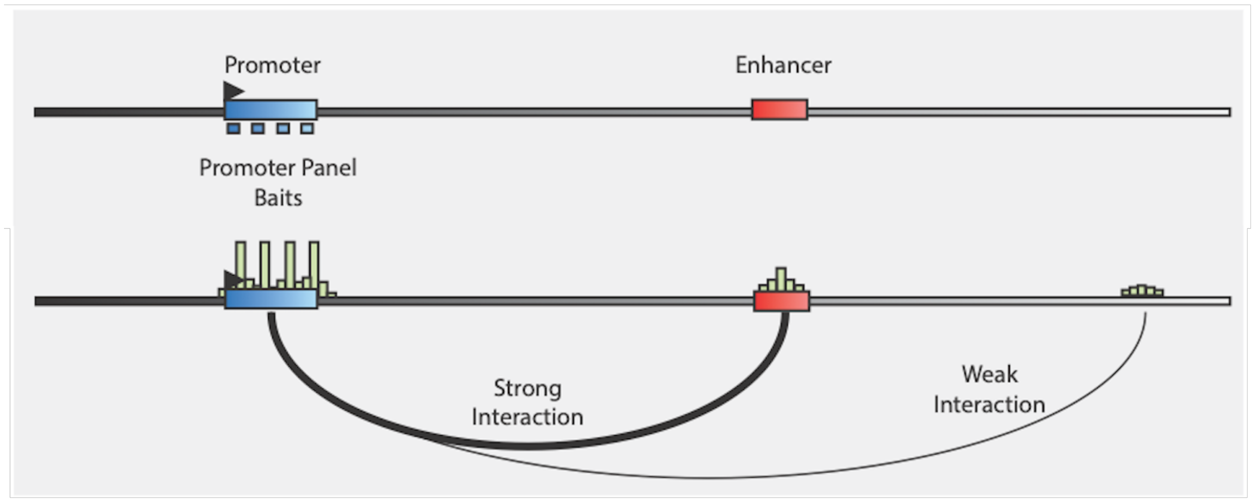
Capture Documentation

Release 0.1

Dovetail Genomics

Aug 16, 2023

CONTENTS:


Promoter
Enhancer
Promoter Panel Baits
Strong Interaction
Weak Interaction

CHAPTER

ONE

OVERVIEW

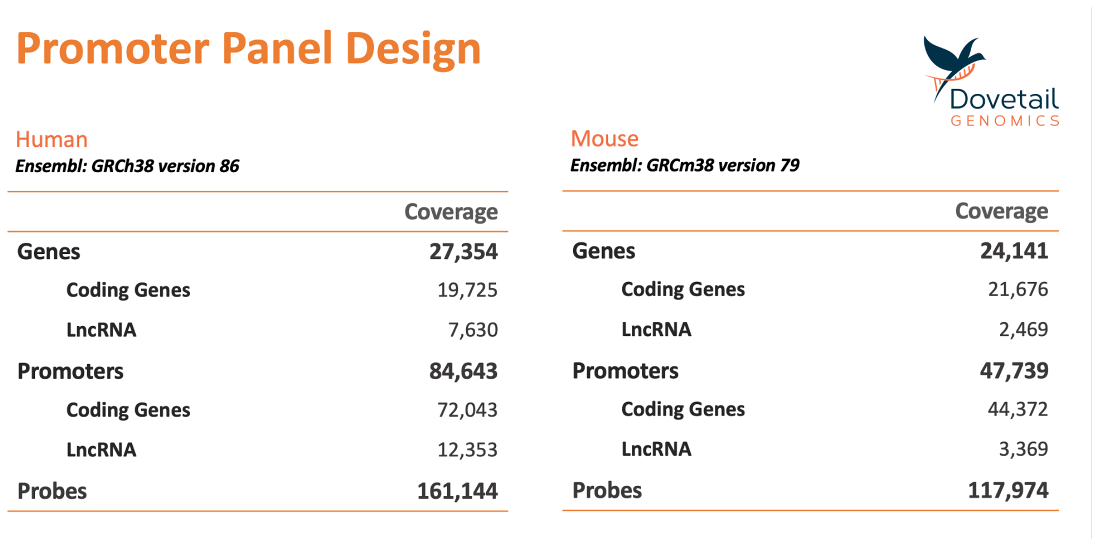
The Dovetail® Pan Promoter Enrichment Panels were designed to include all known coding genes and lncRNA promoters in the human and mouse genomes, respectively. The panels are to be used in conjugation with either Dovetail Omni-C® or Dovetail Micro-C® libraries for the detection of long-range interactions between gene promoters and their various distal-regulatory elements.

Spatial Control of Transcriptional Regulation:

- Transcriptional regulation relies on interactions between protein complexes bound at the transcriptional start site (TSS) and at regulatory sites, including enhancers.
- Promoter and enhancer sites can be separated by kilobases (or even megabases) of intervening sequence. Understanding these interactions is key to unravelling gene regulation and related human disease.
- Promoter-enhancer (P-E) interactions cannot be detected by standard NGS approaches. For example, transcription factor binding to promoters or enhancers sequences is detectable using ChIP-seq maps protein binding to promoter or enhancer this approach is blind to any higher order P-E interactions that may be mediated by the bound transcription factors. Another established method, ATAC-Seq, provides information about nucleosome-free regions but does not offer information about P, E binding or P-E interactions. Lastly, Hi-C sequencing approaches can be used for chromatin loop calling but require a high sequencing depth and offer limited resolution, requiring about 1B reads for the human genome.

Key benefits of Dovetail® Pan Promoter Enrichment Panel:

- The panel was optimized to work using Dovetail's unique restriction enzyme-free Hi-C, Omni-C and Micro-C technologies. Extemely high promoter coverage was achieved using this approach as no promoters were ommited from the panel due to lack of restriction sites in their vicinity as needed in other Capture Hi-C methods. The table below summarizes the genes, promoters and probes that are included in the human and mouse panels:



Promoter Panel Design

Human
Ensembl: GRCh38 version 86

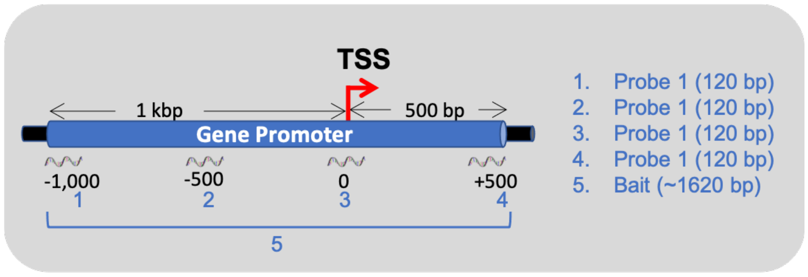
	Coverage
Genes	**27,354**
Coding Genes	19,725
LncRNA	7,630
Promoters	**84,643**
Coding Genes	72,043
LncRNA	12,353
Probes	**161,144**

Mouse
Ensembl: GRCm38 version 79

	Coverage
Genes	**24,141**
Coding Genes	21,676
LncRNA	2,469
Promoters	**47,739**
Coding Genes	44,372
LncRNA	3,369
Probes	**117,974**

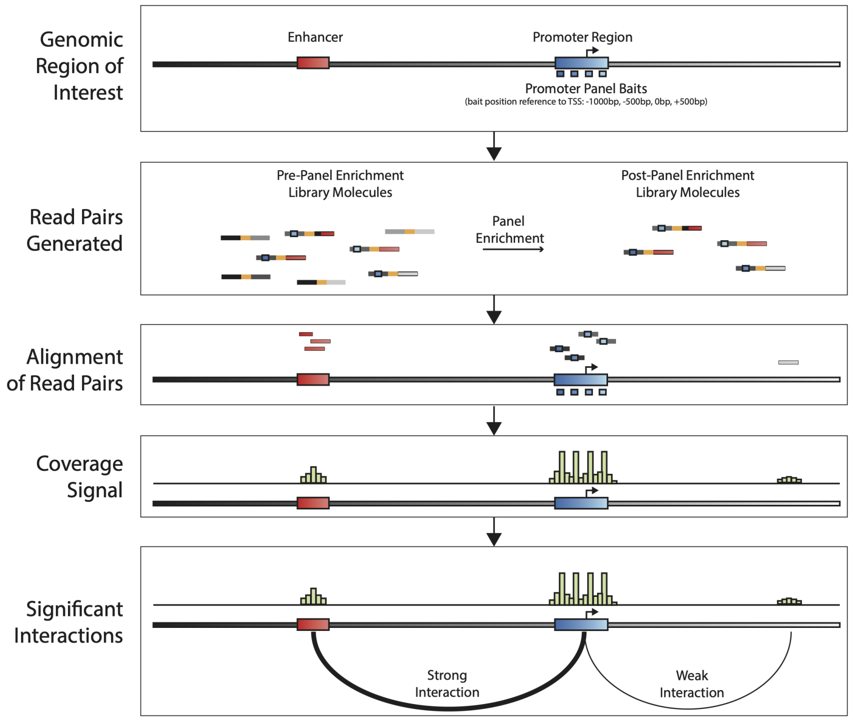
Dovetail GENOMICS

- The Pan Promoter Enrichment Panel offers the highest resolution view of P-E interactions with lowest sequencing burden.
- Captured Libraries maintain Hi-C characteristics in terms of valid read-pairs length distribution while achieving uniform coverage over baited regions.
- The reproducibility of the captured libraries is very high, with high correlation of coverage over baits between replicas, with typical $R^2 > 0.97$. Baits are defined as the combined locations of probe targets that belong to same gene promoter, as area #5 in the image below:



- The robust coverage of the human and mouse panels enables genome-wide research in the field of gene regulation and epigenetics, developmental biology, complex trait mapping and more.

- This guide will take you step-by-step through the *QC* and interpretation of your captured library and evaluation of *reproducibility* between replicas. We will show you how to:
  - Generate *contact maps*
  - *Find* and prioritize significant interactions
  - Combine this information with additional data sets that you might have available.
- Main steps in Pan Promoter captured library preparation and data processing:
- If you have not yet sequenced a Dovetail captured library and simply want to get familiarize yourself with the data, an example dataset from iPSC and NSC captured Micro-C libraries can be downloaded from our publicly-available *repository*. In these *data sets* you will find bed files with the probes and bait locations.
- If this is your first time following this tutorial, please check the *Before you begin page* first.


