDLE, simulation accuracy could also be affected by the fact that MoDLE does not currently support simulating barriers other than CTCF, such as RNAPII and MCMs, nor it supports simulating A/B compartmentalization.

Another aspect that is not currently possible to simulate with MoDLE, is the preferential loading of LEFs at specific loci or regions, such as preferential loading at enhancers and promoters.

Next, because MoDLE does not explicitly simulate polymer dynamics through e.g., MD simulations, the resulting heatmaps only contain interactions directly mediated by loop extrusion, meaning that data for regions far from the matrix diagonal is either very sparse or missing altogether. This makes it difficult to compare MoDLE output with that of Hi-C or Micro-C experiments, as usual approaches such as matrix correlation result in very low correlation values that do not reflect similarities observed when visually inspecting simulated matrices. The uninformativeness of correlation metrics when comparing matrices produced by MoDLE and Hi-C or Micro-C matrices was one of the reasons we developed a custom objective function to use for parameter optimization, as using correlation metrics led to parameter combinations that were obviously incorrect (e.g., average LEF lifetimes of approximately 0 and extremely high or extremely low extrusion barrier occupancies). This also suggests that loop extrusion itself may not be involved in establishing intra-TAD interactions and that other processes, possibly similar to the hypothesized process of phase-separation are responsible for the formation of TAD-cliques. Furthermore, the fact that TAD-cliques are often associated with the nuclear lamina suggests that a subset of TAD-TAD interactions are driven by TAD co-localization at the nuclear lamina.

Finally, there are many differences between MoDLE's approach to simulating loop extrusion and MD approaches. One of the differences is computational efficiency, where MoDLE outclasses MD approaches. However, extending MD models to change how loop extrusion is simulated is much simpler than modifying MoDLE, as the latter requires C++ proficiency and deep knowledge of MoDLE's code base. This lack of flexibility from MoDLE's part, is the result of tradeoffs between performance optimizations, user-friendliness, and extensibility.

4.7 MoDLE: future perspectives

At the time of writing, MoDLE has received a fair bit of attention from the scientific community. However, the use of MoDLE outside of our group is fairly limited, suggesting that our users are facing difficulties when trying to apply MoDLE to answer their research questions. As discussed in the previous section, there are currently several limitations preventing MoDLE from being more widely used. One of them is that while it is easy to use MoDLE to generate a .cool file with interactions resembling that of Hi-C and Micro-C, quantitative analysis of this data is a bit cumbersome, as comparing simulated data with experimental data is not trivial. The MoDLE package consists of two commands: `modle` and `modle_tools`, the first simulates loop extrusion interactions, and the latter includes several subcommands to preprocess MoDLE's inputs or postprocess MoDLE's output.

To improve MoDLE usability, future work should focus on expanding `modle_tools` to provide easy and clear workflows that take MoDLE-simulated heatmaps as input, and produce biologically meaningful results as output. For example, in Paper I we show that MoDLE can recapitulate the effects of local perturbations of extrusion barriers at the HoxD cluster. This suggests, that MoDLE could be used to inform and guide experiments using genome-editing to perturbate loop extrusion on a region of interest. To support this use case, we could develop a workflow that programmatically perturbates extrusion barriers and simulation parameters over a genomic region of interest, then rank extrusion barriers based on the magnitude of the effect of their perturbation. The feasibility of such a workflow depends on the availability of reliable annotations of extrusion barrier occupancies, as well as the ability to robustly compare heatmaps generated by different runs of MoDLE.

Another potential application of MoDLE is that of classifying the genome into two categories: (1) regions whose 3D structure is largely determined by DNA loop extrusion, and (2) regions whose 3D structure is determined by other processes. This information could be used by studies trying to discover and describe new processes involved in the local 3D structure of genomes. However, such a workflow depends on our ability to compare MoDLE's output with experimental data in a meaningful way.

Finally, another application of MoDLE could be that of generating genome-wide annotations of TADs for species who lack publicly available Hi-C data but that have other types of data such as ATAC-seq or CTCF ChIP-seq. These kinds of applications rely on the availability of robust inference methods to generate reliable extrusion barrier annotations from epigenetic data. Unfortunately generating such annotations is not yet a solved problem.

4.8 hictk: blazing fast toolkit to work with .hic and .cool files

hictk is one of the projects I started working on after MoDLE's paper was published. As I will briefly explain in Section 4.10, in practice hictk development overlapped with MoDLE development as well as with the data analysis for Paper III.

4.9 Why develop hictk?

hictk spun out of our need for better interoperability between .hic and cooler file formats.

The .hic and cooler file formats are the *de-facto* standard formats used to store pairwise interactions produced by 3C experiments (see Chapter 3 for more details). However, interoperability between the two formats has historically been rather poor. This was due to the lack of tools that can easily convert between the two file formats. The state of library support to read and write both formats is also not ideal. The cooler Python library can be used to read and write cooler files, while .hic files can be read using the straw library, which is available for C++, MATLAB, Python, and R. However, there is no library support to create .hic files. Furthermore, cooler and straw Application Programming Interfaces (APIs) are significantly different. Thus, writing applications capable of reading interactions from .hic and cooler files requires writing IO code twice, once for .hic files and another time for cooler files. This means that, in practice, most applications can read interactions only from .hic, cooler, or text files, leaving the burden of format conversion on the user. Unfortunately, converting between file formats is not trivial, and sometimes is not possible at all (e.g., converting .hic v9 files to .mcool format was not possible using existing tools). For these reasons, I decided to develop hictk, a blazing fast toolkit to work with .hic and .cool files.

While designing the toolkit, I considered re-using existing libraries for IO on .hic and .cool files, however, I decided to develop hictk from scratch and implement functionality from these libraries from scratch for the following reasons. Cooler is an excellent library to read and write cooler files. The library is written in Python and has a clear, powerful, and thoroughly tested API. Unfortunately, the fact that the library is written in Python makes it very difficult to use outside of Python applications. The straw library is written in C++ and provides bindings for a multitude of languages. Unfortunately, straw API is not as powerful as that of cooler. For example, straw's C++ public API is limited to reading interactions into a vector of triplets (pos1, pos2, count) or as a dense matrix (implemented as a vector of vectors that does not support any operation beside iterations and accessing elements by index). Furthermore, the C++ implementation is affected by several bugs and design issues. For example, the library leaks memory every time the `straw` function is called, causing applications that perform a large number of queries to eventually run out of memory.

Another issue is that the library communicates errors to the user by printing messages to stdout or stderr, then exiting in a way that cannot be caught by applications. This approach to error handling is also followed in case of recoverable errors, such as queries on non-existing chromosomes. The exit signal issued by straw when errors are encountered cannot be caught or avoided, not even in higher-level languages such as Python. As a result, performing a query on invalid chromosomes from Python results in the Python interpreter to quit upon errors.

Finally, contrary to what is stated on straw's GitHub repository ("Straw: rapidly stream data from .hic files"), straw does not stream interactions, but rather reads all interactions into a vector, then returns that vector. This means that performing large queries on high-resolution interaction matrices requires a large amount of memory. I considered, addressing the above issues and contributing my changes upstream, however, the amount of changes required together with the lack of a test infrastructure to ensure my changes did not break straw made me shy away from this option.

Difficulties with format conversions are not only technical issues that individual researchers have to deal with but also an issue for consortia such as the 4DNucleome and ENCODE. For example, the 4DNucleome provides Hi-C datasets in .mcool and .hic format. Surprisingly, downloading files in both formats and comparing them with e.g., hictk reveals that the two files are not identical, but rather have small differences, which are likely due to how pair-wise interactions are binned by the pipelines that generate cooler and .hic files. The ability to reliably convert between the two formats would allow these consortia to use a single pipeline to generate files in .hic or .mcool format, then use a conversion tool such as `hictk convert` to convert files to the other format.

Another consequence of poor format interoperability, is the segregation of the field into at least two separate communities, one using Hi-C data for assembly purposes and the other using Hi-C data for genome analysis. The first community almost exclusively uses files in .hic format, as most tools for genome assembly expect interactions to be in .hic format, while the latter uses a mix of .hic and cooler files, depending on the specifics of the data analysis that is being carried out. Nowadays, a large amount of data is generated for assembly purposes. However, there is no biological reason why these datasets could not be used for other purposes. Unfortunately, difficulties with format conversion are one of the reasons why this is rarely done.

4.10 History of hictk

hictk history is a bit more intricate than one may think. While hictk in its current form was conceptualized and implemented during the summer of 2023, many of its components were developed earlier than that.

The first component that I developed, and that is now part of hictk, is the library part to read and write cooler files. The first version of this library dates back to early spring 2021. Back then I was working on MoDLE and realized that I needed a reasonable, high-performance way to write interactions to disk. This resulted in the development of a small library part of MoDLE code base that supported the bare-minimum operations required for MoDLE to be able to write interactions to disk. Later in the same year, the library was expanded to also support reading interactions, as this functionality was now required by certain subcommands of `modle_tools`. Fast forwarding to April 2022, this library was now mature enough to be used as a general-purpose library to read and write cooler files, so I decided to refactor the code into a library named coolerpp. coolerpp was a short-lived library that was not publicized and, as far as I am aware, was never used outside of our group.

In late 2022 I was working on the data analysis for Paper III, and needed to convert matrices in .mcool format to .hic format, and then back to .mcool format. The workflow to perform the above operations turned out to be relatively complex due to format incompatibilities and issues when trying to read .hic files (e.g. the issues outlined in Section 4.9). This led me to develop a C++ library named hicxx that could read interactions from .hic v8 and v9 files. Using coolerpp and hicxx, I wrote a tool named hic2cool-ng (hic2cool next-generation) to convert .hic files to .mcool format. In the process, I realized that coolerpp and hicxx could have been much more useful if combined into a single library.

This is how hictk was born. hictk's repository was created on June 8, 2023, by merging and refactoring code from hicxx and coolerpp. After merging and integrating the two codebases, I worked on performance optimizations and added the functionality required to implement the commands that are now part of the hictk CLI tool, such as the ability to merge streams of pixels, validating .hic and cooler files, coarsening streams of pixels and much more.

4.11 The importance of documentation

No matter the quality and performance of software, proper documentation is required for software to be useful. hictk documentation is hosted on read the docs at hictk.readthedocs.io. The documentation includes instructions of how to install hictk from bioconda and from source, as well as instructions of how to acquire hictk Docker images. Furthermore, the documentation comes with examples of how the CLI and API can be used to perform common operations. Documentation for hictkpy is also hosted on read the docs at hictkpy.readthedocs.io and has a similar structure to that of hictk's documentation. Considering that hictk is a young project, documentation is adequate. However, more work needs to be done to bring it on par with documentation of more mature projects such as the likes of cooler.

4.12 The importance of correctness

One aspect that is often neglected in bioinformatics software development, is automated testing. Most bioinformatics applications provide a test dataset that can be used to run the application with default settings. If the application does not crash and some output is produced, the application is assumed to be correct. This assumption could not be further from the truth, as any software, especially complex software, can fail or misbehave in a million different ways.

For this reason, while developing hictk I decided to take a more thorough approach to testing, one where the development of new code and its tests proceeded in parallel. When appropriate, I followed a test-driven development, whereby code for the test case was implemented before writing the functions or classes to be tested.

The latest version of hictk's codebase is tested with 80 unit tests and 22 integration tests, achieving over 83% code coverage. Automated tests are run every time a commit is pushed to hictk's repository (including commits pushed as part of pull requests). Furthermore, every week or whenever new commits are pushed to the main branch, a suite of randomized tests is executed. For this purpose, we generated a pair of identical .mcool and .hic files. These files are used as input for the randomized test suite, where thousands of random queries are generated and executed, and the output of hictk and cooler are compared to ensure both applications return identical interactions when given the same query. This level of testing contributes to making us confident that the output produced by hictk implementation is fundamentally correct, and that application and library code deal with invalid inputs in a reasonable way. Furthermore, because hictk is a C++ application, hictk can be affected by bugs related to memory management. To prevent these kinds of bugs, I avoid manually allocating memory whenever possible, and use instead facilities to manage memory allocations and access to uninitialized memory that are provided by the standard library, such as smart pointers, standard containers, `std::variant`, and `std::optional`. Furthermore, automated tests are run using hictk builds that are instrumented with Address Sanitizer (ASan) and Undefined Behavior Sanitizer (UBSan) to detect memory errors and undefined behavior at runtime. As hictk grew in complexity, its test suite grew as well.

hictk test suite has reached a point where running the automated tests on Ubuntu, testing 4 different compilers using a total of 6 different build configurations takes close to 2h. Automated tests are not only run on Ubuntu systems, but also on macOS Big Sur, Monterey, and Ventura, as well as Windows Server 2022. As hictk continues to grow, we will have to find ways to modularize hictk's test suite to support running portions of the full test suite. This will be necessary to improve the speed of automated tests to provide more rapid feedback to developers regarding the correctness of their changes.

4.13 Current limitations of hictk

hictk currently implements the vast majority of the functionality required to read any .cool and .hic file. However, there are a few edge cases that are not currently supported.

First, .hic files older than v6 are not supported, the reason being that we were unable to find files this old. Furthermore, there is no specification for format v6 and older. Support for format v6 is provided by mimicking how straw processes .hic v6 files. There are no current plans to support reading .hic files older than v6.

Second, both .hic files and cooler files support writing interactions using a variable bin size. However, these kinds of files are encountered very rarely, so supporting them was not one of our priorities. That being said, we do plan to support these kinds of files. This will require generalizing our concept of bin table to also support variable bin sizes.

Third, hictk does not natively support writing .hic files. Currently, functions and subcommands that need to write to .hic files depend on JuicerTools and HiCTools to do so. However, this is suboptimal, as both tools depend on the presence of the Java Virtual Machine (JVM), which is a large dependency, and something we would like to avoid. Furthermore, writing files in .hic format currently takes a long time and requires large amounts of memory. With hictk, I believe it will be possible to significantly reduce memory requirements while improving performance. This is a use case we definitely want to support in the near future.

Finally, hictk does not currently support reading remote files. Supporting this for .hic files is relatively easy, as in this case, we have full control over low-level IO operations. However, doing the same for cooler files is not trivial, as in this case, we have no control over low-level IO operations. Supporting remote file access for cooler files will likely require a significant amount of work, and may in the end not be possible at all. Thus, reading remote cooler files may require users to mount remote HTTP or FTP files and folders on the local filesystem using e.g., Filesystem in USErspace (FUSE).

4.14 hictk: future perspectives

I developed hictk with the aim of bridging the .hic and cooler worlds while shielding users from the details and complexity associated with the low-level operations on the two file formats.

hictk was not publicized to the scientific community up until very recently, and as such I believe there are currently no users outside of our group. We decided to not advertise hictk, as the project was made public towards the end of my PhD, as I was busy wrapping up Paper II and Paper III and would most likely not have time to fix bugs and provide adequate user support. However, this is soon going to change, and I hope the community will find hictk useful and embrace the new possibilities that tools like hictk open up.

I intend to continue developing and supporting hictk for as long as possible. Some of the features and improvements I intend to implement have been listed in Section 4.13, however, there are other improvements I would like to introduce, such as supporting balancing interactions using other algorithms beside Iterative Correction and Eigenvector decomposition (ICE), providing bindings for more scripting languages, such as R, and expanding the documentation.

Besides improvements to hictk itself, I am also looking forward to seeing the hictk library being used to write new applications or update old applications to support both .hic and cooler formats. As a matter of fact, I am planning to do some of this work myself by e.g., updating MoDLE to use hictk instead of coolerpp, as well as contributing to third-party projects to help them take advantage of hictk. One of such project is HiGlass, a web application to visualize Hi-C matrices and other types of biological data. The HiGlass backend is written in Python, and up until now only supported reading cooler files using the cooler Python API. Updating HiGlass to use hictk would enable HiGlass to visualize interactions from both .cool and .hic files. Given that cooler's and hictk's Python APIs are very similar, I expect this could be achieved with minimal changes to the HiGlass codebase.

Developing hictk required me to become knowledgeable about the .hic and cooler format specifications. I believe this puts me in a position to provide useful insights when it comes to the development and improvement of file formats to represent genomic interactions such as those produced by Hi-C and Pore-C experiments. While the current version of .hic and cooler enable efficient storage of binned pair-wise interactions, new storage strategies will have to be developed to support multiway interactions such as those produced by Pore-C experiments. Furthermore, there are currently no formats designed to efficiently represent and query pair-wise or multiway interactions directly. Pairix (reviewed in Section 3.3.3.2) is a step in the right direction, however, the format itself operates on compressed text files, where a compressed binary representation would certainly improve file size and query performance. After my PhD, it is my intention to continue being involved in file format standardization and library development.

During my PhD, most of my development efforts were done on my own without collaborating with other developers. However, going forward I would like to increase my collaborations with other developers to improve the quality of the software I develop and receive feedback from other experts in the field. This will likely mean collaborating with groups such as the Open2C and the 4DNucleome.

4.15 Why analyzing cancer data?

Breast cancer is the most common type of cancer and is characterized by complex disorganization of the 3D genome structure. Thus, understanding the reasons behind the 3D structure disorganization is crucial to better understanding how breast cancer and other types of cancer develop and progress. While the disorganization of the 3D structure of genomes in breast cancer has been previously described [235, 236], there are currently no studies characterizing this dysregulation during cancer progression. For this reason, we decided to write Paper III, where we characterize the genome 3D structure in an *in vitro* model of breast cancer. Analyzing human cancer data provided me with a new set of challenges and skills to learn, as well as the opportunity to contribute knowledge to the scientific community that one day may inform the development of new therapies to treat cancer.

First, our cancer model is affected by structural variations. This poses a number of challenges when analyzing Hi-C data, given that data is mapped to reference genomes without structural variations, meaning that interactions that in reality are occurring in cis, appear in the trans portion of Hi-C maps.

Second, the project involved analyzing other kinds of data, including epigenetic data and expression data.

Third, while workflows for the analysis of ChIP-seq and RNA-seq data are fairly standardized, the same cannot be said for Hi-C data analysis. This meant I had to develop my own data analysis workflow as part of this project. Finally, analyzing data in a reproducible way involves an additional set of challenges, as reproducibility must be baked in when designing the analysis workflows.

4.16 Global deterioration of the 3D genome during cancer progression

While perturbations at the TADs have been shown to be the cause of developmental diseases as well as being associated with some types of cancer and degenerative diseases, in Paper III we report that TADs do not undergo significant rearrangements during cancer progression in our breast cancer model. In contrast, we report that TAD cliques undergo significant reconfiguration as cancer progresses. This is quite surprising, as it suggests that the properties of TADs themselves do not change significantly, while the way they interact with each other does. This can be interpreted in two opposite ways: (1) the structural and regulatory role of TADs is so important that significant perturbations involving TADs are not observed because they are lethal, and (2) TADs have a marginal role when it comes to gene expression orchestration and thus their perturbation often does not lead to changes in expression levels. A third hypothesis is that both hypotheses 1 and 2 are true at the same time for different parts of the genome. Experiments depleting factors involved in loop extrusion, such as cohesin and CTCF, suggest that the second hypothesis is true. However, these experiments deplete proteins for a relatively short amount of time, thus it is possible that longer depletion times are required in order to observe a more significant change in gene expression. The high level of conservation of TADs across cell types and, to some extent, different species, supports the first hypothesis. However, the high variability on TAD-TAD interactions suggests that this type of interaction is less fundamental for cell viability, however, due to their pervasiveness in our cancer model it is possible that they play a role in the progression of cancer.

One of the main findings of Paper III is that subcompartments undergo significant rearrangements during cancer progression, with different subcompartments showing distinct switching behavior. Perhaps as expected, intermediate subcompartments such as B0 and A0 show the greatest tendency to switch to any other subcompartment type. Surprisingly, there are relatively few A0→B0 and B0→A0 transitions, suggesting that B0 and A0 subcompartments may characterize regions where chromatin is in an intermediate state, waiting to be remodeled into eu- or heterochromatin. This hypothesis is substantiated by the lack of both enrichment and depletion of active and repressive histone marks in B0 and A0 subcompartments.

Subcompartment switching in T1 and C1 are also associated with the deterioration of chromatin radial positioning. This may suggest that subcompartment transitions involving B3 and B2 subcompartments are either cause or consequence of the deterioration of chromatin radial positioning.

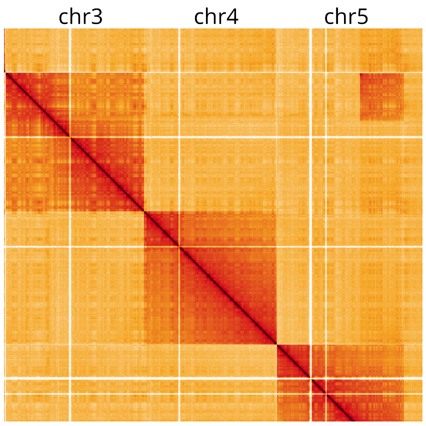
Ultimately, more studies will be needed to further clarify this relationship. Regardless of their cause, we report that compartment switches affect gene expression in the expected direction, that is switches toward open chromatin lead to the upregulation of gene expression, while switches towards closed chromatin lead to gene expression downregulation. However, the correlation between compartment switches and changes in gene expression is fairly mild, suggesting that there is not a 1:1 correspondence between the two variables and that only a subset of differentially expressed genes can be explained by subcompartment switches.

In conclusion, at the global level, we report a general deterioration of the 3D genome organization. This deterioration is especially pronounced at the subcompartment and TAD clique level.

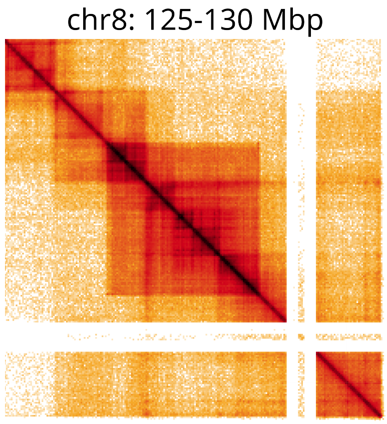
4.17 Structural variation at the *MYC* locus is associated with upregulation of known oncogenes

After characterizing the deterioration of the 3D genome organization at a global level, we decided to focus on the *MYC* locus, which in healthy cells is located on chromosome 8. As outlined in Part III, this locus is involved in a structural rearrangement consisting of a translocation and amplification. Unfortunately, due to the complexity of the structural rearrangement, we were unable to fully resolve the topology of the translocated region. However, we were able to show that the translocation results in two *de novo* contacts between enhancer elements on chromosome 10 and *MYC*'s promoter. The large copy number of the translocated region opens the possibility that the translocated region exists as a circular, extra chromosomal DNA (ecDNA), which is not uncommon for the *MYC* region in cancer cell lines. This hypothesis could be verified e.g., using FISH by using probes that hybridize to chromosome 8, chromosome 10, and the translocated regions. If the translocated region exists as an ecDNA, then we should observe at least three bright dots in the FISH experiment.

4.18 Structural variants complicate the analysis of Hi-C data



(a) Translocation event visualized through Hi-C heatmaps. The translocated region appears as a darker square in the chr3:chr5 region of the interaction matrix.



(b) Amplification event visualized through Hi-C heatmaps. The amplified region appears as a sharp, dark square of enriched interactions.

As briefly mentioned in Section 4.15, the data analysis for Paper III was complicated by the presence of several translocations and copy number alterations. Figure 4.6a shows an example of how translocations appear in Hi-C data. This poses several issues, as interactions that are occurring in cis appear as taking place in trans in the Hi-C data. This can easily interfere with any statistical tests to identify significant interactions in Hi-C matrix, like the one used to call TAD cliques in Paper III. Ultimately, the proper way to deal with translocations in Hi-C data would be to generate a new reference genome assembly that includes translocations and map the Hi-C reads back to this new reference. Unfortunately, this is highly impractical, as generating the new assembly will likely require Whole Genome Sequencing (WGS). This could in principle be avoided by splitting the original assembly at translocation breakpoints, then re-scaffold the assembly. However, there are currently no tools that can automate this process. Furthermore, re-scaffolding may in the end fail due to the lack of phasing information. Finally, this would require a system, such as liftOver, to map between coordinates of different reference genomes, further complicating the analysis. Some of this complexity could be reduced by using a graph representation of the genome assembly. However, tools to operate on genome graphs are still in the early stages. Furthermore, the pipelines used to generate Hi-C maps would have to be updated to work with this kind of genome representation.

In conclusion, the field currently lacks tools to effectively deal with translocations in Hi-C data.

Another issue is that of Copy Number Variations (CNVs), whereby one or more regions are present in more or less copies than what is found in the wild-type genotype. In the context of Hi-C data, amplified regions will appear to interact more strongly (see Figure 4.6b), as reads originated from different copies of the same regions will be mapped back to the same region in the genome assembly. Generating a custom genome assembly with the copy-altered regions will not help in this case, as this would result in multi-mapping reads that cannot be uniquely assigned to a genomic region. The common approach to deal with CNVs in Hi-C data is to balance interaction matrices using algorithms such as ICE, which assume equal visibility across genomic regions. However, this approach has been shown to not be effective, as it leads to an imbalance between cis and trans interactions for CNV regions [237]. To address these issues, specialized balancing algorithms have been developed [237]. Unfortunately, available implementations are not optimized to support processing high-resolution matrices, which forced us to use ICE to balance interaction matrices used in Paper III. However, given that most of our analyses are performed on cis matrices, we used cis interactions only to balance interaction matrices. Furthermore, calling TAD cliques is the only step that requires trans interactions. However, in this step, raw cis and trans interaction frequencies are used. Given the above, our analyses are likely not affected by the cis/trans interaction imbalance outlined earlier.

4.19 Closing remarks

The 3D genome is neatly folded into a multi-level 3D structure which includes A/B compartments, TADs, and nucleosomes.

In the first part of my PhD (Paper I), I focused my research activities at the TAD level by developing a stochastic model of DNA loop extrusion. TADs were initially thought to regulate the formation of enhancer-promoter interactions by promoting certain interactions while preventing others. Accordingly, disruption of TADs have been shown to cause a number of developmental diseases [238, 239], neurological disorders [240], and cancers [241, 242].

Given the prevalence of TADs in metazoan genomes, and how they appear to control enhancer-promoter interactions, one would think that their disruption by e.g., cohesin or CTCF depletion would have dramatic consequences on gene expression. Yet, several studies report that the expression of only a small number of genes is affected by the above perturbations [29, 45, 125, 134, 161, 243]. These observations support the hypothesis that TADs are important for the correct wiring of enhancer-promoter interactions at mitotic exit and that they have a less important role during interphase. This raises several important questions regarding the association of TAD perturbations and cancer progression. Indeed, several papers report that genes located inside perturbed TADs exhibit aberrant expression [244–247]. For example, Luo et al. [246] reported that dysregulation of the *HOX* gene, which is associated with many types of cancer including

Acute Myeloid Leukemia (AML), is driven by the presence of a CTCF boundary between the *HOXA7* and *HOXA9* genes. Taberlay et al. [245] characterized the effects of TAD perturbations in prostate cancer and found a number of cancer-specific TADs. These TADs often overlapped regions affected by CNVs. For example, a deletion of a region spanning the *TP53* tumor suppressor locus resulted in the splitting of a TAD into two smaller TADs with corresponding alteration in gene expression. However, in the cancer progression model used in Paper III, TADs were highly conserved and thus do not appear to be the main driver of cancer progression. This is also one of the reasons why we decided not to take advantage of MoDLE simulations as part of Paper III.

Besides TADs, A/B compartments and subcompartments have been reported to often undergo major changes in many cancer types [248, 249], and Paper III. For example, Martin et al. [248] characterized A/B compartments during prostate cancer progression and showed that cancer progression is associated with a loss of A compartment strength, suggesting an increased intermixing of A and B compartment regions. Overall, the authors report that 300 and 86 genes overlap regions switching from B to A and from A to B compartments, respectively. In Paper III we also report substantial switching between A and B compartments. However, the overall ratio of A to B compartments remains largely unaltered as breast cancer progresses. Furthermore, the use of subcompartments allowed us to assess switching with a greater level of details.

Recent evidence shows that chromatin alternates between A and B compartments at the kbp scale, and that virtually all active enhancers are located in A compartments. While there is not a 1:1 mapping between A compartments and active genes, there is a clear correlation between expression levels and A compartments (see Paper III). The evidence produced by Harris et al. [66] suggests that very small chromatin openings may be sufficient to induce enhancer activation and that a small fraction of the compartment switches reported in Paper III may be important for cancer development and progression.

Ultimately, more studies are needed to elucidate this question.

Chapter 5

Other projects

Besides working on my main project, throughout my PhD I have been working on a few side projects as well as contributing to other Open-Source Software (OSS) projects.

5.1 Side projects

Nextflow workflow to call TAD cliques

In late 2022 I started working on a Nextflow workflow to call TAD cliques. The workflow is available on GitHub at robomics/call_tad_cliques. TAD cliques are called as described in the following tutorial: Chrom3D/INC-tutorial.

The workflow takes as input a sample sheet in TSV format with the following mandatory columns:

- **sample**: sample name
- **cooler_cis**: cooler file used to read cis interactions (e.g., Uniform Resource Identifier (URI) to a 50 kbp cooler)
- **cooler_trans**: cooler file used to read trans interactions (e.g., URI to a 1 Mbp cooler)

The sample sheet can contain a **tads** column containing the path to a BED file with the list of TADs. When not provided, the workflow will use HiCExplorer to call TADs from the cooler_cis file.

The workflow outputs 6 text files and 2 plots. The text files contain the list of TAD cliques called from cis interactions, trans interactions, and all interactions. For each type of clique, the workflow outputs a file with the list of TAD cliques and another file with the list of the domains part of cliques. This workflow was used in Paper III to automate TAD clique calling.

Nextflow workflow to compress the output of nf-core/hic

In May 2023 I developed a Nextflow workflow to compress the output produced by nf-core/hic. The workflow is available on GitHub at robomics/compress-nfcore-hic-output.

The workflow takes as input the path to the output folder produced by nf-core/hic, the path to a FASTA file with the reference genome assembly used by nf-core/hic, and the path to a folder where to store the output files. The workflow will compress files in BAM format using CRAM, a reference-based format to store reads mapped to a reference genome. Furthermore, the workflow will compress statistic files, validPairs (see Section 3.3.1), and MultiQC report using LZMA (i.e., xz). This workflow was used to compress the data produced by Paper III, achieving a 3.1x compression ratio for alignment data and a 4x compression ratio for validPairs.

Nextflow workflow to generate the gene track for HiGlass

Generating gene annotations with HiGlass can be a difficult process, as HiGlass does not natively support GTF or GFF files. Instead, users are first required to generate a .beddb file using clodius, then ingest this file in a HiGlass server using a special set of command line flags. The instructions on how this can be achieved are a bit involved and are listed in HiGlass documentation (link).

In June 2022, I developed a Nextflow workflow to automate the generation of the .beddb file. The workflow requires as input a few files with gene information that can be downloaded from NCBI (e.g., a file to map gene IDs to RefSeq IDs), as well as the taxa ID and assembly name for the species that is being processed. Furthermore, the workflow requires a file with the list of chromosomes and their sizes as well as the refgene file that can be downloaded from UCSC FTP server.

The workflow outputs a single file in .beddb format that can be readily ingested into HiGlass server instance.

5.2 OSS contributions

Bioconda

I contributed a handful of recipes to the bioconda, including recipes for the following packages:

Package	Source	Recipe	Contribution
FitHiC	github.com	anaconda.org	Update recipe
hictk	github.com	anaconda.org	Add new recipe
hictkpy	github.com	anaconda.org	Add new recipe
modle	github.com	anaconda.org	Add new recipe

Conan Center Index

I contributed several recipes to the Conan Center Index, including recipes for the following packages:

Package	Source	Recipe	Contribution
bitflags	github.com	conan.io	Add new recipe
boost	boost.org	conan.io	Update recipe
bshoshany-thread-pool	github.com	conan.io	Update recipe
cli11	github.com	conan.io	Update recipe
fast_float	github.com	conan.io	Update recipe
hdf5	portal.hdfgroup.org	conan.io	Update recipe
hictk	github.com	conan.io	Add new recipe
highfive	github.com	conan.io	Update recipe
libarchive	libarchive.org	conan.io	Update recipe
libbigwig	github.com	conan.io	Add new recipe
libcuckoo	github.com	conan.io	Add new recipe
spdlog	github.com	conan.io	Update recipe
xoshiro-cpp	github.com	conan.io	Add new recipe
xxhash	github.com	conan.io	Update recipe
xz_utils	tukaani.org	conan.io	Update recipe

Other contributions

Beside contributions to bioconda and the Conan Center Index I have also contributed bugfixes to the following projects:

Project	Source	Contributions
Chrom3D	github.com	github.com
cooler	github.com	github.com
dcHiC	github.com	github.com
iced	github.com	github.com
libBigWig	github.com	github.com
nf-core/hic	github.com	github.com
stripenn	github.com	github.com

5.3 Bug report for 4DNucleome

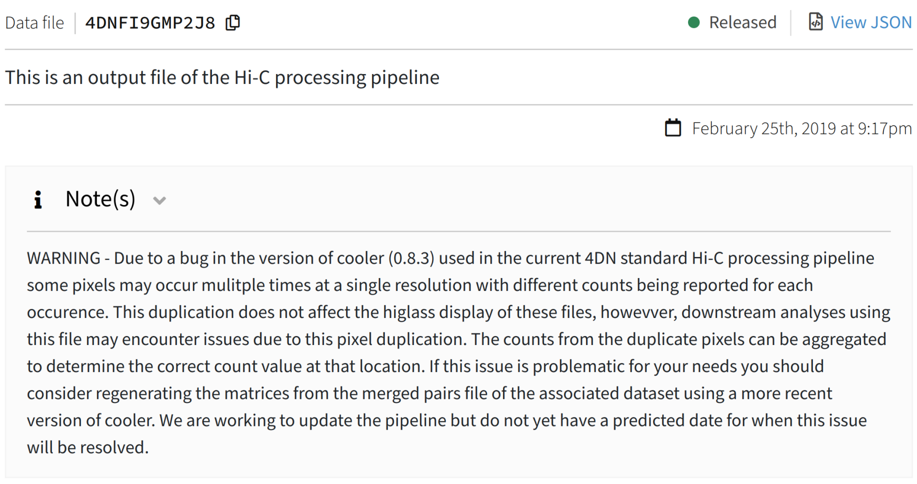
While working on Paper I, I discovered a bug affecting .mcool files generated by cooler v0.8.3, which includes all .mcool files published on the 4DNucleome Data Portal up until the time of writing (19/11/2023). When reading interactions from the affected files, there is a possibility that multiple values will be returned for the same pixel, meaning that for certain rows or columns, the Hi-C matrix appears to be N-dimensional where $N \geq 2$.

For example, the following output was obtained using the Cooler API to read interactions from 4DNFI9GMP2J8.mcool

chrom1	start1	end1	chrom2	start2	end2	count	balanced
chr1	10828000	10830000	chr1	11002000	11004000	1	0.000208987
chr1	10828000	10830000	chr1	11002000	11004000	1	0.000208987
chr1	10828000	10830000	chr1	11006000	11008000	1	0.000199523
chr1	10828000	10830000	chr1	11006000	11008000	3	0.000598569
chr1	10828000	10830000	chr1	11010000	11012000	4	0.000695946
chr1	10828000	10830000	chr1	11010000	11012000	2	0.000347973
chr1	10828000	10830000	chr1	11020000	11022000	1	0.000219669
chr1	10828000	10830000	chr1	11020000	11022000	1	0.000219669
chr1	10828000	10830000	chr1	11030000	11032000	3	0.000499071
chr1	10828000	10830000	chr1	11030000	11032000	2	0.000332714
...	...	...	...	...	...	...	...

Luckily, the base resolution of files affected by this bug does not have duplicate pixels, suggesting that the bug was introduced when running the `cooler zoomify` command. Thus, affected files can be fixed by iteratively coarsening the base resolution with `cooler zoomify` from a recent version of cooler, or using `hictk zoomify`.

I reported the bug to the 4DNucleome, which resulted in them adding the notice shown in Figure 5.1 to the file page for .mcool files:


Data file | 4DNFI9GMP2J8

● Released | View JSON

This is an output file of the Hi-C processing pipeline

February 25th, 2019 at 9:17pm

Note(s)

WARNING - Due to a bug in the version of cooler (0.8.3) used in the current 4DN standard Hi-C processing pipeline some pixels may occur mulitple times at a single resolution with different counts being reported for each occurence. This duplication does not affect the higlass display of these files, howevver, downstream analyses using this file may encounter issues due to this pixel duplication. The counts from the duplicate pixels can be aggregated to determine the correct count value at that location. If this issue is problematic for your needs you should consider regenerating the matrices from the merged pairs file of the associated dataset using a more recent version of cooler. We are working to update the pipeline but do not yet have a predicted date for when this issue will be resolved.

Figure 5.1: Warning notice on the 4DNucleome data portal informing users of the bug affecting .mcool files published on the portal.

A detailed report of this bug is available on GitHub at robomics/20221129_4dnucleome_bug_report.

While developing hictk, I have implemented two subcommands to detect and fix this issue, namely, `hictk validate` and `hictk fix-mcool` (see Paper II). The first detects various problems with .hic, .cool, .mcool and .scool files. When passing the `--validate-index` option, hictk will check the index of cooler files to detect the bug described above. `hictk fix-mcool` is a wrapper around `hictk zoomify` and `hictk balance`. This command will first read the resolutions available in the corrupted .mcool file, and generate a new .mcool file containing the same resolutions. After iterative coarsening, hictk will proceed to balance each resolution using ICE. If the original matrix was balanced using cooler, then hictk will attempt to re-use the same balancing parameters used by cooler, otherwise, it will fall back to use the default balancing parameters used by `hictk balance`.

Chapter 6

Research reproducibility

For the past decade, we have been aware of the reproducibility crisis that is affecting modern scientific studies, meaning that the results of many scientific studies cannot be reproduced by an independent party [250–252]. This crisis affects virtually every field, including the fields of genomics and bioinformatics. Conceptualizing and performing studies with a good degree of reproducibility is no easy task, and often requires an interdisciplinary team committed to reproducibility. Discussing reproducibility in a general sense is not in the scope of this thesis, where instead I will focus on the reproducibility of bioinformatics data analysis and the measures I developed to ensure the results produced by my analysis pipeline are replicable and reproducible by others.

6.1 Why is reproducibility important?

There is no unique definition of reproducibility in the context of the scientific method. Odd Erik Gundersen proposes that “Reproducibility is the ability of independent investigators to draw the same conclusions from an experiment by following the documentation shared by the original investigators” [253]. This is extremely important for scientific studies, to the point that Moonesinghe et al. state that “replication ... is the cornerstone of science and replication of findings is very important before any causal inference can be drawn” [254].

