Data Production and Processing Standard of the Hi-C Mapping Center

I. Abstract

DNA loops – places where the genome bends back on itself, and loci that are distant along the contour of the genome come into close spatial proximity – play a crucial role in genetic regulation: activating genes, establishing regulatory domains, and mediating the activity of transcription factors (TFs) that specify lineages and facilitate cellular responses [1-5]. From a physical standpoint, it is possible for a loop to form between any two DNA sites on the same chromosome that are more than a few kilobases apart (i.e., far enough for the chromatin to bend). In practice, loop formation is guided by specific proteins, such as CTCF [5, 6].

DNA-DNA proximity ligation [7, 8] is a powerful tool for probing the genome's 3D structure, including the formation of loops. In recent years, the decreasing cost of sequencing has enabled researchers to probe the 3D structure of DNA and chromatin genome-wide by coupling DNA-DNA proximity ligation to high-throughput sequencing, using methods such as Hi-C [9, 10]. The resulting contact maps reflect patterns of spatial proximity between genomic loci. These contact maps enable a wide variety of analyses, ranging from the scaffolding of genome assemblies to the reliable, genome-wide identification of loops [10-13].

This document comprehensively describes the computational standards of the Hi-C ENCODE mapping center, *Comprehensive mapping of long-range chromatin interactions in human and mouse genomes* (UM1 HG009409), performing experiments that interrogate the genome's 3D structure. It includes quality control measures and computational methods.

The computational methods described in this document have been extensively validated via detailed experiments. For instance, we examined 4 loops via 3D-FISH experiments. In each case, a loop anchor showed a several-fold increase in co-localization with its partner relative to an equidistant control on the other side [10]. Next, we used CRISPR-mediated genome editing to engineer individual loops, modifying CTCF binding motifs in highly targeted fashion, as short as 1bp. In 13 of 13 cases, these alterations caused

the expected changes to chromatin looping – adding, deleting, and moving loops – as measured by in situ Hi-C [14]. Finally, we mutated three of the zinc-finger domains in CTCF proteins in a manner that disrupts CTCF's ability to bind to a large fraction of its binding sites in wild-type cells. Hundreds of these binding sites were located at loop anchors throughout the genome. Strikingly, disruption of CTCF binding sites at loop anchors consistently resulted in disruption of the corresponding loop [14]. These experiments establish the accuracy of the results produced when the computational standards described here are applied.

II. Hi-C Standards

II.a. Quality Control

II.a.1. Hi-C Library Statistics

A Hi-C experiment can fail in a number of ways. Failures are sometimes obvious when viewing Hi-C heatmaps, but their underlying cause can be difficult to diagnose. Here, we describe quality metrics that can detect a variety of failure mechanisms. Note that, prior to performing a high-resolution Hi-C experiment, we often sequence 200K – 2M reads from a "test aliquot" before deciding whether the library quality is sufficiently high to justify deep sequencing. If the library quality is sufficient then the library, or a collection of libraries for a given experiment, is sequenced to a greater depth. In a "loop resolution" Hi-C heatmap, we expect at least 2 billion total reads to be sequenced for any given experiment. As an example, Table 1 shows library statistics for *in situ* GM12878 libraries from [10]. We discuss these statistics in detail below.

	Primary (HIC*_br1)	Bio Rep 1 (HIC*_br2)	Bio Rep 2 (HIC*_br3)	Bio Rep 3 (HIC*_br4)	Bio Rep4 (HIC*_br5)
Sequenced Reads	3.6B	314M	389M	178M	669M
Normal Paired	2.7B (75%)	244M (78%)	305M (78%)	124M (70%)	477M (71%)
Chimeric Paired	563M (16%)	48M (15%)	59M (15%)	41M (23%)	124M (18%)
Chimeric Ambiguous	153M (4%)	11M (4%)	12M (3%)	7M (4%)	26M (4%)
Unmapped	187M (5%)	11M (4%)	13M (3%)	6M (3%)	43M (6%)
Alignable Reads	3.2B (90%)	291M (93%)	364M (93%)	165M (93%)	600M (90%)
Duplicates	300M (8%)	42M (13%)	15M (4%)	3M (2%)	18M (3%)
Unique Reads	2.9B (81% / 100%)	250M (80% / 100%)	348M (89% / 100%)	163M (91% / 100%)	582M (87% / 100%)
Intra-fragment	43M (1% / 1%)	2M (1% / 1%)	20M (5% / 6%)	6M (3% / 4%)	26M (4% / 5%)
Low Mapping Quality	268M (7% / 9%)	22M (7% / 9%)	31M (8% / 9%)	15M (8% / 9%)	55M (8% / 9%)
HiC Contacts	2.6B (73% / 89%)	226M (72% / 91%)	297M (76% / 85%)	142M (79% / 87%)	501M (75% / 86%)
Inter chromosomal	644M (18% / 22%)	51M (16% / 20%)	70M (18% / 20%)	31M (18% / 19%)	105M (16% / 18%)
Intra chromosomal	2B (55% / 68%)	176M (56% / 70%)	227M (58% / 65%)	110M (62% / 68%)	395M (59% / 68%)
Intra Short Range (<20 Kb)	602M (17% / 20%)	47M (15% / 19%)	82M (21% / 23%)	38M (21% / 24%)	147M (22% / 25%)

Intra Long Range (≥20 Kb)	1.4B (39% / 47%)	129M (41% / 51%)	145M (37% / 42%)	72M (40% / 44%)	248M (37% / 43%)
Ligations	950M (27%/32%)	80M (26%/32%)	97M (25%/28%)	76M (42%/47%)	222M (33%/38%)
3' bias long range	68% - 32%	69% - 31%	69% - 31%	75% - 25%	72% - 28%
Read Pair type (L-I-O-R)	25-25-25-25	25-25-25-25	25-25-25-25	25-25-25-25	25-25-25-25

Table 1. Statistics from typical Hi-C experiments [10].

II.a.1.i. Standard sequencing and alignment statistics: We record a series of statistics that characterize the sequencing and alignment. For alignment, if more than 25% of reads fail to align, it typically indicates either a problem with the sequencing run or sample contamination. In cases where more than 25% of the reads fail to align, the library is considered to have failed. Chimeric frequency is an indicator of the frequency of long-range ligation junctions in the data, although the specific value seen depends on numerous experimental parameters, such as aligner and read length. Typical values for chimeric paired and chimeric ambiguous fall between 10-30% and 0-10%, respectively.

II.a.1.ii. Duplicate frequency: High duplication rate indicates low molecular complexity [15]. A Hi-C library is considered a good candidate for deeper sequencing if complexity estimates suggest it consists of hundreds of millions or billions of unique contacts. In general, in constructing "loop resolution" Hi-C heatmaps, it is more efficient to sequence many replicate libraries at lower depth (e.g. one lane on an Illumina HiSeq, or less than 20% of the total library complexity) rather than sequencing many lanes of a single library, since the latter strategy typically leads to extensive duplication. In addition, the duplicate removal step requires vastly more time and memory if a single library is sequenced very deeply. For instance, if we estimate that an otherwise high-quality library contains 1.5 billion contacts, we usually plan to sequence at most two lanes from that library on an Illumina HiSeq (~300M reads total).

The exact molecular complexity was calculated using the Picard tools formulation of the Lander-Waterman equation [15], which entails solving the following equation for the molecular complexity m (measured in molecules):

$$R/m = 1 - e^{N/m}$$

Here R is the number of distinct reads observed, N is the total number of reads sequenced. Note that "optical duplicates" created by the Illumina sequencing process rather than PCR were not included in either R or N. In our experience, the resulting value is typically an underestimate of the true library complexity.

It is suggested to have less than 40% duplication rate in total sequenced read pairs for deep Hi-C data sets. However, going over the suggested 40% duplication rate does not constitute library failure. A Hi-C library is considered a good candidate for deeper sequencing if complexity estimates suggest it consists of hundreds of millions of unique contacts.

II.a.1.iii. Fraction of "Hi-C contacts": After duplication removal, we filter out read pairs where both ends align to the same fragment. If this step filters out over 20% of read pairs, it indicates that the library failed in the restriction, fill-in or ligation steps of the protocol and thus is not a good candidate for deeper sequencing. We also filter out read pairs where the mapping quality of either read falls short of the desired threshold. (In the example of Table 2, the threshold is MAPQ > 0.) The rest of the quality metrics are calculated using the list of read pairs that remain once all filtering was completed. We refer to these read pairs as "contacts."

II.a.1.iv. Ligations: This statistic measures how often a ligation junction is found inside a read. (A ligation junction is the sequence created when the ends of two filled-in restriction fragments ligate to one another. For MboI, the ligation junction sequence is GATCGATC. For HindIII, the sequence is AAGCTAGCTT.) A paucity of ligation junctions in a Hi-C library suggests that the ligation failed. If a library has less than a 5% occurrence of ligation junction, the library is considered a failure. Note that there can be many causes of such a failure, ranging from a bad batch of DNA ligase to rupture of the cell nuclei. This statistic is also dependent on sequence read length and insert size. We typically sequence a 300-500bp insert using 101bp PE reads, and observe that ligation rates tended to fall into the 30-40% range. Of course, with shorter reads and longer insert sizes, this value tends to be smaller. With longer reads and shorter inserts, it is larger.

II.a.1.v. Proximity to 5' and 3' restriction fragment ends: For long-range contacts (defined as intrachromosomal and over 20Kb apart, or interchromosomal), we look at both read ends to see if the end is closer to the 5' or 3' end of the restriction fragment and to which strand the read maps. When Hi-C libraries are generated using a six-cutter restriction enzyme and, after the shearing step, are size selected for 300-500bp molecules, we find that the large fragment size to insert size ratio causes most contacts (>85%) to come from the 3' ends of fragments. A much lower value indicates that the restriction enzyme had not cut effectively. Note that when the fragment size to insert size ratio declines (i.e., when a four-cutter restriction enzyme is used) we find that the 5' to 3' bias is markedly attenuated.

II.a.1.vi. Percentage of contacts at various distances: We break down contacts into intra-chromosomal and inter-chromosomal contacts. We then further subdivide the intra-chromosomal contacts to short-range (<20 Kb) and long-range (≥ 20 Kb) contacts. A crucial metric is the percentage of long-range intra-chromosomal contacts. In successful Hi-C libraries, we find that at least 20% of unique reads are long-range intra-chromosomal contacts, referred to as the "Intra Long Range (≥20Kb)" statistic. Lower values usually indicate that the experiment has failed. In general, this value is one of the statistics we find most important to scrutinize in performing cost-effective high-depth Hi-C. A library with many inter-chromosomal contacts and a paucity of contacts at shorter distances (i.e., absence of both a strong diagonal and robust distance decay effects in the intra-chromosomal contact matrices) suggests that the library either comprises mostly random ligation products, due to the rupture of a large fraction of nuclei, or reflects an open chromatin structure.

II.a.1.vii. Percentage of contacts by read pair type: We break down intrachromosomal contacts by type: in a "left" pair, both ends map to the reverse strand. In a "right" pair, both reads map to the forward strand. In an "inner" pair, the ends map to different strands and point (5' to 3') towards each other. In an "outer" pair, reads land on opposite strands but point away from one another. If the chimeras observed are due to proximity ligation, this statistic should be random, i.e., each pair type should account for roughly 25% of contacts. Thus, the distance at which the percentage of each pair type converges to 25% is a good indication of the minimum distance at which it is meaningful to examine Hi-C contact patterns. For six-cutter restriction enzymes, such as HindIII and NcoI, this distance is approximately 30 Kb. For four-cutter restriction enzymes (MboI, DpnII), this distance is approximately 3 Kb (Figure 1). Note that the existence of read pairs in the "right" and "left" configuration is rarely seen outside of Hi-C experiments. DNA-Seq reads, for instance, are by design all "inner" pairs; "jumping libraries" tend to produce outer pairs.

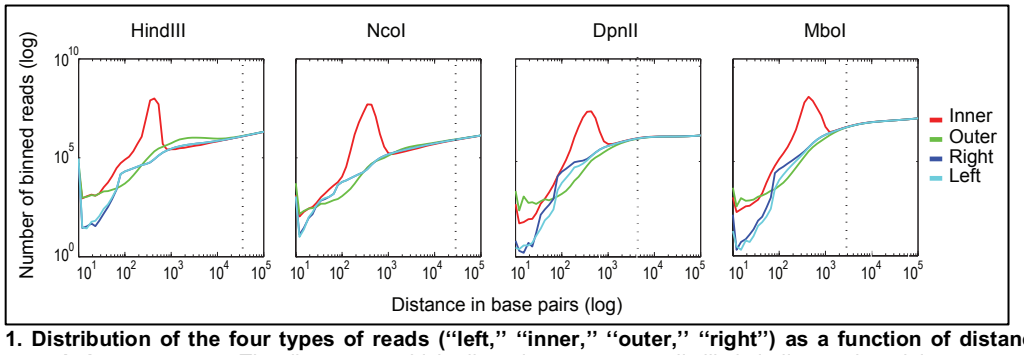


Figure 1. Distribution of the four types of reads ("left," "inner," "outer," "right") as a function of distance for various restriction enzymes. The distance at which all read types are equally likely indicates the minimum meaningful contact distance. For HindIII and NcoI (6 bp restriction enzymes), this distance is ~30 kb; for DpnII and MboI (4bp restriction enzymes), this distance is ~3 kb. (Vertical lines drawn at where the four distributions converge to 25% +/- 1%.)

II.a.1.viii.Summary of Hi-C Computational Standards:

	Fail	Marginal	Pass	Typical
Intra Chromosomal	---	---	---	>40.0%
Inter Chromosomal	---	---	---	<40.0%
Chimeric Ambiguous	---	---	---	<10.0%
Chimeric Paired	---	---	---	10.0-30.0%
Alignable Reads	<75.0%	75.0-90.0%	>90.0%	---
Duplicates	---	>40.0%	<40.0%	---
Intra-fragment	>20.0%	10.0-20.0%	<10.0%	---
Hi-C Contacts	<20.0%	20.0-50.0%	>50.0%	---
Intra Short Range (<20 Kb)	>60.0%	30.0-60.0%	<30.0%	---
Intra Long Range (≥20 Kb)	<20.0%	20.0-35.0%	>35.0%	---
Ligations	<5.00%	5.00-25.0%	>25.0%	---

Table 2. Hi-C Library Statistic Standard Thresholds for Passing, Failing and Marginal. The table shows the intervals of fail, marginal and pass values for each Hi-C statistic from the text above. Percentages refer to unique read pairs.

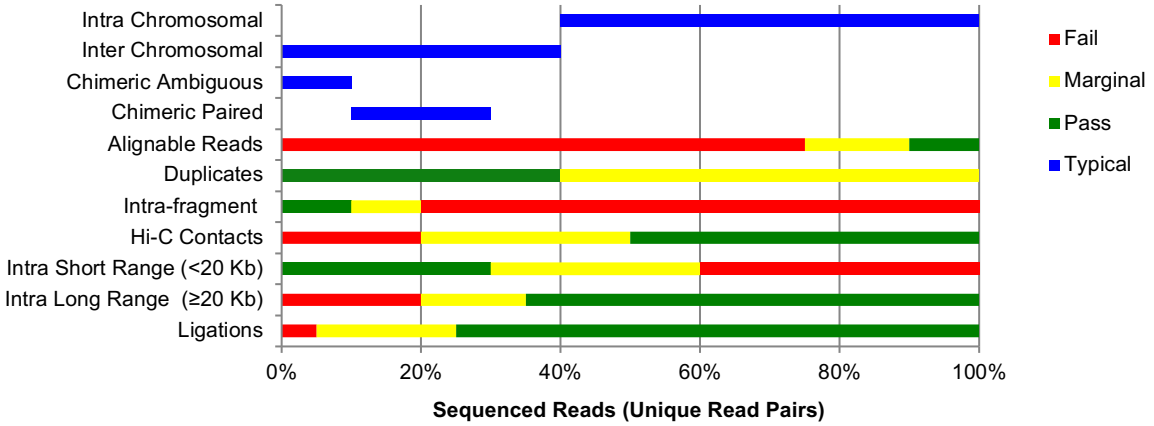


Figure 2. Hi-C Library Statistic Standard Thresholds for Passing, Failing Typical, and Marginal. The bar graph shows the intervals of fail, marginal and pass values for each Hi-C statistic from the text above.

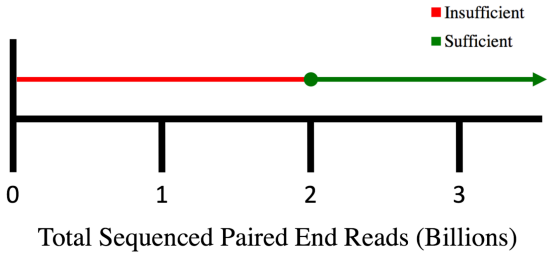


Figure 3. Loop Resolution Hi-C Heatmap Read Threshold. In order for a Hi-C map to be considered loop resolution, the number of total sequenced reads must be at least 2 billion paired end reads for any given experiment. Note that an experiment can be comprised of multiple libraries.

II.a.2. Aggregate Peak Analysis (APA) for Peak Calling Quality Control

Aggregate Peak Analysis (APA) is a method that allows us to test the aggregate enrichment of an entire set of putative two-dimensional peaks, as opposed to verifying individual peaks one-by-one. This method is especially useful for checking a set of peak calls on a low-resolution Hi-C map, where individual peaks may be impossible to discern but where the aggregate signal from the full peak set should be detectable if the peak set is reliable and the Hi-C map was the result of a successful experiment in the same cell type.

APA quantifies the enrichment of a peak set in aggregate by plotting the sum of a series of submatrices derived from a contact matrix. These submatrices are chosen so that each one surrounds a single putative peak pixel (note that for intrachromosomal pixels, we always choose the pixel in the upper-right half of the

matrix). In the resulting APA plot, the total number of contacts that lie within the peak pixel set is shown at the center; the entry immediately to the right of center corresponds to the total number of contacts in the pixel set obtained by shifting the peak set 10 kb to the right; the entry two positions above center corresponds to an upward shift of 20 kb; and so on. Focal enrichment across the peak set in aggregate manifests as larger values at the center of the APA plot.

To perform APA, a resolution and window size is chosen. The APA resolution determines the resolution at which the contact matrix of a Hi-C map is generated. Note that, in APA analyses, contact matrices will often be generated for a given Hi-C map at resolutions vastly higher than the resolution at which the map would usually be examined. For instance, we generate contact matrices for our 2009 Hi-C maps [9] at 10 kb resolution, despite the fact that the map resolution of these maps is orders of magnitude larger. (The aggregation process makes it possible to resolve features at much higher resolutions than would ordinarily be possible with a map, producing a “super-resolution” image.)

We center a submatrix at each peak in the target peak set at the chosen resolution. The width of the matrix is the window size above. For peak calls that span an area larger or smaller than one pixel in the chosen resolution, we choose the center of the peak call as the center of the matrix. Only one submatrix is created per peak call, even if the peak call extends to multiple pixels. If the center pixel of multiple peak calls falls into the same pixel, we use that submatrix only once. To avoid strong distance effects, we only examine peak calls where the peak loci are separated by more than a minimum threshold *t*.

For APA performed at 10 kb resolution with a window of +/- 100 kb, *t* = 300 kb. For APA performed at 5 kb resolution with a window of +/-25 kb, *t* = 100 kb.

Submatrices for the peaks are taken from the normalized (intrachromosomal KR corrected) Hi-C maps. These submatrices are then summed (entry-wise), obtaining an APA matrix in which the center pixel represents the sum of the number of reads in the entire target loop set.

To determine if the center pixel of the APA plot is focally enriched, we calculate the APA score, which is the ratio of the number of reads in the center bin to the average number of reads in the lower-left corner of the APA matrix. We define the lower left bins for 10 kb resolution and a +/- 100 kb window size as the bins lying in the bottom-left 6 x 6 square of the matrix. For 5 kb resolution and a +/- 25 kb window we define the lower left bins as those lying in the lower left 3 x 3 square of the matrix. We chose to use this particular score because it is very simple to understand and calculate, and because it corresponds to the widely-accepted notion of a loop: in order for the APA score to be above 1, the number of contacts between a typical pair of loop anchors in the peak set must be higher than the number of contacts between intervening pairs of loci. See Figure 4.

To calculate a p-value for this score, we calculate the z-score which compares the central bin to the set of bins in the lower left window defined above. The z-score is then converted into a p-value (1-sided).

The color scale in all APA plots is set as follows. The minimum of the color range is 0. The maximum is 5 x UR, where UR is the mean value of the bins in the upper-right corner of the matrix. The upper-right corner of the 10 kb resolution APA plots is a 6 x 6 window (or 3 x 3 for 5 kb resolution APA plots).

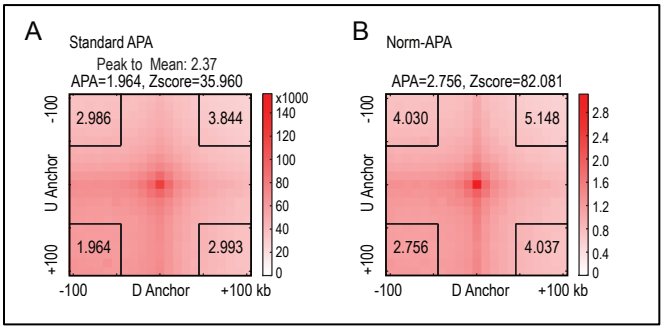


Figure 4. APA Examples. **(A)** Standard APA of the in situ GM12878 peak list examined on the GM12878 dilution map from [10]. The APA score is the ratio of the number of contacts in the central bin to the mean number of contacts in the lower-left corner (outlined by the black box). The ratio of the central bin to the other three corners is also shown inside each corner. APA scores above 1 signify that the peak bin is enriched relative to the bins inside the corner. The ratio of the central bin to the mean of the remaining matrix, along with the zscore of the central bin, using the mean and standard deviation of the lower-left corner, are also shown. Z-scores above 1.64 indicate enrichment ($p<.05$). **(B)** Normalized APA of our in situ GM12878 peak list examined on our GM12878 dilution map. In Normalized APA, each submatrix is first normalized before being added to the aggregate matrix; normalization is performed by dividing each entry by the mean of the submatrix. The final Normalized APA matrix is the

average of all of these submatrices. The maximum color scale in all APA plots is set to five times the mean value in the upper right corner of the matrix.

II.a.3. Experimental validation

Validation of peak calling is especially important in light of previous results in the field. It is essential to confirm that putative peaks are the result of real local enrichment and to verify the loops when possible by independent means.

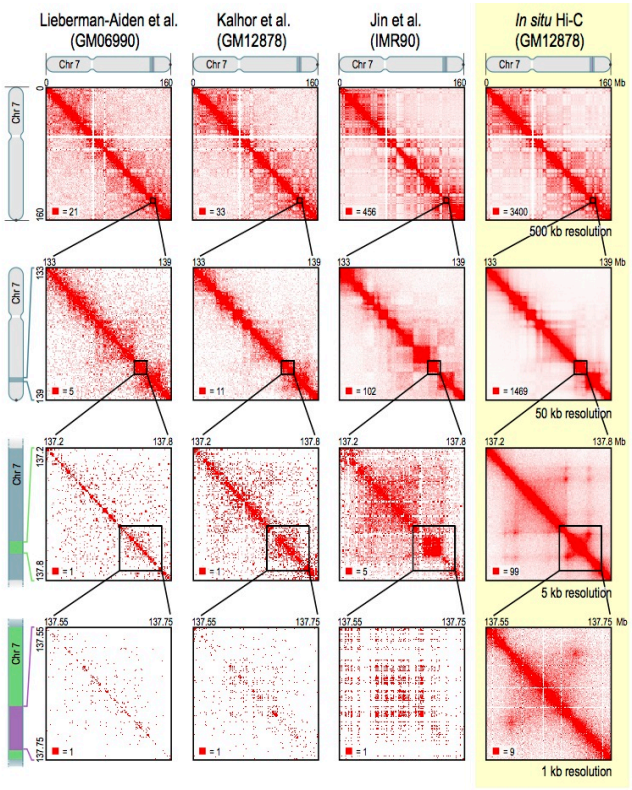


Figure 5. In situ Hi-C map of chr7 in GM12878 (last column) compared to earlier maps. Loops manifest as bright peaks in in situ Hi-C data, and are readily visible at 5kb and 1kb resolution.

Figure 5 (from [10]) shows the depth of our in-situ Hi-C maps when compared to previous protocols. We also independently verified our peak calls with 3D-FISH. As a step in quality control, we suggest processing the files and examining putative peak calls in the data, as visualization can often make clear whether or not a putative peak call is correct.

We have also validated peaks by experimental means. We used CRISPR-mediated genome editing to engineer individual loops, modifying CTCF binding motifs in highly targeted fashion, as short as 1bp. In 13 of 13 cases, these alterations caused the expected changes to chromatin looping – adding, deleting, and moving loops – as measured by in situ Hi-C. We also mutated three of the zinc-finger domains in CTCF proteins such that CTCF's ability to bind in wild-type cells was disrupted. Hundreds of these binding sites were located at loop anchors throughout the genome. Disruption of CTCF binding sites at loop anchors consistently resulted in disruption of the corresponding loop. In all cases, we also found that the ability to modify loops also led to predictable changes in domain formation [14].