Genomic Region of Interest
Enhancer
Promoter Region
Promoter Panel Baits
(bait position reference to TSS: -1000bp, -500bp, 0bp, +500bp)
Read Pairs Generated
Pre-Panel Enrichment Library Molecules
Panel Enrichment
Post-Panel Enrichment Library Molecules
Alignment of Read Pairs
Coverage Signal
Significant Interactions
Strong Interaction
Weak Interaction

1.1 Before you begin

1.1.1 Have a copy of the Pan Promoter Enrichment Panel capture scripts on your machine:

Clone this repository:

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

1.1.2 Dependencies

Make sure that the following dependencies are installed:

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

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 installing all dependencies from scratch, you can use the `installDep.sh` script in this repository for updating your instance including python3. This process will take approximately 10 minutes and again requires sudo privileges to complete. 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 ./capture/installDep.sh
```

And then run the installation script:

```
./capture/installDep.sh
```

Remember!

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

1.1.3 Input files

For this tutorial you will need:

- **Fastq files** R1 and R2 (either fastq or fastq.gz are acceptable)
- **Your genome reference in a fasta file format**, e.g. hg38
- **Coordinates of targeted sequences** in bed format (provided in the *capture Data Sets section*).

Tip: If you don't already have your own input files or want to run a test, you can download sample fastq files from the *Pan Promoter Enrichment Panel Data Sets section*. Links to list of probes, baits and reference genomes are also provided, enabling you to reproduce the results presented in this tutorial.

The datasets include: two replicas of human induced pleuipotent stem cells (iPSC rep1 and iPSC rep2) and two replicas of neuronal stem cells (NSC), derived from the same iPSC cells (NSC rep1 and NSC rep 2).

The NSC rep1 dataset is used as the main example dataset throughout this tutorial (*From fastq to bam files*, *QC section* etc.), NSC rep1 and NSC rep2 are used in the *reproducibility section*, and results from both NSC replicas and iPSC replicas are used for demonstrating *interaction calling* and identifying differential interactions.

1.2 Pre-Alignment

For downstream steps you will need a genome file. A genome file is a tab delimited file with chromosome names and their respective sizes. If you do not 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 in 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 4DN project guidelines and from our own experience, optimal alignment results are obtained with the 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
```

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

To avoid memory issues, some of the steps require writing temporary files into a temp folder. Create a temp folder and remember its full path. Temp files may take up to 3x the space your final fastq.gz files. That is, if the total volume of the fastq files are 5Gb, make sure that the temp folder can store at least 15 Gb.

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 bam files

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 the details, this is a shorter version for you. Otherwise move to the next section *fastq to final valid pairs bam file - step by step*. The piped commands output two different formats of final bam files - bam index file and a dup stats file. The example below is based on the human NSC dataset, replica 1. You can find the fastq files and a link to the reference in the *capture Data Sets section*.

Command:

```
bwa mem -5SP -T0 -t<cores> <ref.fa> <capture.R1.fastq.gz> <capture.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>;samtools view -
↪@ <threads> -Shu -F 2048 <mapped.PT.bam>|samtools sort -n -T <path_to_temp_dir> --
↪threads <threads> -o <chicago.bam> -
```

Example:

```
bwa mem -5SP -T0 -t16 hg38.fa NSC_rep1_R1.fastq.gz NSC_rep1_R2.fastq.gz| 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 NSC_rep1_PT.bam;samtools index␣
↪NSC_rep1_PT.bam;samtools view -@ 16 -Shu -F 2048 NSC_rep1_PT.bam|samtools sort -n -T /
↪home/ubuntu/ebs/temp/ --threads 16 -o NSC_rep1_chicago.bam -
```

The full pipeline, with 250M read pairs on an Ubuntu 18.04 machine with 16 CPUs, 1 TB storage and 64 GB memory takes about 8 hours to complete.

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 captured Micro-C® or 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.

To replicate the output files generated in this workflow, please use the NSC replica 1 fastq files from our *capture Data Sets section*. For your convenience, the reference file is also included. If you are using your own fastq files, the results will be different from the example outputs displayed here.

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 70 bp to 1 Mbp
-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 is performed unless -S also in use
-T0	The T flag sets the minimum mapping quality of alignments to output. At this stage we want all the alignments to be recorded and thus T is set to 0, (this will enable us to gather full stats for 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 no 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> <capture_R1.fastq> <capture_R2.fastq> -o
↪<aligned.sam>
```

Example (one pair of fastq files):

```
bwa mem -5SP -T0 -t16 hg38.fasta NSC_rep1_R1.fastq.gz NSC_rep1_R2.fastq.gz -o  NSC_rep1_
↪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
```

Note: The bwa command will work on either fastq files or fastq.gz files

Recording valid ligation events

We use the `parse` module of the `pairtools` pipeline to find ligation junctions. 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 a `.pairsam` file (pairsam format captures SAM entries together with the Hi-C pair information). In addition, it will assign 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 ligation events resulting in three alignments (instead of two) for a read pair. However, in cases where three alignments arise from a single ligation event, pairtools parse can rescue this event. Walks-policy defines the policy for reporting un-rescuable walks. The 5unique value 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 the algorithm will, ignore the gap, and for gaps >30 bp, these will be marked as "null" alignments
nproc-in	integer, e.g. 16	Pairtools has an automatic "guesses" 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 sets the number of processes used by the auto-guess input decompressing function.If not specified the default is 3.
nproc-out	integer, e.g. 16	Pairtools automatically "guesses" the desired output file format (compressed or not compressed, based on file name extension). When needed, the output is compressed by bgzip/lz4c. The option nproc-out sets the number of processes used by the auto-guess output compressing function. If not specified the default is 8
chroms-path		Defines the path to your .genome file (e.g. hg38.genome)
*.sam		Defines the path to the 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 command (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 NSC_rep1_aligned.sam >   NSC_rep1_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	Provides a full path to a temp directory. A good rule of thumb is to have 3x the size of your fastq.gz files available for this diredtory. 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/  NSC_rep1_parsed.pairsam > NSC_
↪rep1_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 akin /home/user/ebs/temp/ is needed).

Removing 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	Creates an 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␣
↪NSC_rep1_dedup.pairsam NSC_rep1_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`). Note that `.pairsam` has two extra columns containing the alignments from which the Omni-C or Micro-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 files 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 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 NSC_rep1_mapped.pairs --output-
↪sam NSC_rep1_unsorted.bam NSC_rep1_dedup.pairsam
```

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

Generating the dedup, sorted 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	File name. Write final output to FILE rather than standard output
-T	Path to temp file. Using a temp file will help avoid memory issues

Command:

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

Example:

```
samtools sort -@16 -T /home/ubuntu/ebs/temp/ -o NSC_rep1_PT.bam NSC_rep1_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 NSC_rep1_PT.bam
```

The above steps result in multiple intermediate files. To simplify the process and avoid intermediate files, you can pipe the steps.

The **PT.bam* (PT stands for pair tools) is a key bam file that will be used for *library QC*, generating *contact maps* and more. Additional processing of the bam file will be required for *interaction calling*.

1.3.2 CHiCAGO compatible bam file

As will be discussed in the *interaction calling* section, we will use the CHiCAGO tool for calling P-E interactions. CHiCAGO is designed to work with bam files produced with HiCUP pipeline. To match the format of our bam file to that expected by CHiCAGO, we will clean the bam file of alignments not used by CHiCAGO (e.g. supplementary alignment) and modify the sorting from position based to read-name based sorting.

Samtools parameter for generating a CHiCAGO compatible bam format:

Samtools Utility	Parameter	Function
view	-@	Number of threads
view	-S	Ignored (input format is auto-detected)
view	-h	Include header in SAM output
view	-u	Output uncompressed bam file. Since this is not a final output, but piped to another samtools step the u option will save time
view	-F 2048	Remove supplementary alignments
sort	-T	Provide a full path to a temp directory. Using a temp directory will help avoid memory issues
sort	-n	Sort by read name
sort	-o	Output file
sort	-T	path to temp file

Command:

```
samtools view -@ <threads> -Shu -F 2048 <input bam file>|samtools sort -n -T <path to␣
↪temp dir> --threads <threads> -o <output bam file> -
```

Example:

```
samtools view -@ 16 -Shu -F 2048 NSC_rep1_PT.bam|samtools sort -n -T /home/ubuntu/ebs/
↪temp/temp.bam --threads 16 -o NSC_rep1_chicago.bam -
```

1.4 Library QC

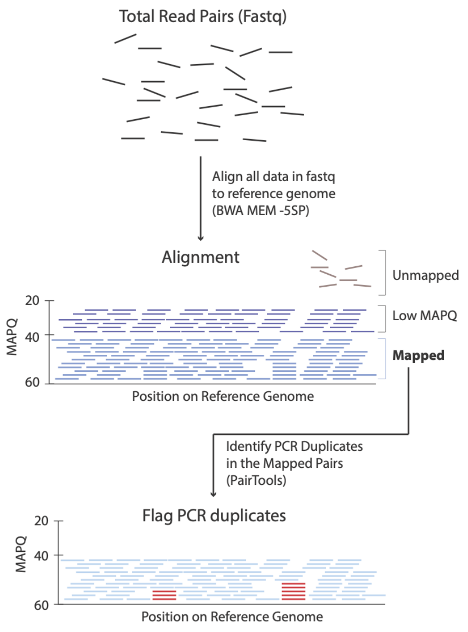
1.4.1 Proximity ligation properties

At step *Removing 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, and total pairs for each pair of chromosomes. Although you can directly use the pairtools stats file as is to get informed on the quality of the Omni-C® or Micro-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 presents them in percentage values in addition to absolute values.