In the field of life sciences, reproducibility is a glaring issue, with a survey finding that “More than 70% of researchers have tried and failed to reproduce another scientist’s experiments, and more than half have failed to reproduce their own experiments” [250]. This has several ramifications, ranging from slower scientific progress, wasted resources, and even negative effects on patient’s health. A study from 2015 estimated that over $28 billion per year are spent on preclinical research that is not reproducible in the United States alone [255]. Thus, as a scientist, I feel the responsibility to do my best to ensure the results of my research activities are reproducible by others.

6.2 Reproducibility in the context of bioinformatics

In the context of bioinformatics data analysis, for a study to be reproducible, the raw and processed data, as well as all the code used to generate processed data from raw data should be shared. If any analysis step requires manual intervention, these steps should be thoroughly documented to enable a third party to reproduce the manual steps. Thus, when it comes to bioinformatics analysis, documentation consists of source code and documentation for the operator running the analysis, with instructions on how to prepare the data and run the analysis steps.

While there is no unique definition of reproducibility and replicability, throughout this section I will use the definition of reproducibility suggested by Odd Erik Gundersen, and I am defining replicability as the ability of independent investigators to obtain identical results when applying the original protocol on the original data following the provided documentation. So, study replicability is a more stringent requirement than study reproducibility.

In the vast majority of genomic studies, raw data is generated through complex wet lab experiments. This raw data is then analyzed by one or more bioinformatic pipelines. While achieving end-to-end replicability of a study is usually not possible, as we do not control all experimental variables, reproducibility of the computational analysis is often achievable, provided that analysis pipelines have been developed with this objective in mind. There are several tools that enable the development of reproducible analysis pipelines at scale, some of which will be discussed in the next sections.

6.3 My workflow for reproducible data analysis

During the course of my PhD, I became familiar with several tools to improve the reproducibility of data analysis pipelines (see Appendix C) and have developed my own data analysis workflow using these tools. This section briefly outlines and motivates my analysis workflow.

For each project, I create a new GitHub repository. This repository will host all code used for data analysis. The repository is structured using 5 different folders:

- `bin/`: folder containing all scripts used by task in the workflow management system, Nextflow in this case.
- `configs/`: folder containing workflow config files.
- `containers/`: folder containing the Dockerfiles used to build the container images used by the workflows.
- `data/`: folder containing input, output, and scratch data.
- `workflows/`: folder containing the Nextflow workflow files.

Each workflow has its own config. Workflows and configs have the same name, as the file name is used to pair configs and workflows. For example, workflow `fetch_data.nf` would be paired with config `fetch_data.config`. Config files define the input files and parameters used by workflows.

Dockerfiles under `containes/` are named like `image_name__vx.y.z.Dockerfile`. The double underscore is used by the build infrastructure to split the image name from its version. Ideally, there should be one image for each application used in the workflows, however, this is practice not always possible, as certain workflow tasks need access to two or more applications. When this is the case, images are given a generic name (e.g., utils or functional_analysis), and all required tools are installed side by side. When a Dockerfile is modified, a GitHub Actions workflow is triggered. This workflow will first gather metadata about the container image to be built, then will proceed to build the container image using resources from GitHub's cloud infrastructure. The resulting container images are published in the package section of the repository that triggered the workflow. The GitHub Actions workflow is defined using reusable actions. Thus, adding a new Dockerfile requires adding a minimal GitHub workflow for the image to be picked up and built by the build infrastructure.

The number and type of Nextflow workflows defined in the `workflows/` folder depends on the analyses to be carried out. However, there are some workflows that are fairly common. For example, virtually all my projects have a `fetch_data.nf` workflow. This workflow takes a `downloads.tsv` file as input with the following columns:

- url: the URL to the file to be downloaded.
- sha256: the SHA256 digest of the file to be downloaded.
- dest: the path (including the file name) where the file should be stored after being downloaded.
- action: when set to none, files are simply downloaded. When set to compress, files are compressed using gzip before being stored at their final destination.

The workflow consists of a single task that processes a single row from the `downloads.tsv` file, thus enabling concurrent download, checksumming, renaming, and optionally compressing of all the input files required by other workflows. Another common workflow is `preprocess_data.nf`. This workflow contains tasks to preprocess input data and generate data used by other downstream workflows. Examples of common tasks performed by this workflow are sorting .chrom.sizes files, filtering reference genome assemblies to remove unplaced scaffolds, and building indices used by read mappers. In general, I try to define one workflow for each data analysis unit. For example, for the data analysis for Paper III, TAD analysis and compartment analysis are performed by two separate workflows. Crucially, draft image generation is usually done as part of the analysis workflows. This ensures that the latest figures represent the latest results produced by the workflow.

Finally, launching workflows is automated through bash scripts. Thus, reproducing the results of my data analysis requires the following steps:

- Installing and configuring Nextflow.
- Cloning the data analysis repository.
- Read the documentation to learn in which order workflows should be launched.

- Launch workflows in the proper order. As an example, launching workflow `fetch_data.nf` is as simple as running `./run_fetch_data.sh`.

The workflows will take care to download the appropriate Docker images and use them to run tasks using the input files and parameters defined in config files. This will result in fully reproducible data analysis, as input files are checksummed immediately after being downloaded and tasks are run using containers.

6.4 Sharing data analysis pipelines

Openly sharing workflows for reproducible data analysis is equally important to writing the workflows in the first place. The development of my pipelines can be tracked on GitHub (e.g., the pipeline used for Paper III is available at paulsengroup/2022-mcf10a-cancer-progression). Furthermore, pipeline releases are archived on Zenodo using Zenodo's GitHub integration. As an example, the pipeline used for Paper III is archived at 10.5281/zenodo.10069548. The pipeline README contains instructions on the pipeline structure and how it can be executed.

6.5 Data sharing

As the data produced by genomics studies tend to be very large, special care should be taken when publishing this kind of data. Furthermore, the data should be publicly available and discoverable for a long period of time (as in decades), and thus services such as Dropbox and Google Drive are not suitable for this purpose. The details of how data should be archived and published depend on the nature of the study as well as data type. To publish the data produced by my research activities I used a variety of services, namely the NRID Research Data Archive and Zenodo to archive general-purpose data (Paper I and Paper II), and the Sequence Read Archive (SRA) and Gene Expression Omnibus (GEO) to archive biological data (Paper II and Paper III). These services provide long-term storage for research and other types of data. Data uploaded to these services cannot be modified. Furthermore, each dataset is assigned a unique identifier (often in the form of a DOI string). Using an archival strategy like the one just described is very important, as it ensures that the data produced by scientific studies is discoverable and reusable by other researchers.

6.6 Research reproducibility: closing remarks

In Section 6.1 I briefly outlined some reasons why reproducibility of data analysis is an important matter.

In later sections, I presented some technical solutions that can be used to achieve data analysis reproducibility. These solutions provide several benefits that nicely complement one another, however, they all have one glaring problem: they address complexity with more complexity. While it can be argued that the tradeoff in complexity is one worth making, the burden of this additional complexity falls on the shoulders of the researchers in charge of the data analysis. In practice, this means that researchers have to learn the basics of containerized applications, as well as a new domain-specific language for workflow management. With the right set of incentives, this would not be a problem, however, at the time of writing, there are relatively few incentives for researchers to do the above. However, the field is slowly moving in the right direction. For example, several journals, such as Genome Biology [256] and Bioinformatics [257] not only require that data used as part of scientific publication be made public through appropriate data repository but also that the same is done with code, albeit with almost no requirements placed on software quality and reproducibility, as depositing code in a GitHub repository is usually sufficient to fulfill the requirements.

Given the huge economic and time loss caused by non-reproducible research, I think there should be stronger incentives to adopt some of these technologies. For example, scientific journals could start requiring that software used as part of the data analysis be made available as software containers. Funding agencies could also play a role in incentivizing the adoption of technologies to enable reproducible data analysis. Some funding agencies, such as the European Research Council (ERC) already have strong requirements when it comes to open science and Findability, Accessibility, Interoperability, and Reusability (FAIR) principles, however, these requirements focus on research and data discoverability, and availability, not on accessibility and quality of the software used to process the data itself. I believe there should be a set of principles in line with FAIR also for scientific software and analysis pipelines. Another entity that could facilitate the adoption of these tools is for universities to introduce mandatory courses at the PhD level to educate new PhD candidates on the causes of the reproducibility crisis, as well as the technical solutions to tackle these problems. Many universities, including Universitetet i Oslo (UiO), already have mandatory courses covering these topics, however, in many cases these courses cover the topics at a fairly high level, without providing clear actionable that a PhD candidate can immediately apply to their day-to-day research. I think these courses should be updated and expanded to also cover how to practically tackle ethical issues, including reproducibility issues, in practice.

Bibliography

1. Jordan Rowley M and Corces VG. Organizational Principles of 3D Genome Architecture. Nature reviews Genetics. 2018 Dec; vol. 19, no. 12 :789–800
2. Steensel B van. Chromatin: constructing the big picture. The EMBO Journal. 2011 May 18; vol. 30, no. 10 :1885–95
3. Luger K, Dechassa ML, and Tremethick DJ. New insights into nucleosome and chromatin structure: an ordered state or a disordered affair? Nature Reviews Molecular Cell Biology. 2012 Jul; vol. 13, no. 7. Number: 7 Publisher: Nature Publishing Group:436–47
4. Fyodorov DV, Zhou BR, Skoultchi AI, and Bai Y. Emerging roles of linker histones in regulating chromatin structure and function. Nature Reviews Molecular Cell Biology. 2018 Mar; vol. 19, no. 3. Number: 3 Publisher: Nature Publishing Group:192–206
5. Henikoff S, Furuyama T, and Ahmad K. Histone variants, nucleosome assembly and epigenetic inheritance. Trends in Genetics. 2004 Jul 1; vol. 20, no. 7. Publisher: Elsevier:320–6
6. Chadwick BP and Willard HF. Cell cycle–dependent localization of macroH2A in chromatin of the inactive X chromosome. Journal of Cell Biology. 2002 Jun 24; vol. 157, no. 7 :1113–23
7. Jenuwein T and Allis CD. Translating the Histone Code. Science. 2001 Aug 10; vol. 293, no. 5532. Publisher: American Association for the Advancement of Science:1074–80
8. Bannister AJ and Kouzarides T. Regulation of chromatin by histone modifications. Cell Research. 2011 Mar; vol. 21, no. 3. Number: 3 Publisher: Nature Publishing Group:381–95
9. Millán-Zambrano G, Burton A, Bannister AJ, and Schneider R. Histone post-translational modifications — cause and consequence of genome function. Nature Reviews Genetics. 2022 Sep; vol. 23, no. 9. Number: 9 Publisher: Nature Publishing Group:563–80
10. McBryant SJ, Adams VH, and Hansen JC. Chromatin architectural proteins. Chromosome Research. 2006 Feb 1; vol. 14, no. 1 :39–51
11. Hammond CM, Strømme CB, Huang H, Patel DJ, and Groth A. Histone chaperone networks shaping chromatin function. Nature Reviews Molecular Cell Biology. 2017 Mar; vol. 18, no. 3 :141–58
12. Reyes AA, Marcum RD, and He Y. Structure and Function of ATP-dependent Chromatin Remodeling Complexes. Journal of molecular biology. 2021 Jul 9; vol. 433, no. 14 :166929

13. Golbabapour S, Majid NA, Hassandarvish P, Hajrezaie M, Abdulla MA, and Hadi AHA. Gene Silencing and Polycomb Group Proteins: An Overview of their Structure, Mechanisms and Phylogenetics. OMICS : a Journal of Integrative Biology. 2013 Jun; vol. 17, no. 6 :283–96

14. Finch JT and Klug A. Solenoidal model for superstructure in chromatin. Proceedings of the National Academy of Sciences of the United States of America. 1976 Jun; vol. 73, no. 6 :1897–901

15. Woodcock CL, Frado LL, and Rattner JB. The higher-order structure of chromatin: evidence for a helical ribbon arrangement. The Journal of Cell Biology. 1984 Jul; vol. 99, no. 1 :42–52

16. Robinson PJJ, Fairall L, Huynh VAT, and Rhodes D. EM measurements define the dimensions of the "30-nm" chromatin fiber: evidence for a compact, interdigitated structure. Proceedings of the National Academy of Sciences of the United States of America. 2006 Apr 25; vol. 103, no. 17 :6506–11

17. Robinson PJ and Rhodes D. Structure of the ‘30nm’ chromatin fibre: A key role for the linker histone. Current Opinion in Structural Biology. Nucleic acids/Sequences and topology 2006 Jun 1; vol. 16, no. 3 :336–43

18. Bassett A, Cooper S, Wu C, and Travers A. The folding and unfolding of eukaryotic chromatin. Current Opinion in Genetics & Development. 2009 Apr; vol. 19, no. 2 :159–65

19. Eltsov M, MacLellan KM, Maeshima K, Frangakis AS, and Dubochet J. Analysis of cryo-electron microscopy images does not support the existence of 30-nm chromatin fibers in mitotic chromosomes in situ. Proceedings of the National Academy of Sciences. 2008 Dec 16; vol. 105, no. 50. Publisher: Proceedings of the National Academy of Sciences:19732–7

20. Fussner E, Strauss M, Djuric U, Li R, Ahmed K, Hart M, Ellis J, and Bazett-Jones DP. Open and closed domains in the mouse genome are configured as 10-nm chromatin fibres. EMBO reports. 2012 Nov 6; vol. 13, no. 11 :992–6

21. Joti Y, Hikima T, Nishino Y, Kamada F, Hihara S, Takata H, Ishikawa T, and Maeshima K. Chromosomes without a 30-nm chromatin fiber. Nucleus (Austin, Tex). 2012; vol. 3, no. 5 :404–10

22. Nishino Y, Eltsov M, Joti Y, Ito K, Takata H, Takahashi Y, Hihara S, Frangakis AS, Imamoto N, Ishikawa T, and Maeshima K. Human mitotic chromosomes consist predominantly of irregularly folded nucleosome fibres without a 30-nm chromatin structure. The EMBO journal. 2012 Apr 4; vol. 31, no. 7 :1644–53

23. Maeshima K, Hihara S, and Eltsov M. Chromatin structure: does the 30-nm fibre exist in vivo? Current Opinion in Cell Biology. Nucleus and gene expression 2010 Jun 1; vol. 22, no. 3 :291–7

24. Ou HD, Phan S, Deerinck TJ, Thor A, Ellisman MH, and O'Shea CC. ChromEMT: Visualizing 3D chromatin structure and compaction in interphase and mitotic cells. Science. 2017 Jul 28; vol. 357, no. 6349. Publisher: American Association for the Advancement of Science:eaag0025

25. Chen B, Gilbert LA, Cimini BA, Schnitzbauer J, Zhang W, Li GW, Park J, Blackburn EH, Weissman JS, Qi LS, and Huang B. Dynamic Imaging of Genomic Loci in Living Human Cells by an Optimized CRISPR/Cas System. Cell. 2013 Dec 19; vol. 155, no. 7. Publisher: Elsevier:1479–91

26. Nagano T, Lubling Y, Stevens TJ, Schoenfelder S, Yaffe E, Dean W, Laue ED, Tanay A, and Fraser P. Single-cell Hi-C reveals cell-to-cell variability in chromosome structure. Nature. 2013 Oct; vol. 502, no. 7469. Number: 7469 Publisher: Nature Publishing Group:59–64

27. Ricci MA, Manzo C, García-Parajo MF, Lakadamyali M, and Cosma MP. Chromatin Fibers Are Formed by Heterogeneous Groups of Nucleosomes In Vivo. Cell. 2015 Mar 12; vol. 160, no. 6. Publisher: Elsevier:1145–58

28. Krietenstein N, Abraham S, Venev SV, Abdennur N, Gibcus J, Hsieh THS, Parsi KM, Yang L, Maehr R, Mirny LA, Dekker J, and Rando OJ. Ultrastructural Details of Mammalian Chromosome Architecture. Molecular Cell. 2020 May 7; vol. 78, no. 3 :554–565.e7

29. Szabo Q, Donjon A, Jerković I, Papadopoulos GL, Cheutin T, Bonev B, Nora EP, Bruneau BG, Bantignies F, and Cavalli G. Regulation of single-cell genome organization into TADs and chromatin nanodomains. Nature Genetics. 2020 Nov; vol. 52, no. 11. Number: 11 Publisher: Nature Publishing Group:1151–7

30. 4DNFI9GMP2J8; 4DN Data Portal — identifiers.org

31. Rocha PP, Raviram R, Bonneau R, and Skok JA. Breaking TADs: insights into hierarchical genome organization. Epigenomics. 2015; vol. 7, no. 4 :523–6

32. Dixon JR, Selvaraj S, Yue F, Kim A, Li Y, Shen Y, Hu M, Liu JS, and Ren B. Topological domains in mammalian genomes identified by analysis of chromatin interactions. Nature. 2012 May; vol. 485, no. 7398. Number: 7398 Publisher: Nature Publishing Group:376–80

33. Nora EP, Lajoie BR, Schulz EG, Giorgetti L, Okamoto I, Servant N, Piolot T, Berkum NL van, Meisig J, Sedat J, Gribnau J, Barillot E, Blüthgen N, Dekker J, and Heard E. Spatial partitioning of the regulatory landscape of the X-inactivation centre. Nature. 2012 Apr 11; vol. 485, no. 7398 :381–5

34. Dekker J and Heard E. Structural and Functional Diversity of Topologically Associating Domains. FEBS letters. 2015 Oct 7; vol. 589, no. 20 :2877–84

35. Acemel RD and Lupiáñez DG. Evolution of 3D chromatin organization at different scales. Current Opinion in Genetics & Development. 2023 Feb 1; vol. 78 :102019

36. Wit E de. TADs as the Caller Calls Them. Journal of Molecular Biology. Perspectives on Chromosome Folding 2020 Feb 7; vol. 432, no. 3 :638–42

37. Crane E, Bian Q, McCord RP, Lajoie BR, Wheeler BS, Ralston EJ, Uzawa S, Dekker J, and Meyer BJ. Condensin-Driven Remodeling of X-Chromosome Topology during Dosage Compensation. Nature. 2015 Jul 9; vol. 523, no. 7559 :240–4

38. Rao SSP, Huntley MH, Durand NC, Stamenova EK, Bochkov ID, Robinson JT, Sanborn AL, Machol I, Omer AD, Lander ES, and Aiden EL. A 3D Map of the Human Genome at Kilobase Resolution Reveals Principles of Chromatin Looping. Cell. 2014 Dec 18; vol. 159, no. 7 :1665–80

39. Weinreb C and Raphael BJ. Identification of hierarchical chromatin domains. Bioinformatics. 2016 Jun 1; vol. 32, no. 11 :1601–9

40. Forcato M, Nicoletti C, Pal K, Livi CM, Ferrari F, and Bicciato S. Comparison of computational methods for Hi-C data analysis. Nature Methods. 2017 Jul; vol. 14, no. 7. Number: 7 Publisher: Nature Publishing Group:679–85

41. Zufferey M, Tavernari D, Oricchio E, and Ciriello G. Comparison of computational methods for the identification of topologically associating domains. Genome Biology. 2018 Dec 10; vol. 19, no. 1 :217

42. Sefer E. A comparison of topologically associating domain callers over mammals at high resolution. BMC Bioinformatics. 2022 Apr 12; vol. 23, no. 1 :127

43. Dixon JR, Gorkin DU, and Ren B. Chromatin Domains: The Unit of Chromosome Organization. Molecular Cell. 2016 Jun 2; vol. 62, no. 5 :668–80

44. Kaiser VB and Semple CA. When TADs go bad: chromatin structure and nuclear organisation in human disease. F1000Research. 2017 Mar 24; vol. 6 :F1000 Faculty Rev–314

45. Nora EP, Goloborodko A, Valton AL, Gibcus JH, Uebersohn A, Abdennur N, Dekker J, Mirny LA, and Bruneau BG. Targeted Degradation of CTCF Decouples Local Insulation of Chromosome Domains from Genomic Compartmentalization. Cell. 2017 May 18; vol. 169, no. 5 :930–944.e22

46. Hsieh THS, Cattoglio C, Slobodyanyuk E, Hansen AS, Darzacq X, and Tjian R. Enhancer–promoter interactions and transcription are largely maintained upon acute loss of CTCF, cohesin, WAPL or YY1. Nature Genetics. 2022 Dec; vol. 54, no. 12. Number: 12 Publisher: Nature Publishing Group:1919–32

47. Wit E de and Nora EP. New insights into genome folding by loop extrusion from inducible degron technologies. Nature Reviews Genetics. 2023 Feb; vol. 24, no. 2. Number: 2 Publisher: Nature Publishing Group:73–85

48. GSE199059; GEO Accession viewer — identifiers.org

49. Guo Y, Al-Jibury E, Garcia-Millan R, Ntagiantas K, King JWD, Nash AJ, Galjart N, Lenhard B, Rueckert D, Fisher AG, Pruessner G, and Merkenschlager M. Chromatin jets define the properties of cohesin-driven in vivo loop extrusion. Molecular Cell. 2022 Oct 20; vol. 82, no. 20 :3769–3780.e5

50. Hsieh THS, Cattoglio C, Slobodyanyuk E, Hansen AS, Rando OJ, Tjian R, and Darzacq X. Resolving the 3D Landscape of Transcription-Linked Mammalian Chromatin Folding. Molecular Cell. 2020 May 7; vol. 78, no. 3 :539–553.e8

51. Goel VY, Huseyin MK, and Hansen AS. Region Capture Micro-C reveals coalescence of enhancers and promoters into nested microcompartments. Nature Genetics. 2023 Jun; vol. 55, no. 6. Number: 6 Publisher: Nature Publishing Group:1048–56

52. Isiaka BN, Semple JI, Haemmerli A, Thapliyal S, Stojanovski K, Das M, Gilbert N, Glauser DA, Towbin B, Jost D, and Meister P. Cohesin forms fountains at active enhancers in C. elegans. Pages: 2023.07.14.549011 Section: New Results. 2023 Jul 16

53. Kim J and Ercan S. Cohesin mediated loop extrusion from active enhancers form chromatin jets in C. elegans. Pages: 2023.09.18.558239 Section: New Results. 2023 Sep 20

54. Wike CL, Guo Y, Tan M, Nakamura R, Shaw DK, Díaz N, Whittaker-Tademy AF, Durand NC, Aiden EL, Vaquerizas JM, Grunwald D, Takeda H, and Cairns BR. Chromatin architecture transitions from zebrafish sperm through early embryogenesis. Genome Research. 2021 Jun 1; vol. 31, no. 6. Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution: Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab:981–94

55. Galitsyna A, Ulianov SV, Bykov NS, Veil M, Gao M, Perevoschikova K, Gelfand M, Razin SV, Mirny L, and Onichtchouk D. Extrusion fountains are hallmarks of chromosome organization emerging upon zygotic genome activation. Pages: 2023.07.15.549120 Section: New Results. 2023 Jul 15

56. ENCFF447ERX; ENCODE — identifiers.org

57. Lieberman-Aiden E, Berkum NL van, Williams L, Imakaev M, Ragoczy T, Telling A, Amit I, Lajoie BR, Sabo PJ, Dorschner MO, Sandstrom R, Bernstein B, Bender MA, Groudine M, Gnirke A, Stamatoyannopoulos J, Mirny LA, Lander ES, and Dekker J. Comprehensive Mapping of Long-Range Interactions Reveals Folding Principles of the Human Genome. Science. 2009 Oct 9; vol. 326, no. 5950. Publisher: American Association for the Advancement of Science:289–93

58. Miura H, Poonperm R, Takahashi S, and Hiratani I. Practical Analysis of Hi-C Data: Generating A/B Compartment Profiles. *X-Chromosome Inactivation: Methods and Protocols*. Ed. by Sado T. Methods in Molecular Biology. New York, NY: Springer, 2018 :221–45

59. Kalluchi A, Harris HL, Reznicek TE, and Rowley MJ. Considerations and caveats for analyzing chromatin compartments. Frontiers in Molecular Biosciences. 2023 Apr 5; vol. 10 :1168562

60. Durand NC, Shamim MS, Machol I, Rao SSP, Huntley MH, Lander ES, and Aiden EL. Juicer Provides a One-Click System for Analyzing Loop-Resolution Hi-C Experiments. Cell Systems. 2016 Jul 27; vol. 3, no. 1. Publisher: Elsevier:95–8

61. Giorgetti L, Lajoie BR, Carter AC, Attia M, Zhan Y, Xu J, Chen CJ, Kaplan N, Chang HY, Heard E, and Dekker J. Structural organization of the inactive X chromosome in the mouse. Nature. 2016 Jul; vol. 535, no. 7613. Number: 7613 Publisher: Nature Publishing Group:575–9

62. Kruse K, Hug CB, and Vaquerizas JM. FAN-C: a feature-rich framework for the analysis and visualisation of chromosome conformation capture data. Genome Biology. 2020 Dec 17; vol. 21, no. 1 :303

63. Weide RH van der, Brand T van den, Haarhuis JHI, Teunissen H, Rowland BD, and Wit E de. Hi-C analyses with GENOVA: a case study with cohesin variants. NAR Genomics and Bioinformatics. 2021 Jun 1; vol. 3, no. 2 :lqab040

64. Zheng X and Zheng Y. CscoreTool: fast Hi-C compartment analysis at high resolution. Bioinformatics. 2018 May 1; vol. 34, no. 9 :1568–70

65. Nichols MH and Corces VG. Principles of 3D compartmentalization of the human genome. Cell Reports. 2021 Jun 29; vol. 35, no. 13 :109330

66. Harris HL, Gu H, Olshansky M, Wang A, Farabella I, Eliaz Y, Kalluchi A, Krishna A, Jacobs M, Cauer G, Pham M, Rao SSP, Dudchenko O, Omer A, Mohajeri K, Kim S, Nichols MH, Davis ES, Gkountaroulis D, Udupa D, Aiden AP, Corces VG, Phanstiel DH, Noble WS, Nir G, Di Pierro M, Seo JS, Talkowski ME, Aiden EL, and Rowley MJ. Chromatin alternates between A and B compartments at kilobase scale for subgenic organization. Nature Communications. 2023 Jun 6; vol. 14, no. 1. Number: 1 Publisher: Nature Publishing Group:3303

67. Yaffe E and Tanay A. Probabilistic modeling of Hi-C contact maps eliminates systematic biases to characterize global chromosomal architecture. Nature Genetics. 2011 Nov; vol. 43, no. 11. Number: 11 Publisher: Nature Publishing Group:1059–65

68. Ashoor H, Chen X, Rosikiewicz W, Wang J, Cheng A, Wang P, Ruan Y, and Li S. Graph embedding and unsupervised learning predict genomic sub-compartments from HiC chromatin interaction data. Nature Communications. 2020 Mar 3; vol. 11, no. 1. Number: 1 Publisher: Nature Publishing Group:1173

69. Xiong K and Ma J. Revealing Hi-C subcompartments by imputing interchromosomal chromatin interactions. Nature Communications. 2019 Nov 7; vol. 10, no. 1. Number: 1 Publisher: Nature Publishing Group:5069

70. Liu Y, Nanni L, Sungalee S, Zufferey M, Tavernari D, Mina M, Ceri S, Oricchio E, and Ciriello G. Systematic inference and comparison of multi-scale chromatin sub-compartments connects spatial organization to cell phenotypes. Nature Communications. 2021 May 10; vol. 12, no. 1. Number: 1 Publisher: Nature Publishing Group:2439

71. Chakraborty A, Wang JG, and Ay F. dcHiC detects differential compartments across multiple Hi-C datasets. Nature Communications. 2022 Nov 11; vol. 13, no. 1. Number: 1 Publisher: Nature Publishing Group:6827

72. Wen Z, Zhang W, Zhong Q, Xu J, Hou C, Qin ZS, and Li L. Extensive Chromatin Structure-Function Associations Revealed by Accurate 3D Compartmentalization Characterization. Frontiers in Cell and Developmental Biology. 2022; vol. 10

73. Dodero-Rojas E, Mello MF, Brahmachari S, Oliveira Junior AB, Contessoto VG, and Onuchic JN. PyMEGABASE: Predicting Cell-Type-Specific Structural Annotations of Chromosomes Using the Epigenome. Journal of Molecular Biology. 2023 Aug 1; vol. 435, no. 15 :168180

74. Hildebrand E and Dekker J. Mechanisms and functions of chromosome compartmentalization. Trends in biochemical sciences. 2020 May; vol. 45, no. 5 :385–96

75. Collas P, Liyakat Ali TM, Brunet A, and Germier T. Finding Friends in the Crowd: Three-Dimensional Cliques of Topological Genomic Domains. Frontiers in Genetics. 2019; vol. 10

76. Paulsen J, Liyakat Ali TM, Nekrasov M, Delbarre E, Baudement MO, Kurscheid S, Tremethick D, and Collas P. Long-range interactions between topologically associating domains shape the four-dimensional genome during differentiation. Nature Genetics. 2019 May; vol. 51, no. 5. Number: 5 Publisher: Nature Publishing Group:835–43

77. Bron C and Kerbosch J. Algorithm 457: finding all cliques of an undirected graph. Communications of the ACM. 1973 Sep 1; vol. 16, no. 9 :575–7

78. Liyakat Ali TM, Brunet A, Collas P, and Paulsen J. TAD cliques predict key features of chromatin organization. BMC Genomics. 2021 Jul 3; vol. 22, no. 1 :499

79. Zhao Y, Ding Y, He L, Zhou Q, Chen X, Li Y, Alfonsi MV, Wu Z, Sun H, and Wang H. Multiscale 3D genome reorganization during skeletal muscle stem cell lineage progression and aging. Science Advances. 2023 Feb 17; vol. 9, no. 7. Publisher: American Association for the Advancement of Science:eabo1360

80. Szabo Q, Jost D, Chang JM, Cattoni DI, Papadopoulos GL, Bonev B, Sexton T, Gurgo J, Jacquier C, Nollmann M, Bantignies F, and Cavalli G. TADs are 3D structural units of higher-order chromosome organization in Drosophila. Science Advances. 2018 Feb 28; vol. 4, no. 2 :eaar8082

81. Pei L, Huang X, Liu Z, Tian X, You J, Li J, Fang DD, Lindsey K, Zhu L, Zhang X, and Wang M. Dynamic 3D genome architecture of cotton fiber reveals subgenome-coordinated chromatin topology for 4-staged single-cell differentiation. Genome Biology. 2022 Feb 3; vol. 23, no. 1 :45

82. Li Y, Xu W, Wang Y, Kou J, Zhang J, Hu S, Zhang L, Wang J, Liu J, Liu H, Luo L, Wang C, Lan J, Hou R, and Shen F. An improved, chromosome-level genome of the giant panda (Ailuropoda melanoleuca). Genomics. 2022 Nov 1; vol. 114, no. 6 :110501

83. Rabl C. Uber zellthilung. Morphol Jahrb. 1885; vol. 10 :214–330

84. Cremer T and Cremer C. Chromosome territories, nuclear architecture and gene regulation in mammalian cells. Nature Reviews Genetics. 2001 Apr; vol. 2, no. 4. Number: 4 Publisher: Nature Publishing Group:292–301

85. Sun HB, Shen J, and Yokota H. Size-Dependent Positioning of Human Chromosomes in Interphase Nuclei. Biophysical Journal. 2000 Jul 1; vol. 79, no. 1 :184–90

86. Bolzer A, Kreth G, Solovei I, Koehler D, Saracoglu K, Fauth C, Müller S, Eils R, Cremer C, Speicher MR, and Cremer T. Three-Dimensional Maps of All Chromosomes in Human Male Fibroblast Nuclei and Prometaphase Rosettes. PLOS Biology. 2005 Apr 26; vol. 3, no. 5. Publisher: Public Library of Science:e157

87. Parada LA, McQueen PG, and Misteli T. Tissue-specific spatial organization of genomes. Genome Biology. 2004 Jun 21; vol. 5, no. 7 :R44

88. Olivares-Chauvet P, Fennessy D, Jackson DA, and Maya-Mendoza A. Innate Structure of DNA Foci Restricts the Mixing of DNA from Different Chromosome Territories. PLOS ONE. 2011 Dec 21; vol. 6, no. 12. Publisher: Public Library of Science:e27527

89. Branco MR and Pombo A. Intermingling of Chromosome Territories in Interphase Suggests Role in Translocations and Transcription-Dependent Associations. PLOS Biology. 2006 Apr 25; vol. 4, no. 5. Publisher: Public Library of Science:e138

90. Maharana S, Iyer KV, Jain N, Nagarajan M, Wang Y, and Shivashankar GV. Chromosome intermingling—the physical basis of chromosome organization in differentiated cells. Nucleic Acids Research. 2016 Jun 20; vol. 44, no. 11 :5148–60

91. Szczepińska T, Rusek AM, and Plewczynski D. Intermingling of chromosome territories. Genes, Chromosomes and Cancer. 2019; vol. 58, no. 7. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/gcc.22736:500–6

92. Hetzer MW. The Nuclear Envelope. Cold Spring Harbor Perspectives in Biology. 2010 Mar; vol. 2, no. 3 :a000539

93. De Magistris P and Antonin W. The Dynamic Nature of the Nuclear Envelope. Current biology: CB. 2018 Apr 23; vol. 28, no. 8 :R487–R497

94. Gruenbaum Y, Goldman RD, Meyuhas R, Mills E, Margalit A, Fridkin A, Dayani Y, Prokocimer M, and Enosh A. The nuclear lamina and its functions in the nucleus. International Review of Cytology. 2003; vol. 226 :1–62

95. Leeuw R de, Gruenbaum Y, and Medalia O. Nuclear Lamins: Thin Filaments with Major Functions. Trends in Cell Biology. 2018 Jan; vol. 28, no. 1 :34–45

96. Briand N and Collas P. Lamina-associated domains: peripheral matters and internal affairs. Genome Biology. 2020 Apr 2; vol. 21, no. 1 :85

97. Beck M and Hurt E. The nuclear pore complex: understanding its function through structural insight. Nature Reviews Molecular Cell Biology. 2017 Feb; vol. 18, no. 2. Number: 2 Publisher: Nature Publishing Group:73–89

98. Lin DH and Hoelz A. The Structure of the Nuclear Pore Complex (An Update). Annual review of biochemistry. 2019 Jun 20; vol. 88 :725–83

99. Staněk D and Fox AH. Nuclear bodies: news insights into structure and function. Current Opinion in Cell Biology. 2017 Jun; vol. 46 :94–101

100. Dubois ML and Boisvert FM. The Nucleolus: Structure and Function. The Functional Nucleus. Publisher: Nature Publishing Group:29

101. Iarovaia OV, Minina EP, Sheval EV, Onichtchouk D, Dokudovskaya S, Razin SV, and Vassetzky YS. Nucleolus: A Central Hub for Nuclear Functions. Trends in Cell Biology. 2019 Aug 1; vol. 29, no. 8. Publisher: Elsevier:647–59

102. Chang HR, Munkhjargal A, Kim MJ, Park SY, Jung E, Ryu JH, Yang Y, Lim JS, and Kim Y. The functional roles of PML nuclear bodies in genome maintenance. Mutation Research. 2018 May; vol. 809 :99–107

103. Spector DL and Lamond AI. Nuclear Speckles. Cold Spring Harbor Perspectives in Biology. 2011 Feb; vol. 3, no. 2 :a000646

104. Galganski L, Urbanek MO, and Krzyzosiak WJ. Nuclear speckles: molecular organization, biological function and role in disease. Nucleic Acids Research. 2017 Oct 13; vol. 45, no. 18 :10350–68

105. Fox AH and Lamond AI. Paraspeckles. Cold Spring Harbor Perspectives in Biology. 2010 Jul; vol. 2, no. 7 :a000687

106. Wang Y and Chen LL. Organization and function of paraspeckles. Essays in Biochemistry. 2020 Dec 7; vol. 64, no. 6 :875–82

107. Kim S, Yu NK, and Kaang BK. CTCF as a multifunctional protein in genome regulation and gene expression. Experimental & Molecular Medicine. 2015 Jun; vol. 47, no. 6. Number: 6 Publisher: Nature Publishing Group:e166–e166

108. Li Y, Haarhuis JHI, Sedeño Cacciatore Á, Oldenkamp R, Ruiten MS van, Willems L, Teunissen H, Muir KW, Wit E de, Rowland BD, and Panne D. The structural basis for cohesin–CTCF-anchored loops. Nature. 2020 Feb; vol. 578, no. 7795. Number: 7795 Publisher: Nature Publishing Group:472–6

109. Nichols MH and Corces VG. A tethered-inchworm model of SMC DNA translocation. Nature Structural & Molecular Biology. 2018 Oct; vol. 25, no. 10. Number: 10 Publisher: Nature Publishing Group:906–10

110. Diebold-Durand ML, Lee H, Ruiz Avila LB, Noh H, Shin HC, Im H, Bock FP, Bürmann F, Durand A, Basfeld A, Ham S, Basquin J, Oh BH, and Gruber S. Structure of Full-Length SMC and Rearrangements Required for Chromosome Organization. Molecular Cell. 2017 Jul 20; vol. 67, no. 2 :334–347.e5

111. Ryu JK, Katan AJ, Sluis EO van der, Wisse T, Groot R de, Haering CH, and Dekker C. The condensin holocomplex cycles dynamically between open and collapsed states. Nature Structural & Molecular Biology. 2020 Dec; vol. 27, no. 12. Number: 12 Publisher: Nature Publishing Group:1134–41

112. Bauer BW, Davidson IF, Canena D, Wutz G, Tang W, Litos G, Horn S, Hinterdorfer P, and Peters JM. Cohesin mediates DNA loop extrusion by a "swing and clamp" mechanism. Cell. 2021 Oct 14; vol. 184, no. 21 :5448–5464.e22

113. Zhang S, Übelmesser N, Barbieri M, and Papantonis A. Enhancer–promoter contact formation requires RNAPII and antagonizes loop extrusion. Nature Genetics. 2023 May; vol. 55, no. 5. Number: 5 Publisher: Nature Publishing Group:832–40

114. Ciosk R, Shirayama M, Shevchenko A, Tanaka T, Toth A, Shevchenko A, and Nasmyth K. Cohesin's binding to chromosomes depends on a separate complex consisting of Scc2 and Scc4 proteins. Molecular Cell. 2000 Feb; vol. 5, no. 2 :243–54

115. Takahashi TS, Yiu P, Chou MF, Gygi S, and Walter JC. Recruitment of Xenopus Scc2 and cohesin to chromatin requires the pre-replication complex. Nature Cell Biology. 2004 Oct; vol. 6, no. 10. Number: 10 Publisher: Nature Publishing Group:991–6

116. Gandhi R, Gillespie PJ, and Hirano T. Human Wapl is a cohesin-binding protein that promotes sister-chromatid resolution in mitotic prophase. Current biology: CB. 2006 Dec 19; vol. 16, no. 24 :2406–17

