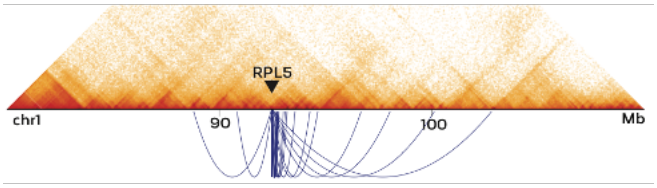
Omni-C Documentation

Release 0.1

Dovetail Genomics

Jul 20, 2021

CONTENTS:


RPL5
chr1
90
100
Mb

CHAPTER

ONE

OVERVIEW

- The Dovetail™ Omni-C™ library uses a sequence-independent endonuclease for chromatin digestion prior to proximity ligation and library generation.
- Chromatin interactions: Omni-C Libraries Enable Genome Wide Resolution of Chromatin Interactions. By employing a sequence-independent endonuclease for chromatin digestion, Omni-C™ offers all the characteristics of Hi-C libraries, without the sequence bias inherent to restriction enzyme (RE) based Hi-C approaches. Omni-C™ data contains a significant overlap with data generated using RE based approaches, but are enriched in long-range cis reads. Improved resolution for chromatin conformation and looping interactions can be measured due to this lack of sequence bias. Omni-C™ libraries enable the most complete view of genome-wide chromatin conformation by dramatically increasing resolution of topological interactions that occur in regions with low restriction enzyme density.
- Key benefits of Omni-C:
  - Sequence independent chromatin fragmentation enables fully genome-wide detection of chromatin contacts (up to 20% of the genome lacks coverage using restriction enzyme based Hi-C approaches)
  - Shotgun sequencing-like even genome coverage enabling SNP calling, chromosome phasing and structural variant detection
  - Lower sequencing burden to reach desired sequence depth saving time and cost
- SNPs and chromosome phasing: The even sequence coverage from Omni-C™ libraries enables genome-wide SNP calling and downstream applications reliant on SNP information, such as chromosome phasing due to low switch error rates. Omni-C™ technology offers the best possible approach for whole genome physical phasing using Illumina short reads.
- Large SVs are aaptured in Omni-C™ data: The proximity ligation data can be used to detect and confirm chromosomal rearrangements in cancer samples at a high resolution. Using open-source software tools such as HiGlass, contact matrices enable the quick visualization of such large structural variants.
- This guide will take you step by step on how to QC your Omni-C library, how to interparate the QC results and how to generate *contact maps*, study *chromatin structure*, use Omni-C data analyzing and enhancing your assembly and more. If you don't yet have a sequenced Omni-C library and you want to get familiar with the data, you can download Omni-C sequenced libraries from our publicaly available *data sets*.
- The QC process starts with aligning the reads to a reference genome then retaining high quality mapped reads. From there the mapped data will be used to generating a pairs file with pairtools, which categorizes pairs by read type and insert distance, this step both flags and removes PCR duplicates. Once pairs are categorized, counts of each class are summed and reported.
- If this is your first time following this tutorial, please check the *Before you begin page* first.


The full process including *installation*, *aligning and filtering*, *library QC*, *generating contact map* and *identifying chromatin structures* can be completed in less than 48hr compute time for *a 800M reads human data set* on an Ubuntu 18.04 machine with a 2T volume, 16 CPUs and 64GiB.

1.1 Before you begin

1.1.1 Have a copy of the Omni-C scripts on your machine:

Clone this repository:

```
git clone https://github.com/dovetail-genomics/Omni-C.git
```

1.1.2 Dependencies

Make sure that the following dependencies are installed:

- pysam
- tabulate
- bedtools
- matplotlib
- pandas
- bwa
- pairtools
- samtools
- preseq

If you are facing any issues with the installation of any of the dependencies, please contact the supporter of the relevant package.

python3 and pip3 are required, if you don't already have them installed, you will need sudo privileges.

- Update and install python3 and pip3:

```
sudo apt-get update
sudo apt-get install python3 python3-pip
```

- To set python3 and pip3 as primary alternative:

```
sudo update-alternatives --install /usr/bin/python python /usr/bin/python3 1
sudo update-alternatives --install /usr/bin/pip pip /usr/bin/pip3 1
```

If you are working on a new machine and don't have the dependencies, you can use the `installDep.sh` script in this repository for updating your instance and installing the dependencies and python3. This process will take approximatley 10' and requires sudo privileges. The script was tested on Ubuntu 18.04 with the latest version as of 04/11/2020

If you choose to run the provided installation script you will first need to set the permission to the file:

```
chmod +x ./Omni-C/installDep.sh
```

And then run the installation script:

```
./Omni-C/installDep.sh
```

Remember!

Once the installation is completed, sign off and then sign back to your instance to refresh the database of applications.

1.1.3 Input files

For this tutorial you will need:

- **fastq files** R1 and R2, either fastq or fastq.gz are acceptable
- **reference in a fasta file format**, e.g. hg38

If you don't already have your own input files or want to run a test on a small data set, you can download sample fastq files from the *Omni-C Data Sets section*. The 2M data set is suitable for a quick testing of the instruction in this tutorial.

```
wget https://s3.amazonaws.com/dovetail.pub/HiC/fastqs/OmniC_2M_R1.fastq
wget https://s3.amazonaws.com/dovetail.pub/HiC/fastqs/OmniC_2M_R2.fastq
```

1.2 Pre-Alignment

For downstream steps you will need a genome file, genome file is a tab delimited file with chromosome names and their respective sizes. If you don't already have a genome file follow these steps:

1. Generate an index file for your reference, a reference file with only the main chromosomes should be used (e.g. without alternative or unplaced chromosomes).

Command:

```
samtools faidx <ref.fasta>
```

Example:

```
samtools faidx hg38.fasta
```

faidx will index the ref file and create <ref.fasta>.fai on the reference directory.

2. Use the index file to generate the genome file by printing the first two columns into a new file.

Command:

```
cut -f1,2 <ref.fasta.fai> > <ref.genome>
```

Example:

```
cut -f1,2 hg38.fasta.fai > hg38.genome
```

In line with the 4DN project guidelines and from our own experience optimal alignment results are obtained with Burrows-Wheeler Aligner (bwa). Prior to alignment, generate a bwa index file for the chosen reference.

```
bwa index <ref.fasta>
```

Example:

```
bwa index hg38.fasta
```

No need to specify an output path, the bwa index files are automatically generated at the reference directory. Please note that this step is time consuming, however you need to run it only once for a reference.

To avoid memory issues, some of the steps require writing temporary files into a temp folder, please generate a temp folder and remember its full path. Temp files may take up to x3 of the space that the fastq.gz files are taking, that is, if the total volume of the fastq files is 5Gb, make sure that the temp folder can store at least 15Gb.

Command:

```
mkdir <full_path/to/tmpdir>
```

Example:

```
mkdir /home/ubuntu/ebs/temp
```

In this example the folder *temp* will be generated on a mounted volume called *ebs* on a user account *ubuntu*.

1.3 From fastq to final valid pairs bam file

fastq to final valid pairs bam file - for the impatient!