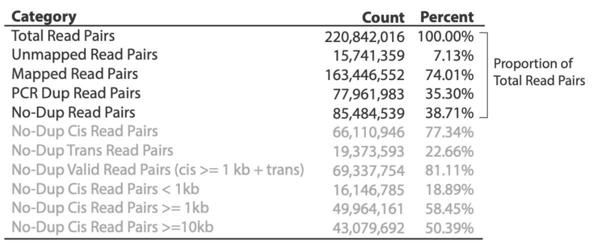
The figures below outline how the values on the QC report are calculated:

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

Process

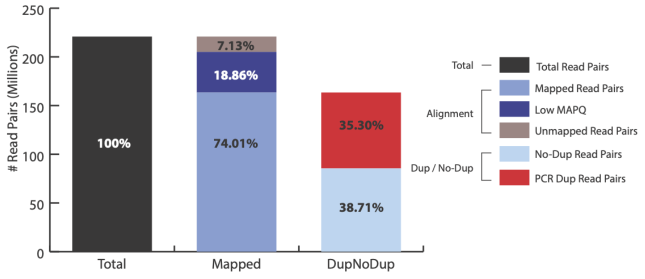

Total Read Pairs (Fastq)
Align all data in fastq to reference genome (BWA MEM -5SP)
Alignment
Unmapped
Low MAPQ
Mapped
MAPQ
20
40
60
Position on Reference Genome
Identify PCR Duplicates in the Mapped Pairs (PairTools)
Flag PCR duplicates
MAPQ
20
40
60
Position on Reference Genome

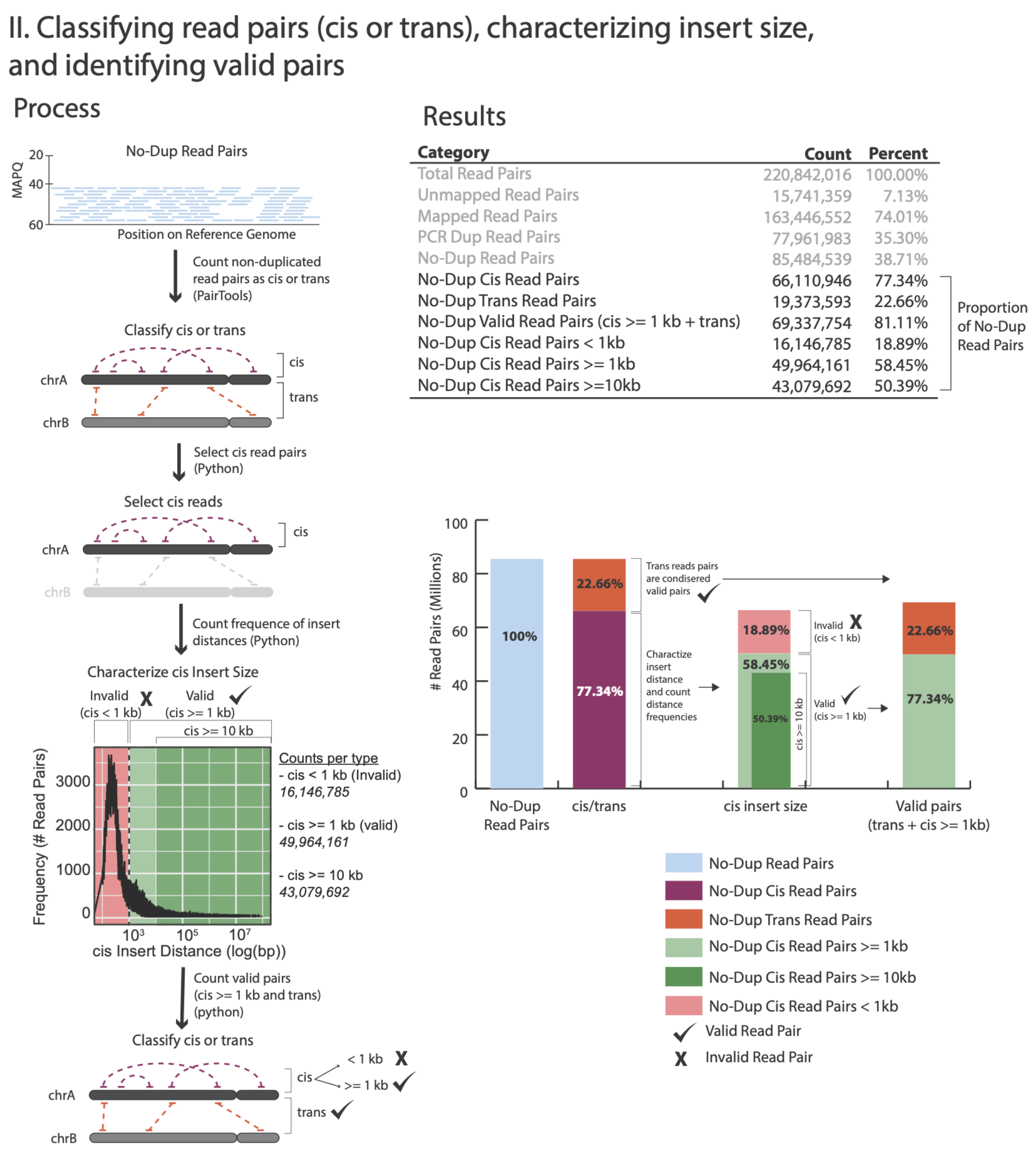
Results



Category	Count	Percent
Total Read Pairs	220,842,016	100.00%
Unmapped Read Pairs	15,741,359	7.13%
Mapped Read Pairs	163,446,552	74.01%
PCR Dup Read Pairs	77,961,983	35.30%
No-Dup Read Pairs	85,484,539	38.71%
No-Dup Cis Read Pairs	66,110,946	77.34%
No-Dup Trans Read Pairs	19,373,593	22.66%
No-Dup Valid Read Pairs (cis >= 1 kb + trans)	69,337,754	81.11%
No-Dup Cis Read Pairs < 1kb	16,146,785	18.89%
No-Dup Cis Read Pairs >= 1kb	49,964,161	58.45%
No-Dup Cis Read Pairs >=10kb	43,079,692	50.39%

Proportion of Total Read Pairs





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

Results

Category	Count	Percent
Total Read Pairs	220,842,016	100.00%
Unmapped Read Pairs	15,741,359	7.13%
Mapped Read Pairs	163,446,552	74.01%
PCR Dup Read Pairs	77,961,983	35.30%
No-Dup Read Pairs	85,484,539	38.71%
No-Dup Cis Read Pairs	66,110,946	77.34%
No-Dup Trans Read Pairs	19,373,593	22.66%
No-Dup Valid Read Pairs (cis >= 1 kb + trans)	69,337,754	81.11%
No-Dup Cis Read Pairs < 1kb	16,146,785	18.89%
No-Dup Cis Read Pairs >= 1kb	49,964,161	58.45%
No-Dup Cis Read Pairs >=10kb	43,079,692	50.39%

Proportion of No-Dup Read Pairs

Process
No-Dup Read Pairs
MAPQ
20
40
60
Position on Reference Genome
Count non-duplicated read pairs as cis or trans (PairTools)
Classify cis or trans
chrA
chrB
cis
trans
Select cis read pairs (Python)
Select cis reads
Count frequence of insert distances (Python)
Characterize cis Insert Size
Invalid (cis < 1 kb)
Valid (cis >= 1 kb)
cis >= 10 kb
Frequency (# Read Pairs)
3000
2000
1000
0
10³
10⁵
10⁷
cis Insert Distance (log(bp))
Counts per type
- cis < 1 kb (Invalid) 16,146,785
- cis >= 1 kb (valid) 49,964,161
- cis >= 10 kb 43,079,692
Count valid pairs (cis >= 1 kb and trans) (python)
Classify cis or trans
< 1 kb
>= 1 kb
# Read Pairs (Millions)
100
80
60
40
20
0
100%
22.66%
77.34%
Trans reads pairs are condisered valid pairs
Charactize insert distance and count distance frequencies
18.89%
58.45%
50.39%
cis >= 10 kb
Invalid (cis < 1 kb)
Valid (cis >= 1 kb)
22.66%
77.34%
No-Dup Read Pairs
cis/trans
cis insert size
Valid pairs (trans + cis >= 1kb)
No-Dup Read Pairs
No-Dup Cis Read Pairs
No-Dup Trans Read Pairs
No-Dup Cis Read Pairs >= 1kb
No-Dup Cis Read Pairs >= 10kb
No-Dup Cis Read Pairs < 1kb
Valid Read Pair
Invalid Read Pair

Command:

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

Example:

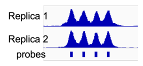
```
python3 ./capture/get_qc.py -p NSC_rep1_stats.txt
```

After the script completes, it will output (if you ran the NSC rep1 sample you should expect to generate these same results):

```
Total Read Pairs                              253,341,035  100%
Unmapped Read Pairs                           14,393,599   5.68%
Mapped Read Pairs                             195,566,613  77.2%
PCR Dup Read Pairs                            51,471,702   20.32%
No-Dup Read Pairs                             144,094,911  56.88%
No-Dup Cis Read Pairs                         94,499,595   65.58%
No-Dup Trans Read Pairs                       49,595,316   34.42%
No-Dup Valid Read Pairs (cis >= 1kb + trans)  125,160,568  86.86%
No-Dup Cis Read Pairs < 1kb                   18,934,343   13.14%
No-Dup Cis Read Pairs >= 1kb                  75,565,252   52.44%
No-Dup Cis Read Pairs >= 10kb                 46,482,376   32.26%
```

- For each library, we recommend a minimum of 150 M **Total Read Pairs**
- Typically, **PCR Dup Read Pairs** range from 10% up to 35%
- **No-Dup Trans Read Pairs** are typically around 20% to 30% but lower or higher values alone are not reflective of poor library quality (e.g. 12%-35% are often observed with good quality libraries)
- **No-Dup Cis Read Pairs >= 1kb** is typically between 50% to 65%

1.4.2 Target enrichment QC



To evaluate the level of target enrichment we will compare the coverage over the probes (targeted regions) and the overall fraction of the captured reads in our library.

On-target rate

We define the on-target rate as the percentage of read pairs that map to targeted regions. The on-target rate uses read pairs and not reads since due to proximity ligation, half of the read-pair is expected to map elsewhere in the genome, such that only one read from the pair maps to the targeted region.