117. Kueng S, Hegemann B, Peters BH, Lipp JJ, Schleiffer A, Mechtler K, and Peters JM. Wapl Controls the Dynamic Association of Cohesin with Chromatin. Cell. 2006 Dec 1; vol. 127, no. 5. Publisher: Elsevier:955–67

118. Shintomi K and Hirano T. Releasing cohesin from chromosome arms in early mitosis: opposing actions of Wapl–Pds5 and Sgo1. Genes & Development. 2009 Aug 20. Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution: Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab

119. Davidson IF, Bauer B, Goetz D, Tang W, Wutz G, and Peters JM. DNA loop extrusion by human cohesin. Science. 2019 Dec 13; vol. 366, no. 6471. Publisher: American Association for the Advancement of Science:1338–45

120. Kim Y, Shi Z, Zhang H, Finkelstein IJ, and Yu H. Human cohesin compacts DNA by loop extrusion. Science. 2019 Dec 13; vol. 366, no. 6471. Publisher: American Association for the Advancement of Science:1345–9

121. Minamino M, Ishibashi M, Nakato R, Akiyama K, Tanaka H, Kato Y, Negishi L, Hirota T, Sutani T, Bando M, and Shirahige K. Esco1 Acetylates Cohesin via a Mechanism Different from That of Esco2. Current biology: CB. 2015 Jun 29; vol. 25, no. 13 :1694–706

122. Deardorff MA, Bando M, Nakato R, Watrin E, Itoh T, Minamino M, Saitoh K, Komata M, Katou Y, Clark D, Cole KE, De Baere E, Decroos C, Di Donato N, Ernst S, Francey LJ, Gyftodimou Y, Hirashima K, Hullings M, Ishikawa Y, Jaulin C, Kaur M, Kiyono T, Lombardi PM, Magnaghi-Jaulin L, Mortier GR, Nozaki N, Petersen MB, Seimiya H, Siu VM, Suzuki Y, Takagaki K, Wilde JJ, Willems PJ, Prigent C, Gillessen-Kaesbach G, Christianson DW, Kaiser FJ, Jackson LG, Hirota T, Krantz ID, and Shirahige K. HDAC8 mutations in Cornelia de Lange syndrome affect the cohesin acetylation cycle. Nature. 2012 Sep; vol. 489, no. 7415. Number: 7415 Publisher: Nature Publishing Group:313–7

123. Ruiten MS van, Gent D van, Sedeño Cacciatore Á, Fauster A, Willems L, Hekkelman ML, Hoekman L, Altelaar M, Haarhuis JHI, Brummelkamp TR, Wit E de, and Rowland BD. The cohesin acetylation cycle controls chromatin loop length through a PDS5A brake mechanism. Nature Structural & Molecular Biology. 2022 Jun; vol. 29, no. 6. Number: 6 Publisher: Nature Publishing Group:586–91

124. Parelho V, Hadjur S, Spivakov M, Leleu M, Sauer S, Gregson HC, Jarmuz A, Canzonetta C, Webster Z, Nesterova T, Cobb BS, Yokomori K, Dillon N, Aragon L, Fisher AG, and Merkenschlager M. Cohesins Functionally Associate with CTCF on Mammalian Chromosome Arms. Cell. 2008 Feb 8; vol. 132, no. 3. Publisher: Elsevier:422–33

125. Wutz G, Várnai C, Nagasaka K, Cisneros DA, Stocsits RR, Tang W, Schoenfelder S, Jessberger G, Muhar M, Hossain MJ, Walther N, Koch B, Kueblbeck M, Ellenberg J, Zuber J, Fraser P, and Peters JM. Topologically associating domains and chromatin loops depend on cohesin and are regulated by CTCF, WAPL, and PDS5 proteins. The EMBO Journal. 2017 Dec 15; vol. 36, no. 24. Publisher: John Wiley & Sons, Ltd:3573–99

126. Pugacheva EM, Kubo N, Loukinov D, Tajmul M, Kang S, Kovalchuk AL, Strunnikov AV, Zentner GE, Ren B, and Lobanenkov VV. CTCF mediates chromatin looping via N-terminal domain-dependent cohesin retention. Proceedings of the National Academy of Sciences. 2020 Jan 28; vol. 117, no. 4. Publisher: Proceedings of the National Academy of Sciences:2020–31

127. Zhang S, Übelmesser N, Josipovic N, Forte G, Slotman JA, Chiang M, Gothe HJ, Gusmao EG, Becker C, Altmüller J, Houtsmuller AB, Roukos V, Wendt KS, Marenduzzo D, and Papantonis A. RNA polymerase II is required for spatial chromatin reorganization following exit from mitosis. Science Advances. 2021 Oct 22; vol. 7, no. 43. Publisher: American Association for the Advancement of Science:eabg8205

128. Banigan EJ, Tang W, Berg AA van den, Stocsits RR, Wutz G, Brandão HB, Busslinger GA, Peters JM, and Mirny LA. Transcription shapes 3D chromatin organization by interacting with loop extrusion. Proceedings of the National Academy of Sciences. 2023 Mar 14; vol. 120, no. 11. Publisher: Proceedings of the National Academy of Sciences:e2210480120

129. Pradhan B, Barth R, Kim E, Davidson IF, Bauer B, Laar T van, Yang W, Ryu JK, Torre J van der, Peters JM, and Dekker C. SMC complexes can traverse physical roadblocks bigger than their ring size. Cell Reports. 2022 Oct 18; vol. 41, no. 3 :111491

130. Davidson IF, Barth R, Zaczek M, Torre J van der, Tang W, Nagasaka K, Janissen R, Kerssemakers J, Wutz G, Dekker C, and Peters JM. CTCF is a DNA-tension-dependent barrier to cohesin-mediated loop extrusion. Nature. 2023 Apr; vol. 616, no. 7958. Number: 7958 Publisher: Nature Publishing Group:822–7

131. Fudenberg G, Imakaev M, Lu C, Goloborodko A, Abdennur N, and Mirny LA. Formation of Chromosomal Domains by Loop Extrusion. Cell Reports. 2016 May 31; vol. 15, no. 9 :2038–49

132. Corsi F, Rusch E, and Goloborodko A. Loop extrusion rules: the next generation. Current Opinion in Genetics & Development. 2023 Aug 1; vol. 81 :102061

133. Symmons O, Uslu VV, Tsujimura T, Ruf S, Nassari S, Schwarzer W, Ettwiller L, and Spitz F. Functional and topological characteristics of mammalian regulatory domains. Genome Research. 2014 Mar 1; vol. 24, no. 3. Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution: Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab:390–400

134. Rao SSP, Huang SC, Glenn St Hilaire B, Engreitz JM, Perez EM, Kieffer-Kwon KR, Sanborn AL, Johnstone SE, Bascom GD, Bochkov ID, Huang X, Shamim MS, Shin J, Turner D, Ye Z, Omer AD, Robinson JT, Schlick T, Bernstein BE, Casellas R, Lander ES, and Aiden EL. Cohesin Loss Eliminates All Loop Domains. Cell. 2017 Oct 5; vol. 171, no. 2 :305–320.e24

135. Schwarzer W, Abdennur N, Goloborodko A, Pekowska A, Fudenberg G, Loe-Mie Y, Fonseca NA, Huber W, Haering CH, Mirny L, and Spitz F. Two independent modes of chromatin organization revealed by cohesin removal. Nature. 2017 Nov; vol. 551, no. 7678. Number: 7678 Publisher: Nature Publishing Group:51–6

136. Karpinska MA and Oudelaar AM. The role of loop extrusion in enhancer-mediated gene activation. Current Opinion in Genetics & Development. 2023 Apr 1; vol. 79 :102022

137. Peters JM. How DNA loop extrusion mediated by cohesin enables V(D)J recombination. Current Opinion in Cell Biology. Cell Nucleus 2021 Jun 1; vol. 70 :75–83

138. Bassing CH, Swat W, and Alt FW. The Mechanism and Regulation of Chromosomal V(D)J Recombination. Cell. 2002 Apr 19; vol. 109, no. 2. Publisher: Elsevier:S45–S55

139. Jain S, Ba Z, Zhang Y, Dai HQ, and Alt FW. CTCF-Binding Elements Mediate Accessibility of RAG Substrates During Chromatin Scanning. Cell. 2018 Jun 28; vol. 174, no. 1 :102–116.e14

140. Lin SG, Ba Z, Alt FW, and Zhang Y. RAG Chromatin Scanning During V(D)J Recombination and Chromatin Loop Extrusion are Related Processes. Advances in Immunology. 2018; vol. 139 :93–135

141. Zhang Y, Zhang X, Ba Z, Liang Z, Dring EW, Hu H, Lou J, Kyritsis N, Zurita J, Shamim MS, Presser Aiden A, Lieberman Aiden E, and Alt FW. The fundamental role of chromatin loop extrusion in physiological V(D)J recombination. Nature. 2019 Sep; vol. 573, no. 7775 :600–4

142. Ba Z, Lou J, Ye AY, Dai HQ, Dring EW, Lin SG, Jain S, Kyritsis N, Kieffer-Kwon KR, Casellas R, and Alt FW. CTCF orchestrates long-range cohesin-driven V(D)J recombinational scanning. Nature. 2020 Oct; vol. 586, no. 7828. Number: 7828 Publisher: Nature Publishing Group:305–10

143. Hill L, Ebert A, Jaritz M, Wutz G, Nagasaka K, Tagoh H, Kostanova-Poliakova D, Schindler K, Sun Q, Bönelt P, Fischer M, Peters JM, and Busslinger M. Wapl repression by Pax5 promotes V gene recombination by Igh loop extrusion. Nature. 2020 Aug; vol. 584, no. 7819 :142–7

144. Dai HQ, Hu H, Lou J, Ye AY, Ba Z, Zhang X, Zhang Y, Zhao L, Yoon HS, Chapdelaine-Williams AM, Kyritsis N, Chen H, Johnson K, Lin S, Conte A, Casellas R, Lee CS, and Alt FW. Loop extrusion mediates physiological Igh locus contraction for RAG scanning. Nature. 2021 Feb; vol. 590, no. 7845 :338–43

145. Ward JF. The yield of DNA double-strand breaks produced intracellularly by ionizing radiation: a review. International Journal of Radiation Biology. 1990 Jun; vol. 57, no. 6 :1141–50

146. Ross WE and Bradley MO. DNA double-stranded breaks in mammalian cells after exposure to intercalating agents. Biochimica Et Biophysica Acta. 1981 Jun 26; vol. 654, no. 1 :129–34

147. Chiarle R, Zhang Y, Frock RL, Lewis SM, Molinie B, Ho YJ, Myers DR, Choi VW, Compagno M, Malkin DJ, Neuberg D, Monti S, Giallourakis CC, Gostissa M, and Alt FW. Genome-wide Translocation Sequencing Reveals Mechanisms of Chromosome Breaks and Rearrangements in B Cells. Cell. 2011 Sep 30; vol. 147, no. 1. Publisher: Elsevier:107–19

148. Barlow JH, Faryabi RB, Callén E, Wong N, Malhowski A, Chen HT, Gutierrez-Cruz G, Sun HW, McKinnon P, Wright G, Casellas R, Robbiani DF, Staudt L, Fernandez-Capetillo O, and Nussenzweig A. Identification of early replicating fragile sites that contribute to genome instability. Cell. 2013 Jan 31; vol. 152, no. 3 :620–32

149. Rogakou EP, Boon C, Redon C, and Bonner WM. Megabase chromatin domains involved in DNA double-strand breaks in vivo. The Journal of Cell Biology. 1999 Sep 6; vol. 146, no. 5 :905–16

150. Lee JH and Paull TT. Activation and regulation of ATM kinase activity in response to DNA double-strand breaks. Oncogene. 2007 Dec; vol. 26, no. 56. Number: 56 Publisher: Nature Publishing Group:7741–8

151. Arnould C, Rocher V, Finoux AL, Clouaire T, Li K, Zhou F, Caron P, Mangeot PE, Ricci EP, Mourad R, Haber JE, Noordermeer D, and Legube G. Loop extrusion as a mechanism for formation of DNA damage repair foci. Nature. 2021 Feb; vol. 590, no. 7847. Number: 7847 Publisher: Nature Publishing Group:660–5

152. Alipour E and Marko JF. Self-organization of domain structures by DNA-loop-extruding enzymes. Nucleic Acids Research. 2012 Dec 1; vol. 40, no. 22 :11202–12

153. Naumova N, Imakaev M, Fudenberg G, Zhan Y, Lajoie BR, Mirny LA, and Dekker J. Organization of the Mitotic Chromosome. Science. 2013 Nov 22; vol. 342, no. 6161. Publisher: American Association for the Advancement of Science:948–53

154. Goloborodko A, Marko JF, and Mirny LA. Chromosome Compaction by Active Loop Extrusion. Biophysical Journal. 2016 May 24; vol. 110, no. 10 :2162–8

155. Sanborn AL, Rao SSP, Huang SC, Durand NC, Huntley MH, Jewett AI, Bochkov ID, Chinnappan D, Cutkosky A, Li J, Geeting KP, Gnirke A, Melnikov A, McKenna D, Stamenova EK, Lander ES, and Aiden EL. Chromatin extrusion explains key features of loop and domain formation in wild-type and engineered genomes. Proceedings of the National Academy of Sciences. 2015 Nov 24; vol. 112, no. 47. Publisher: Proceedings of the National Academy of Sciences:E6456–E6465

156. Nuebler J, Fudenberg G, Imakaev M, Abdennur N, and Mirny LA. Chromatin organization by an interplay of loop extrusion and compartmental segregation. Proceedings of the National Academy of Sciences. 2018 Jul 17; vol. 115, no. 29. Publisher: Proceedings of the National Academy of Sciences:E6697–E6706

157. Pereira MCF, Brackley CA, Michieletto D, Annunziatella C, Bianco S, Chiariello AM, Nicodemi M, and Marenduzzo D. Complementary chromosome folding by transcription factors and cohesin. Pages: 305359 Section: New Results. 2018 Apr 23

158. Buckle A, Brackley CA, Boyle S, Marenduzzo D, and Gilbert N. Polymer Simulations of Heteromorphic Chromatin Predict the 3D Folding of Complex Genomic Loci. Molecular Cell. 2018 Nov 15; vol. 72, no. 4 :786–797.e11

159. Banigan EJ, Berg AA van den, Brandão HB, Marko JF, and Mirny LA. Chromosome organization by one-sided and two-sided loop extrusion. eLife. 2020 Apr 6; vol. 9. Ed. by Struhl K. Publisher: eLife Sciences Publications, Ltd:e53558

160. Xi W and Beer MA. Loop competition and extrusion model predicts CTCF interaction specificity. Nature Communications. 2021 Feb 16; vol. 12, no. 1. Number: 1 Publisher: Nature Publishing Group:1046

161. Gabriele M, Brandão HB, Grosse-Holz S, Jha A, Dailey GM, Cattoglio C, Hsieh THS, Mirny L, Zechner C, and Hansen AS. Dynamics of CTCF- and cohesin-mediated chromatin looping revealed by live-cell imaging. Science. 2022 Apr 29; vol. 376, no. 6592. Publisher: American Association for the Advancement of Science:496–501

162. Dequeker BJH, Scherr MJ, Brandão HB, Gassler J, Powell S, Gaspar I, Flyamer IM, Lalic A, Tang W, Stocsits R, Davidson IF, Peters JM, Duderstadt KE, Mirny LA, and Tachibana K. MCM complexes are barriers that restrict cohesin-mediated loop extrusion. Nature. 2022 Jun; vol. 606, no. 7912. Number: 7912 Publisher: Nature Publishing Group:197–203

163. Dekker J, Rippe K, Dekker M, and Kleckner N. Capturing Chromosome Conformation. Science. 2002 Feb 15; vol. 295, no. 5558. Publisher: American Association for the Advancement of Science:1306–11

164. Zhao Z, Tavoosidana G, Sjölinder M, Göndör A, Mariano P, Wang S, Kanduri C, Lezcano M, Singh Sandhu K, Singh U, Pant V, Tiwari V, Kurukuti S, and Ohlsson R. Circular chromosome conformation capture (4C) uncovers extensive networks of epigenetically regulated intra- and interchromosomal interactions. Nature Genetics. 2006 Nov; vol. 38, no. 11. Number: 11 Publisher: Nature Publishing Group:1341–7

165. Dostie J, Richmond TA, Arnaout RA, Selzer RR, Lee WL, Honan TA, Rubio ED, Krumm A, Lamb J, Nusbaum C, Green RD, and Dekker J. Chromosome Conformation Capture Carbon Copy (5C): A massively parallel solution for mapping interactions between genomic elements. Genome Research. 2006 Oct; vol. 16, no. 10 :1299–309

166. Belaghzal H, Dekker J, and Gibcus JH. HI-C 2.0: AN OPTIMIZED HI-C PROCEDURE FOR HIGH-RESOLUTION GENOME-WIDE MAPPING OF CHROMOSOME CONFORMATION. Methods (San Diego, Calif). 2017 Jul 1; vol. 123 :56–65

167. Lafontaine DL, Yang L, Dekker J, and Gibcus JH. Hi-C 3.0: Improved Protocol for Genome-Wide Chromosome Conformation Capture. Current Protocols. 2021 Jul; vol. 1, no. 7 :e198

168. Dryden NH, Broome LR, Dudbridge F, Johnson N, Orr N, Schoenfelder S, Nagano T, Andrews S, Wingett S, Kozarewa I, Assiotis I, Fenwick K, Maguire SL, Campbell J, Natrajan R, Lambros M, Perrakis E, Ashworth A, Fraser P, and Fletcher O. Unbiased analysis of potential targets of breast cancer susceptibility loci by Capture Hi-C. Genome Research. 2014 Nov; vol. 24, no. 11 :1854–68

169. Ma W, Ay F, Lee C, Gulsoy G, Deng X, Cook S, Hesson J, Cavanaugh C, Ware CB, Krumm A, Shendure J, Blau CA, Disteche CM, Noble WS, and Duan Z. Fine-scale chromatin interaction maps reveal the cis-regulatory landscape of human lincRNA genes. Nature Methods. 2015 Jan; vol. 12, no. 1. Number: 1 Publisher: Nature Publishing Group:71–8

170. Ramani V, Cusanovich DA, Hause RJ, Ma W, Qiu R, Deng X, Blau CA, Disteche CM, Noble WS, Shendure J, and Duan Z. Mapping 3D genome architecture through in situ DNase Hi-C. Nature Protocols. 2016 Nov; vol. 11, no. 11. Number: 11 Publisher: Nature Publishing Group:2104–21

171. Hsieh THS, Weiner A, Lajoie B, Dekker J, Friedman N, and Rando OJ. Mapping Nucleosome Resolution Chromosome Folding in Yeast by Micro-C. Cell. 2015 Jul 2; vol. 162, no. 1 :108–19

172. Hsieh THS, Fudenberg G, Goloborodko A, and Rando OJ. Micro-C XL: assaying chromosome conformation from the nucleosome to the entire genome. Nature Methods. 2016 Dec; vol. 13, no. 12. Number: 12 Publisher: Nature Publishing Group:1009–11

173. Deshpande AS, Ulahannan N, Pendleton M, Dai X, Ly L, Behr JM, Schwenk S, Liao W, Augello MA, Tyer C, Rughani P, Kudman S, Tian H, Otis HG, Adney E, Wilkes D, Mosquera JM, Barbieri CE, Melnick A, Stoddart D, Turner DJ, Juul S, Harrington E, and Imieliński M. Identifying synergistic high-order 3D chromatin conformations from genome-scale nanopore concatemer sequencing. Nature Biotechnology. 2022 Oct; vol. 40, no. 10. Number: 10 Publisher: Nature Publishing Group:1488–99

174. Zhong JY, Niu L, Lin ZB, Bai X, Chen Y, Luo F, Hou C, and Xiao CL. High-throughput Pore-C reveals the single-allele topology and cell type-specificity of 3D genome folding. Nature Communications. 2023 Mar 6; vol. 14, no. 1. Number: 1 Publisher: Nature Publishing Group:1250

175. Li W, Lu J, Lu P, Gao Y, Bai Y, Chen K, Su X, Li M, Liu J, Chen Y, Wen L, and Tang F. scNanoHi-C: a single-cell long-read concatemer sequencing method to reveal high-order chromatin structures within individual cells. Nature Methods. 2023 Oct; vol. 20, no. 10. Number: 10 Publisher: Nature Publishing Group:1493–505

176. Kempfer R and Pombo A. Methods for mapping 3D chromosome architecture. Nature Reviews Genetics. 2020 Apr; vol. 21, no. 4. Number: 4 Publisher: Nature Publishing Group:207–26

177. Beagrie RA, Scialdone A, Schueler M, Kraemer DCA, Chotalia M, Xie SQ, Barbieri M, Santiago I de, Lavitas LM, Branco MR, Fraser J, Dostie J, Game L, Dillon N, Edwards PAW, Nicodemi M, and Pombo A. Complex multi-enhancer contacts captured by genome architecture mapping. Nature. 2017 Mar; vol. 543, no. 7646. Number: 7646 Publisher: Nature Publishing Group:519–24

178. Beagrie RA, Thieme CJ, Annunziatella C, Baugher C, Zhang Y, Schueler M, Kukalev A, Kempfer R, Chiariello AM, Bianco S, Li Y, Davis T, Scialdone A, Welch LR, Nicodemi M, and Pombo A. Multiplex-GAM: genome-wide identification of chromatin contacts yields insights overlooked by Hi-C. Nature Methods. 2023 Jul; vol. 20, no. 7. Number: 7 Publisher: Nature Publishing Group:1037–47

179. Quinodoz SA, Ollikainen N, Tabak B, Palla A, Schmidt JM, Detmar E, Lai MM, Shishkin AA, Bhat P, Takei Y, Trinh V, Aznauryan E, Russell P, Cheng C, Jovanovic M, Chow A, Cai L, McDonel P, Garber M, and Guttman M. Higher-Order Inter-chromosomal Hubs Shape 3D Genome Organization in the Nucleus. Cell. 2018 Jul 26; vol. 174, no. 3 :744–757.e24

180. Quinodoz SA, Bhat P, Chovanec P, Jachowicz JW, Ollikainen N, Detmar E, Soehalim E, and Guttman M. SPRITE: a genome-wide method for mapping higher-order 3D interactions in the nucleus using combinatorial split-and-pool barcoding. Nature Protocols. 2022 Jan; vol. 17, no. 1. Number: 1 Publisher: Nature Publishing Group:36–75

181. Quinodoz SA, Jachowicz JW, Bhat P, Ollikainen N, Banerjee AK, Goronzy IN, Blanco MR, Chovanec P, Chow A, Markaki Y, Thai J, Plath K, and Guttman M. RNA promotes the formation of spatial compartments in the nucleus. Cell. 2021 Nov 11; vol. 184, no. 23 :5775–5790.e30

182. Arrastia MV, Jachowicz JW, Ollikainen N, Curtis MS, Lai C, Quinodoz SA, Selck DA, Ismagilov RF, and Guttman M. Single-cell measurement of higher-order 3D genome organization with scSPRITE. Nature Biotechnology. 2022 Jan; vol. 40, no. 1. Number: 1 Publisher: Nature Publishing Group:64–73

183. Zheng M, Tian SZ, Capurso D, Kim M, Maurya R, Lee B, Piecuch E, Gong L, Zhu JJ, Li Z, Wong CH, Ngan CY, Wang P, Ruan X, Wei CL, and Ruan Y. Multiplex chromatin interactions with single-molecule precision. Nature. 2019 Feb; vol. 566, no. 7745. Number: 7745 Publisher: Nature Publishing Group:558–62

184. Lai B, Tang Q, Jin W, Hu G, Wangsa D, Cui K, Stanton BZ, Ren G, Ding Y, Zhao M, Liu S, Song J, Ried T, and Zhao K. Trac-looping measures genome structure and chromatin accessibility. Nature Methods. 2018 Sep; vol. 15, no. 9. Number: 9 Publisher: Nature Publishing Group:741–7

185. Langer-Safer PR, Levine M, and Ward DC. Immunological method for mapping genes on Drosophila polytene chromosomes. Proceedings of the National Academy of Sciences of the United States of America. 1982 Jul; vol. 79, no. 14 :4381–5

186. Croft JA, Bridger JM, Boyle S, Perry P, Teague P, and Bickmore WA. Differences in the Localization and Morphology of Chromosomes in the Human Nucleus. Journal of Cell Biology. 1999 Jun 14; vol. 145, no. 6 :1119–31

187. Westendorf C, Bae AJ, Erlenkamper C, Galland E, Franck C, Bodenschatz E, and Beta C. Live cell flattening — traditional and novel approaches. PMC Biophysics. 2010 Apr 19; vol. 3 :9

188. Cremer M, Grasser F, Lanctôt C, Müller S, Neusser M, Zinner R, Solovei I, and Cremer T. Multicolor 3D Fluorescence In Situ Hybridization for Imaging Interphase Chromosomes. *The Nucleus: Volume 1: Nuclei and Subnuclear Components*. Ed. by Hancock R. Methods in Molecular Biology. Totowa, NJ: Humana Press, 2008 :205–39

189. Jerkovic´ I and Cavalli G. Understanding 3D genome organization by multidisciplinary methods. Nature Reviews Molecular Cell Biology. 2021 Aug; vol. 22, no. 8. Number: 8 Publisher: Nature Publishing Group:511–28

190. Xie SQ, Lavitas LM, and Pombo A. CryoFISH: Fluorescence In Situ Hybridization on Ultrathin Cryosections. *Fluorescence in situ Hybridization (FISH): Protocols and Applications*. Ed. by Bridger JM and Volpi EV. Methods in Molecular Biology. Totowa, NJ: Humana Press, 2010 :219–30

191. Hu M and Wang S. Chromatin Tracing: Imaging 3D Genome and Nucleome. Trends in Cell Biology. 2021 Jan 1; vol. 31, no. 1 :5–8

192. Beliveau BJ, Apostolopoulos N, and Wu Ct. Visualizing genomes with Oligopaint FISH probes. Current protocols in molecular biology / edited by Frederick M Ausubel [et al]. 2014 Jan 6; vol. 105 :14.23.1–14.23.20

193. Bouwman BAM, Crosetto N, and Bienko M. The era of 3D and spatial genomics. Trends in Genetics. 2022 Oct 1; vol. 38, no. 10. Publisher: Elsevier:1062–75

194. Bintu B, Mateo LJ, Su JH, Sinnott-Armstrong NA, Parker M, Kinrot S, Yamaya K, Boettiger AN, and Zhuang X. Super-resolution chromatin tracing reveals domains and cooperative interactions in single cells. Science. 2018 Oct 26; vol. 362, no. 6413. Publisher: American Association for the Advancement of Science:eaau1783

195. Wang S, Su JH, Beliveau BJ, Bintu B, Moffitt JR, Wu Ct, and Zhuang X. Spatial organization of chromatin domains and compartments in single chromosomes. Science (New York, NY). 2016 Aug 5; vol. 353, no. 6299 :598–602

196. Nir G, Farabella I, Estrada CP, Ebeling CG, Beliveau BJ, Sasaki HM, Lee SD, Nguyen SC, McCole RB, Chattoraj S, Erceg J, Abed JA, Martins NMC, Nguyen HQ, Hannan MA, Russell S, Durand NC, Rao SSP, Kishi JY, Soler-Vila P, Pierro MD, Onuchic JN, Callahan SP, Schreiner JM, Stuckey JA, Yin P, Aiden EL, Marti-Renom MA, and Wu Ct. Walking along chromosomes with super-resolution imaging, contact maps, and integrative modeling. PLOS Genetics. 2018 Dec 26; vol. 14, no. 12. Publisher: Public Library of Science:e1007872

197. Cardozo Gizzi A, Cattoni D, Fiche JB, Espinola S, Gurgo J, Messina O, Houbron C, Ogiyama Y, Papadopoulos G, Cavalli G, Lagha M, and Nollmann M. Microscopy-Based Chromosome Conformation Capture Enables Simultaneous Visualization of Genome Organization and Transcription in Intact Organisms. Molecular Cell. 2019; vol. 74, no. 1 :212–222.e5

198. Mateo LJ, Murphy SE, Hafner A, Cinquini IS, Walker CA, and Boettiger AN. Visualizing DNA folding and RNA in embryos at single-cell resolution. Nature. 2019 Apr; vol. 568, no. 7750. Number: 7750 Publisher: Nature Publishing Group:49–54

199. Liu M, Lu Y, Yang B, Chen Y, Radda J, Hu M, Katz S, and Wang S. Multiplexed imaging of nucleome architectures in single cells of mammalian tissue. Nature Communications. 2020; vol. 11, no. 1

200. Liu M, Yang B, Hu M, Radda JSD, Chen Y, Jin S, Cheng Y, and Wang S. Chromatin tracing and multiplexed imaging of nucleome architectures (MINA) and RNAs in single mammalian cells and tissue. Nature Protocols. 2021 May; vol. 16, no. 5. Number: 5 Publisher: Nature Publishing Group:2667–97

201. Su JH, Zheng P, Kinrot S, Bintu B, and Zhuang X. Genome-Scale Imaging of the 3D Organization and Transcriptional Activity of Chromatin. Cell. 2020; vol. 182, no. 6 :1641–1659.e26

202. Lubeck E, Coskun AF, Zhiyentayev T, Ahmad M, and Cai L. Single-cell in situ RNA profiling by sequential hybridization. Nature Methods. 2014 Apr; vol. 11, no. 4. Number: 4 Publisher: Nature Publishing Group:360–1

203. Eng CHL, Lawson M, Zhu Q, Dries R, Koulena N, Takei Y, Yun J, Cronin C, Karp C, Yuan GC, and Cai L. Transcriptome-scale super-resolved imaging in tissues by RNA seqFISH+. Nature. 2019 Apr; vol. 568, no. 7751. Number: 7751 Publisher: Nature Publishing Group:235–9

204. Stevens TJ, Lando D, Basu S, Atkinson LP, Cao Y, Lee SF, Leeb M, Wohlfahrt KJ, Boucher W, O'Shaughnessy-Kirwan A, Cramard J, Faure AJ, Ralser M, Blanco E, Morey L, Sansó M, Palayret MGS, Lehner B, Di Croce L, Wutz A, Hendrich B, Klenerman D, and Laue ED. 3D structures of individual mammalian genomes studied by single-cell Hi-C. Nature. 2017 Apr; vol. 544, no. 7648. Number: 7648 Publisher: Nature Publishing Group:59–64

205. Gu B, Swigut T, Spencley A, Bauer MR, Chung M, Meyer T, and Wysocka J. Transcription-coupled changes in nuclear mobility of mammalian cis-regulatory elements. Science. 2018 Mar 2; vol. 359, no. 6379. Publisher: American Association for the Advancement of Science:1050–5

206. Oudelaar AM, Davies JO, Hanssen LL, Telenius JM, Schwessinger R, Liu Y, Brown JM, Downes DJ, Chiariello AM, Bianco S, Nicodemi M, Buckle VJ, Dekker J, Higgs DR, and Hughes JR. Single-allele chromatin interactions identify regulatory hubs in dynamic compartmentalized domains. Nature genetics. 2018 Dec; vol. 50, no. 12 :1744–51

207. Olivares-Chauvet P, Mukamel Z, Lifshitz A, Schwartzman O, Elkayam NO, Lubling Y, Deikus G, Sebra RP, and Tanay A. Capturing pairwise and multi-way chromosomal conformations using chromosomal walks. Nature. 2016 Dec; vol. 540, no. 7632. Number: 7632 Publisher: Nature Publishing Group:296–300

208. Allahyar A, Vermeulen C, Bouwman BAM, Krijger PHL, Verstegen MJAM, Geeven G, Kranenburg M van, Pieterse M, Straver R, Haarhuis JHI, Jalink K, Teunissen H, Renkens IJ, Kloosterman WP, Rowland BD, Wit E de, Ridder J de, and Laat W de. Enhancer hubs and loop collisions identified from single-allele topologies. Nature Genetics. 2018 Aug; vol. 50, no. 8. Number: 8 Publisher: Nature Publishing Group:1151–60

209. Belton JM, McCord RP, Gibcus J, Naumova N, Zhan Y, and Dekker J. Hi-C: A comprehensive technique to capture the conformation of genomes. Methods (San Diego, Calif). 2012 Nov; vol. 58, no. 3 :10.1016/j.ymeth.2012.05.001

210. Arima-HiC Kit: User Guide for Mammalian Cell Lines. 2019 Nov

211. Garg S, Fungtammasan A, Carroll A, Chou M, Schmitt A, Zhou X, Mac S, Peluso P, Hatas E, Ghurye J, Maguire J, Mahmoud M, Cheng H, Heller D, Zook JM, Moemke T, Marschall T, Sedlazeck FJ, Aach J, Chin CS, Church GM, and Li H. Chromosome-scale, haplotype-resolved assembly of human genomes. Nature Biotechnology. 2021 Mar; vol. 39, no. 3. Number: 3 Publisher: Nature Publishing Group:309–12

212. Imakaev M, Fudenberg G, McCord RP, Naumova N, Goloborodko A, Lajoie BR, Dekker J, and Mirny LA. Iterative correction of Hi-C data reveals hallmarks of chromosome organization. Nature Methods. 2012 Oct; vol. 9, no. 10. Number: 10 Publisher: Nature Publishing Group:999–1003

213. Servant N, Varoquaux N, Lajoie BR, Viara E, Chen CJ, Vert JP, Heard E, Dekker J, and Barillot E. HiC-Pro: an optimized and flexible pipeline for Hi-C data processing. Genome Biology. 2015 Dec 1; vol. 16, no. 1 :259

214. Zhang H, Song L, Wang X, Cheng H, Wang C, Meyer CA, Liu T, Tang M, Aluru S, Yue F, Liu XS, and Li H. Fast alignment and preprocessing of chromatin profiles with Chromap. Nature Communications. 2021 Nov 12; vol. 12, no. 1. Number: 1 Publisher: Nature Publishing Group:6566

215. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R, and 1000 Genome Project Data Processing Subgroup. The Sequence Alignment/Map format and SAMtools. Bioinformatics (Oxford, England). 2009 Aug 15; vol. 25, no. 16 :2078–9

216. Docker for 4DN Hi-C processing pipeline. original-date: 2017-03-03T05:30:33Z. 2023 Sep 14

217. Goloborodko A, Venev S, Spracklin G, Abdennur N, agalitsyna, Shaytan A, Flyamer I, Tommaso PD, and sergey-kolchenko. open2c/distiller-nf: v0.3.4. Version v0.3.4. 2022 Nov

218. ENCODE. ENCODE Hi-C uniform processing pipeline. original-date: 2018-02-01T23:47:50Z. 2023 Sep 28

219. Servant N, c bot nf-core, Ewels P, Garcia MU, Talbot A, Peltzer A, Miller E, Rossini R, Zohren J, Menden K, and Philippe LR. nf-core/hic: nf-core/hic v2.1.0. Version 2.1.0. 2023 Jun

220. Abdennur N and Mirny LA. Cooler: scalable storage for Hi-C data and other genomically labeled arrays. Bioinformatics. 2019 Jul 10; vol. 36, no. 1 :311–6

221. Wolff J, Bhardwaj V, Nothjunge S, Richard G, Renschler G, Gilsbach R, Manke T, Backofen R, Ramírez F, and Grüning BA. Galaxy HiCExplorer: a web server for reproducible Hi-C data analysis, quality control and visualization. Nucleic Acids Research. 2018 Jul 2; vol. 46 :W11–W16

222. Wolff J, Rabbani L, Gilsbach R, Richard G, Manke T, Backofen R, and Grüning BA. Galaxy HiCExplorer 3: a web server for reproducible Hi-C, capture Hi-C and single-cell Hi-C data analysis, quality control and visualization. Nucleic Acids Research. 2020 Jul 2; vol. 48 :W177–W184

223. Lee S, Bakker CR, Vitzthum C, Alver BH, and Park PJ. Pairs and Pairix: a file format and a tool for efficient storage and retrieval for Hi-C read pairs. Bioinformatics. 2022 Mar 4; vol. 38, no. 6 :1729–31

224. Li H. Tabix: fast retrieval of sequence features from generic TAB-delimited files. Bioinformatics. 2011 Mar 1; vol. 27, no. 5 :718–9

225. Open2C, Abdennur N, Fudenberg G, Flyamer IM, Galitsyna AA, Goloborodko A, Imakaev M, and Venev SV. Pairtools: from sequencing data to chromosome contacts. bioRxiv: The Preprint Server for Biology. 2023 Feb 15 :2023.02.13.528389

226. Soohyun Lee. pairix: 1D/2D indexing and querying on bgzipped text file with a pair of genomic coordinates. original-date: 2016-10-28T18:57:34Z. 2023 Oct 5

227. Lun ATL, Perry M, and Ing-Simmons E. Infrastructure for genomic interactions: Bioconductor classes for Hi-C, ChIA-PET and related experiments. F1000Research. 2016; vol. 5 :950

228. The HDF Group. Hierarchical Data Format, version 5. 1997

229. Durand NC, Robinson JT, Shamim MS, Machol I, Mesirov JP, Lander ES, and Aiden EL. Juicebox Provides a Visualization System for Hi-C Contact Maps with Unlimited Zoom. Cell Systems. 2016 Jul 27; vol. 3, no. 1 :99–101

230. .hic format specification. original-date: 2020-07-08T13:32:40Z. 2022 Mar 17

231. Hansen AS, Pustova I, Cattoglio C, Tjian R, and Darzacq X. CTCF and cohesin regulate chromatin loop stability with distinct dynamics. eLife. 2017 May 3; vol. 6. Ed. by Sherratt D. Publisher: eLife Sciences Publications, Ltd:e25776

232. Rodríguez-Carballo E, Lopez-Delisle L, Zhan Y, Fabre PJ, Beccari L, El-Idrissi I, Huynh THN, Ozadam H, Dekker J, and Duboule D. The HoxD cluster is a dynamic and resilient TAD boundary controlling the segregation of antagonistic regulatory landscapes. Genes & Development. 2017 Nov 15; vol. 31, no. 22. Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution: Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab:2264–81

233. Rodríguez-Carballo E, Lopez-Delisle L, Willemin A, Beccari L, Gitto S, Mascrez B, and Duboule D. Chromatin topology and the timing of enhancer function at the HoxD locus. Proceedings of the National Academy of Sciences. 2020 Dec 8; vol. 117, no. 49. Publisher: Proceedings of the National Academy of Sciences:31231–41

234. Bailey TL and Gribskov M. Combining evidence using p-values: application to sequence homology searches. Bioinformatics (Oxford, England). 1998; vol. 14, no. 1 :48–54

