cooler Documentation

Release 0.8.10

Nezar Abdennur

Sep 25, 2020

Contents

Cooler is a Python support library for .cool/.mcool files: an efficient storage format for high resolution genomic interaction matrices.

The cooler package aims to facilitate:

- Creation, aggregation and manipulation of genomically-labeled sparse matrices.
- Querying: sequential and range query patterns and tabular and sparse/dense array retrieval.
- Scalable out-of-core operations on the data.
- Data export and visualization.

Follow cooler development on GitHub.

Contents:

CHAPTER 1

Quickstart

1.1 Installation

Install `cooler` from PyPI using **`pip`**.

```
$ pip install cooler
```

Requirements:

- Python 2.7 or 3.4 and higher
- libhdf5
- Python packages `numpy`, `scipy`, `pandas`, `h5py`.

We highly recommend using the conda package manager to install scientific packages like these. To get **`conda`**, you can download either the full Anaconda Python distribution which comes with lots of data science software or the minimal Miniconda distribution which is just the standalone package manager plus Python. In the latter case, you can install the packages as follows:

```
$ conda install numpy scipy pandas h5py
```

If you are using conda, you can alternatively install cooler from the bioconda channel.

```
$ conda install -c conda-forge -c bioconda cooler
```

1.2 Command line interface

See:

- Jupyter Notebook CLI walkthrough.
- The CLI Reference for more information.

The `cooler` package includes command line tools for creating, querying and manipulating cooler files.

```
$ cooler cload pairs hg19.chrom.sizes:10000 $PAIRS_FILE out.10000.cool
$ cooler balance -p 10 out.10000.cool
$ cooler dump -b -t pixels --header --join -r chr3:10M-12M -r2 chr17 out.10000.cool |␣
↪head
```

Output:

```
chrom1  start1  end1    chrom2  start2  end2    count   balanced
chr3    10000000        10010000        chr17   0       10000   1       0.810766
chr3    10000000        10010000        chr17   520000  530000  1       1.2055
chr3    10000000        10010000        chr17   640000  650000  1       0.587372
chr3    10000000        10010000        chr17   900000  910000  1       1.02558
chr3    10000000        10010000        chr17   1030000 1040000 1       0.718195
chr3    10000000        10010000        chr17   1320000 1330000 1       0.803212
chr3    10000000        10010000        chr17   1500000 1510000 1       0.925146
chr3    10000000        10010000        chr17   1750000 1760000 1       0.950326
chr3    10000000        10010000        chr17   1800000 1810000 1       0.745982
```

1.3 Python API

See:

- Jupyter Notebook API walkthrough.
- The *API Reference* for more information.

The `cooler` library provides a thin wrapper over the excellent NumPy-aware h5py Python interface to HDF5. It supports creation of cooler files and the following types of **range queries** on the data:

- Tabular selections are retrieved as Pandas DataFrames and Series.
- Matrix selections are retrieved as NumPy arrays, DataFrames, or SciPy sparse matrices.
- Metadata is retrieved as a json-serializable Python dictionary.
- Range queries can be supplied using either integer bin indexes or genomic coordinate intervals.

```
>>> import cooler
>>> import matplotlib.pyplot as plt
>>> c = cooler.Cooler('bigDataset.cool')
>>> resolution = c.binsize
>>> mat = c.matrix(balance=True).fetch('chr5:10,000,000-15,000,000')
>>> plt.matshow(np.log10(mat), cmap='YlOrRd')
```

```
>>> import multiprocessing as mp
>>> import h5py
>>> pool = mp.Pool(8)
>>> c = cooler.Cooler('bigDataset.cool')
>>> weights, stats = cooler.balance_cooler(c, map=pool.map, ignore_diags=3, min_
↪nnz=10)
```

CHAPTER 2

What is cooler?

Cooler is the *implementation* of a data model for genomically-labeled sparse 2D arrays (matrices) with identical axes in HDF5. It is also the name of the Python package that supports the format.

We use the term genomically-labeled array to refer to a data structure that assigns unique quantitative values to tuples of *genomic bins* obtained from an interval partition of a reference genome assembly. The tuples of bins make up the coordinates of the array's elements. By omitting elements possessing zero or no value, the representation becomes sparse.

Cooler was designed for the storage and manipulation of extremely large Hi-C datasets at any resolution, but is not limited to Hi-C data in any way.

2.1 Genomically-labeled arrays

We can describe two tabular representations of such data.

BG2

By extending the bedGraph format, we can encode a 2D array with the following header.

chrom1	start1	end1	chrom2	start2	end2	value

Other bin-related attributes (e.g. X and Y) and be appended as columns X1, X2, Y1, Y2, and so on. One problem with this representation is that each bin-related attribute can be repeated many times throughout the table, leading to great redundancy.

Note: bedGraph is technically different from BED: the former describes a quantitative track supported by **non-overlapping** intervals (a step function), while the latter describes genomic intervals with no such restrictions. BG2 is different from BEDPE in the same way: intervals on the same axis are non-overlapping and interval pairs are not repeated (describing a heatmap).

COO

A simple solution is to decompose or "normalize" the single table into two files. The first is a bin table that describes the genomic bin segmentation on both axes of the matrix (in the one-dimensional bedGraph style). The second table contains single columns that reference the rows of the bin table, providing a condensed representation of the nonzero elements of the array. Conveniently, this corresponds to the classic coordinate list (COO) sparse matrix representation. This two-table representation is used as a text format by HiC-Pro.

bins		
chrom	start	end

elements		
bin1_id	bin2_id	value

The table of elements (non-zero pixels) is often too large to hold in memory, but for any small selection of elements we can reconstitute the bin-related attributes by "joining" the bin IDs against the bin table. We refer to this process as element *annotation*.

2.2 Data model

We model a genomically-labeled sparse matrix using three tables. It corresponds to the bin and element (pixel) tables above. We include a third chromosome description table for completeness, and indexes to support random access.

2.2.1 Tables

chroms

- Required columns: `name[, length]`
- Order: *enumeration*

An semantic ordering of the chromosomes, scaffolds or contigs of the assembly that the data is mapped to. This information can be extracted from the bin table below, but is included separately for convenience. This enumeration is the intended ordering of the chromosomes as they would appear in a global genomic matrix. Additional columns can provide metadata on the chromosomes, such as their length.

bins

- Required columns: `chrom, start, end [, weight]`
- Order: `chrom` (*enum*), `start`

An enumeration of the concatenated genomic bins that make up a single dimension or axis of the global genomic matrix. Genomic bins can be of fixed size or variable sizes (e.g. restriction fragments). A genomic bin is defined by the triple (chrom, start, end), where start is zero-based and end is 1-based. The order is significant: the bins are sorted by chromosome (based on the chromosome enumeration) then by start, and each genomic bin is implicitly endowed with a 0-based bin ID from this ordering (i.e., its row number in the table). A reserved but optional column called `weight` can store weights for normalization or matrix balancing. Additional columns can be added to describe other bin-associated properties such as additional normalization vectors or bin-level masks.

pixels

- Required columns: `bin1_id, bin2_id, count`
- Order: `bin1_id, bin2_id`

The array is stored as a single table containing only the nonzero upper triangle elements, assuming the ordering of the bins given by the bin table. Each row defines a non-zero element of the genomic matrix. Additional columns can be appended to store pixel-associated properties such as pixel-level masks or filtered and transformed versions of the data. Currently, the pixels are sorted lexicographically by the bin ID of the 1st axis (matrix row) then the bin ID of the 2nd axis (matrix column).

2.2.2 Indexes

The sort order on the pixels and types of indexing strategies that can be used are strongly related. We stipulate that the records of the pixel table must be sorted lexicographically by the bin ID along the first axis, then by the bin ID along the second axis. This way, the `bin1_id` column can be substituted with its run length encoding, which serves as a lookup index for the rows of the ma- trix. With this index, we obtain a compressed sparse row (CSR) sparse matrix representation.

Given an enumeration of chromosomes, the bin table must also be lexicographically sorted by chromosome then by start coordinate. Then similarly, the chrom column of the bin table will reference the rows of the chrom table, and can also be substituted with a run length encoding.

2.3 Container

The reference implementation of this data model uses HDF5 as the container format. HDF5 is a hierarchical data format for homongenenously typed multidimensional arrays, which supports chunking, compression, and random access. The HDF5 file specification and open source standard library is maintained by the nonprofit HDF Group.