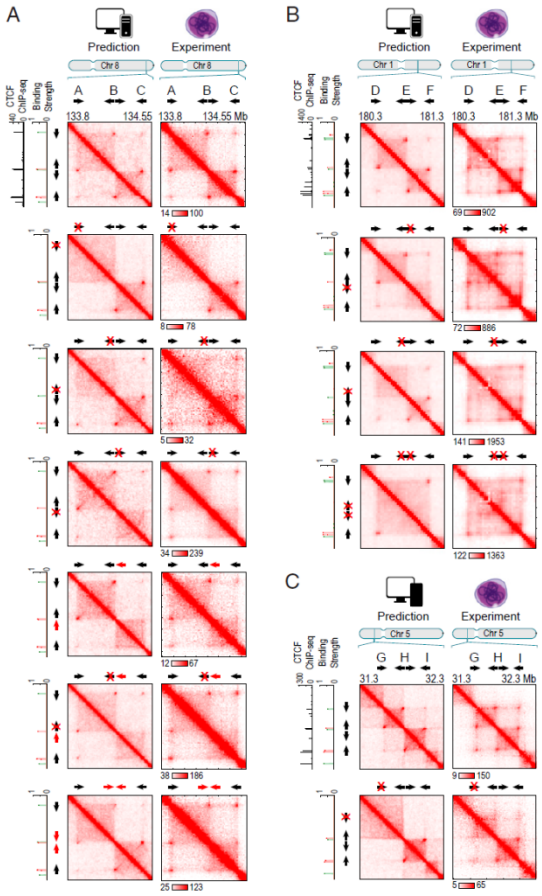


Figure 6. Genome editing of CTCF motifs allows re-engineering of loops in accordance with the convergent rule; the resulting contact maps can be predicted *in silico* using extrusion simulations. (**A**) Results of CRISPR/Cas9-based genome editing experiments at the same locus. Extrusion simulations are shown on the left, and experimental data is shown at right. (**A**, 1st row) The contact map for the wild-type locus, calculated using *in silico* simulations (left), closely matches the map observed using Hi-C[2] experiments (right). (**A**, 2nd row) Deletion of A/Forward eliminates the A-B and A-C loops and the contact domain boundary at locus A. The predictions of our *in silico* simulations (left) closely match the contact map observed using Hi-C[2] experiments (right). All parameters in this and subsequent simulations of mutant regions use exactly the same parameters as the simulations of the corresponding wild-type contact map. The only difference in the mutant simulation is the modification of the appropriate CTCF binding site (in this case, deletion of A/Forward). (**A**, 3rd row) Deletion of B/Reverse eliminates the A-B loop. (**A**, 4th row) Deletion of B/Forward eliminates the B-C loop. (**A**, 5th row) Inversion of B/Forward eliminates the B-C loop. (**A**, 6th row) Simultaneous deletion of B/Reverse and inversion of B/Forward eliminates the B-C loop. (**A**, 7th row) Inversion of both B/Forward and B/Reverse does not eliminate loops. (**B**) A similar series of results for chr1: 180.3 – 181.3Mb. Notably, the elimination of one loop anchor motif at the middle locus fails to eliminate either the D-E or E-F contact domain. When both loop anchor motifs are eliminated, both the D-E and E-F contact domains disappear. (**C**) We disrupted a forward CTCF motif by inserting a single base at chr5:31,581,788. Two loops are disrupted.

II.b. Hi-C Meta Data Recording Standard

For cell lines the following information should be recorded and provided:

- Cell line source and lot number.
- Unique Biological Replicate Index #, with pointers to other experimental data and other available aliquots from the replicate
- Protocol used to culture cell lines.
- Growth time/passage number.
    - culture start date
    - culture duration & culture harvest date
    - passage number
- Cell Population at time of Harvest.
    - Density
        - For suspension cells, per unit volume
        - For adherent cell, per unit area
    - Viability, >90%
    - A microscopic image of the cells showing cell morphology.
- Results of tissue culture contaminant (e.g. mycoplasma/wolbacia) tests, if conducted.
- Results of karyotyping and marker analysis, if conducted.

For tissues, organs or whole organisms, the following should be recorded and provided:

- Protocols for purification or isolation of tissue or cell types.
- Amounts of starting material (tissue/organ weights, cell number, etc).

For all experiments, the following should be recorded and provided:

- Cross-link date and conditions

- Starting Cell Number
- Restriction Enzyme, if applicable
- Biotinylated base used, if applicable
- Ligation volume
- Amplification cycles
- Final concentration (Qubit) [ng/ul]
- Kapa qPCR molarity (nM)
- Average Size (Bioanalyzer) [bp]
- Sequencer Used
- Read Length (R1/R2)
- Cluster Density (K/mm2)

III. Hi-C Data Processing

Hi-C data processing encompasses many different processes, from the analysis and visualization of Hi-C experiments to de novo assembly.

III.a Juicer pipeline

The Juicer Hi-C data processing pipeline transforms raw sequence data into a highly compressed, random access binary file, called a ".hic" file, that can be used across many modalities. Here, we outline the steps of the pipeline and describe the files it produces. Below, we also describe two companion software packages that utilize the .hic file: a fully-featured visualization tool for exploring Hi-C data together with other seq data called Juicebox, and the Straw data API written in multiple languages for fast extraction of matrices.

All software packages are open source and available on GitHub at https://github.com/theaidenlab. Juicer and Juicebox were recently published in Cell Systems ([16];[17]). This text is heavily borrowed from the supplemental materials of ([10]), where the pipeline and feature annotation algorithms were first described. The software is in widespread use in the 3D genomics community, as measured both in downloads and in the level of engagement on the 3D genomics forum: https://groups.google.com/forum/#!forum/3d-genomics

Broadly speaking, the Juicer pipeline consists of two main parts: the transformation of raw reads into a highly compressed binary file that provides fast random access to contact matrices stored at multiple resolutions with multiple normalization schemes; and the analysis portion, which operates on the binary file and automatically annotates contact domains, loops, anchors, and compartments.

We start by detailing the first part of the pipeline, where raw fastq files are aligned, ambiguous chimeric and duplicate reads are filtered, and contact matrices are created and normalized at many different resolutions. In the next section we describe the feature annotation algorithms, including Arrowhead for contact domains and HiCCUPS and MotifFinder for loops.

Juicer transforms raw fastq files into a highly compressed binary *.hic* file containing the contact matrices and normalization vectors at many different resolutions. The time estimates used below are taken from the Univa cluster system with default parameters; please see section III.a.4. Cluster Systems for a discussion of clusters. Below is table of both the expected input files for Juicer and the expected output of Juicer. For more information on file formats see section III.c.3. Data Formats or visit https://github.com/theaidenlab/juicer/wiki/Data.

Input

File format	File description
read_R1.fastq read_R2.fastq	These are the raw data files that come off the sequencer. They include the read name, the read (a string of A,C,T,G, or N) and base quality information. (Ex. HIC001 _R1_001.fastq, HIC001_R1_002.fastq)
refrence.fasta	This is a file that contains the nucleic acid sequence of the organism in your experiment. (Ex. hg19.fasta)
reference_restriction_enzyme.txt	This text file contains all locations of the nucleic acid motif recognized by the restriction enzyme in the reference genome. (Ex. hg19_MboI.txt)
refrence.chrom.sizes	A text file containing the name and length of each chromosome.

Table 3. Expected input files for running the Juicer pipeline.

Output

File format	File description
inter.hic/inter_30.hic	The .hic files for Hi-C contacts at MAPQ > 0 and at MAPQ >= 30, respectively.
merged_nodups.txt	The Hi-C contacts with duplicates removed. This file is also input to the assembly and diploid pipelines.
stats_dups.txt / stats_dups_hists.m	Statistics and graphs on the duplicates.
merged_sort.txt	This is a combination of merged_nodups / dups / opt_dups and can be deleted once the pipeline has successfully completed.
collisions.txt	Reads that map to more than two places in the genome.
dups.txt, opt_dups.txt	PCR duplicates and optical duplicates.
inter.txt/inter_30.txt inter_hists.m/inter_30_hists.m	The statistics and graphs files for Hi-C contacts at MAPQ > 0 and at MAPQ >= 30, respectively. These are also stored within the respective .hic files in the header. The .m files can be loaded into Matlab. The statistics and graphs are displayed under Dataset Metrics when loaded into Juicebox.

Table 4. Expected output from the Juicer pipeline.

III.a.1. Fastqs to contact matrices

V.a.1.i. Sequence alignment: The sequencer produces two fastq files, one for each read end. Each file is sorted by "read name." One lane of HiSeq data comprises approximately 150 million raw reads. Uncompressed, the data occupies roughly 80 GB of disk space. The pipeline begins by splitting each of the two fastq files into chunks containing 1.5 million single end reads, with roughly 200 chunks for one lane of data (see

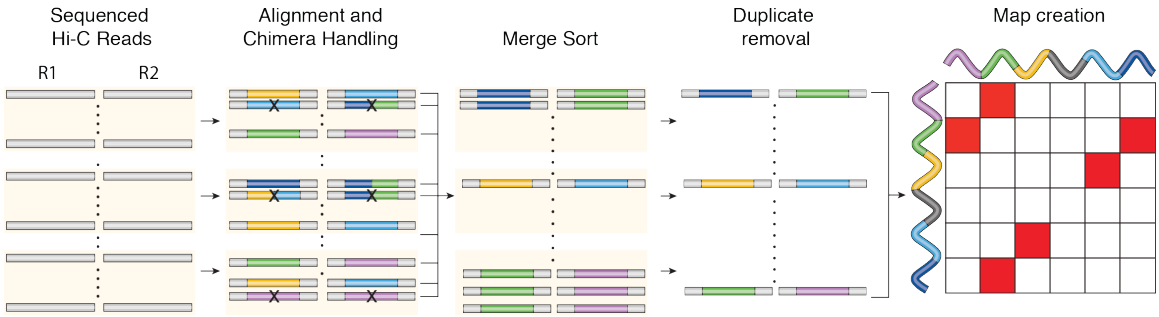


Figure 6. Juicer pipeline. Sequenced read pairs (horizontal bars) are aligned to the genome in parallel. Color indicates genomic position. Read pairs aligning to more than two positions are excluded. Those remaining are sorted by position and merged into a single list, at which point duplicate reads are removed. The .hic file stores contact matrices at many resolutions, which can be loaded into Juicebox for visualization.

A note on aligners: the short end aligner *bwa aln* does not return chimeric reads and is therefore suboptimal for aligning long-read paired end Hi-C data where the chimera rate is likely to be high. Paired end aligners should be avoided, as they make assumptions about the insert size that are false for Hi-C data. Since we expect a ligation product, the read ends may be quite far from one another.

After alignment, each fastq file chunk has a corresponding SAM file. Since the alignment doesn't change the order of the reads, the SAM files for individual chunks are sorted by read name.

Next, the two sorted SAM files for each chunk (corresponding to both the first and second read) are merged into a single, paired-end SAM file. The latter is also sorted by read name. This procedure takes linear time with respect to the length of the chunk (in this case, 1.5 million reads). The procedure is equivalent to the final stage of the classic Merge Sort algorithm. This reduces the number of files from ~200 to ~100 "chunks."

III.a.1.ii. Filtering of abnormal alignments: About 75% of the time, each read in a read pair will align to a single site in the genome. We call such read pairs "normal."

Another 20% of read pairs are "chimeric". This means that at least one of the two reads comprises multiple subsequences, each of which align to different parts of the genome. For instance, the first 50 base pairs might map perfectly to one position, whereas the next 50 map perfectly to a second position several megabases away.

Chimeric read pairs are classified as "unambiguous" or "ambiguous." In an "unambiguous" chimeric read pair, one read maps chimerically to both locus A and locus B, and the other read maps to either locus A or

locus B, but not to both. Such reads commonly arise from ligation products between locus A and locus B, in a case where one of the two reads crosses the ligation junction. These “unambiguous” chimeric read pairs comprise roughly 15% of all read pairs and are included in our maps as ligation junctions between locus A and locus B. All other chimeric read pairs are “ambiguous” and are not included in our Hi-C maps.

Finally, about 5% of read pairs are “unalignable”: they have at least one end that cannot be successfully aligned. We do not use data from unalignable read pairs in our Hi-C maps.

Note that low-quality “normal” and “unambiguous chimeric” alignments are filtered, but not at this stage; see below.

III..a.1.iii. Filtering of duplicates: After eliminating both ambiguous chimeric reads and unalignable reads, we retain roughly 90% of the original read pairs in the form of paired-end SAM files for each “chunk”.

Next, we use binary search to append a restriction fragment number (index of the fragment demarcated by restriction sites in the genome) to each record based on the genome and the restriction enzyme used in the experiment. To do this, we search the genome for all instances of the restriction enzyme’s motif, producing a text file that lists all motif locations. Using this text file as a reference, we determine the fragment in which each read lies on the basis of the aligned position of the read, and append this fragment information to each record. At this point, each file chunk consists of one line per read, containing the following fields: read name, strand1, chromosome1, position1, fragment1, strand2, chromosome2, position2, fragment2. These fields describe the alignment of both reads in a read pair. (We also include additional fields that are relevant to downstream processing; more on this below.)

We then rearrange the record for each read pair so that the chromosome of the first read precedes the chromosome of the second read. If both reads in a read pair are on the same chromosome, we rearrange the read pair so that the fragment of the first read precedes the fragment of the second read. If reads share both chromosome and fragment, we sort by strand; and finally, by position in base pairs. We then use Unix sort to sort all records in the file chunk, with precedence for chromosome, then for fragment, then for strand, and finally for position. Unix sort on relatively small files, such as these, is quite efficient. For each file chunk, assigning the fragment and sorting takes on average 3.5 minutes.

Once they are organized in this way, we merge all 100 chunks into a single master file sorted by chromosome, then fragment, then strand, then position. That is, all chromosome 1 reads will be grouped together; within that set, all fragment 1 reads will come before fragment 2 reads; within the set of chromosome 1/fragment 1 reads, all forward strand reads will come before all reverse strand reads; and finally all position 1 reads will come before all position 2 reads. This can be accomplished in O(n) time by taking the 100 sorted chunks and employing the same merge sort methodology pointed out above. Essentially, this merge takes only the time required to write the file, which is approximately 25 minutes in practice.

Using this sorted master file, it is possible to identify and remove duplicate read pairs in linear time. We consider two read pairs to be duplicates of one another if their reads lie at closely corresponding positions (i.e., within 4bp of one another). More precisely, given two sorted read pairs (A1, A2) and (B1, B2), we compare read A1 to read B1 and read A2 to read B2. The read pairs are considered duplicates if (i) A1 and B1 align to the same chromosome and strand; (ii) the 5’ base of A1 aligns to a position within 4bp of the position of the 5’ base of B1; (iii) A2 and B2 align to the same chromosome and strand; and (iv) the 5’ base of A2 aligns to a position within 4bp of the position of the 5’ base of B2.

The duplication removal step is accomplished by a simple awk script. To further speed up the duplicate removal step, we parallelize it. We first split the sorted master file into chunks containing roughly 1 million read pairs each. The files are split at points known not to be duplicates: we start at the beginning of the file, and after 1 million reads, we look for the instance where there is a sufficiently large difference in position between the read pair in one record and the read pair in the next record. The split is performed in “real time” in the following sense: as soon as a break-point is found, the chunk is written out, and the chunk is immediately filtered for duplicates. The filtering takes 30 seconds and occurs at the same time as the sorted master file continues to read and write chunks; total time until the final chunk is written is usually 16 minutes. Because the chunks are numbered, putting them back together merely requires us to concatenate the file. This is dominated, as usual, by the time it takes to write the file, roughly 25 minutes.

The rate of duplication varies from experiment to experiment, as a function of both the Hi-C library’s molecular complexity (i.e., the number of unique ligation products contained in the library) and the number of

read pairs sequenced. Indeed, the complexity of a given Hi-C library can be estimated based on the number of read pairs sequenced, and the number of duplicates observed (see below).

At this point, the text file contains a duplicate-free list of read pairs. The record for each read pair contains two alignments. To create the final master list of read pairs, we throw out all read pairs where both reads align to the same fragment.

III.a.1.iv. Filtering of low-quality alignments: When creating our Hi-C maps, we require a minimum alignment quality for each read that is included in the map. This requirement is enforced in our very final step, in which we throw out read pairs where the alignment of one or both reads fails to meet this threshold. One of two thresholds is applied: MAPQ > 0 (which means that a unique "best" alignment exists) or MAPQ ≥ 30 (which means that the chances that an alignment is erroneous is at most 1 in 1000). The results of this procedure are two lists of Hi-C "contacts," one for MAPQ>0, and one for MAPQ ≥ 30. Thus, in all of our maps, reads mapping to repeats where a unique mapping cannot be determined are thrown out. While our longer read length allows us to map more often around smaller repeats, there are still regions of low sequence complexity where our contact maps are sparse due to high repeat density. To determine whether, and how well, the Hi-C experiment worked, we calculate a variety of library statistics using the final contact lists (see below).

III.a.1.iv. Construction of contact matrices: From the filtered contact list, we generate contact matrices using varying locus sizes. For example, to calculate the contact matrix with a 1 Mb locus size (also called "1 Mb matrix resolution"), we divide the linear genome into 1 Mb bins and count the number of contacts we observe between each pair of bins. The number of contacts observed between locus i and locus j is denoted M_{ij}. For each of the two filtered files (MAPQ > 0 and MAPQ ≥ 30), we calculate contact matrices at resolutions of 2.5 Mb, 1 Mb, 500 kb, 250 kb, 100 kb, 50 kb, 25 kb, 10 kb, and 5 kb. The set of resolutions for creating the contact matrices is a parameter that can be passed in.

We also create fragment-delimited contact matrices, using the fragment number assigned to each read in the procedure above. It is thus straightforward to bin using a fixed number of fragments. For each of the two filtered files, we calculate fragment-delimited contact matrices at resolutions of 500f, 200f, 100f, 50f, 20f, 5f, 2f, and 1f.

III.a.2. Contact Matrix Normalization

Ideally, the entries of the matrix of raw contact counts, M_{ij}, would be proportional to the true contact frequency of locus i and locus j. However, due to biases in the Hi-C experiment, this is not the case. Chromatin accessibility, nucleosome occupancy, alignability and restriction site density at a locus can all affect the contact count. We refer to such effects as "one-dimensional biases": biases that are a function of the locus itself and influence contact frequency between that locus and any other locus. If there is a strong bias towards observing contacts containing locus i, then entry M_{ij} will tend to have more reads, regardless of whether or not locus i and locus j actually interact very frequently. One-dimensional biases affect not only Hi-C, but also all ligation assays (NLA, 3C, 4C, and 5C).

III.a.2.i. Vanilla coverage normalization ([9]): There have been various approaches in the literature towards normalizing Hi-C matrices in order to eliminate bias. [9] addressed this issue via a coverage normalization step. A row-specific normalization term R_i was calculated by summing the counts in a row (the L_1 norm) and taking the reciprocal. A column-specific normalization term C_j was calculated similarly, by summing the counts in a column and taking the reciprocal. For symmetric (typically intrachromosomal) matrices, $C_i=R_i$. For every entry in the matrix M_{ij}, the normalized matrix entry M^*_{ij} is therefore $R_iM_{ij}C_j$. Here, we refer to this coverage normalization as "vanilla coverage normalization," or "VC normalization". VC normalization is very simple to implement, can be calculated quickly, and is highly robust, even in the setting of extremely sparse data.

One problem with VC normalization is that it tends to overcorrect. A simple approach toward reducing this effect is to use the square root of the VC vector. The square root can be motivated very briefly by observing that such a correction makes the entries of M^*_{ij} dimensionless by changing units of [reads] to units of [reads]/[reads$^{0.5}$][reads$^{0.5}$]. We have found that square root normalization provides values that are surprisingly close to those of much more sophisticated and computationally intensive algorithms (see Figure 7).

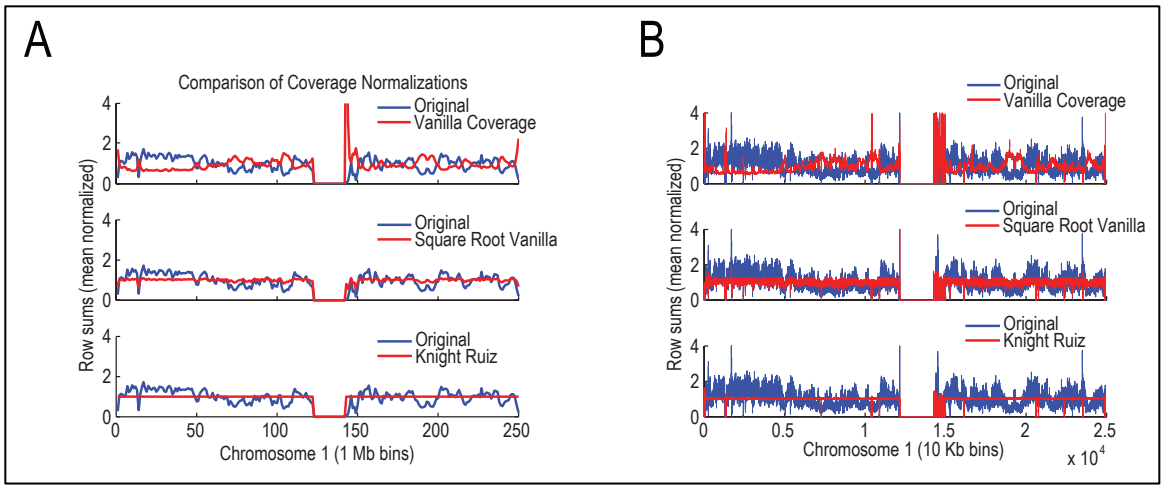


Figure 7. Row sums on chromosome 1 subsequent to applying the coverage normalization techniques computed by our pipeline (top: vanilla, middle: square root, bottom: KR), performed on the intrachromosomal contact matrix at 1 Mb resolution **(A)** and 10 kb resolution **(B).** At low resolutions vanilla overcorrects low coverage loci, while square root vanilla creates a matrix whose row and column sums are all approximately equal. KR normalization works at both low and high resolutions.

III.a.2.ii. A role for matrix balancing in Hi-C: It is also possible to normalize matrices without making any presuppositions about which factors are responsible for the observed biases, using a method that is nearly a century old known as "matrix balancing." In matrix balancing we make the assumption – similar to that of VC normalization – that the only biases present are scalar, multiplicative, one-dimensional biases; i.e. that the true matrix of contact probabilities M^*_{ij} is of the form $C_i M_{ij} C_j$, where the C_i are unknown, locus-specific bias factors. By enforcing M^*_{ij} to be doubly stochastic (each of its rows and columns must sum to 1), it is possible to solve for the bias factors C_i.

The problem of matrix balancing is extremely well-studied in data analysis, with the oldest algorithms dating back to the 1930s [18]. A famous 1967 paper by Sinkhorn and Knopp provided an algorithm that converts any square nonnegative matrix to a doubly stochastic matrix via multiplication by diagonal matrices [19]. The algorithm works by repeatedly performing VC normalization on M_{ij} until convergence is achieved, that is, until all of the rows and columns sum to the same value. Sinkhorn and Knopp proved that this algorithm would converge as long as the matrix is nonnegative and has total support. Though Sinkhorn and Knopp were the first to formalize the convergence results, this algorithm has had a very long history; see [20, 21] for a historical review. The literature on "iterative proportional fitting," dating back to the 1940s, also includes many related methods [22, 23].

Note that the Sinkhorn-Knopp algorithm has been independently rediscovered many times. Recently, two groups working on Hi-C data used matrix balancing for normalization, Cournac, Marie-Nelly [24] and Imakaev, Fudenberg [25]. Cournac et al. calculated the row-specific normalization term R_i using the L_2 norm of the row vector instead of the traditional L_1 norm used in VC normalization and by Sinkhorn and Knopp. Imakaev et al. use the standard Sinkhorn-Knopp-style approach in which VC normalization is repeatedly performed until convergence is achieved.

III.a.2.iii. New methods for matrix balancing: Within the matrix balancing literature, dramatic algorithmic improvements have been made since the 1960s. Recently, Knight and Ruiz introduced a new matrix balancing algorithm whose convergence properties closely resemble those of the Sinkhorn-Knopp algorithm, but which converges much faster [26]. Their approach applies a combination of the inexact Newton's method and inner-outer iteration with conjugate gradients to a system of linear equations to quickly find the next matrix in the iteration process. Empirically, their method converges two orders of magnitude faster than the Sinkhorn-Knopp algorithm [26]. They provide MATLAB code for their method (hereafter referred to as "KR normalization"), which we reimplemented in Java and incorporated into our pipeline.

KR normalization always results in a balanced matrix as long as the original matrix is not too sparse. We only observed sparsity issues at very high resolutions, which we handled by throwing out the sparsest rows (up to 5% of the total number of rows) and rerunning the algorithm. Because the KR algorithm is so fast and

numerically stable, it makes it possible to reliably balance our Hi-C contact matrices at extremely high matrix resolutions.

III.a.3. Diploid and Higher Order Maps

III.a.3.i. Construction of Diploid Hi-C maps: Any cell line for which we are given a set of phased SNPs can be divided into diploid Hi-C maps. Each parental allele will have its own Hi-C map, which can be compared to each other and to other cell lines. Given a SNP list, we examine all Hi-C contacts filtered by MAPQ ≥10 that overlap a SNP within the first 70 nucleotides of the read. For each overlapping read, we locate the nucleotide at the appropriate position and matched it to the paternal or maternal SNP. If it didn't match either paternal or maternal SNP, we exclude it from further processing.

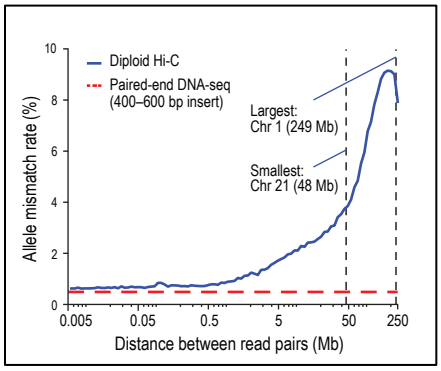


Figure 8. Allele mismatch rate

One way of judging the quality of our diploid assignments is to look at the allele mismatch rate for intra-chromosomal reads. When read ends are close together and both overlap SNPs, we expect that the read is almost always intramolecular and thus we should see the same allelic assignment. A typical graph of allelic mismatch rate by distance is shown in Figure 9. Our observations suggest that the quality of diploid Hi-C maps depends significantly on the quality of the SNP annotation used.

III.a.3.ii. Higher-order Contact Analysis: The Juicer pipeline annotates Hi-C contacts that map to three or more places in the genome as ambiguously chimeric, and these are usually discarded from 2-d contact map analysis. For higher order contact analysis, these chimeric contacts can be examined to create a list of triples, quadruples, and quintuples in the genome (Figure 10). Due to the overwhelming contribution of triples as compared to other chimeras, further description of the analysis is limited to triples.

In order to ensure that no duplicate triple reads enter the analysis, we first create a 'pseudo genome', concatenating all of the triple reads. We then use bwa [27] to align each of the individual chimeric triples to the pseudo genome. Reads that aligned to multiple places (alpha ≤ 0.04) are discarded as duplicates.

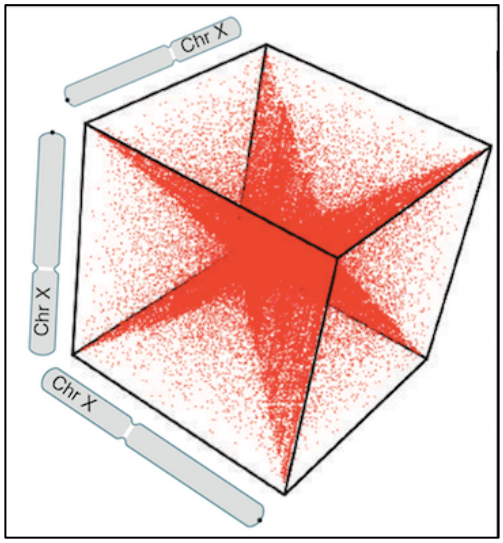


Figure 9. Higher order contact map. The bright diagonal seen in 2-dimensional contact matrices naturally manifests as an n-dimensional hyperstar.