Since DNA fragments can extend beyond the sequenced region and as little as a 30 bp match is sufficient for capture, we treat reads that fall in close proximity to the targeted regions (up to 200 bp upstream or downstream from a probe) as on-target reads.

To count the number of on target pairs we will use:

- **PT.bam* file that you generated in the *previous section* (not the CHiCAGO compatible bam)
- Bed file with the probe positions with 200 on both sides (the files h_probes_200bp_v1.0.bed for human and m_probes_200bp_v1.0.bed for mouse can be found in the *Data sets section*).

Count on-target read pairs:

Command:

```
samtools view <*PT.bam file> -L <padded bed file> -@ <threads> \
|awk -F "\t" '{print "@"$1}'|sort -u|wc -l
```

Example:

```
samtools view NSC_rep1.PT.bam -L h_probes_200bp_v1.0.bed -@ 16|awk -F "\t" '{print "@"$1}
↪'|sort -u|wc -l
```

Samtools view with the `-L` argument enables the extraction of only the reads that mapped to the region of interest. The awk command helps us parse the file and extract the read ID information. The sort command with a `-u` (unique) argument will remove any multiple occurrences of the same read ID (to avoid counting read1 and read2 of the same pair if both mapped to the target region). And finally, `wc -l` counts the read IDs in this list.

The example above will output the value: 93,171,111 (**On-Target Read Pairs**)

There is no need to count the total read pairs in the bam file (which represents the total number of pairs, or 100%) as it was already reported by the QC script above, labeled as **No-Dup Read Pairs** (in our example: 144,094,911).

Now you can calculate the on-target rate:

$$\frac{OnTargetReadPairs}{NoDupReadPairs} * 100$$

And in the example above:

$$\frac{93,171,111}{144,094,911} * 100 = 64.7\%$$

The **on-target rate** of the NSC replica1 example library is 64.7%. This is a typical on target rate, although occasionally lower values may be observed (as low as 40%).

Coverage depth

There are multiple methods and tools that enable the calculation of coverage depth from a bam file at different regions. We chose to use the tool mosdepth as we find it to be easy to use and relatively fast.

Use the probe bed file (and bait bed file if desired) to calculate coverage using the position sorted bam file (e.g. mapped.PT.bam. Do not use the CHiCAGO compatible bam file):

Command:

```
mosdepth -t <threads> -b <bed file> -x <output prefix> -n <bam file>
```

Example:

```
mosdepth -t 16 -b h_probes_v1.0.bed -x NSC_rep1_probes -n NSC_rep1.PT.bam
```

This command will yield multiple output files. Specifically, two files useful for QC-ing your libraries are: **a)** a bed file detailing the mean coverage per region (region being probe location, based on the input bed file), e.g. NSC_rep1_probes.regions.bed.gz and **b)** a summary output file, e.g. NSC_rep1_probes.mosdepth.summary.txt. The summary file provides information on the mean coverage of the total genome (second to last row) and mean coverage of the total_region (targeted region of interest - the last row in the summary). To print the header and two last summarizing rows, follow this example:

```
head -1 NSC_rep1_probes.mosdepth.summary.txt;tail -2 NSC_rep1_probes.mosdepth.summary.txt
```

This will output the following:

```
chrom          length      bases       mean     min   max
total          3088269832  39020721947 12.64    0     482767
total_region   19337280    7835787504  405.22   0     8129
```

In this example (NSC rep1), the mean coverage over targeted regions is 405.22, while non-targeted regions have a mean coverage depth of only 12.64. Overall, the coverage depth is 32 times higher at targeted regions vs non-targeted regions: $405.22/12.64 = 32$. The fold difference between the mean coverage depth of targeted regions and non-targeted regions is typically around 30, just as seen in this example.

The bed files with mean coverage values at on-target regions (e.g. NSC_rep1_probes.regions.bed.gz and NSC_rep2_probes.regions.bed.gz) will be used to assess *replica reproducibility*.

Running the QC steps can be completed in less than 2 hours on an Ubuntu 18.04 machine with 16 CPUs, 1TB storage and 64GB memory.

1.5 Replica Reproducibility

It is highly recommended that 2-4 replicas be generated for each condition (e.g. cell type, treatment etc.) in your experiment. Our experience shows that Pan Promoter Enrichment Panel experiments are highly reproducible and that coverage over probes and baits is highly correlated between replicas.

To calculate the R^2 value of mean coverage between replicas, you can use the the *output mosdepth bed files* (ends with regions.bed.gz) that are generated in the *QC step*.

In the *QC step* we guided you to generate a coverage profile of probe regions. When evaluating reproducibility between samples you may be interested, in addition, to evaluate the coverage reproducibility across the bait regions (in most cases one promoter is represented by one bait, or, alternatively, by 4 probes). You can simply repeat the *mosdepth command* with the bait file (e.g. https://s3.amazonaws.com/dovetail.pub/capture/human/h_baits_v1.0.bed) in place of the probe bed file.

The last column in the mosdepth output bed file (e.g. NSC_rep1_probes.regions.bed.gz) specifies the mean coverage at each bait or probe location. You can import the last column for each replica of interest to excel, R data frame, or your choice of statistical tool to calculate R^2 values. In the example below, you can find guidelines for plotting the coverage information of one replica vs. another for calculating the R^2 value.

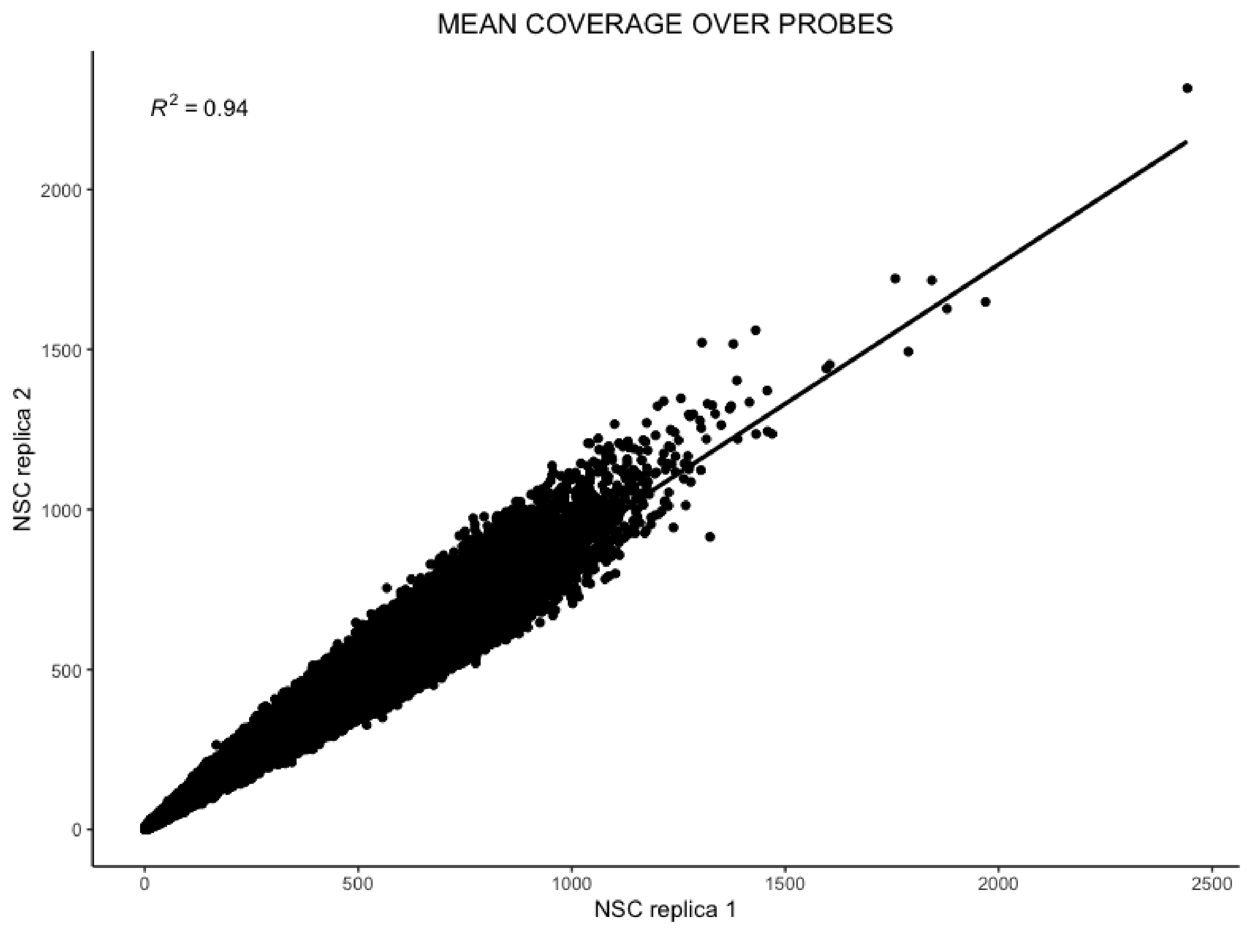
In your R console:

```
library(tidyverse)
library(ggplot) #for plotting coverage values of rep1 vs rep2
library(ggpmisc) #for adding regression values to the plot

#read coverage information of rep1 (output bed file from mosdepth step) and rename␣
↪columns
NSC_rep1_probes <- read.table(gzfile("NSC_rep1_probes.regions.bed.gz"),sep="\t",␣
↪header=FALSE)
NSC_rep1_probes <-rename(NSC_rep1_probes, chr = V1, start = V2, end = V3, probe = V4,␣
↪rep1_coverage = V5)

#read coverage information of rep2 (output bed file from mosdepth step) and rename␣
```

(continues on next page)

(continued from previous page)

```
↪columns
NSC_rep2_probes <- read.table(gzfile("NSC_rep2_probes.regions.bed.gz"),sep="\t",␣
↪header=FALSE)
NSC_rep2_probes <-rename(NSC_rep2_probes, chr = V1, start = V2, end = V3, probe = V4,␣
↪rep2_coverage = V5)

#combine replicates into one data frame
df<-full_join(NSC_rep1_probes,NSC_rep2_probes)

#Plot coverage of probes of replica1 vs replica2
ggplot(df, aes(x = rep1_coverage, y = rep2_coverage)) + geom_point()

#calculate R-squared value
cor(df$rep1_coverage,df$rep2_coverage)^2

#Alternatively, you can add the regression function and the R-squared value to the graph:

ggplot(df, aes(x = rep1_coverage, y = rep2_coverage)) + geom_point() + stat_
↪smooth(method = "lm", color = "black", formula = y ~ x) + stat_poly_eq (formula = y ~␣
↪x)

#Final plot (with title and no background)

ggplot(df, aes(x = rep1_coverage, y = rep2_coverage)) + geom_point() + stat_
↪smooth(method = "lm", color = "black", formula = y ~ x) + stat_poly_eq (formula = y ~␣
↪x) + labs(title="MEAN COVERAGE OVER PROBES",x="NSC replica 1", y = "NSC replica 2") +␣
↪theme_classic() + theme(plot.title = element_text(hjust = 0.5))
```



Typically R^2 values for mean probe coverage are around 0.9 (ranging from 0.85 - 0.95) and R^2 values for mean bait coverage are around 0.95 (ranging from 0.95 - 0.99).