If you just want to give it a shot and run all the alignment and filtering steps without going over all the details, we made a shorter version for you, with all the steps piped, outputting a final bam file with its index file and a dup stats file, otherwise move to the next section *fastq to final valid pairs bam file - step by step*

Command:

```
bwa mem -5SP -T0 -t<cores> <ref.fa> <OmniC.R1.fastq.gz> <OmniC.R2.fastq.gz>| \
pairtools parse --min-mapq 40 --walks-policy 5unique \
--max-inter-align-gap 30 --nproc-in <cores> --nproc-out <cores> --chroms-path <ref.
↪genome> | \
pairtools sort --tmpdir=<full_path/to/tmpdir> --nproc <cores>|pairtools dedup --nproc-in
↪<cores> \
--nproc-out <cores> --mark-dups --output-stats <stats.txt>|pairtools split --nproc-in
↪<cores> \
--nproc-out <cores> --output-pairs <mapped.pairs> --output-sam -|samtools view -bS -@
↪<cores> | \
samtools sort -@<cores> -o <mapped.PT.bam>;samtools index <mapped.PT.bam>
```

Example:

```
bwa mem -5SP -T0 -t16 hg38.fasta OmniC_2M_R1.fastq OmniC_2M_R2.fastq| pairtools parse --
↪min-mapq 40 --walks-policy 5unique --max-inter-align-gap 30 --nproc-in 8 --nproc-out 8␣
↪--chroms-path hg38.genome | pairtools sort --tmpdir=/home/ubuntu/ebs/temp/ --nproc␣
↪16|pairtools dedup --nproc-in 8 --nproc-out 8 --mark-dups --output-stats stats.
↪txt|pairtools split --nproc-in 8 --nproc-out 8 --output-pairs mapped.pairs --output-
↪sam -|samtools view -bS -@16 | samtools sort -@16 -o mapped.PT.bam;samtools index␣
↪mapped.PT.bam
```


The full command above, with 2M read pairs on an Ubuntu 18.04 machine with 16 CPUs and 64GiB was completed in less than 5 minutes. On the same machine type.

1.3.1 fastq to final valid pairs bam file - step by step

Alignment

Now that you have a genome file, index file and a reference fasta file you are all set to align your Omni-C library to the reference. Please note the specific settings that are needed to map mates independently and for optimal results with our proximity library reads.

Parameter	Alignment function
mem	set the bwa to use the BWA-MEM algorithm, a fast and accurate alignment algorithm optimized for sequences in the range of 70bp to 1Mbp
-5	for split alignment, take the alignment with the smallest coordinate (5' end) as primary, the mapq assignment of the primary alignment is calculated independent of the 3' alignment
-S	skip mate rescue
-P	skip pairing; mate rescue performed unless -S also in use
-T0	The T flag set the minimum mapping quality of alignments to output, at this stage we want all the alignments to be recorded and thus T is set up to 0, (this will allow us to gather full stats of the library, at later stage we will filter the alignments by mapping quality
-t	number of threads, default is 1. Set the numbers of threads to not more than the number of cores that you have on your machine (If you don'd know the number of cores, used the command lscpu and multiply Thread(s) per core x Core(s) per socket x Socket(s))
*.fasta or *.fa	Path to a reference file, ending with .fa or .fasta, e,g, hg38.fasta
*.fastq or *.fastq.gz	Path to two fastq files; path to read 1 fastq file, followed by fastq file of read 2 (usually labeled as R1 and R2, respectively). Files can be in their compressed format (.fastq.gz) or uncompressed (.fastq). In case your library sequence is divided to multiple fastq files, you can use a process substitution < with the cat command (see example below)
-o	sam file name to use for output results [stdout]. You can choose to skip the -o flag if you are piping the output to the next command using '|'

Bwa mem will output a sam file that you can either pipe or save to a path using -o option, as in the example below (please note that version 0.7.17 or higher should be used, older versions do not support the *-5* flag)

Command:

```
bwa mem -5SP -T0 -t<threads> <ref.fasta> <OmniC_R1.fastq> <OmniC_R2.fastq> -o <aligned.
↪sam>
```

Example (one pair of fastq files):

```
bwa mem -5SP -T0 -t16 hg38.fasta OmniC_2M_R1.fastq OmniC_2M_R2.fastq -o aligned.sam
```

Example (multiple pairs of fastq files):

```
bwa mem -5SP -T0 -t16 hg38.fasta <(zcat file1.R1.fastq.gz file2.R1.fastq.gz file3.R1.
↪fastq.gz) <(zcat file1.R2.fastq.gz file2.R2.fastq.gz file3.R2.fastq.gz) -o aligned.sam
```

Recording valid ligation events

We use the `parse` module of the `pairtools` pipeline to find ligation junctions in Omni-C (and other proximity ligation) libraries. When a ligation event is identified in the alignment file the pairtools pipeline will record the outer-most (5') aligned base pair and the strand of each one of the paired reads into `.pairsam` file (pairsam format captures SAM entries together with the Hi-C pair information). In addition, it will also asign a pair type for each event. e.g. if both reads aligned uniquely to only one region in the genome, the type UU (Unique-Unique) will be assigned to the pair. The following steps are necessary to identify the high quality valid pairs over low quality events (e.g. due to low mapping quality):

`pairtools parse` options:

Parameter	Value	Function
min-mapq	40	Mapq threshold for defining an alignment as a multi-mapping alignment. Alignment with mapq <40 will be marked as type M (multi)
walks-policy	5unique	Walks is the term used to describe multiple ligations events, resulting three alignments (instead of two) for a read pair. However, there are cases in which three alignment in read pairs are the result of one ligation event, pairtool parse can rescue this event. walks-policy is the policy for reporting un-rescuable walk. 5unique is used to report the 5'-most unique alignment on each side, if present (one or both sides may map to different locations on the genome, producing more than two alignments per DNA molecule)
max-inter-align-gap	30	In cases where there is a gap between alignments, if the gap is 30 or smaller, ignore the gap, if the gap is >30bp, mark as "null" alignment
nproc-in	integer, e.g. 16	pairtools has an automatic-guess function to identify the format of the input file, whether it is compressed or not. When needed, the input is decompressed by bgzip/lz4c. The option nproc-in set the number of processes used by the auto-guessed input decompressing command, if not specified, default is 3
nproc-out	integer, e.g. 16	pairtools automatic-guess the desired format of the output file (compressed or not compressed, based on file name extension). When needed, the output is compressed by bgzip/lz4c. The option nproc-out set the number of processes used by the auto-guessed output compressing command, if not specified, default is 8
chroms-path		path to .genome file, e.g. hg38.genome
*.sam		path to sam file used as an input. If you are piping the input (stdin) skip this option
*pairsam		name of pairsam file for writing output results. You can choose to skip and pipe the output directly to the next commad (pairtools sort)

`pairtools parse` command example for finding ligation events:

Command:

```
 pairtools parse --min-mapq 40 --walks-policy 5unique --max-inter-align-gap 30 --nproc-
↪in <cores>\
--nproc-out <cores> --chroms-path <ref.genome> <aligned.sam> > <parsed.pairsam>
```

Example:

```
pairtools parse --min-mapq 40 --walks-policy 5unique --max-inter-align-gap 30 --nproc-in␣
↪8 --nproc-out 8 --chroms-path hg38.genome aligned.sam >  parsed.pairsam
```

At the parsing step, pairs will be flipped such that regardless of read1 and read2, pairs are always recorded with first side of the pair having the lower genomic coordinates.

Sorting the pairsam file

The parsed pairs are then sorted using *pairtools sort*

`pairtools sort` options:

Parameter	Function
–tmpdir	Provide a full path to a temp directory. A good rule of thumb is to have a space available for this directory at a volume of x3 of the overall volume of the fastq.gz files. Using a temp directory will help avoid memory issues
–nproc	Number of processes to split the sorting work

Command:

```
pairtools sort --nproc <cores> --tmpdir=<path/to/tmpdir> <parsed.pairsam> > <sorted.
↪pairsam>
```

Example:

```
pairtools sort --nproc 16 --tmpdir=/home/ubuntu/ebs/temp/  parsed.pairsam > sorted.
↪pairsam
```

Important!