For the most stringent filtering of higher order contacts, we require that all alignments were high-quality (MAPQ >10), all loci were on the same chromosome, and that the distance between any pair of loci was at least 20 kb. Enrichment analysis of peak data is performed with respect to a global and a local expected model, using a Poisson distribution for significance testing. In the global model, the expected number of reads is calculated based on the number of triples with anchors separated by the same distances as the putative anchors in question. Since the global model does not account for possible biases due to pairwise contacts (for example contacts between any three loci will be enriched with respect to a global expected if two loci form a loop), we also calculate an expected value based on the local neighborhood of the putative triple (local model), closely following the HiCCUPs strategy described in section III.b.2. HiCCUPS: Hi-C Computational Unbiased Peak Search for Peak Calling.

The local model assesses whether the observed enrichment in a 3D voxel can be explained solely by a combination of 1D coverage biases and 2D looping biases (increased pairwise interaction frequency). This model computes the bias enrichment at site (A,B,C) as f_{ABC} such that the expected number of reads, E_{ABC}, is $f_{ABC}*M$, where M is the mean number of reads in the local neighborhood. This model assumes that f_{ABC} is composed only of independent 1D and 2D biases:

$f_{ABC} = f_A*f_B*f_C*f_{AB}*f_{BC}*f_{AC}$

$E_{ABC} = f_{ABC} * M$

We estimate each of the terms on the right hand side as follows.

f_A is the 1D bias of A, i.e. the fold enrichment of contacts involving locus A with respect to the local mean; this can be estimated by computing the average number of reads found in the plane located at A, which we denote as <A>, and dividing this by the local mean. Thus $f_A \approx <A>/M$. (And similarly for 1D biases for B and C.)

f_{AB} represents the 2D bias of AB, i.e. the fold enrichment in contacts between pairs of loci A and B, above what would be expected based on 1D biases. One is tempted to estimate this by simply computing the average enrichment of AB with respect to the mean, which would amount to taking the average in the AB line, which we denote as <AB>, and dividing by the local mean. However, the enrichment in <AB> is due to a combination of 1D biases of A and B, and an independent 2D bias. Similar to the expected number of reads in a voxel based on known biases, we consider the expected number of reads for two loci (in the AB line) as
$E_{AB} = f_{AB} * f_A * f_B * M$
We estimate $E_{AB} \approx <AB>$ and so $f_{AB} \approx <AB> / (f_A * f_B * M) \approx <AB>*M/(<A>*<B>)$.

Combining all terms we find (see Figure 10 for visual explanation):
$E_{ABC}=f_{ABC} *M= <AB>*<BC>*<AC>/(<A>*<B>*<C>)*M$.

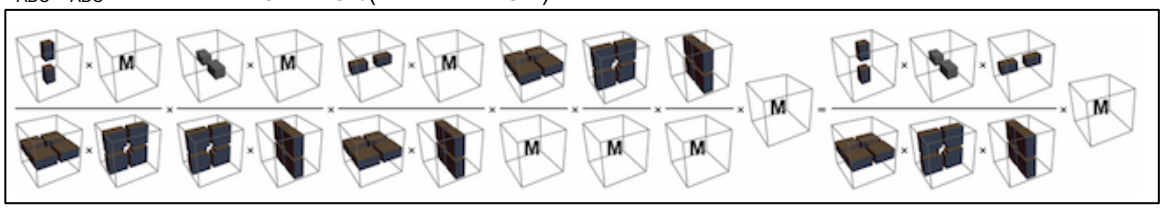


Figure 10. A schematic derivation of the expected number of reads in the center of a 3D tensor according to the local model.

To compute M, the local mean, we choose either the mean number of reads in the entire cube, or the number of reads in the cube's corner that is closest to the diagonal, based on whichever yields a more conservative (higher) value for the expected enrichment.

III.a.4. Cluster Systems

The Juicer pipeline is optimized for parallelized computation on a cluster; currently, it is available for Univa (or Sun Grid), LSF, SLURM, and in the cloud at Amazon Web Services. A single node version exists but is not practical for loop-resolution data sets. At the other extreme, we currently have a powerful IBM computer, Voltron, with a FPGA for aligning the data that can finish alignment very quickly.

The pipeline can be easily tuned to the particular job and memory limits of a given cluster, by sending in different parameters. We compared the performance of three different cluster systems running the Juicer pipeline for processing Hi-C data: Amazon Web Services (AWS) Intel-based OpenLava cluster, Rice IBM Power8-based SLURM cluster, and Broad Institute Intel-based Univa GE (UGER) cluster. As detailed in Table 5, these clusters are each using different operating systems, hardware configurations, and cluster management software; performance is determined by the unique combination of software and hardware in each system. In particular, AWS is a dedicated system, whereas the performance of the clusters at Rice and Broad is affected by how many other users are sharing resources. We provide the code for Juicer on all three systems in our GitHub repository.

We used the public IMR90 dataset from [10], which contains more than 1.5 billion paired-end reads. We generated contact maps down to 5kb resolution together with the list of contact domains returned by Arrowhead (see section III.b.1. Arrowhead Algorithm for Domain Annotation) and the list of loops returned by HiCCUPS (see section III.b.2. HiCCUPS: Hi-C Computational Unbiased Peak Search for Peak Calling).

System	Amazon Web Services g2.8xlarge			Broad Univa Grid Engine			Rice PowerOmics			Rice PowerOmics + FPGA		
CPU	Intel Xeon E5-2670 @2.60GHz			Intel Xeon X5650 @2.66GHz			IBM POWER8E@2.061GHz revision : 2.1			IBM POWER8E@2.061GHz revision : 2.1		
Cores/ node	4x8 cores			4x6 cores			2x24 cores			2x24 cores		
RAM	60GB			32GB			256GB			256GB		
Cluster OS	OpenLava 2.2 (LSF Compatible)			UGE 8.3.0			Slurm 14.11.8			Slurm 14.11.8		
GPU	NVIDIA Quadro K5000			None			NVIDIA Tesla K80			NVIDIA Tesla K80		
FPGA	None			None			None			Edico Genome DRAGEN Bio-IT Platform		
Max Parallel Cores	32			1200			1536			1536		
	Core Hours	**RAM**	**VM**	**Core Hours**	**RAM**	**VM**	**Core Hours**	**RAM**	**VM**	**Core Hours**	**RAM**	**VM**
Align	8744:49	12.3	13.5	11614:07	10.8	11.9	4221:29	13.1	14.0	1:29	0	0
Merge Sort	35:36	9.9	10.1	117:03	8.7	198.1	452:13	14.0	120.0	426:30	30.0	120.0
Duplicate Removal	12:21	0.5	0.5	17:04	0.4	0.5	3:12	0.4	0.0	1:28	0.4	0.0
.hic Creation	112:43	21.8	34.9	209:43	13.4	19.5	139:17	19.3	8	177:04	19.3	8
Feature Annotation	2:07	10.5	139.3	1:04	6.4	19.5	3:25	4.2	9.1	4:28	77.1	9.1
Total	8906:11			11959:01			4819:36			608:59		

Table 5. Using Juicer to process 1.5 billion paired-end Hi-C reads on different cluster systems. "RAM (Gb)" (resp., "VM(Gb)") are the maximum RAM (resp., virtual memory") used for each task. Loop annotation was not performed on the Broad cluster, which does not offer GPUs.

As described above, Juicer first splits the reads into subsets of 90 million reads each and aligns them in parallel to the human genome reference. Ligations are also counted during the alignment phase. After alignment, each pair is merged into single sorted files and these are then merged into one large file. In the next step, duplicates are removed in parallel. The Hi-C Creation phase consists of using the resulting file to calculate statistics and create the normalized contact maps, which are stored in the *hic* file. Features are then annotated using the *hic* file as input. On systems without GPUs, only contact domains are annotated.

III.b. Feature Annotation and Other Downstream Analysis Methods

III.b.1. Arrowhead Algorithm for Domain Annotation

III.b.1.i. Motivation and related work: The formation of square megadomains along the diagonal of a contact map is a striking feature that was apparent in our 2009 maps, and which we explained in terms of compartmentalization [9]. Subsequent work has highlighted the computational problem of identifying domains (which manifest as squares along the diagonal of a contact map) as distinct from the problem of identifying compartments [28, 29].

The fact that domains manifest as squares along the diagonal of a contact map suggests that they should be straightforward to identify. In practice, however, the identification of domains is tricky. This is due to experimental factors such as noise and inadequate coverage. It is also because of the intrinsic difficulty of the problem: the decline in contact frequency at domain edges can be subtle, and the very rapid decline in contact probability observed as one moves away from the diagonal of a contact map is a major confound for most approaches.

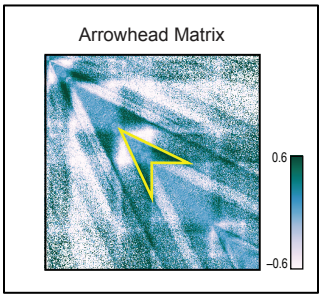


Figure 11. Arrowhead transformation. This transformation replaces domains with an arrowhead-shaped motif pointing toward the domain's upper-left corner

Nevertheless, several methods exist for identifying domains. Notably, Dixon, Selvaraj [29] defined a directionality index (DI), which measures the tendency of a locus to interact with upstream vs. downstream sites. This is useful for identifying domains because the upstream boundary of a domain should prefer to interact with downstream loci, and vice-versa.

III.b.1.ii. Description of Arrowhead transformation: The arrowhead transformation (see Figure 12) is a matrix transformation defined as $A_{i,i+d} = (M^*_{i,i-d} - M^*_{i,i+d})/(M^*_{i,i-d} + M^*_{i,i+d})$. This transformation can be thought of as equivalent to calculating a matrix equal to -1*((observed/expected)-1), where the expected model controls for local background and distance from the diagonal in the simplest possible way: the "expected" value at *i,i+d* is simply the mean observed value at *i,i-d* and *i,i+d*. By choosing variants on this expected model, one can create a family of related transformations with similar properties. Alternatively, one can think of $A_{i,i+d}$ as a measurement of the directionality preference of locus *i*, restricted to contacts at a linear distance of *d*.

Consider the behavior of this transformation when a domain is present in M^* between locus *a* and locus *b* (i.e., there is a square of enriched contact frequency whose vertices lie at <*a,a*>, <*a,b*>, <*b,b*>, and <*b,a*>). $A_{i,i+d}$ will be strongly positive if and only if locus *i-d* is inside the domain (i.e., in the range *[a,b]*) and locus *i+d* is not. $A_{i,i+d}$ will be strongly negative when locus *i+d* is inside the domain and locus *i-d* is not. If both loci are inside the domain, or both loci are outside the domain, $A_{i,i+d}$ will be close to zero. (Note that this behavior also exploits the fact that one typically observes squares of depleted contact frequency adjacent to domains.)

Thus, the general behavior of the arrowhead matrix A can be seen by solving a series of simple inequalities that follow from the above statement. If we think of the solution geometrically, we see that *A* takes on very negative values inside an "upper" triangle $U_{a,b}$, whose vertices lie at *[a,a]*, *[a,b]*, and *[(a+b)/2,b]*. We also see that *A* takes on very positive values inside a "lower" triangle $L_{a,b}$, whose vertices lie at *[(a+b)/2,b]*, *[b,b]*, and *[b,2b-a]*. Everywhere else, the entries of *A* are close to zero.

One can think of the "upper" and "lower" triangles as a smear that exaggerates the original edges of the domain, making these features easier to detect. The negligible values seen everywhere else also have an important consequence: they replace the steep decline seen inside a domain in M^* – which tends to confound feature detection algorithms – with a relatively constant region in *A*. Because of the mirror symmetry of the matrix *A*, the effect of the transformation, when examined as a whole, is to transform an (abnormally-hard-to-annotate) square feature into a (relatively-easy-to-annotate) arrowhead shaped feature.

III.b.1.iii. Arrowhead scoring: The goal of our algorithm is to identify the pairs of loci *a* and *b*, where there is a domain between *a* and *b* (equivalently, where the pixel $M^*_{a,b}$ is the corner of a domain). As noted above, it is useful to apply the arrowhead transform to M^*, yielding the arrowhead matrix *A*. Every domain will produce the two triangles $U_{a,b}$ and $L_{a,b}$ described above. By

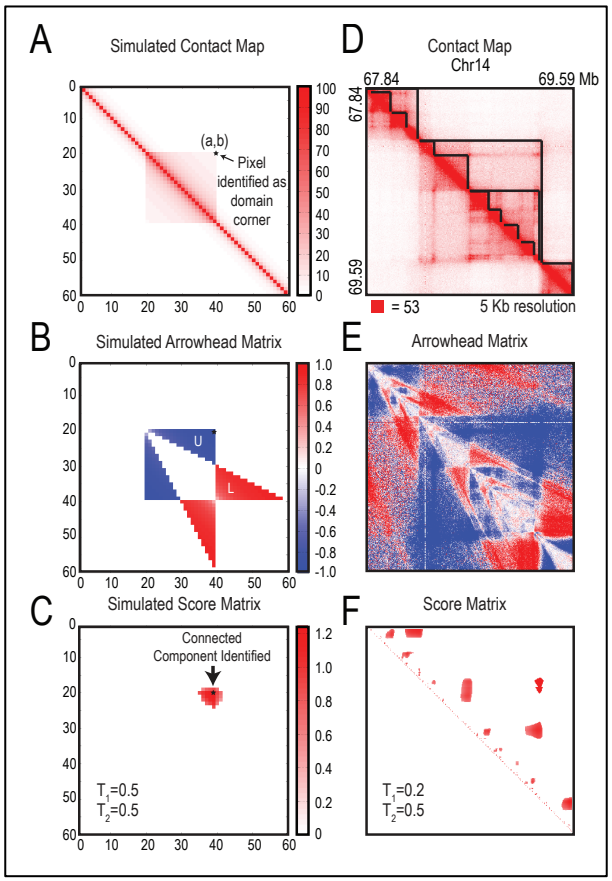


Figure 12. (A–C) We apply the Arrowhead algorithm to a simulated contact map (A); the result is the Arrowhead matrix (B), where the domain has been transformed into a pair of triangles of opposite signs, named U and L. Using a heuristic scoring algorithm (with thresholds T1 = 0.5 and T2 = 0.5) we obtain the Score matrix (C), where the pixels located in the corner of the contact matrix appear as a patch of high values. The pixel with the maximum score in this patch is annotated as the domain corner, which in this case is exactly the corner of the simulated domain. (D–F) Same as on the left, except we apply the Arrowhead algorithm to a 1.75 Mb region in chromosome 14 in our in situ GM12878 map (D). Domains are transformed into arrowheads (or pairs of red and blue triangles) in the Arrowhead matrix (E). The Score matrix (obtained using T1 = 0.2 and T2 = 0.5) identifies areas of high corner likelihoods and the maxima are marked as domain corners (F). Our domain calls for this region are shown in black in (D).

empirically studying the results of A on a series of domains, we noted the following facts about $U_{a,b}$ and $L_{a,b}$:

(i) almost all entries in $U_{a,b}$ are negative, and almost all entries of $L_{a,b}$ are positive.
(ii) when the sum of the entries in $U_{a,b}$ is subtracted from the sum of the values in $L_{a,b}$, the resulting value is large (relative to a random model)
(iii) the variance of the entries in $U_{a,b}$ and $L_{a,b}$ were both small (relative to a random model).

These properties were not satisfied when $M^*_{a,b}$ was not a domain corner. We therefore used these three observations as a heuristic to find domain corners. To calculate the corner score for a pixel $M^*_{a,b}$, we first calculate a set of subscores for the corresponding $U_{a,b}$ and $L_{a,b}$: $S_{sign,}$ the sum of the signs of entries in $L_{a,b}$ minus the sum of the signs of the entries in $U_{a,b}$; $S_{sum,}$ the sum of the values of entries in $L_{a,b}$ minus the sum of the values of entries in $U_{a,b}$; and $S_{variance,}$ the total variances of both $U_{a,b}$ and $L_{a,b}$. We normalize each of these three subscores by calculating each score for every possible a,b, and then dividing by the maximal value observed. The "raw corner score" matrix S' comprises the sum of the three normalized scores for all pixels $M^*_{a,b}$. If $M^*_{a,b}$ is a true domain corner, the value of $S'_{a,b}$ will typically be large.

To identify domain corners using the corner score, we create a filtered version of the matrix S', labeled S, in which we set all pixels whose individual subscores do not pass certain thresholds to zero. These thresholds were determined empirically; we believe most of the genome to be partitioned into domains, but erred on the side of fewer false positives when choosing thresholds. We apply thresholding twice, and in each round choose two thresholds, t_1 and t_2. In the first pass, we look for small, very distinct blocks with low variance ($S_{variance}$ <0.2 = t_1; Mean(sgn($U_{a,b}$))<-0.5 = -t_2.; Mean(sgn($L_{a,b}$))>0.5 = t_2.). In the second pass, we identify larger blocks (Mean(sgn($U_{a,b}$))<-0.4 = -t_2.; Mean(sgn($L_{a,b}$))>0.4 = t_2.). These larger blocks are not permitted to contain any of the previously annotated smaller blocks.

When we examine the matrix S, we find that corners of domains appear as blobs of high scoring pixels. To precisely annotate domain corners, we first use MATLAB's connected component algorithm to identify groups of adjacent pixels. The pixel within the connected component whose corner score S is largest is marked as the domain corner. See Figure 13.

iii.b.1.iv. Dynamic programming for fast calculation: Naively, the above algorithm would require us to calculate all the above noted scores for $U_{a,b}$ and $L_{a,b}$ for all a,b. Thus, the naive running time of the above algorithm is O(n^4), where n is the number of loci in the genome. This makes the algorithm infeasible on large-scale datasets.

However, we developed a dynamic programming implementation of this scheme which requires only O(n^2) operations, which makes the algorithm much more useful in practice.

To create a more practical implementation, we realized that summing entries of a matrix contained in $U_{a,b}$ and $L_{a,b}$ can be thought of as summing the calculations for smaller triangles, plus a sum for the additional row or column. In particular, given the sum for $U_{a,b-1}$, we add the column b sum from rows (a+b)/2 to a, and similarly for $L_{a,b-1}$. The additional column and row sums are themselves calculated ahead of time via dynamic programming and then accessed when needed to calculate sums for all possible $U_{a,b}$ and $L_{a,b}$

This approach can be applied very broadly. For instance, the variance of the entries in $U_{a,b}$ and $L_{a,b}$, the score matrix $S_{variance}$, can be calculated using dynamic programming by transforming the problem into a sum, relying on the fact that $Var(X) = E[(X-\mu)^2] = E[X^2] - (E[X])^2$.

By exploiting this method, all the above scores can be calculated using only O(n^2) operations.

iii.b.1.v Characteristics of domains in Hi-C maps: In our earlier Hi-C experiments at 1 Mb map resolution, we saw large squares of enhanced contact frequency tiling the diagonal of the contact matrices. These squares partitioned the genome into 5–20 Mb intervals, which we call ‘‘megadomains.’’ We also found that individual 1 Mb loci could be assigned to one of two long-range contact patterns, which we called compartments A and B, with loci in the same compartment showing more frequent interaction. Megadomains—and the associated squares along the diagonal—arise when all of the 1 Mb loci in an interval exhibit the same genome-wide contact pattern. Compartment A is highly enriched for open chromatin; compartment B is enriched for closed chromatin.

In our new, higher resolution maps (200- to 1,000-fold more contacts), we observe many small squares of enhanced contact frequency that tile the diagonal of each contact matrix. We used the Arrowhead algorithm to annotate these contact domains genome-wide. The observed domains ranged in size from 40 kb to 3 Mb (median size 185 kb). As with megadomains, there is an abrupt drop in contact frequency (33%) for pairs of loci on opposite sides of the domain boundary.

III.b.2. HiCCUPS: Hi-C Computational Unbiased Peak Search for Peak Calling

Much of the work on genome architecture so far has centered on the study of chromatin looping. Chromatin loops manifest as local peaks in a proximity ligation dataset, which occur between two points whenever they interact with each other significantly more than with random points in their neighborhood.

HiCCUPS (Hi-C Computational Unbiased Peak Search) is a local peak caller that integrates information from multiple local neighborhoods in its peak calling procedure and handles multiple hypothesis testing.

Our peak-calling algorithm examines each pixel in a Hi-C contact matrix and compares the number of contacts in the pixel to the number of contacts in a series of regions surrounding the pixel. The algorithm thus identifies pixels $M^*_{i,j}$ where the contact frequency is higher than expected, and where this enrichment is not the result of a larger structural feature. For instance, we rule out the possibility that the enrichment of pixel $M^*_{i,j}$ is the result of L_i and L_j lying in the same domain by comparing the pixel’s contact count to an expected model derived by examining the “lower-left” neighborhood. (The “lower-left” neighborhood samples pixels $M_{i',j'}$ where i≤i’≤j’≤j; if a pixel is in a domain, these pixels will necessarily be in the same domain.) We require that the pixel being tested contain at least 50% more contacts than expected based on the lower-left neighborhood, and that the enrichment be statistically significant after correcting for multiple hypothesis testing (FDR<10%). The same criteria are applied to three other neighborhoods. To be labeled an “enriched pixel,” a pixel must therefore be significantly enriched relative to four neighborhoods: (i) pixels to its lower-left; (ii) pixels to its left and right; (iii) pixels above and below; and (iv) a donut surrounding the pixel of interest (Figure 14).

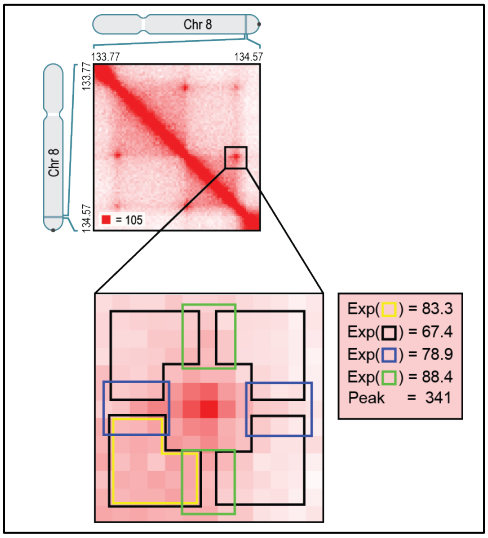


Figure 13. Local neighborhoods for peak calculation. We identify peaks by detecting pixels that are enriched with respect to four local neighborhoods (blowout): horizontal (blue), vertical (green), lower-left (yellow),and donut (black).

The resulting enriched pixels tend to form contiguous interaction regions comprising 5-20 pixels each. We define the “peak pixel” (or simply the “peak”) to be the pixel in an interaction region with the most contacts. Because over 10 billion $(10Kb)^2$ pixels must be examined, this calculation requires weeks of CPU time to execute. To accelerate it, we created a highly parallelized implementation using general-purpose graphical processing units, resulting in a 200-fold speedup.

HiCCUPS calculates multiple local expectation values for every pixel in order to rule out the possibility that another local feature, such as the edge of a domain, could lead to a spurious, peak-like enrichment. HiCCUPS uses a modified Benjamini-Hochberg FDR control procedure, dubbed “λ-chunking,” which is specifically designed to work with the unique statistical structure of Hi-C data and rigorously enforces thresholds of local enrichment.

III.b.2.i. Local expected value calculation. In order to assess the local background and thus the level of contact frequency one would expect to see in a given pixel if a peak was not present, we examine contacts in a donut-shape around the pixel. This establishes a local baseline level of interaction that is seen between neighboring loci.

We begin with a normalized contact matrix M^* whose corresponding one dimensional expected matrix E^* has been calculated.

For every pixel $M^*_{i,j}$ where $|i-j| \leq 2$ Mb, the local expected is calculated by sampling pixels in a donut surrounding $M^*_{i,j}$ as follows:

$$donut\ filter: \quad E^{d*}_{i,j} = \frac{\sum_{a=i-w}^{i+w}\sum_{b=j-w}^{j+w} M^*_{a,b} - \sum_{a=i-p}^{i+p}\sum_{b=j-p}^{j+p} M^*_{a,b} - \sum_{a=i-w}^{i-p-1} M^*_{a,j} - \sum_{a=i+p+1}^{i+w} M^*_{a,j} - \sum_{b=j-w}^{j-p-1} M^*_{i,b} - \sum_{b=j+p+1}^{j+w} M^*_{i,b}}{\sum_{a=i-w}^{i+w}\sum_{b=j-w}^{j+w} E^*_{a,b} - \sum_{a=i-p}^{i+p}\sum_{b=j-p}^{j+p} E^*_{a,b} - \sum_{a=i-w}^{i-p-1} E^*_{a,j} - \sum_{a=i+p+1}^{i+w} E^*_{a,j} - \sum_{b=j-w}^{j-p-1} E^*_{i,b} - \sum_{b=j+p+1}^{j+w} E^*_{i,b}} \times E^*_{i,j}$$

For a simple visual diagram of the pixels sampled by this local neighborhood, see Figure 14. The $M^*_{a,b}$ terms in the numerator are counts from the normalized contact matrix. The remaining terms use the one-dimensional model E^* to correct for the fact that the sampled pixels and the central pixel $M^*_{i,j}$ may be at different distances from the diagonal.

The parameters p and w in the above equation specify the width of the interaction region surrounding the peak and the size of the donut sampled, respectively. Both p and an initial minimum w are given as inputs to the algorithm. Both parameters must be multiples of the matrix resolution. We always set p to correspond to a 20-25 kb distance (at 25 kb resolution, we set $p=1$; at 10 kb resolution, we set $p=2$; at 5 kb resolution, we set $p=4$) based on empirical observation of the size of true interaction regions. While an initial minimum w is provided as input to the algorithm, w is set individually for each pixel by starting at the minimum w, checking if the sum of pixels in the lower left region of M^* is ≥16 reads, and if not (suggesting an overly noisy estimate) incrementing w by 1. This is repeated until either the sum of the pixels is ≥16 reads or w=20. Our initial minimum w is set to 3 at 25kb resolution, to 5 at 10 kb resolution, and to 7 at 5 kb resolution. We only include pixels in the upper triangle of the contact matrix in our calculations, i.e. pixels where $j>i$. Thus any pixels inside the neighborhood window where $j \leq i$ are automatically excluded from the calculations. Furthermore, in order to ensure that sufficient numbers of pixels are included inside our local neighborhood, we only examine pixels that are between loci $>p+2$ apart (measured in units of resolution; i.e., at 25 kb resolution, p=1, and (1+2)*25 kb=3*25 kb implies that we only examine pixels between loci ≥75 kb apart; at 10 kb resolution, we similarly examine pixels between loci ≥50 kb apart; at 5 kb resolution, we examine pixels between loci ≥35 kb apart).

III.b.2.ii. Additional local neighborhoods: We realized that the donut neighborhood described above led to a poor expected model when the target pixel was near the perimeter of a large-scale feature.