235. Kim T, Han S, Chun Y, Yang H, Min H, Jeon SY, Kim Ji, Moon HG, and Lee D. Comparative characterization of 3D chromatin organization in triple-negative breast cancers. Experimental & Molecular Medicine. 2022 May; vol. 54, no. 5. Number: 5 Publisher: Nature Publishing Group:585–600

236. Wang X, Yan J, Ye Z, Zhang Z, Wang S, Hao S, Shen B, and Wei G. Reorganization of 3D chromatin architecture in doxorubicin-resistant breast cancer cells. Frontiers in Cell and Developmental Biology. 2022; vol. 10 :974750

237. Servant N, Varoquaux N, Heard E, Barillot E, and Vert JP. Effective normalization for copy number variation in Hi-C data. BMC Bioinformatics. 2018 Sep 6; vol. 19, no. 1 :313

238. Noordermeer D, Leleu M, Splinter E, Rougemont J, De Laat W, and Duboule D. The Dynamic Architecture of Hox Gene Clusters. Science. 2011 Oct 14; vol. 334, no. 6053. Publisher: American Association for the Advancement of Science:222–5

239. Lupiáñez DG, Kraft K, Heinrich V, Krawitz P, Brancati F, Klopocki E, Horn D, Kayserili H, Opitz JM, Laxova R, Santos-Simarro F, Gilbert-Dussardier B, Wittler L, Borschiwer M, Haas SA, Osterwalder M, Franke M, Timmermann B, Hecht J, Spielmann M, Visel A, and Mundlos S. Disruptions of Topological Chromatin Domains Cause Pathogenic Rewiring of Gene-Enhancer Interactions. Cell. 2015 May 21; vol. 161, no. 5 :1012–25

240. Giorgio E, Robyr D, Spielmann M, Ferrero E, Di Gregorio E, Imperiale D, Vaula G, Stamoulis G, Santoni F, Atzori C, Gasparini L, Ferrera D, Canale C, Guipponi M, Pennacchio LA, Antonarakis SE, Brussino A, and Brusco A. A large genomic deletion leads to enhancer adoption by the lamin B1 gene: a second path to autosomal dominant adult-onset demyelinating leukodystrophy (ADLD). Human Molecular Genetics. 2015 Jun 1; vol. 24, no. 11 :3143–54

241. Flavahan WA, Drier Y, Liau BB, Gillespie SM, Venteicher AS, Stemmer-Rachamimov AO, Suvà ML, and Bernstein BE. Insulator dysfunction and oncogene activation in IDH mutant gliomas. Nature. 2016 Jan; vol. 529, no. 7584. Number: 7584 Publisher: Nature Publishing Group:110–4

242. Hnisz D, Weintraub AS, Day DS, Valton AL, Bak RO, Li CH, Goldmann J, Lajoie BR, Fan ZP, Sigova AA, Reddy J, Borges-Rivera D, Lee TI, Jaenisch R, Porteus MH, Dekker J, and Young RA. Activation of proto-oncogenes by disruption of chromosome neighborhoods. Science. 2016 Mar 25; vol. 351, no. 6280. Publisher: American Association for the Advancement of Science:1454–8

243. Thiecke MJ, Wutz G, Muhar M, Tang W, Bevan S, Malysheva V, Stocsits R, Neumann T, Zuber J, Fraser P, Schoenfelder S, Peters JM, and Spivakov M. Cohesin-Dependent and -Independent Mechanisms Mediate Chromosomal Contacts between Promoters and Enhancers. Cell Reports. 2020 Jul 21; vol. 32, no. 3 :107929

244. Valton AL and Dekker J. TAD disruption as oncogenic driver. Current opinion in genetics & development. 2016 Feb; vol. 36 :34–40

245. Taberlay PC, Achinger-Kawecka J, Lun ATL, Buske FA, Sabir K, Gould CM, Zotenko E, Bert SA, Giles KA, Bauer DC, Smyth GK, Stirzaker C, O'Donoghue SI, and Clark SJ. Three-dimensional disorganization of the cancer genome occurs coincident with long-range genetic and epigenetic alterations. Genome Research. 2016 Jun 1; vol. 26, no. 6. Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution: Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab:719–31

246. Luo H, Wang F, Zha J, Li H, Yan B, Du Q, Yang F, Sobh A, Vulpe C, Drusbosky L, Cogle C, Chepelev I, Xu B, Nimer SD, Licht J, Qiu Y, Chen B, Xu M, and Huang S. CTCF boundary remodels chromatin domain and drives aberrant HOX gene transcription in acute myeloid leukemia. Blood. 2018 Aug 23; vol. 132, no. 8 :837–48

247. Campbell MJ. Tales from Topographic Oceans: Topologically associated domains and cancer. Endocrine-related cancer. 2019 Nov; vol. 26, no. 11 :R611–R626

248. San Martin R, Das P, Dos Reis Marques R, Xu Y, Roberts JM, Sanders JT, Golloshi R, and McCord RP. Chromosome compartmentalization alterations in prostate cancer cell lines model disease progression. Journal of Cell Biology. 2021 Dec 10; vol. 221, no. 2 :e202104108

249. Ren B, Yang J, Wang C, Yang G, Wang H, Chen Y, Xu R, Fan X, You L, Zhang T, and Zhao Y. High-resolution Hi-C maps highlight multiscale 3D epigenome reprogramming during pancreatic cancer metastasis. Journal of Hematology & Oncology. 2021 Aug 4; vol. 14, no. 1 :120

250. Baker M. 1,500 scientists lift the lid on reproducibility. Nature. 2016 May 1; vol. 533, no. 7604. Number: 7604 Publisher: Nature Publishing Group:452–4

251. Miyakawa T. No raw data, no science: another possible source of the reproducibility crisis. Molecular Brain. 2020 Feb 21; vol. 13, no. 1 :24

252. Gosselin RD. Statistical Analysis Must Improve to Address the Reproducibility Crisis: The ACcess to Transparent Statistics (ACTS) Call to Action. BioEssays. 2020; vol. 42, no. 1. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/bies.201900189:1900189

253. Gundersen OE. The fundamental principles of reproducibility. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences. 2021 Mar 29; vol. 379, no. 2197. Publisher: Royal Society:20200210

254. Moonesinghe R, Khoury MJ, and Janssens ACJW. Most Published Research Findings Are False—But a Little Replication Goes a Long Way. PLoS Medicine. 2007 Feb; vol. 4, no. 2 :e28

255. Freedman LP, Cockburn IM, and Simcoe TS. The Economics of Reproducibility in Preclinical Research. PLOS Biology. 2015 Jun 9; vol. 13, no. 6. Publisher: Public Library of Science:e1002165

256. Genome Biology - reporting standards. BioMed Central. Available from: https://genomebiology.biomedcentral.com/submission-guidelines/preparing-your-manuscript/software [Accessed on: 2023 Nov 21]

257. Oxford Bioinformatics: Instructions for Authors. Oxford Academic. Available from: https://academic.oup.com/bioinformatics/pages/instructions_for_authors [Accessed on: 2023 Nov 21]

258. zlib Home Site. Available from: https://www.zlib.net/ [Accessed on: 2023 Nov 9]

259. LZ4 - Extremely fast compression. Available from: https://lz4.org/ [Accessed on: 2023 Nov 9]

260. XZ Utils. Available from: https://tukaani.org/xz/ [Accessed on: 2023 Nov 9]

261. Apache Parquet. Available from: https://parquet.apache.org/ [Accessed on: 2023 Oct 28]

262. Di Tommaso P, Chatzou M, Floden EW, Barja PP, Palumbo E, and Notredame C. Nextflow enables reproducible computational workflows. Nature Biotechnology. 2017 Apr; vol. 35, no. 4. Number: 4 Publisher: Nature Publishing Group:316–9

263. Mölder F, Jablonski KP, Letcher B, Hall MB, Tomkins-Tinch CH, Sochat V, Forster J, Lee S, Twardziok SO, Kanitz A, Wilm A, Holtgrewe M, Rahmann S, Nahnsen S, and Köster J. Sustainable data analysis with Snakemake. 10:33. Type: article. F1000Research, 2021 Jan 18

Papers

Paper I

MoDLE: high-performance stochastic modeling of DNA loop extrusion interactions

Roberto Rossini, Vipin Kumar, Anthony Mathelier, Torbjørn Rognes, Jonas Paulsen

Published in *Genome Biology*, 2022 Nov 30;23(1):247
DOI: 10.1186/s13059-022-02815-7.

SOFTWARE

Open Access

MoDLE: high-performance stochastic modeling of DNA loop extrusion interactions

Roberto Rossini[1], Vipin Kumar[2], Anthony Mathelier[2], Torbjørn Rognes[3,4] and Jonas Paulsen[1,3*]

*Correspondence: jonas.paulsen@ibv.uio.no

[1] Department of Biosciences, University of Oslo, 0316 Oslo, Norway
[2] Centre for Molecular Medicine Norway (NCMM), Nordic EMBL Partnership, University of Oslo, 0318 Oslo, Norway
[3] Centre for Bioinformatics, Department of Informatics, University of Oslo, 0316 Oslo, Norway
[4] Department of Microbiology, Oslo University Hospital, Rikshospitalet, 0424 Oslo, Norway

Abstract

DNA loop extrusion emerges as a key process establishing genome structure and function. We introduce MoDLE, a computational tool for fast, stochastic modeling of molecular contacts from DNA loop extrusion capable of simulating realistic contact patterns genome wide in a few minutes. MoDLE accurately simulates contact maps in concordance with existing molecular dynamics approaches and with Micro-C data and does so orders of magnitude faster than existing approaches. MoDLE runs efficiently on machines ranging from laptops to high performance computing clusters and opens up for exploratory and predictive modeling of 3D genome structure in a wide range of settings.

Keywords: Loop extrusion, Stochastic modeling, Hi-C, Micro-C, TAD

Background

DNA loop-extrusion, in which DNA is progressively reeled into transient loops, emerges as a key process in genome structure and function. The growing list of cellular processes where loop-extrusion plays a critical role now includes transcriptional regulation [1, 2], DNA repair [3], VDJ-recombination [4], and cell division [5]. Recent single-molecule imaging experiments have provided direct observations of loop extrusion in vitro [6, 7].

High-throughput chromosome conformation capture sequencing, including Hi-C [8] and Micro-C [9, 10], has advanced our abilities to map three-dimensional (3D) genome organization through quantification of spatially proximal genome regions. The resulting data is usually rendered as a matrix of intrachromosomal and interchromosomal contact frequencies. These data increasingly deepen our understanding of 3D genome organization and show DNA loop extrusion as a key process shaping genome structure [11–13]. In fact, topologically associating domains (TADs), which show up as sub-megabase-sized domains covering most of higher eukaryote genomes, are formed by loop extrusion [12]. TADs are relevant units of gene expression regulation and are associated with disease when disrupted [14].

DNA loop extrusion is carried out by ring-shaped proteins (including cohesin and condensin) belonging to the structural maintenance of chromosomes (SMC) family. These proteins are often referred to as loop extrusion factors (LEFs) [15]. The exact mechanism of how loop extrusion takes place in interphase is not fully understood. There is, however, convincing evidence that SMCs bind DNA to perform ATP-dependent loop extrusion in a symmetric or asymmetric fashion. Recent evidence suggests that cohesin can extrude DNA with a "swing-and-clamp" mechanism [16] and in a nontopological configuration where DNA is not encircled by the cohesin ring [17, 18]. A loop starts extruding when a LEF binds to a genomic region and continues processively until it is stalled by a DNA-bound CCCTC binding factor (CTCF) oriented with its N-terminus pointing towards the extruding cohesin complex. A pair of CTCFs arranged in a convergent orientation can thus stall loop growth on both sides creating semi-stable loops visible in Hi-C as a characteristic "dot" at TAD corners [19]. Similarly, when extruding loops are stalled only on one side, a "stripe" can be observed along one or both TAD borders [20]. The protein WAPL transiently releases cohesin from chromatin, terminating the loop extrusion process [21, 22]. The resulting loop-extrusion patterns have been found in a range of Hi-C datasets so far, emphasizing the evolutionary conserved role of loop extrusion in shaping 3D genome organization [19, 23].

Disrupting any of the key proteins involved in DNA loop extrusion has a dramatic effect on genome 3D structure. WAPL depletion causes an increase in loop stability, with an accumulation of axial elements and weakening of compartments [21, 24]. Depletion of cohesin causes a large fraction of TADs and loops to disappear [24–26]. Similarly, depletion of CTCF induces loss of loops and TADs [24, 27].

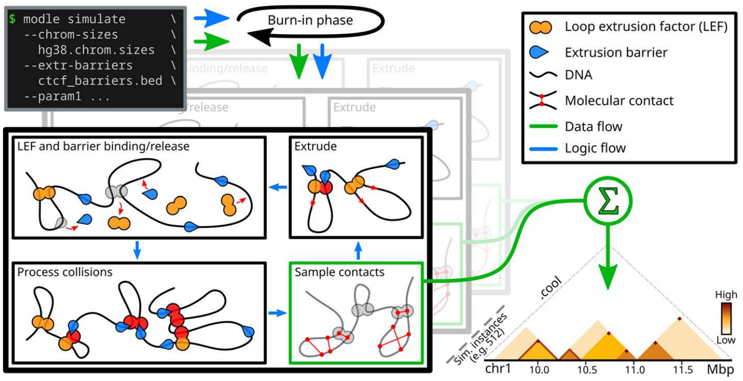
Modeling and simulation of DNA-DNA contact patterns is a powerful approach for understanding underlying molecular mechanisms and for predicting the effect of DNA perturbations. Polymer simulations and molecular dynamics (MD) have been used for modeling of TADs to study their structure and dynamics [28–31]. Computational modeling and simulation of loop extrusion has proven useful for predicting the effects of perturbations to TAD borders and to properly understand patterns seen in Hi-C data. Initial models [15, 32] of loop extrusion used the Gillespie algorithm to characterize looping properties and chromatin compaction and did not sample contact maps. Subsequent models used HOOMD particle simulation [33] to perform homopolymer simulations where modeled LEFs extrude the polymers and halt at boundaries with properties defined from CTCF motif instance orientation and ChIP-seq signal strength [11, 25]. Recently, to efficiently simulate larger genome regions, a combination of one-dimensional (1D) simulations with 3D polymer modeling has been applied to sample multiple conformations combined into contact maps. LEF binding, release, and stalling probabilities are then modeled explicitly [34–36]. These simulations are typically implemented using the OpenMM molecular simulation framework [37]. The simulations can be used to explore and rule out molecular mechanisms. For example, Banigan et al. assessed the level of DNA compaction that can be achieved by different loop extrusion mechanisms and concluded that one-sided loop extrusion alone fails to achieve the level of compaction observed in large metazoan genomes [36]. Other approaches embed epigenetic data in combination with crosslinking proteins to model and study conformational variability across complex chromatin regions [38, 39]. To the best of our knowledge, no standalone software for modeling and simulation of loop extrusion exists.

We introduce MoDLE (modeling of DNA loop extrusion), a high-performance stochastic model of DNA loop extrusion capable of efficiently simulating contacts from loop extrusion genome wide. In contrast to MD simulation approaches, simulating loop extrusion contacts using MoDLE is a straightforward process only requiring two input files and execution through a command line interface (CLI). MoDLE can simulate a contact matrix with the molecular interactions generated by DNA loop extrusion on the entire human genome in a matter of minutes using less than 1 GB of RAM. Typical use cases include predicting Hi-C contact patterns from ChIP-seq (or similar) data and predicting the effect of alterations, mutations, and structural variation to TAD borders. MoDLE opens up for rapid simulation and parameter exploration of DNA loop extrusion on genomes of any size, including large mammalian genomes.

Results

MoDLE: modeling of DNA loop extrusion

MoDLE uses fast stochastic simulation to sample DNA-DNA contacts generated by loop extrusion. Binding and release of LEFs and barriers and the extrusion process is modeled as an iterative process (see Fig. 1). At the beginning of a simulation, MoDLE goes through a burn-in phase where LEFs are progressively bound to DNA, without sampling molecular contacts. The burn-in phase runs until the average loop size has stabilized. Active LEFs are extruded through randomly sampled strides along the DNA in reverse and forward directions. Each epoch, LEFs are released with a probability based on the average LEF processivity and extrusion speed. LEFs that are released in the current epoch will rebind to randomly sampled DNA regions in the next epoch. Extrusion barriers (e.g., CTCF binding sites) are modeled using a two-state (bound and unbound)



```
$ modle simulate        \
  --chrom-sizes         \
    hg38.chrom.sizes    \
  --extr-barriers       \
    ctcf_barriers.bed   \
  --param1 ...
```

Fig. 1 Schematic and simplified overview of MoDLE. Input files specify genome regions to be simulated (e.g., a chrom.sizes file) and their barrier positions (e.g., CTCF binding sites and orientation) in BED format. Optional parameters control the specifics of a simulation. Loop extruding factors (LEFs) bind to, extrude, and release from the regions and interact with modeled barriers according to input parameters. Loop extrusion and intra-TAD contacts of a randomized subset of loops are recorded each epoch and aggregated into an output cooler file containing the final simulated contact frequencies. Simulation halts when a target number of epochs or a target number of loop extrusion contacts have been simulated

Markov process. Each extrusion barrier consists of a position, a blocking direction and the Markov process transition probabilities. The occupancy of each extrusion barrier can be specified individually through the score field in the input BED file. Alternatively, users can specify a uniform barrier occupancy that is applied to all extrusion barriers. MoDLE accepts a large number of optional parameters to specify the model's behavior. For example, users can specify the number of LEFs to be instantiated for each Mbp of simulated DNA using the --lef-density parameter. LEF-barrier and LEF-LEF collisions are processed each simulation epoch. Collision information is used to update candidate strides to satisfy the constraints imposed by collision events and to compute how extrusion in the next epoch should proceed.

During a simulation, sampled molecular contacts are accumulated into a specialized contact matrix data structure with low memory overhead. MoDLE execution continues until a target number of epochs or a target number of loop extrusion contacts are simulated. Finally, contacts generated by all simulation instances for a given chromosome are written to an output file in cooler format [40] (Fig. 1).

With default settings, MoDLE will run over 500 simulation instances for each chromosome simulated. Thus, simulation instances can run in parallel, making efficient use of the computational resources of modern multi-core CPUs. We designed MoDLE such that each simulation instance requires less than 10 MB of memory to simulate loop extrusion on large mammalian chromosomes, such as chromosome 1 from the human genome. To achieve high-performance, MoDLE stores most of its data in contiguous memory. Data is indexed such that extrusion barriers and extrusion units in a simulation instance can be efficiently traversed in 5′-3′ and 3′-5′ directions. This allows MoDLE to bind/release LEFs, process collisions, register contacts, and extrude DNA in linear time-complexity.

More design and implementation details are available in Additional file 1 as well as MoDLE's GitHub repository github.com/paulsengroup/modle.

Comparison with Micro-C data and MD simulations

To assess MoDLE's ability to reproduce contact data features known to be stemming from loop extrusion, we simulated genome-wide DNA-DNA contacts based on available CTCF and RAD21 ChIP-seq data in H1-hESC cells (see Methods). MoDLE is capable of simulating loop extrusion molecular contacts and intra-TAD contacts separately (see Additional file 1: Section 9 for details). A rendering of the resulting loop extrusion molecular contacts heatmaps show characteristic stripe and dot patterns at TAD borders (Fig. 2A). Simulated TAD contacts show enrichment of contacts within TADs, including a nested structure of the TADs (Fig. 2B). In combination, these patterns resemble well-characterized patterns observed in Micro-C and Hi-C data (Fig. 2C).

Even though no stand-alone software exists for direct side-by-side comparison, we adapted available code based on OpenMM [36] to systematically compare the output with that of MoDLE (see Methods). We chose OpenMM for comparison as it is an efficient and widely used system for simulating loop extrusion [12, 34–36, 41].

Using the same input data, we simulated contacts in five different 10 Mbp regions on five different chromosomes. In general, MoDLE produces contact patterns similar to OpenMM (Fig. 2D and Additional file 2: Fig. S1), and MoDLE output and OpenMM

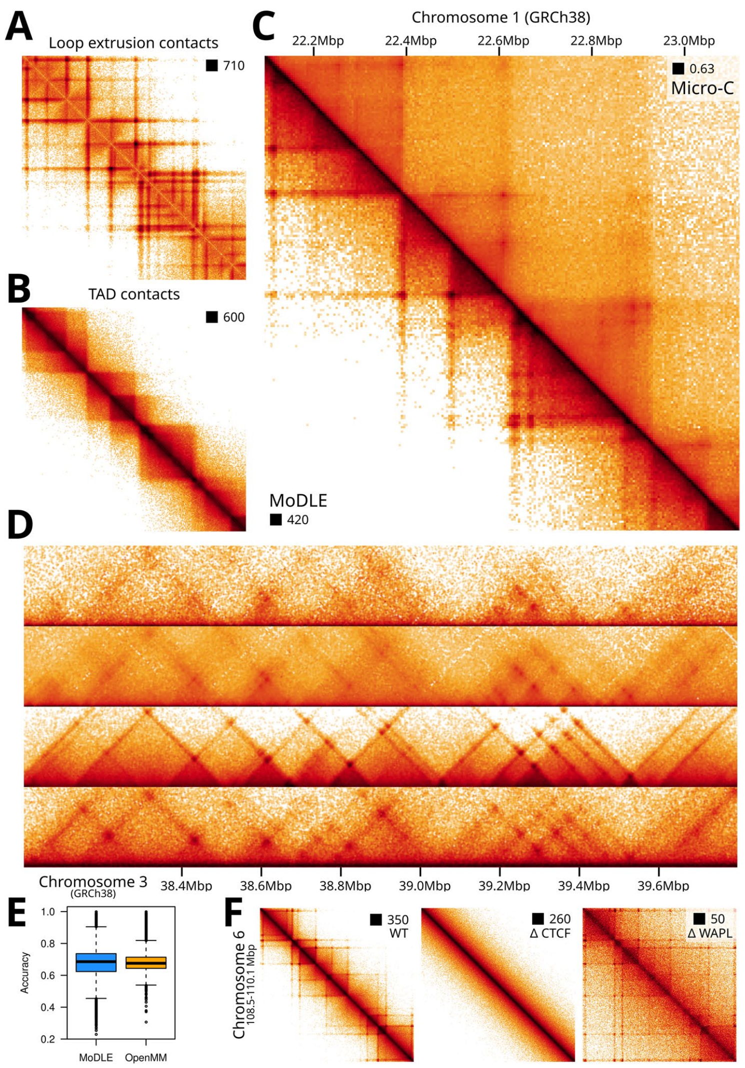


Fig. 2 Comparison of MoDLE with OpenMM and Micro-C data. **A** Simulated MoDLE contact frequencies solely mediated by LEFs. **B** Intra-TAD contacts (only) generated with MoDLE. **C** Lower triangle: Loop extrusion and intra-TAD contacts from MoDLE in the same region as for A and B. Upper triangle: Micro-C data from the same region. **D** Side-by-side comparison of Micro-C data, MoDLE output, and OpenMM output for a region on chromosome 3 in H1-hESC. **E** Quantitative comparison of the accuracy (fraction of correctly classified pixels relative to all pixels) of MoDLE and OpenMM in reproducing stripe and dot pixel-patterns observed in modeled regions in H1-hESC cells (see the "Methods" section). **F** In silico simulated molecular contacts mimicking CTCF and WAPL depletion. Left: Wildtype (WT) output of MoDLE in a region on chromosome 6 in H1-hESC. Middle: effect on MoDLE output when CTCF barriers weakly associate with their binding sites. Right: effect on MoDLE output when LEFs are less likely to be released from DNA, thus mimicking WAPL depletion

correlate strongly (Pearson $\rho = 0.93$; see Additional file 2: Fig. S2). By comparing contacts with corresponding Micro-C and Hi-C data (Fig. 2D), we see a median pixel accuracy (i.e., the ability to correctly classify pixels as a dot/stripe or not, relative to all pixels; see Methods) of 0.69 for MoDLE and 0.68 for OpenMM, signifying that MoDLE indeed simulates contacts observed in Micro-C similar to OpenMM (Fig. 2E). Note that contacts generated by OpenMM involve 3D polymer modeling and thus, unlike MoDLE, considers random polymer contacts. As a consequence, contacts not generated by loop-extrusion will be included in the OpenMM output. Therefore, long-range contacts (~2–3 Mbp) are generally not as enriched in the MoDLE output as these contacts are mainly compartmental or dominated by random polymer interactions. This can be seen when employing a diagonal-by-diagonal correlation between MoDLE and OpenMM, which shows that the two methods correlate better at short range contacts than at long range contacts (see Additional file 2: Fig. S3). It implies that MoDLE does not by default recapitulate the relationship between the distance from the diagonal and the contact frequencies as seen in Hi-C or Micro-C data. However, when LEF processivity is increased, this trend is gradually approached (see Additional file 2: Fig. S4). Comparing the output of MoDLE and OpenMM in A and B compartments separately shows minimal difference of performance between compartments (Additional file 2: Fig. S5).

Altering MoDLE's input parameters to in silico mimicking depletion of CTCF and WAPL shows an expected loss of TAD insulation patterns [27] upon in silico depletion of CTCF and more pronounced stripe and dot patterns [22] when mimicking WAPL depletion (Fig. 2F). Similarly, altering the parameters specifying LEF density, LEF processivity and LEF-LEF collisions shows relevant and predictable consequences in the data output (see Additional file 2: Fig. S6-10). We conclude that MoDLE is capable of simulating loop extrusion and TAD contact patterns similar to existing state-of-the-art molecular dynamics (OpenMM) approaches.

Benchmarking of execution time and memory usage

MoDLE is designed for fast genome-wide simulation of loop extrusion contact patterns. A genome-wide run with default settings, simulating loop extrusion on the entire human genome using barriers from H1-hESC (38,815 CTCF barriers and 61,766 LEFs; see Methods) takes ~40 s on a compute server (server A; see Table 1) and ~5 min on a laptop (laptop A; see Table 1), generating over 370 million contacts. To systematically compare MoDLE execution time and memory usage with OpenMM, we generated synthetic input datasets with increasing genome size (1–500 Mbp) and number of CTCF barriers (4 barriers per Mbp of DNA simulated) (see the "Methods" section for details). The inputs were identical in MoDLE and OpenMM. Each measurement was repeated 10 times for MoDLE and 5 times for OpenMM. For MoDLE, we run benchmarks using 1–52 CPU cores, while for OpenMM, we tested the CPU (server C; see Table 1) and GPU (server D; see Table 1) implementations. We computed median elapsed wall clock time and peak memory usage for MoDLE and OpenMM. The resulting comparisons show that MoDLE simulations using 52 CPU cores complete within 0.7–71 s from the smallest to the largest genome region. OpenMM requires 2 h and 35 min for the smallest genome region and over 41 h for a genomic region of 250 Mbp (Fig. 3A). Due to very long execution times, OpenMM

Table 1 Hardware specifications of computational resources used for simulation and benchmarking

Identifier	CPU model	System memory	Operating system	Accelerator (GPU)
Laptop A	Intel Core i9-9880H(8 cores)	64 GB (4x 16 GB, DDR4 UDIMM 2667 MT/s dual-channel)	Arch Linux (Linux v5.17.1)	NVIDIA Quadro RTX 4000 (8 GB)
Server A	2x AMD EPYC 7742 (2x 64 cores)	2048 GB (32x 64 GB, RDIMM DDR4 2933 MT/s eight-channel)	RHEL 8.5 (Linux v4.18.0-305)	N/A
Server B	2x Intel Xeon Gold 6138 (2x 20 cores)	192 GB (12x 16 GB, RDIMM DDR4 2666 MT/s six-channel)	RHEL 7.9.2009 (Linux v3.10.0-1160.6.1)	
Server C	2x Intel Xeon Gold 6230R (2x 26 cores)	192 GB (12x 16 GB, RDIMM DDR4 2933 MT/s six-channel)		
Server D	2x Intel Xeon Gold 6126 (2x 12 cores)	384 GB (24x 16 GB, RDIMM DDR4 2666 MT/s six-channel)		4x NVIDIA Tesla P100 (16 GB)

runs above 250 Mbp were not performed. For the compared genome regions, MoDLE is 4000–5000 times faster than OpenMM (Fig. 3A). OpenMM simulations without GPU acceleration were particularly slow and were only used to simulate genome regions below 5 Mbp and required up to 35 h and 20 min of execution time (Fig. 3A). Thus, in practice running OpenMM requires access to GPUs, while MoDLE runs efficiently using CPUs.

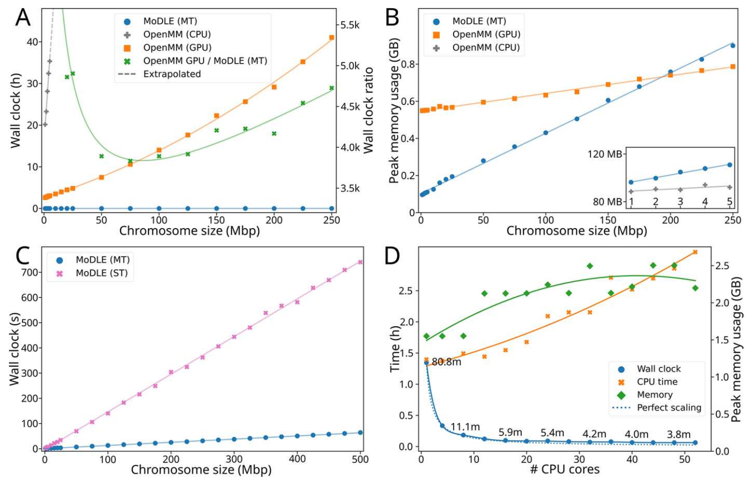


Fig. 3 Benchmarking MoDLE and OpenMM. **A** Median memory usage (in MBs) of MoDLE with multithreading (blue) compared to OpenMM with GPU (orange) for chromosome regions ranging in size from 1 to 250 Mbp. Inset shows comparison between MoDLE (blue) OpenMM with CPU (gray) for chromosome regions ranging in size from 1 to 5 Mbp. **B** Median elapsed execution time (hours) of MoDLE with multithreading (blue), OpenMM with CPU (gray), OpenMM with GPU (orange), and the ratio of OpenMM (GPU) to MoDLE. Dotted lines are extrapolated. **C** Comparison of the median elapsed execution time (seconds) of MoDLE with (blue) and without (pink) multithreading for chromosome regions ranging in size from 1 to 500 Mbp. **D** Comparison of median elapsed execution time (hours) of MoDLE utilizing from 1 to 52 CPU cores. Blue line shows elapsed wall clock time (hours), whereas the orange line shows the CPU time (hours). The dotted line illustrates the corresponding theoretical perfect scaling of the executing time. Green line shows median peak memory usage (right axis; MB)

Comparing peak memory usage, MoDLE uses less memory than OpenMM for regions smaller than 200 Mbp and requires more memory for larger systems. Nevertheless, memory usage of both MoDLE and OpenMM scales linearly for increasing genome region sizes and is for all practical purposes within reasonable limits on today's computers regardless of genome size (Fig. 3A, B).

Multithreading efficiently reduces MoDLE's execution time for increasingly large genome sizes. With multithreading (52 CPU cores on server B; see Table 1), MoDLE can simulate loop extrusion contacts for a genome size of 500 Mbp in a little over one minute (Fig. 3C). Using a single thread (1 CPU core on server B; see Table 1), the same run takes around 12 minutes (Fig. 3C), which is still reasonable from a practical perspective and much faster than GPU accelerated OpenMM simulations. MoDLE peak memory usage is only slightly affected by multithreading, as each simulation instance only requires an additional 1–10 MB of memory (Fig. 3D). When simulating more than one chromosome, peak memory usage does not follow a simple linear pattern (Fig. 3D), as it is affected by the order in which simulation tasks are executed. This can lead to scenarios where, for a brief period, two or more contact matrices are stored in system memory. We conclude that MoDLE, in contrast to OpenMM, runs efficiently even on systems with few CPU cores, such as laptop computers.

Further, we analyzed the strong scaling properties of MoDLE by simulating loop extrusion on the entire human genome (GRCh38; 3088 Mbp). Increasing the number of CPU cores from 1 to 52, MoDLE execution time scales close to theoretical optimum (see the "Methods" section for details) (Fig. 3D; blue lines). Simulating loop extrusion on the human genome takes from 1 h and 21 min (1 CPU core on server B; see Table 1) to 1 min and 48 s (52 CPU cores on server B; see Table 1). We conclude that MoDLE can efficiently run on machines with a wide range of capabilities, ranging from laptop computers with 4–8 CPU cores, to multi-socket servers with over 50 CPU cores. Memory usage increases with the number of CPU cores, but never beyond reasonable limits on modern computers (Fig. 3D; orange line).

In conclusion, MoDLE is orders of magnitude faster than OpenMM in simulating loop extrusion contacts and is especially efficient in simulating large genome regions or large input data sets. MoDLE can run efficiently on machines ranging from low-powered laptop computers to powerful multi-socket servers.

Genome wide parameter optimization

Since MoDLE simulates genome-wide loop extrusion in a few minutes, systematic exploration of features underlying loop extrusion becomes feasible. To illustrate this point, we optimized the parameters underlying the modeled binding kinetics of CTCF. MoDLE implements this as a Markov process with an "Unbound" and a "Bound" state. With this model, the self-transition probabilities P_{UU} and P_{BB} specify how stably associated CTCF is once bound to DNA. The stationary distribution of the Markov chain reflects the probability of a given CTCF binding site to be bound (π_B) in a simulation epoch (see Fig. 4A). Simulation of loop extrusion contacts using MoDLE or OpenMM can take advantage of ChIP-seq data from CTCF or cohesin to infer CTCF binding probabilities. Yet, when ChIP-seq data is not available, it is possible to simulate loop extrusion using a constant and uniform CTCF binding probability that is chosen to optimize similarity

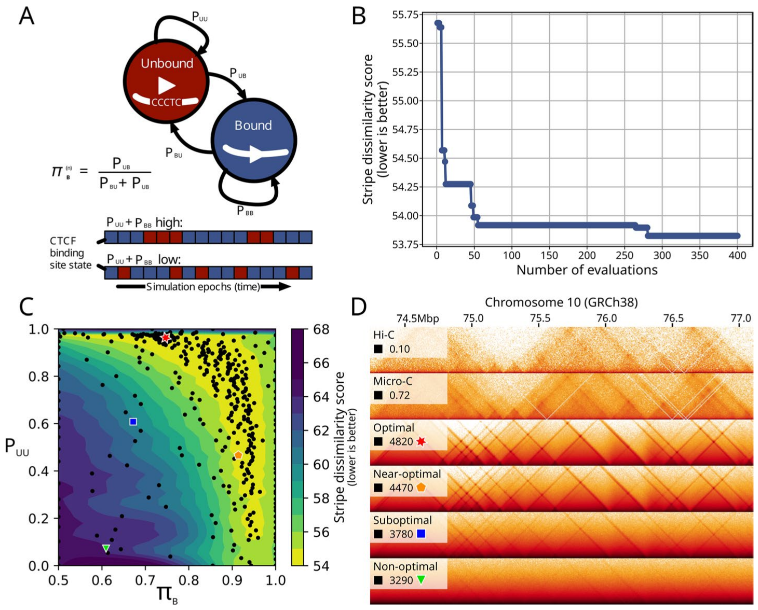


Fig. 4 Genome-wide optimization of CTCF binding kinetics underlying loop extrusion. **A** A Markov chain with an Unbound (red) and Bound (blue) state underlies MoDLE loop extrusion barrier modeling. The self-transition probability for the Bound state (P_{BB}) reflects how stably barrier elements (i.e., CTCF) are bound to their binding sites. The stationary distribution of the Markov chain (π_B) provides the CTCF binding probability at a given epoch in the simulation. The bottom diagram (red/blue boxes) shows an illustration of how the binding state (Bound in blue, and Unbound in red) of a single CTCF site would change during a simulation depending on P_{UU} and P_{BB}. **B** Convergence of the objective function during the Bayesian optimization procedure. The objective function is a dissimilarity score comparing the pixels showing stripes and dots in the observed Micro-C data with the corresponding stripes and dots in the MoDLE output. See the "Methods" section (part 6) for details. **C** Comparison of objective function in the parameter search space of P_{UU} and π_B. Optimal, near-optimal, suboptimal and non-optimal combinations are highlighted with a red star, orange pentagon, blue square and green triangle respectively. **D** Side-by-side comparison of H1-hESC Micro-C data (top panel) and progressively less optimal combinations of P_{UU} and π_B parameters

with the Micro-C (or Hi-C) data. To optimize these parameters, we make use of an approach based on Bayesian optimization using Gaussian processes (see the "Methods" section). This optimization procedure attempts to minimize an objective function without making assumptions on its analytic form. To assess MoDLE's performance, we devised an objective function representing the similarity in stripe position and length between two contact matrices using H1-hESC Micro-C data (see the "Methods" section for details). After convergence (Fig. 4B), the optimization procedure revealed a range of near-optimal combinations of transition probabilities and CTCF occupancy probabilities instead of a single, optimal combination (Fig. 4C). Comparing the resulting loop contacts of selected parameter combinations with the optimal combination ($\pi_B = 0.747$ and $P_{UU} = 0.963$) confirms that CTCF can occupy its motif instances with probabilities ranging widely between 0.6 and 0.9 as long as the stability of binding (P_{UU}) is high (> 0.8). However, low binding stabilites ($P_{UU} < 0.8$) can also yield near-optimal concordance with the Micro-C data when CTCF occupancies >0.9. Notably, the latter parameter combination is compatible with a dynamic exchange model where CTCF transiently occupies its motif instances but still maintains stable loops [42]. From a selected set of parameter combinations (Fig. 4C), we simulated genome-wide loop extrusion contacts aiming at comparing these with Hi-C and Micro-C data. The resulting comparison shows that even uniform, optimized CTCF binding probabilities (Fig. 4C red star) can recapitulate many of the features seen in Micro-C and Hi-C data (Fig. 4D). Visualization of simulated contacts using a near-optimal parameter combination from another part of the plot (Fig. 4C; orange pentagon) reinforces that a range of parameter combinations can recapitulate the patterns seen in the Hi-C and Micro-C data (Fig. 4D). Selecting a suboptimal or non-optimal combination of parameters (Fig. 4C, green triangle and blue square) results in unrealistic contact patterns (Fig. 4D; Additional file 2: Fig. S11 for an extensive comparison of different parameter combinations). In conclusion, MoDLE opens up for efficient exploration of parameters underlying DNA-DNA contact dynamics genome wide.