Please note that an absolute path for the temp directory is required for `pairtools sort`, e.g. path of the structure ~/ebs/temp/ or ./temp/ will not work, instead, something of this sort is needed /home/user/ebs/temp/

Removig PCR duplicates

`pairtools dedup` detects molecules that could be formed via PCR duplication and tags them as "DD" pair type. These pairs should be excluded from downstream analysis. Use the pairtools dedup command with the *–output-stats* option to save the dup stats into a text file.

`pairtools dedup` options:

Parameter	Function
–mark-dups	If specified, duplicate pairs are marked as DD in "pair_type" and as a duplicate in the sam entries
–output-stats	Output file for duplicate statistics. Please note that if a file with the same name already exists, it will be opened in the append mode

Command:

```
pairtools dedup --nproc-in <cores> --nproc-out <cores> --mark-dups --output-stats <stats.
↪txt> \
--output <dedup.pairsam> <sorted.pairsam>
```

Example:

```
pairtools dedup --nproc-in 8 --nproc-out 8 --mark-dups --output-stats stats.txt --output␣
↪dedup.pairsam sorted.pairsam
```

Generating .pairs and bam files

The `pairtools split` command is used to split the final `.pairsam` into two files: `.sam` (or `.bam`) and `.pairs` (`.pairsam` has two extra columns containing the alignments from which the Omni-C pair was extracted, these two columns are not included in `.pairs` files)

`pairtools split` options:

Parameter	Function
–output-pairs	Output pairs file. If the path ends with .gz or .lz4 the output is pbgzip-/lz4c-compressed. If you wish to pipe the command and output the pairs fils to stdout use - instead of file name
–output-sam	Output sam file. If the file name extension is .bam, the output will be written in bam format. If you wish to pipe the command, use - instead of a file name. please note that in this case the sam format will be used (and can be later converted to bam file e.g. with the command samtools view -bS -@16 -o temp.bam

Command:

```
pairtools split --nproc-in <cores> --nproc-out <cores> --output-pairs <mapped.pairs> \
--output-sam <unsorted.bam> <dedup.pairsam>
```

Example:

```
pairtools split --nproc-in 8 --nproc-out 8 --output-pairs mapped.pairs --output-sam␣
↪unsorted.bam dedup.pairsam
```

The `.pairs` file can be used for generating *contact matrix*

Generating the final bam file

For downstream steps, the bam file should be sorted, using the command *samtools sort*

`samtools sort` options:

Parameter	Function
-@	number of threads to use
-o	ile name. Write final output to FILE rather than standard output
-T	path to temp file. Using a temp file will help avoiding memory issues

Command:

```
samtools sort -@<threads> -T <path/to/tmpdir/tempfile.bam>-o <mapped.PT.bam> <unsorted.
↪bam>
```

Example:

```
samtools sort -@16 -T /home/ubuntu/ebs/temp/temp.bam -o mapped.PT.bam unsorted.bam
```

For future steps an index (.bai) of the bam file is also needed. Index the bam file:

Command:

```
samtools index <mapped.PT.bam>
```

Example:

```
samtools index mapped.PT.bam
```

The *mapped.PT.bam* is the final bam file that will be used downstream steps.

The above steps resulted in multiple intermediate files, to simplify the process and avoid intermediate files, you can pipe the steps as in the example above (*fastq to final valid pairs bam file - for the impatient*)

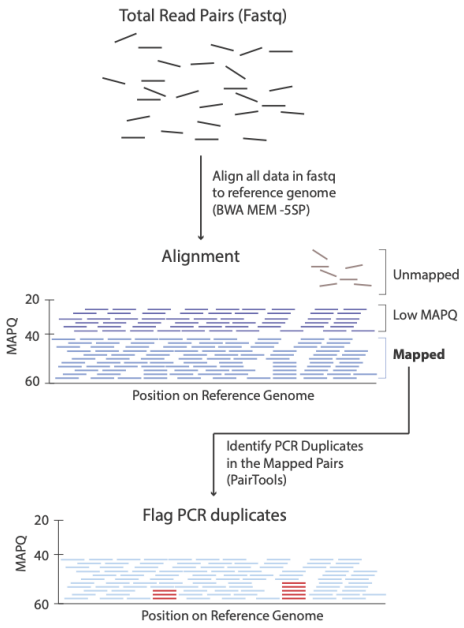
1.4 Library QC

At step *Removig PCR duplicates* you used the flag `--output-stats`, generating a stats file in addition to the pairsam output (e.g. –output-stats stats.txt). The stats file is an extensive output of pairs statistics as calculated by pairtools, including total reads, total mapped, total dups, total pairs for each pair of chromosomes etc'. Although you can use directly the pairtools stats file as is to get informed on the quality of the Omni-C library, we find it easier to focus on a few key metrics. We include in this repository the script `get_qc.py` that summarize the paired-tools stats file and present them in percentage values in addition to absolute values.

The images below explains how the values on the QC report are calculated:

I. Aligning and filtering to remove low mapping quality and PCR duplicate read pairs

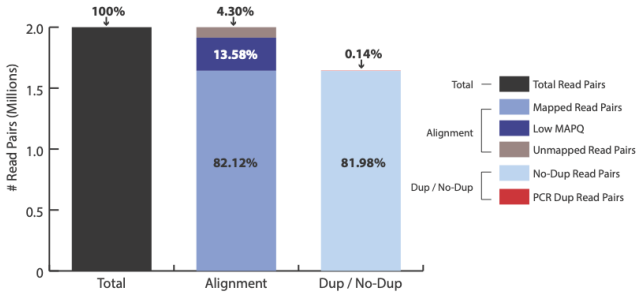
Process



Results

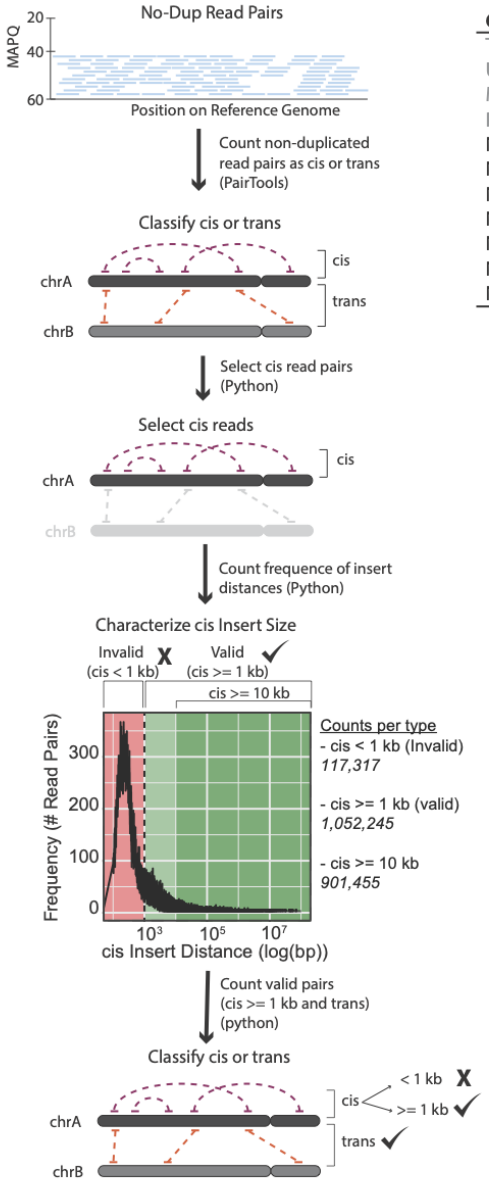
Category	Count	Percent
Total Read Pairs	2,000,000	100.00%
Unmapped Read Pairs	86,000	4.30%
Mapped Read Pairs	1,642,400	82.12%
PCR Dup Read Pairs	2,800	0.14%
No-Dup Read Pairs	1,640,800	81.98%
No-Dup Cis Read Pairs	1,169,563	71.28%
No-Dup Trans Read Pairs	471,317	28.72%
No-Dup Valid Read Pairs (cis >= 1 kb + trans)	1,523,483	92.85%
No-Dup Cis Read Pairs < 1kb	117,317	7.15%
No-Dup Cis Read Pairs >= 1kb	1,052,245	64.13%
No-Dup Cis Read Pairs >=10kb	901,455	54.94%