To remedy this HiCCUPS examines four neighborhoods surrounding the pixel of interest to determine if it is enriched. These neighborhoods also serve as "filters": by requiring certain enrichment thresholds with respect to a particular neighborhood, we can filter out certain types of enrichment artifacts. We note that we only sample pixels in the upper half of the symmetric contact matrix, so for example an area to the "lower left" of the pixel represent contacts between loci closer together.

The lower left neighborhood, which is just the lower left quadrant of the donut neighborhood, was designed to ensure that pixels inside domains were not erroneously identified as peaks. This neighborhood takes advantage of the fact that if pixel $M_{i,j}$ is located in the interior of a domain, then all the pixels located to its lower left (i.e. pixels $M_{i',j'}$ such that $i'>i$ and $j'<j$) will also be located inside of the domain. As such this neighborhood gives a more accurate assessment of the local background for pixels contained within domains.

Quantitatively, the lower-left neighborhood is defined as follows:

$$lower\ left\ filter:\ E^{ll*}_{i,j} = \frac{\sum_{a=i+1}^{i+w}\sum_{b=j-w}^{j-1} M^*_{a,b} - \sum_{a=i+1}^{i+p}\sum_{b=j-p}^{j-1} M^*_{a,b}}{\sum_{a=i+1}^{i+w}\sum_{b=j-w}^{j-1} E^*_{a,b} - \sum_{a=i+1}^{i+p}\sum_{b=j-p}^{j-1} E^*_{a,b}} \times E^*_{i,j}$$

See Figure 14 for a visual diagram. This equation reflects a sum of all values at pixels in the lower-left neighborhood, corrected for the difference in distance from the diagonal between the neighborhood pixel and the target pixel *i,j*.

The vertical and horizontal neighborhoods are intended to ensure that peaks are not erroneously called on the edges of domains. These are defined as follows:

$$vertical\ filter:\ E^{v*}_{i,j} = \frac{\sum_{a=i-w}^{i-p-1}\sum_{b=j-1}^{j+1} M^*_{a,b} - \sum_{a=i+p+1}^{i+w}\sum_{b=j-1}^{j+1} M^*_{a,b}}{\sum_{a=i-w}^{i-p-1}\sum_{b=j-1}^{j+1} E^*_{a,b} - \sum_{a=i+p+1}^{i+w}\sum_{b=j-1}^{j+1} E^*_{a,b}} \times E^*_{i,j}$$

$$horizontal\ filter:\ E^{h*}_{i,j} = \frac{\sum_{b=j-w}^{j-p-1}\sum_{a=i-1}^{i+1} M^*_{a,b} - \sum_{b=j+p+1}^{j+w}\sum_{a=i-1}^{i+1} M^*_{a,b}}{\sum_{b=j-w}^{j-p-1}\sum_{a=i-1}^{i+1} E^*_{a,b} - \sum_{b=j+p+1}^{j+w}\sum_{a=i-1}^{i+1} E^*_{a,b}} \times E^*_{i,j}$$

The parameters *p* and *w* in the above three neighborhoods are defined and treated in exactly the same way as we described for the donut neighborhood above.

As with the donut neighborhood procedure described above, we rescaled our calculated local expectations by the KR coverage factors for the target pixel in order to obtain a value for the number of raw contacts we expect to see; this is, as noted above, essential for the applicability of Poisson statistics. We next test the hypothesis that the number of contacts seen in M_{ij} is significantly enriched with respect to the expected value for a given neighborhood, i.e., with respect to a Poisson process whose parameter $\lambda = E^{*local}_{i,j} \times C'_i \times C'_j$. For HiCCUPS to call a pixel "enriched," we require that it be enriched with respect to all four neighborhoods.

III.b.2.iii. Multiple hypothesis testing: In order to perform multiple hypothesis correction while probing all intra-chromosomal pixels (rather than restricting to pixels between loci ≤ 2 Mb apart), we employ a modified FDR control strategy. While the Benjamini-Hochberg FDR control procedure is adequate for controlling the FDR when restricting to pixels between loci ≤ 2 Mb apart, it does not perform well after including all intra-chromosomal pixels. A basic issue is that the Benjamini-Hochberg procedure and other commonly used FDR control procedures are designed to control the FDR given that the hypotheses that are being tested are independent and identically distributed. However, in a Hi-C data set, this requirement is very dramatically violated. Due to the strong genomic distance dependence of contact frequency, testing whether every pixel is a peak involves testing hypotheses where the expectations range by nearly 6 orders of magnitude (from thousands of contacts expected in some pixels, to thousandths of contacts expected in others, all at 10 kb resolution in a single contact matrix). Additionally, Hi-C data has the property that there are orders of magnitude more low expectation hypotheses than high expectation hypotheses, because there are far more possible long-range interactions (i.e. far off the diagonal of the contact matrix) than potential interactions between closely spaced loci (i.e. close to the diagonal). We found that, because there are millions of low-expectation pixels for every high-expectation pixel, at the FDR thresholds at which we observe many easily apparent peaks (2-5 fold enrichment over an expectation of tens of reads), we see many low-expectation pixels with similar p-values arising by chance (2 or more reads over an expectation of thousandths of reads). To get rid of the spurious low-expectation pixels, the FDR must be set so high that no pixels are called at all. As such, FDR control procedures like Benjamini-Hochberg, where all hypotheses are lumped together, are not suitable for our data. (Although some widely used FDR control procedures exist that allow the independent-and-identically-distributed assumptions to be relaxed, these cannot address the dramatic range of hypotheses seen in Hi-C loop calling procedures.)

To overcome these challenges, we developed a method in which pixels are assigned to hypothesis families together with other pixels whose expected contact frequency – based on local background – is similar. We call this procedure "λ-chunking." Each pixel is assigned to a λ-chunk based on its expected value, *E*. All pixels with $E<1$ were placed in one bin. Subsequent bins were logarithmically spaced every $2^{1/3}$. Thus, for each type of expected value *E*, bin 1 contained all pixels with $E < 1$, bin 2 contained all pixels where $1 < E$

$<2^{1/3}$, bin 3 contained all pixels where $2^{1/3}< E <2^{2/3}$, and so on. We then perform an FDR procedure separately for each λ-chunk. Within each λ-chunk, the distribution of observed counts over all pixels was compared to a null Poisson distribution with λ equal to the maximum expectation assigned to the bin (i.e. for bin 1, we compared to a null Poisson distribution with λ=1; for bin 2, we compared to a null Poisson distribution with $\lambda=2^{1/3}$, and so on). An FDR threshold corresponding to 10% FDR was identified by finding the minimum value *t* such that the integral of the null Poisson distribution from *t* to ∞ was less than 0.1 times the integral of the distribution of observed counts from *t* to ∞. This procedure is essentially equivalent to applying the Benjamini-Hochberg FDR control procedure at an FDR rate of α=.1 on each λ-chunk independently and then combining the results (albeit with a slightly modified calculation of the p-values in each λ-chunk).

After identifying the FDR thresholds for each bin separately and repeating this process separately for each local expected filter, pixels were identified as locally enriched if the number of contacts in the pixel was greater than the 10% FDR threshold on each of their four local expected values (i.e., the values obtained using the four neighborhoods). Because this enrichment was assessed for each neighborhood separately, the resulting procedure is stringent and markedly improves peak-call reliability.

III.b.2.iv. Additional filtering of peak pixels based on local enrichment thresholds: In order to be extremely stringent in our peak calls, we take pixels that were enriched with respect to all four local neighborhoods using the above FDR procedure and filter them further. In this additional filtering step, we remove all pixels that do not show a minimum fold-enrichment with respect to the expected values for each of the four neighborhoods. Thus, even pixels that show statistically significant enrichment after multiple hypothesis testing with respect to all four neighborhoods are excluded if they did not further show sufficient fold enrichment with respect to all four neighborhoods.

More specifically, we require that every pixel in our final annotation of peak pixels be enriched by at least 50% over the horizontal and vertical expected values, and at least 75% over the lower-left and donut expected values. Finally, we require that each pixel was at least 2-fold enriched above either the donut expected value or the lower-left expected value.

III.b.2.v. Additional filtering of "singlet clusters." We noticed that when the HiCCUPS algorithm called a pixel that was not part of a cluster of nearby pixels, that pixel was usually a false positive, i.e., true focal peaks typically lead to enrichment of multiple nearby pixels.

This was especially true if they had a higher q-value (>.01, q-value is defined as the minimum FDR threshold that the peak would be called at, i.e. the ratio of the integral of the expected distribution from the peak value to infinity to the integral of the observed distribution from the peak value to infinity) on at least one of the local filters. As such, we filter out peak pixels if there are no other enriched pixels that were collapsed into it and the sum of its four q-values (for the four local filters) is >.02.

III.b.2.vi. Combining peak annotations at different resolutions. In order to localize peaks as effectively as possible, but at the same time to annotate as many peaks as possible, we apply HiCCUPS at several resolutions. Depending on the strength of a peak and the sequencing depth of the map, some peaks might reach statistical significance at 5 kb resolution, while others might only reach statistical significance at 10 kb resolution or 25 kb resolution.

As such, we developed a method to combine peak lists generated at various resolutions, in a way that does not double count the same peak called at different resolutions. When combining peak lists at different resolutions, we always accept the highest resolution version of a peak. Thus, if two peaks are within 20 kb of each other (or within 50 kb if one was a 25 kb peak), we accept the higher resolution peak and discarded the lower resolution peak.

We found that the failure to call a peak at coarse resolution that was seen at fine resolution was in certain cases a warning sign of a false positive. Specifically, we throw out peaks between loci >100 kb apart that were called at 5 kb resolution but not at 10 or 25 kb resolution, because we found that such peaks were enriched for false positives. (Note that these long-distance "orphan" 5 kb peaks were usually no more than 5% of peaks called at 5 kb).

III.b.2.vii. Computational considerations: Searching all intrachromosomal pixels for local peaks at 10 kb resolution requires surveying roughly 10 billion pixels (at 5 kb resolution, the number is 40 billion). This is a computationally intensive but highly parallelizable process. We therefore code HiCCUPS using CUDA in order to perform these local expected calculations on a workstation containing 4 NVIDIA GPUs. Using an

NVIDIA Tesla C2075 GPU (which contains 448 cores and is capable of highly parallelized computation), we obtain a 200-fold speedup over the CPU implementation. HiCCUPS is also compatible with NVIDIA's faster Kepler architecture; we tested HiCCUPS on a Tesla K40c GPU. In general we have found that extensive genome-wide peak-calling on CPU architectures is extremely difficult and time-consuming.

III.b.2.viii Characteristics of loops in Hi-C maps: The vast majority of peaks (98%) reflect loops between loci that are <2 Mb apart and loops tend to be conserved across cell types. HiCCUPS was used to annotate 9448 peaks in our primary and replicate combined GM12878 map, 8040 peaks in IMR90, 5152 peaks in HMEC, 4929 peaks in NHEK, 6057 peaks in K562, 2634 peaks in KBM7, 3865 peaks in HUVEC, 3094 peaks in HeLa. Using our standard metric for overlap, we find that approximately 60% of peaks called in any cell type are conserved. It is expected that the minimum number one would expect for a Hi-C heatmap of adequate depth would contain at least 1000 loops.

We find that the vast majority of peak loci are bound by the insulator protein CTCF (86%) and the cohesin subunits RAD21 (86%) and SMC3 (87%). We also find that most peak loci encompass a unique DNA site containing a CTCF-binding motif, to which all three proteins (CTCF, SMC3, and RAD21) were bound (5-fold enrichment). The consensus DNA sequence for CTCF-binding sites is typically written as 5' - CCACNAGGTGGCAG-3'. We were thus able to associate most peak loci with a specific CTCF-motif "anchor." When we examined the 4,322 peaks in GM12878 where the two corresponding peak loci each contained a single CTCF-binding motif, we found that the vast majority (92%) of motif pairs are convergent. Overall, the presence, at pairs of peak loci, of bound CTCF sites in the convergent orientation was enriched 102- fold over random expectation. The convergent orientation was overwhelmingly more frequent than the divergent orientation, despite the fact that divergent motifs also lie on opposing strands: in GM12878, the counts were 3,971-78 (51-fold enrichment, convergent versus divergent); in IMR90, 1,456-5 (291-fold); in HMEC, 968-11 (88- fold); in K562, 723-2 (362-fold); in HUVEC, 671-4 (168-fold); in HeLa, 301-3 (100-fold); in NHEK, 556-9 (62-fold); and in CH12- LX, 625-8 (78-fold). This pattern suggests that a pair of CTCF sites in the convergent orientation are required for the formation of a loop. For this reason, Motif Finder can be used to assess the reliability of loop calls.

III.b.3. Motif Finder

For each peak annotation, we identify a list of loci involved in the peaks by separating each peak into its two component peak loci, removing non-unique loci from the resulting list, and merging any adjacent intervals into one larger interval.

We then compare the ENCODE uniform ChIP-Seq peak calls for transcription factors (TF). We iterate through our list of peak loci in that cell type and count how many peak loci had at least one called ChIP-Seq peak for the TF inside the locus. In cases where multiple replicate experiments were performed, we handle this in one of two ways: we either only allowed peaks called in all replicates (i.e. overlapping intervals in all replicates) for all TFs or we allow all peaks called in any replicate for all TFs. For peak loci smaller than 15 kb, we extend the locus on either side until the interval is 15 kb. Enrichments for each TF were then calculated by dividing the number of peak loci with a ChIP-Seq peak by the number of control loci with a ChIP-Seq peak. The fraction of peak loci containing a ChIP-Seq peak was calculated by dividing the number of peak loci with a ChIP-Seq peak by the total number of peak loci.

To localize peak loci down to a small number of nucleotides, in every peak locus we checked if there was one and only one CTCF ChIP-Seq peak (in the 15 kb+ window, only allowing ChIP-Seq peaks that were present in all replicates). If there was, we additionally required that the CTCF ChIP-Seq peak overlapped a RAD21 ChIP-Seq peak, an SMC3 ChIP-Seq peak and a good CTCF sequence motif (in cell types with no SMC3 ChIP-Seq data, we only required that the CTCF ChIP-Seq peak overlap a RAD21 peak and contain a good motif; in cell types with no RAD21 or SMC3 data; we only required a good motif). The presence of a good motif is ascertained using STORM [30]. For every candidate CTCF ChIP-Seq peak, we identified the underlying 20bp sequence with the highest match to the consensus CTCF position weight matrix (CTCF PWM from Kim, Abdullaev [31]). Positive scores returned by STORM indicate good alignments to the PWM used, and negative scores indicate bad alignments. We required that the best motif within candidate CTCF ChIP-Seq peaks have a positive PWM match score.

III.b.4. Eigenvector

The most common method used for classifying Hi-C patterns is the principal component (PC) approach, which we introduced in Lieberman-Aiden, van Berkum [9]. In this approach, each intrachromosomal contact

matrix is converted to an observed/expected matrix, and the first principal component of this matrix (given by the eigenvector) is used to bifurcate the data into two clusters. When this was performed at 1 Mb resolution on the original Hi-C maps, it was found that one cluster (A) was enriched for open chromatin marks, while the other cluster (B) was enriched for closed chromatin.

III.b.5. Subcompartments

V.b.5.i. Clustering Algorithm: To cluster loci based on long-range contact patterns, we construct a 100 kb resolution contact matrix *C* comprising a subset of the interchromosomal contact data. 100 kb loci on odd chromosomes appeared on the rows, and 100 kb loci from the even chromosomes appeared on the columns. The total length in base pairs of these two groups is roughly equal. (Chromosome X is excluded due to the differences in interaction pattern seen for the active and inactive homologs.) Thus $C_{i,j}$ represents the number of normalized contacts between the *i*-th locus on the odd chromosomes and the *j*-th locus on the even chromosomes. Genome-wide KR is used for normalization. Rows and columns for which more than 30% of the entries were either undefined or zeros are removed from the matrix. We then take the logarithm of each entry in the *C* matrix.

To cluster loci on the odd chromosomes, we apply the z-score function from Python's *scipy* library to each row of *C*. We use the resulting matrix as input to the *scikit-learn* library's unsupervised Gaussian hidden Markov model clustering algorithm (GaussianHMM) [32]. We set the covariance type to diagonal and allowed 1000 iterations.

To perform clustering on loci located on the even chromosomes, we begin by transposing the matrix *C* and then performed all steps exactly as they are performed for the clustering of odd chromosomes.

We find that each of the clusters on the odd chromosomes preferentially interacted with one of the clusters on the even chromosomes. This defines a one-to-one mapping between the odd and even cluster annotations.

The result of a clustering algorithm typically depends on the choice of a parameter, *k*, which determines the number of clusters to be identified. We use k=5 clusters; however, we also performed clustering using all values between k=2 and k=14 clusters as input. The Akaike Information Criterion and Bayes Information Criterion for the different cluster results clearly ruled out a value of k=2, and suggested a value of *k* between 4 and 8. Our final use of k=5 is based on this finding as well as careful examination of the data to determine how many clusters are necessary to explain the patterns that could be visually discerned. We find that, for k=5, the clusters corresponded to visually distinct patterns; this is no longer true if we increased *k* beyond 5. Nonetheless, it is possible that there are additional clusters that our algorithm cannot identify; our results should be considered a lower bound on the number of subcompartments rather than an exact determination. Clustering via k-means and hierarchical clustering yielded similar results.

III.b.5.ii. Creating an A/B Pattern Annotation: To define the pattern of a cluster, we use the first derivative of interactions along the linear genome. More precisely, for each locus *i* on an odd chromosome, we obtain its 1-dimensional interchromosomal interaction vector with all of the even chromosomes, C_i, and then calculate $d_i(j)=[C_i(j)-C_i(j-1)]$, where *j* and *j-1* are adjacent loci on an even chromosome. The intuition for using such a measure is based in how we expect the interaction vector C_i for a given locus to change (or switch) when it exits one cluster and enters another cluster. When locus *i* is interacting with a stretch of loci (on an even chromosome) that are all in the same cluster, the derivative is close to zero, as the amount of interaction, whether high or low, does not change. However, at the border between two different clusters, when *j* and *j-1* are in different clusters on the other chromosome, we expect $|d_i(j)|$ to be large. It is these switches that we use to determine cluster similarity. Using the derivative as a measure of pattern similarity is a simple way to account for the one-dimensional nature of the polymer. (This is akin to measures in finance that correlate returns of prices to identify similarities between stocks.)

To use this measure, we first create the difference matrix *D* by taking the difference between every adjacent pair of columns in the odd/even interchromosomal matrix *C* mentioned above. We then calculate a mean vector for each of the clusters (on the odd chromosomes) by averaging the rows of *D* for loci within the same cluster. Next, we examine the Spearman correlation of these mean derivative vectors for different pairs of clusters.

When examining the correlation matrix of the patterns, we find that the 5 patterns separate into two groups, with A1 and A2 in one group, and B1, B2 and B3 in the other. Patterns in a group correlate with each other

and anticorrelate with patterns of the other group. These correlations were confirmed by visually examining the five patterns. The first group of patterns correlated with the A compartment and the second group correlated with the B compartment.

III.b.6. Assembly

Our ability to adequately interpret Hi-C and other epigenetic data that facilitate the recognition of functional elements in the human genome depends crucially on the availability and correctness of the reference genome of the individual organism for which the data was produced. Although high quality reference genome assemblies have been produced for the species that the ENCODE focuses on ([33],[34],[35]), individual variation and cell population-specific genomic aberrations may hamper our ability to correctly map the reads, skewing the predictive models and data analysis.

Our team have recently released a method for inexpensive genome assembly of mammalian-scale genomes into chromosome-length scaffolds using Hi-C: the 3D de novo assembly (3D DNA) pipeline ([13]). The method relies on the fact that Hi-C data can provide links across a variety of length scales, spanning even whole chromosomes, thus allowing chromosome-scale assembly from relatively small contigs, including those that that can be produced from short-read Illumina PE250 DNA-seq data. A key aspect of the approach is to first use Hi-C data to identify and correct errors in the scaffolds of the initial assembly. Briefly, we correct misjoins by identifying positions where a scaffold's long-range contact pattern changes abruptly, which is unlikely for a correctly assembled scaffold. Next, we use a novel algorithm to anchor, order, and orient the resulting sequences, employing the contact frequency between a pair of sequences as an indicator of their proximity in the one-dimensional genome. Finally, the assembly pipeline includes an optional module to recognize alternative haplotypes in highly heterozygous species.

The algorithms and the overview of the workflow is given in detail in the supplementary materials of [13] and excerpted below. For the sake of generality, we will always refer to the inputs to our assembly algorithms as scaffolds.

IV.b.6.i Pipeline description: The assembly pipeline takes in as input the fasta file describing the draft assembly and the duplicate-free list of paired alignments of Hi-C reads to the draft fasta (merged_nodups.txt) as generated by the Juicer pipeline ([17]). For the sake of generality, we will always refer to the inputs to our assembly algorithms as scaffolds. This is meant in a broad sense; the algorithms are agnostic as to whether the inputs are scaffolds (sequences that are permitted to contain gaps), or contigs (gap-free sequences). Input scaffolds can come from a wide variety of sources and technologies.

In characterizing sets of input scaffolds, it is also useful to define the "effective N50 length" of the input scaffolds. This is simply the N50 of the scaffolds after all misjoins they contain have been corrected. Of course, for a typical published set of scaffolds, the effective N50 is not known, since it may contain misjoins and other scaffolding errors that the authors were unaware of. Naturally, the actual N50 of the scaffold set furnishes an upper bound for the effective N50 – but the two are often not equal in practice.

The overall strategy of our assembler is to remove misjoins until the underlying scaffold set is largely free of misjoins. If there is a disparity between the actual N50 length of the input scaffolds and their effective N50 length, it will be greatly reduced by this step. After misjoin detection, the resulting input scaffolds are used to create the final ordered-and-oriented chromosome-length scaffolds.

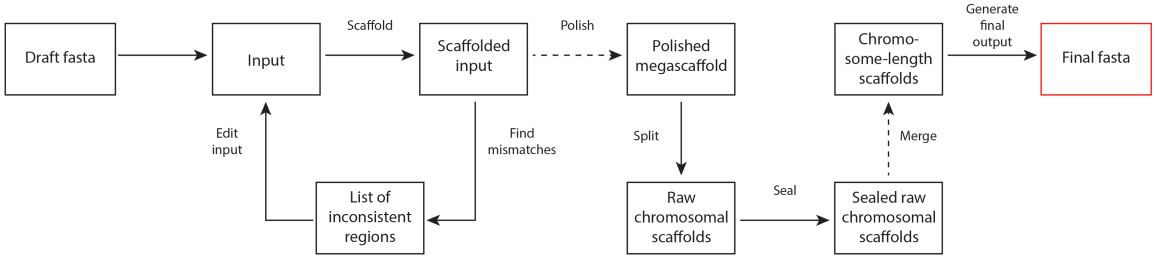


Figure 14. Workflow diagram describing the computational pipeline. The pipeline starts with scaffolding the input and assessing the result for misassemblies. The detected misassemblies are linked to misjoins in the draft contigs/scaffolds. Once the misjoins have been removed from the input, the scaffolding and misassembly analysis is repeated. Iterative scaffolding and misassembly detection constitute the core section of the pipeline. For some genomes polishing and merging is also employed. Additional steps include chromosome splitting and sealing (see Pipeline description).

An overview of the detailed workflow is schematically given in Figure 14. We begin with a series of iterative steps whose goal is to eliminate misjoins in the input scaffolds. Each step begins with a scaffold pool (initially, this pool is the set of input scaffolds themselves); the scaffolding algorithm is used to order and orient these scaffolds; and the misjoin correction algorithm is applied to detect errors in the scaffold pool. Finally, the edited scaffold pool is used as an input for the next iteration of the misjoin correction algorithm. The ultimate effect of these iterations is to reliably detect misjoins in the input scaffolds without removing correctly assembled sequence. After the iterations are complete, the scaffolding algorithm is applied to the revised input scaffolds, and the output – a single "megascaffold" which concatenates all the chromosomes – is retained for post-processing. This post-processing includes four step: (i) a polishing algorithm, which is required for genomes in the Rabl configuration; (ii) a chromosome splitting algorithm, which is used to extract the chromosome-length scaffolds from the megascaffold; (iii) a sealing algorithm, which detects false positives in the misjoin correction process, and restores the erroneously removed sequence from the original scaffold; and (iv) a merge algorithm, which corrects misassembly errors due to undercollapsed heterozygosity in the input scaffolds. Step (ii) is omitted for genomes that are not in the Rabl configuration; step (iv) is omitted if the original scaffolds lack substantial undercollapsed heterozygosity. See Figure 15 for an overview of the scaffolding terminology and workflow.

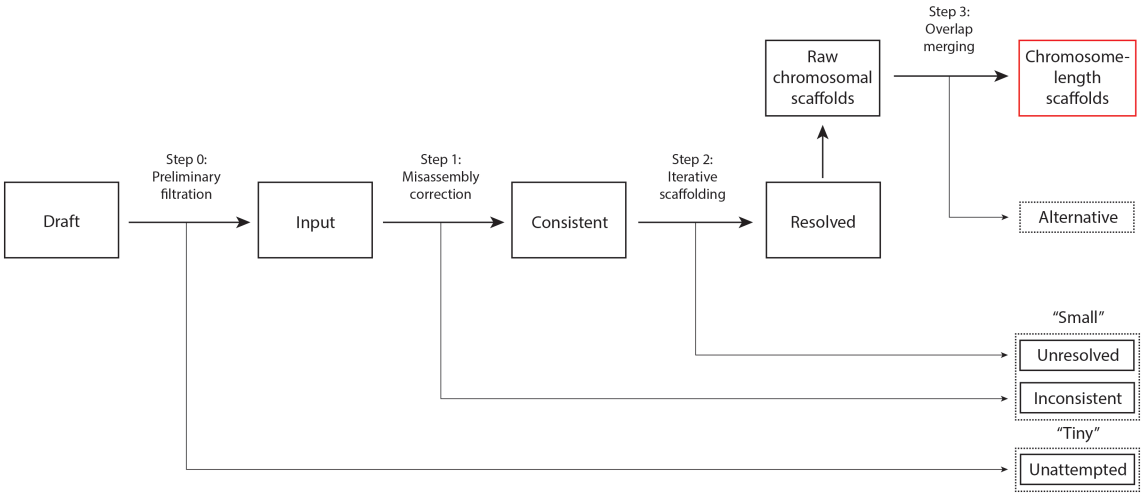


Figure 15. A workflow diagram illustrating the processing of various scaffold populations, beginning with draft scaffolds and ending with chromosome-length scaffolds. Each node corresponds to a set of scaffolds; the terminology used to refer to that set of scaffolds is shown.

Preliminary Filtration. First, a set of scaffolds (referred to as 'Tiny') is removed from the draft. Due to their small size, these scaffolds have relatively few Hi-C contacts, making them more difficult to reliably analyze. These are not processed further or included in the subsequent analysis.