HDF5 files consist of three fundamental entities: groups, datasets, and attibutes. The hierarchical organization of an HDF5 file is conceptually analogous to a file system: *groups* are akin to directories and *datasets* (arrays) are akin to files. Additionally, key-value metadata can be attached to groups and datasets using *attributes*. The standard library provides the ability to access and manipulate these entities. There are bindings for virtually every platform and programming environment. To learn more in detail about HDF5, I recommend the book HDF5 and Python by Andrew Collette, the author of `h5py`.

To implement the data model in HDF5, data tables are stored in a columnar representation as HDF5 groups of 1D array datasets of equal length. Metadata is stored using top-level attributes. See the *schema*.

2.3.1 HDF5 bindings in other languages

- canonical C-library libhdf5
- C++: C++ API
- IDL: bindings
- Java: Java HDF5 Interface
- Julia: HDF5.jl
- Mathematica: API
- MATLAB: high and low level API

- node.js: hdf5.node
- Perl: PDL::IO::HDF5
- R: rhdf5, h5
- Apache Spark

2.3.2 Caveats

HDF5 is not a database system and is not journalled. It supports concurrent read access but not simultaneous reads and writes (with upcoming support for the SWMR access pattern). One must be careful using multi-process concurrency based on Unix `fork()`: if a file is already open before the fork, the child processes will inherit state such that they won't play well with each other on that file. HDF5 will work fine with Python's `multiprocessing` as long as you make sure to close file handles before creating a process pool. Otherwise, you'll need to use locks or avoid opening the file in worker processes completely (see this blog post for a simple workaround). For more information on using multiprocessing safely, see this discussion.

CHAPTER 3

Concepts

3.1 Resource String

The default location for a single-cooler .cool file is the root group `/` of the HDF5 file. It does not need to be explicitly specified.

```
>>> import cooler
>>> c = cooler.Cooler('data/WT.DpnII.10kb.cool')
>>> c = cooler.Cooler('data/WT.DpnII.10kb.cool::/')  # same as above
```

However, coolers can be stored at any level of the HDF5 hierarchy and qualified using a URI string of the form `/path/to/cool/file::/path/to/cooler/group`.

```
>>> c1 = cooler.Cooler('data/WT.DpnII.mcool::resolutions/10000')
>>> c2 = cooler.Cooler('data/WT.DpnII.mcool::resolutions/1000')
```

The current standard for Hi-C coolers is to name multi-resolution coolers under `.mcool` extension, and store differrent resolutions in an HDF5 group `resolutions`, as shown above.

3.2 Data selection

Several *`cooler.Cooler`* methods return data selectors. Those include selecting tables and matrices (see below). Data selectors don't retrieve any data from disk until queried. There are several ways to query using selectors. Genomic range strings may be provided as 3-tuples `(chrom: str, start: int, end: int)` or in UCSC-style strings of the style `{chrom}:{start}-{end}`. Unit prefixes `k, M, G` are supported in range strings. For regions with start and end that are not multiples of the resolution, selectors return the range of shortest range bins that fully contains the open interval [start, end).

3.2.1 Table selectors (chroms, bins, pixels)

There are data selectors for the three tables: *cooler.Cooler.chroms()*, *cooler.Cooler.bins()*, *cooler.Cooler.pixels()*. They support the following:

- lazily select columns or lists of columns, returning new selectors
- query table rows using integer/slice indexing syntax

```
>>> c.bins()
<cooler.core.RangeSelector1D at 0x7fdb2e4f0710>

>>> c.bins()[:10]
chrom    start       end    weight
0  chr1        0   1000000       NaN
1  chr1  1000000   2000000  1.243141
2  chr1  2000000   3000000  1.313995
3  chr1  3000000   4000000  1.291705
4  chr1  4000000   5000000  1.413288
5  chr1  5000000   6000000  1.165382
6  chr1  6000000   7000000  0.811824
7  chr1  7000000   8000000  1.056107
8  chr1  8000000   9000000  1.058915
9  chr1  9000000  10000000  1.035910

>>> c.pixels()[:10]
   bin1_id  bin2_id  count
0        0        0  18578
1        0        1  11582
2        0        2    446
3        0        3    196
4        0        4     83
5        0        5    112
6        0        6    341
7        0        7    255
8        0        8    387
9        0        9    354

>>> c.bins()['weight']
 <cooler.core.RangeSelector1D at 0x7fdb2e509240>

>>> weights = c.bins()['weight'].fetch('chr3')
>>> weights.head()
494    1.144698
495    1.549848
496    1.212580
497    1.097539
498    0.871931
Name: weight, dtype: float64

>>> mybins1 = c.bins().fetch('chr3:10,000,000-20,000,000')
>>> mybins2 = c.bins().fetch( ('chr3', 10000000, 20000000) )
>>> mybins2.head()
    chrom     start       end    weight
504  chr3  10000000  11000000  0.783160
505  chr3  11000000  12000000  0.783806
506  chr3  12000000  13000000  0.791204
507  chr3  13000000  14000000  0.821171
508  chr3  14000000  15000000  0.813079
```

3.2.2 Matrix selector

The *cooler.Cooler.matrix()* selector supports two types of queries:

- 2D bin range queries using slice indexing syntax
- 2D genomic range range queries using the `fetch` method

The matrix selector's fetch method is intended to represent a **2D range query** (rectangular window), similar to the slice semantics of a 2D array. Given a matrix selector `sel`, when calling `sel.fetch(region1, region2)` the `region1` and `region2` are single contiguous genomic ranges along the first and second axes of the contact matrix. This mirrors the global slice indexing interface of the matrix selector `sel[a:b, c:d]`, where the only difference is that the genomic range syntax cannot cross chromosome boundaries. If `region2` is not provided, it is taken to be the same as `region1`. That means that `sel.fetch('chr2:10M-20M')` returns the same result as `sel.fetch('chr2:10M-20M', 'chr2:10M-20M')`. As a single rectangular window, queries like `sel.fetch('chr2', 'chr3')` will return *inter*-chromosomal values and not intra-chromosomal ones.

```
>>> c.matrix(balance=False)[1000:1005, 1000:1005]
array([[120022,  34107,  17335,  14053,   4137],
       [ 34107,  73396,  47427,  16125,   3642],
       [ 17335,  47427,  80458,  25105,   5394],
       [ 14053,  16125,  25105, 104536,  27214],
       [  4137,   3642,   5394,  27214, 114135]])

>>> matrix = c.matrix(sparse=True, balance=False)
>>> matrix
<cooler.core.RangeSelector2D at 0x7fdb2e245908>

>>> matrix[:]
<3114x3114 sparse matrix of type '<class 'numpy.int64'>'
    with 8220942 stored elements in COOrdinate format>

>>> c.matrix(balance=False, as_pixels=True, join=True)[1000:1005, 1000:1005]
   chrom1     start1       end1 chrom2     start2       end2   count
0    chr5  115000000  116000000   chr5  115000000  116000000  120022
1    chr5  115000000  116000000   chr5  116000000  117000000   34107
2    chr5  115000000  116000000   chr5  117000000  118000000   17335
3    chr5  115000000  116000000   chr5  118000000  119000000   14053
4    chr5  115000000  116000000   chr5  119000000  120000000    4137
5    chr5  116000000  117000000   chr5  116000000  117000000   73396
6    chr5  116000000  117000000   chr5  117000000  118000000   47427
7    chr5  116000000  117000000   chr5  118000000  119000000   16125
8    chr5  116000000  117000000   chr5  119000000  120000000    3642
9    chr5  117000000  118000000   chr5  117000000  118000000   80458
10   chr5  117000000  118000000   chr5  118000000  119000000   25105
11   chr5  117000000  118000000   chr5  119000000  120000000    5394
12   chr5  118000000  119000000   chr5  118000000  119000000  104536
13   chr5  118000000  119000000   chr5  119000000  120000000   27214
14   chr5  119000000  120000000   chr5  119000000  120000000  114135


>>> A1 = c.matrix().fetch('chr1')
>>> A2 = c.matrix().fetch('chr3:10,000,000-20,000,000')
>>> A3 = c.matrix().fetch( ('chr3', 10000000, 20000000) )
>>> A4 = c.matrix().fetch('chr2', 'chr3')
>>> A5 = c.matrix().fetch('chr3:10M-20M', 'chr3:35M-40M')
```