Proportion of Total Read Pairs



II. Classifying read pairs (cis or trans), characterizing insert size, and identifying valid pairs

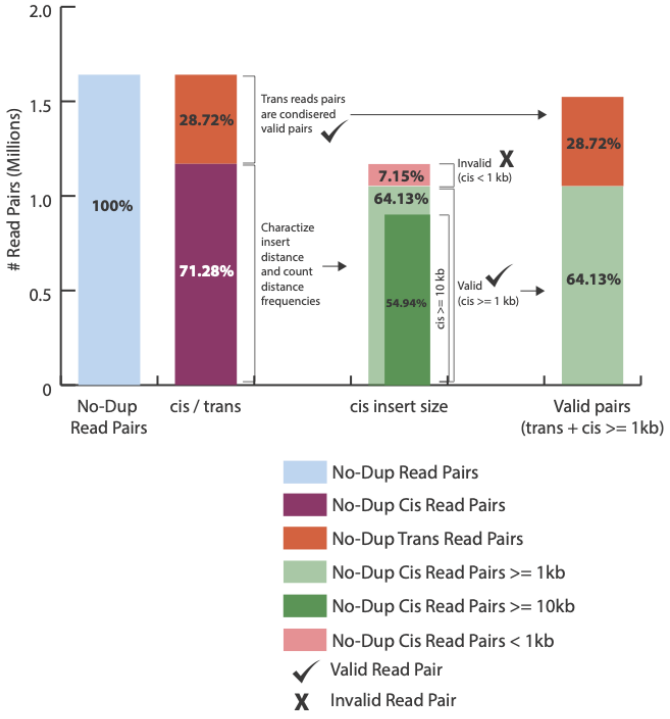
Process



Results

Category	Count	Percent
Total Read Pairs	2,000,000	100.00%
Unmapped Read Pairs	86,000	4.30%
Mapped Read Pairs	1,642,400	81.98%
PCR Dup Read Pairs	2,800	0.14%
No-Dup Read Pairs	1,640,800	82.04%
No-Dup Cis Read Pairs	1,169,563	71.28%
No-Dup Trans Read Pairs	471,317	28.72%
No-Dup Valid Read Pairs (cis >= 1 kb + trans)	1,523,483	92.85%
No-Dup Cis Read Pairs < 1kb	117,317	7.15%
No-Dup Cis Read Pairs >= 1kb	1,052,245	64.13%
No-Dup Cis Read Pairs >=10kb	901,455	54.94%

Proportion of No-Dup Read Pairs



Command:

```
python3 ./Omni-C/get_qc.py -p <stats.txt>
```

Example:

```
python3 ./Omni-C/get_qc.py -p stats.txt
```

After the script completes, it will print:

```
Total Read Pairs                              2,000,000  100%
Unmapped Read Pairs                           92,059     4.6%
Mapped Read Pairs                             1,637,655  81.88%
PCR Dup Read Pairs                            5,426      0.27%
No-Dup Read Pairs                             1,632,229  81.61%
No-Dup Cis Read Pairs                         1,288,943  78.97%
No-Dup Trans Read Pairs                       343,286    21.03%
No-Dup Valid Read Pairs (cis >= 1kb + trans)  1,482,597  90.83%
No-Dup Cis Read Pairs < 1kb                   149,632    9.17%
No-Dup Cis Read Pairs >= 1kb                  1,139,311  69.8%
No-Dup Cis Read Pairs >= 10kb                 870,490    53.33%
```

1.5 Library Complexity

If you preformed a shallow sequencing experiment (e.g. 2M reads) and running a QC analysis to decide which library to use for deep sequencing (DS), it is recommended to evaluate the complexity of the library before moving to DS.

The *lc_extrap* utility of the *preseq* package aims to predict the complexity of sequencing libraries.

`preseq` options:

Parameter	Value	Function
bam		Specifies that the input file type is bam. Please note that for a bam file to be a recognized input file htslib sould be installed as well and preseq should be built with htslib support (for more details see preseq documentation or our installDep.sh script as example)
pe		Specifies that paired end data is used
extrap	2.10E+09	Maximum extrapolation
step	1.00E+08	Extrapolation step size
seg_len	1000000000	maximum segment length when merging paired end bam
output		output file

Please note that the input bam file should be a version prior to dups removal.

`preseq lc_extrap` command example for extrapolating library complexity:

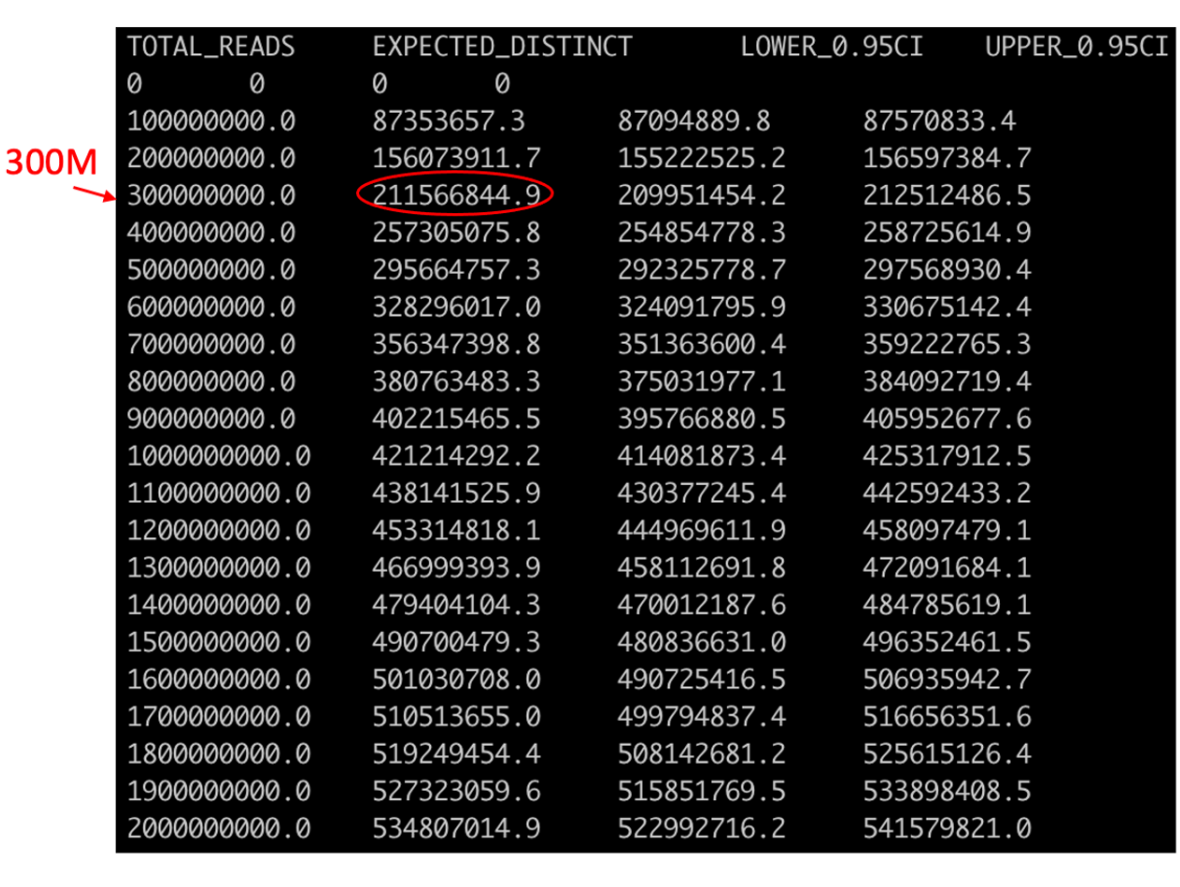
Command:

```
preseq lc_extrap -bam -pe -extrap 2.1e9 -step 1e8 -seg_len 1000000000 -output <output␣
↪file> <input bam file>
```

Example:

```
preseq lc_extrap -bam -pe -extrap 2.1e9 -step 1e8 -seg_len 1000000000 -output out.preseq␣
↪mapped.bam
```

In this example the output file *out.preseq* will detail the extrapolated complexity curve of your library, with the number of reads in the first column and the expected distinct read value in the second column. For a typical experiment (human sample) check the expected complexity at 300M reads (to show the content of the file, type **cat out.preseq**). Expected unique pairs at 300M sequencing is at least ~ 120 million.