Misjoin correction. The input scaffolds are examined for Hi-C signal consistent with a misjoin. Scaffolds with no evidence of a misjoin are labeled as 'consistent.' Scaffolds with evidence of a misjoin are partitioned into segments; each segment is classified as either a 'consistent' scaffold or an 'inconsistent' scaffold on the basis of the Hi-C signal. Inconsistent scaffolds are not processed further. (This partitioning procedure makes it possible to remove errors while retaining the portions of a scaffold that are correctly assembled for subsequent steps.) Note that the terms above specifically refer to the results of the last round of misjoin correction, not the results of intermediate rounds.

Ordering and orientation. The consistent scaffolds are then ordered and oriented; the results are polished (if needed), and chromosomes are extracted. As a result of this process, most of the consistent scaffolds are 'resolved', i.e., placed into a 'raw chromosomal scaffold', although a few remain 'unresolved'. The 'unresolved' scaffolds are not processed further. (However, note that although the unresolved scaffolds cannot be reliably localized within a chromosome-length scaffold, they can often be correctly associated with a particular chromosome. We do not do so in this manuscript.)

Overlap merging. Pairs of resolved scaffolds in each raw chromosomal scaffold are examined for overlaps. When an overlap is detected, the scaffolds are merged. The result of this process is the final 'chromosome-

length scaffold'. This step is crucial when assembling highly heterozygous genomes such as Ae. aegypti and Cx. quinquefasciatus.

The central components of the pipeline are the misjoin correction algorithm, the scaffolding algorithm, and the merging algorithm. We describe each of the three blocks in detail below. We also include additional sections describing polishing, sealing and chromosome splitting algorithms.

All of these steps are fully automated and available as open-source code; more generally, furthermore, code is available which deploys each of these steps on input scaffolds from AaegL2, CpipJ2, and Hs1, producing AaegL4, CpipJ3, and Hs2-HiC (respectively) in a single click. The current version of the pipeline is written in the AWK programming language in combination with bash scripting. (We use AWK because the pipeline requires extremely rapid i/o.) It is optimized for speed using GNU Parallel shell tool [36], but can be run without parallelization. The pipeline is designed to take full advantage of the Juicebox visualization software for Hi-C data and produces Juicebox-compatible heatmaps as well as various supplementary annotation outputs at every step. Obligatory external dependencies are the Command line tools for Juicebox and Juicer [16, 17] as well as the LASTZ sequence aligner [37].

III.b.6.ii. Algorithm for misjoin correction. The misjoin correction algorithm consists of two parts: (i) misjoin detection and (ii) editing of input scaffolds (i.e., misjoin correction).

Our method for misjoin detection using Hi-C relies on the fact that sequences which have been erroneously concatenated in a scaffold form fewer contacts with one another than correctly joined sequences. This is because correctly joined sequences lie adjacent to one another in 1D, and are therefore proximate to one another in 3D, facilitating the formation of Hi-C contacts. Because they do not actually lie in close proximity in the one-dimensional (1D) sequence of the chromosome, misjoined sequences usually do not exhibit similar 3D proximity or similar contact frequency.

III.b.6.ii.A. Calculating an expected model for contact frequency in the absence of an accurate genome. To detect this depletion in contact frequency, one must compare the observed contact frequency between adjacent genomic loci with an expected model that describes the contact frequency typically observed for correctly joined sequences. Given a genome assembly with chromosome-length scaffolds, calculating the expected frequency of contact for a typical pair of sequences at a particular distance during a given experiment is straightforward. Such calculations have been commonplace since our original paper on the Hi-C method [9].

However, the results of such contact probability calculations are influenced by disparate factors, ranging from the organism of interest, the cell population interrogated, the details of the experimental approach, the particular computational methods used to analyze the data, and seemingly random inter-experimental variability. For this reason, expected models derived from a particular experiment in a particular cell population in a particular organism cannot be reliably applied to all experiments in all cell populations in all species. Thus, in the absence of a genome assembly with chromosome-length scaffolds, it is unclear how to determine the relationship between contact probability and distance even if Hi-C data are available. To the best of our knowledge, no such calculations have been performed in the literature to date.

A second challenge is that the contact probability between a pair of loci varies greatly, with frequent "jackpot" effects where the number of contacts is markedly enhanced with respect to the background model. This variability makes raw contact probability a very noisy indicator of the presence of a misjoin.

To overcome these challenges, we developed a method that estimates contact probability, as a function of genomic distance, using data from a Hi-C experiment without utilizing a high quality genome. Instead, our method only assumes the availability of a collection of scaffolds that may be short and contain numerous errors. Specifically, we show that, even in this scenario, it is possible to calculate a lower bound for the expected number of contacts between a pair of loci at a given distance. Our estimation scheme relies on the fact that the frequency of contact between a pair of loci tends to decrease as the 1D distance between the loci increases. For this reason, pixels closer to the diagonal of a Hi-C matrix (which reflect contact frequency between loci that are nearby in 1D) tend to have higher contact counts than pixels further away from the diagonal.

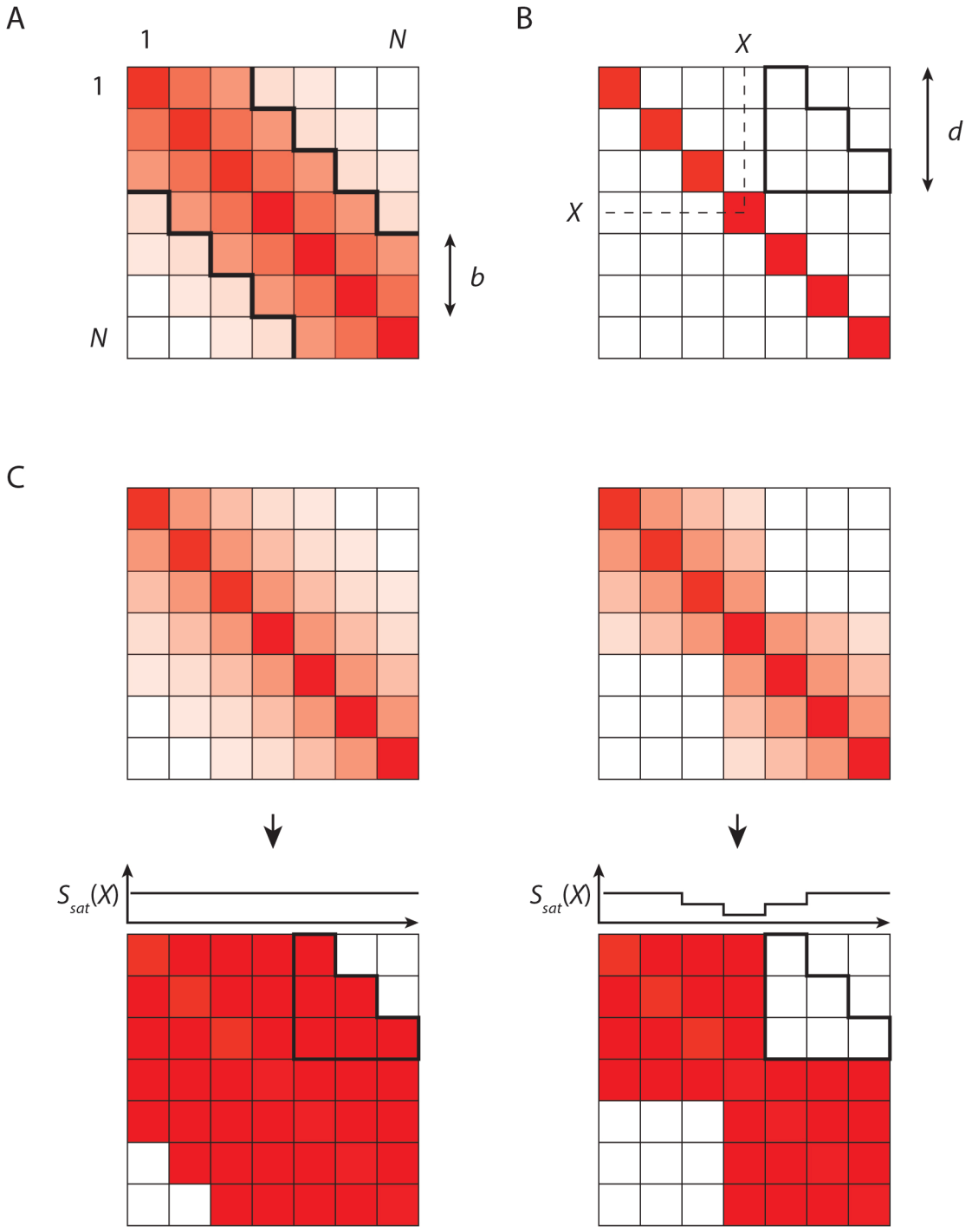


Figure 16. Misassembly detection notation and algorithm. (A) Calculating the number of bins in between the diagonals from $c_{(1+b,1)}$ to $c_{(N,N-b)}$ and from $c_{(1,1+b)}$ to $c_{(N-b,N)}$ (B) Triangular shape used to calculate the scores $S(X)$ and $S_{sat}(X)$ along the assembly. (C) Schematic representation of matrix saturation and the distribution of the score $S_{sat}(X)$ along the genome. Bright red signifies the highest scoring bin in a given matrix.

Consider a $N \times N$ Hi-C contact matrix M generated using a known, correct reference genome (Figure 16). To do so, the genome has been partitioned into N loci of fixed length that is matrix resolution r (measured in base pairs). Each pixel M_{ij} corresponds to all contacts between a pair of loci (in this case, the i^{th} locus and the j^{th} locus). Of course, N is simply the genome length divided by the matrix resolution, r. Note that, in such

a setting, it is often convenient to measure 1D distance in terms of loci (which are all of fixed size *r)*, which correspond to rows and columns of the matrix, rather than in terms of base pairs.

In such a matrix, we can consider the set of pixels that derive from pairs of loci that are within *b* loci of one another, i.e. the pixels M_{ij} such that ($i-b \le j \le i+b$). Our principal goal is to estimate the function *Q(b),* which is the minimum value of all these pixels: $Q(b)=min(M_{ij}),i-b \le j \le i+b$. This function provides a lower bound for the values M_{ij} for pixels within *b* of the diagonal. *Q(b)* is of obvious utility in identifying misjoins, for the following reason: if we were to align the Hi-C data against an incorrect reference genome, containing numerous misjoins, the presence of a value lower than *Q(b)* within *b* pixels of the diagonal would indicate the presence of a misjoin at that position with complete certainty.

Before we address the estimation of *Q(b)* in the general case, it is worth considering an idealized example. In Figure 16A we show an idealized Hi-C matrix *M'* where contact probability decreases monotonically as the distance between a pair of loci increases, and the shape of this decay does not vary across the genome. Notably, in such a matrix, the fraction of pixels M_{ij} that derive from pairs of loci that are within *b* loci of one another *(*$i-b \le j \le i+b$*)* can be calculated by simply summing the lengths of the principal diagonal and 2×*b* non-principal diagonals, and dividing by the size of the matrix as a whole (N^2). This yields:

$$F(b) = (N + b * (2N – b - 1))/N^2 .$$

Thus, if we select a pixel from the matrix *M* at random, the probability that the pixel lies within *b* loci of the diagonal is exactly *F*.

Similarly, it is possible to determine the probability that a random pixel in the matrix contains a value larger than any arbitrary threshold *C*, denoted *F'(C)*, by simply counting the number of pixels that contain more than *C* contacts and again dividing by the size of the matrix (N^2).

It is therefore possible to define a function *C(b)* so that *F'*(*C(b)) = F(b*). In other words, *C(b)* is the number of contacts such that the fraction of pixels in *M* that is larger than *C(b)* is the same as the fraction of pixels in *M* that are within *b* of the diagonal. Furthermore, in an idealized Hi-C matrix such as the one shown in Figure 16A, the pixels that lie within *b* of the diagonal will be exactly the pixels whose contact count is larger than *C(b).*

It follows from the above that – for an idealized, perfectly monotonic Hi-C matrix – *Q(b)* and *C(b)* are exactly the same function.

In practice, this is relevant because, like the contact probability scaling, *Q(b)* can be challenging to reliably estimate without an accurate genome assembly including chromosome-length scaffolds. By contrast, *F(b)* can be calculated analytically using the formula above, without any experimental data at all.

Moreover, *F'(C)* can be estimated for a given Hi-C experiment even assuming that a genome assembly with chromosome-length scaffolds is not available. In fact, *F'(C)* can be accurately estimated using almost any reference genome assembly, so long as the effective scaffold N50 is much larger than the matrix resolution *r*. A simple way to see why is that one can generate a proxy for the actual reference genome by concatenating all of the available scaffolds in an arbitrary order. In this proxy genome, the relative order and orientation of loci of size *r* will be entirely wrong. Nevertheless, most individual loci in the proxy genome will have a counterpart, containing the same sequence and having exactly the same size, in the true (albeit unknown) genome. For this reason, the vast majority of pairs of loci in the proxy genome will correspond to a pair of loci in the true (albeit unknown) genome. Thus, a Hi-C matrix generated with the proxy genome can be thought of as a permutation of the pixels of the Hi-C matrix that would be generated with the true genome. Consequently, the distribution of pixel values *F'(C)* is unaffected by the use of a scrambled proxy genome. (Note that in practice *F'(C)* can also be calculated from a raw scaffold set, without concatenation.)

Given estimates for *F'(C)* and *F(b)*, estimating *C(b)* is straightforward. Thus, it is possible to estimate *C(b)* even with a relatively poor, and error-prone, input genome. Although *C(b)* is not identical to *Q(b)* for a real Hi-C matrix, it nevertheless provides a serviceable estimate for *Q(b)*. For this reason, *C(b)* is useful in detecting misjoins.

III.*b.6.ii.B. Misjoin detection strategy.* Consider a fragment of the Hi-C map shown in Fig. S3 (B). One possible misjoin score would be to place a triangular motif along the diagonal, summing the values of the pixels it contains to create a score associated with the particular genomic position:

$$S(X) = \sum_{i=X-d}^{X-1} \sum_{j=X+1}^{i+d+1} c_{ij}.$$

This score reflects the average contact frequency between a particular index locus being examined (*X*), and all other loci within *d* loci of the index locus. If the value of the misjoin score *S* is anomalously low, it suggests that the corresponding index locus spans a misjoin. Unfortunately, there is no simple and reliable way to calculate an expected value for this particular score. Thus, it is impossible to know whether the score is indeed anomalously low. Moreover, this score is extremely sensitive to "jackpot" effects, when a pixel with an anomalously high value (such as a loop or an alignment error) falls within the triangular motif.

By contrast, consider a slightly modified misjoin score. The score is calculated exactly as before, but with one change. Before calculating this score, we will apply a threshold to the Hi-C heatmap, such that, whenever the value of a pixel is larger than *C**, we will change that value to exactly match *C*.* Furthermore, we will exploit our ability to calculate *C(b)* for a proxy genome in order to select *C** to be much less than *C(d)*, such that nearly all pixels in the triangle motif shown will have a value equal to C^* in the saturated matrix – except in the case of a misjoin! When combined with our ability to calculate *C(b)* for a low quality genome, this saturation step makes it simple to calculate an expected value for the misjoin score. Now we can obtain the following for the saturated score S_{sat} *(X)* and the expected value (see Figure 16C):

$$S_{sat}(X) = \sum_{i=X-d}^{X-1} \sum_{j=X+1}^{i+d+1} \min\left(c_{ij}, C^*\right);$$

$$S_{sat}^{ex} = \sum_{i=X-d}^{X-1} \sum_{j=X+1}^{i+d+1} C^* = d * (d+1) * C^*/2.$$

On this basis, we annotate a locus as a putative misjoin whenever the misjoin score for that locus satisfies $S_{sat}(X) < k * S_{sat}^{ex}$,, where *k* is an arbitrary stringency parameter such that 0 ≤ *k* <1. The availability of a reliable expected model greatly improves the sensitivity and specificity of such an approach. The approach is also much less susceptible to errors due to "jackpot" effects, since the impact of a single pixel is greatly dampened by the saturation step.

Note that, so long as *C**<*C(d)*, there is considerable latitude in selecting *C**. In practice, since the function *C(b)* can only be estimated, rather than exactly calculated, it is useful to use *C(d)* as an upper bound for *C**, but to choose values that are smaller, such as *C(2 * d).*

III.b.6.ii.C. Misjoin localization. In practice, we perform misassembly detection using two different values of the matrix resolution *r.* First, we annotate misassemblies at coarse resolution (*r*=25kb), to eliminate noise. In areas flagged by the coarse resolution misassembly detection algorithm, we pinpoint the exact position of the misassembly by repeating the procedure at a higher matrix resolution (*r*=1kb). This approach achieves high positional accuracy in misjoin identification with relatively few false positives.

III.b.6.ii.D. Scaffolding during misjoin detection. Importantly, the misjoin detection algorithm is not performed directly on individual input scaffolds. Both misjoin detection and *C(b)* estimation are more accurate the longer the effective N50 of the input scaffolds. Moreover, misjoin detection is significantly less sensitive if the effective N50 is less than *d*×*r*. For this reason, we maximize the effective N50 of the scaffold set by running the scaffolding algorithm (see below) on the input scaffolds prior to misjoin detection. The input scaffolds are embedded in the resulting output scaffold, and thus misjoins detected in this output scaffold can be associated with misjoins in the input scaffolds.

III.b.6.ii.E. Misjoin classification and correction. After misjoins are identified, we classify them based on whether the misjoin lies inside one of the input scaffolds – implying that there is an error in the input scaffold, which needs to be corrected – or whether the misjoin lies at the junction between two scaffolds, suggesting that the misjoin is a consequence of an error in the input sequence located at a different position. (Notably, our Hi-C based scaffolding step very rarely introduces a misjoin unless there is another, "causative" misjoin in one of the input scaffolds. For instance, a misjoin connecting loci from two different chromosomes can lead to the fusion of two large segments from those chromosomes into a single scaffold; as the scaffolder proceeds, this anomalous scaffold will tend to create many "non-causative" misjoins. Correction of the causative misjoin leads to resolution of the downstream misjoins.)

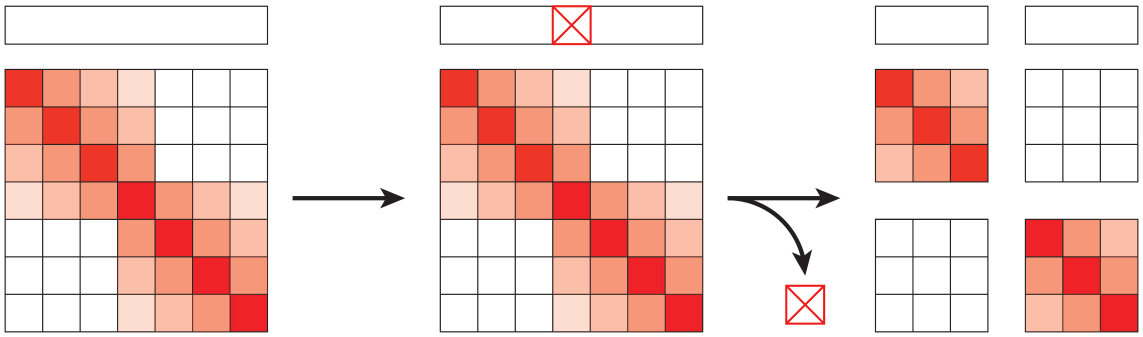


Figure 17. Misassembly correction. Once the misassembly detection algorithm has identified a problematic region that lies inside an input scaffold (a bin marked with an X), the region gets excised resulting in two internally consistent fragments of the original input scaffold. The third fragment that spans a misassembled region is labeled as inconsistent. Inconsistent fragments do not participate in the next round of scaffolding.

If a misjoin lies inside a scaffold, the scaffold is edited by excising sequence intervals flagged by the misjoin detection algorithm (see Figure 17). The excised fragment is labeled as an additional, 'inconsistent' scaffold and excluded from subsequent assembly iterations, since its continued presence during the scaffolding process could lead to further misjoins. If the misjoin is sufficiently far from both ends of the scaffold, this results in splitting the affected scaffold into two scaffolds at the site of the misjoin (in addition to the formation of an inconsistent scaffold). Note that multiple misjoins can be identified in a single scaffold during a single round of misjoin detection, which could lead to repeatedly splitting one scaffold into multiple smaller scaffolds.

The misassembly correction procedure can be described using the pseudocode listed in Table 6. Overall, our misjoin detection algorithm is characterized by low false positive error rates and accurate localization. It is especially sensitive to large misassemblies that give rise to large-scale errors in the genome. Several examples of automatic misassembly detection are given in Fig. S5.

Misassembly correction:
1) Calculate $C(b)$ for the contact matrix at a coarse resolution r_1
2) Compute the saturated score function $S_{sat}(X, r_1)$ at the coarse resolution
3) Calculate $C(b)$ for the contact matrix at fine resolution r_2
4) Compute the saturated score function $S_{sat}(X, r_2)$
5) Flag misjoined loci satisfying $S_{sat}(X, r_1) < k * S_{sat}^{ex}$
6) For each misjoined locus identified:
a. Localize the misjoin at resolution r_2 by finding the minimum of $S_{sat}(X, r_2)$ in the locus
b. Compare localized misjoins with scaffold boundaries to distinguish scaffolds containing misjoins from misjoins that lie between scaffolds
c. Correct the input scaffolds by excising misjoins inside scaffolds and labeling the excised fragment as inconsistent; in addition, if the misjoin is far from the ends of the scaffold, divide the input scaffold into two scaffolds by splitting it at the misjoin site

Table 6. Pseudocode for misassembly correction algorithm.

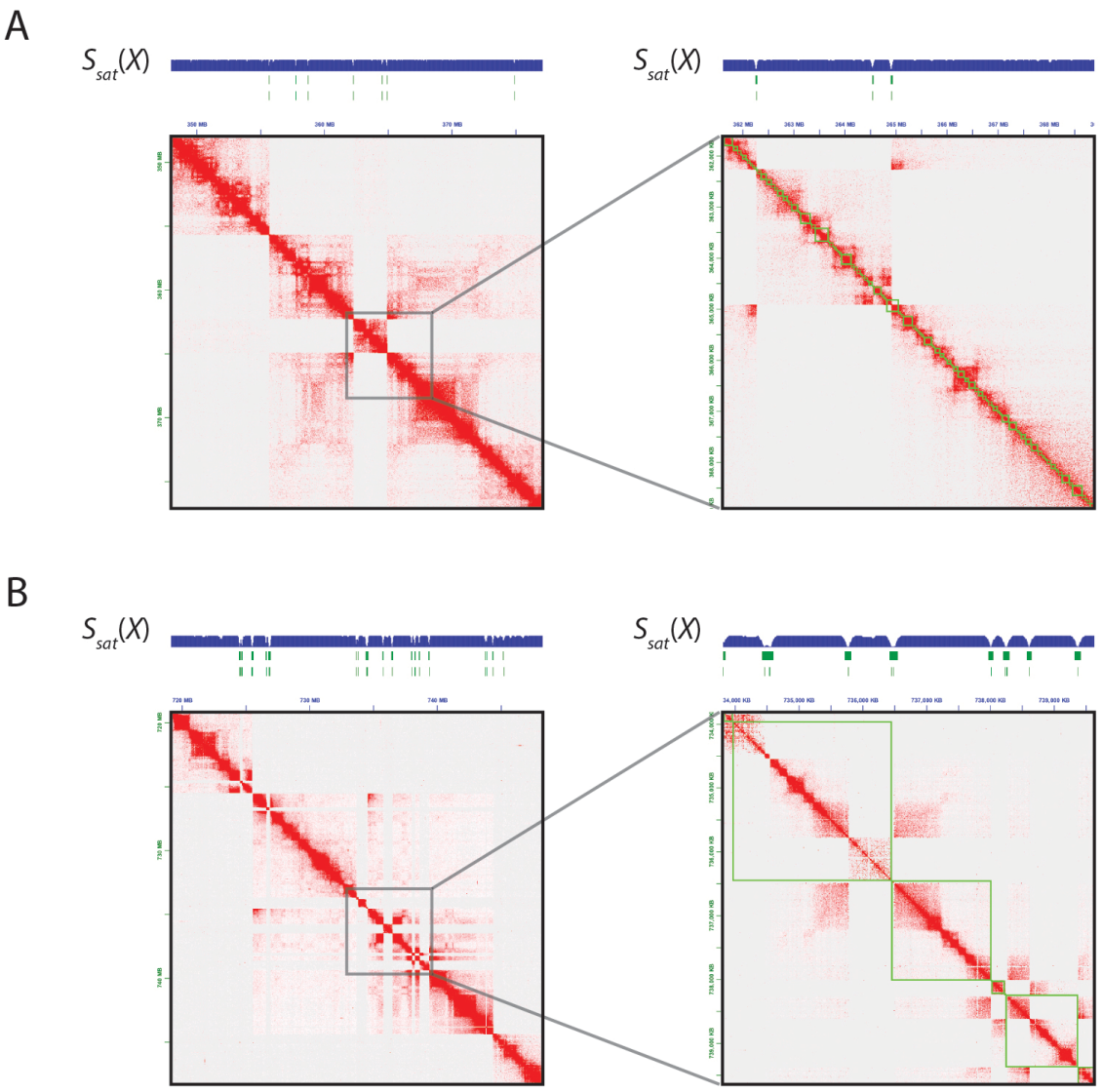


Figure 18. Misassembly detection algorithm performance on Hs1 (A) and AaegL2 (B) input. Left panel shows a fragment of the Hi-C map for the assembly obtained by scaffolding the original input scaffolds, without any editing. The tracks on top of the map show the distribution of $S_{sat}(X,r_1)$ along the assembly (blue) as well as coarse (top green track) and fine (bottom green track) positioning of misassembled sequences as identified by the misassembly detector. Right panel shows a zoom-in on a fragment of the map with input scaffold boundaries superimposed to assist in classifying the detected misassemblies. Intrascaffold misassemblies constitute a list of edits to be applied to the original scaffold set; misassemblies that overlap with scaffold boundaries are ignored. There is one intrascaffold misassembly in Hs1 and 5 intrascaffold misassemblies in AaegL2 in the corresponding fields of view.

III.b.6.iii. Algorithm for scaffolding. To transform a set of input scaffolds into chromosome-length scaffolds, three problems must be solved. "Anchoring" assigns each scaffold to a chromosome, thus partitioning the set of scaffolds into multiple subsets. "Ordering" assigns a relative position to each scaffold on each chromosome with respect to the other scaffolds assigned to the same chromosome. "Orienting" determines which of the two ends of each scaffold is adjacent to the preceding scaffold in the ordering, and which end is adjacent to the next scaffold in the ordering. (This step is equivalent to assigning each scaffold to one of the two complementary strands that comprise a chromosome.) Our algorithm for constructing chromosome-length scaffolds begins with a set of input scaffolds, and simultaneously anchors, orders, and orients them.