3.2.3 Dask

Dask data structures provide a way to manipulate and distribute computations on larger-than-memory data using familiar APIs. The experimental `read_table` function can be used to generate a dask dataframe backed by the pixel table of a cooler as follows:

```
>>> from cooler.sandbox.dask import read_table
>>> df = daskify(c.filename, 'pixels')

>>> df
Dask DataFrame Structure:
                bin1_id bin2_id  count
npartitions=223
0                 int64   int64  int64
9999999             ...     ...    ...
...                 ...     ...    ...
2219999999          ...     ...    ...
2220472929          ...     ...    ...
Dask Name: daskify, 223 tasks

>>> df = cooler.annotate(df, c.bins(), replace=False)
>>> df
Dask DataFrame Structure:
                chrom1 start1   end1  weight1  chrom2 start2   end2  weight2 bin1_id␣
↪bin2_id  count
npartitions=31
None            object  int64  int64  float64  object  int64  int64  float64   int64 ␣
↪ int64  int64
None               ...    ...    ...      ...     ...    ...    ...      ...     ... ␣
↪   ...    ...
...                ...    ...    ...      ...     ...    ...    ...      ...     ... ␣
↪   ...    ...
None               ...    ...    ...      ...     ...    ...    ...      ...     ... ␣
↪   ...    ...
None               ...    ...    ...      ...     ...    ...    ...      ...     ... ␣
↪   ...    ...
Dask Name: getitem, 125 tasks

>>> df = df[df.chrom1 == df.chrom2]
>>> grouped = df.groupby(df.bin2_id - df.bin1_id)
>>> x = grouped['count'].sum()
>>> x
Dask Series Structure:
npartitions=1
None    int64
None      ...
Name: count, dtype: int64
Dask Name: series-groupby-sum-agg, 378 tasks

>>> x.compute()
0       476155231
1       284724453
2       139952477
3        96520218
4        71962080
5        56085850
6        45176881
7        37274367
```

(continues on next page)

(continued from previous page)

```
8         31328555
9         26781986
10        23212616
11        20366934
12        18066135
13        16159826
14        14584058
15        13249443
16        12117854
17        11149845
...
```

Learn more about the Dask project.

3.3 Create a scool file

The creation of a single-cell cooler file is similar to a regular cooler file. Each cell needs to have a name, bin table and a pixel table. All cells must have the same dimensions, and the bins and pixels needs to be provided as two dicts with the cell names as keys.

```
>>> name_pixel_dict = {'cell1': pixels_cell1, 'cell2': pixels_cell2, 'cell3': pixels_
↪cell3}
>>> name_bins_dict = {'cell1': bins_cell1, 'cell2': bins_cell2, 'cell3': bins_cell3}
>>> cooler.create_scool('single_cell_cool.scool', name_bins_dict, name_pixel_dict)
```

To read the content, each individual cell must be handled as a regular cool file.

```
>> content_of_scool = cooler.fileops.list_coolers('single_cell_cool.scool')
['/', '/cells/cell1', '/cells/cell2', '/cells/cell3']
>>> c1 = cooler.Cooler('single_cell_cool.scool::cells/cell1')
>>> c2 = cooler.Cooler('single_cell_cool.scool::cells/cell2')
>>> c3 = cooler.Cooler('single_cell_cool.scool::cells/cell3')
```

CHAPTER 4

Schema

Schema Version	3

The following document describes a compressed sparse row (CSR) storage scheme for a matrix (i.e., a quantitative heatmap) with genomically labeled dimensions/axes.

HDF5 does not natively implement sparse arrays or relational data structures: its datasets are dense multidimensional arrays. We implement tables and sparse array indexes in HDF5 using groups of 1D arrays. The descriptions of tables and indexes in this document specify required groups and arrays, conventional column orders, and default data types.

Summary of changes

- Version 3 introduces the `storage-mode` metadata attribute to accomodate square matrices that are non-symmetric. Version 2 files which lack the `storage-mode` attribute should be interpreted as using the "symmetric-upper" storage mode. See *Storage mode*.
- The multi-resolution cooler file layout has been standardized. See *File flavors*.

4.1 Data collection

We refer to the object hierarchy describing a single matrix as a cooler *data collection*. A cooler data collection consists of **tables**, **indexes** and **metadata** describing a genomically-labelled sparse matrix.

A typical data collection has the following structure. At the top level, there are four HDF5 Groups, each containing 1D arrays (HDF5 Datasets). The depiction below shows an example group hierarchy as a tree, with arrays at the leaves, printed with their shapes in parentheses and their data type symbols.

```
/
├── chroms
│   ├── length (24,) int32
│   └── name (24,) |S64
```

(continues on next page)

(continued from previous page)

```
├── bins
│   ├── chrom (3088281,) int32
│   ├── start (3088281,) int32
│   ├── end (3088281,) int32
│   └── weight (3088281,) float64
├── pixels
│   ├── bin1_id (271958554,) int64
│   ├── bin2_id (271958554,) int64
│   └── count (271958554,) int32
└── indexes
    ├── bin1_offset (3088282,) int64
    └── chrom_offset (25,) int64
```

4.2 URI syntax

We identify a cooler data collection using a **URI string** to its top-level group, separating the system path to the container file from the **group path** within the container file by a double colon `::`.

```
path/to/container.cool::/path/to/cooler/group
```

For any URI, the leading slash after the `::` may be omitted. To reference the root group `/`, the entire `::/` suffix may be omitted (i.e., just a file path).

4.3 Tables

A **table** is a group of equal-length 1D arrays representing **columns**.

Additional groups and tables may be added to a data collection as long as they are not nested under the group of another table.

This storage mode does not enforce specific **column orders**, but conventional orders for *required* columns is provided in the listings below.

This storage mode does not set limits on the **number or length of columns**. Additional arrays may be inserted into a table to form new columns, but they must conform to the common length of the table.

The table descriptions below are given in the datashape layout language. The column **data types** are given as numpy equivalents. They are only defaults and may be altered as desired.

GZIP is chosen as the default **compression** filter for all columns. This is for portability reasons, since all versions of the HDF5 library ship with it.

4.3.1 chroms

```
chroms: {
  # REQUIRED
  name:     typevar['Nchroms'] * string['ascii'],
  length:   typevar['Nchroms'] * int32
}
```

In HDF5, `name` is a null-padded, fixed-length ASCII array, which maps to numpy's `S` dtype.

4.3.2 bins

```
bins: {
  # REQUIRED
  chrom:    typevar['Nbins'] * categorical[typevar['name'], type=string,␣
↪ordered=True],
  start:    typevar['Nbins'] * int32,
  end:      typevar['Nbins'] * int32,

  # RESERVED
  weight:   typevar['Nbins'] * float64
}
```

In HDF5, we use the integer-backed ENUM type to encode the `chrom` column. For data collections with a very large number of scaffolds, the ENUM type information may be too large to fit in the object's metadata header. In that case, the `chrom` column is stored using raw integers and the enumeration is inferred from the `chrom` table.

Genomic intervals are stored using a 0-start, half-open representation. The first interval in a scaffold should have `start` = 0 and the last interval should have `end` = the chromosome length. Intervals are sorted by `chrom`, then by `start`.

Because they measure the same quantity in the same units, the coordinate columns `chroms/length`, `bins/start` and `bins/end` should be encoded using the same data type.

The **`cooler balance`** command stores balancing weights in a column called `weight` by default. NaN values indicate genomic bins that were blacklisted during the balancing procedure.

4.3.3 pixels

```
pixels: {
  # REQUIRED
  bin1_id:  typevar['Nnz'] * int64,
  bin2_id:  typevar['Nnz'] * int64,

  # RESERVED
  count:    typevar['Nnz'] * int32
}
```

In the matrix coordinate system, `bin1_id` refers to the ith axis and `bin2_id` refers to the jth. Bin IDs are zero-based, i.e. we start counting at 0. Pixels are sorted by `bin1_id` then by `bin2_id`.

The `count` column is integer by default, but floating point types can be substituted. Additional columns are to be interpreted as supplementary value columns.