TOTAL_READS	EXPECTED_DISTINCT	LOWER_0.95CI	UPPER_0.95CI
0	0	0	0
100000000.0	87353657.3	87094889.8	87570833.4
200000000.0	156073911.7	155222525.2	156597384.7
300000000.0	211566844.9	209951454.2	212512486.5
400000000.0	257305075.8	254854778.3	258725614.9
500000000.0	295664757.3	292325778.7	297568930.4
600000000.0	328296017.0	324091795.9	330675142.4
700000000.0	356347398.8	351363600.4	359222765.3
800000000.0	380763483.3	375031977.1	384092719.4
900000000.0	402215465.5	395766880.5	405952677.6
1000000000.0	421214292.2	414081873.4	425317912.5
1100000000.0	438141525.9	430377245.4	442592433.2
1200000000.0	453314818.1	444969611.9	458097479.1
1300000000.0	466999393.9	458112691.8	472091684.1
1400000000.0	479404104.3	470012187.6	484785619.1
1500000000.0	490700479.3	480836631.0	496352461.5
1600000000.0	501030708.0	490725416.5	506935942.7
1700000000.0	510513655.0	499794837.4	516656351.6
1800000000.0	519249454.4	508142681.2	525615126.4
1900000000.0	527323059.6	515851769.5	533898408.5
2000000000.0	534807014.9	522992716.2	541579821.0

1.6 Generating Contact Matrix

There are two common formats for contact maps, the Cooler format and Hic format. Both are compressed and sparsed formats to avoid large storage volumes; For a given n number of bins in the genome, the size of the matrix would be n^2, in addition, typically more than one resolution (bin size) is being used.

In this section we will guide you on how to generate both matrices types, *HiC* and *cool* based on the *.pairs file* that you generated in the *previous section* and how to visualize them.

1.6.1 Generating HiC contact maps using Juicer tools

Additional Dependencies

- Juicer Tools - Download the JAR file for juicertools and place it in the same directory as this repository and name it as `juicertools.jar`. You can find the link to the most recent version of Juicer tools here e.g.:

```
wget https://s3.amazonaws.com/hicfiles.tc4ga.com/public/juicer/juicer_tools_1.22.01.jar
mv juicer_tools_1.22.01.jar ./Omni-C/juicertools.jar
```

- Java - If not already installed, you can install Java as follows:

```
sudo apt install default-jre
```

From `.pairs` to `.hic` contact matrix

- Juicer Tools is used to convert `.pairs` file into a HiC contact matrix.
- `HiC` is highly compressed binary representation of the contact matrix
- Provides rapid random access to any genomic region matrix
- Stores contact matrix at 9 different resolutions (2.5M, 1M, 500K, 250K, 100K, 50K, 25K, 10K, and 5K)
- Can be programmatically manipulated using straw python API

The *.pairs* file that you generated in the *From fastq to final valid pairs bam file* section can be used directly with `Juicer tools` to generate the *HiC* contact matrix:

Parameter	Function
-Xmx	The flag Xmx specifies the maximum memory allocation pool for a Java virtual machine, from our experience 48000m works well when processing human data sets. If you are not sure how much memory your system has, run the command `free -h` and check the value under *total*.
Djava.awt.headless=true	Java is ran in a headless mode when the application does not interact with a user (if not specified, the default is Djava.awt.headless=false)
pre	The pre command allows users to create .hic files from their own data
–threads	Specifies the numbers of threads to be used (integer number)
*.pairs or *.pairs.gz	input file for generating the contact matrix
*.genome	genome file, listing the chromosomes and their sizes
*.hic	hic output file, containing the contact matrix

Tip no.1

Please note that if you have an older vesrion of `Juicer tools`, generating contact map directly from `.pairs` file may not be supported. We recommend updating to a newer version. As we tested, the `pre` utility of the version 1.22.01 support the .pairs to HiC function.

Command:

```
java -Xmx<memory>  -Djava.awt.headless=true -jar <path_to_juicer_tools.jar> pre --
↪threads <no_of_threads> <mapped.pairs> <contact_map.hic> <ref.genome>
```

Example:

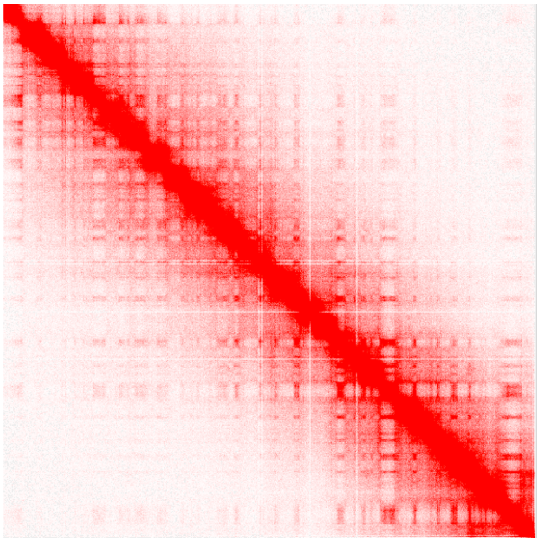
```
java -Xmx48000m  -Djava.awt.headless=true -jar ./Omni-C/juicer_tools.jar pre --threads␣
↪16 mapped.pairs contact_map.hic hg38.genome
```

Tip no.2

`Juicer tools` offers additional functions that were not discussed here, including matrix normalization and generating matrix for only specified regions in the genome. To learn more about advanced options, please refer to the Juicer Tools documentation.

Visualizing `.hic` contact matrix

The visualization tool `Juicebox` can be used to visualize the contact matrix. You can either download a local version of the tool to your computer as a Java application or use a web version of Juicebox. Load your `.hic` file to visualize the contact map and zoom in to areas of interest.



You can use the `.hic` contact matrix for calling *TADs*, identifying *A/B compartments* or even observing large structural variations and misassemblies.

1.6.2 Generating `cooler` contact maps

Additional Dependencies

Installing Cooler and its dependencies

- libhdf5 - `sudo apt-get install libhdf5-dev`
- h5py - `pip3 install h5py`
- cooler - `pip3 install cooler`

For any issues with `cooler` installation or its dependencies, please refer to the cooler installation documentation

Installing Pairix

Pairix is a tool for indexing and querying on a block-compressed text file containing pairs of genomic coordinates. You can install it directly from its github repository as follows:

```
git clone https://github.com/4dn-dcic/pairix
cd pairix
make
```

Add the bin path, and utils path to PATH and exit the folder:

```
PATH=~/pairix/bin/:~/pairix/util:~/pairix/bin/pairix:$PATH
cd ..
```

Important!

make sure to modify the following example with the path to your *pairix* installation folder. If you are not sure what is the path you can check it with the command *pwd* when located in the *pairix* folder.