The algorithm we employ is iterative; the same steps are performed over and over, often thousands of times. In each step, subsets of the input scaffolds are ordered and oriented with respect to one another to create a new, longer set of scaffolds, which are then used as inputs for the next step. (These might be called "superscaffolds," although we will not use that terminology here. Similarly, when the inputs to our algorithm

are scaffolds, rather than contigs, the algorithm itself might be thought of as a “super-scaffolding” algorithm, rather than a “scaffolding” algorithm; again, we will not make that distinction here.) For the remainder of this section, we will use the term “input scaffolds” to refer to the scaffolds which are the inputs to each step; when needed, we will use the term “initial input scaffolds” to refer to the scaffolds which are the inputs to the iterative algorithm as a whole.

Each iterative step involves constructing and solving a graph optimization problem.

We assume that the input to a given step includes two or more scaffolds. (Otherwise, the scaffolding problem is already solved!)

We begin by splitting each input scaffold into two “hemi-scaffolds” by bisecting the scaffold sequence at the midpoint. The pair of hemi-scaffolds that derive from a single hemi-scaffold are dubbed “sister hemi-scaffolds.”

Next, we construct the “density graph,” S. Each hemi-scaffold is represented as a single vertex in the density graph. Then we append edges to the density graph as follows (see Figure 19):

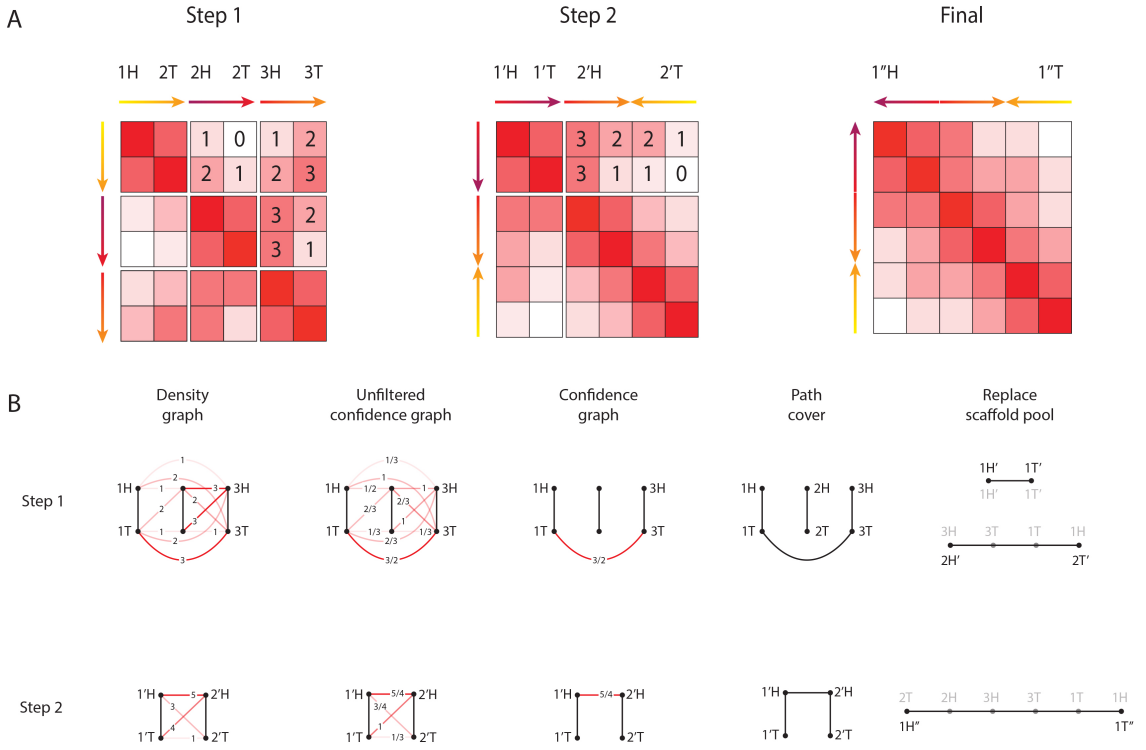


Figure 19. An example of applying an iterative scaffolding algorithm to a mock Hi-C dataset. The input scaffold pool consists of three scaffolds: 1, 2 and 3. The scaffolds are split into hemi-scaffolds. (To be able to distinguish between the hemi-scaffolds we annotate one as H for head and T for tail. The choice in each case is arbitrary.) The number of pairwise Hi-C contacts observed between all loci in all scaffolds is given as a Hi-C contact map. The assembly finishes in two steps. We show the intermediate results for both steps: density graph, unfiltered confidence graph, confidence graph, path cover and redefinition of scaffold pool. Note that to reduce cluttering the weights on the density graph are given without normalization. For the same reason the weights of sister edges are not shown in the density and confidence graphs; instead the sister edges are marked with black color.

First, we append edges between all pairs of vertices that do not correspond to sister hemi-scaffolds. We call these “non-sister” edges. The weight of each non-sister edge corresponds to the density of Hi-C contacts between the corresponding hemi-scaffolds. To calculate this density (i.e., the edge-weight), we count number of Hi-C contacts where one read is incident on one of the hemi-scaffolds, and the other read is incident on the other hemi-scaffold. We then take the resulting value and divide it by the product of the sequence length of the two hemi-scaffolds to arrive at the density. Note that, all else being equal, having an edge of greater weight between two hemi-scaffolds indicates that the two hemi-scaffolds tend to be more proximate in 3D, and thus are more likely to be nearby along the one-dimensional (1D) chromosome sequence as well.

Next, we append edges between all pairs of vertices that correspond to sister hemi-scaffolds. All of these edges are assigned a weight of 2*MAXS, where MAXS is the maximum weight of all of the non-sister edges. This is done in order to encode the fact that sister hemi-scaffolds are adjacent to one another according to the input scaffold set, and that this evidence is – during each scaffolding iteration – regarded as more reliable than any evidence derived from Hi-C. (Of course, in the preceding section we described strategies for correcting scaffolds using Hi-C data. The results of these strategies influence the input scaffold set for any given step of the scaffolding algorithm. However, within the individual iterations, the accuracy of the input scaffold set is regarded as a constraint that takes precedence over the Hi-C data.)

Note that prior approaches for scaffolding using Hi-C rely directly on measures of contact density. This approach can be error-prone. For instance, high-coverage scaffolds, scaffolds containing loci engaged in strong long-range interactions, scaffolds containing repeat sequences, etc. might all display very frequent contacts with scaffolds that are far away from them along the 1D chromosome sequence. Conversely, input scaffolds from low-coverage regions of the genome might exhibit a relatively low contact density, even with scaffolds that are adjacent to them in 1D.

In order to reliably determine the relative positioning and orientation of scaffolds given these potential pitfalls, we have developed a method for identifying adjacent scaffolds that is not directly based on absolute read density. (We call a pair of input scaffolds "adjacent" if they are on the same chromosome, and no other input scaffold has a true genomic position that lies on that same chromosome, in between them.)

To accomplish this, we first use our density graph to define an "unfiltered confidence graph," *C'*. The vertices of the unfiltered confidence graph again correspond to the hemi-scaffolds. The edges of the unfiltered confidence graph are defined as follows (see Figure 19):

- If A and B are not sister homologs, then we append an edge between them whose weight is the ratio of the weight of the edge connecting them in the density graph (s_{AB}), and the weight of the second-largest non-sister edge incident on either A or B in the density graph. Note that $c_{AB}>1$ if and only if there is no hemi-scaffold whose contact density with either A or B exceeds the contact density between A and B. Informally, this means that, based on the Hi-C data, A is the best partner for B, and B is also the best partner for A. Thus we can be confident that A and B are adjacent. We therefore call any edge in the unfiltered confidence graph whose weight is greater than 1 "reliable." Edges whose weight is 1 or smaller are called "unreliable."

- Next, we append edges between all pairs of vertices that correspond to sister hemi-scaffolds. All of these edges are assigned a weight of 2*MAXC, where MAXC is the maximum weight of all of the non-sister edges in the unfiltered confidence graph. This is done in order to encode the fact that sister hemi-scaffolds are adjacent to one another according to the input scaffold set, and that this evidence is – during each scaffolding iteration – regarded as more reliable than any evidence derived from Hi-C.

If all of the non-sister edges are unreliable, then the iteration has failed, in the sense that no reliable adjacency information could be extracted from the Hi-C data. Therefore, if all the non-sister edges are unreliable, we remove the smallest input scaffold from the input scaffold set and repeat the step, constructing a new density graph, etc. Note that removing the smallest input scaffold and repeating the step might still not yield a reliable edge, in which case another scaffold is removed, and so on. Eventually, either a reliable edge will be found or there will only be one scaffold left (at which point the algorithm halts and outputs the remaining scaffold.)

Assuming there is a reliable non-sister edge in the unfiltered confidence graph, we next filter the unfiltered confidence graph by removing all edges whose weight is less than or equal to 1. The resulting graph is called the confidence graph. Note that every vertex is adjacent to 1 sister edge in the confidence graph, and to at most 1 non-sister edge. Thus, all vertices in the confidence graph have either degree 1 or degree 2. Hence (by an elementary fact of graph theory) the confidence graph is a collection of disjoint paths and cycles.

(Note that it is possible to use our methods to reconstruct circular plasmids and chromosomes, in which case the following steps must be modified; we will not do so here. Instead, for the sake of simplicity, we describe methods that apply to organisms with linear chromosomes.)

Vertices that are adjacent in the confidence graph are very likely to correspond to hemi-scaffolds that are adjacent in 1D. Therefore, each path in the confidence graph corresponds to a high-confidence scaffold. Furthermore, each path in the confidence graph whose length is greater than 2 corresponds to a new scaffold comprised of multiple input scaffolds whose relative order and orientation has been determined using Hi-C.

Cycles in the confidence graph are harder to interpret, since they contain multiple possible paths (i.e., new scaffolds) spanning all the vertices (hemi-scaffolds) in the cycle. Here the key is to identify the maximal path contained in the cycle, which corresponds to the new scaffold in which we have the highest confidence given the Hi-C data. This can be easily accomplished by removing the edge in each cycle whose weight is smallest. (Because of how the confidence graph is constructed, this edge will always be a non-sister edge.) In graph theoretic terms, this procedure can be thought of as a way to construct the maximal (in terms of total edge weight) vertex-disjoint path cover of the confidence graph.

A complementary way of formulating this procedure (which is mathematically guaranteed to produce exactly the same output) is to greedily select the highest weight edges from the confidence graph, ensuring that no vertex in the graph will be incident on two or more edges. (This constraint is equivalent to the rule that, after selecting non-sister edge AB, we must remove all other non-sister edges incident on either A or B.) Following this procedure, a maximum weight vertex-disjoint path cover is eventually obtained. Notably, for the special case of confidence graphs, this complementary formulation is exactly equivalent to Kruskal's algorithm for constructing a maximal spanning forest [38]. (Because a confidence graph consists of disjoint paths and cycles, its maximal spanning forest is always a vertex-disjoint path cover.) In fact, it is this complementary procedure that is implemented in our code.

Since vertices that are adjacent in the confidence graph are very likely to correspond to hemi-scaffolds that are adjacent in 1D, and since each path in the confidence graph corresponds to a high-confidence scaffold, the maximum weight vertex-disjoint path cover in the confidence graph corresponds to a new set of scaffolds that is optimal with respect to the input scaffolds and the Hi-C data. Each of these new "output" scaffolds comprises one or more input scaffolds; within each output scaffold, the relative order and orientation of the input scaffolds has been determined using Hi-C.

Once the output scaffolds are obtained, the iteration ends. The next iteration can now begin; the output scaffolds from the previous iteration can be used as input scaffolds for the new iteration, and the density and confidence graphs are constructed for the new inputs. Note that reconstructing the graph from scratch allows more Hi-C data spanning larger scales to be incorporated into the analysis. Of course, reconstructing the density and confidence graphs is computationally expensive, and is therefore only done once no more information can be extracted from the reliable edges in the previous iteration; i.e., after the vertex-disjoint path cover of the confidence graph has been calculated, new scaffolds have been obtained, and the previous iteration is complete.

(Note that a more computationally efficient alternative to recalculating the density graph is to use a more permissive threshold for including edges in the confidence graph; i.e. require $c_{AB}>k$ where $k<1$. This would allow more scaffolding information to be extracted at each step, and thus fewer steps would be required to complete the scaffolding procedure. However, this strategy could also increase the frequency of errors, and it is not the strategy we employ here.)

Scaffolding:

Initialize the scaffold pool using a set of input scaffolds
While there is more than one scaffold in the scaffold pool:

a. Construct the density graph for the scaffolds in the scaffold pool
b. Transform the density graph into a confidence graph
c. If the confidence graph does not contain edges linking hemi-scaffolds from distinct scaffolds in the pool ("non-sister edges"):
    i. Remove the smallest scaffold from the scaffold pool
d. Else:
    i. Find maximum weight vertex-disjoint path cover of the confidence graph
    ii. Determine the corresponding output scaffolds
    iii. Replace the scaffold pool with the output scaffolds

Table 7. Pseudocode for iterative scaffolding algorithm.

If there is only one output scaffold, the process ends and the single output scaffold is reported. To summarize we give pseudocode in Table 7.

V.b.6.iv. Polishing. Polishing is an optional step in our pipeline that has been designed to address challenges associated with unusual 3D features that arise for organisms exhibiting strong telomere and centromere clustering. This can create false positives during scaffolding, since extremely strong off-diagonal 3D signals associated with telomere and centromere clustering can sometimes be strong enough to rival the contact frequencies observed for loci that are adjacent in 1D.

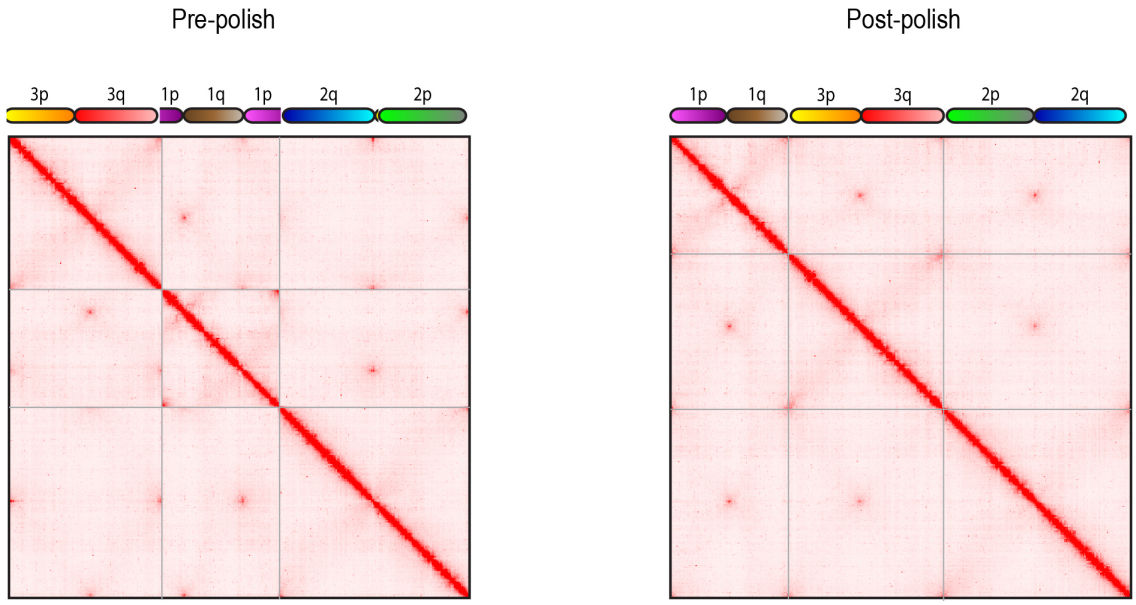


Figure 20. Polishing the assembly during the construction of AaegL4 genome. Clustering of telomeres and centromeres can create false positives during scaffolding, since extremely strong off-diagonal 3D signals associated with telomere and centromere clustering can sometimes be strong enough to rival the contact frequencies observed for loci that are adjacent in 1D. Such errors are corrected by low-resolution misassembly detection and reassembly of the resulting multimegabase fragments.

Figure 20 shows the Hi-C contact map built with respect to the Ae. aegypti genome assembly before and after the polishing step. The map suggests that chromosome 3 is very accurately assembled, but chromosomes 1 and 2 contain a type of error that is characteristic of assembly in genomes that exhibit strong telomere-to-telomere clustering. In this error, the enhanced proximity between the two telomeres is mistaken for 1D proximity. As a result, the raw chromosomal scaffolds corresponding to chromosomes 1 and 2 exhibit a cyclic permutation with respect to the true chromosome.

As an example, if the sequence of the true chromosome was ABCDEFG, where locus A and G are telomeres, then the erroneous sequence might be DEFGABC. We call this sort of error a "cycle break." Note that, when a cycle break occurs, an off-diagonal peak linking the two putative ends of the chromosome (in the example, D and C) is still seen in the Hi-C map. However, this signal is actually due to the true 1D proximity between the two ends of the putative chromosome, rather than at true 3D signal. Conversely, the on-diagonal signal between A and G, which appears to reflect 1D proximity, is in fact due to the telomere clustering. (Similar errors may arise due to strong interaction between telomeres of two different chromosomes. They are addressed in the same way. See Figure 20, chromosomes 2 and 3.)

Such errors can be corrected by a single additional round of misjoin correction, performed at extremely low resolution (r~1Mb). The low-resolution misassembly detection identifies reliable "superscaffolds", each of which is many megabases in length. These superscaffolds are then ordered and oriented using a version of the scaffolder that exploits the large size of the superscaffolds to more reliably distinguish 1D and 3D signal by utilizing Hi-C contacts incident only on the superscaffold ends, rather than on the whole superscaffold.

III.b.6.v. Algorithm for extracting raw chromosomal scaffolds from the megascaffold. The scaffolding algorithm produces a single megascaffold that concatenates all the chromosomes. For genomes that do not exhibit pronounced telomere clustering in the Hi-C map (such as human), we split the megascaffold into chromosomes by running a variant of the misassembly detector to identify the chromosome boundaries. This algorithm relies on the fact that the contact frequency between scaffolds that are adjacent on the megascaffold but which lie on different chromosomes is relatively low, since they are not actually in 1D proximity. Thus the boundaries between chromosomes generate a signal that is similar to a typical misjoin. Moreover, this effect is enhanced by the tendency of loci on the same chromosome to exhibit elevated contact frequency.

If the spatial clustering of telomeres is evident in the Hi-C map, the phenomenon can be exploited in the effort to partition the genome into chromosomes. In particular, the first scaffold in the megascaffold must come from the end of a chromosome, and therefore derives from a telomere. Identifying positions in the Hi-C matrix that have an enriched number of contacts with the megascaffold edge thus enables the detection of chromosome boundaries.

III.b.6.vi. Algorithm for detection and correction of false positives that occurred during misjoin detection ("Sealing"). During the sealing step, sequences that were erroneously excised during misjoin correction may be re-introduced. In particular, if the two parts of a corrected scaffold remain adjacent to one another in the raw chromosomal scaffold, it suggests that the original scaffold was correct, since the independent contact patterns from both parts are consistent with the original scaffold. In this case, the misjoin that led to the correction is judged to be a false positive and the intervening sequence is restored.

III.b.6.vii. Algorithm for merging assembly errors due to undercollapsed heterozygosity. A frequent error modality found in draft haploid genome assemblies is undercollapsed heterozygosity. This is when there exists a subset of the scaffolds such that each scaffold accurately corresponds to a single locus in the genome, but these loci overlap one another. Consequently, there are individual loci in the genome that are covered multiple times by different scaffolds. This error is typically caused by the presence of multiple haplotypes in the input sample material, which are sufficiently different from one another that the contig and scaffold generation algorithms do not recognize them as emerging from a single locus. This class of error is frequent in AaegL2; the step can be omitted when assembling genomes of organisms with low heterozygosity such as Hs1.

Undercollapsed heterozygosity error leads to highly fragmented draft assemblies with a larger-than-expected total size [39],[40]. This, in turn, causes numerous problems in downstream analyses such as erroneous gene copy number estimates, fragmented gene models etc. The challenge remains significant even when special effort is taken to reduce the levels of heterozygosity in genomic data by inbreeding as has been done with the draft AaegL2 assembly [41]. It is therefore important to ensure that the final genome reported by our assembler minimizes the number of assembly errors due to undercollapsed heterozygosity.

To specifically address this class of misassembly error, we have developed an algorithm whose goal is to merge these overlapping scaffolds into a single scaffold accurately incorporating the sequence from the individual scaffolds. The result of this is a merged haploid reference scaffold.

One assumption of the overlap merging algorithm is that, when multiple scaffolds correspond to multiple haplotypes, these scaffolds will exhibit extremely similar contact patterns, genome-wide. Because they exhibit similar long-range contact patterns, the scaffolding algorithm tends to assign such scaffolds to nearby positions in the genome. Thus, the merge algorithm seeks to identify pairs of resolved scaffolds that (i) lie near one another in the raw chromosomal scaffolds, and (ii) exhibit long stretches of extremely high sequence identity.

Briefly, we search for undercollapsed loci by running a sliding window of fixed width along the raw chromosomal scaffolds. We then use LASTZ to do pairwise alignment of all pairs of resolved scaffolds that fall in the sliding window [37]. The total score of all collinear alignment blocks (stanzas), normalized by the length of the overlap, is used as a primary filtering criterion to distinguish between alternative haplotypes and false positive sequence similarity. The location of the overlap relative to input scaffold boundaries is also taken into account in determining whether the scaffolds can be correctly merged (see Figure 21).

Figure 21. The location of the overlap relative to input scaffold boundaries is taken into account in determining whether the scaffolds can be correctly merged

We next construct a graph whose nodes are resolved scaffolds, and where edges reflect significant sequence overlap between resolved scaffolds that are proximate on the raw chromosomal scaffold. The resulting graph contains a series of connected components. Cycles in the graph are analyzed in order to filter out components with overlaps on conflicting strands.

Finally, we construct a tiling path through the scaffolds of each individual connected component, recursively aligning scaffolds to an already collapsed portion of the group, finding the highest scoring alignment block and switching from one haploid sequence to the other at the endpoints of the alignment.

Ideally the resolved scaffolds in each connected component are consecutive on the raw chromosomal scaffold, and with relative orientations that match those suggested by the pairwise alignments. In practice, however, this is not always the case. This can be due to differences in haplotype representation between the genomic data used to produce the draft assembly and that of the Hi-C experiment. For example, sequences belonging to different clusters may be intertwined. Similarly, the orientation of contigs/scaffolds within the cluster as suggested by pairwise alignment may not match those suggested by the scaffolding step. In such cases the relative position and orientation of the connected components with respect to the rest of the assembly is decided by majority vote with each input scaffold's contribution weighed by its length. Alternatively, assembly can be rerun using the merged components as input.

Note that although it is possible to add additional constraints when appropriate, such as the exact number of haplotypes present in the data, we do not rely on such knowledge in general. This allows us to work with polymorphic assemblies, such as when multiple individuals were used to produce the draft assembly. In particular it allows us to handle cases where the degree of polymorphism is unknown.

III.c Visualization and Data Integration

III.c.1 Juicebox for Data Visualization and Integration with ENCODE tracks

V.c.1.i. Juicebox: Juicebox is a tool for exploring Hi-C and other contact data. Juicebox allows users to zoom in and out of Hi-C maps interactively, just as a user of Google Earth might zoom in and out of a geographic map. Maps can be compared to one another, to ENCODE 1D tracks and other seq data, or 2D feature sets.

Typically, the pairwise interactions produced by Hi-C experiments are visualized as a heat map: the linear genome is partitioned into loci of a fixed size, or “resolution” (e.g., 1Mb or 1Kb) and each entry in the two-dimensional heat map corresponds to the number of contacts observed between a pair of loci during the experiment.

Developing adequate visualizations for Hi-C heat maps is a challenge, because the size of the meaningful biological features they contain ranges over at least seven orders of magnitude: from loops anchored at 20bp CTCF sites, to territories that extend across 200Mb chromosomes. At the highest meaningful resolutions, published Hi-C heat maps contain trillions of entries, and only a tiny portion can be displayed at any given time. At coarser resolutions, more of the map can be shown, but the fine structure can no longer be resolved.

Juicebox allows users to explore Hi-C heat maps interactively, zooming in and out just as a user of *Google Earth* might zoom in and out of a geographic map; it integrates many technologies developed for the Integrative Genomics Viewer [42] with a broad ensemble of methods specifically designed for handling 2D contact data (Figure 22). Individual maps can be normalized (corrected for experimental bias), compared to 1D tracks (such as ENCODE ChIP-Seq data), and compared to 2D feature lists (such as loop and domain annotations). Multiple maps can be browsed side-by-side simultaneously, and compared with one another in various ways, revealing both conservation and variation across cell types and species. Users can create their own heat maps to explore their own experiments.