> **Warning:** float16 has limited support from 3rd party libraries and is not recommended. For floating point value columns consider using either single- (float32) or double-precision (float64).

4.4 Indexes

Indexes are stored as 1D arrays in a separate group called `indexes`. They can be thought of as run-length encodings of the `bins/chrom` and `pixels/bin1_id` columns, respectively. Both arrays are required.

```
indexes: {
  chrom_offset:   (typevar['Nchroms'] + 1) * int64,
  bin1_offset:    (typevar['Nbins'] + 1) * int64
}
```

- `chrom_offset`: indicates which row in the bin table each chromosome first appears. The last element stores the length of the bin table.
- `bin1_offset`: indicates which row in the pixel table each bin1 ID first appears. The last element stores the length of the pixel table. This index is usually called *indptr* in CSR data structures.

4.5 Storage mode

Storing a symmetric matrix requires only the *upper triangular part, including the diagonal*, since the remaining elements can be reconstructed from the former ones. To indicate the use of this **mode of matrix storage** to client software, the value of the metadata attribute `storage-mode` must be set to `"symmetric-upper"` (see *Metadata*).

New in version 3: To indicate the absence of a special storage mode, e.g. for **non-symmetric** matrices, `storage-mode` must be set to `"square"`. This storage mode indicates to client software that 2D range queries should not be symmetrized.

Warning: In schema v2 and earlier, the symmetric-upper storage mode is always assumed.

4.6 Metadata

Essential key-value properties are stored as HDF5 attributes at the top-level group of the data collection. Note that depending on where the data collection is located in the file, this can be different from the root group of the entire file `/`.

Required attributes

`format : string (constant)`
: "HDF5::Cooler"

`format-version : int`
: The schema version used.

`bin-type : { "fixed", "variable" }`
: Indicates whether the resolution is constant along both axes.

`bin-size : int or "null"`
: Size of genomic bins in base pairs if bin-type is "fixed". Otherwise, "null".

`storage-mode : { "symmetric-upper", "square" }`
: Indicates whether ordinary sparse matrix encoding is used ("square") or whether a symmetric matrix is encoded by storing only the upper triangular elements ("symmetric-upper").

Reserved, but optional

`assembly : string`
: Name of the genome assembly, e.g. "hg19".

generated-by : string
Agent that created the file, e.g. "cooler-x.y.z".

creation-date : datetime string
The moment the collection was created.

metadata : JSON
Arbitrary JSON-compatible **user metadata** about the experiment.

All scalar string attributes, including serialized JSON, must be stored as **variable-length UTF-8 encoded strings**.

Warning: When assigning scalar string attributes in Python 2, always store values having `unicode` type. In h5py, assigning a Python text string (Python 3 `str` or Python 2 `unicode`) to an HDF5 attribute results in variable-length UTF-8 storage.

Additional metadata may be stored in other top-level attributes and the attributes of table groups and columns.

4.7 File flavors

Many cooler data collections can be stored in a single file. We recognize two conventional **layouts**:

4.7.1 Single-resolution

- A single-resolution cooler file that contains a single data collection under the `/` group. Conventional file extension: `.cool`.

```
XYZ.1000.cool
/
  ├── bins
  ├── chroms
  ├── pixels
  └── indexes
```

4.7.2 Multi-resolution

- A multi-resolution cooler file that contains multiple "coarsened" resolutions or "zoom-levels" derived from the same dataset. Multires cooler files should store each data collection underneath a group called `/resolutions` within a sub-group whose name is the bin size (e.g, `XYZ.1000.mcool::resolutions/10000`). If the base cooler has variable-length bins, then use `1` to designate the base resolution, and the use coarsening multiplier (e.g. `2`, `4`, `8`, etc.) to name the lower resolutions. Conventional file extension: `.mcool`.

```
XYZ.1000.mcool
/
  └── resolutions
      ├── 1000
      │   ├── bins
      │   ├── chroms
      │   ├── pixels
      │   └── indexes
      ├── 2000
      │   ├── bins
```

(continues on next page)

(continued from previous page)

```
      │   ├── chroms
      │   ├── pixels
      │   └── indexes
      ├── 5000
      │   ├── bins
      │   ├── chroms
      │   ├── pixels
      │   └── indexes
      ├── 10000
      │   ├── bins
      │   ├── chroms
      │   ├── pixels
      │   └── indexes
      .
      .
      .
```

In addition, a multi-resolution cooler file may indicate to clients that it is using this layout with the following `/`-level attributes:

`format : string (constant)`
"HDF5::MCOOL"

`format-version : int`
2

`bin-type : { "fixed", "variable" }`
Indicates whether the resolution is constant along both axes.

Note: The old multi-resolution layout used resolutions strictly in increments of *powers of 2*. In this layout (MCOOL version 2), the data collections are named by zoom level, starting with `XYZ.1000.mcool::0` being the coarsest resolution up until the finest or "base" resolution (e.g., `XYZ.1000.mcool::14` for 14 levels of coarsening).

Changed in version 0.8: Both the legacy layout and the new mcool layout are supported by HiGlass. Prior to cooler 0.8, the new layout was produced only when requesting a specific list of resolutions. As of cooler 0.8, the new layout is always produced by the **`cooler zoomify`** command unless the `--legacy` option is given. Files produced by *`cooler.zoomify_cooler()`*, hic2cool, and the mcools from the 4DN data portal also follow the new layout.

4.7.3 Single-cell (single-resolution)

A single-cell cooler file contains all the matrices of a single-cell Hi-C data set. All cells are stored under a group called `/cells`, and all cells share the primary bin table columns i.e. `bins['chrom']`, `bins['start']` and `bins['end']` which are hardlinked to the root-level bin table. Any individual cell can be accessed using the regular *`cooler.Cooler`* interface. Conventional file extension: `.scool`.

```
XYZ.scool
/
 ├── bins
 ├── chroms
 └── cells
     ├── cell_id1
     │   ├── bins
     │   ├── chroms
     │   ├── pixels
```

(continues on next page)

(continued from previous page)

```
    │   └── indexes
    ├── cell_id2
    │   ├── bins
    │   ├── chroms
    │   ├── pixels
    │   └── indexes
    ├── cell_id3
    │   ├── bins
    │   ├── chroms
    │   ├── pixels
    │   └── indexes
    ├── cell_id4
    │   ├── bins
    │   ├── chroms
    │   ├── pixels
    │   └── indexes
    .
    .
    .
```

In addition, a single-cell single-resolution cooler file may indicate to clients that it is using this layout with the following `/`-level attributes:

`format : string (constant)`
: "HDF5::SCOOL"

`format-version : int`
: 1

`bin-type : { "fixed", "variable" }`
: Indicates whether the resolution is constant along both axes.

`bin-size : int`
: The bin resolution

`nbins : int`
: The number of bins

`nchroms : int`
: The number of chromosomes of the cells

`ncells : int`
: The number of stored cells

4.8 Previous schema versions

- v1
- v2

CHAPTER 5

API Reference

5.1 Quick reference

5.1.1 Cooler class

cooler.Cooler(store[, root])	A convenient interface to a cooler data collection.
cooler.Cooler.binsize	Resolution in base pairs if uniform else None
cooler.Cooler.chromnames	List of reference sequence names
cooler.Cooler.chromsizes	Ordered mapping of reference sequences to their lengths in bp
cooler.Cooler.bins(**kwargs)	Bin table selector
cooler.Cooler.pixels([join])	Pixel table selector
cooler.Cooler.matrix([field, balance, ...])	Contact matrix selector
cooler.Cooler.open([mode])	Open the HDF5 group containing the Cooler with `h5py`
cooler.Cooler.info	File information and metadata
cooler.Cooler.offset(region)	Bin ID containing the left end of a genomic region
cooler.Cooler.extent(region)	Bin IDs containing the left and right ends of a genomic region

5.1.2 Creation/reduction

cooler.create_cooler(cool_uri, bins, pixels)	Create a cooler from bins and pixels at the specified URI.
cooler.merge_coolers(output_uri, input_uris, ...)	Merge multiple coolers with identical axes.
cooler.coarsen_cooler(base_uri, output_uri, ...)	Coarsen a cooler to a lower resolution by an integer factor *k*.
cooler.zoomify_cooler(base_uris, outfile, ...)	Generate multiple cooler resolutions by recursive coarsening.