Predicting effects of TAD border alterations

To illustrate how MoDLE can be used to predict the effects of alterations to borders between TADs, we picked the well-characterized HoxD cluster which harbors several coordinated chromatin looping changes critical for proper limb formation in tetrapods [43, 44]. We focused on deletions between the centromeric and telomeric domain (C-Dom and T-Dom, respectively) known to cause an increase in interactions between the two domains [43], including a rewiring of multiple enhancers [44]. First, using the same parameter optimization approach described above, we inferred CTCF barrier occupancies in the wildtype condition based on JM8.N4 data. Then, we inactivated (in silico) inter-domain barrier elements by setting the occupancy of the CTCF motif instances to 0 and used MoDLE to simulate the resulting changes to the predicted loop extrusion contact maps. MoDLE correctly predicts that loops protrude beyond the deleted borders merging the two (C-Dom and T-Dom) TADs (Fig. 5). We also confirm that the border is highly resilient and requires a deletion of a large region encompassing the entire HoxD cluster to merge the TADs (see Fig. 5D–E). Inspecting enhancer signals in the region (Fig. 5E upper panel) confirms that the merging of the two domains indeed involves a rewiring of interactions of several enhancer elements, and a depletion of stripes at their borders. In conclusion, MoDLE can be used to predict changes to loop extrusion contact patterns from in silico alterations of TAD border properties.

Optimization of individual barrier parameters

In the absence of CTCF or Cohesin ChIP-seq data, MoDLE can utilize Micro-C or Hi-C data in combination with CTCF motif instances to effectively infer the occupancy of each individual barrier. To illustrate this, we selected a 5 Mb region on chromosome 1 with 2103 CTCF candidate binding sites, corresponding to over 4000 parameters to be inferred. The large number of parameters for this genome region renders a Gaussian optimization approach computationally infeasible and inadequate. Thus, we developed a system to optimize extrusion barrier parameters using genetic algorithms (GA) (see the "Methods" section part 10 for details). A comparison of the input Micro-C data (Fig. 6A) and the corresponding optimized MoDLE output (Fig. 6B) shows that even without ChIP-seq

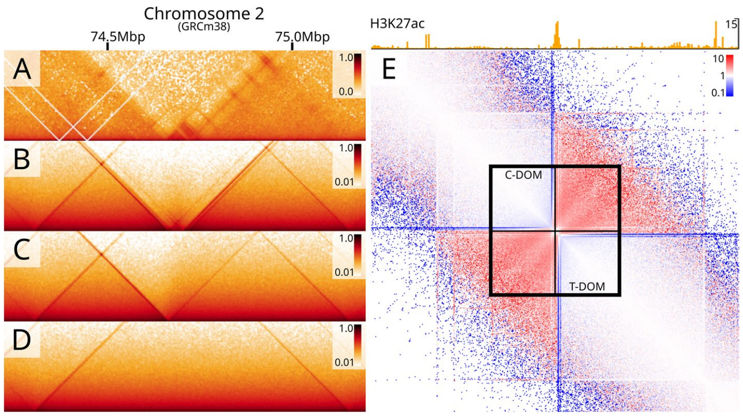


Fig. 5 Using MoDLE to predict effects of deletions to TAD borders in the HoxD locus. **A** Micro-C data in JM8. N4 mESC WT cells showing the interactions surrounding the HoxD cluster and the centromeric (C-DOM) and telomeric (T-DOM) domains in a non-mutated wildtype (WT) condition. **B** MoDLE output from the same region in the WT condition. **C** MoDLE output produced with a partial deletion of the border between the domains. **D** MoDLE output with a complete deletion of the border between the domains. **E** Differential contact map showing the ratio of MoDLE (WT condition; **B**) vs. MoDLE (full deletion, **D**). Regions enriched in MoDLE full deletion are shown in red, whereas regions enriched in MoDLE WT are shown in blue

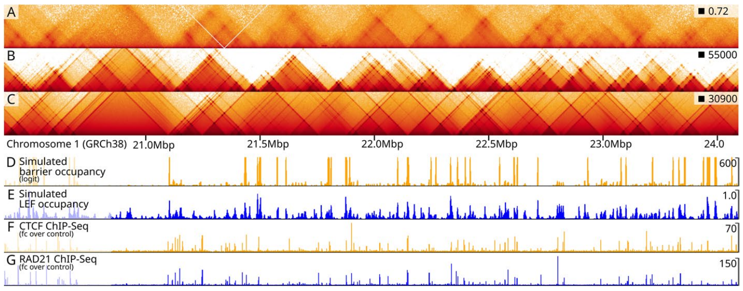


Fig. 6 Optimization of individual barriers and computation of barrier and LEF profiles. **A** Micro-C (hESC) data from a 5Mb region on chromosome 1 (20–25 Mbp). **B** MoDLE output for the same region, where individual barriers are optimized from Micro-C data. **C** MoDLE output for the same region using Rad21 ChIP-seq data as input, **D** Computed barrier occupancy profile from MoDLE trained on Micro-C data (normalized with $P_{UU} = 0.7$). **E** Computed LEF occupancy profile from MoDLE trained on Micro-C data. **F** CTCF ChIP-seq data from the same region. **G** Rad21 ChIP-seq data from the same region

information, MoDLE can be used to infer CTCF barrier occupancies individually to reproduce patterns seen in the Micro-C data. Comparing this MoDLE output with the corresponding output from MoDLE based on Rad21 ChIP-seq data (Fig. 6C) shows that TADs and borders are placed in analogous regions, yet with local differences in barrier strengths and stripe lengths. From MoDLE data simulated using optimized barrier occupancies (Fig. 6D), it is possible to compute the modeled binding profile of the LEF during the simulation (Fig. 6E; see Additional file 1: Section 9 for details). Comparing these with ChIP-seq profiles of CTCF and Rad21 (Fig. 6F and G, respectively) shows that peaks and valleys coincide in a large fraction of regions, signifying that MoDLE can indeed infer biologically meaningful signals from its input data. We conclude that MoDLE, in the absence of ChIP-seq input data, can reliably infer CTCF occupancies of individual barriers to simulate loop extrusion contact patterns and to recapitulate binding profiles of CTCF and cohesin ChIP-seq data.

Discussion

Efficient and realistic simulation of DNA-DNA spatial contacts is increasingly required for modeling and exploring genome structure and regulation. For example, our ability to reliably predict effects of mutations to TAD borders relies on available tools for simulating and comparing spatial contact data from normal and pathogenic states [14]. Further, simulations can be invaluable for exploring general genome folding principles [11] or underlying principles of loop extrusion [12, 35, 36]. Efficient tools for loop extrusion simulation will contribute to increasing our understanding of mechanisms ranging from gene regulation [1, 2] to DNA repair [3]. MoDLE represents, to the best of our knowledge, the first command-line tool for high-throughput loop extrusion contact simulation. We expect MoDLE to supplement, rather than replace existing MD tools; especially in cases where large genome regions or large data sets need to be analyzed or simulated. This would in particular be the case for large-scale exploration of parameters underlying genome structure properties, as exemplified here for the binding kinetics of CTCF. In cases where Hi-C data is not available, we expect MoDLE to be useful for high-throughput loop extrusion contact prediction based on ChIP-seq, ATAC-seq, or similar data in combination with CTCF motif instances (as exemplified in Figs. 2 and 4). In such cases, MoDLE could be useful for prediction of enhancer-promoter contacts aiding identification of functional regulatory interactions [45]. When Hi-C (or similar) data is available in a wildtype condition, MoDLE can be used for large scale prediction of mutations or alterations to TAD borders (as shown in Figs. 5 and 6). This would be useful for prioritization of mutations in genome editing settings.

New developments in experimental techniques augmented by integrated computational modeling will continue to shed light on new genome organization principles at a rapid pace [46]. With MoDLE's focus on computational speed and its modular architecture, new developments and knowledge are expected to easily be integrated into the tool to increase the complexity and realism of the underlying modeling parameters.

Conclusions

We have developed MoDLE (Modeling of DNA Loop Extrusion), allowing high-performance stochastic modeling of DNA loop extrusion. MoDLE simulates loop extrusion contact matrices on large genome regions in a few minutes, even on low-powered laptop computers. MoDLE is available as a command line tool and can be accessed at github.com/paulsengroup/modle.

Methods

MoDLE implementation and design overview

MoDLE is implemented in C++17 and is compiled with CMake. MoDLE uses a producer-consumer architecture where a single producer (a thread) communicates with multiple consumers through asynchronous message passing. The producer thread is responsible for reading input files and generating a set of simulation tasks to be consumed by a pool of worker threads. Tasks are implemented as light-weight C++ structs that are computationally cheap to generate and consume. A single task contains all the information needed for simulating DNA loop extrusion on a single chromosome in a specific simulation instance. Simulation instances are for the most part independent from each other and can thus run in parallel. We designed MoDLE such that each simulation instance requires less than 10 MB of memory to simulate loop extrusion on large mammalian chromosomes, such as chromosome 1 from the human genome. The space complexity of the thread–local state is linear with respect to the number of LEFs or extrusion barriers, whichever is largest. For a more detailed overview of MoDLE's implementation, see Additional file 1: Section 1.

Most of MoDLE's memory budget is used to store molecular contacts generated by loop extrusion. MoDLE stores one instance of its custom contact matrix data structure for each chromosome that is being actively simulated. The space complexity of a contact matrix instance depends on the chromosome length, diagonal width and bin size. With default settings, representing contacts for chromosome 1 of the human genome requires approximately 120 MB of memory. Common operations on the contact matrix class are made thread-safe using lock striping implemented through hashing. For more details regarding the specialized contact matrix data structure, refer to Additional file 1: Section 2.

To achieve high-performance, MoDLE stores most of its data in contiguous memory using simple data structures such as vectors and bitsets (see Additional file 1: Section 3). Data is indexed such that extrusion barriers and extrusion units in a simulation instance can be efficiently traversed in $5'$-$3'$ and $3'$-$5'$ directions (see Additional file 1: Section 8). This allows MoDLE to bind/release LEFs, process collisions, register contacts, and extrude DNA in linear time-complexity and with good locality of reference. The only step relying on an algorithm with super-linear time complexity is indexing, which has a worst-case time complexity of $O(n\,log\ n)$ while approaching $O(n)$ for the typical case.

More design and implementation details are available in Additional file 1. The latest MoDLE source code can be obtained in MoDLE's GitHub repository: github.com/paulsengroup/modle

Running a simulation instance

The entire simulation instance is executed by a single worker thread and consists of the following phases:

- Wait until one or more tasks are available on the task queue.
- Setup the simulation internal state based on the task specification, this includes seeding the PRNG and setting the initial state for the extrusion barriers based on the occupancy (see Additional file 1: Sections 1, 3, and 4).
- Run the simulation loop until a stopping criterion is met.

A single simulation epoch is articulated in the following steps:

- Select (inactive) LEFs that are currently not associated with DNA, and activate them. This is done by loading LEFs to a random position on the chromosome that is being simulated. The position is sampled from a uniform distribution (see Additional file 1: Section 5).
- Index extrusion units moving in the same direction so that they can be visited in 5′-3′ and 3′-5′ order (see Additional file 1: Section 8).
- Randomly select a subset of the active LEFs and use their position along the chromosome to generate molecular contacts in the chromosome contact matrix (see Additional file 1: Section 9).
- Generate candidate moves for each extrusion unit (see Additional file 1: Section 10).
- Update the extrusion barrier states by computing the next state in the Markov chain used to model extrusion barriers (see Additional file 1: Section 6).
- Detect collision events between LEFs and extrusion barriers as well as between LEFs (see Additional file 1: Sections 12b-d and g).
- Update the candidate moves for extrusion units involved in collision events to satisfy the constraints imposed by the collision events (see Additional file 1: Sections 12e-g).
- Advance LEFs' extrusion units by their respective moves (see Additional file 1: Section 5). Because of the preceding steps, this will yield a new valid simulation state, as moves have been updated to satisfy all the constraints imposed by collision events.
- Iterate over active LEFs and release them based on the outcome of a Bernoulli trial whose probability of success is computed based on the average LEF processivity and LEF state (e.g., LEFs whose extrusion units are involved in collision events with a pair of extrusion barriers in convergent orientation have a lower probability of being released). LEFs that are being released go back in the pool of available LEFs and will be loaded on a new genomic region during the next epoch (see Additional file 1: Section 5).

MoDLE will continue iterating through the above steps until one of the simulation stopping criteria is met:

- A given number of epochs have been simulated.
- Enough contacts have been registered to reach a target contact density.

Both stopping criteria can be modified by users. By default, MoDLE will simulate loop extrusion until reaching an average contact density of 1 contact per pixel.

Hardware specifications

Analysis and benchmark code used to generate the data accompanying was run using the hardware specifications listed in Table 1.

MoDLE simulations

MoDLE's data used for the heat map comparison shown in Fig. 2 were generated using the heatmap_comparison_pt1 Nextflow [47] workflow available on GitHub [48] and archived on Zenodo [49].

The list of candidate extrusion barrier positions and directions were generated by running MAST from the MEME suite [50] on GRCh38.p13 (GCF_000001405.39 [51] using the CTCF frequency matrix MA0139.1 from JASPAR 2022 [52].

The list of candidate barriers was then filtered using CTCF and RAD21 ChIP-seq data (fold-change over control and optimal IDR thresholded peaks). In brief, candidate barriers were intersected with the narrow-peak BED files for CTCF and RAD21. Then, each filtered barrier region was assigned with an occupancy computed by passing the RAD21 fold-change over control signal through a logistic function. Finally, the output of the logistic function was binned at 1 kbp to yield a barrier occupancy that is proportional to the number of CTCF motif instances as well as RAD21 fold-change over control signal. This procedure is largely based on [Fudenberg 2016]. The result of the procedure outlined above is a list of extrusion barrier occupancies binned at 1 kbp resolution. CTCF and RAD21 ChIP-seq for H1-hESC data was downloaded from ENCODE [53, 54] (ENCFF255FRL [55], ENCFF473IZV [56], ENCFF821AQO [57], and ENCFF913JGA [58].

Contact matrices were generated using MoDLE v1.0.0-rc.7 with the parameters from Additional file 3: Table S1. Parameters not listed in the table were left at default.

Contact matrices produced by MoDLE were then subsampled to an average contact density of 3 using cooltools random-sample v0.5.1 [59]. The resulting cooler files were then converted to multi-resolution cooler files using cooler zoomify [40]. Finally, multi–resolution contact matrices were visualized in HiGlass (v1.11.7) [60].

Molecular dynamics (OpenMM) simulations

Molecular dynamics data used for the heat map comparison in Fig. 2 were generated using the heatmap_comparison_pt1 Nextflow workflow available on GitHub [48] and archived on Zenodo [49]. This workflow uses OpenMM [37] to run MD simulations.

Simulation code is largely based on [61]. Simulations were carried out on 10 Mbp regions from chromosomes 2, 3, 5, 7, and 17 using a monomer size of 1 kbp and 200 kbp for LEF processivity and separation. Extrusion barrier positions, directions, and occupancy were generated following the procedure outlined in the "Methods" section (part 1).

Contact matrices were generated with Polychrom [62] using a bin size of 5 kbp. The resulting cooler files were then converted to multi-resolution cooler files using cooler zoomify v0.8.11 [40].

Assessing loop extrusion feature similarities from contact frequencies

To objectively compare the contact matrices produced by MoDLE with contact matrices generated from Micro-C experiments and MD simulations, we developed a specialized scoring algorithm. The algorithm was inspired by Stripenn [63].

The score is computed on rows and columns of a pair of contact matrices of identical resolutions transformed as follows.

First, both matrices are convolved using the difference of Gaussian (DoG). This highlights stripe and dot patterns found in contact matrices. Next, the transformed contact matrices are discretized using a step function mapping values below a certain threshold to 0 and all the others to 1. This results in two binary matrices, where non-zero pixels can be interpreted as part of a stripe or dot. Finally, we take advantage of the fact that stripes produced by loop extrusion always should start from the matrix diagonal. Thus, given a row or column of pixels starting on the matrix diagonal, and extending away from it, we stipulate that the last non-zero pixel in the vector of values represents the end of a stripe produced by DNA loop extrusion.

Given the above, we can measure the similarity of stripes between two contact matrices by considering the same row of pixels in a pair of contact matrices, computing the last non-zero pixels in both rows, and counting the number of matches. The same approach can be applied to columns of pixels. Finally, counting mismatches instead of matches can be used as a measure of dissimilarity. Contact matrix convolution and discretization, as well as computing this special score, can be done using MoDLE's helper tools (modle_tools transform and modle_tools evaluate respectively).

Contact matrix comparison

For comparison with MoDLE and OpenMM output, we used available Hi-C and Micro-C data from H1-hESC because these were of high resolution and had accompanying ChIP-seq data for both CTCF and RAD21 (4DNFIFJH2524 [64], 4DNFI9GMP2J8 [65], ENCFF255FRL [55], ENCFF473IZV [56], ENCFF821AQO [57], and ENCFF913JGA [58]). To assess stripe similarity of a pair of contact matrices, we used the scoring algorithm described in the "Methods" section (part 6). The score was computed using Micro-C data as the ground truth. Pixel accuracy was computed as the ratio of correctly classified pixels to the total number of pixels in a 3 Mbp subdiagonal window around each barrier. The Pearson correlation between OpenMM and MoDLE was calculated based on all corresponding 5 kbp-pixel values in the 3 Mbp subdiagonal window of the OpenMM simulation regions.

Benchmark methodology

Benchmarks were run on a computing cluster using the run_benchmarks Nextflow workflow available on GitHub [48] and archived on Zenodo [49].

We ran two suites of benchmarks to assess the performance of MoDLE and compare it with that of molecular dynamics simulations based on OpenMM.

The first suite (Fig. 3A–C) compared the performance of MoDLE and OpenMM when simulating loop extrusion on an artificial chromosome with increasing length (ranging from 1 to 250 Mbp).

This benchmark was run using MoDLE (1 and 52 CPU cores) as well as OpenMM GPU and CPU implementation (1 CPU core, 1 GPU, and 8 CPU cores respectively). CPU benchmarks were run on server C while benchmarks relying on GPU acceleration were run on server D (see Table 1). For OpenMM CPU implementation, we limited the number of CPU cores to 8 (16 SMT cores) as the CPU implementation is known to not scale well past 16 threads [66]. OpenMM CPU implementation was used to simulate chromosome lengths up to 5 Mbp for practical reasons. MoDLE was run with default settings except for the number of cells, which was set to 104 to match the maximum number of available SMT cores.

OpenMM simulations were run using a monomer size of 2 kbp and LEF processivity and separation of 200 kbp.

The second suite of benchmarks involved simulating loop extrusion on the human genome (GRCh38) using MoDLE with a number of CPU cores ranging from 1 to 52. MoDLE was run with default settings except for the number of cells which was set to 104. The extrusion barrier annotation was generated as described in the "Methods" section (part 1).

In both cases, measurements were repeated 10 times for MoDLE and 5 times for OpenMM.

Genome-wide extrusion barrier parameter optimization

The genome-wide optimization of parameters affecting extrusion barrier occupancies was carried out using the gw_param_optimization Nextflow workflow available on GitHub [48] and archived on Zenodo [49].

The first step in the optimization procedure is running Stripenn v1.1.65.7 [63] on the H1-hESC Micro-C (4DNFI9GMP2J8 [67]) dataset to identify architectural stripes, which resulted in the identification of 5254 stripes. A small subset of these stripes were visually validated by comparing the annotated stripes with stripes that are visible from Micro-C data. Annotated stripes were split into two equally sized datasets by random sampling without replacement. One dataset was used for parameter optimization while the other was used for validation.

Parameter optimization is performed through the Bayesian optimization from scikit-optimize v0.9.0 [68] using an objective function based on the scoring metric described in Methods (part 6).

The parameters that are being optimized are the extrusion barrier occupancy (π_B) and P_{UU}, the self-transition probability of the unbound state.

The evaluation of the objective function proceeds as follows:

- A new genome-wide simulation is performed using the parameters proposed by the optimizer.
- The resulting cooler file is transformed with modle_tools transform by applying the difference of Gaussians followed by a binary discretization step, where pixel values above a certain threshold are discretized to 1 and all the others to 0.
- The score described in Methods (part 6) is then computed row and column-wise on the entire genome using modle_tools eval, producing two BigWig files. Here, the transformed Micro-C 4DNFI9GMP2J8 [67] dataset is used as reference.
- Scores are intersected with the extrusion barrier dataset for optimization and validation considering stripe direction (i.e., vertical stripes are intersected with column-wise scores while horizontal stripes are intersected with row-wise scores).
- Scores resulting from the intersection are then averaged, producing a floating-point number that is then returned to the optimizer, which will try to minimize this number.

In the transformation step, a σ of 1.0 and 1.6 are used to generate the less and more blurry contact matrices to subtract when computing the difference of Gaussians. For the binary discretization of the Micro-C data, a threshold of 1.5 was used, while simulated data was discretized using 0.75 as threshold.

The optimizer evaluated the objective function 400 times, each time computing the average score for the training and validation datasets.

Finally, the parameters that yielded the best score on the training dataset were used to generate a contact matrix in cooler format (see Fig. 4D, bottom panel).

Local extrusion barrier parameter optimization

The local extrusion barrier parameter optimization was carried out using the extrusion_barrier_param_optimization Nextflow workflow available on GitHub [48] and archived on Zenodo [49].

In brief, this workflow takes as input an extrusion barrier annotation consisting of barrier position and direction, and then optimizes the parameters for each individual barrier to maximize similarities with a reference HiC matrix.

The optimization approach is based on evolutionary algorithms (EAs) and was developed using primitives from the DEAP library [69].

Optimization was performed on a 5 Mbp region of the human chromosome 1 (20–25 Mbp, GRCh38) using the list of candidate CTCF binding sites overlapping this region as extrusion barrier annotation, for a total of 2103 extrusion barriers. Candidate CTCF binding sites were annotated using MAST as described in Methods (part 4). The H1-hESC Micro-C (4DNFI9GMP2J8 [65]) matrix was used as reference.

At a high level, the optimization workflow consists of running the same optimization script three times, using the output of an optimization run as input for the next run. The first run is tuned to favor exploration over exploitation, while the second and third runs used more conservative optimization parameters to progressively reduce the rate of exploration and favor exploitation.

The following is an overview of how the optimization strategy was developed:

- The optimization uses μ, λ as evolution strategy, where μ is the population size and λ is the number of offspring produced each generation. With this strategy, offspring that make it through the selection stage replace the previous population entirely. By default, $\mu = 256$ and $\lambda = 512$.
- Individuals are represented as two lists of real numbers of size N, where N is the number of extrusion barriers to be optimized. The first list of numbers represent s extrusion barrier occupancies (π_B), while the second list represents the self-transition probability of the unbound state (P_{UU}).
- Individuals are mutated by adding a relatively small offsets to $\overrightarrow{\pi_B}$ and $\overrightarrow{P_{UU}}$. Offsets are drawn from a normal distribution with $\mu = 0$ and σ set based on the desired degree of exploration. Values are clamped between 0.0 and 1.0, so mutating an individual always leads to another valid individual.
- The two-point crossover operator is used for mating.
- During selection, offsprings are sorted based on their fitness, and the top μ offsprings are selected to proceed to the next generation.
- The population is initialized differently depending on whether results from a previous optimization run are available.

– Results from previous optimization are available: population initialized through random sampling with replacement from the set of fittest individuals that ever lived in the previous optimization run.
– Otherwise, population is randomly initialized by generating μ individuals with π_B and P_{UU} set to random numbers drawn from the uniform distribution $U(0.0,1.0)$.
– Fitness is computed using a slightly modified version of the scoring function $f\left(\overrightarrow{x}\right)$described in Methods (part 6). Function $f\left(\overrightarrow{x}\right)$ is not effective at guiding the optimization when occupancy is relatively low (e.g., < 0.5), and there are no stripes or dots in the reference matrix, as any parameter combination resulting in such a low occupancy produces no visible stripe or dot. To this end, we define a penalty function $p(\pi_B)$ that returns a coefficient between 1.0 and 2.0. The returned coefficient is close to 2.0 when π_B approaches 0.5 and rapidly falls to 1.0 when π_B moves towards 0.0 or 1.0. π_B very close to 1.0 are also penalized. See Additional file 2: Fig. S12 for more details regarding the penalty function $p(\pi_B)$.
– The output of the scoring function $f\left(\overrightarrow{x}\right)$ and penalty function $p(\pi_B)$ are multiplied together to produce the score used to compute the fitness of an individual $s\left(\overrightarrow{x}, \pi_B\right) = f\left(\overrightarrow{x}\right) \cdot p(\pi_B)$. The fitness of an individual is computed as the average of the scores $s\left(\overrightarrow{x}, \pi_B\right)$ computed in correspondence of each extrusion barrier object of the optimization.
– The optimization runs until one of the following conditions is met:

  – A target number of generations have been simulated (i.e., 1000 generations).
  – Optimizer failed to significantly improve the population fitness (e.g., less than 1% fitness improvement over the last 25 generations).
  – The population variability approaches 0.

To improve the performance of the optimizer on these regions, we split the population into mainland population and one or more insular populations and change some aspects of the optimization strategy.

First, we initialize and optimize the mainland population ($\mu = 256$ and $\lambda = 512$). When one of the stopping criteria is met, the fittest individuals from mainland are used to initialize the population of m islands. For each island, we randomly select and mask k consecutive alleles or barriers. k is generated by rounding a number drawn from a normal distribution with $\mu = 25$ and $\sigma = 5.0$. Crucially, masked barriers are inactive and are not allowed to mutate.

For one of the m islands, instead of masking a random stretch of extrusion barriers, we inactivate all weak barriers when initializing the population. Thus, we replace alleles with $\pi_B < 0.5$ with the $\pi_B = 0.0$; $P_{UU} = 1.0$ allele. In this case, all loci are allowed to mutate. Islands have $\mu = 128$ and $\lambda = 256$. Islands evolve independently from each other and from the mainland and follow the same stopping criteria used for the mainland.

Once all islands have been optimized, half of the mainland individuals are replaced with individuals from any of the islands. Island individuals are selected using fitness proportionate selection (i.e., random sampling with replacement, weighted by fitness).

Mainland and island optimization continue alternating until a total target number of mainland generations have been simulated, or when an optimization cycle fails to significantly improve the average mainland population fitness.

Simulations to predict the effect of TAD border alterations

Data for this section was generated using the comparison_with_mut Nextflow workflow available on GitHub [48] and archived on Zenodo [49].

Simulations were carried out using GRCm38.p6 as reference genome (GCF_000001635.26 [70].

CTCF and RAD21 ChIP-seq fold-change over control for JM8.N4 was generated by processing data from GSE90994 [71] (SRR5085152 [72], SRR5085153 [73], SRR5085154 [74], SRR5085155 [75], SRR5085156 [76], SRR5085157 [77]) using the ENCODE ChIP-seq pipeline v2 [78] and using ENCODE4 genomic datasets for GRCm38.

The wild-type extrusion barrier annotation was generated following the procedure outlined in the "Methods" section (part 4).

The barrier annotation was further refined using the parameter optimization strategy described in the "Methods" section (part 10) using a JM8.N4 Micro-C dataset as reference (4DNFINNZDDXV [79]).

The optimized extrusion barrier annotation was then mutated by removing extrusion barriers overlapping the del1-13d9lac and delattP-Rel5d9lac regions from Rodríguez-Carballo 2017 [43].

Supplementary Information

The online version contains supplementary material available at https://doi.org/10.1186/s13059-022-02815-7.

Additional file 1. Supplementary text. Detailed description of MoDLE's underlying simulation model and implementation [16, 88, 89].

Additional file 2. Supplementary figures. Supplementary Figures 1-12.

Additional file 3. Supplementary tables. Supplementary Tables 1-2 [90–93].

Additional file 4. Review history.

Acknowledgements
Simulations and data analyses were performed on resources provided by Sigma2 - the National Infrastructure for High Performance Computing and Data Storage in Norway, with account number NN8041K.
Data produced by simulations and data analyses are stored in the NIRD Research Data Archive with the DOI: 10.11582/2022.00056.
We thank the ENCODE Consortium and the labs of Peter Park and Job Dekker for contributing with ENCODE data used as part of this study.

Peer review information
Andrew Cosgrove was the primary editor of this article and managed its editorial process and peer review in collaboration with the rest of the editorial team.

Review history
The review history is available as Additional file 4.

Authors' contributions
JP and RR conceived and designed the study; RR developed the software and the analysis code; RR performed benchmarking analysis and genome wide and local parameter optimization as well as the comparative analysis for the HoxD mutants; JP performed comparison analyses of MoDLE with OpenMM and Micro-C data; JP and RR wrote the original draft; AM helped with the manuscript revising; VK, AM, and TR provided critical feedback and assisted in the improvement of the tool and analyses; JP supervised the project. All authors read and approved the final manuscript.

Funding
JP acknowledges funding from the Norwegian Research Council (project 324137).

Availability of data and materials

Project name: MoDLE
Project home page: https://github.com/paulsengroup/modle
Archived version: 10.5281/zenodo.6424697
License: MIT
Operative system(s): UNIX-like (Platform independent when using containers)
Programming language: C++
Other requirements:

• C++17 compiler (e.g., GCC 8+, Clang8+, AppleClang 10+)
• CMake 3.18 or newer
• Conan 1.50 or newer
• Python3 (to install and run Conan)
• Scipy (Python3 package, required to run unit tests)
• wCorr (R package, required to run unit tests)

The complete list of MoDLE dependencies is available in Additional file 3: Table S2. Dependency installation is automated using CMake and Conan.
MoDLE source code is available on GitHub at https://github.com/paulsengroup/modle [80] and is archived on Zenodo at https://doi.org/10.5281/zenodo.6424697 [81]. MoDLE's version used throughout the manuscript is MoDLE v1.0.0-rc.7 [82], except for performance benchmarks which used MoDLE v1.0.0-rc.3 [83].
Code used for the data analysis is available on GitHub at https://github.com/paulsengroup/2021-modle-paper-001-data-analysis [48] and is archived on Zenodo at https://doi.org/10.5281/zenodo.7072939 [49].
Data produced by running the runme.sh script from the data analysis code repository, including simulated contact matrices in cooler format were archived on the NIRD research data archive at DOI 10.11582/2022.00056 [84].
Reference genome assembly and assembly report for GCF_000001405.39_GRCh38.p13 [51] and GCF_000001635.26 [70] were downloaded through the NCBI FTP server [85].
ChIP-seq data for the following accession numbers were downloaded from the ENCODE portal [53, 54]: ENCFF255FRL [55], ENCFF473IZV [56], ENCFF821AQO [57], and ENCFF913JGA [58].
ChIP-seq data for the following accession number was downloaded from the Gene Expression Omnibus: GSM4665702 [86].
ChIP-seq sequencing data for GSE90994 [71] were downloaded from EBI's mirror of the SRA: SRR5085152 [72], SRR5085153 [73], SRR5085154 [74], SRR5085155 [75], SRR5085156 [76], and SRR5085157 [77].
H1-hESC Hi-C and Micro-C data as well as JM8.N4 Micro-C data in multi-resolution cooler format (4DNFIFJH2524 [64], 4DNFI9GMP2J8 [67], 4DNFINNZDDXV [79]) were downloaded from the 4DNucleome Data Portal [87].
The frequency matrix in MEME format for the CTCF motif (MA0139.1) was downloaded from JASPAR 2022 CORE non-redundant database [52].

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 13 April 2022 Accepted: 17 November 2022
Published online: 30 November 2022

References

1. Braccioli L, de Wit E. CTCF: a Swiss-army knife for genome organization and transcription regulation. Essays Biochem. 2019;63:157–65.
2. Razin SV, Gavrilov AA, Vassetzky YS, Ulianov SV. Topologically-associating domains: gene warehouses adapted to serve transcriptional regulation. Transcription. 2016;7:84–90.
3. Arnould C, Rocher V, Finoux A-L, Clouaire T, Li K, Zhou F, et al. Loop extrusion as a mechanism for formation of DNA damage repair foci. Nature. 2021;590:660–5.
4. Peters J-M. How DNA loop extrusion mediated by cohesin enables V(D)J recombination. Curr Opin Cell Biol. 2021;70:75–83.
5. Goloborodko A, Marko JF, Mirny LA. Chromosome compaction by active loop extrusion. Biophys J. 2016;110:2162–8.
6. Ganji M, Shaltiel IA, Bisht S, Kim E, Kalichava A, Haering CH, et al. Real-time imaging of DNA loop extrusion by condensin. Science. 2018;360:102–5.
7. Golfier S, Quail T, Kimura H, Brugués J. Cohesin and condensin extrude DNA loops in a cell cycle-dependent manner. Elife. 2020:9. https://doi.org/10.7554/eLife.53885.
8. Lieberman-Aiden E, van Berkum NL, Williams L, Imakaev M, Ragoczy T, Telling A, et al. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. Science. 2009;326:289–93.
9. Hsieh T-HS, Weiner A, Lajoie B, Dekker J, Friedman N, Rando OJ. Mapping nucleosome resolution chromosome folding in yeast by Micro-C. Cell. 2015;162:108–19.

10. Krietenstein N, Abraham S, Venev SV, Abdennur N, Gibcus J, Hsieh T-HS, et al. Ultrastructural details of mammalian chromosome architecture. Mol Cell. 2020;78:554–65.e7.
11. Sanborn AL, Rao SSP, Huang S-C, Durand NC, Huntley MH, Jewett AI, et al. Chromatin extrusion explains key features of loop and domain formation in wild-type and engineered genomes. Proc Natl Acad Sci U S A. 2015;112:E6456–65.
12. Fudenberg G, Imakaev M, Lu C, Goloborodko A, Abdennur N, Mirny LA. Formation of chromosomal domains by loop extrusion. Cell Rep. 2016;15:2038–49.
13. Brandão HB, Ren Z, Karaboja X, Mirny LA, Wang X. DNA-loop-extruding SMC complexes can traverse one another in vivo. Nat Struct Mol Biol. 2021;28:642–51.
14. Lupiáñez DG, Spielmann M, Mundlos S. Breaking TADs: How alterations of chromatin domains result in disease. Trends Genet. 2016;32:225–37.
15. Alipour E, Marko JF. Self-organization of domain structures by DNA-loop-extruding enzymes. Nucleic Acids Res. 2012;40:11202–12.
16. Bauer BW, Davidson IF, Canena D, Wutz G, Tang W, Litos G, et al. Cohesin mediates DNA loop extrusion by a "swing and clamp" mechanism. Cell. 2021;184:5448–64.e22.
17. Golov AK, Golova AV, Gavrilov AA, Razin SV. Sensitivity of cohesin-chromatin association to high-salt treatment corroborates non-topological mode of loop extrusion. Epigenetics Chromatin. 2021;14:36.
18. Pradhan B, Barth R, Kim E, Davidson IF, Bauer B, van Laar T, et al. SMC complexes can traverse physical roadblocks bigger than their ring size [Internet]. bioRxiv. 2021:2021.07.15.452501 Available from: https://www.biorxiv.org/content/10.1101/2021.07.15.452501v1.abstract. [Cited 2022 Apr 11].
19. Rao SSP, Huntley MH, Durand NC, Stamenova EK, Bochkov ID, Robinson JT, et al. A 3D map of the human genome at kilobase resolution reveals principles of chromatin looping. Cell. 2014;159:1665–80.
20. Vian L, Pękowska A, Rao SSP, Kieffer-Kwon K-R, Jung S, Baranello L, et al. The energetics and physiological impact of cohesin extrusion. Cell. 2018;173:1165–78.e20.
21. Tedeschi A, Wutz G, Huet S, Jaritz M, Wuensche A, Schirghuber E, et al. Wapl is an essential regulator of chromatin structure and chromosome segregation. Nature. 2013;501:564–8.
22. Haarhuis JHI, van der Weide RH, Blomen VA, Yáñez-Cuna JO, Amendola M, van Ruiten MS, et al. The cohesin release factor WAPL restricts chromatin loop extension. Cell. 2017;169:693–707.e14.
23. Nakamura R, Motai Y, Kumagai M, Wike CL, Nishiyama H, Nakatani Y, et al. CTCF looping is established during gastrulation in medaka embryos. Genome Res. 2021;31:968–80.
24. Wutz G, Várnai C, Nagasaka K, Cisneros DA, Stocsits RR, Tang W, et al. Topologically associating domains and chromatin loops depend on cohesin and are regulated by CTCF, WAPL, and PDS5 proteins. EMBO J. 2017;36:3573–99.
25. Rao SSP, Huang S-C, Glenn St Hilaire B, Engreitz JM, Perez EM, Kieffer-Kwon K-R, et al. Cohesin loss eliminates all loop domains. Cell. 2017;171:305–20.e24.
26. Liu NQ, Magnitov M, Schijns M, van Schaik T, van der Weide RH, Teunissen H, et al. Rapid depletion of CTCF and cohesin proteins reveals dynamic features of chromosome architecture [Internet]. bioRxiv. 2021:2021.08.27.457977 Available from: https://www.biorxiv.org/content/10.1101/2021.08.27.457977v1.full. [Cited 2022 Apr 11].
27. Nora EP, Goloborodko A, Valton A-L, Gibcus JH, Uebersohn A, Abdennur N, et al. Targeted degradation of CTCF decouples local insulation of chromosome domains from genomic compartmentalization. Cell. 2017;169:930–44.e22.
28. Barbieri M, Chotalia M, Fraser J, Lavitas L-M, Dostie J, Pombo A, et al. Complexity of chromatin folding is captured by the strings and binders switch model. Proc Natl Acad Sci U S A. 2012;109:16173–8.
29. Naumova N, Imakaev M, Fudenberg G, Zhan Y, Lajoie BR, Mirny LA, et al. Organization of the mitotic chromosome. Science. 2013;342:948–53.
30. Jost D, Carrivain P, Cavalli G, Vaillant C. Modeling epigenome folding: formation and dynamics of topologically associated chromatin domains. Nucleic Acids Res. 2014;42:9553–61.
31. Benedetti F, Dorier J, Burnier Y, Stasiak A. Models that include supercoiling of topological domains reproduce several known features of interphase chromosomes. Nucleic Acids Res. 2014;42:2848–55.
32. Goloborodko A, Imakaev MV, Marko JF, Mirny L. Compaction and segregation of sister chromatids via active loop extrusion. Elife. 2016:5. https://doi.org/10.7554/eLife.14864.
33. Anderson JA, Glaser J, Glotzer SC. HOOMD-blue: a Python package for high-performance molecular dynamics and hard particle Monte Carlo simulations. Comput Mater Sci. 2020;173:109363.
34. Schwarzer W, Abdennur N, Goloborodko A, Pekowska A, Fudenberg G, Loe-Mie Y, et al. Two independent modes of chromatin organization revealed by cohesin removal. Nature. 2017;551:51–6.
35. Nuebler J, Fudenberg G, Imakaev M, Abdennur N, Mirny LA. Chromatin organization by an interplay of loop extrusion and compartmental segregation. Proc Natl Acad Sci U S A. 2018;115:E6697–706.
36. Banigan EJ, van den Berg AA, Brandão HB, Marko JF, Mirny LA. Chromosome organization by one-sided and two-sided loop extrusion. Elife. 2020:9. https://doi.org/10.7554/eLife.53558.
37. Eastman P, Swails J, Chodera JD, McGibbon RT, Zhao Y, Beauchamp KA, et al. OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. PLoS Comput Biol. 2017;13:e1005659.
38. Buckle A, Brackley CA, Boyle S, Marenduzzo D, Gilbert N. Polymer simulations of heteromorphic chromatin predict the 3D folding of complex genomic loci. Mol Cell. 2018;72:786–97.e11.
39. Zhang S, Übelmesser N, Josipovic N, Forte G, Slotman JA, Chiang M, et al. RNA polymerase II is required for spatial chromatin reorganization following exit from mitosis. Sci Adv. 2021;7:eabg8205.
40. Abdennur N, Mirny LA. Cooler: scalable storage for Hi-C data and other genomically labeled arrays. Bioinformatics. 2020;36:311–6.
41. Gabriele M, Brandão HB, Grosse-Holz S, Jha A, Dailey GM, Cattoglio C, et al. Dynamics of CTCF and cohesin mediated chromatin looping revealed by live-cell imaging [Internet]. bioRxiv. 2021:2021.12.12.472242 Available from: https://www.biorxiv.org/content/10.1101/2021.12.12.472242v1. [Cited 2022 Apr 11].
42. Hansen AS, Pustova I, Cattoglio C, Tjian R, Darzacq X. CTCF and cohesin regulate chromatin loop stability with distinct dynamics. Elife. 2017:6. https://doi.org/10.7554/eLife.25776.