For any issues with `pairix`, please refer to the pairix documentation

From `.pairs` to `cooler` contact matrix

- Cooler tools is used to convert **indexed** `.pairs` file into cool and mcool contact matrices
- `Cooler` generates a sparse, compressed, and binary persistent representation of proximity ligation contact matrix
- Store matrix as HDF5 file object
- Provides python API to manipulate contact matrix
- Each cooler matrix is computed at a specific resolution
- Multi-cool (mcool) files store a set of cooler files into a single HDF5 file object
- Multi-cool files are helpful for visualization

Indexing the `.pairs` file

We will use the `cload pairix` utility of `Cooler` to generate contact maps. This utility requires the `.pairs` file to be indexed. `Pairix` is used for indexing compressed `.pairs` files. The files should be compresses with bgzip (which should already be installed on your machine). If your `.pairs` file is not yet bgzip compressed, first compress it as follows:

Command:

```
bgzip <mapped.pairs>
```

Example:

```
bgzip mapped.pairs
```

Following this command `mapped.pairs` will be replaced with its compressed form `mapped.pairs.gz`

Note!

Compressing the `.pairs` file with `gzip` instead of `bgzip` will also result in a compressed file with the `.gz` suffix, but due to format differnces it will not be accepted as an input for `pairix`.

Next, index the file `.pairs.gz` file:

Command:

```
pairix <mapped.pairs.gz>
```

Example:

```
pairix mapped.pairs.gz
```

Genereting single resolution contact map files

As mentioned above, we will use the `cload pairix` utility of `Cooler` to generate contact maps:

`cooler cload pairix` usage:

Parameter	Function
<genome_fils>:<bin size>	Specifies the reference *.genome file*, followed with``:`` and the desired bin size in bp
-p	Number of processes to split the work between (integer), default: 8
*.pairs.gz	Path to `bgzip` compressed and indexed `.pairs` file
*.cool	Name of output file

Command:

```
cooler cload pairix -p <cores> <ref.genome>:<bin_size_in_bp> <mapped.pairs.gz> <matrix.
↪cool>
```

Example:

```
cooler cload pairix -p 16 hg38.genome:1000 mapped.pairs.gz matrix_1kb.cool
```

Genereting multi-resolutions files and visualizing the contact matrix

When you wish to visualize the contact matrix, it is highly recommended to generate a multi-resolution `.mcool` file to allow zooming in and out to inspect regions of interest. The cooler `zoomify` utility allows you to generate a multi-resolution cooler file by coarsening. The input to `cooler zoomify` is a single resolution `.cool` file, to allow zooming in into regoins of interest we suggest to generate a `.cool` file with a small bin size, e.g. 1kb. Multi-resolution files uses the suffix `.mcool`.

`cooler zoomify` usage:

Parameter	Function
–balance	Apply balancing to each zoom level. Off by default
-p	Number of processes to use for batch processing chunks of pixels, default: 1
*.cool	Name of contact matrix input file

*Command:**

```
cooler zoomify --balance -p <cores> <matrix.cool>
```

Example:

```
cooler zoomify --balance -p 16 matrix_1kb.cool
```

The example above will result in a new file named *matrix_1kb.mcool* (no need to specify output name)

Tip

`Cooler` offers additional functions that were not discussed here, including generating a cooler from a pre-binned matrix, matrix normalization and more. To learn more about advanced options, please refer to the cooler documentation

HiGlass is an interactive tool for visualizing `.mcool` files. To learn more about how to set up and use HiGlass follow the HiGlass tutorial

1.7 Epigenetics applications with Omni-C

In this section you will learn how to:

- *Identify Topologically Associated Domains (TADs)*
- *Identify A/B compartments*

1.7.1 Identify Topologically Associated Domains (TADs)

The *.hic* file that you generated can be used for identifying Topologically Associated Domains (TADs) in the contact map using the `arrowhead` utility of `Juicer Tools`

Parameter	Function
-Xmx	The flag Xmx specifies the maximum memory allocation pool for a Java virtual machine, from our experience 48000m works well when processing human data sets. If you are not sure how much memory your system has, run the command `free -h` and check the value under *total*.
Djava.awt.headless=true	Java is ran in a headless mode when the application does not interact with a user (if not specified, the default is Djava.awt.headless=false)
arrowhead	The arrowhead command is used to identify contact domains along the diagonal
--threads	Specifies the numbers of threads to be used (integer number)
-k	Normalization to be used. KR (Knight-Ruiz) balancing is recommended (other available normalization methods are VC and VC_SQRT or NONE, (this is not recommended)
-m	Size of the sliding window along the diagonal in which contact domains will be found. Must be an even number as (m/2) is used as the increment for the sliding window. Recommended value: 2000 (this is also the default)
-r	Resolution for which Arrowhead will be run. We recommend running with various resolutions, e.g.: 5kB (5000), 10kB (10000), 25kb (25000) and 50kb (50000). The quality of the results at the various resolutions is highly dependent on the depth of sequencing
*.hic	hic input file, containing the contact matrix
output directory	a path to the directory in which a bedpe file containing the contact domains will be saved (the name of the file is composed of the resolution followed by *_blocks.bedpe*). The bedpe file, is a text file with 16 fields in each row. For more details on the file format, see the *table* below

Command:

```
java -jar -Xmx<memory> -Djava.awt.headless=true -jar <path_to_juicer_tools.jar>␣
↪arrowhead --threads <no_of_threads> -k <normalization_type> -m <sliding_window> -r
↪<resolution> <*.hic> <output_path>
```

Example:

```
java -jar -Xmx48000m  -Djava.awt.headless=true -jar ./Omni-C/juicer_tools.jar arrowhead -
↪-threads 16 -k KR -m 2000 -r 10000 contact_map.hic TAD_calls
```

The above command will result in a new file "10000_blocks.bedpe" inside the "TAD_calls" directory.

The `bedpe` output file will contain a list of contact domains with the following 16-field header:

```
chr1 x1 x2 chr2  y1 y2 name  score strand1  strand2  color score uVarScore   lVarScore  ␣
↪upSign   loSign
```

Field	Description
chr1/chr2	The chromosome that the domain is located on
x1,x2/y1,y2	The interval spanned by the domain (contact domains manifest as squares on the diagonal of a Hi-C matrix and as such: x1=y1, x2=y2)
name	typically this field is empty (contains ".")
score	typically this field is empty (contains ".")
strand1	typically this field is empty (contains ".")
strand2	typically this field is empty (contains ".")
color	The color that the feature will be rendered as if loaded in Juicebox
score	The corner score, a score indicating the likelihood that a pixel is at the corner of a contact domain. Higher values indicate a greater likelihood of being at the corner of a domain
uVarScore	The variance of the upper triangle
lVarScore	The variance of the lower triangle
upSign	-1*(sum of the sign of the entries in the upper triangle)
loSign	Sum of the sign of the entries in the lower triangle

1.7.2 Principle component of the contact matrix

The contact matrix *.hic* can also be used for calculating the Pearson's correlation matrix of the Observed/Expected for intra-chromosomal contacts. The eigenvector is the first principal component of the Pearson's matrix and it implies on the compartmentalization of the DNA at high level. The sign of the eigenvector typically indicates the compartment type A/B (active/inactive).