1.6 Detection of DNA Looping Interactions

1.6.1 Introduction

CHiCAGO (Capture Hi-C Analysis of Genomic Organisation) is the most common tool for the detection of DNA looping interactions in capture Hi-C data. The underlying assumption of the approach is that background signal is the result of Brownian noise (distance dependent with a negative binomial distribution) and Poisson distributed technical noise. Signal that is higher than the background is considered a result of significant interactions, such as in the case of P-E interactions.

Please note that the entire analysis approach that we describe here is bait based and not probe based, meaning all probes of the same promoter are pooled together and interactions called at the bait level (see the *main page* to refresh your memory on the difference between probes and baits). If you are interested in analyzing the data in a probe-centric approach you will need to adjust the steps below. This will also require much higher sequencing coverage as CHiCAGO requires, by default, a minimum of 250 reads per captured fragment.

Installing CHiCAGO package and CHiCAGO tools

Please install CHiCAGO from this forked repository (includes minor bug fixes) as follows:

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

And install the R package (and its dependencies, if not already installed; requires R version >= 3.1.2.).

In your R console:

```
install.packages("argparser")
install.packages("devtools")
library(devtools)
install_bitbucket("chicagoTeam/Chicago", subdir="Chicago")
```

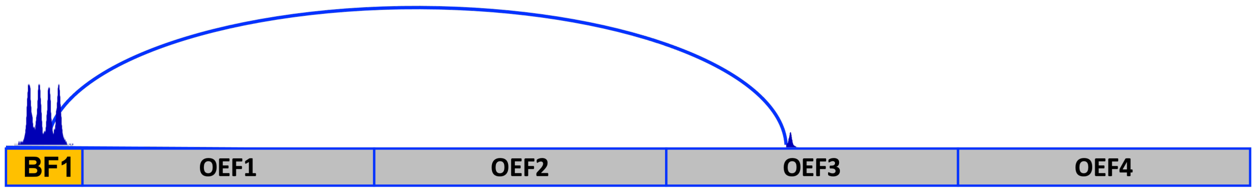
1.6.2 Input files

Genomic fragments and design directory

CHiCAGO requires as an input two sets of genomic intervals:

- **Bait Fragments (BF)** - fragments overlapping baits
- **Other End Fragments (OEF)** - all other genomic fragments to be included in the analysis

In the example below, you can see the two types of fragments, BF (in orange) and OEF (in gray). The locations of the BF and OEF are required as an input (more details below). In this example, an interaction detected by CHiCAGO between BF no.1 and OEF no.3 is depicted.



Here, since RE-free methods are used (Omni-C® or Micro-C® assays), DNA can be fragmented at almost any position. To generate the **OEF** we simply bin the genome at the desired window size (e.g. 10 kb). The final **OEF** set is composed of all the binned intervals that do not intersect with the **BF** set. To generate **BF** intervals, probes of the same gene are pooled together (usually covering about 2 kb) into one fragment (and in some cases, if overlap occurs, probes of two genes may be pooled into a single fragment).

The **OEF** and **BF** are captured in .rmap and .baitmap files, and are used to generate additional **design files** that are required for CHiCAGO. For your convenience we have generated all design files needed to be used with CHiCAGO. These can be found in the *data-sets section*. If you are interested in generating the files on your own, please review the *For advanced users section* at the bottom of this document.

We recommend starting with 10 kb (size of **OEF** fragment size, labeled `h_10kDesignFiles` for human or `m_10kDesignFiles` for mouse). Design files with 5 kb, 10 kb and 20 kb are provided (*data-sets section*).

Download the desired CHiCAGO design directory and uncompress it:

```
wget https://s3.amazonaws.com/dovetail.pub/capture/human/CHiCAGO_files/h_10kDesignFiles.
↪tar.gz
tar xzvf h_10kDesignFiles.tar.gz
```

10 kb typically works well with libraries of 150 M read-pairs or more. However, if the number of reads fall below that value (or if the library is of lower quality, e.g. due to high dup rate) 20 kb may be a better choice. For better resolutions (5 kb and even 2 kb), higher sequence coverage is required (thids can be achieved by sequencing more or

by pooling together multiple replicas as will be discussed below). A good practice would be to start with 10 kb, and, after reviewing the *CHiCAGO results*, consider whether to rerun with alternative fragment size.

chinputs

The mapping information to be used as input for identifying significant interactions needs to be adjusted to the CHiCAGO format, named chinput. To generate chinput files you will need the .rmap and .baitmap from your chosen design files directory and the *CHiCAGO compatible bam file*. The bam2chicago.sh script is used to generate input files (chinput) using your choice of design files.

Create chinput files:

Important!

Make sure to use the CHiCAGO compatible bam file such as `chicago.bam` from the *From fastq to bam files section*

Command:

```
./chicago/chicagoTools/bam2chicago.sh <capture.bam> <baitmap> <rmap> <output_prefix>
```

Example:

```
./chicago//chicagoTools/bam2chicago.sh chicago_capture_NSC_rep1.bam \
./h_10kDesignFiles/pooled_10kb_120bp.baitmap \
./h_10kDesignFiles/digest10kb_pooled120bp.rmap \
10kb_chinput_NSC_rep1
```

The output following the above example is a new directory, *10kb_chinput_NSC_rep1*, with the desired chinput, *file 10kb_chinput_NSC_rep2.chinput*, that will be used in the next steps for running CHiCAGO. The output directory also includes a bait2bait*bedpe* file with pairs overlapping baits at both ends of the pair. We will not use this file in our analysis.

Generating chinput files is the most time-consuming step in the CHiCAGO pipeline, taking about 1.5 hours for a dataset with 250 M read-pairs on an Ubuntu 18.04 machine with 16 CPUs, 1TB storage and 64GB memory.

1.6.3 Interaction calling

Now that you have all the needed input files, you can run CHiCAGO to obtain a list of significant interactions. Chinputs from one replica or more can be used. As mentioned above, we recommend using the 10 kb design files for initial interaction calling. Depending on your goals and the quality and depth of your data, you may choose to experiment with other fragment sizes.

Parameter	Value	Details
--design-dir		Path to the design directory
--cutoff	5	Score threshold for calling significant interactions (typically set to 5). Scores are calculated as -log weighted p-values based on the expected true positive rates at different distance ranges (estimated from data). Increasing the cutoff will increase the stringency and less interactions will be called.
--export-format		Multiple formats are available for exporting the interactions. You can specify multiple formats to be used. We recommend the following:
	interBed	Each interaction detailed as one row with the following fields: bait_chr, bait_start, bait_end, bait_name, otherEnd_chr, otherEnd_start, otherEnd_end, otherEnd_name, N_reads, and score
	washU_text	Can be read by the WashU browser. Upload to the browser using: **Tracks** → **Local Text Tracks** → **long-range text**
	washU_track	Can be read by the WashU browser. Upload to the browser using" **Tracks** → **Local Tracks** → **longrange** (both track file and index file should be uploaded)
	seqMonk	Each interaction is represented by two rows: bait row followed by other-end row. Includes the following fields: chromosome, start, end, name, number of reads and interaction score
<path_to_chinput>		Path to chinput file(s). If you wish to run CHiCAGO on multiple replicas together, you can specify multiple chinput files (in a comma separated format). All the chinput files should be made using the same settings and with thesame rmap and baitmap files located in the specified design directory
<output_prefix>		

Command:

```
Rscript ./chicago/chicagoTools/runChicago.R --design-dir <path_to_design_dir> \
--cutoff 5 \
--export-format interBed,washU_text,seqMonk,washU_track \
<path_to_chinput> \
<output_prefix>
```

Example, one chinput file:

```
Rscript ./chicago/chicagoTools/runChicago.R --design-dir ./h_10kDesingFiles \
--cutoff 5 \
--export-format interBed,washU_text,seqMonk,washU_track \
.10kb_chinput_NSC_rep1/10kb_chinput_NSC_rep1.chinput \
NSC_rep1_chicago_calls
```

Example, chinput files from two replicas:

```
Rscript ./chicago/chicagoTools/runChicago.R --design-dir ./h_10kDesingFiles \
--cutoff 5 \
--export-format interBed,washU_text,seqMonk,washU_track \
.10kb_chinput_NSC_rep1/10kb_chinput_NSC_rep1.chinput, .10kb_chinput_NSC_rep2/10kb_
↪chinput_NSC_rep2.chinput \
NSC_chicago_calls
```

1.6.4 Output files

CHiCAGO outputs will be saved to 4 different directories:

Diagnostic plots directory (diag_plots)