43. Rodríguez-Carballo E, Lopez-Delisle L, Zhan Y, Fabre PJ, Beccari L, El-Idrissi I, et al. The *HoxD* cluster is a dynamic and resilient TAD boundary controlling the segregation of antagonistic regulatory landscapes. Genes Dev. 2017;31:2264–81.
44. Rodríguez-Carballo E, Lopez-Delisle L, Willemin A, Beccari L, Gitto S, Mascrez B, et al. Chromatin topology and the timing of enhancer function at the *HoxD* locus. Proc Natl Acad Sci U S A. 2020;117:31231–41.
45. Xu H, Zhang S, Yi X, Plewczynski D, Li MJ. Exploring 3D chromatin contacts in gene regulation: The evolution of approaches for the identification of functional enhancer-promoter interaction. Comput Struct Biotechnol J. 2020;18:558–70.
46. Di Stefano M, Paulsen J, Jost D, Marti-Renom MA. 4D nucleome modeling. Curr Opin Genet Dev. 2021;67:25–32.
47. Di Tommaso P, Chatzou M, Floden EW, Barja PP, Palumbo E, Notredame C. Nextflow enables reproducible computational workflows. Nat Biotechnol. 2017;35:316–9.
48. 2021-modle-paper-001-data-analysis: Data analysis code for the first paper about MoDLE (preprint available soon) [Internet]. Github. Available from: https://github.com/paulsengroup/2021-modle-paper-001-data-analysis. [Cited 2022 Apr 11].
49. Rossini R, Kumar V, Mathelier A, Rognes T, Paulsen J. Data analysis code for: "MoDLE: High-performance stochastic modeling of DNA loop extrusion interactions." 2022. Available from: https://zenodo.org/record/7072939. [Cited 2022 Sep 13].
50. Bailey TL, Gribskov M. Combining evidence using p-values: application to sequence homology searches. Bioinformatics. 1998;14:48–54.
51. GRCh38.p13 - hg38 - Genome - Assembly - NCBI [Internet]. Available from: https://identifiers.org/assembly:GCF_000001405.39. [Cited 2022 Apr 12].
52. Castro-Mondragon JA, Riudavets-Puig R, Rauluseviciute I, Lemma RB, Turchi L, Blanc-Mathieu R, et al. JASPAR 2022: the 9th release of the open-access database of transcription factor binding profiles. Nucleic Acids Res. 2022;50:D165–73.
53. ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. Nature. 2012;489:57–74.
54. Davis CA, Hitz BC, Sloan CA, Chan ET, Davidson JM, Gabdank I, et al. The Encyclopedia of DNA elements (ENCODE): data portal update. Nucleic Acids Res. 2018;46:D794–801.
55. ENCFF255FRL – ENCODE [Internet]. Available from: https://identifiers.org/encode:ENCFF255FRL. [Cited 2022 Apr 12].
56. ENCFF473IZV – ENCODE [Internet]. Available from: https://identifiers.org/encode:ENCFF473IZV. [Cited 2022 Apr 12].
57. ENCFF821AQO – ENCODE [Internet]. Available from: https://identifiers.org/encode:ENCFF821AQO. [Cited 2022 Apr 12].
58. ENCFF913JGA – encode [Internet]. Available from: https://identifiers.org/encode:ENCFF913JGA. [Cited 2022 Apr 12].
59. Venev S, Abdennur N, Goloborodko A, Flyamer I, Fudenberg G, Nuebler J, et al. open2c/cooltools: v0.5.1 [Internet]. 2022. Available from: https://zenodo.org/record/6324229.
60. Kerpedjiev P, Abdennur N, Lekschas F, McCallum C, Dinkla K, Strobelt H, et al. HiGlass: web-based visual exploration and analysis of genome interaction maps. Genome Biol. 2018;19:125.
61. Banigan EJ, Mirny LA. The interplay between asymmetric and symmetric DNA loop extrusion. Elife. 2020:9. https://doi.org/10.7554/eLife.63528.
62. Imakaev M, Goloborodko A, hbbrandao. mirnylab/polychrom: v0.1.0 [Internet]. 2019. Available from: https://zenodo.org/record/3579473. [Cited 2022 Sep 13].
63. Yoon S, Chandra A, Vahedi G. Stripenn detects architectural stripes from chromatin conformation data using computer vision. Nat Commun. 2022;13:1602.
64. 4DNFIFJH2524.mcool – 4DN Data Portal [Internet]. Available from: https://identifiers.org/4dn:4DNFIFJH2524. [Cited 2022 Sep 13].
65. 4DNFI9GMP2J8.mcool – 4DN Data Portal [Internet]. Available from: https://identifiers.org/4dn:4DNFI9GMP2J8. [Cited 2022 Sep 13].
66. openmm [Internet]. Github. Available from: https://github.com/openmm/openmm/issues/3267. [Cited 2022 Apr 11].
67. 4DNFI9GMP2J8.Mcool – 4DN data portal [Internet]. Available from: https://identifiers.org/4dn:4DNFI9GMP2J8. [Cited 2022 Apr 12].
68. scikit-optimize: Sequential model-based optimization with a `scipy.optimize` interface [Internet]. Github. Available from: https://github.com/scikit-optimize/scikit-optimize. [Cited 2022 Apr 11].
69. Fortin F-A, De Rainville F-M, Gardner M-A, Parizeau M, Gagné C. DEAP: evolutionary algorithms made easy. J Mach Learn Res. 2012;13:2171–5.
70. GRCm38.p6 - Genome - Assembly - NCBI [Internet]. Available from: https://identifiers.org/assembly:GCF_000001635.26. [Cited 2022 Sep 13].
71. GEO Accession viewer [Internet]. Available from: https://identifiers.org/GEO:GSE90994. [Cited 2022 Sep 13].
72. GSM2418858: WT_IgG_ChIPSeq; Mus musculus; ChIP-Seq - SRA - NCBI [Internet]. Available from: https://identifiers.org/insdc.sra:SRR5085152. [Cited 2022 Sep 13].
73. GSM2418858: WT_IgG_ChIPSeq; Mus musculus; ChIP-Seq - SRA - NCBI [Internet]. Available from: https://identifiers.org/insdc.sra:SRR5085153. [Cited 2022 Sep 13].
74. GSM2418859: WT_Rad21_ChIPSeq; Mus musculus; ChIP-Seq - SRA - NCBI [Internet]. Available from: https://identifiers.org/insdc.sra:SRR5085154. [Cited 2022 Sep 13].
75. GSM2418859: WT_Rad21_ChIPSeq; Mus musculus; ChIP-Seq - SRA - NCBI [Internet]. Available from: https://identifiers.org/insdc.sra:SRR5085155. [Cited 2022 Sep 13].
76. GSM2418860: WT_CTCF_ChIPSeq; Mus musculus; ChIP-Seq - SRA - NCBI [Internet]. Available from: https://identifiers.org/insdc.sra:SRR5085156. [Cited 2022 Sep 13].
77. GSM2418860: WT_CTCF_ChIPSeq; Mus musculus; ChIP-Seq - SRA - NCBI [Internet]. Available from: https://identifiers.org/insdc.sra:SRR5085157. [Cited 2022 Sep 13].

78. GitHub - ENCODE-DCC/chip-seq-pipeline2: ENCODE ChIP-seq pipeline [Internet]. GitHub. Available from: https://github.com/ENCODE-DCC/chip-seq-pipeline2. [Cited 2022 Sep 13].
79. 4DNFINNZDDXV.mcool – 4DN Data Portal [Internet]. Available from: https://identifiers.org/4dn:4DNFINNZDDXV. [Cited 2022 Sep 13].
80. modle: High-performance stochastic modeling of DNA loop extrusion interactions [Internet]. Github. Available from: https://github.com/paulsengroup/modle. [Cited 2022 Apr 11].
81. Rossini R, Kumar V, Mathelier A, Rognes T, Paulsen J. MoDLE [Internet]. Zenodo; 2022. Available from: https://zenodo.org/record/6424697.
82. Rossini R, Kumar V, Mathelier A, Rognes T, Paulsen J. MoDLE. 2022. Available from: https://zenodo.org/record/6992533. [Cited 2022 Sep 13].
83. Rossini R, Vipin K, Mathelier A, Rognes T, Paulsen J. MoDLE: High-performance stochastic modeling of DNA loop extrusion interactions [Internet]. 2022. Available from: https://doi.org/10.5281/zenodo.6424890.
84. Nird research data archive [Internet]. https://doi.org/10.11582/2022.00056. [Cited 2022 Nov 2].
85. O'Leary NA, Wright MW, Brister JR, Ciufo S, Haddad D, McVeigh R, et al. Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation. Nucleic Acids Res. 2016;44:D733–45.
86. GEO Accession viewer [Internet]. Available from: https://identifiers.org/GEO:GSM4665702. [Cited 2022 Sep 13].
87. Reiff SB, Schroeder AJ, Kirli K, Cosolo A, Bakker C, Mercado L, et al. The 4D Nucleome Data Portal: a resource for searching and visualizing curated nucleomics data [Internet]. bioRxiv. 2021 . 2021.10.14.464435. Available from: https://www.biorxiv.org/content/10.1101/2021.10.14.464435v1. [Cited 2022 Apr 11].
88. Blackman D, Vigna S. Scrambled linear pseudorandom number generators. ACM Trans Math Softw. New York, NY, USA: Association for Computing Machinery; 2021;47:1–32.
89. Levcopoulos C, Petersson O. Splitsort—an adaptive sorting algorithm. Inf Process Lett. 1991;39:205–11.
90. The HDF Group. Hierarchical Data Format, version 5 [Internet]. Available from: http://www.hdfgroup.org/HDF5/. [Cited 2022 Apr 11].
91. Fan B, Andersen DG, Kaminsky M. MemC3: Compact and concurrent MemCache with dumber caching and smarter hashing. Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13). 2013; 371–84.
92. Xiaozhou Li Princeton University, David G. Andersen Carnegie Mellon University, Labs MKI, Michael J. Freedman Princeton University. Algorithmic improvements for fast concurrent Cuckoo hashing [Internet]. ACM Conferences. https://doi.org/10.1145/2592798.2592820. [Cited 2022 Sep 13].
93. Shoshany B. A C++17 thread pool for high-performance scientific computing [Internet]. arXiv [cs.DC]. 2021. Available from: http://arxiv.org/abs/2105.00613. [Cited 2022 Sep 13].

Publisher's Note

MoDLE: High-performance stochastic modeling of DNA loop extrusion interactions

Supplementary text

1. MoDLE: design overview

One of the motivations for developing MoDLE is the need for a high-performance in-silico model of DNA loop extrusion. In this day and age writing high-performance software almost always requires developers to target parallel and heterogeneous hardware architectures, as a single-threaded, sequential application can only take advantage of a miniscule fraction of the performance offered by modern hardware. Given this, it was clear from the beginning that in order to meet our performance target we would need to design MoDLE so that it could efficiently run on machines with many CPU cores, ideally splitting the computation in many small independent work units that can be executed on different CPU cores requiring little or no synchronization.

The version of MoDLE presented in this manuscript uses a producer-consumer architecture, where a single producer communicates with multiple consumers through asynchronous message passing.

In this programming model we have a single thread with the role of producer, here and onwards referred to as the main thread. The main thread is responsible for reading input files and generating a set of simulation tasks to be consumed by a pool of worker threads. Tasks are implemented as light-weight C++ structs, and are extremely cheap to generate and consume. A single task contains all the information needed by a worker thread to run a simulation instance, that is simulating DNA loop extrusion on a single chromosome in a specific cell. Simulation instances are for the most part independent from each other and can thus run concurrently. This means that when simulating loop extrusion on the human genome with default settings using 512 cells, MoDLE could in theory run all 11776 simulation instances in parallel. This level of parallelism is orders of magnitude larger than what is available on modern high-performance servers and workstations. This means that MoDLE can efficiently take advantage of all CPU cores available on the host machine, and this is likely to remain true for the foreseeable future.

Another aspect we had to consider during MoDLE's design was memory usage and layout. Modern CPUs take advantage of several techniques such as instruction pipelining, out-of-order and speculative execution to greatly improve the instruction throughput of a single CPU core. Because of this, the bottleneck of many applications running on modern CPUs is rarely instruction throughput, and is instead memory bandwidth. The most important aspect to consider when optimizing a memory-bound application is memory locality of reference, so that the CPU prefetcher can efficiently predict and prefetch data the application will access next, making efficient use of the CPU cache and hiding memory access latencies.

With this in mind we identified a data topology that would minimize the memory used by a

single simulation instance without significantly compromising locality of reference (see Section 3).
Data stored by MoDLE in system memory can be partitioned into two logical classes: data shared across multiple worker threads and data that is owned and accessed by a single worker thread.
A small fraction of shared data is immutable, and can thus be accessed concurrently from different threads without requiring any synchronization. Examples of shared immutable data are the list of chromosomes that comprise the genome that is being simulated or the position, direction and occupancy of extrusion barriers mapping on a given chromosome (see Section 6).
Extrusion barrier state (i.e. occupied or not occupied), LEF positions and states are instead examples of data that must be stored locally by each worker thread (see Sections 5 and 6). MoDLE does not explicitly model DNA, as this is not necessary with the current design, and would greatly increase MoDLE's memory footprint. Instead, for each chromosome MoDLE only tracks chromosome name, total length and begin/end position (in case chromosomes subsets are being simulated). Extrusion barriers and LEFs are both stored in contiguous memory. Extrusion barriers are sorted by their genomic coordinates, so visiting them in 5'-3' or 3'-5' direction is trivial and can be done in $O(n)$. Sorting needs to be done only once, as extrusion barrier positions are immutable throughout the entire simulation (see Section 6). LEFs are also stored in contiguous memory but in unspecified order (see Section 5). At the beginning of every epoch, MoDLE builds two indices such that visiting extrusion units moving in forward and reverse direction sorted by their genomic coordinates becomes trivial and can be done in $O(n)$ (see Section 8).
With the ability of visiting extrusion units sorted by their genomic coordinates, all kinds of collision events can be detected by comparing the distance of adjacent elements (e.g. an extrusion unit moving in forward direction and the first extrusion barrier located downstream of said extrusion unit) with the distance the extrusion unit is set to travel during the epoch that is being simulated (see Section 12). If the proposed move is greater than the distance that separates the two entities, then MoDLE predicts a collision may occur in the current epoch. Whether or not a collision actually takes place is determined by a stochastic process (usually a Bernoulli trial). Collisions are processed one type at a time (e.g. first we process collisions between extrusion units and extrusion barriers, next we process collisions between extrusion units moving in opposite directions and so on). For each extrusion unit MoDLE records whether a collision occurred as well as a reference to the entities involved in the collision event (see Section 7). MoDLE also tracks instances where a collision was avoided due to stochastic events (e.g. because two colliding extrusion units were able to bypass each other). Once all collisions have been detected, the proposed moves are updated to satisfy the constraints introduced by collision events (see Sections 12a and 12e-g). The updated moves are then used to update the simulation state before registering contacts, releasing a fraction of the active LEFs, and moving to the next epoch(see Section 6).

The majority of MoDLE's memory budget is used to track molecular contacts generated by DNA loop extrusion. Molecular contacts are stored using a specialized contact matrix data structure described in Section 2, which was developed with three aims: enabling efficient pixel random access, thread safety and low memory overhead.
Storing a single instance of the contact matrix for each chromosome that is being simulated and sharing it across all worker threads allows MoDLE to minimize the amount of memory required to store the state of a single simulation instance. The size of this state varies from

500 KB or less to 5-10MB depending on the number of LEFs and extrusion barriers involved in the simulation instance (see Section 3). The size of the shared contact matrix depends on chromosome size, width of the region along the diagonal that is being represented as well as bin size. With default settings, the contact matrix for chromosome 1 from the human genome requires approximately 120 MB of memory (see Section 2).

2. MoDLE: contact matrix

The main goal when designing the specialized contact matrix data structure used by MoDLE was to take advantage of the nature of DNA loop extrusion to minimize the memory footprint of the contact matrix as well as making most of the class public API thread-safe.
The main strategy used to limit the memory usage of the contact matrix class is to avoid representing regions in the matrix where loop extrusion is unlikely to generate contacts. With default setting the contact matrix is only representing contacts for a 3 Mbp window around the diagonal of the upper triangle of the contact matrix. When a simulation instance tries to set or increment a value outside of the space represented by the contact matrix, the action is effectively a no-op and will lead to a missed update. The number of missed updates are tracked using an atomic counter and a warning is issued if the fraction of missed updates becomes significant. Attempts to fetch the value of a pixel that lies outside of the 3Mbp window is also a no-op and always returns zero.
Using this strategy, storing contacts for chromosome 1 of the human genome using a bin size of 5000 bp and 32-bit integers requires around 120 MB of memory while storing the entire square matrix would require close to 10 GB.
The public API of the contact matrix class is rendered thread-safe through mutex locking. When instantiating the contact matrix class with a certain shape (i.e. the number of rows and number of columns, where the number of rows represents at most 3 Mbp of DNA with default settings), the class constructor will also allocate a vector of mutexes that will be used to protect subsets of the contact matrix from concurrent access from different threads. Access to a given pixel is protected by a single mutex. Given the pixel corresponding to row r and column c, to identify which mutex to acquire prior to accessing the pixels, the coordinates r and c are hashed. The output of the hash function is then divided by the number of mutexes protecting the current contact matrix instance, and the remainder of the division is used to index the mutex protecting the pixel that we are about to access. The exact number of mutexes protecting a contact matrix instance is computed based on the matrix shape and the number of available CPU cores. This number is always rounded to the next power of two so that the index of the mutex protecting a given pixel can be computed using bitwise operations instead of slower modulo operations. The number of mutexes is currently clamped between 2 and $1000 * t$, where t is the maximum number of CPU cores available on the machine where MoDLE is being executed. As an example, on server A (see Table 1), MoDLE allocates up to 256,000 mutexes for each contact matrix, using roughly 10 MB of memory.

3. MoDLE: Private state of a simulation instance

As soon as the first task is consumed by a given worker thread, the worker thread will allocate and initialize the memory required to store the private state of a single simulation instance. The state object itself consists of a struct with several member variables allocated on the stack, such as a 64-bit integer field used to keep track of the simulation epoch and the state of the pseudo-random number generator used by the current simulation instance (see Section 4). The state object also owns and manages a dozen buffers that are used for various purposes throughout the simulation. These buffers are allocated once, then cleared and resized when starting a new simulation instance after reading a new task. The majority of these buffers have a size that depends on the number of LEFs that are being simulated. Given that most of the time chromosomes are processed from the largest to the smallest, and that in MoDLE the number of LEFs is a function of chromosome size, each worker thread is expected to perform several relatively large allocations when processing the first task, and then progressively shrink the buffers, which is a cheap operation to perform, especially compared to growing buffers. This memory allocation strategy allows MoDLE to greatly reduce memory re-allocations, preventing performance issues due to frequent memory allocations and memory fragmentation.

4. MoDLE: PRNG and simulation reproducibility

The PRNG algorithm used by MoDLE to model stochastic events can be customized at compile-time. By default MoDLE will be compiled to use xoshiro256++ [88] as PRNG. Alternatively, the Boost or STL implementation of the Mersenne-Twister algorithm (64-bit) can be used.
Each worker thread owns an instance of the PRNG class. After consuming a new task from the queue, worker threads seed the PRNG using a seed computed by hashing chromosome and cell metadata (namely chromosome name, chromosome size and cell ID) with a user-provided seed (optional, specified through the --seed CLI option).
This seeding strategy not only makes simulation results reproducible across runs on the same machine (assuming simulation parameters are unchanged), but also across machines with a different number of CPU cores. It should be noted that result reproducibility is currently not guaranteed across operating systems and on machines using different standard library implementations. Users relying on result reproducibility are encouraged to use the Docker images available on GHCR.io and DockerHub at https://github.com/paulsengroup/modle/pkgs/container/modle or https://hub.docker.com/r/paulsengroup/modle

5. MoDLE: modeling LEFs

Loop extrusion factors are modeled as pairs of extrusion units, with one extrusion unit moving in reverse direction and the other in forward. In addition, each LEF keeps track of the epoch corresponding to the latest re-bind event.
Extrusion units are implemented as C++ structs consisting of a single integer representing their position along a chromosome and that implement most comparison operators to allow comparing LEF positions with that of other objects, such as extrusion barriers. Every simulation epoch MoDLE randomly samples a candidate moving distance for each extrusion unit. Moves are drawn from a normal distribution parameterized using the extrusion speed average and standard deviation. Candidate moves correspond to the maximum distance a given extrusion unit will cover during the current epoch (assuming there are no collision events). Upon activation, LEFs are bound to a random position on the chromosome that is

being simulated in the current simulation instance. Binding positions are sampled from a uniform distribution. At the end of every epoch a subset of LEFs are released from the DNA, so that they can be bound at a new location in the next epoch. Whether or not a specific LEF will be released is determined by a Bernoulli trial whose probability of success is computed based on the average LEF processivity and extrusion speed. This probability of success is also affected by whether a LEF is stalled on both sides and for which reason. As an example, with default settings, given a LEF whose extrusion units are stalled by a pair of extrusion barriers in convergent orientation, the probability of release for this specific LEF is reduced by a factor of 5.
With default settings extrusion units have a low probability of bypassing each other. This probability can be controlled through the CLI and can be set to 0 to forbid extrusion unit bypass.

6. MoDLE: modeling extrusion barriers

The version of MoDLE here presented models extrusion barriers based on the CTCF transcription factor using a two-state Markov process.
The two states of the Markov process represent the occupied and not-occupied states of a CTCF binding site by the CTCF transcription factor.
The state of each extrusion barrier at a given epoch is represented with a bitset and is stored outside of the extrusion barrier class and in the thread-local simulation state, so that extrusion barrier objects can be stored once, and shared across all simulation instances.
Each extrusion barrier consists of a position, blocking direction and two transition probabilities. An extrusion barrier instance thus loosely represents a CTCF binding site on the DNA, while its corresponding entry in the bitset signals whether a CTCF protein is bound to this specific binding site.
The recommended way of setting transition probabilities for extrusion barriers is using the data-driven approach described in the Method section, which relies on CTCF and Cohesin ChIP-seq data. Should this data not be available, users can directly specify the transition probabilities on the CLI through the options --extrusion-barrier-bound-stp and --extrusion-barrier-not-bound-stp. Alternatively, users can specify the --extrusion-barrier-occupancy option which is then used to compute the the occupied-to-occupied transition probability with the following formula:

$$p_{BB} = 1 - \frac{p_{UU} - (\pi_B - p_{UU})}{\pi_B}$$

where p_{BB} is the probability that a CTCF site that is currently bound by a CTCF protein will remain in this state in the current epoch, while p_{UU} is the probability that an unoccupied CTCF binding site will remain unoccupied throughout the epoch that is being simulated. Finally, π_B is the CTCF binding site occupancy.

7. MoDLE: tracking collisions

MoDLE defines four different kinds of collision events:

- LEF-extrusion barrier collisions (LB) - any collision occurring between a LEF extrusion unit and an extrusion barrier in the bound state.

- Primary LEF-LEF collisions (LL1) - collisions between two extrusion units belonging to two different LEFs and moving towards each other.
- Secondary LEF-LEF collisions (LL2) - collisions between two extrusion units belonging to two different LEFs that are moving in the same direction.
- Collisions between LEFs and chromosomal boundaries (CB).

MoDLE models chromosomal boundaries as impenetrable barriers to prevent LEFs from falling off chromosome ends. Collision events are stored in two vectors, one for extrusion units moving in reverse direction and the other for extrusion units moving forward.
Collision events are represented using unsigned 64-bit integers. Each integer encodes the cause of collision as well as a reference to the entity that caused the collision.
The high bits of a collision event are used to encode the collision type while the lower bits are used to encode the reference to the entity causing the collision. Usually, the reference is an index pointing into one of the vectors of extrusion units or barriers.
Using this encoding scheme makes collision encoding and decoding very efficient, as both operations can be implemented using bitwise operations such as bit shifting and masking.

	CO	CB	LB	LL1	LL2	Index			
bit idx	63	62	61	60	59	…	3	2	1	0
bit value	1	0	1	0	0	0	1	1	0	1

Legend:
CO - Collision Occurred (1 when collision occurred, 0 when collisions did not occur).
CB - Collision with Chromosomal Boundary.
LB - LEF extrusion barrier collision.
LL1 - LEF-LEF primary collision.
LL2 - LEF-LEF secondary collision.

The above example encodes a collision between the extrusion unit at index i and the extrusion barrier at index 13 (1101 in binary), where i is the index of the collision event in the collision vector.

8. MoDLE: indexing LEFs

MoDLE stores LEFs in contiguous memory in an unspecified order. Extrusion units moving in the same direction are indexed so that they can be visited in 5'-3' or 3'-5' direction.
Indexing is accomplished by indirectly sorting extrusion units by their positions and dealing with ties using the LEF binding epoch. Ties are rare events which can occur when a newly bound LEF is assigned a binding position that is already occupied by another extrusion unit.
Indexing is potentially an expensive operation due to the sorting step, as even state of the art general-purpose sorting algorithms run in $O(n \log n)$ where n is the number of LEFs in this case. MoDLE takes advantage of the fact that after the first epoch only a few index entries need to be updated to significantly reduce the time complexity of the sorting step. In other words, traversing extrusion units using the index from the previous epoch yields a sequence of extrusion units that is for the most part already in the correct order. Extrusion units are indirectly sorted using Splitsort [89], an adaptive sorting algorithm that can take

advantage of the level of pre-sortedness of extrusion units indexed using the index from the previous iteration. In brief, given a collection of items to be sorted, Splitsort removes a number of items from the collection, splitting the collection in two partitions: a partition containing already sorted items, and a partition with items yet to be sorted. The unsorted partition is then sorted using a traditional sorting algorithm (Quicksort in the Splitsort implementation used by MoDLE). Finally the two sorted partitions are merged back into the original sequence. Splitsort is not a stable sorting algorithm, meaning that the order of ties in the sorted collection is not deterministic. MoDLE uses LEF activation epoch to make Splitsort practically stable. Let us consider the following scenario: two LEFs l_1 and l_2 have been assigned to the same chromosome in the same simulation instance. At the end of epoch e_0, l_1 is active and extruding: its forward-moving extrusion unit ef_1 is located at position p. At this time, l_2 is inactive (i.e. not bound to the chromosome). During the next epoch (e_1), l_2 becomes active and is bound to position p, the same position as ef_1. In this scenario ef_1 and ef_2, the forward-moving units of l_1 and l_2 respectively are tied. The tie is resolved by observing that ef_2 became active after ef_1 reached position p, and that thus reached position p after ef_1. Because of this, ef_2 is ranked lower than ef_1. The case where two extrusion units become active at the same time and bind to the same position is not explicitly handled, as under normal circumstances these are extremely rare events.

9. MoDLE: generating molecular contacts

At the end of each epoch after the burn-in phase, MoDLE selects a number of LEFs by random sampling with replacement for contact registration. The genomic coordinates of these LEFs will be used to generate molecular contacts. Contact sampling rate can be tweaked through the --contact-sampling-interval parameter, which sets the average number of base pairs a LEF extrudes between two subsequent sampling events (assuming loop extrusion proceeds unencumbered). The default value for this parameter is 50 kbp.
MoDLE can sample two kinds of contacts:

- Molecular contact directly mediated by a LEF, that is contacts between the two regions brought together by the extrusion units of a LEF. This type of contact will very often occur in correspondence of stripes and dots, see Fig 2A.
- Contacts between regions within the same loop. This type of contact almost always occurs inside squares on the diagonal corresponding to TADs, see Fig 2B.

Furthermore, contacts can be randomized by adding noise sampled from a GEV distribution whose location, scale and shape parameters can be customized from the CLI.
At the time of writing, MoDLE supports the following contact sampling strategies:

- Loop only.
- TADonly.
- TAD + loop.
- Loop only with noise.
- TAD only with noise.
- TAD + loop with noise.

With default settings, MoDLE will, on average, sample 5 TAD contacts for every loop contact. This ratio was determined empirically, with the aim of generating contact matrices resembling that produced by Hi-C and Micro-C. The ratio between the two types of contacts

can be set through the --tad-to-loop-contact-ratio CLI option.
Let us consider LEF l with extrusion units e_1 and e_2, located ad positions p_1 and p_2 respectively, where $p_1 \leq p_2$.
To sample one loop contact for LEF l, random noise from GEV is added to positions p_1 and p_2 producing positions p_1^* and p_2^* respectively. In case contact randomization was disabled by the user, a noise of 0 is added. Next, positions p_1^* and p_2^* are mapped to their respective bins in the contact matrix b_1 and b_2 by dividing them by the bin size bs. Finally the value of the pixel corresponding to b_1 and b_2 is increased by 1.
The procedure to sample one TAD contact is identical to the one described above except for the last two steps. After computing p_1^* and p_2^*, MoDLE will sample two integral numbers between p_1^* and p_2^* from a uniform distribution, p_3^* and p_4^* respectively. p_3^* and p_4^* are then mapped to their corresponding bins b_3 and b_4, and the value of the corresponding pixel is increased by one.
In MoDLE v1.0.0-rc.7 the CLI option --track-1d-lef-position can be used to track LEF coordinates in 1D space. This information is written to disk in BigWig format. LEF occupancy is recorded similarly as for loop contacts, except that the two positions are mapped to their respective bins, and counts in a 1D vector corresponding to these two bins are incremented accordingly.

10. MoDLE: generating candidate moves

At the beginning of every epoch each active extrusion unit is assigned a candidate move. Moves are randomly sampled from a normal distribution with $\mu = \bar{s}$ and $\sigma = 0.05\mu$, where $\bar{s}$ corresponds to the average extrusion speed. All three parameters μ, σ and $\bar{s}$ can be tweaked through MoDLE's CLI. This strategy can lead to unlikely moves for consecutive extrusion units, which is why once all moves have been sampled, moves for consecutive extrusion units are adjusted as follows.
Let us consider two consecutive extrusion units ef_1 and ef_2 both moving in forward direction. Extrusion units are currently located at positions p_1 and p_2 respectively, and at a distance $d_{12} = p_2 - p_1$ with $p_1 < p_2$. Let us also consider two candidate moves m_1 and m_2 which have been assigned to ef_1 and ef_2 respectively.Applying these moves to their respective units will bring ef_1 and ef_2 to positions $p_1^* = p_1 + m_1$ and $p_2^* = p_2 + m_2$ respectively.
Finally let us stipulate that $p_1^* \geq p_2^*$ (i.e. ef_1 will surpass ef_2 by the end of the current epoch), and that ef_2 is the only entity located between positions p_1 and p_1^* (i.e. ef_1 cannot be stalled by an extrusion barrier). The scenario described above does not seem very plausible. A more plausible scenario is that after approaching ef_2, ef_1 pushes against ef_2, leading to a transient increase of the extrusion speed of ef_2, lasting until the end of the current epoch. To improve simulation realism, MoDLE, candidate moves are adjusted to reflect the second

scenario: m_2 is adjusted to $m_2 = p_1^* - (p_2 + 1)$, so that at the end of the current epoch ef_2 will be located immediately downstream of ef_1. A similar procedure is used to adjust the moves of adjacent extrusion units moving in reverse direction. The above procedure is used to adjust moves for all pairs of consecutive extrusion units moving in the same direction. Finally, moves are clamped such that no extrusion unit will move past chromosomal boundaries.

11. MoDLE: burn-in phase

With the intent of progressively activate LEFs, as well as limiting the chance of having batches of LEFs in highly correlated states (e.g. groups of LEFs all being bound or released around the same epoch), MoDLE starts new simulation instances by running a burn-in phase.
The burn-in phase consists of two stages: a first stage where LEFs are activated and bound to chromosomes for the first time, and a second stage where all LEFs are active and extruding. During both stages molecular contacts are not recorded. Thus LEF states and positions during burn-in are not directly observable in the output contact matrix.
The burn-in phase is implemented as a bit of extra logic running at the beginning of every epoch. The duration of the first burn-in stage is controlled through the --burnin-target-epochs-for-lef-activation option, which controls the number of epochs over which LEFs are activated. By default --burnin-target-epochs-for-lef-activation is computed from the average LEF processivity and extrusion speed such that by the time the last LEF is activated, the first LEF to have been activated has been released 5 times (approximately 190 epochs with default settings).
Let us consider a chromosome c of size $s_c = 10\ Mbp$ with a LEF density $\lambda_{LEF} = 10\ LEFs\ /\ Mbp$. Let us also assume MoDLE was configured to activate LEFs during a period of 25 epochs, $\Delta A_{LEF} = 25$. In this scenario, during the first burn-in stage, MoDLE will activate on average 4 LEFs every epoch. The exact number of LEFs to be activated in a given epoch are sampled from a Poisson distribution with $\mu = \frac{\lambda_{LEF} s_c}{\Delta A_{LEF}}$. LEFs are activated by drawing a genomic coordinate on chromosome c from a uniform distribution, and binding both LEF extrusion units at that position (see Section 5).
The second burn-in stage begins as soon as all LEFs have been activated and have extruded some amount of DNA at least once. The purpose of the second burn-in phase is to let the state of LEFs that have been activated during the same epoch to de-correlate from one another, as well as allowing the formation of some large loops of DNA. MoDLE will execute the second burn-in phase for a variable number of epochs, until the average loop size has stabilized. Users can set an upper bound to the duration of the second burn-in phase through the CLI.
The stages through which MoDLE steps every simulation epoch are the same, regardless of whether or not the burnin phase has been completed. The only difference is that steps involved in contact registration are not executed during the burn-in phase.

12. MoDLE: automatic parameter adjustment

MoDLE v1.0.0-rc.7 introduces functionality to automatically adjust certain parameters when changing the average extrusion speed directly (i.e. through --rev-extrusion-speed or --fwd-extrusion-speed CLI options) or indirectly (e.g. by changing resolution, as the overall average extrusion speed is set to 1.6 times the resolution by default).
Automatic parameter adjustment is enabled by default, but can be disabled through the --no-normalize-probabilities CLI option.
As of MoDLE v1.0.0-rc.7, the following parameters are adjusted:

- --extrusion-barrier-bound-stp
- --extrusion-barrier-not-bound-stp
- --probability-of-lef-bypass
- --lef-bar-major-collision-prob
- --lef-bar-minor-collision-prob

To motivate the existence of this functionality, let us consider the following scenario.
A MoDLE simulation is started with the following CLI options:

- --extrusion-barrier-occupancy=0.5 (π_B)
- --extrusion-barrier-not-bound-stp=0.5 (P_{UU})
- --rev-extrusion-speed=4kbp (s_{REV})
- --fwd-extrusion-speed=4kbp (s_{FWD})
- --probability-of-lef-bypass=0.1 (P_{bypass})

In this scenario, active LEFs are expected to extrude an average of 8kbp of DNA per simulation epoch (assuming no collisions take place). With the above extrusion barrier transition probabilities, we can expect barriers to switch between the occupied and not-occupied states once every epoch (on average). Assuming LEFs being simulated are Cohesin-like (extrusion speed=2.1kbp/s [16]), this means extrusion barriers switch states once every 3.81 seconds. Following the same logic, LEF-LEF collisions are expected to resolve themselves within ten epochs on average, which amount to about 50 seconds.
Let us now consider another simulation scenario, where:

- --extrusion-barrier-occupancy=0.5
- --extrusion-barrier-not-bound-stp=0.5
- --rev-extrusion-speed=500bp
- --fwd-extrusion-speed=500bp
- --probability-of-lef-bypass=0.1

With an overall average extrusion speed of just 1kbp/epoch, extrusion barriers are now changing state once every 0.48 seconds, while LEF-LEF collisions are expected to only last 4.76 seconds on average. The two scenarios are clearly not equivalent, in fact simulating at lower extrusion speeds will lead to weaker stripes (especially those produced by already weak barriers).
The automated parameter adjustment here described addresses the above issue.
We introduce a normalization factor n, which is equal to twice the default extrusion speed ($n = 8000$). The factor n expresses that transition and collision probabilities specified through the CLI are expressed assuming unencumbered LEFs extrude an average of n base pairs of DNA every epoch. The actual transition and collision probabilities used by MoDLE are then recomputed based on the normalization factor.
We can now revisit the two scenarios described earlier.

In the first scenario probabilities require no adjustment, as the average extrusion speed is already 8kbp/epoch.
In the second scenario, probabilities need to be updated, as the average extrusion speed and n are different. As an example, to update P_{bypass}, first we compute the ratio between the average extrusion speed an the normalization factor n: $r = \frac{s_{REV} + s_{FWD}}{n} = \frac{500 + 500}{8000} = 0.125$, the adjusted P_{bypass}, P^*_{bypass} is then computed as $P^*_{bypass} = r \cdot P_{bypass}$, which is the probability of observing n independent events with equal probabilities.
For the extrusion barrier transition probabilities, first we compute P_{BB} from π_B and P_{UU}, then we compute the adjusted P_{BB} as $P^*_{BB} = e^{ln(P_{BB}) \cdot r}$. Finally we also compute the adjusted P_{UU}, P_{UU} from P^*_{BB} and π_B. Given that extrusion barriers are simulated with a 2 state Markov process, the transition probability adjustment is equivalent to modeling extrusion barriers using a continuous-time Markov chain.