The Principle component of the contact matrix can be calculated using the `eigenvector` utility of `Juicer Tools`

Parameter	Function
-Xmx	The flag Xmx specifies the maximum memory allocation pool for a Java virtual machine, from our experience 48000m works well when processing human data sets. If you are not sure how much memory your system has, run the command `free -h` and check the value under *total*.
Djava.awt.headless=true	Java is ran in a headless mode when the application does not interact with a user (if not specified, the default is Djava.awt.headless=false)
eigenvector	The eigenvector command is used to calculate the first principal component of the Pearson's Observed/Expected matrix
-p	Optional, force with curren parameters. When running with resolution < 500kb this may take very long time and by default the run will be canceled (output file will be empty). To force the run with any resolution < 500kb use the *-p* flag
VC/VC_SQRT/KR	Normalization to be used. KR (Knight-Ruiz) balancing is recommended.
*.hic	hic input file, containing the contact matrix
chr	The chromosome for which the principle component will be calculated (make sure that same format as in the reference is used, e.g. *chrX* or *x*, depending on the reference)
output directory	a path to the directory in which a bedpe file containing the contact domains will be saved (the name of the file is composed of the resolution followed by *_blocks.bedpe*). The bedpe file, is a text file with 16 fields in each row. For more details on the file format, see the *table* below
BP	resolution type (FRAG is another resolution type, this option is not covered in this documentation)
Resolution	Which resolution to be used for the calculation (integer). Since the goal here is to identify high level compartments 1M (100000) or 500k (500000) are typically sufficient
Output file	Name of text file for recording the calculated values. The output is a list of eigenvalues recorede for each bin, sorted by order on the chromosome. E.g. if the chromosome lenght 43Mb and a bin size of 1000000 was used, the output will be a list of 43 eigenvalues in text format.

Command:

```
java -jar -Xmx<memory>  -Djava.awt.headless=true -jar <path_to_juicer_tools.jar>␣
↪eigenvector <normalization_type> <*.hic> <chromosome> <resolution type> <resolution>
↪<output file>
```

Example 1 (resolution >= 500kb):

```
java -jar -Xmx48000m  -Djava.awt.headless=true -jar ./Omni-C/juicer_tools.jar␣
↪eigenvector KR contact_map.hic chr1 BP 1000000 PC_chr1.txt
```

Example 2 (resolution <= 500kb):

```
java -jar -Xmx48000m  -Djava.awt.headless=true -jar ./Omni-C/juicer_tools.jar␣
↪eigenvector -p KR contact_map.hic chr1 BP 100000 PC_chr1.txt
```

As described in the table above, the output is a one columnn list of the eigenvalues as calculated for each one of the bins. For most of the common applications this format is not useful and we will guide you on how to convert it to more useful formats:

bedGraph format

To generate a bedgraph which details each bin interval followed by the associated eigenvalue you will generate a list of the bins intervals using the `bedtools makewindows` command and combine it with the output file of the previous step (juicer tools `eigenvector`).

Specify the *chromosome of interest*, *first bp of the chromosome*, *last bp of the chromosome* (you can find this information in the *.genome file*) in a tab delmitied format (as in a bed file format) and feed it to `bedtools makewindows` alongside the window size (same as the resolution used for juicer tools `eigenvector`). Finally, combine the two files using the `paste` command:

Command:

```
echo -e '<chromosome>\t<start position>\t<end position>'|bedtools makewindows -b stdin -
↪w <window>|paste - <eigenvector output file> > <output.bedgraph>
```

Example:

```
echo -e 'chr1\t0\t248956422'|bedtools makewindows -b stdin -w 1000000|paste - PC_chr1.
↪txt > PC_chr1.bedgraph
```

A/B compartments

Active regions in the genome were observed to have similar eigenvalues, the same is true for inactive regions. To identify the distinct compartments all the neighboring bins with same sign (>0 or <0) are merged into one region. In each chromosome all the regions with the same sign are classified as the same compartment type (A/B). Please note that the signs of PC values are arbitrary, to accurately assign compartment type to signs of PC values, you need knowledge beyond Omni-C data such as RNA-Seq, Methylation states, etc`

We will use the `bedGraph` file for generating a `bedpe` file containing the borders of the calculated A/B compartments:

`bedtools` allows merging of overlapping or neighboring (`-d` flag) intervals based on strandness (`-s` flag). We will use this utility by assigning + to all intervals with PC1 values >0 and - to all intervals with PC1 values <0 (A/B compartments has nothing to do with strandness, the + and - assignments are only from practical reasons for easy manupulation of the bed file). The strandness option assumes the strand information is on the 6th column, we will use the `awk` command to record + or - on the 6th column based on the PC values. Adjacent intervals with the same sign are merged together using `bedtools merge` Following merging, we will use the `awk` command to genetrate the final bedpe format.

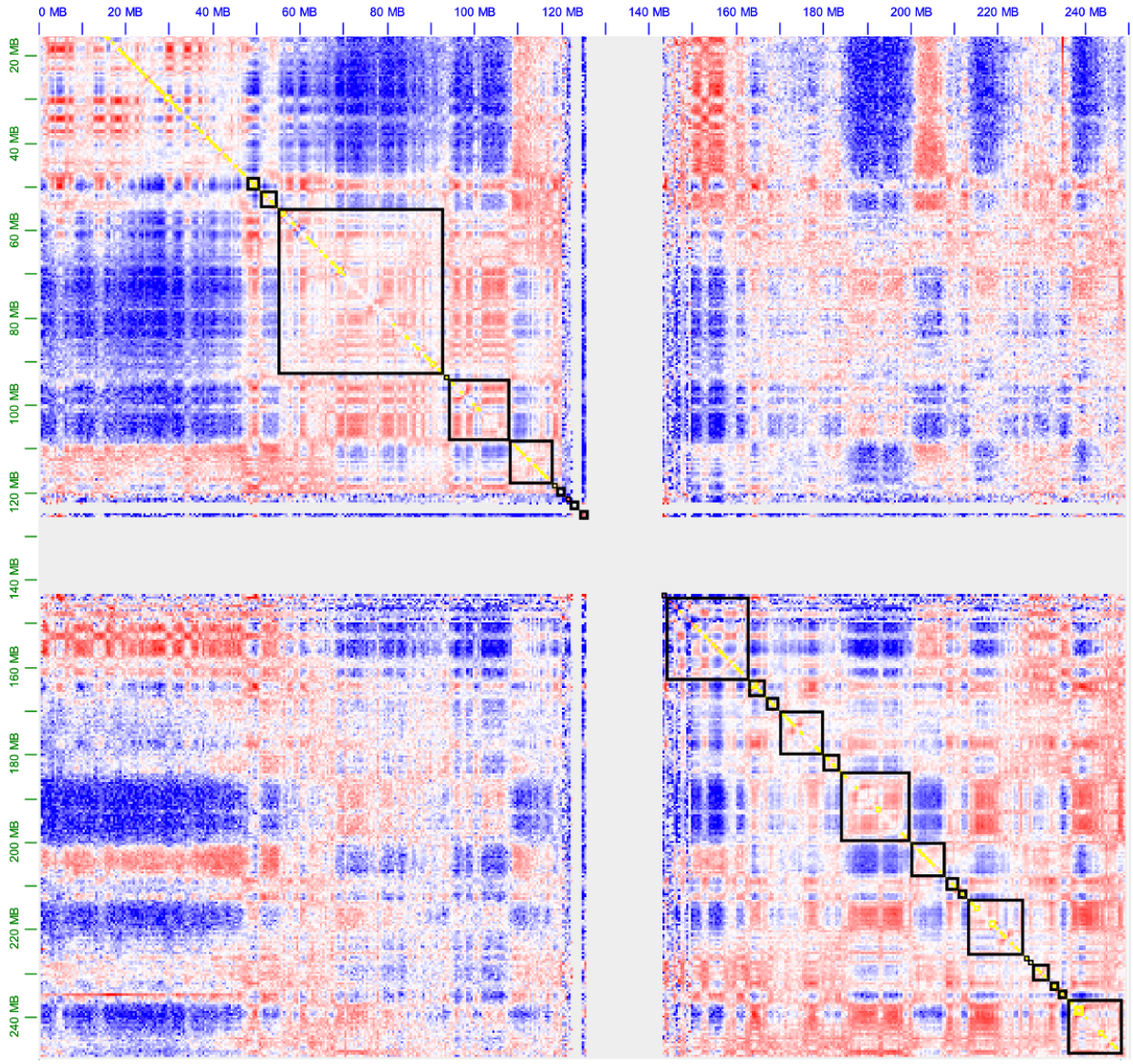
Command:

```
awk '{if ($4 > 0) print $1"\t"$2"\t"$3"\t"."\t"."\t"+"; else if ($4 < 0) print $1"\t
↪"$2"\t"$3"\t"."\t"."\t"-"}' <*.bedgraph>|bedtools merge -s -d 10  -c 6 -o␣
↪distinct|awk '{print $1"\t"$2"\t"$3"\t"$1"\t"$2"\t"$3"\t"$4}'> <output.bedpe>
```