Continued on next page

Table 2 – continued from previous page

cooler.create_scool(cool_uri, bins, ... [, ...])	Create a single-cell (scool) file.

5.1.3 Manipulation

cooler.annotate(pixels, bins[, replace])	Add bin annotations to a data frame of pixels.
cooler.balance_cooler(clr[, chunksize, map, ...])	Iterative correction or matrix balancing of a sparse Hi-C contact map in Cooler HDF5 format.
cooler.rename_chroms(clr, rename_dict[, h5opts])	Substitute existing chromosome/contig names for new ones.

5.1.4 File operations

cooler.fileops.is_cooler(uri)	Determine if a URI string references a cooler data collection.
cooler.fileops.is_multires_file(filepath[, ...])	Determine if a file is a multi-res cooler file.
cooler.fileops.list_coolers(filepath)	List group paths to all cooler data collections in a file.
cooler.fileops.cp(src_uri, dst_uri[, overwrite])	Copy a group or dataset from one file to another or within the same file.
cooler.fileops.mv(src_uri, dst_uri[, overwrite])	Rename a group or dataset within the same file.
cooler.fileops.ln(src_uri, dst_uri[, soft, ...])	Create a hard link to a group or dataset in the same file.

5.1.5 Sandbox

cooler.sandbox.dask.read_table

5.2 cooler

class cooler.**Cooler**(*store*, *root=None*, ***kwargs*)

A convenient interface to a cooler data collection.

Parameters

- **store** (str, h5py.File or h5py.Group) – Path to a cooler file, URI string, or open handle to the root HDF5 group of a cooler data collection.
- **root** (*str, optional [deprecated]*) – HDF5 Group path to root of cooler group if store is a file. This option is deprecated. Instead, use a URI string of the form <file_path>::<group_path>.
- **kwargs** (*optional*) – Options to be passed to h5py.File() upon every access. By default, the file is opened with the default driver and mode='r'.

Notes

If store is a file path, the file will be opened temporarily in when performing operations. This allows *Cooler* objects to be serialized for multiprocess and distributed computations.

Metadata is accessible as a dictionary through the *info* property.

Table selectors, created using *chroms()*, *bins()*, and *pixels()*, perform range queries over table rows, returning `pd.DataFrame` and `pd.Series`.

A matrix selector, created using *matrix()*, performs 2D matrix range queries, returning `numpy.ndarray` or `scipy.sparse.coo_matrix`.

bins (***kwargs*)
Bin table selector

Returns *Table selector*

binsize
Resolution in base pairs if uniform else None

chromnames
List of reference sequence names

chroms (***kwargs*)
Chromosome table selector

Returns *Table selector*

chromsizes
Ordered mapping of reference sequences to their lengths in bp

extent (*region*)
Bin IDs containing the left and right ends of a genomic region

Parameters **region** (*str or tuple*) – Genomic range

Returns *2-tuple of ints*

Examples

```
>>> c.extent('chr3')  # doctest: +SKIP
(1311, 2131)
```

info
File information and metadata

Returns *dict*

matrix (*field=None*, *balance=True*, *sparse=False*, *as_pixels=False*, *join=False*, *ignore_index=True*, *divisive_weights=None*, *max_chunk=500000000*)
Contact matrix selector

Parameters

- **field** (*str, optional*) – Which column of the pixel table to fill the matrix with. By default, the 'count' column is used.
- **balance** (*bool, optional*) – Whether to apply pre-calculated matrix balancing weights to the selection. Default is True and uses a column named 'weight'. Alternatively, pass the name of the bin table column containing the desired balancing weights. Set to False to return untransformed counts.
- **sparse** (*bool, optional*) – Return a scipy.sparse.coo_matrix instead of a dense 2D numpy array.
- **as_pixels** (*bool, optional*) – Return a DataFrame of the corresponding rows from the pixel table instead of a rectangular sparse matrix. False by default.

- **join** (*bool, optional*) – If requesting pixels, specifies whether to expand the bin ID columns into (chrom, start, end). Has no effect when requesting a rectangular matrix. Default is True.
- **ignore_index** (*bool, optional*) – If requesting pixels, don't populate the index column with the pixel IDs to improve performance. Default is True.
- **divisive_weights** (*bool, optional*) – Force balancing weights to be interpreted as divisive (True) or multiplicative (False). Weights are always assumed to be multiplicative by default unless named KR, VC or SQRT_VC, in which case they are assumed to be divisive by default.

Returns *Matrix selector*

Notes

If `as_pixels=True`, only data explicitly stored in the pixel table will be returned: if the cooler's storage mode is symmetric-upper, lower triangular elements will not be generated. If `as_pixels=False`, those missing non-zero elements will automatically be filled in.

offset (*region*)

Bin ID containing the left end of a genomic region

Parameters **region** (*str or tuple*) – Genomic range

Returns *int*

Examples

```
>>> c.offset('chr3')  # doctest: +SKIP
1311
```

open (*mode='r', **kwargs*)

Open the HDF5 group containing the Cooler with `h5py`

Functions as a context manager. Any `open_kws` passed during construction are ignored.

Parameters **mode** (*str, optional [default: 'r']*) –

- `'r'` (readonly)
- `'r+'` or `'a'` (read/write)

Notes

For other parameters, see `h5py.File`.

pixels (*join=False, **kwargs*)

Pixel table selector

Parameters **join** (*bool, optional*) – Whether to expand bin ID columns into chrom, start, and end columns. Default is `False`.

Returns *Table selector*

storage_mode

Indicates whether ordinary sparse matrix encoding is used (`"square"`) or whether a symmetric matrix is encoded by storing only the upper triangular elements (`"symmetric-upper"`).

cooler.**annotate**(*pixels*, *bins*, *replace=False*)

Add bin annotations to a data frame of pixels.

This is done by performing a relational "join" against the bin IDs of a table that describes properties of the genomic bins. New columns will be appended on the left of the output data frame.

Changed in version 0.8.0: The default value of `replace` changed to False.

Parameters

- **pixels** (`DataFrame`) – A data frame containing columns named `bin1_id` and/or `bin2_id`. If columns `bin1_id` and `bin2_id` are both present in `pixels`, the adjoined columns will be suffixed with '1' and '2' accordingly.
- **bins** (`DataFrame` or DataFrame selector) – Data structure that contains a full description of the genomic bins of the contact matrix, where the index corresponds to bin IDs.
- **replace** (*bool, optional*) – Remove the original `bin1_id` and `bin2_id` columns from the output. Default is False.

Returns `DataFrame`

cooler.**create_cooler**(*cool_uri*, *bins*, *pixels*, *columns=None*, *dtypes=None*, *metadata=None*, *assembly=None*, *ordered=False*, *symmetric_upper=True*, *mode='w'*, *mergebuf=20000000*, *delete_temp=True*, *temp_dir=None*, *max_merge=200*, *boundscheck=True*, *dupcheck=True*, *triucheck=True*, *ensure_sorted=False*, *h5opts=None*, *lock=None*)

Create a cooler from bins and pixels at the specified URI.

Because the number of pixels is often very large, the input pixels are normally provided as an iterable (e.g., an iterator or generator) of DataFrame **chunks** that fit in memory.

New in version 0.8.0.

Parameters

- **cool_uri** (*str*) – Path to cooler file or URI string. If the file does not exist, it will be created.
- **bins** (*pandas.DataFrame*) – Segmentation of the chromosomes into genomic bins as a BED-like DataFrame with columns `chrom`, `start` and `end`. May contain additional columns.
- **pixels** (*DataFrame, dictionary, or iterable of either*) – A table, given as a dataframe or a column-oriented dict, containing columns labeled `bin1_id`, `bin2_id` and `count`, sorted by (`bin1_id`, `bin2_id`). If additional columns are included in the pixel table, their names and dtypes must be specified using the `columns` and `dtypes` arguments. For larger input data, an **iterable** can be provided that yields the pixel data as a sequence of chunks. If the input is a dask DataFrame, it will also be processed one chunk at a time.
- **columns** (*sequence of str, optional*) – Customize which value columns from the input pixels to store in the cooler. Non-standard value columns will be given dtype `float64` unless overriden using the `dtypes` argument. If `None`, we only attempt to store a value column named `"count"`.
- **dtypes** (*dict, optional*) – Dictionary mapping column names to dtypes. Can be used to override the default dtypes of `bin1_id`, `bin2_id` or `count` or assign dtypes to custom value columns. Non-standard value columns given in `dtypes` must also be provided in the `columns` argument or they will be ignored.
- **metadata** (*dict, optional*) – Experiment metadata to store in the file. Must be JSON compatible.