13. MoDLE: detecting and processing collisions

a. Overview

MoDLE approach to simulate loop extrusion can be summarized as:

- Proposing a set of candidate moves (see Sections 5 and 9).
- Detecting collisions and adjusting the candidate moves to satisfy the constraints introduced by collision events (see Section 12).
- Extrude DNA (i.e. advance extrusion units by their respective adjusted moves).

Collision detection is by far the most complex part of the model. This is one of the reasons why MoDLE does not attempt to detect all kinds of collisions at the same time.
To detect whether a collision event has already occurred or will occur in the current epoch, candidate moves are summed to the positions of forward-moving extrusion units or subtracted to the position of units moving in reverse direction. Then, MoDLE checks whether advancing extrusion units in this way will cause units to collide with other entities (e.g. extrusion barriers or other extrusion units). MoDLE can be parametrized such that certain collision events are not deterministic. For example, with default settings, two extrusion units that are on track to collide with one another during the current epoch have a probability of bypassing each others $p_{bypass} = 0.1$. Whether or not a stochastic collision event will take place in the current epoch is determined by a Bernoulli trial with probability of success directly set or computed from the input parameters. The occurrence and avoidance of collision events is recorded in a collision mask using the approach outlined in Section 7.
For reasons that will become clear in the next paragraphs, the order in which collisions are detected and processed is important. In the implementation here presented collisions are processed in the following order:

- Detect collisions with chromosomal boundaries (i.e. extrusion units that are located at, or are about to reach the 5' or 3'-end of a chromosome).
- Detect collisions between extrusion units and extrusion barriers (LB collisions).
- Detect primary LEF-LEF (LL1) collisions, that is collisions between extrusion units belonging to different LEFs and moving towards each other.
- Correct moves for extrusion units involved in LB collisions.
- Correct moves for extrusion units involved in LL1 collisions.

- Detect secondary LEF-LEF (LL2) collisions and correct moves immediately. LL2 collisions are defined as collisions between extrusion units moving in the same direction and belonging to different LEFs.
- Deal with rare edge-cases that have not been properly handled in the previous steps.

b. Detecting extrusion units at chromosomal boundaries

MoDLE models the 5' and 3' ends of chromosomes as impenetrable barriers to prevent LEFs from falling off the simulated chromosome. Thanks to the LEF indexing strategy described in Section 8, detecting collisions with chromosomal boundaries is trivial. First, let us observe that collisions with the 5'-end can only involve extrusion units moving in reverse direction, while collisions with the 3'-end can involve forward-moving units only.
Thus, we can detect all collisions with the 5'-end by traversing reverse-moving extrusion units in 5'-3' order, and looking for extrusion unit er_i, which is the last extrusion unit in 5'-3' order that satisfies the following two conditions:

- After extrusion, er_i will be located upstream of unit ef_0, which is the first forward-moving extrusion unit in 5'-3' order.
- er_i position after extrusion will be equal to p_{5P}, the position of the 5'-end (i.e. position 0 when simulating entire chromosomes).

The first condition is necessary, as if there are one or more extrusion units moving in opposite direction between er_i and the 5'-end, then it is possible that er_i will be stalled by a LL1 collision (see Section 12d). Note that the second condition does not handle the case where $er_i < p_{5P}$, as this cannot happen (see Section 10).
The same logic can be applied to detect collisions with the 3'-end.Collisions detected in this step are registered in the appropriate collision mask (see Section 7). The index portion of the encoded collision event is set to 5 and 3 for collisions with the 5'-end and 3'-end respectively.

c. Detecting LEF-barrier (LB) collisions

To simplify the implementation of this step, MoDLE first detects LB collisions involving reverse-moving extrusion units, and later LB collisions for forward-moving units.
Let us consider the algorithm to detect LB collisions with reverse units. A similar algorithm is used to detect LB collisions involving forward-moving units. Throughout the rest of the section, the term extrusion unit implicitly refers to reverse-moving extrusion units (unless otherwise specified).
The algorithm for LB collision detection for reverse-moving units operates on the following data:

- E - Vector of extrusion units. Units are stored in an unspecified order.
- I - Index vector used to traverse units in E sorted by their genomic coordinates.
- M - Vector of candidate moves.
- B - Vector of extrusion barriers sorted in 5'-3' direction.
- BS - Bitset representing extrusion barrier states (i.e. whether a given barrier is occupied in the current epoch and simulation instance).
- C - Collision mask.

All of the above vectors except B and B_S have size n, where n corresponds to the number of LEFs that are being simulated in the current simulation instance. B and B_S are of size m, where m is equal to the number of extrusion barriers mapping on the chromosome that is being simulated.
Finally let us consider indices i, j and k, which point to extrusion barrier B_i, index I_j and extrusion unit E_k respectively.
First, indices i, j and k initialized such that $i = 0$, $j = 0$ and $k = I_j = I_0$.
Next, we enter a loop that continues until $i = n$ or $j = m$.
Each loop iteration the following steps are performed:

1. If barrier B_i is not occupied (i.e. $BS_i = 0$), jump to step 8.
2. Index j is incremented until $k = I_j$ points to the first extrusion unit located downstream of B_i.
3. Compute distance $d_{E_k B_i}$ between unit E_k and barrier B_i as $d_{E_k B_i} = E_k - B_i$.
4. Distance $d_{E_k B_i}$ is compared with move M_k, which is the candidate move for extrusion unit E_k.
5. If $M_k < d_{E_k B_i}$, then E_k will not collide with B_i in the current epoch and no further processing of this pair of extrusion barrier and unit is needed. Continue from step 8.
6. If $M_k \geq d_{E_k B_i}$, then it is possible that a collision between E_k and B_i will occur during the current epoch. Whether or not a LB collision will occur depends on the outcome of a Bernoulli trial with probability of success p_S, where a success indicates a collision will occur and a failure indicates that the collision will be avoided. p_S is equal to $p_{block\ major}$ if the motif underlying B_i points against the direction of extrusion and $p_{block\ minor}$ otherwise. With default settings $p_{block\ major} = 1$ and $p_{block\ minor} = 0$, meaning that the outcome of the Bernoulli trial is deterministic, and only depends on barrier and extrusion directions.
7. Collision occurrence or avoidance is recorded in the collision mask C at entry C_k. In this case, the index portion of the encoded collision C_k is set to j, so that later simulation stages can inspect the collision mask and determine which extrusion barrier caused the collision.
8. Increment index i by one and continue from step 1.

Notice how indices i and j are never decremented, meaning that extrusion units and barriers are visited at most once, leading to a worst-case performance of $O(n + m)$.
A similar procedure is followed to detect LB collisions involving forward-moving extrusion barriers. In this case extrusion units are processed in 3’-5’ order.

d. Detecting primary LEF-LEF (LL1) collisions

The algorithm for LL1 collision detection for reverse-moving units operates on the following data:

- ER and EF - Vector of reverse and forward-moving extrusion units respectively. Both vectors store units in an unspecified order.
- IR and IF - Index vectors for units from ER and EF respectively.
- MR and MF - Vector of candidate moves for ER and EF respectively.
- B - Vector of extrusion barriers sorted in 5'-3' direction.
- CR and CF - Collision masks for ER and EF respectively.

All of the above vectors except B have size n, where n corresponds to the number of LEFs that are being simulated in the current simulation instance. B is instead of size m, where m is equal to the number of extrusion barriers mapping on the chromosome that is being simulated.
Let us also consider indices i and j, which will be used to traverse extrusion units in 5'-3' order using IR and IF respectively.
First, i and j are initialized such that $i = IF_0$ and $j = IR_0$.
Next, we enter a loop that will continue until the end of ER or EF is reached. In the loop, the following steps are performed:

1. Advance index j until j points to the first extrusion unit ER_j that is located downstream of EF_i.
2. Advance index i until i points to the last extrusion unit EF_i that is located upstream of ER_j.
3. Compute distance $d_{ER_jEF_i}$ between units ER_j and EF_i using $d_{ER_jEF_i} = ER_j - EF_i$.
4. Compare distance $d_{ER_jEF_i}$ with the sum of candidate moves MR_j and MF_i.
5. If $MR_j + MF_i < d_{ER_jEF_i}$, then a LL1 collision between ER_j and EF_i during the current epoch is not possible. Continue from step 1.
6. If $MR_j + MF_i \geq d_{ER_jEF_i}$, then a LL1 collision between ER_j and EF_i may take place during the current epoch. This depends on the outcome of a Bernoulli trial with probability of success $p_S = 1 - pb_{LEF}$, where pb_{LEF} is the probability of LEF bypass ($pb_{LEF} = 0.1$ with default settings). If the Bernoulli trial has a negative outcome, then the LL1 collision was avoided. Register collision avoidance in the appropriate collision mask and continue from step 1.
7. The next depends on the current state of units ER_j and EF_i, more specifically on whether they can be moved, (i.e. whether one or both of them are already involved in a LB or LL1 collision).
    a. If both units are free to move, register a LL1 collision in the collision masks for entries CR_j and CF_i using i and j for the index part of the encoded collision respectively (i.e. record that CR_j is colliding with CF_i and vice versa). Finally, jump to step 1.
    b. If only one of the colliding extrusion units can be moved (i.e. one of the extrusion units is already involved in another collision), then only register a LL1 collision for the extrusion unit that is free to move.
    In reality this step is more complicated than this, because we need to handle

an edge case where a LB collision needs to be converted to a LL1 collision. This can happen when ER_j and EF_i are located on opposite sides of an extrusion barrier b, and one of them is being stalled by the extrusion barrier b. Depending on the effective extrusion speed of the two units, the LL1 collision may occur before the LB does. If this is the case, then the LB collision should be replaced with an LL1 collision.

Notice how also in this case, i and j are never decremented, meaning that extrusion units are visited at most once, leading to a worst-case performance of $O(n)$.

e. Correcting moves for units involved in LB collisions

Correcting moves to satisfy the constraints introduced by LB collisions is fairly trivial. All we have to do is to loop over the collision masks for reverse and forward moving extrusion units, MR and MF respectively, and look for entries where the appropriate bits are set, namely the CO and LB bits (see Section 7), and adjust the corresponding candidate move.
Let us consider the case where we are adjusting moves for extrusion units moving towards the 5'-end. In this step we operate on the following data:

- E - Vector of extrusion units. Units are stored in an unspecified order.
- M - Vector of candidate moves.
- B - Vector of extrusion barriers sorted in 5'-3' direction.
- C - Collision mask.

Let us stipulate that entry C_i in the collision mask C for $i = 5$ has the following value (see Section 7 for more details on the collision encoding scheme):

	CO	CB	LB	LL1	LL2	Index (j)
bit idx	63	62	61	60	59	58-0
bit value	1	0	1	0	0	15 (decimal)

This entry encodes a LB collision between the reverse unit E_i and extrusion barrier B_j.
With this information, we can now look up the positions for E_i and B_j, and compute the corrected move M_i^* as $M_i^* = E_i - B_j - 1$. Advancing unit E_i by M_i^* will thus cause E_i to be located 1 bp upstream of barrier B_j.
The same process is repeated for the remaining reverse-moving extrusion units as well as for the forward-moving extrusion units.

f. Correcting moves for units involved in LL1 collisions

The procedure to correct moves for extrusion units involved in LL1 collisions is similar to the one described in Section 12e, with two important differences. The first difference is that we are now interested in collision mask entries where both the CO and LL1 bits are set. Second, computing the corrected moves for a pair of colliding extrusion units is a bit more involved.

Let us consider two extrusion units e_R and e_F that are involved in the same LL1 collision event. Unit e_R is located at position $p_R = 500\ bp$ and is moving towards the 5'-end, while unit e_F is located at position $p_F = 150\ bp$ and is moving towards the 3'-end. Let us also stipulate that the candidate moves for e_R and e_F are $m_R = 250\ bp$ p and $m_F = 200\ bp$ respectively. The position of collision is then calculated with $p_C = p_F + \left(m_F \frac{m_R - m_F}{m_R + m_F}\right)$, and the corrected moves m_R^* and m_F^* are computed as:

$$m_R^* = \begin{cases} p_C + 1 & \text{if } p_C = p_F \\ p_C & \text{otherwise} \end{cases}$$

$$m_F^* = \begin{cases} p_C & \text{if } p_C = p_F \\ p_C - 1 & \text{otherwise} \end{cases}$$

g. Processing secondary LEF-LEF (LL2) collisions

In contrast to previous collision detection steps, candidate moves of extrusion units involved in LL2 collisions are corrected immediately after they are detected.
First we process LL2 collisions for reverse-moving extrusion units, and later those involving forward-moving units.
Unless otherwise specified, data and operations outlined in this section are assumed to refer to extrusion units moving in reverse direction.
This step operates on the following data:

- E - Vector of extrusion units. Units are stored in an unspecified order.
- I - Index vector.
- M - Vector of candidate moves.
- C - Collision mask.

In brief, to detect LL2 collisions we loop over consecutive extrusion units in 5'-3' order, and we detect instances where the downstream unit is set to bypass the upstream unit. Given how candidate moves are generated (see Section 10), this can only occur if the upstream unit is being stalled.
Let us consider indices $i = 1$, $j = I_{i-1}$ and $k = I_i$, which will be used to index extrusion units E_j and E_k. Let us also consider the candidate moves and entries in the collision mask for E_j and E_k, M_j, M_k and C_j, C_k respectively.
Next, we enter a loop that continues until $i = n$, where n is the number of LEFs in the current simulation instance.
Each loop iteration, the following steps are performed:

1. Keep advancing i until index j points to a stalled extrusion unit and k points to an extrusion unit that is free to move.
2. Compute the position after extrusion for units E_j and E_k: $p_j^* = p_j - M_j$ and $p_k^* = p_k - M_k$, where p_j and p_k are the current positions of E_j and E_k.

3. If $p_j^* < p_k^*$, then a LL2 collision is not possible. Continue from step 1.
4. $p_j^* \geq p_k^*$, then we a LL2 collision between E_j and E_k may occur during the current epoch. Whether or not this is the case, is determined based on the outcome of a Bernoulli trial with probability of success $p_S = 1 - pb_{LEF}$, where pb_{LEF} is the probability of LEF bypass ($pb_{LEF} = 0.1$ with default settings). If the Bernoulli trial has a negative outcome, then the LL1 collision was avoided. Record collision avoidance in the collision mask and continue from step 1.
5. If the Bernoulli trial from step 4 is successful, then register a LL2 collision with E_j in entry C_k in the collision mask C.
6. Lastly, compute the corrected move M_k^* with $M_k^* = p_k - p_j^* - 1$.

A similar procedure is used to detect LL2 collisions for forward-moving extrusion units.

After all LL2 collisions have been processed, we do a final pass over units involved in LL2 collisions to handle an edge-case not handled in the previous step.
Let us consider two consecutive extrusion units moving in forward direction, ef_1 and ef_2, with ef_1 located upstream of ef_2. ef_2 is stalled due to a LB collision with extrusion barrier b. Let us also stipulate that the position of ef_1 after move is greater than that of unit ef_2 after move, as well as that of barrier b: $p_{ef_1}^* > p_{ef_2}^*$ and $p_{ef_1}^* > p_b$. In this scenario, a LL2 collision may have occurred. However, if the LL2 collision was avoided, meaning that the candidate move for ef_1 has not been corrected, then extruding ef_1 would cause this unit to jump past barrier b (and any other obstacle between p_{ef_1} and $p_{ef_1}^*$ for that matter).

The final pass through units involved in LL2 collisions detect instances like the scenario outlined above, and corrects the move for ef_1 such that after extrusion, ef_1 will be located 1 bp downstream of ef_2.

MoDLE: High-performance stochastic modeling of DNA loop extrusion interactions

Supplementary figures

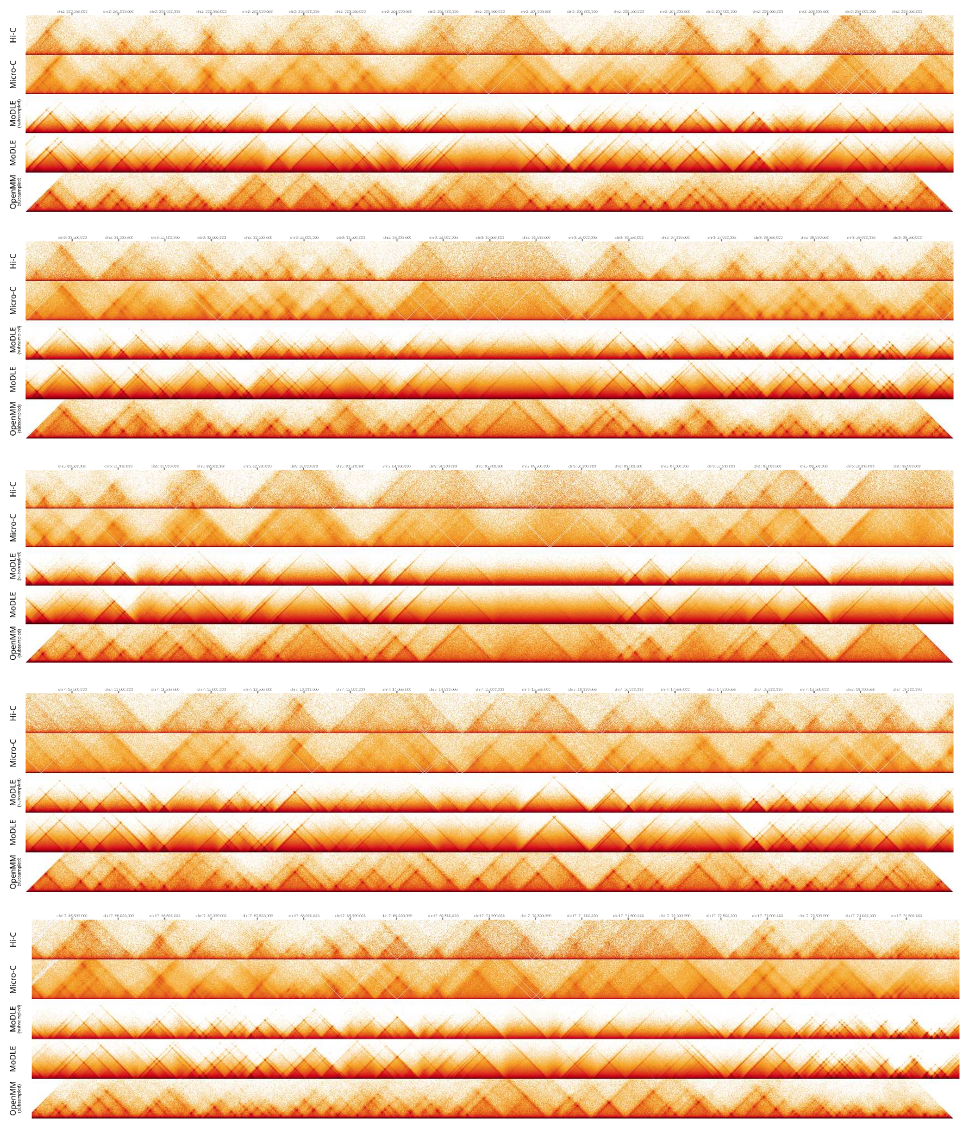


Fig. S1: Side-by-side comparison of Hi-C and Micro-C data (top two panels), MoDLE output with

and without subsampling (third and fourth panels respectively) and OpenMM output (bottom panels for all selected 10Mbp simulation regions on chromosomes 2, 3, 5, 7 and 17).

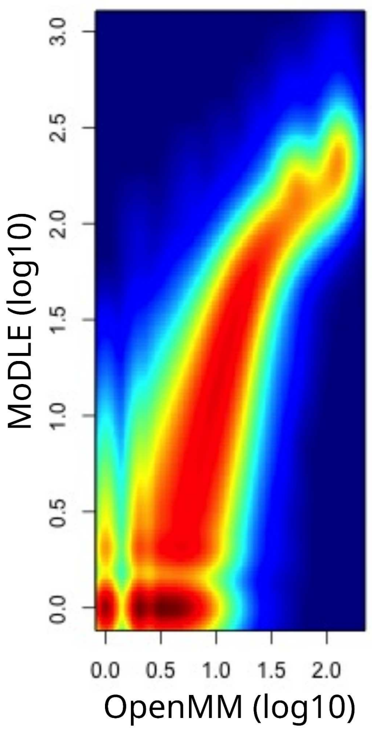


Fig. S2: Density plot showing frequency (blue = low; red = high) of pixel values in MoDLE (horizontal axis; log10-transformed) vs. OpenMM (vertical axis; log10-transformed). Pearson correlation coefficient is ρ=0.93.

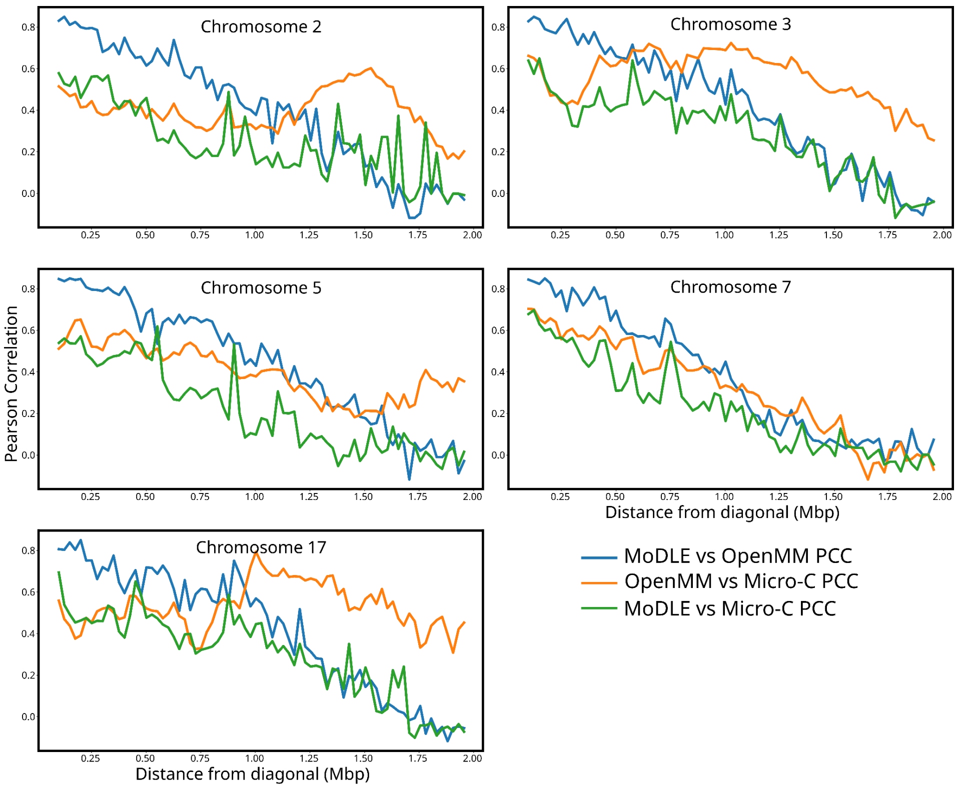


Fig. S3: Diagonal-by-diagonal correlation plots. Each panel shows the Pearson correlation of MoDLE vs. OpenMM (blue), OpenMM vs. Micro-C (yellow), and MoDLE vs. Micro-C (green) for all selected 10Mbp simulation regions on chromosomes 2, 3, 5, 7 and 17). The correlation is shown for all contact map sub-diagonals from the diagonal to 2Mbp from the diagonal (horizontal axes).

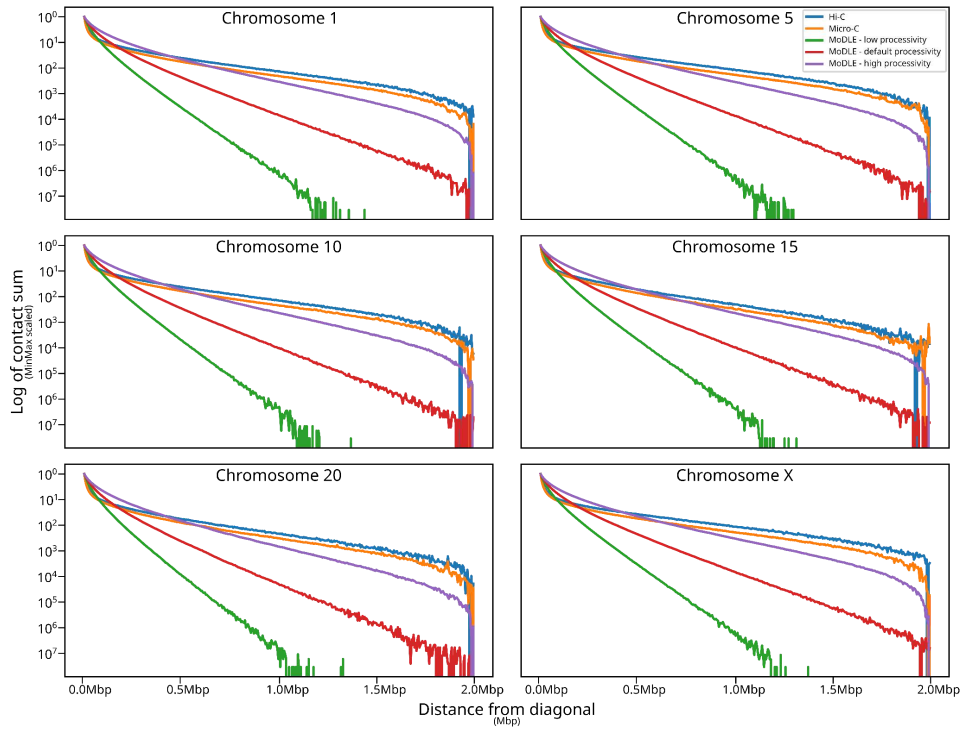


Fig. S4: Relationship between distance from diagonal (Mbp) and contact frequencies (log of contact sum) for hESC Hi-C data (blue), hESC Micro-C data (yellow), MoDLE using low processivity (150 kbp; green), MoDLE using default processivity (300 kbp; red), and MoDLE using high processivity (600 kbp; purple).

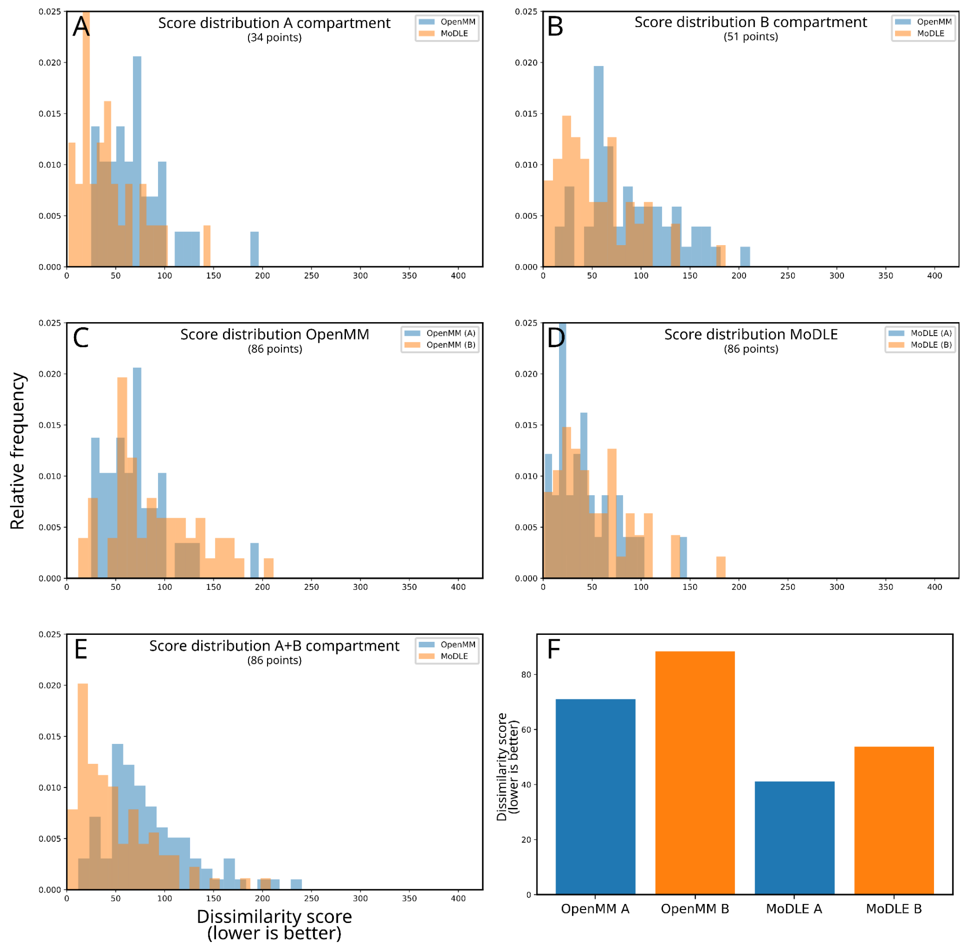


Fig. S5: Comparison of mean and distribution of dissimilarity scores of MoDLE and OpenMM simulations using H1 hESC Micro-C as reference.
Panel A and B contrast score distribution for MoDLE and OpenMM simulated matrices in A and B compartments respectively.
Panel C and D contrast score distribution of A and B compartments for OpenMM and MoDLE simulated matrices respectively.
Panel E shows the score distribution for OpenMM and MoDLE simulated matrices, regardless of compartment.
Panel F shows the average score for OpenMM and MoDLE simulations in A and B compartments.
Scores are computed as described in Methods (part 6). Before intersecting scores with compartment information, scores are filtered by taking the intersection of the score themselves, the architectural stripes on the reference matrix (stripes annotated with Stripenn) and the genomic regions simulated with OpenMM (chr2:200-210Mbp, chr3:30-40Mbp, chr5:80-90Mbp, chr7:10-20Mbp and chr17:65-75Mbp).

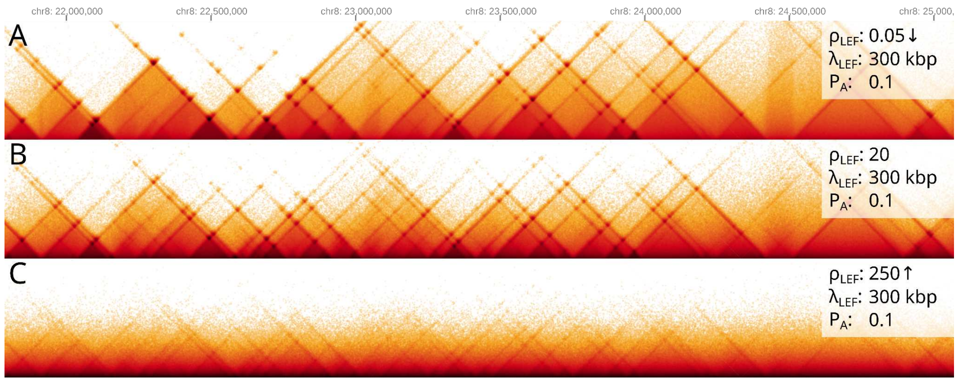


Fig. S6: Simulated heat maps showing the effect of increasing LEF density (ρ_{LEF}). Higher LEF densities cause contacts to distribute closer to the diagonal. This is due to an increased probability of LEF-LEF collisions. Furthermore, at very high ρ_{LEF} (panel C), stripes and dots appear less defined. This can once again be explained by an increased rate of LEF-LEF collisions, which are stochastic in nature, and do not necessarily occur at or near extrusion barriers.

Legend:
ρ_{LEF}: LEF density expressed as the number of LEFs per Mbp of simulated DNA
λ_{LEF}: Average LEF processivity
P_A: Probability of LEF-LEF collision avoidance

↓: Parameter is smaller than the default value
↑: Parameter is larger than the default value

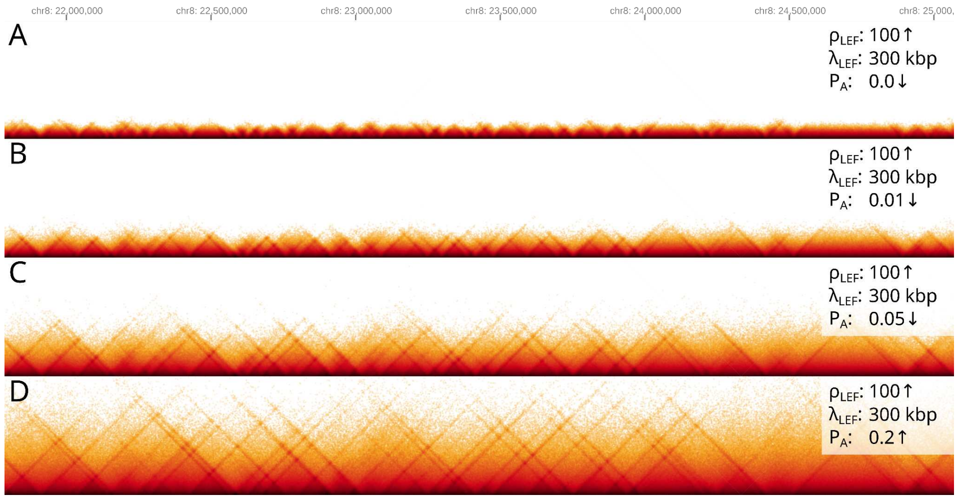


Fig. S7: Simulated heat maps showing the interplay between LEF density (ρ_{LEF}) and probability of LEF-LEF collision avoidance (P_A).
This scenario further expands what is shown in Fig. S4, and shows how increasing P_A can partially counteract very high LEF densities.

Legend:
ρ_{LEF}: LEF density expressed as the number of LEFs per Mbp of simulated DNA
λ_{LEF}: Average LEF processivity
P_A: Probability of LEF-LEF collision avoidance

↓: Parameter is smaller than the default value
↑: Parameter is larger than the default value

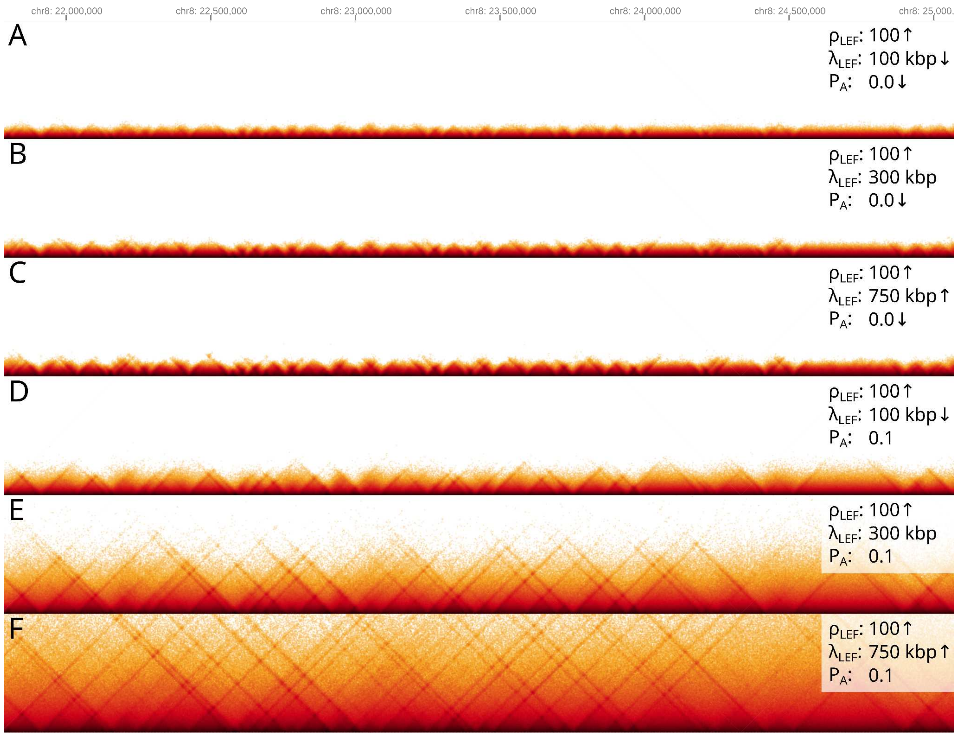


Fig. S8: Simulated heat maps showing the effect of (dis)allowing LEF-LEF collision avoidance at increasing levels of LEF processivity (λ_{LEF}).
Increasing λ_{LEF} has virtually no effect when the rate of LEF-LEF collisions is high and collisions are deterministic (i.e. LEF-LEF collisions cannot be avoided). Allowing LEFs to bypass one another with a relatively low probability is sufficient to observe progressively longer loops produced by increasingly higher λ_{LEF}.

Legend:
ρ_{LEF}: LEF density expressed as the number of LEFs per Mbp of simulated DNA
λ_{LEF}: Average LEF processivity
P_A: Probability of LEF-LEF collision avoidance

↓: Parameter is smaller than the default value
↑: Parameter is larger than the default value

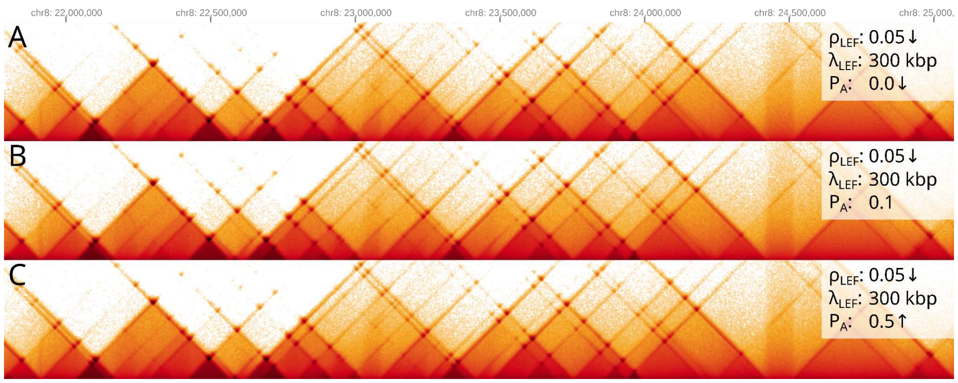


Fig. S9: Simulated heat maps showing the consequences of simulating with very low LEF densities (ρ_{LEF}).
When simulating with very low ρ_{LEF}, the probability of two or more LEFs extruding the same region at the same time is approximately 0. Thus, increasing P_A has no effect in practice.
This scenario also highlights one of the consequences of MoDLE simulations not accounting for polymer physics. Very low ρ_{LEF} would likely result in no appreciable contact enrichment due to loop extrusion in vivo or in MD polymer simulations, as once a loop has been extruded, there is plenty of time for the chromatin polymer to relax before the another LEF extrudes through the same region. This is not the case in MoDLE, as loops cease to exist as soon as LEFs are released.