Example:

```
awk '{if ($4 > 0) print $1"\t"$2"\t"$3"\t"."\t"."\t"+"; else if ($4 < 0) print $1"\t
↪"$2"\t"$3"\t"."\t"."\t"-"}' PC_chr1.bedgraph|bedtools merge -s -d 10  -c 6 -o␣
↪distinct|awk '{print $1"\t"$2"\t"$3"\t"$1"\t"$2"\t"$3"\t"$4}'>chr1_AB.bedpe
```

The figure below shows the observed/expected matrix of chr1 with the A/B compartments (black boxes) as generated with the commands above (visualized using `Juicebox`)



1.8 Assembly Enhancement

1.8.1 Additional Dependencies

cutadapt - sudo apt install cutadapt

meryl

merqury

scaffolding

In preparation

Kmer analysis, QV and completness

In this section we will not use the long distance infomation that is captured by the Omni-C proximity ligation libraries. Instead, we take advandage of the uniform coverage of Omni-C libraries and demonstrate how, after trimming, the reads can be used just as shotgun sequencing, for kmer analysis, measuring assembly completness and QV.

In short, kmer composition of the assembly and the reads will be used to evaluate the completness of the assembly and bp quality.

1.8.2 Trimming

Before we build a kmer database from the Omni-C fastq files, we first need to remove the bridge sequecnce. We use *cutadapt* to trim the bridge sequence from the reads.

Option	Function
-j	Number of threads (integer)
-b	For specifiying a 5' or 3' adapters for trimming. When trimming paired-end reads this option is used for R1
-B	For specifiying a 5' or 3' adapters for trimming of the reverse read. When trimming paired-end reads this option is used for R2
-o	Output file, when trimming paired-end reads this option is used for output of R1
-p	R2 output file, in paired end reads, the second read in a pair is always written to the file specified by this option
*R1.fastq or *R1.fastq.gz	R1 input file
*R2.fastq or *R2.fastq.gz	R2 input file

Command:

```
cutadapt -j <cores> -b <bridge sequence> -B <bridge sequence> -o <trimmed_output_R1.
↪fastq> -p <trimmed_output_R2.fastq> <input_R1.fastq> <input_R2.fastq>
```

Example:

```
cutadapt -j 16 -b GGTTCGTCCA -B GGTTCGTCCA -o trim_OmniC_800M_R1.fastq -p trim_OmniC_
↪800M_R2.fastq OmniC_800M_R1.fastq OmniC_800M_R2.fastq
```

1.8.3 Generate kmer databases

If you don't know the optimal kmer size for your genome, run the `best_k.sh` script from the merqury repository

Command:

```
./best_k.sh <genome_size>
```

Example:

```
./best_k.sh 3100000000
```

In this example, human genome size the optimal kmer size is 21.

Next, use the `count` utility of `meryl` tool to generate a meryl database from each one of the fastq files

Command:

```
meryl k=<k size> count output <output name> <input file>
```

Example:

```
meryl k=21 count output R1_21.meryl trim_OmniC_800M_R1.fastq
meryl k=21 count output R2_21.meryl trim_OmniC_800M_R2.fastq
```

The output from the example above is two directories: R1_21.meryl and R2_21.meryl, each one containing kmer database generated based on one fastq file. For downstream steps, all the kmer databases need to be merged into one database using the `union-sum` utility of `meryl`.

Command:

```
meryl union-sum output <output name> <path to inputs>
```

Example:

```
meryl union-sum output reads_21.meryl *meryl
```

1.8.4 Evaluate assembly completness and QV

Merqury was developed to allow reference free assembly evaluation, based on k-mer spectrum of low error rate reads (Omni-C in this case). For more details see merqury documentation . Merqury accept as an input the assembly of interest and meryl kmer database and outputs information on reads and assembly spectrum, assembly kmer completness, and assembly base level QV estimation.

Command:

```
merqury.sh <kmer DB> <assembly> <output prefix>
```

Example:

```
merqury.sh reads_21.meryl asm.fasta merqury_out
```

Detailed describtion of the outputs can be found here. Here are some highlights, the example below is using OmniC library of human sample HG002:

completeness.stats - details the assembly name (column #1), assembly k-mers used in the analysis (column #3), reads k-mers used in the analysis (column #4) and % kmer completness in the assembly (column #5).

Example:

```
HG002.pri.ctg        all     2243267808      2305938845      97.2822
```

K-mers that are found only in the assembly are assumed to be bp errors, the <output prefix>.qv summarize the QV results across the assembly. An additional file, <output prefix><assembly>.qv details QV values for each scaffold in the assembky. Assembly summary QV file include the following details: assembly name (column #1), assembly unique kmers (column #2), kmers shared in assembly and reads (column #3), QV (column #4), Error rate (column #5). See example below:

Example:

```
HG002.ctg    726218  3063065775      49.4726 1.12912e-05
```

1.9 Omni-C Data Sets

To download one of the data sets, simply use the wget command:

```
wget https://s3.amazonaws.com/dovetail.pub/HiC/fastqs/OmniC_2M_R1.fastq
wget https://s3.amazonaws.com/dovetail.pub/HiC/fastqs/OmniC_2M_R2.fastq
```

For testing purposes, we recommend using the 2M reads data sets, for any other purpose we recommend using the 800M reads data set.

Library	Link
GM12878 Omni-C 2M	• https://s3.amazonaws.com/dovetail.pub/HiC/fastqs/OmniC_2M_R1.fastq • https://s3.amazonaws.com/dovetail.pub/HiC/fastqs/OmniC_2M_R2.fastq
GM12878 Omni-C 100M	• https://s3.amazonaws.com/dovetail.pub/HiC/fastqs/OmniC_100M_R1.fastq • https://s3.amazonaws.com/dovetail.pub/HiC/fastqs/OmniC_100M_R2.fastq
GM12878 Omni-C 200M	• https://s3.amazonaws.com/dovetail.pub/HiC/fastqs/OmniC_200M_R1.fastq • https://s3.amazonaws.com/dovetail.pub/HiC/fastqs/OmniC_200M_R2.fastq
GM12878 Omni-C 400M	• https://s3.amazonaws.com/dovetail.pub/HiC/fastqs/OmniC_400M_R1.fastq • https://s3.amazonaws.com/dovetail.pub/HiC/fastqs/OmniC_400M_R2.fastq
GM12878 Omni-C 800M	• https://s3.amazonaws.com/dovetail.pub/HiC/fastqs/OmniC_800M_R1.fastq • https://s3.amazonaws.com/dovetail.pub/HiC/fastqs/OmniC_800M_R2.fastq

1.10 Support

For help or questions related please open a new issue on the github repository or send an email to: support@dovetail-genomics.com

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search