- **assembly** (*str, optional*) – Name of genome assembly.
- **ordered** (*bool, optional [default: False]*) – If the input chunks of pixels are provided with correct triangularity and in ascending order of (`bin1_id`, `bin2_id`), set this to `True` to write the cooler in one step. If `False` (default), we create the cooler in two steps using an external sort mechanism. See Notes for more details.
- **symmetric_upper** (*bool, optional [default: True]*) – If True, sets the file's storage-mode property to `symmetric-upper`: use this only if the input data references the upper triangle of a symmetric matrix! For all other cases, set this option to False.
- **mode** (*{'w' , 'a'}, optional [default: 'w']*) – Write mode for the output file. 'a': if the output file exists, append the new cooler to it. 'w': if the output file exists, it will be truncated. Default is 'w'.

Other Parameters

- **mergebuf** (*int, optional*) – Maximum number of records to buffer in memory at any give time during the merge step.
- **delete_temp** (*bool, optional*) – Whether to delete temporary files when finished. Useful for debugging. Default is False.
- **temp_dir** (*str, optional*) – Create temporary files in a specified directory instead of the same directory as the output file. Pass – to use the system default.
- **max_merge** (*int, optional*) – If merging more than `max_merge` chunks, do the merge recursively in two passes.
- **h5opts** (*dict, optional*) – HDF5 dataset filter options to use (compression, shuffling, checksumming, etc.). Default is to use autochunking and GZIP compression, level 6.
- **lock** (*multiprocessing.Lock, optional*) – Optional lock to control concurrent access to the output file.
- **ensure_sorted** (*bool, optional*) – Ensure that each input chunk is properly sorted.
- **boundscheck** (*bool, optional*) – Input validation: Check that all bin IDs lie in the expected range.
- **dupcheck** (*bool, optional*) – Input validation: Check that no duplicate pixels exist within any chunk.
- **triucheck** (*bool, optional*) – Input validation: Check that `bin1_id` <= `bin2_id` when creating coolers in symmetric-upper mode.

See also:

cooler.create_scool(), *cooler.create.sanitize_records()*, *cooler.create.sanitize_pixels()*

Notes

If the pixel chunks are provided in the correct order required for the output to be properly sorted, then the cooler can be created in a single step by setting `ordered=True`.

If not, the cooler is created in two steps via an external sort mechanism. In the first pass, the sequence of pixel chunks are processed and sorted in memory and saved to temporary coolers. In the second pass, the temporary coolers are merged into the output file. This way the individual chunks do not need to be provided in any particular order. When `ordered=False`, the following options for the merge step are available: `mergebuf`, `delete_temp`, `temp_dir`, `max_merge`.

Each chunk of pixels will go through a validation pipeline, which can be customized with the following options: `boundscheck`, `triucheck`, `dupcheck`, `ensure_sorted`.

`cooler.`**`merge_coolers`**(*output_uri*, *input_uris*, *mergebuf*, *columns=None*, *dtypes=None*, *agg=None*, ***kwargs*)

Merge multiple coolers with identical axes.

The merged cooler is stored at `output_uri`.

New in version 0.8.0.

Parameters

- **`output_uri`** (*`str`*) – Output cooler file path or URI.
- **`input_uris`** (*`list of str`*) – List of input file path or URIs of coolers to combine.
- **`mergebuf`** (*`int`*) – Maximum number of pixels processed at a time.
- **`columns`** (*`list of str, optional`*) – Specify which pixel value columns to include in the aggregation. Default is to use all available value columns.
- **`dtypes`** (*`dict, optional`*) – Specific dtypes to use for value columns. Default is to propagate the current dtypes of the value columns.
- **`agg`** (*`dict, optional`*) – Functions to use for aggregating each value column. Pass the same kind of dict accepted by `pandas.DataFrame.groupby.agg`. Default is to apply 'sum' to every value column.
- **`kwargs`** – Passed to `cooler.create`.

Notes

The default output file mode is 'w'. If appending output to an existing file, pass *mode='a'*.

See also:

`cooler.coarsen_cooler()`, *`cooler.zoomify_cooler()`*

`cooler.`**`coarsen_cooler`**(*base_uri*, *output_uri*, *factor*, *chunksize*, *nproc=1*, *columns=None*, *dtypes=None*, *agg=None*, ***kwargs*)

Coarsen a cooler to a lower resolution by an integer factor *k*.

This is done by pooling *k*-by-*k* neighborhoods of pixels and aggregating. Each chromosomal block is coarsened individually. Result is a coarsened cooler stored at `output_uri`.

New in version 0.8.0.

Parameters

- **`base_uri`** (*`str`*) – Input cooler file path or URI.
- **`output_uri`** (*`str`*) – Input cooler file path or URI.
- **`factor`** (*`int`*) – Coarsening factor.
- **`chunksize`** (*`int`*) – Number of pixels processed at a time per worker.
- **`nproc`** (*`int, optional`*) – Number of workers for batch processing of pixels. Default is 1, i.e. no process pool.
- **`columns`** (*`list of str, optional`*) – Specify which pixel value columns to include in the aggregation. Default is to use all available value columns.
- **`dtypes`** (*`dict, optional`*) – Specific dtypes to use for value columns. Default is to propagate the current dtypes of the value columns.

- **agg** (*dict, optional*) – Functions to use for aggregating each value column. Pass the same kind of dict accepted by `pandas.DataFrame.groupby.agg`. Default is to apply 'sum' to every value column.
- **kwargs** – Passed to `cooler.create`.

See also:

cooler.zoomify_cooler(), *cooler.merge_coolers()*

`cooler.`**zoomify_cooler**(*base_uris*, *outfile*, *resolutions*, *chunksize*, *nproc=1*, *columns=None*, *dtypes=None*, *agg=None*, ***kwargs*)

Generate multiple cooler resolutions by recursive coarsening.

Result is a "zoomified" or "multires" cool file stored at `outfile` using the MCOOL v2 layout, where coolers are stored under a hierarchy of the form `resolutions/<r>` for each resolution `r`.

New in version 0.8.0.

Parameters

- **base_uris** (*str or sequence of str*) – One or more cooler URIs to use as "base resolutions" for aggregation.
- **outfile** (*str*) – Output multires cooler (mcool) file path.
- **resolutions** (*list of int*) – A list of target resolutions to generate.
- **chunksize** (*int*) – Number of pixels processed at a time per worker.
- **nproc** (*int, optional*) – Number of workers for batch processing of pixels. Default is 1, i.e. no process pool.
- **columns** (*list of str, optional*) – Specify which pixel value columns to include in the aggregation. Default is to use only the column named 'count' if it exists.
- **dtypes** (*dict, optional*) – Specific dtypes to use for value columns. Default is to propagate the current dtypes of the value columns.
- **agg** (*dict, optional*) – Functions to use for aggregating each value column. Pass the same kind of dict accepted by `pandas.DataFrame.groupby.agg`. Default is to apply 'sum' to every value column.
- **kwargs** – Passed to `cooler.create`.

See also:

cooler.coarsen_cooler(), *cooler.merge_coolers()*

`cooler.`**balance_cooler**(*clr*, *chunksize=None*, *map=<class 'map'>*, *tol=1e-05*, *min_nnz=0*, *min_count=0*, *mad_max=0*, *cis_only=False*, *trans_only=False*, *ignore_diags=False*, *max_iters=200*, *rescale_marginals=True*, *use_lock=False*, *blacklist=None*, *x0=None*, *store=False*, *store_name='weight'*)

Iterative correction or matrix balancing of a sparse Hi-C contact map in Cooler HDF5 format.

Parameters

- **clr** (*cooler.Cooler*) – Cooler object
- **chunksize** (*int, optional*) – Split the contact matrix pixel records into equally sized chunks to save memory and/or parallelize. Default is to use all the pixels at once.
- **map** (*callable, optional*) – Map function to dispatch the matrix chunks to workers. Default is the builtin `map`, but alternatives include parallel map implementations from a multiprocessing pool.