Legend:
ρ_{LEF}: LEF density expressed as the number of LEFs per Mbp of simulated DNA
λ_{LEF}: Average LEF processivity
P_A: Probability of LEF-LEF collision avoidance

↓: Parameter is smaller than the default value
↑: Parameter is larger than the default value

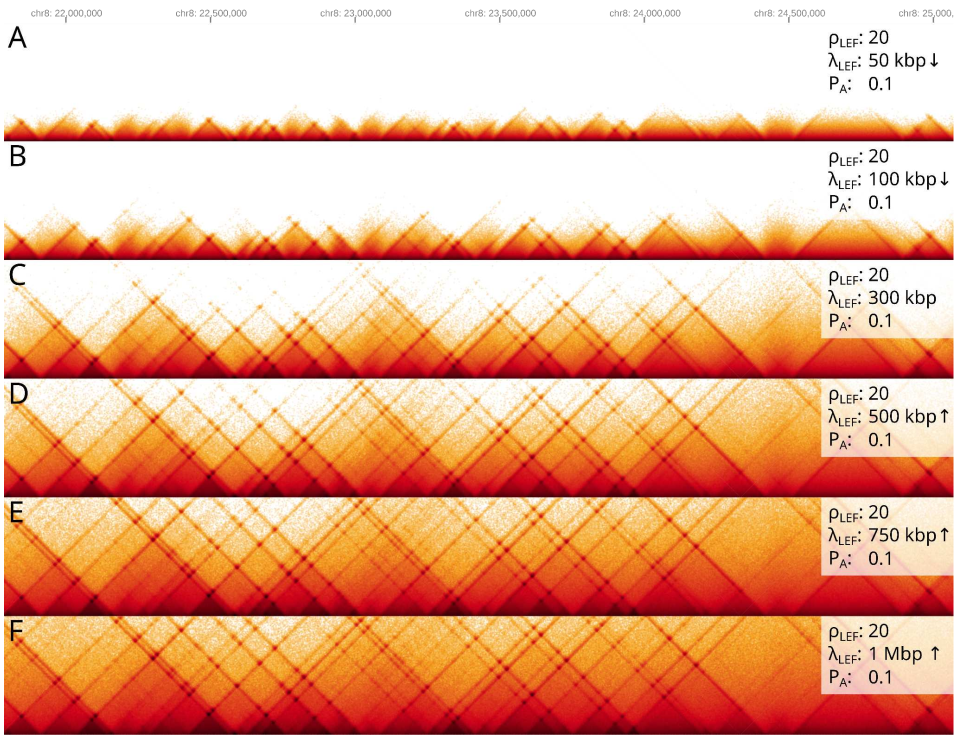


Fig. S10: Simulated heat maps showing the effect of different LEF processivities (λ_{LEF}) when LEF density (ρ_{LEF}) and probability of LEF-LEF collision avoidance (P_A) are left at default.
With realistic LEF concentrations, altering λ_{LEF} has the expected effect: higher processivities lead to loops, which result in longer stripes and stronger dots even at relatively high distances from the diagonal.

Legend:
ρ_{LEF}: LEF density expressed as the number of LEFs per Mbp of simulated DNA
λ_{LEF}: Average LEF processivity
P_A: Probability of LEF-LEF collision avoidance

↓: Parameter is smaller than the default value
↑: Parameter is larger than the default value

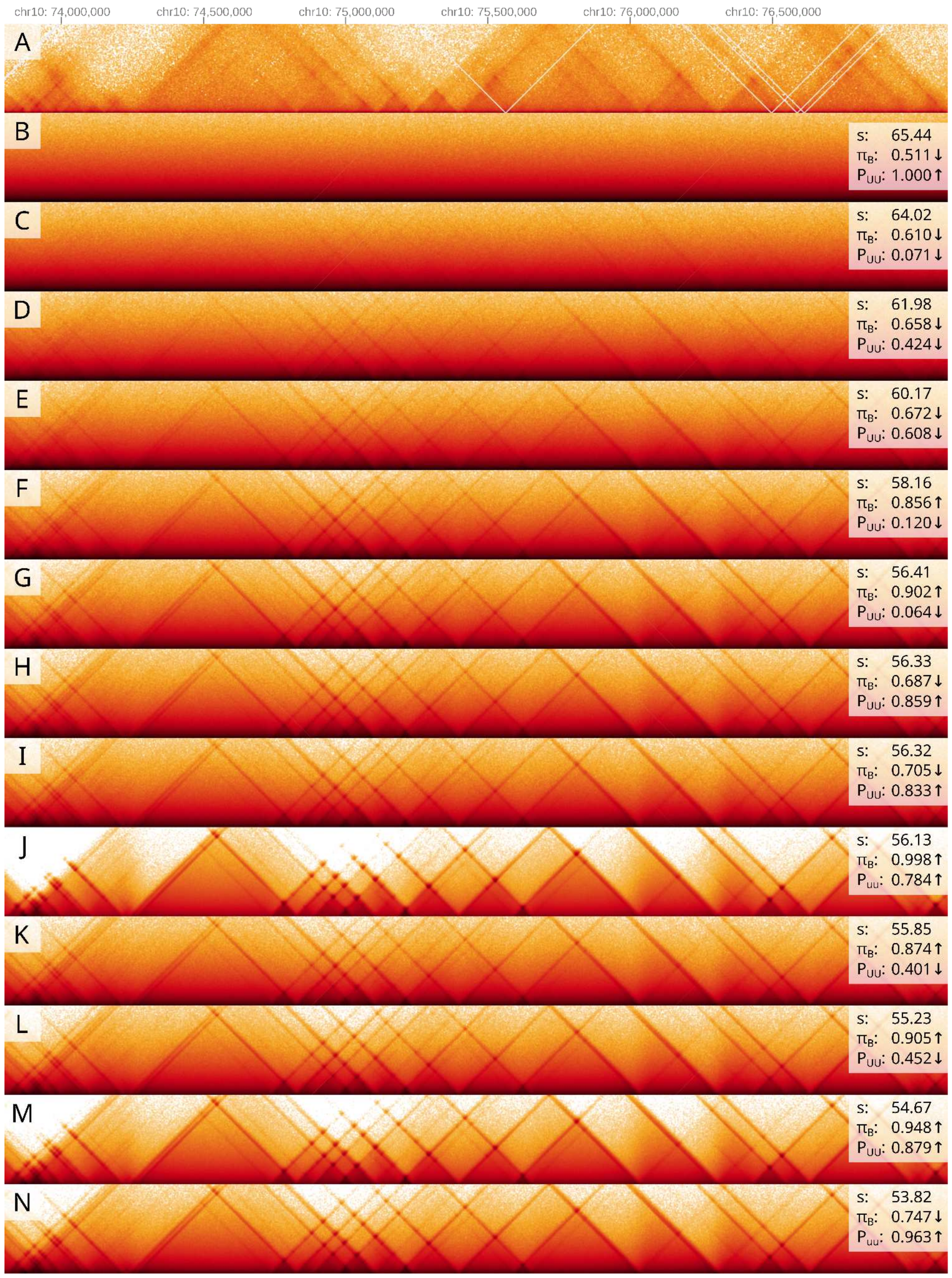


Fig. S11: Side-by-side comparison of H1-hESC Micro-C data (top panel) and matrices simulated by MoDLE. Panels B-N show a selection of the parameter combinations explored by the Gaussian optimization described in Methods (part 9).
On the right, each panel is labelled with its corresponding score s, occupancy π_B and self-transition probability of the unbound state P_{UU}.

Panels are sorted by score in descending order (lower scores are better).
Upwards and downwards arrows next to parameter values indicate whether a value is higher or lower than the default parameter.

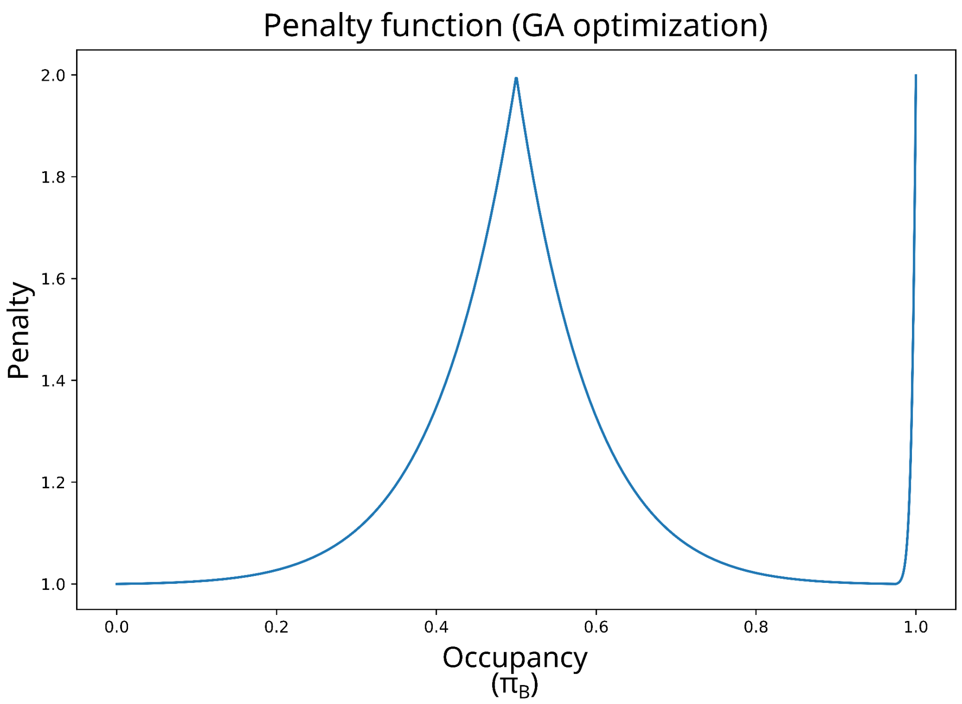


Fig. S12: Plot of the penalty function used in the GA optimization of extrusion barrier parameters.

MoDLE: High-performance stochastic modeling of DNA loop extrusion interactions

Supplementary tables

Table S1: Parameter table for MoDLE simulations used to generate Fig. 2.

Figure	**Bin size (bp)**	**Extrusion barrier occupancy**	**Target contact density**	**# of LEFs per Mbp**	**Diagonal width (Mbp)**	**Avg. LEF processivity (kbp)**	**Contact sampling strategy**
2A	5000 (default)	Variable	20.0	20 (default)	3 (default)	300 (default)	Loop only with noise
2B	5000 (default)	Variable	20.0	20 (default)	3 (default)	300 (default)	TAD only with noise
2C, 2D, 2F (WT)	5000 (default)	Variable	20.0	20 (default)	3 (default)	300 (default)	TAD plus loop with noise (default)
2F ΔCTCF	5000 (default)	0.2	20.0	20 (default)	3 (default)	300 (default)	TAD plus loop with noise (default)
2F ΔWAPL	5000 (default)	Variable	10.0	20 (default)	6	3000	TAD plus loop with noise (default)

Table S2: List of MoDLE software dependencies.

Name	**Project URL**	**Version**	**License**
Abseil C++	abseil.io	20211102 LTS	Apache 2.0
bitflags	github.com/m-peko/bitflags	1.5.0	MIT
Boost C++	boost.org	1.79.0	BSL 1.0
bzip2	sourceware.org/bzip	1.0.8	BSD-like

	2		
Catch2	github.com/catchorg/Catch2	3.1.0	BSL 1.0
CLI11	github.com/CLIUtils/CLI11	2.2.0	BSD 3
cmake-git-version-tracking	github.com/andrew-hardin/cmake-git-version-tracking	N/A	MIT
concurrentqueue	github.com/cameron314/concurrentqueue	1.0.3	Simplified BSD
cpp-sort	github.com/Morwenn/cpp-sort	1.13.0	MIT
fast_float	github.com/fastfloat/fast_float	3.5.1	MIT
fmt	github.com/fmtlib/fmt	9.0.0	MIT
HDF5 [90]	hdfgroup.org	1.12.2	BSD 3
libarchive	libarchive.org	3.6.1	BSD 2
libBigWig	github.com/dpryan79/libBigWig	0.4.7	MIT
libcuckoo [91,92]	github.com/efficient/libcuckoo	0.3.1	Apache 2.0
LZ4	lz4.org	1.9.3	BSD 2
LZO	oberhumer.com/opensource/lzo	2.1.0	GPL 2.0
range-v3	github.com/ericniebler/range-v3	0.12.0	BSL 1.0
readerwriterqueue	github.com/cameron314/readerwriterqueue	1.0.6	Simplified BSD
spdlog	github.com/gabime/spdlog	1.9.2	MIT
bshoshany-thread-pool [93]	github.com/bshoshany/thread-pool	3.3.0	MIT
TOML++	marzer.github.io/tomlplusplus	3.1.0	MIT
Xoshiro-cpp	github.com/Reputele	1.1	MIT

	ss/Xoshiro-cpp		
xxHash	xxhash.com	0.8.1	BSD 2
XZ Utils	tukaani.org/xz	5.2.5	GPL 3.0
zlib	zlib.net	1.2.11	Zlib
zstd	github.com/facebook/zstd	1.5.2	BSD 3

Paper II

hictk: blazing fast toolkit to work with .hic and .cool files

Roberto Rossini, Jonas Paulsen

Submitted for publication.
DOI: doi.org/10.1101/2023.11.26.568707

Paper III

Multi-level 3D genome organization deteriorates during breast cancer progression

Roberto Rossini, Mohammadsaleh Oshaghi, Maxim Nekrasov, Aurélie Bellanger, Renae Domaschenz, Yasmin Dijkwel, Mohamed Abdelhalim, Philippe Collas, David Tremethick, Jonas Paulsen

Submitted for publication.
DOI: doi.org/10.1101/2023.11.26.568711

III

Appendices

Appendix A

A slightly longer history of MoDLE

The current version of MoDLE (MoDLE v1.1.0) is a relatively large and complex C++17 project, counting close to 30 thousand lines of code. However, MoDLE started out as a much simpler project, counting less than 10 thousand lines of code.

In the initial version of MoDLE, v0.1.0 (09/03/2021), the genome was represented as a collection of chromosomes, where each chromosome consisted of a sequence of bins, each representing a fixed amount of DNA. Each bin could be bound by one or more extrusion barriers and LEFs. Informations regarding which molecules were bound to DNA was stored in the bin object itself. LEFs were modeled as a pair of extrusion units, where each extrusion unit knew its sibling unit, the bin to which it was currently bound and whether the unit was stalled by an extrusion barrier or another extrusion unit. Extruding DNA thus amounted to looping over each LEF, generating a candidate move and trying to move LEFs by that distance. If performing the proposed move resulted in a collision, the extrusion unit was assigned a number of iterations during which the extrusion unit was not allowed to move (i.e., it was stalled). This number was sampled from a geometric distribution $P(i|p) = p \cdot (1-p)^i$ with p equal to the probability of collision avoidance. Upon binding, each LEF was assigned a lifetime, that is a number of iteration the current LEF-DNA binding was supposed to last. LEF lifetime could be increased in case of LEFs stalled by a pair of convergent extrusion barriers. This was done to simulate the increased DNA-binding stability observed when cohesin is stalled by a pair of convergent CTCF sites.

This design was simple to understand and implement, however it was affected by several issues. First, performance was acceptable but not excellent, with most of the CPU time spent re-assigning extrusion units to different bins. Memory usage was also fairly large, and depended on the resolution at which loop extrusion was being simulated. This seemed incredibly wasteful, as on average only one every 20 bins was bound by a LEF, meaning that at any given time, most of the memory used to store genomic bins was used to represent genomic regions where nothing was happening as far as loop extrusion is concerned. Another issue was that the level of parallelism achievable with this implementation was limited by the number of chromosomes in a genome, as parallelizing past this level would require very fine locking and tight synchronization, likely resulting in a complex and only marginally faster implementation. Besides performance issues, the model logic was also affected by several issues. To explain one of the more severe issues, let us consider the following scenario. It is epoch 0. An extrusion unit EU_1 with lifetime 100 collides with extrusion barrier B_0 and is stalled for the next 10 iterations. Fast forwarding 10 iterations, EU_1 is now free to move past B_0, meaning that during epoch 10, barrier B_0 is non-blocking (i.e., is not bound by CTCF). In the same epoch, another extrusion unit EU_2 collides with barrier B_0 and is stalled for the next 7 epochs. The fact that an extrusion barrier can appear as occupied and unoccupied in the same epoch to different extrusion units brought the model to an inconsistent state.

Despite the outlined issues, MoDLE v0.1.0 was already capable of generating heatmaps that looked realistic and were able to recapitulate important features of Hi-C data, such as stripes and dots (see Figure 4.1). Not being happy with the computational performance of our model, we went back to the drawing board to develop MoDLE v0.2.0.

The differences between MoDLE v0.1.0 and v0.2.0 (21/03/2021) are several. First, we re-designed the in-memory representation of chromosomes. Second, we developed a new set of algorithms to process collision events. Third, we reworked the way simulation load was spread across multiple CPU cores. In MoDLE v0.2.0, chromosomes consisted of a chromosome name, length and a list of extrusion barriers, making memory usage proportional to the number of extrusion barriers instead that to the number of bins. This meant we could no longer depend on bin objects to keep track of extrusion unit positions, nor to detect collisions between LEFs and extrusion barriers. Instead, we developed a set of algorithms to detect collisions in linear time-complexity given two lists of genomic entities (i.e., extrusion barriers or units) sorted by genomic coordinates. The idea behind these algorithms is simple, and is based on the same idea of the merge step of the merge-sort algorithm.

Let us consider the following example. Given a sorted list of extrusion barriers $\overrightarrow{B}$ and a sorted list of extrusion units moving in forward direction $\overrightarrow{EU}$, we assume that forward-moving extrusion units can be blocked only by extrusion barriers blocking in reverse direction. Next, we initialize two pointers B_i and EU_i pointing to the beginning of $\overrightarrow{B}$ and $\overrightarrow{EU}$, respectively. Because we know that all units in $\overrightarrow{EU}$ are moving in forward direction, we know that no collisions can occur between barriers located upstream of EU_0 and EU_0, so we advance pointer B_i until the pointer points to the first barrier located downstream of EU_0. Next, we generate a candidate move m_i for EU_i, and we try to advance EU_i by m_i bp. If the B_i is located downstream of position $p_{EU_i} + m_i$, then the move is, for the time being, legal, as it doesn't cause any collision with extrusion barriers. However, if $p_{B_i} < p_{EU_i} + m_i$, then there is a possibility that extrusion barrier B_i will stall EU_i for k epochs, where k is a number sampled from a geometric distribution similarly to what was done in MoDLE v0.1.0. The number of epochs the extrusion unit EU_i will be stalled by barrier B_i is stored inside EU_i similarly to what was done in MoDLE v0.1.0. Once EU_i has been processed, we move onto EU_{i+1}, find the first barrier located downstream of EU_{i+1} and detect and process collision events as outlined above. A similar procedure is followed to detect collisions between extrusion barriers and extrusion units moving in reverse direction as well as collisions between extrusion units themselves. This means that detecting and processing collisions is now an operation with linear time-complexity, and that the most expensive operation is keeping extrusion units indexed (i.e., sorted by genomic coordinates).

The best general-purpose sorting algorithms have a time complexity of $O(n \log n)$. However, we reasoned that after the first indexing, extrusion units will for the most part already sorted, as ordering can be broken only by LEF rebinding and bypass events between consecutive extrusion units moving in the same direction. Thus, we looked for sorting algorithm that could take advantage of the high level of pre-sortedness of our data, and found a family of sorting algorithms named *Inv*-adaptive algorithms that were designed exactly for this type of applications. Given a collection of sortable objects, such as a list of integral numbers, the amount of work performed by *Inv*-adaptive algorithms is proportional to the number of inversions in the collection to be sorted. After benchmarking available *Inv*-adaptive sorting algorithms against general-purpose algorithms such as mergesort and quicksort, we decided to use drop-mergesort to sort extrusion units at the beginning of every simulation epoch.

Another important change introduced with MoDLE v0.2.0 is the notion of simulation instances. Instead of instantiating one simulation instance for each chromosome, MoDLE v0.2.0 instantiates multiple instances per chromosome, thus enabling efficient scaling and load-balancing when running simulations on multicore machines. This required developing a class implementing an in-memory, thread-safe interaction matrix such that multiple simulation instances simulating the same chromosomes could use a single interaction matrix instance to track simulated interactions. This means that the result of multiple, possibly concurrent, simulation instances, is aggregated into a single interaction matrix to achieve a result similar to that of bulk Hi-C, where interactions from multiple nuclei are combined into a single dataset.

While MoDLE v0.2.0 was significantly faster than v0.1.0 (took 4 minutes 6 seconds to simulate loop extrusion on hg38 using 256 CPU cores versus over 8 hours in v0.1.0), the model was still susceptible to falling in an inconsistent state due to the way collisions were handled.

It was in MoDLE v0.3.0 (14/06/2021), that the logic bug affecting previous versions was finally fixed. Addressing this bug required a significant rework of how extrusion barriers were modeled and how collisions were handled.

In previous versions, extrusion barriers were modeled as objects with a genomic position, direction and a probability of blocking (i.e., a probability of being bound by CTCF). When an extrusion unit collided with an extrusion barrier, the barrier used to sample a number from the geometric distribution, which corresponded to the number of iterations the extrusion unit will be stalled. Starting with MoDLE v0.3.0, extrusion barriers are modeled as a two-state (bound and unbound) Markov process. In this way, at any given time, an extrusion barrier is either bound by CTCF (i.e., blocking) it is not. Another change introduced by v0.3.0 was the removal of the concept of LEF lifetime. Instead of, upon LEF binding, computing the epoch in which a given LEF will be released, each LEF is now characterized by a probability of being released, which depends on the average LEF processivity. This probability is decreased when LEFs are stalled by a pair of convergent extrusion barriers. At the end of every epoch, k LEFs are selected for release by weighted random-sampling without replacement, where k is a random number sampled from a Poisson distribution $P(i|\mu)\,\frac{\mu^i}{i!}e^{-u}, i \geq 0$ with μ equal to the average number of LEFs to be released every epoch.

Given the introduction of the above changes, LEF-barrier and LEF-LEF collsions are now checked every iteration and stalls last a single epoch. We define three types of collisions: (1) LEF-barrier collisions; (2) primary LEF-LEF collisions; (3) secondary LEF-LEF collisions. Primary LEF-LEF collisions are collisions taking place between extrusion units moving in opposite directions, while secondary LEF-LEF collisions take place between extrusion units moving in the same direction. The latter type of collisions are called secondary because they are direct consequence of other of collision events.

Distinghushing between different kinds of collisions allowed us to refactor MoDLE collision handling into a detection and processing phase, which enabled us to properly handle complex collision events. First, we stipulate that LEF-barrier collisions have precedence over all other types of collisions and that primary LEF-LEF collisions have precedence over secondary LEF-LEF collisions. Next, we proceed to detect LEF-barrier and primary LEF-LEF collisions. The order in which collisions are detected is important, as when detecting LEF-barrier collisions, it is possible that some LEF-LEF collisions will be incorrectly labeled a LEF-barrier collisions.

Let us consider the following scenario. In a simulation instance, extrusion barrier B_i is located at position 100 and is oriented such that it blocks extrusion units moving in forward direction. In the same simulation instance there are two extrusion units EU_i and EU_j belonging to two different LEFs and moving in forward and reverse direction, respectively. Furthermore, both units are free to move and are located at position 50 and 101, respectively. EU_i is assigned a candidate move of 51 bp, while EU_j is set to move 100 bp. Given that we first detect collisions between LEFs and extrusion barriers, we may conclude that EU_i collides with barrier B_i, thus ending up at position 99, and then EU_j collides with unit EU_i, thus ending up at position 100. However, this would be incorrect, as assuming extrusion units are moving at uniform speeds, EU_j will cross barrier B_i before EU_i is able to reach it, and a primary LEF-LEF collision will occur at position 67 bp, which is computed as the collision point between two bodies moving towards each other at uniform speeds. Thus, after detecting LEF-barrier collisions and before correcting candidate moves, MoDLE detects primary LEF-LEF collisions, such that scenarios like the one outlined above are correctly handled. Collision events are stored in a vector using one integral number to encode each collision. Integers can then be decoded to get the collision type and the index of the entities involved in the collision event. After detecting LEF-barrier and primary LEF-LEF collisions, candidate moves are corrected to not violate the constraints imposed by collisions. Finally, secondary LEF-LEF collisions are detected and processed one by one.

Besides changes outlined above, this release also involved improvements to MoDLE's test suite, improvements to the CLI interface and various other small changes to MoDLE's algorithm. Most of the changes to the simulation algorithm were aimed at making MoDLE's output reproducible across different platforms as well as reworking the implementation of several stochastic processes, such as how candidate moves are generated. Performance of MoDLE v0.3.0 was in line with that of MoDLE v0.2.0, taking 4 minutes 49 seconds to simulate loop extrusion on hg38.

The simulation algorithm of MoDLE v0.3.0 was also implemented using CUDA. However, this implementation did not yield the expected performance improvements, and so the GPU implementation was abandoned in favor of the CPU one.

MoDLE v0.4.0 (09/07/2021) introduced a new mode of running MoDLE that was removed in later version. This new mode was named "perturbate" and allowed to run MoDLE on a subset of the genome, perturbating extrusion barriers to test the effects of the removal/addition of extrusion barriers on the resulting heatmap. For the sake of brevity, I will not discuss MoDLE perturbate further, as this mode was never part of a stable release of MoDLE.

MoDLE v0.4.1 (14/09/2021) introduced a new subcommand, MoDLE replay, that could be used to replay simulations generated by MoDLE perturbate. Similarly to MoDLE perturbate, I will not discuss this subcommand further, as its implementation was removed before the first stable release of MoDLE. Besides MoDLE replay, v0.4.1 introduced several change to make simulations behave as expected even when using unconventional simulation parameters (e.g., very low LEF density or very large extrusion speeds). This release also involved several minor performance optimizations and bugfixes.

After v0.4.1, we released 7 release candidates of MoDLE v1.0.0 where we fixed many bugs, added several features, and further improved performance. However, the core of MoDLE's simulation algorithm remained for the most part unchanged.

MoDLE v1.0.0-rc.7 (15/08/2022) is the version described in Paper I. From an algorithmic perspective, the main difference between MoDLE v0.4.1 and v1.0.0-rc.7 is how molecular interactions are sampled and collected.

The latest version of MoDLE is v1.1.0. This version includes several improvements to modle_tools, MoDLE's suite of helper tools, bugfixes, improvements to the CLI, and minor performance improvements. MoDLE v1.1.0 also supports writing LEF occupancy as 1D vector in bigWig format. Simulating loop extrusion on hg38 using 256 CPU cores with MoDLE v1.1.0 takes 2 minutes 59 seconds.

Appendix B

Improving existing file formats used to store Hi-C interactions

Both the .hic and cooler file formats (reviewed in Chapter 3) are valid format specifications, with implementations that can achieve good to excellent performance when reading and writing .hic and .cool files [60, 220]. However, after having implemented the hictk library to read both formats and write cooler files, I have developed some opinions on how each format could be improved.

.hic

The .hic format specification mandates that blocks of interactions should be compressed individually with zlib [258]. Zlib is a widely popular implementation of the DEFLATE compression algorithm that was developed in 1995 and has since been used to compress all kinds of data, including genomic data, image data, and web traffic. However, there are many alternatives to zlib, some trading compression ratio for (de)compression speeds and vice versa [259, 260]. Zstandard, also known as zstd was developed in 2016 by Facebook with the aim of achieving compression ratios similar to DEFLATE while providing much greater (de)compression speed. For this reason, in recent years, zstd replaced zlib in many applications, such as various parts of the Linux kernel and several components of web services hosted by Facebook and other big-tech companies. I believe zstd is particularly suitable for compressing interaction blocks in .hic files for the following reasons:

- zstd offers excellent compression and decompression speeds (540 MB/s and 1700 MB/s on the Silesia corpus, where zlib achieves throughputs of 95 MB/s and 400 MB/s).
- zstd offers 22 compression levels instead of the 9 levels offered by zlib. This could help finding a better compromise between compression and decompression speed.
- zstd can make use of a compression dictionary to greatly improve compression and decompression speeds as well as compression ratio when compressing small amounts of data. This could be leveraged by .hic by embedding the compression dictionary e.g., in the header section of the .hic file, and using smaller block sizes without compromising on compression ratio and (de)compression speed. Providing a good training set for zstd should be fairly simple, given that interaction blocks consist of triplets of numeric data with similar properties (e.g., positive, relatively small integer numbers).

The .hic format specification defines an efficient indexing scheme for blocks of interactions but does not specify how interactions should be stored inside individual blocks. In practice, JuicerTools and HiCTools, the only two software currently capable of creating .hic files, simply store interactions sorted by their genomic coordinates. I believe that there is a missed opportunity to also index interactions within blocks. My suggestion would be to sort interactions by genomic coordinates (like is already done in practice), and then to index each row using an index stored at the end of each block. Each block would then start with an offset to the beginning of the block index. This offset is then followed by all genomic interactions sorted by their genomic coordinates, without storing the interaction row (see below). Finally, the index consists of a list of pairs of numbers, where the first number represents the row coordinates and the second number represents the offset in the interaction block with the first interaction overlapping the current row. Empty rows are not listed in the index. Using this index, applications can quickly determine whether a block contains interactions overlapping a query by parsing the index and by occasionally checking the first interaction in a row stored in the interaction block. If blocks are compressed using zstd or any other compression algorithm supporting compression dictionaries, index and interactions could be compressed separately, such that most negative hits can be identified without decompressing interactions.

Finally, as already mentioned in Section 3.5, the .hic specification does not specify the endianness that should be used when writing serialized data. I propose that the specification should be updated to state that serialized data should be written using little-endianness, as this is the native bit-ordering employed by most modern desktop and server CPU architectures.

Cooler

Before hictk was released to the public, one of the advantages of cooler was the ability to efficiently traverse genome-wide interactions sorted by their genomic coordinates. While developing hictk, I implemented the functionality required to traverse two or more chromosome-chromosome matrices as if they were one large matrix, returning pixels sorted by their genomic coordinates. Careful implementation of this functionality enabled hictk to achieve a low overhead in terms of CPU time and memory usage. This means that any application using hictk or hictkpy to read .hic and cooler files gets this functionality for free. Given the above, there are fewer reasons left for cooler to continue storing interactions using a single matrix. Splitting interactions across multiple chromosome-chromosome matrices would have the following benefits:

- Improved performance of trans queries, as by using cooler's current indexing strategy with chromosome-chromosome matrices, it would be trivial to know the first row in the chromosome-chromosome matrix overlapping queries. With the current implementation, it is only possible to know the first row in the genome-wide matrix overlapping a query, and extra work (such as a binary search) must be carried out to figure out the index of the first pixel overlapping the queried chromosome-chromosome matrix.

- Splitting the current genome-wide index into multiple chromosome-chromosome indexes would open the possibility of opportunistically loading only portions of the index that are required to satisfy a given query, thus reducing peak memory usage (especially at high resolutions). hictk currently implements a strategy to opportunistically load parts of the cooler index. However, the current implementation of this strategy is more complex than it needs to be if pairs of chromosomes were indexed individually.

Another improvement to the cooler format could be to make the `bins` group optional for files using a constant bin size (i.e., the vast majority of publicly available cooler files), as in this case the bin table can be constructed at runtime based on the list of chromosome sizes and the bin size. hictk already implements this strategy, meaning that the bin table is not read from the cooler files, but instead is constructed from the list of chromosomes with their sizes, the prefix sum of chromosome sizes (i.e., the first bin in the genome-wide table corresponding to a given chromosome) and the bin size. All operations required to map bins to genomic coordinates and vice-versa can then be efficiently achieved through simple arithmetic operations, meaning that the memory usage of the bin table only depends on the number of chromosomes, not on the resolution.

Finally, while using HDF5 greatly simplifies the implementation of the file format, as all low-level IO operations are taken care by the HDF5 library, HDF5 itself has some design flaws that make it difficult to achieve peak performance in multithreaded applications. The HDF5 runtime keeps several global data structures that are shared across files opened by the same process, this includes one file opened in two different threads, two different files opened in one or multiple threads, and a single file opened in one thread and shared across multiple threads. This makes HDF5 inherently thread-unsafe. The C API can be made thread-safe by specifying the appropriate flags when building the HDF5 library. However, this is suboptimal, as thread safety is achieved by using a global lock that is acquired by the library before any IO operations modifying the global state (this includes most read and write operations). In an era when even entry-level CPUs have multiple cores, the single-threaded nature of HDF5 poses a significant limit to IO throughput. Luckily there are alternative formats, such as Apache Parquet [261], that offer a similar set of features as HDF5 while supporting concurrent reads and writes, and that could be used instead of HDF5.

Appendix C

Components of a reproducible analysis pipeline

Analysis pipelines often consist of a number of tools that, through a series of steps, transform data provided as input. In this context, code can be broadly classified into two families: application code and glue code. The first type of code is responsible for performing a specific analysis step, while the second type is code interconnecting different components of a computational pipeline.

Reproducibility at the application level

Reproducibility starts at the lowest level, the application level. If running a given application twice on the same data using the same parameters yields different outputs, then there is no chance of achieving replicability of the data analysis. However, achieving reproducibility may still be possible.

Applications can produce non-deterministic outputs due to a variety of reasons. It is often the case that applications use Pseudo-Random Number Generator (PRNG) as part of the data processing. Subsampling the original data is for example one step that usually involves random number generation. The default behavior in most programming languages is that if a PRNG is not explicitly seeded (i.e., initialized), then the PRNG is initialized using one of the available sources of entropy, such as the time when the code is being executed. Thus, not initializing PRNGs is one way to make the output of an application, and thus of pipelines where the application is being used, not replicable. In this case, making the application replicable is simple: update the application to properly seed the PRNG. Unfortunately, there are other factors that are more difficult to control that can make applications non-deterministic. One such factor is concurrency. In concurrent applications, the order in which operations are performed is not pre-determined. Careful application design and implementation can often ensure that the final result does not change when the order in which intermediate steps are performed is changed, however, in practice, this is not always possible. There are also bugs, such as logic bugs and data races that can make applications non-deterministic.

The bottom line is that if an application does not produce the same output when given the same inputs, replicability of data analysis is not possible. However, reproducibility may still be possible. For example, if an application is simulating a stochastic process without seeding the PRNG, running the application multiple times and comparing its output may be an approach to ensure that different runs produce similar outputs, thus fulfilling the requirements for reproducibility.

Containers enable reproducible software setup and configuration

Modern applications often rely on a complex network of software dependencies. Applications often interact with their dependencies through APIs. When APIs change, applications break, as users of an API try to interact with their dependency in ways that are no longer supported.

Let us consider the following example. Application A was written in 2022 using v1.0.0 of library B. As soon as application A is released, a user installs and uses application A to produce some results that end up part of a scientific publication. In 2023, library B is updated to v2.0.0. The update includes changes to library B's API. Another user, while trying to reproduce the results of the study mentioned above, tries to install application A using their favorite package manager. If the package manager is not aware of the API changes introduced by v2.0.0 of library B, the package manager will likely install the latest version of library B, which will break application A. Unless the user trying to reproduce the results of the study has the technical capabilities to debug API incompatibilities, the user will not be able to reproduce the findings from the original study. To make things worse, even minor updates that do not involve API changes may silently change the application behavior, leading to different outputs.

Let us consider another scenario. A computational pipeline takes file F_1 as input for program A v1.0.0, which generates file F_2 as its output. File F_2 is then given to application B that generates the final output F_3. File F_2 consists of a list of terms. At the time when the pipeline was written, program A produced the list of terms sorted in alphabetical order. Accordingly, application B was designed to take advantage of this property of file F_2. Thus, application B will properly function if and only if terms in F_2 are sorted alphabetically. Let us assume that the original pipeline was used to generate the results upon which the conclusions of a scientific study are based. Some time later, application A is updated to v1.1.0. The update involves a rework of application A internals, which causes the output of application A to no longer be sorted in alphabetical order. In the best-case scenario, application B checks that its input is sorted alphabetically before proceeding. In this case, the pipeline will be aborted and the results from the original study will not be reproducible, and the detected issue with F_2 is reported to the user. In the worst-case scenario, application B does not validate its inputs and will produce incorrect outputs, making the study certainly not replicable, and likely not reproducible. A solution to this problem is to meticulously specify the version of each application used by the pipeline. However, this is often not practical, as application A may depend on library C, which in turn depends on library D and so on.

A better approach is to take advantage of software containers, such as Docker and Apptainer containers. Software containers are standard units of software that package applications with their dependencies and configuration files to allow running applications across different systems as installed and configured by e.g., the application developer. In this scenario, the developers of the above pipeline test their pipeline with v1.0.0 of application A and make a container including both application A v1.0.0 and application B. The pipeline developers then adapt the pipeline to use containers, thus providing their users with a combination of software that has been tested and is known to function properly. The results of the pipeline are now not only reproducible but also replicable, regardless of the system where the pipeline is executed.

Workflow systems enable writing containerized analysis pipelines at scale

Writing pipelines can be tedious and difficult work, especially when a pipeline is supposed to run on different systems. Furthermore, writing complex pipelines in common scripting languages, such as bash and Python, can be prohibitively complex, and often leads to pipelines that are not very robust.

For this reason, a number of domain-specific languages known as scientific workflow systems have been developed. Examples of popular bioinformatics workflow management systems are Nextflow and Snakemake [262, 263]. These languages and frameworks greatly simplify composing and executing many applications in a way that is independent of available computational resources, meaning that workflows can be developed locally, on a development machine, and later deployed to a compute cluster or to the cloud. These systems often represent tasks as a graph, where nodes represent tasks to be executed and edges represent data and execution dependencies between different tasks.

Most workflow systems are tightly integrated with containers, thus enabling reproducible, and often replicable data analysis.

Furthermore, workflow systems like Nextflow and Snakemake provide integration with common cluster schedulers, such as SLURM, LFS, and PBS, as well as integration with common cloud providers, thus enabling writing parallel and distributed pipelines.

Another common feature of scientific workflow systems is that of checkpointing, meaning that before executing each task, the task inputs are hashed, and the results produced by the task are associated with the hashed inputs. Thus, if a workflow is launched a second time, the task inputs are hashed one more time, and if the resulting hash matches any of the known hashes, then the task can be safely skipped, as the task's outputs have already been computed. This can greatly simplify developing and debugging pipelines, as in case of failures, the pipeline will resume from the last known successful task.

In recent years, a number of communities have developed around Nextflow, Snakemake, and other workflow systems. nf-core is a community-driven project developing high-quality Nextflow workflows for bioinformatics analysis. As of Nov 2023, the project counts 55 pipelines released and 26 under development. Another community-driven collection of scientific workflows is the Snakemake workflow catalog, which counts 195 standard-compliant pipelines. These communities enable any bioinformatician to conduct state-of-the-art data analysis without having to reinvent the wheel to work around known application bugs or convert between file formats to appeal to some applications. Furthermore, the fact that these workflows are open-source, allows community members to chime in and suggest improvements to the pipelines so that pipelines can remain up-to-date with the state of the art, something that individual bioinformaticians would struggle to accomplish.