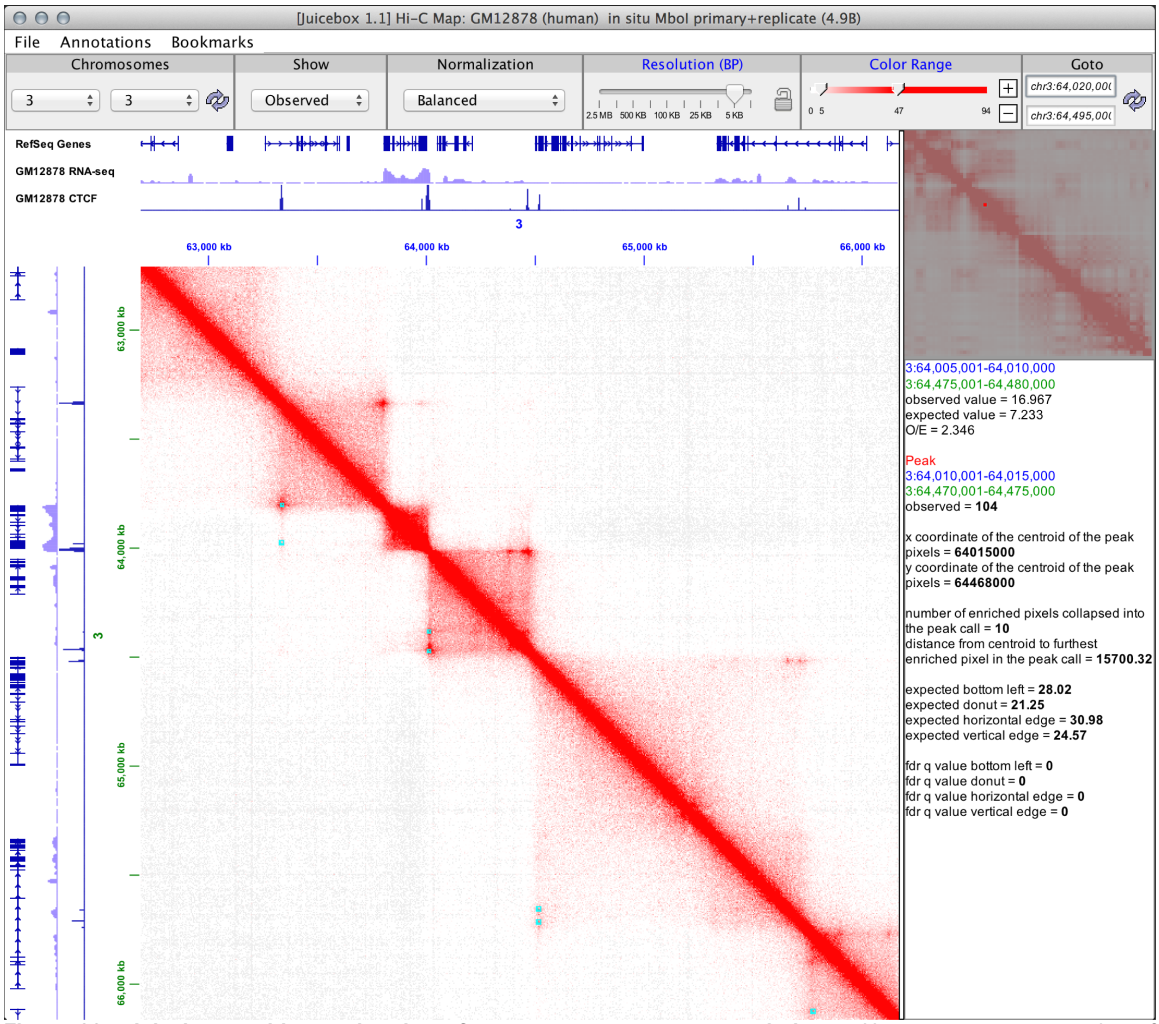


Figure 22. Juicebox enables exploration of contact maps at many resolutions. Here, we see a screenshot of Juicebox zoomed into 5 Kb resolution on chromosome 3. The toolbar at the top allows users to quickly navigate between different chromosomes, views, normalizations, and resolutions. Users can load one-dimensional tracks and compare them to features seen in the heat map. Two-dimensional features can be superimposed on the main map. Here, several peaks are annotated (cyan), each peak indicating the presence of a chromatin loop. A peak in the CTCF track, indicating

CTCF binding, is seen at most loop anchors. At the top right, a mini-map shows the whole chromosome at low resolution. Below, hover text shows additional data for one of the highlighted peaks.

In Juicebox—when using Hi-C maps of adequate depth and quality—increasingly small features can be resolved as the user zooms in. In a genome-wide view, chromosome territories are evident, as are chromosomal rearrangements such as translocations. Clicking on a particular chromosome zooms into its intrachromosomal map, optimized for the user's monitor. The broad compartmentalization of the genome, which manifests as alternating long-range patterns, is typically visible at this resolution. On the X chromosome, two superdomains may also become apparent, partitioning the chromosome. These are accompanied by superloops, bright peaks many megabases away from the diagonal.

Zooming in further can be accomplished by double-clicking, by using the resolution slider, or by drawing a box around a region of interest. At 50Kb resolution, subcompartments can be seen, reflecting finer differences in the long-range contact pattern. At 25Kb resolution, contact domains appear: intervals containing loci, which preferentially form contacts with one another, and that form squares along the diagonal. Juicebox makes it easy to compare these structures to a large number of broad-source chromatin marks at once. We find that the epigenetic marks that decorate particular contact domains correlate strongly with differences in long-range contact pattern.

At 5Kb resolution, chromatin loops are readily seen: bright peaks in which contact frequency is enhanced relative to the local neighborhood. These tend to lie at the corners of contact domains. Finally, at 1Kb, the relationship between the loops and point-source epigenetic tracks can be interrogated. For instance, it becomes clear the chromatin loops are frequently anchored at convergent CTCF motifs, i.e., CTCF motifs that "point toward" one another (Rao and Huntley et al., 2014). Juicebox can be used to zoom in much further, although maps with enough data to support such studies do not yet exist.

Juicebox is available as a Java application that can be downloaded and launched via aidenlab.org/juicebox. The code is open source and licensed under the MIT license, available at github.com/theaidenlab/Juicebox. Users can explore their own data, or examine data from over 38 Hi-C, 5C, and CHIA-PET publications.

III.c.1.ii. Juicebox Web: We have recently implemented a pure Javascript version of Juicebox that runs inside a web client. It can be explored by going to aidenlab.org/juicebox. As with our other software, the code is online and open source, available at https://github.com/igvteam/juicebox.js. Juicebox Web can load all of the same maps as the desktop version of Juicebox. Though not yet as fully-featured, one big advantage of Juicebox Web is the ability to share exactly what you are looking at with collaborators, simply by sending a link. See Figure 23 for a screenshot.

Figure 23. Juicebox Web. A web version of Juicebox written purely in Javascript.

III.c.1.iii. Juicebox VR: In addition to Juicebox, we have explored the usage of virtual reality devices to visualize Hi-C data in novel ways that may allow for new insight into the datasets. We have publicly released Juicebox VR as an Android application on the Google Play Store to highlight a demo of what such a visualization can look like (play.google.com/store/apps/details?id=com.visor.JuiceboxVR). The public application runs on any Android-based VR headset and provides the user with a 1st-person perspective while viewing the 3D Hi-C landscape (*Figure 24*). Juicebox VR creates a virtual representation of data generated by Hi-C. Every 2D position in the landscape corresponds to a pair of loci, and the height at the 2D position illustrates with the contact frequency.

In the public demo, the viewer can traverse the map by looking in any direction within the 3 rotational axes. Looking straight up will allow the user to jump, which visually correlates to seeing the map at lower resolutions (Figure 25). An internal version of the app built for Oculus Gear VR allows for additional controls with more precise movement within the map, utilizing the Oculus touch pad.

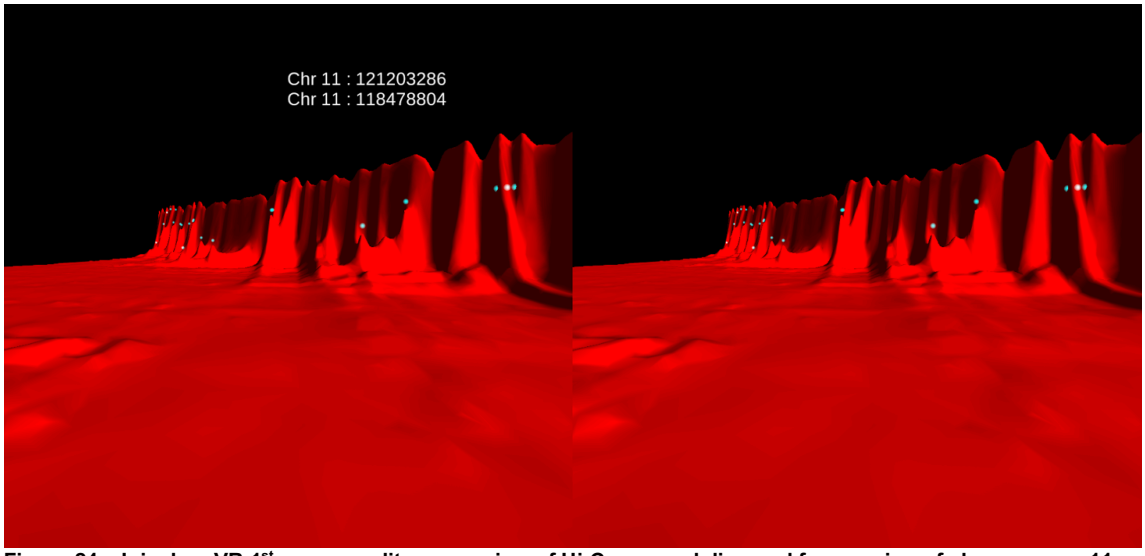


Figure 24. Juicebox VR 1st-person split-screen view of Hi-C map and diagonal from region of chromosome 11.

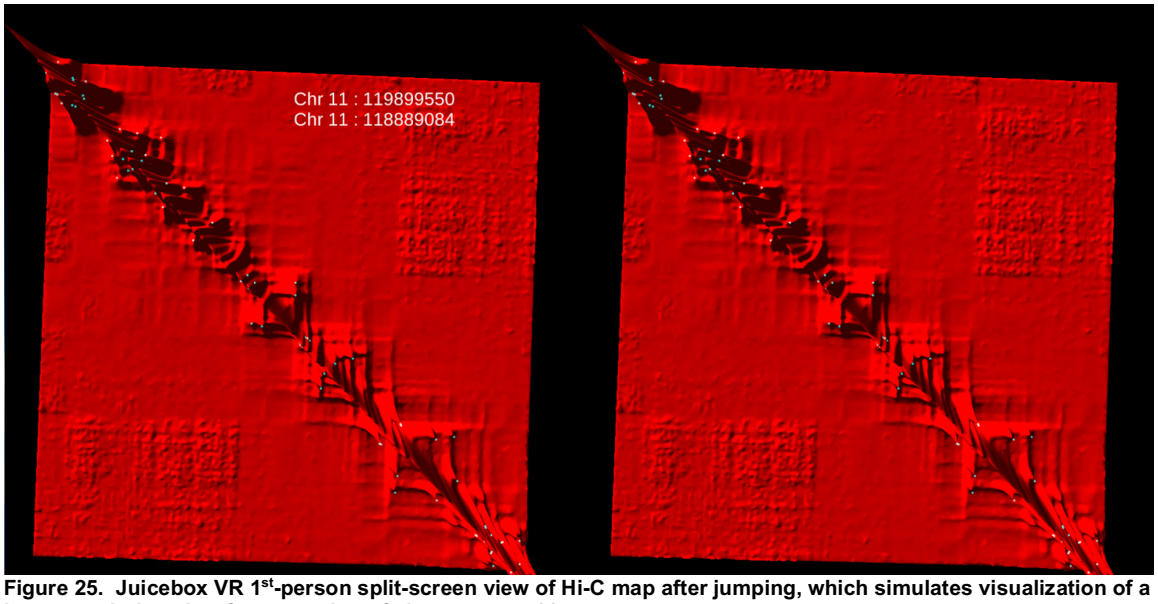


Figure 25. Juicebox VR 1st-person split-screen view of Hi-C map after jumping, which simulates visualization of a lower resolution view from a region of chromosome 11.

The current process is as follows.

1. A region of the Hi-C map is exported from Juicer Tools using the dump command with balanced Knight-Ruiz (KR) normalization.
2. The data is log-scaled and converted to a dense matrix format.
3. The data is imported as an elevation/height map into the Blender 3D graphics and animation software.
4. Blender is used to smooth out the 3D surface and reduce the polygon count for faster rendering.
5. The data is exported from blender as a *.obj* file. Other formats can also be used. Due to symmetry of the Hi-C data across the diagonal, the *.obj* can be made to include just the upper right triangle of the Hi-C matrix.
6. The *.obj* file is imported into the Unity Game Engine and reflected across the diagonal to represent a full matrix. Gradients are calculated to allow physics-based interactions with the landscape.
7. The existing codebase for handling movement is based on 1st-person game controls. Depending on the version of Juicebox VR, the Durovis Dive plugin or Oculus Gear VR plugin is used to assist with head tracking and handling the split-screen.

8. Loop annotations from HiCCUPS are added with manual editing of the height and location of the loop positions.
9. The application is built and signed as an APK for the appropriate Android OS.

Currently, the visualization shows pre-rendered regions of the Hi-C map, but the lab is actively working on utilizing the Straw API for live-streaming of data into a VR landscape.

III.c.2. Straw Data API

The Straw data API enables fast extraction of contact matrices and normalization data from .hic files. Written in multiple languages, including Python, R, and C++, Straw is designed to help bioinformaticians easily utilize the highly compressed .hic format and the large public archive of .hic files available. With Straw, users can extract the slice of data they are most interested in and process it using their own scripts in their preferred language. The Hi-C data archive contains over 440 experiments from over 38 different published Hi-C papers. Straw can access these files by URL, so users avoid costly downloads. Straw is open source; code and documentation are available at github.com/theaidenlab/straw

III.c.3. Data Formats

Juicebox and the Juicer data analysis tools rely heavily on the .hic file, a binary, highly compressed format that stores matrices in sparse block format to allow very quick query time. An arbitrary number of matrices can be stored without affecting access time. There is also a large public online archive of ChIA-PET and Hi-C experiments described via .hic files that anyone can access using Juicebox, Juicer, or Straw.

III.c.3.i. .hic file format: The hic file format was created by Jim Robinson for Juicebox ([16]) and is specially designed to provide fast random access to any contact matrix at any resolution. The footer of a hic file contains pointers to all the matrices, together with the size (in bytes) of the matrix. To quickly access a matrix at a particular resolution, the reader looks up the pointer in the footer and reads all the data from that matrix into main memory. Data is also stored in "blocks" of adjacent contacts, enabling fast visualization in two dimensions. This design makes it possible to visualize billions of Hi-C contacts quickly and to zoom in to extremely high resolution in real time in Juicebox. All of the subsequent tools in the Juicer pipeline run on hic files.

The header portion starts with a field identifying the file as *hic* format and establishing the version number (currently 8). The master index position (the location in bytes in the file where the reader can look up the position of all the matrices) follows. The subsequent fields of genomeID, chromosome dictionary, base-pair resolution dictionary, and fragment resolution dictionary are important functional metadata used to properly read and render the matrices. The attribute dictionary is meant for pure metadata that describes the experiment and is completely customizable.

The body of the *hic* file contains the data for every chromosome-chromosome combination; if there are N chromosomes, there will be N^2 matrices represented. Each matrix starts with its unique chromosome-chromosome combination, followed by the number of resolutions stored. Then for each resolution, a header portion with the metadata necessary to read the matrix is followed by the matrix data, stored in block format to ensure good locality. That is, each matrix is subdivided into sub-matrices, called blocks. The "block column count" is the number of columns for the block grid. The "block bin count" is the size of each block in bins, or pixels. Since the blocks are square, their size is blockBinCount x blockBinCount. Finally, each block contains the cell data: the actual count data for that block at that resolution, stored in sparse upper triangular format.

The footer of the *hic* file format contains pointers to each matrix, ensuring fast lookup and direct access to the data without reading through the entire file. It also contains the expected value vectors, which enable calculation of observed/expected for all the matrices. Following the expected value vectors are the normalized expected value vectors, used for the observed/expected for normalized matrices, and the normalization vectors. There are two main types of normalization: VC, or vanilla coverage, and KR, or Knight-Ruiz balanced. However, the format is flexible, and additional types of normalization can easily be added. The normalization vectors are used with the count to calculate and display normalized matrices.

The code for reading and writing the *hic* file format is available at the Straw and Juicebox GitHub repositories. In Juicebox, consult the class DatasetReaderV2 in package juicebox.data for reading and the class Preprocessor in package juicebox.tools.utils.original for writing. In Straw, the main method straw is used to read the file.

III.c.3.ii. Hi-C contacts text format: To create a .hic file, Juicer Tools takes as input a text file that contains, on each line, one Hi-C read per line. There are several different formats possible:

Medium format (most common)
A whitespace separated file that contains, on each line
<readname> <str1> <chr1> <pos1> <frag1> <str2> <chr2> <pos2> <frag2> <mapq1> <mapq2>

- str = strand (0 for forward, anything else for reverse)
- chr = chromosome (must be a chromosome in the genome)
- pos = position
- frag = restriction site fragment
- mapq = mapping quality score

If not using the restriction site file option, frag will be ignored, but please see above note on dummy values. If not using mapping quality filter, mapq will be ignored. readname and strand are also not currently stored within .hic files.

Short format
A whitespace separated file that contains, on each line
<str1> <chr1> <pos1> <frag1> <str2> <chr2> <pos2> <frag2>

- str = strand (0 for forward, anything else for reverse)
- chr = chromosome (must be a chromosome in the genome)
- pos = position
- frag = restriction site fragment

If not using the restriction site file option, frag will be ignored, but please see above note on dummy values. readname and strand are also not currently stored within *.hic* files.

Short with score format
This format is useful for reading in already processed files, e.g. those that have been already binned and/or normalized; this format can be easily used in conjunction with the -r flag to create a *.hic* file that contains a single resolution.
A whitespace separated file that contains, on each line
<str1> <chr1> <pos1> <frag1> <str2> <chr2> <pos2> <frag2> <score>

- str = strand (0 for forward, anything else for reverse)
- chr = chromosome (must be a chromosome in the genome)
- pos = position
- frag = restriction site fragment
- score = the score imputed to this read

If not using the restriction site file option, frag will be ignored, but please see above note on dummy values. readname and strand are also not currently stored within *.hic* files.

Long format
The long format is used by Juicer and takes in directly the *merged_nodups.txt* file.
A whitespace separated file that contains, on each line
<str1> <chr1> <pos1> <frag1> <str2> <chr2> <pos2> <frag2> <mapq1> <cigar1> <sequence1> <mapq2> <cigar2> <sequence2> <readname1> <readname2>

- str = strand (0 for forward, anything else for reverse)
- chr = chromosome (must be a chromosome in the genome)
- pos = position
- frag = restriction site fragment
- mapq = mapping quality score
- cigar = cigar string as reported by aligner
- sequence = DNA sequence

If not using the restriction site file option, frag will be ignored, but please see above note on dummy values. If not using mapping quality filter, mapq will be ignored. readname, strand, cigar, and sequence are also not currently stored within *.hic* files.

V.c.3.iii. Sparse text output: The output of Juicer Tools data extraction and of Straw is a matrix in sparse, upper-triangular format; each line contains the x-coordinate of the bin, the y-coordinate of the bin, and the count (possibly normalized) within the bin. Since the data are the same across the diagonal, the x-

coordinate reported is always less than or equal to the y-coodinate. Typical output of Juicer Tools data extraction at 5Kb would look like:

x_value	*y_value*	*score*
5000	5000	1292
5000	10000	327
5000	15000	205

III.c.3.iv. Feature annotation formats: Features such as loops and domains can be loaded into Juicebox to be viewed on top of heat maps. The feature annotation format is agnostic to the method used to create the loops; in particular, ChIA-PET loops can be represented just as easily as HiCCUPS loops. The 2D annotation format contains a header line describing the columns; the first seven columns are mandatory, and any subsequent columns are optional. The file is tab delimited. For example:

chr1	x1	x2	chr2	y1	y2	color	comment
chrX	85000000	89000000	chrX	85000000	89000000	0,255,0	My green region
chrX	90000000	99100000	chrX	90000000	99100000	0,0,255	My blue region

III.c.4. Community Uptake of Juicebox and Juicer

Juicebox has been publicly available since December 2014. From December 2014 to February 2017, Juicebox has been downloaded over 10,000 times from over 1,300 distinct locations (we have geolocated the IP addresses and do not double count IP addresses that map to the same geolocation). Those locations range all over the world, from the United States and South America, across Europe and Africa, to Russia, China, Korea, Japan, and Australia, and span all six habited continents (Figure 26). Over 6Tb of data has been streamed since release and over 38 million maps have been rendered or analyzed.

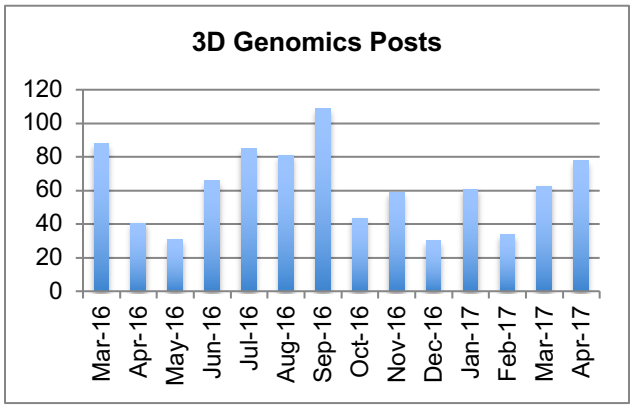


Figure 26. Juicebox use around the world as of February 2017. Map of IP addresses accessing the main Juicebox menu from Juicebox's release in December 2014 to February 2017. Juicebox has been used as far south as New Zealand and as far north as Iceland; it is used extensively across the United States and Europe, but also in Asia, Africa, and South America.

The three cluster implementations of Juicer have been available since January 2016 on GitHub for beta testing, and a 3D Genomics forum has been available since spring 2016. There are over 180 different users generating over 740 different posts with over 9000 views, indicating a high level of interest and user adoption of our software. Some screenshots of the forum are shown in Figure 27, Figure 28, and Figure 29.

There are posts from all over the world, including many different laboratories that are not a part of the ENCODE/4DN community. Often, users will announce the availability of new Hi-C data that they've published and want included in Juicebox on the forum (Figure 27). We have found the forum extremely helpful in identifying features that are important to the community. Thanks to user interest, we have implemented new code bases, such as the single machine version of Juicer and the Straw data API (Figure 28). Finally, we are grateful to help others and happy to know that our software is appreciated by the community (Figure 29).

Juicebox and .hic files are also used by other visualization groups in the broader genomics community. The Washington University EpiGenome browser includes Juicebox .hic files in its publicly available track hub. Users can go to their Public Tracks area and add any of the experiments to view in the Epigenome Browser (Figure 30). The HiCPro pipeline also includes .hic files as output [43].

Many others labs have used Juicer to process or analyze their data, including multiple different labs publishing loop-resolution maps [44-46]; all told, we estimate hundreds of terabytes of data have been processed around the world with Juicer.

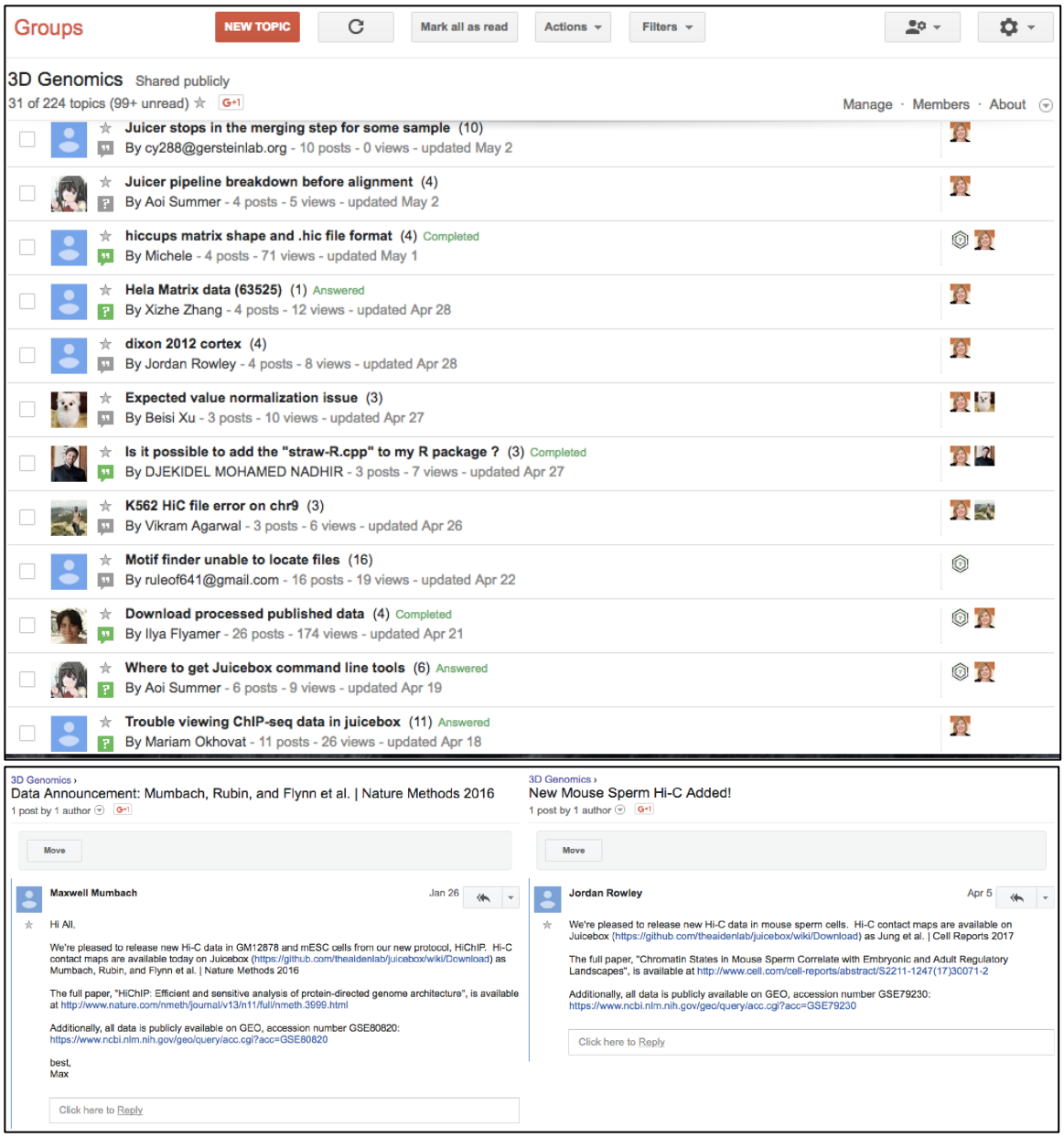
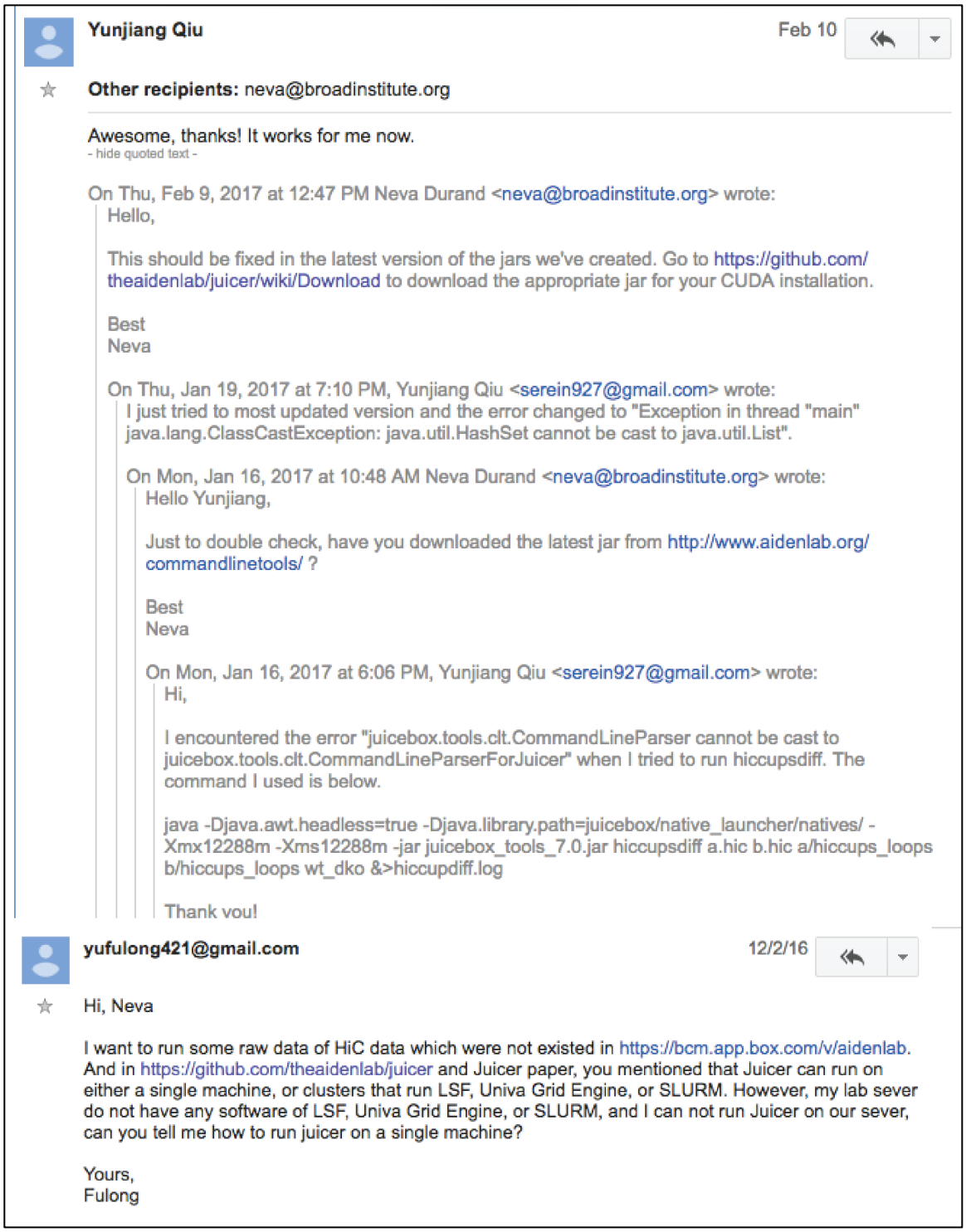


Figure 27. 3D Genomics forum. A diverse array of users post to the forum, asking questions about Juicer, Juicebox, Straw, assembly, and the Hi-C protocol. Users also make data announcements forum, broadcasting the publication of new data for the community to view and download.