- **tol** (*float, optional*) – Convergence criterion is the variance of the marginal (row/col) sum vector.
- **min_nnz** (*int, optional*) – Pre-processing bin-level filter. Drop bins with fewer nonzero elements than this value.
- **min_count** (*int, optional*) – Pre-processing bin-level filter. Drop bins with lower marginal sum than this value.
- **mad_max** (*int, optional*) – Pre-processing bin-level filter. Drop bins whose log marginal sum is less than `mad_max` median absolute deviations below the median log marginal sum.
- **cis_only** (*bool, optional*) – Do iterative correction on intra-chromosomal data only. Inter-chromosomal data is ignored.
- **trans_only** (*bool, optional*) – Do iterative correction on inter-chromosomal data only. Intra-chromosomal data is ignored.
- **blacklist** (*list or 1D array, optional*) – An explicit list of IDs of bad bins to filter out when performing balancing.
- **ignore_diags** (*int or False, optional*) – Drop elements occurring on the first `ignore_diags` diagonals of the matrix (including the main diagonal).
- **max_iters** (*int, optional*) – Iteration limit.
- **rescale_marginals** (*bool, optional*) – Normalize the balancing weights such that the balanced matrix has rows / columns that sum to 1.0. The scale factor is stored in the `stats` output dictionary.
- **x0** (*1D array, optional*) – Initial weight vector to use. Default is to start with ones(n_bins).
- **store** (*bool, optional*) – Whether to store the results in the file when finished. Default is False.
- **store_name** (*str, optional*) – Name of the column of the bin table to save to. Default name is 'weight'.

Returns

- **bias** (1D array, whose shape is the number of bins in `h5`.) – Vector of bin bias weights to normalize the observed contact map. Dropped bins will be assigned the value NaN. N[i, j] = O[i, j] * bias[i] * bias[j]
- **stats** (*dict*) – Summary of parameters used to perform balancing and the average magnitude of the corrected matrix's marginal sum at convergence.

`cooler.`**rename_chroms**(*clr*, *rename_dict*, *h5opts=None*)

Substitute existing chromosome/contig names for new ones. They will be written to the file and the Cooler object will be refreshed.

Parameters

- **clr** (`Cooler`) – Cooler object that can be opened with write permissions.
- **rename_dict** (*dict*) – Dictionary of old -> new chromosome names. Any names omitted from the dictionary will be kept as is.
- **h5opts** (*dict, optional*) – HDF5 filter options.

cooler.**create_scool**(*cool_uri*, *bins*, *cell_name_pixels_dict*, *columns=None*, *dtypes=None*, *metadata=None*, *assembly=None*, *ordered=False*, *symmetric_upper=True*, *mode='w'*, *mergebuf=20000000*, *delete_temp=True*, *temp_dir=None*, *max_merge=200*, *boundscheck=True*, *dupcheck=True*, *triucheck=True*, *ensure_sorted=False*, *h5opts=None*, *lock=None*, ***kwargs*)

Create a single-cell (scool) file.

For each cell store a cooler matrix under **/cells**, where all matrices have the same dimensions.

Each cell is a regular cooler data collection, so the input must be a bin table and pixel table for each cell. The pixel tables are provided as a dictionary where the key is a unique cell name. The bin tables can be provided as a dict with the same keys or a single common bin table can be given.

New in version 0.8.9.

Parameters

- **cool_uri** (*str*) – Path to scool file or URI string. If the file does not exist, it will be created.
- **bins** (pandas.DataFrame or Dict[str, DataFrame]) – A single bin table or dictionary of cell names to bins tables. A bin table is a dataframe with columns chrom, start and end. May contain additional columns.
- **cell_name_pixels_dict** (*Dict[str, DataFrame]*) – Cell name as key and pixel table DataFrame as value. A table, given as a dataframe or a column-oriented dict, containing columns labeled bin1_id, bin2_id and count, sorted by (bin1_id, bin2_id). If additional columns are included in the pixel table, their names and dtypes must be specified using the columns and dtypes arguments. For larger input data, an **iterable** can be provided that yields the pixel data as a sequence of chunks. If the input is a dask DataFrame, it will also be processed one chunk at a time.
- **columns** (*sequence of str, optional*) – Customize which value columns from the input pixels to store in the cooler. Non-standard value columns will be given dtype float64 unless overriden using the dtypes argument. If None, we only attempt to store a value column named "count".
- **dtypes** (*dict, optional*) – Dictionary mapping column names to dtypes. Can be used to override the default dtypes of bin1_id, bin2_id or count or assign dtypes to custom value columns. Non-standard value columns given in dtypes must also be provided in the columns argument or they will be ignored.
- **metadata** (*dict, optional*) – Experiment metadata to store in the file. Must be JSON compatible.
- **assembly** (*str, optional*) – Name of genome assembly.
- **ordered** (*bool, optional [default: False]*) – If the input chunks of pixels are provided with correct triangularity and in ascending order of (bin1_id, bin2_id), set this to True to write the cooler in one step. If False (default), we create the cooler in two steps using an external sort mechanism. See Notes for more details.
- **symmetric_upper** (*bool, optional [default: True]*) – If True, sets the file's storage-mode property to symmetric-upper: use this only if the input data references the upper triangle of a symmetric matrix! For all other cases, set this option to False.
- **mode** (*{'w' , 'a'}, optional [default: 'w']*) – Write mode for the output file. 'a': if the output file exists, append the new cooler to it. 'w': if the output file exists, it will be truncated. Default is 'w'.

Other Parameters

- **mergebuf** (*int, optional*) – Maximum number of records to buffer in memory at any give time during the merge step.
- **delete_temp** (*bool, optional*) – Whether to delete temporary files when finished. Useful for debugging. Default is False.
- **temp_dir** (*str, optional*) – Create temporary files in a specified directory instead of the same directory as the output file. Pass – to use the system default.
- **max_merge** (*int, optional*) – If merging more than `max_merge` chunks, do the merge recursively in two passes.
- **h5opts** (*dict, optional*) – HDF5 dataset filter options to use (compression, shuffling, checksumming, etc.). Default is to use autochunking and GZIP compression, level 6.
- **lock** (*multiprocessing.Lock, optional*) – Optional lock to control concurrent access to the output file.
- **ensure_sorted** (*bool, optional*) – Ensure that each input chunk is properly sorted.
- **boundscheck** (*bool, optional*) – Input validation: Check that all bin IDs lie in the expected range.
- **dupcheck** (*bool, optional*) – Input validation: Check that no duplicate pixels exist within any chunk.
- **triucheck** (*bool, optional*) – Input validation: Check that `bin1_id` <= `bin2_id` when creating coolers in symmetric-upper mode.

See also:

cooler.create_cooler(), *cooler.zoomify_cooler()*

Notes

If the pixel chunks are provided in the correct order required for the output to be properly sorted, then the cooler can be created in a single step by setting `ordered=True`.

If not, the cooler is created in two steps via an external sort mechanism. In the first pass, the sequence of pixel chunks are processed and sorted in memory and saved to temporary coolers. In the second pass, the temporary coolers are merged into the output file. This way the individual chunks do not need to be provided in any particular order. When `ordered=False`, the following options for the merge step are available: `mergebuf`, `delete_temp`, `temp_dir`, `max_merge`.

Each chunk of pixels will go through a validation pipeline, which can be customized with the following options: `boundscheck`, `triucheck`, `dupcheck`, `ensure_sorted`.

5.3 cooler.create

`cooler.create.`**`sanitize_pixels`**(*bins*, ***kwargs*)

Builds a function to sanitize an already-binned genomic data with genomic bin assignments.

Parameters

- **`bins`** (*`DataFrame`*) – Bin table to compare pixel records against.
- **`is_one_based`** (*`bool, optional`*) – Whether the input bin IDs are one-based, rather than zero-based. They will be converted to zero-based.