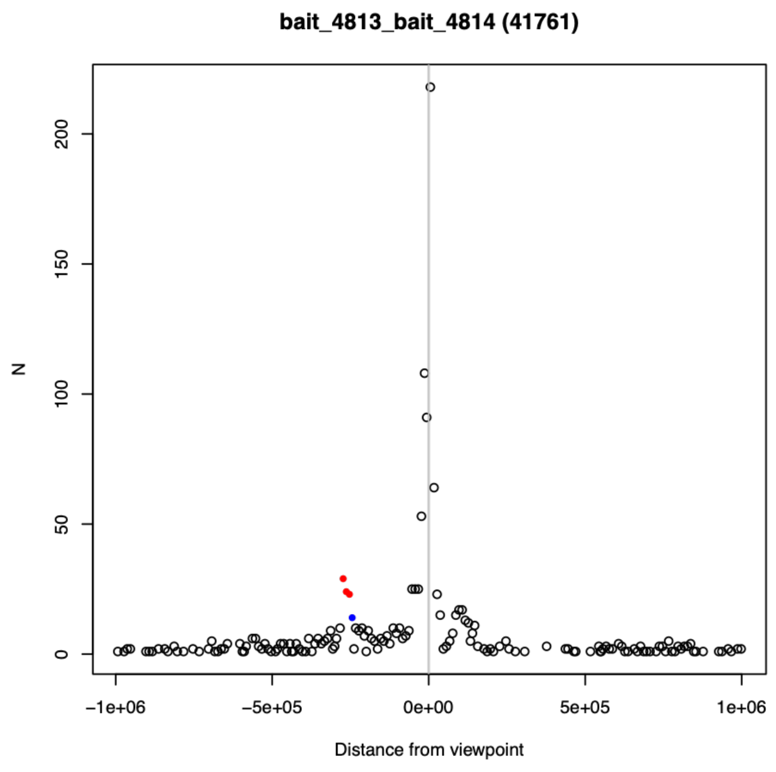
This will include 3 diagnostic plots:

- The distance plot
- Brownian factors
- Technical noise estimates

The CHiCAGO Vignette reviews these plots and offers guidance on how to interprete them. If reviewing the diagnostic plots brings you to the conclusion that the results are not ideal (e.g. the curve of the distance plot poorly matches the data points), consider generating new chinputs with 20 kb OEF (instead of the recommended 10 kb) and rerun CHiCAGO. There are also more advanced options for fine tuning CHiCAGO runs that are out of the scope of this documentation (such as re-estimating the P-value weights using the fitDistCurve.R script of chicagoTools). Please refer to the CHiCAGO publication, its bitbucket repository and the CHICAGO vignette for more details and ideas.

Examples directory (examples)

In this directory, 16 random baits are shown with their associated raw reads (up to 1 Mb from the bait). Interactions above the threshold (default 5) are shown in red, interactions with score 3-5 are shown in blue, and interactions with a score below 3 are in black (see example below). You can also choose to plot specific baits of interest (more on that in the interactions analysis section).



Data directory (data)

This is the main directory you will use for further data analysis.

The output will include all the specified export formats (`--export-format`, see *table above* and the CHiCAGO Vignette) as well as R database file (with an extension .Rds) and an interBed file (.ibed extension).

The ibed file is the main file that you will use to analyzeing the detected interactions. The ibed file contains 10 columns:

- **Columns 1-4 (chr, start, end, bait name)** define the bait side of the interaction.
- **Columns 5-8 (chr, start, end, OE name)** define the other end (OE) of the interaction. In some cases, interactions between two bait regions are called, in which case the name of the second bait will be recorded on column 8 (otherwise there is no OE name and it will be marked with `.`).
- **Column 9-10 (no. of pairs, score)** define the number of pairs that support the interaction call and the associated score. Only interactions above the cutoff are recorded in the .ibed file, but the full data, including interactions with lower scores, are saved in the .Rds database.

Tip: Filter out trans interactions, too short interactions (e.g. < 10 kb) and too long interactions (e.g. > 2 Mb). Here is a simple awk command for filtering the .ibed file:

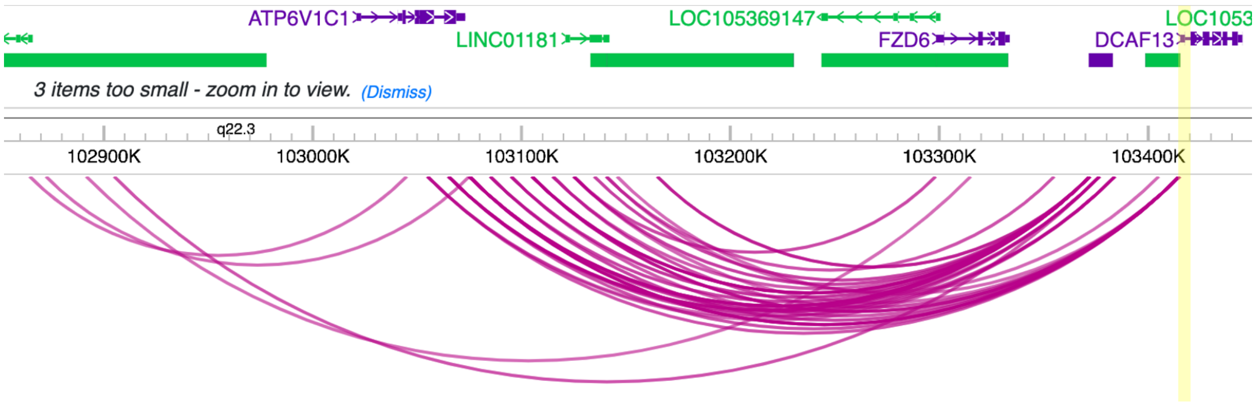
```
awk  'NR>1 {if (($1 == $5) && \
(($6 > $3 && ($6 -$3) < 2000000) || ($6 < $3 && ($2 -$7) < 2000000)) && \
(($6 > $3 && ($6 -$3) > 10000) || ($6 < $3 && ($2 -$7) > 10000))) \
print}' <interactions.ibed> >filtered_interactions.ibed
```

In the NSC rep1 example, 57,194 interactions were called with 52,411 of them passing the above filtering criteria.

As an example, let's inspect interactions that involve the DCAF13 promoter as detailed (header line was added for convenience):

```
bait_chr bait_start  bait_end    bait_name             otherEnd_chr   otherEnd_start␣
↪otherEnd_end    otherEnd_name   N_reads  score
chr8      103413593   103416224   bait_39283_bait_39284   chr8           103050000      ␣
↪103060000      .                23       6.41
chr8      103413593   103416224   bait_39283_bait_39284   chr8           103060000      ␣
↪103070000      .                24       6.62
chr8      103413593   103416224   bait_39283_bait_39284   chr8           103070000      ␣
↪103080000      .                44       13.75
chr8      103413593   103416224   bait_39283_bait_39284   chr8           103080000      ␣
↪103090000      .                27       7.34
chr8      103413593   103416224   bait_39283_bait_39284   chr8           103090000      ␣
↪103100000      .                24       6.15
chr8      103413593   103416224   bait_39283_bait_39284   chr8           103100000      ␣
↪103110000      .                39       10.99
chr8      103413593   103416224   bait_39283_bait_39284   chr8           103110000      ␣
↪103120000      .                27       6.78
chr8      103413593   103416224   bait_39283_bait_39284   chr8           103120000      ␣
↪103130000      .                23       5.32
chr8      103413593   103416224   bait_39283_bait_39284   chr8           103141449      ␣
↪103150000      .                24       5.3
```

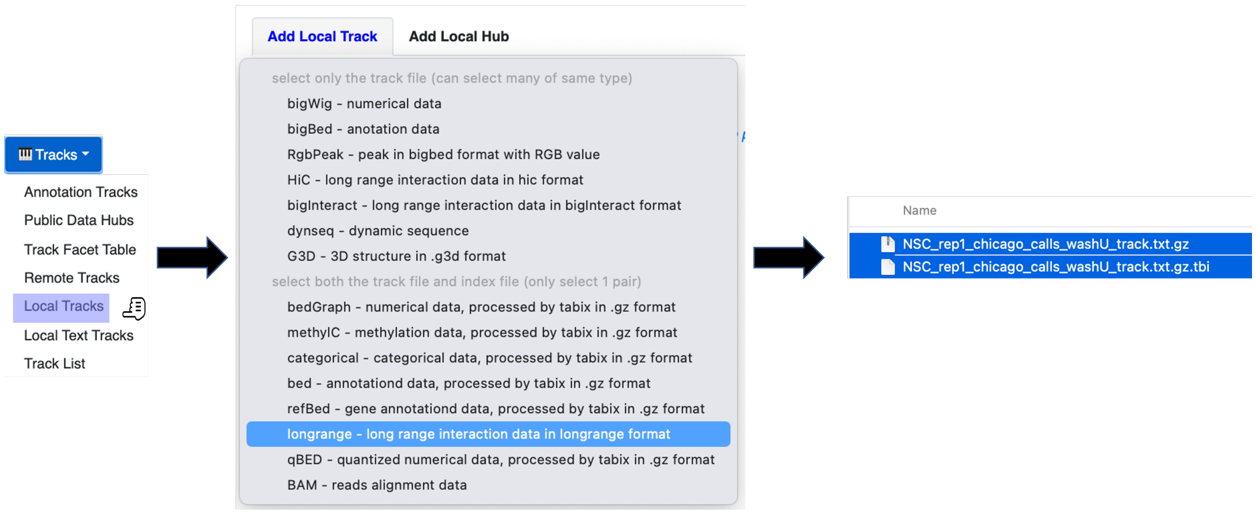
We can also visualize these interactions by uploading the track file to Wash-U browser:



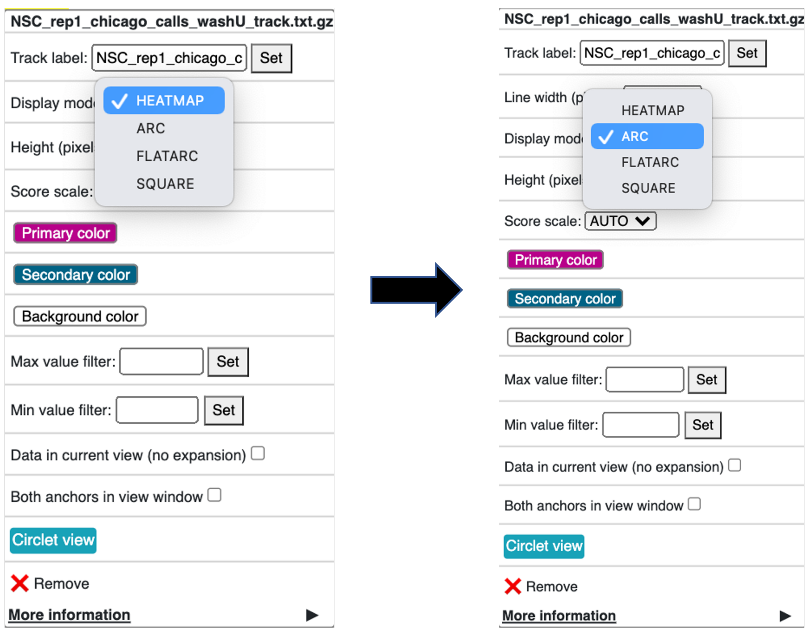
Tip: To visualize interactions on the Wash-U