Yunjiang Qiu Feb 10

Other recipients: neva@broadinstitute.org

Awesome, thanks! It works for me now.
- hide quoted text -

On Thu, Feb 9, 2017 at 12:47 PM Neva Durand <neva@broadinstitute.org> wrote:
> Hello,
>
> This should be fixed in the latest version of the jars we've created. Go to https://github.com/theaidenlab/juicer/wiki/Download to download the appropriate jar for your CUDA installation.
>
> Best
> Neva
>
> On Thu, Jan 19, 2017 at 7:10 PM, Yunjiang Qiu <serein927@gmail.com> wrote:
> > I just tried to most updated version and the error changed to "Exception in thread "main" java.lang.ClassCastException: java.util.HashSet cannot be cast to java.util.List".
> >
> > On Mon, Jan 16, 2017 at 10:48 AM Neva Durand <neva@broadinstitute.org> wrote:
> > > Hello Yunjiang,
> > >
> > > Just to double check, have you downloaded the latest jar from http://www.aidenlab.org/commandlinetools/ ?
> > >
> > > Best
> > > Neva
> > >
> > > On Mon, Jan 16, 2017 at 6:06 PM, Yunjiang Qiu <serein927@gmail.com> wrote:
> > > > Hi,
> > > >
> > > > I encountered the error "juicebox.tools.clt.CommandLineParser cannot be cast to juicebox.tools.clt.CommandLineParserForJuicer" when I tried to run hiccupsdiff. The command I used is below.
> > > >
> > > > java -Djava.awt.headless=true -Djava.library.path=juicebox/native_launcher/natives/ -Xmx12288m -Xms12288m -jar juicebox_tools_7.0.jar hiccupsdiff a.hic b.hic a/hiccups_loops b/hiccups_loops wt_dko &>hiccupdiff.log
> > > >
> > > > Thank you!

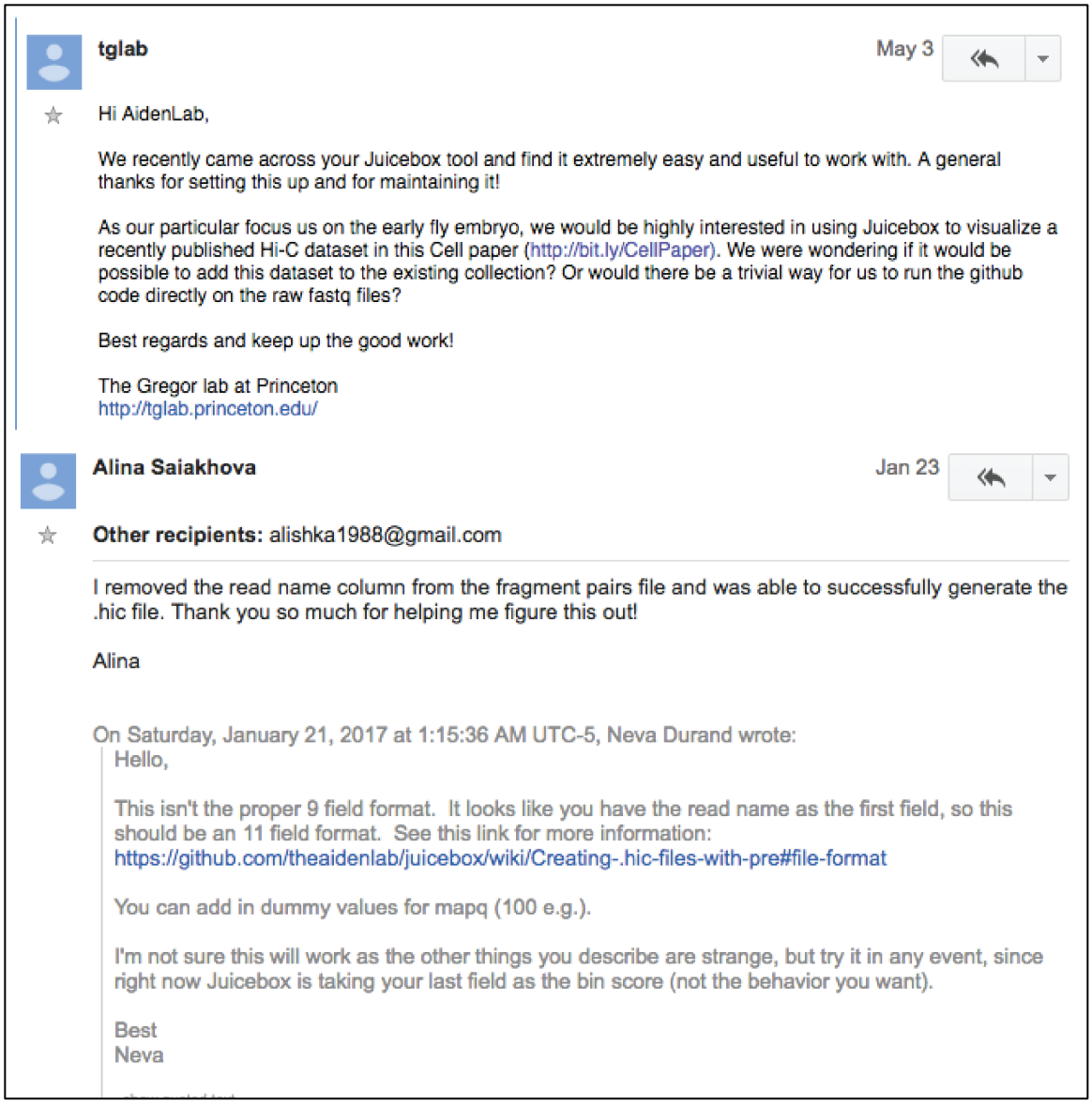
yufulong421@gmail.com 12/2/16

Hi, Neva

I want to run some raw data of HiC data which were not existed in https://bcm.app.box.com/v/aidenlab. And in https://github.com/theaidenlab/juicer and Juicer paper, you mentioned that Juicer can run on either a single machine, or clusters that run LSF, Univa Grid Engine, or SLURM. However, my lab sever do not have any software of LSF, Univa Grid Engine, or SLURM, and I can not run Juicer on our sever, can you tell me how to run juicer on a single machine?

Yours,
Fulong

Figure 28. Forum screenshots. We actively engage with the users of our software, fixing bugs they encounter and implementing new features based on their desires. We support many different modalities. Based on the second post here, we published a single machine version of Juicer shortly after.



tglab May 3

Hi AidenLab,

We recently came across your Juicebox tool and find it extremely easy and useful to work with. A general thanks for setting this up and for maintaining it!

As our particular focus us on the early fly embryo, we would be highly interested in using Juicebox to visualize a recently published Hi-C dataset in this Cell paper (http://bit.ly/CellPaper). We were wondering if it would be possible to add this dataset to the existing collection? Or would there be a trivial way for us to run the github code directly on the raw fastq files?

Best regards and keep up the good work!

The Gregor lab at Princeton
http://tglab.princeton.edu/

Alina Saiakhova Jan 23

Other recipients: alishka1988@gmail.com

I removed the read name column from the fragment pairs file and was able to successfully generate the .hic file. Thank you so much for helping me figure this out!

Alina

On Saturday, January 21, 2017 at 1:15:36 AM UTC-5, Neva Durand wrote:

> Hello,
>
> This isn't the proper 9 field format. It looks like you have the read name as the first field, so this should be an 11 field format. See this link for more information:
> https://github.com/theaidenlab/juicebox/wiki/Creating-.hic-files-with-pre#file-format
>
> You can add in dummy values for mapq (100 e.g.).
>
> I'm not sure this will work as the other things you describe are strange, but try it in any event, since right now Juicebox is taking your last field as the bin score (not the behavior you want).
>
> Best
> Neva

Figure 29. Forum requests and praise. We try to be quick and responsive to questions and are gratified when we hear that people find our software useful.

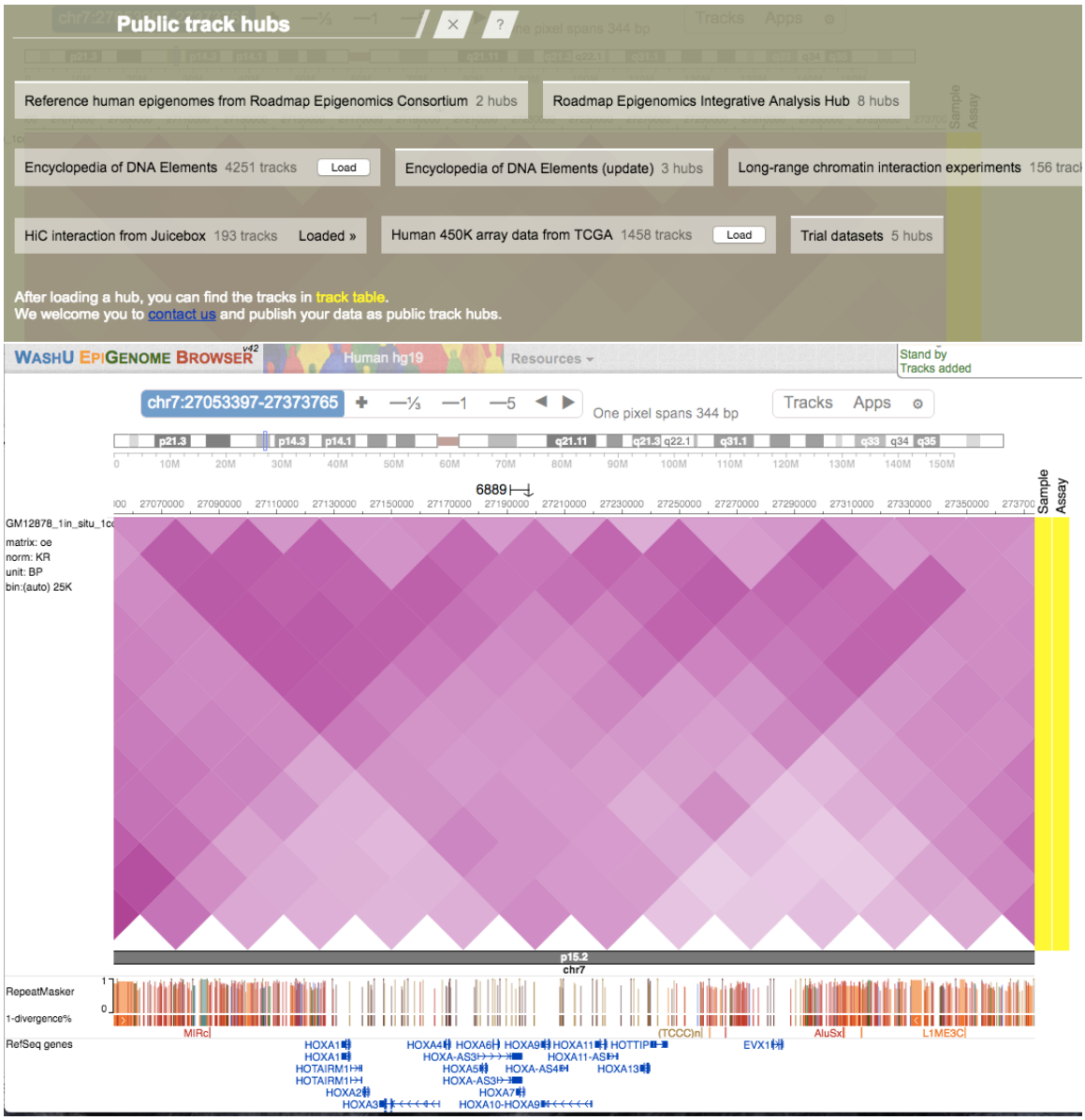


Figure 30. The WashU EpiGenome Browser includes Hi-C data from Juicebox in its track hub.

IV. Interoperability

ChIA-PET and Hi-C are different experimental approaches to the same problem: interrogating the conformation of the 3D genome. The processed data after both experiments is a list of contacts: locations in the genome where the experiment captured proximity events. The Juicer pipeline produces a *.hic* file, storing the contacts binned at multiple resolutions. As mentioned earlier, the ChIA-PET pipeline can also easily produce *.hic* files. The output of the ChIA-PET pipeline is a list of contacts in BAM format; a single step suffices to transform this into a valid pairs file. The *.hic* files can in turn be visualized in Juicebox (Figure 33).

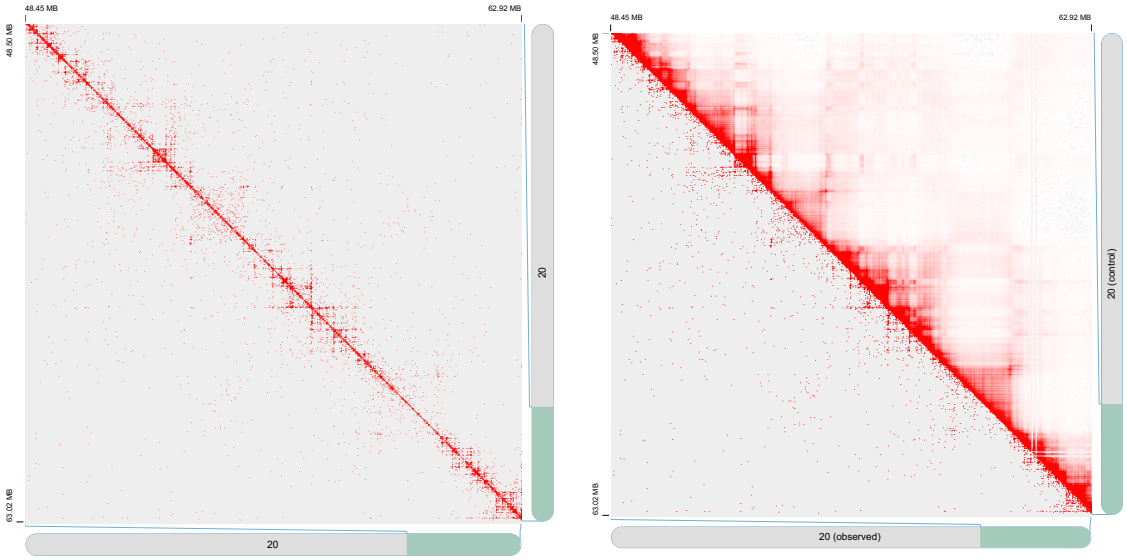


Figure 32. ChIA-PET and Hi-C GM12878 data in Juicebox at 25Kb. On the left, the ChIA-PET data shows the same structures visible in the much deeper Hi-C data on the right. The image shown is "Observed versus Control", where the observed data set is ChIA-PET (beneath the diagonal) and the control data set is Hi-C.

Visualizing the experiments aides in analysis and comparison. In Figure 33, we see a region of chromosome 20 in ChIA-PET and then look at it together with Hi-C data. Structures that are visible in the ChIA-PET data are also noticeable in the Hi-C data.

Both experiments aim to annotate loops. Any loop annotation can be easily visualized in Juicebox by following the format outlined in III.c.3. Data Formats. The format is agnostic as to how the loops were called in the first place. For example, Figure 32 shows ChIA-PET loops loaded onto the Observed versus Control map from before, this time at 10Kb resolution. With this kind of view, the user can see both the evidence in the ChIA-PET data leading to the loop call and what the denser Hi-C map looks like in the same location.

The loop calls from many different experiments are available in Juicebox for cross-validation, under *Load 2D Annotation*. In particular, the ChIA-PET loops from [47] and the 5C ENCODE loops from [48] can easily be examined, together with all of the loop calls from [10]. Furthermore, any new loop annotations may be loaded into the software via the *Local* button under the Annotation menu.

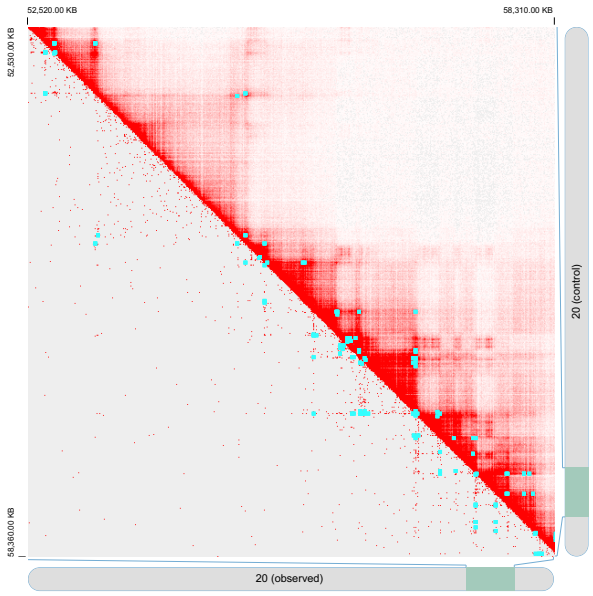


Figure 31. ChIA-PET loops in Juicebox.

V. References

1. Schleif, R., *DNA looping.* Annual review of biochemistry, 1992. **61**(1): p. 199-223.
2. Blackwood, E. and J. Kadonaga, *Going the distance: a current view of enhancer action.* Science (New York, N.Y.), 1998. **281**(5373): p. 60-63.
3. Ptashne, M., *Gene regulation by proteins acting nearby and at a distance.* Nature, 1986. **322**(6081): p. 697-701.
4. Tolhuis, B., et al., *Looping and interaction between hypersensitive sites in the active beta-globin locus.* Molecular cell, 2002. **10**(6): p. 1453-1465.
5. Gaszner, M. and G. Felsenfeld, *Insulators: exploiting transcriptional and epigenetic mechanisms.* Nature Reviews: Genetics, 2006. **7**(9): p. 703-713.
6. Phillips, J. and V. Corces, *CTCF: master weaver of the genome.* Cell, 2009. **137**(7): p. 1194-1211.
7. Cullen, K., M. Kladde, and M. Seyfred, *Interaction between transcription regulatory regions of prolactin chromatin.* Science, 1993. **261**(5118): p. 203-206.
8. Dekker, J., et al., *Capturing chromosome conformation.* Science, 2002. **295**(5558): p. 1306-1311.
9. Lieberman-Aiden, E., et al., *Comprehensive mapping of long-range interactions reveals folding principles of the human genome.* Science, 2009. **326**(5950): p. 289-293.
10. Rao, S.S., et al., *A 3D map of the human genome at kilobase resolution reveals principles of chromatin looping.* Cell, 2014. **159**(7): p. 1665-80.
11. Burton, J.N., et al., *Chromosome-scale scaffolding of de novo genome assemblies based on chromatin interactions.* Nature biotechnology, 2013. **31**(12): p. 1119-1125.
12. Selvaraj, S., et al., *Whole-genome haplotype reconstruction using proximity-ligation and shotgun sequencing.* Nature biotechnology, 2013. **31**(12): p. 1111-1118.
13. Dudchenko, O., et al., *De novo assembly of the Aedes aegypti genome using Hi-C yields chromosome-length scaffolds.* Science, 2017. **356**(6333): p. 92-95.
14. Sanborn, A.L., et al., *Chromatin extrusion explains key features of loop and domain formation in wild-type and engineered genomes.* Proceedings of the National Academy of Sciences, 2015. **112**(47): p. E6456-E6465.
15. Lander, E.S. and M.S. Waterman, *Genomic mapping by fingerprinting random clones: a mathematical analysis.* Genomics, 1988. **2**(3): p. 231-9.
16. Durand, N.C., et al., *Juicebox provides a visualization system for Hi-C contact maps with unlimited zoom.* Cell Systems, 2016. **3**(1): p. 99-101.
17. Durand, N.C., et al., *Juicer Provides a One-Click System for Analyzing Loop-Resolution Hi-C Experiments.* Cell Syst, 2016. **3**(1): p. 95-8.
18. Kruithof, J., *Telefoonverkeersrekening.* De Ingenieur, 1937. **52**(8): p. E15-E25.
19. Sinkhorn, R. and P. Knopp, *Concerning nonnegative matrices and doubly stochastic matrices.* Pacific Journal of Mathematics, 1967. **21**(2): p. 343-348.
20. Brown, J.B., P.J. Chase, and A.O. Pittenger, *Order independence and factor convergence in iterative scaling.* Linear Algebra Applications, 1993. **190**: p. 7-11.
21. Knight, P., *The Sinkhorn-Knopp algorithm: Convergence and applications.* SIAM Journal on Matrix Analysis and Applications, 2008. **30**(1): p. 261-275.
22. Deming, W.E. and F.F. Stephan, *On a Least Squares Adjustment of a Sampled Frequency Table When the Expected Marginal Totals are Known.* The Annals of Mathematical Statistics, 1940. **11**(4): p. 427-444.
23. Csiszár, I., *I-Divergence Geometry of Probability Distributions and Minimization Problems.* The Annals of Probability, 1975. **3**(1): p. 146-158.
24. Cournac, A., et al., *Normalization of a chromosomal contact map.* BMC genomics, 2012. **13**: p. 436.
25. Imakaev, M., et al., *Iterative correction of Hi-C data reveals hallmarks of chromosome organization.* Nature methods, 2012. **9**(10): p. 999-1003.
26. Knight, P. and D. Ruiz, *A fast algorithm for matrix balancing.* IMA Journal of Numerical Analysis, 2012.
27. Li, H., *Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM.* arXiv preprint arXiv:1303.3997, 2013.
28. Sexton, T., et al., *Three-dimensional folding and functional organization principles of the Drosophila genome.* Cell, 2012. **148**(3): p. 458-472.
29. Dixon, J., et al., *Topological domains in mammalian genomes identified by analysis of chromatin interactions.* Nature, 2012. **485**(7398): p. 376-380.
30. Schones, D.E., A.D. Smith, and M.Q. Zhang, *Statistical significance of cis-regulatory modules.* BMC Bioinformatics, 2007. **8**: p. 19.

31. Kim, T., et al., *Analysis of the vertebrate insulator protein CTCF-binding sites in the human genome.* Cell, 2007. **128**(6): p. 1231-1245.
32. Pedregosa, F., et al., *Scikit-lern: Machine Learning in Python.* Journal of Machine Learning Research, 2011. **12**: p. 2825-2830.
33. Lander, E.S., et al., *Initial sequencing and analysis of the human genome.* Nature, 2001. **409**(6822): p. 860-921.
34. Chinwalla, A.T., et al., *Initial sequencing and comparative analysis of the mouse genome.* Nature, 2002. **420**(6915): p. 520-562.
35. Consortium, S., *Genome sequence of the nematode $ C. elegans $: A platform for investigating biology.* Science, 1998. **282**: p. 2012-2018.
36. Darrow, E.M., et al., *Deletion of DXZ4 on the human inactive X chromosome alters higher-order genome architecture.* Proceedings of the National Academy of Sciences, 2016: p. 201609643.
37. Harris, R.S., *Improved pairwise alignment of genomic DNA*. 2007: The Pennsylvania State University.
38. Kruskal, J.B., *On the shortest spanning subtree of a graph and the traveling salesman problem.* Proceedings of the American Mathematical society, 1956. **7**(1): p. 48-50.
39. Vinson, J.P., et al., *Assembly of polymorphic genomes: algorithms and application to Ciona savignyi.* Genome research, 2005. **15**(8): p. 1127-1135.
40. Chin, C.-S., et al., *Phased diploid genome assembly with single-molecule real-time sequencing.* Nature Methods, 2016. **13**(12): p. 1050-1054.
41. Nene, V., et al., *Genome sequence of Aedes aegypti, a major arbovirus vector.* Science, 2007. **316**(5832): p. 1718-1723.
42. Robinson, J.T., et al., *Integrative genomics viewer.* Nature biotechnology, 2011. **29**(1): p. 24-26.
43. Servant, N., et al., *HiC-Pro: an optimized and flexible pipeline for Hi-C data processing.* Genome biology, 2015. **16**(1): p. 259.
44. Kubo, N., et al., *Preservation of Chromatin Organization after Acute Loss of CTCF in Mouse Embryonic Stem Cells.* bioRxiv, 2017.
45. Haarhuis, J.H.I., et al., *The Cohesin Release Factor WAPL Restricts Chromatin Loop Extension.* Cell. **169**(4): p. 693-707.e14.
46. Jung, Y.H., et al., *Chromatin States in Mouse Sperm Correlate with Embryonic and Adult Regulatory Landscapes.* Cell reports, 2017. **18**(6): p. 1366-1382.
47. Li, G., et al., *Extensive promoter-centered chromatin interactions provide a topological basis for transcription regulation.* Cell, 2012. **148**(1-2): p. 84-98.
48. Sanyal, A., et al., *The long-range interaction landscape of gene promoters.* Nature, 2012. **489**(7414): p. 109-113.