1. Under the **Tracks** menue, choose the option **Local Tracks**
2. From the list of track types, choose **longrange**

3. Choose from the **data** directory two files together: the file labeled as *washU_track.txt.gz and its associated file *washU_track.txt.gz.tbi (e.g. NSC_rep1_chicago_calls_washU_track.txt.gz and NSC_rep1_chicago_calls_washU_track.txt.gz.tbi)


Tracks
Annotation Tracks
Public Data Hubs
Track Facet Table
Remote Tracks
Local Tracks
Local Text Tracks
Track List
Add Local Track
Add Local Hub
select only the track file (can select many of same type)
bigWig - numerical data
bigBed - anotation data
RgbPeak - peak in bigbed format with RGB value
HiC - long range interaction data in hic format
bigInteract - long range interaction data in bigInteract format
dynseq - dynamic sequence
G3D - 3D structure in .g3d format
select both the track file and index file (only select 1 pair)
bedGraph - numerical data, processed by tabix in .gz format
methylC - methylation data, processed by tabix in .gz format
categorical - categorical data, processed by tabix in .gz format
bed - annotationd data, processed by tabix in .gz format
refBed - gene annotationd data, processed by tabix in .gz format
longrange - long range interaction data in longrange format
qBED - quantized numerical data, processed by tabix in .gz format
BAM - reads alignment data
Name
NSC_rep1_chicago_calls_washU_track.txt.gz
NSC_rep1_chicago_calls_washU_track.txt.gz.tbi

4. Right click on the new track that was added to the genome browser
5. Change the **Display mode** from **HEATMAP** (default) to *ARC*
6. Choose your region of interest to inspect the interactions


NSC_rep1_chicago_calls_washU_track.txt.gz
Track label: NSC_rep1_chicago_c Set
Display mode ✓ HEATMAP ARC FLATARC SQUARE
Height (pixel
Score scale:
Primary color
Secondary color
Background color
Max value filter: Set
Min value filter: Set
Data in current view (no expansion)
Both anchors in view window
Circlet view
Remove
More information
NSC_rep1_chicago_calls_washU_track.txt.gz
Track label: NSC_rep1_chicago_c Set
Line width (p
Display mode HEATMAP ✓ ARC FLATARC SQUARE
Height (pixel
Score scale: AUTO
Primary color
Secondary color
Background color
Max value filter: Set
Min value filter: Set
Data in current view (no expansion)
Both anchors in view window
Circlet view
Remove
More information

Optional - Enrichment plots directory (enrichment_plots)

By default, this directory is empty. However, if you already have position information of different genomic features of interest, CHiCAGO has a built-in feature to estimate if these regions are enriched in the OE side of the detected interactions. Click here for more details.

Typically, we expect the promoter interactions to be enriched for specific genomic features, like CTCF biding sites, specific histone marks, etc. If you want to use CHiCAGO to calculate fold enrichment for genomic regions, gather the bed files (e.g. chip-seq narrow-peak files) and list the feature names and file locations in a tab delimited text file as in the example below. For demonstration, we will name this file *genomic_features.txt*.

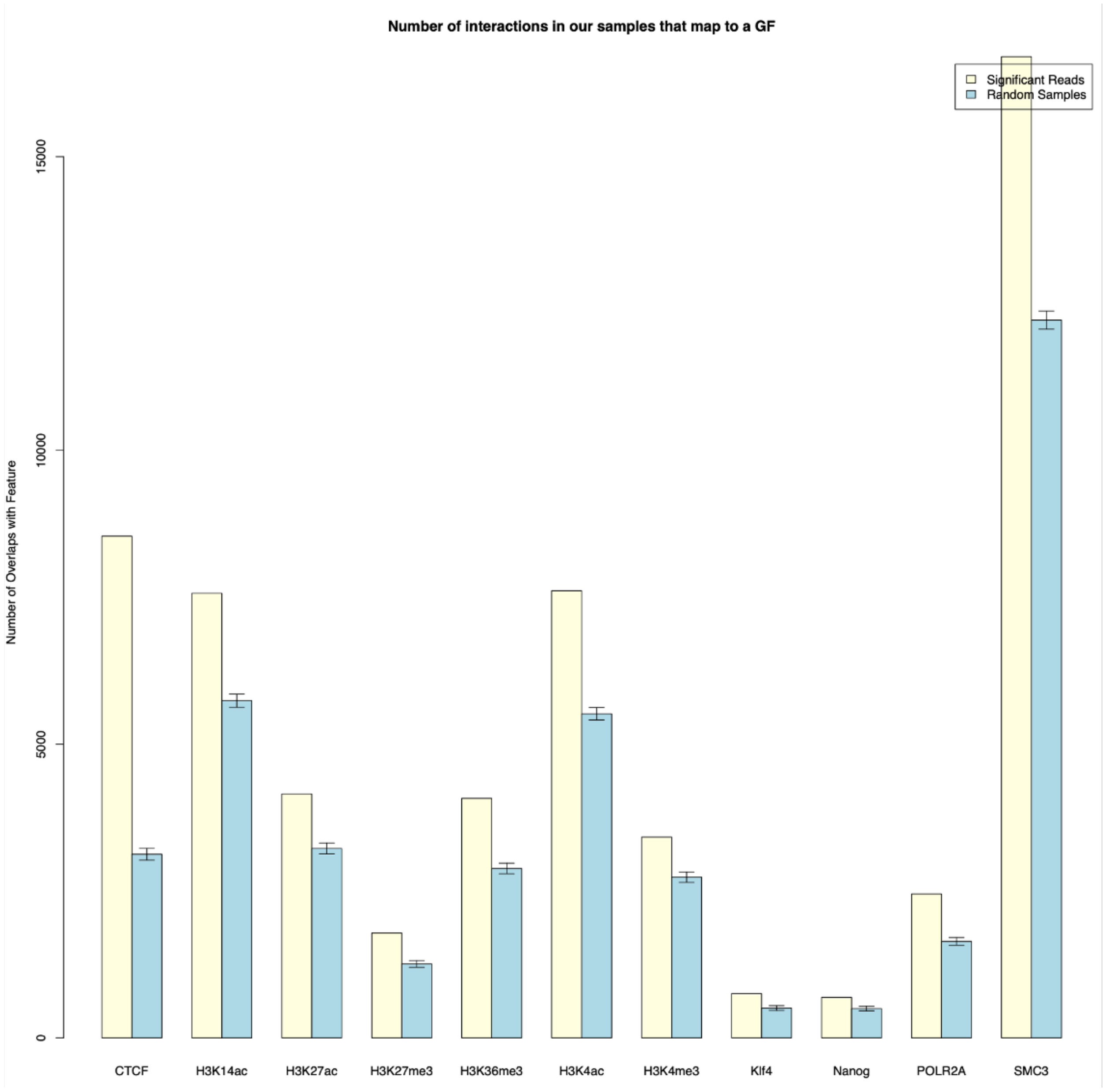
genomic_features.txt:

```
CTCF   /home/user/data/CTCF.bed
H3K4me1   /home/user/data/H3K4me1.bed
H3k27ac   /home/user/data/H3k27ac.bed
```

You will also need to specify the option `--en-feat-list` followed by a direct path to the genomic feature file location (e.g. ./genomic_features.txt) when running CHiCAGO, as in this example:

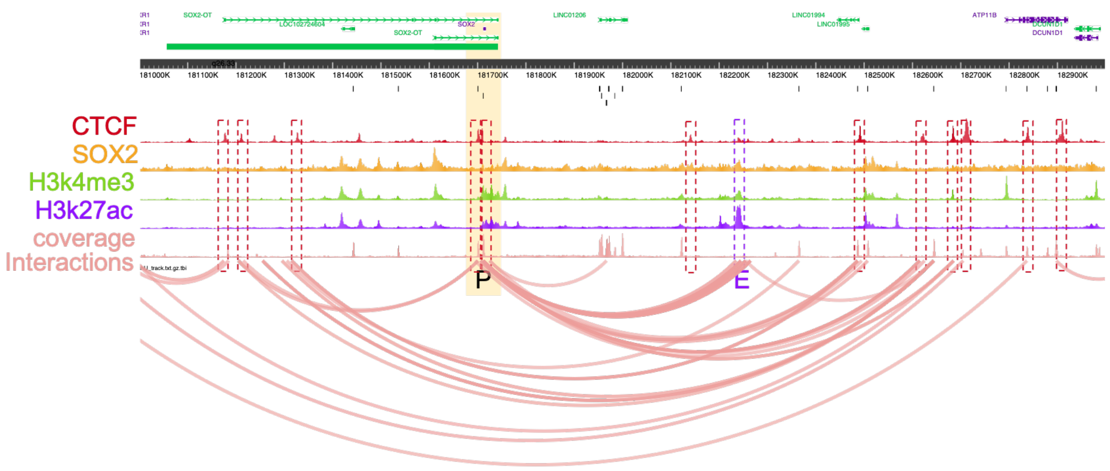
```
Rscript chicago/chicagoTools/runChicago.R --design-dir ./h_10kDesingFiles \
--cutoff 5 \
--export-format interBed,washU_text,seqMonk,washU_track \
--en-feat-list ./genomic_features.txt \
.10kb_chinput_NSC_rep1/10kb_chinput_NSC_rep1.chinput \
NSC_rep1_chicago_calls
```

Under the **enrichment_plots** directory, you will find an enrichment plot, as in the example below, showing how many OE fragments overlap with each genomic feature, and how many would have overlapped as a result of random shuffling of the genomic feature across the genome. The data that was used to generate the plot is also found in the **enrichment_plots** directory (as a .txt file).



1.6.5 Additional suggestions for interactions data analysis:

- **TADs** - Calculate how many of the interactions are within TADs vs across TADs. We anticipate that a significant majority of the interactions will be within TADs.
- **A/B compartments** - How do the interactions segregate between A/B compartments? Typically, more interactions will be observed in active regions, enriched with open chromatin. These open regions can be detected using ATAC-seq or or inferred through RNA-seq experiments.
- Overlay the information from your experiment with additional data types such as RNA-seq, Chip-seq or existing databases of regulatory elements, either for *enrichment studies* or for exploring specific promoter regions. In the example below you can see how the majority of the interactions associated with the SOX2 promoter co-occur with CTCF Chip-seq peaks:



- **GO analysis** and **motif enrichment** studies are also potential exploration paths.
- In most cases, users will have datasets from multiple sample types (e.g. different cell lines, different growth conditions, etc.) so detecting differential interactions using chicdiff,as discussed in the next section, is another approach worth exploring.

1.6.6 For advanced users

Generate your own CHiCAGO design directory

In the *Input Files section* we described the files and design directory that is required by CHiCAGO.

We have created these design libraries with pooled fragments in sizes of 5 kb, 10 kb and 20 kb and are provided in the *data-sets section*. This should be sufficient for most analyses. However, in the case that you do want to generate your own files, follow the directions in this section. Our example models **OEF** 5 kb fragments across the genome and will generate pooled **BF** (each bait can consists of multiple probes, pooled together to generate a longer fragment).

```
#Create a new directory for the design files:

mkdir h_5kDesingFiles

#Download the list of (human or mouse) baits:

wget https://s3.amazonaws.com/dovetail.pub/capture/human/h_baits_v1.0.bed

#Add 120bp on both sides of each bait, rename baits and merge overlapping baits

cut -f1,2,3 h_baits_v1.0.bed |bedtools slop -g hg38.genome  -b 120 -i stdin|\
awk -F'\t' 'NR>0{$0=$0"\t""bait_"NR} 1'|\
bedtools merge -i stdin -c 4 -o collapse -delim "_">pooled_baits120bp.bed

#5kb OEF fragments. Change -w value if you wish to change the fragment size

bedtools makewindows -g hg38.genome -w 5000 > genome.5kb.bed

#Subtract regions with probe fragments
```

(continues on next page)

(continued from previous page)

```
bedtools subtract -a genome.5kb.bed -b pooled_baits120bp.bed > 5kb_sub_probe.
↪bed

#Combine intervals

cat pooled_baits120bp.bed >temp.bed
awk '{print $1"\t"$2"\t"$3"\t""label"}' 5kb_sub_probe.bed >>temp.bed
bedtools sort -i temp.bed |awk -F'\t' 'NR>0{$0=$0"\t"NR} 1'>digest_and_probes.
↪bed

#Generate rmap:

awk '{print $1"\t"$2"\t"$3"\t"$5}' digest_and_probes.bed \
> h_5kDesingFiles/digest5kb_pooled120bp.rmap

#Generate baitmap:

awk '{if ($4 != "label") print $1"\t"$2"\t"$3"\t"$5"\t"$4}' digest_and_probes.
↪bed \
> h_5kDesingFiles/pooled120bp.baitmap

#Generate design files (adjust parameters as needed):
#Depending on the python version supported by your system,
#use either ./chicago/chicagoTools/makeDesignFiles.py or
#./chicago/chicagoTools/makeDesignFiles_py3.py

cd h_5kDesingFiles
python ../chicago/chicagoTools/makeDesignFiles.py \
--minFragLen 75 \
--maxFragLen 30000 \
--maxLBrownEst 1000000 \
--binsize 20000 \
--rmapfile ./h_5kDesingFiles/5_digest5kb_pooled120bp.rmap \
--baitmapfile ./h_5kDesingFiles/pooled120bp.baitmap --outfilePrefix␣
↪5kDesingFiles
cd ..
```

To learn more about other advanced usage of CHiCAGO please see the CHiCAGO publication, its bitbucket repository and the CHICAGO vignette

1.7 Generating Contact Matrix

There are two common formats for contact maps, the Cooler format and Hic format. To avoid large storage volumes, both are compressed, sparse formats. 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 used.

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

1.7.1 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 ./capture/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 a `.pairs` file into a HiC contact matrix. . Advantages of the HiC format include:
- `HiC` is a highly compressed binary representation of the contact matrix
- Provides rapid random access to any genomic region matrix
- Stores contact matrix at 9 different resolutions (2.5 M, 1 M, 500 K, 250 K, 100 K, 50 K, 25 K, 10 K, and 5 K)
- 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 run 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 enables 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 version of `Juicer tools`, generating contact maps directly from `.pairs` files 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:

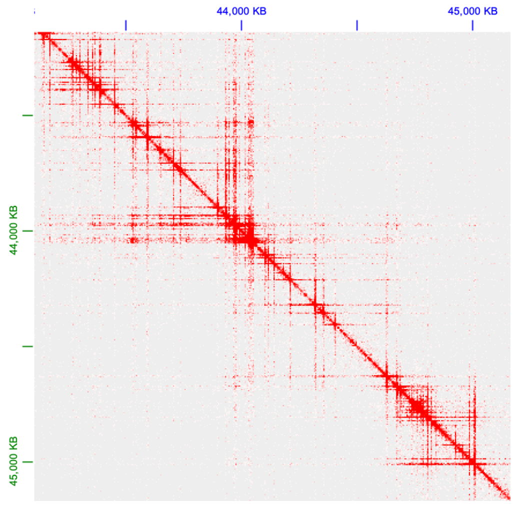
```
java -Xmx48000m  -Djava.awt.headless=true -jar ./capture/juicer_tools.jar pre --threads␣
↪16 mapped.pairs contact_map.hic hg38.genome
```

Tip no.2

`Juicer tools` offers additional functions that were not discussed here, including matrix normalization and generating a 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.



1.7.2 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 you modify the following example with the path to your *pairix* installation folder. If you are not sure of your 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` files into cool and mcool contact matrices
- `Cooler` generates a sparse, compressed, and binary persistent representation of proximity ligation contact matrix
- Stores the matrix as HDF5 file object
- Provides a python API to enable contact matrix data manipulation
- 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 result in a compressed file with the `.gz` suffix. However due to format differences 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
```

Generating 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
```

Generating multi-resolution 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 enable zooming in and out of interesting regions. The cooler `zoomify` utility enables you to generate a multi-resolution cooler file by coarsening. The input to `cooler zoomify` is a single resolution `.cool` file. To enable zooming in into interesting regions we suggest you generate a `.cool` file with a small bin size, e.g. 1 kb. 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* (there is no need to specify the output name).

Tip: `Cooler` offers additional functions that were not discussed here, including generating a cooler file from a pre-binned matrix, matrix normalization and more. To learn more about these advanced options, 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.

1.8 Captured libraries Data Sets

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

```
wget https://s3.amazonaws.com/dovetail.pub/capture/human/h_probes_v1.0.bed
```

1.8.1 Human capture panel

Panel associated files

File	Link
Probes	https://s3.amazonaws.com/dovetail.pub/capture/human/h_probes_v1.0.bed
Baits	https://s3.amazonaws.com/dovetail.pub/capture/human/h_baits_v1.0.bed
Features table (ensembl v86)	https://s3.amazonaws.com/dovetail.pub/capture/human/h_probe_metaData.csv
reference (GRCh38.p12)	https://s3.amazonaws.com/dovetail.pub/reference/human/hg38.fa

Sequenced libraries

Library	Link
NSC capture rep1	• https://s3.amazonaws.com/dovetail.pub/capture/fastqs/NSC_rep1_R1.fastq.gz • https://s3.amazonaws.com/dovetail.pub/capture/fastqs/NSC_rep1_R2.fastq.gz
NSC capture rep2	• https://s3.amazonaws.com/dovetail.pub/capture/fastqs/NSC_rep2_R1.fastq.gz • https://s3.amazonaws.com/dovetail.pub/capture/fastqs/NSC_rep2_R2.fastq.gz
iPSC capture rep1	• https://s3.amazonaws.com/dovetail.pub/capture/fastqs/iPSC_rep1_R1.fastq.gz • https://s3.amazonaws.com/dovetail.pub/capture/fastqs/iPSC_rep1_R2.fastq.gz
iPSC capture rep2	• https://s3.amazonaws.com/dovetail.pub/capture/fastqs/iPSC_rep2_R1.fastq.gz • https://s3.amazonaws.com/dovetail.pub/capture/fastqs/iPSC_rep2_R2.fastq.gz

CHiCAGO supporting files

Library	Link
5kb pooled fragments Design files	https://s3.amazonaws.com/dovetail.pub/capture/human/CHiCAGO_files/h_5kDesignFiles.tar.gz
10kb pooled fragments Design files	https://s3.amazonaws.com/dovetail.pub/capture/human/CHiCAGO_files/h_10kDesignFiles.tar.gz
20kb pooled fragments Design files	https://s3.amazonaws.com/dovetail.pub/capture/human/CHiCAGO_files/h_20kDesignFiles.tar.gz

1.8.2 Mouse capture panel

Panel associated files

File	Link
Probes	• https://s3.amazonaws.com/dovetail.pub/capture/mouse/m_probes_v1.0.bed
Baits	• https://s3.amazonaws.com/dovetail.pub/capture/mouse/m_baits_v1.0.bed
Features table	• https://s3.amazonaws.com/dovetail.pub/capture/mouse/m_probe_metaData.csv

CHiCAGO supporting files

Library	Link
5kb pooled fragments Design files	https://s3.amazonaws.com/dovetail.pub/capture/mouse/CHiCAGO_files/m_5kDesignFiles.tar.gz
10kb pooled fragments Design files	https://s3.amazonaws.com/dovetail.pub/capture/mouse/CHiCAGO_files/m_10kDesignFiles.tar.gz
20kb pooled fragments Design files	https://s3.amazonaws.com/dovetail.pub/capture/mouse/CHiCAGO_files/m_20kDesignFiles.tar.gz

1.9 Support

For help or related questions 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