- **tril_action** (*'reflect', 'drop', 'raise' or None*) – How to handle lower triangle ("tril") pixels. If set to 'reflect' [default], tril pixels will be flipped or "reflected" to their mirror image: "sided" column pairs will have their values swapped. If set to 'drop', tril pixels will be discarded. This is useful if your input data is duplexed, i.e. contains mirror duplicates of every record. If set to 'raise', an exception will be raised if any tril record is encountered.
- **bin1_field** (*str*) – Name of the column representing ith (row) axis of the matrix. Default is 'bin1_id'.
- **bin2_field** (*str*) – Name of the column representing jth (col) axis of the matrix. Default is 'bin2_id'.
- **sided_fields** (*sequence of str*) – Base names of column pairs to swap values between when mirror-reflecting pixels.
- **suffixes** (*pair of str*) – Suffixes used to identify pairs of sided columns. e.g.: ('1', '2'), ('_x', '_y'), etc.
- **sort** (*bool*) – Whether to sort the output dataframe by bin_id and bin2_id.

Returns *callable* – Function of one argument that takes a raw dataframe and returns a sanitized dataframe.

cooler.create.**sanitize_records**(*bins*, *schema=None*, ***kwargs*)

Builds a funtion to sanitize and assign bin IDs to a data frame of paired genomic positions based on a provided genomic bin segmentation.

Parameters

- **bins** (*DataFrame*) – Bin table to compare records against.
- **schema** (*str, optional*) – Use pre-defined parameters for a particular format. Any options can be overriden via kwargs. If not provided, values for all the options below must be given.
- **decode_chroms** (*bool*) – Convert string chromosome names to integer IDs based on the order given in the bin table. Set to False if the chromosomes are already given as an enumeration, starting at 0. Records with either chrom ID < 0 are dropped.
- **is_one_based** (*bool*) – Whether the input anchor coordinates are one-based, rather than zero-based. They will be converted to zero-based.
- **tril_action** (*'reflect', 'drop', 'raise' or None*) – How to handle lower triangle ("tril") records. If set to 'reflect', tril records will be flipped or "reflected" to their mirror image: "sided" column pairs will have their values swapped. If set to 'drop', tril records will be discarded. This is useful if your input data is symmetric, i.e. contains mirror duplicates of every record. If set to 'raise', an exception will be raised if any tril record is encountered.
- **chrom_field** (*str*) – Base name of the two chromosome/scaffold/contig columns.
- **anchor_field** (*str*) – Base name of the positional anchor columns.
- **sided_fields** (*sequence of str*) – Base names of column pairs to swap values between when mirror-reflecting records.
- **suffixes** (*pair of str*) – Suffixes used to identify pairs of sided columns. e.g.: ('1', '2'), ('_x', '_y'), etc.
- **sort** (*bool*) – Whether to sort the output dataframe by bin_id and bin2_id.

- **validate** (*bool*) – Whether to do type- and bounds-checking on the anchor position columns. Raises BadInputError.

Returns *callable* – Function of one argument that takes a raw dataframe and returns a sanitized dataframe with bin IDs assigned.

5.4 cooler.fileops

cooler.fileops.**is_cooler**(*uri*)
Determine if a URI string references a cooler data collection. Returns False if the file or group path doesn't exist.

cooler.fileops.**is_multires_file**(*filepath*, *min_version=1*)
Determine if a file is a multi-res cooler file. Returns False if the file doesn't exist.

cooler.fileops.**list_coolers**(*filepath*)
List group paths to all cooler data collections in a file.

Parameters **filepath** (*str*) –

Returns *list* – Cooler group paths in the file.

cooler.fileops.**cp**(*src_uri*, *dst_uri*, *overwrite=False*)
Copy a group or dataset from one file to another or within the same file.

cooler.fileops.**mv**(*src_uri*, *dst_uri*, *overwrite=False*)
Rename a group or dataset within the same file.

cooler.fileops.**ln**(*src_uri*, *dst_uri*, *soft=False*, *overwrite=False*)
Create a hard link to a group or dataset in the same file. Also supports soft links (in the same file) or external links (different files).

5.5 cooler.util

cooler.util.**partition**(*start*, *stop*, *step*)
Partition an integer interval into equally-sized subintervals. Like builtin `range()`, but yields pairs of end points.

Examples

```
>>> for lo, hi in partition(0, 9, 2):
        print(lo, hi)
0 2
2 4
4 6
6 8
8 9
```

cooler.util.**fetch_chromsizes**(*db*, ***kwargs*)
Download chromosome sizes from UCSC as a `pandas.Series`, indexed by chromosome label.

cooler.util.**read_chromsizes**(*filepath_or*, *name_patterns=('^chr[0-9]+$', '^chr[XY]$', '^chrM$')*, *all_names=False*, ***kwargs*)
Parse a `<db>.chrom.sizes` or `<db>.chromInfo.txt` file from the UCSC database, where `db` is a genome assembly name.

Parameters

- **filepath_or** (*str or file-like*) – Path or url to text file, or buffer.
- **name_patterns** (*sequence, optional*) – Sequence of regular expressions to capture desired sequence names. Each corresponding set of records will be sorted in natural order.
- **all_names** (*bool, optional*) – Whether to return all contigs listed in the file. Default is `False`.

Returns `pandas.Series` – Series of integer bp lengths indexed by sequence name.

References

- UCSC assembly terminology
- GRC assembly terminology

`cooler.util.`**binnify**(*chromsizes*, *binsize*)

Divide a genome into evenly sized bins.

Parameters

- **chromsizes** (*Series*) – pandas Series indexed by chromosome name with chromosome lengths in bp.
- **binsize** (*int*) – size of bins in bp

Returns **bins** (`pandas.DataFrame`) – Dataframe with columns: `chrom`, `start`, `end`.

`cooler.util.`**digest**(*fasta_records*, *enzyme*)

Divide a genome into restriction fragments.

Parameters

- **fasta_records** (*OrderedDict*) – Dictionary of chromosome names to sequence records.
- **enzyme** (*str*) – Name of restriction enzyme (e.g., 'DpnII').

Returns **frags** (`pandas.DataFrame`) – Dataframe with columns: `chrom`, `start`, `end`.

5.6 cooler.sandbox

CHAPTER 6

CLI Reference

6.1 Quick reference

```
cooler [OPTIONS] COMMAND [ARGS]...
```

Data ingest	
cooler cload	Create a cooler from genomic point pairs and bins.
cooler load	Create a cooler from a pre-binned matrix.

Reduction	
cooler merge	Merge multiple coolers with identical axes.
cooler coarsen	Coarsen a cooler to a lower resolution.
cooler zoomify	Generate a multi-resolution cooler file by coarsening.

Normalization	
cooler balance	Out-of-core matrix balancing.

Export/visualization	
cooler info	Display a cooler's info and metadata.
cooler dump	Dump a cooler's data to a text stream.
cooler show	Display and browse a cooler with matplotlib.

File manipulation/info	
cooler tree	Display a file's data hierarchy.
cooler attrs	Display a file's attribute hierarchy.
cooler ls	List all coolers inside a file.
cooler cp	Copy a cooler from one file to another or within the same file.
cooler mv	Rename a cooler within the same file.
cooler ln	Create a hard, soft or external link to a cooler.

Helper commands	
cooler makebins	Generate fixed-width genomic bins.
cooler digest	Generate fragment-delimited genomic bins.
cooler csort	Sort and index a contact list.

Options

-v, --verbose
Verbose logging.

-d, --debug
On error, drop into the post-mortem debugger shell.

-V, --version
Show the version and exit.

See the cooler_cli.ipynb Jupyter Notebook for specific examples on usage: (https://github.com/mirnylab/cooler-binder).

6.2 cooler cload

Create a cooler from genomic pairs and bins.

Choose a subcommand based on the format of the input contact list.

```
cooler cload [OPTIONS] COMMAND [ARGS]...
```

Commands

- **hiclib**
- **pairix**
- **pairs**
- **tabix**

6.3 cooler cload pairs

Bin any text file or stream of pairs.

Pairs data need not be sorted. Accepts compressed files. To pipe input from stdin, set PAIRS_PATH to '-'.

BINS : One of the following

<TEXT:INTEGER> : 1. Path to a chromsizes file, 2. Bin size in bp

<TEXT> : Path to BED file defining the genomic bin segmentation.

PAIRS_PATH : Path to contacts (i.e. read pairs) file.

COOL_PATH : Output COOL file path or URI.

```
cooler cload pairs [OPTIONS] BINS PAIRS_PATH COOL_PATH
```

Arguments

BINS
Required argument

PAIRS_PATH
Required argument

COOL_PATH
Required argument

Options

--metadata <metadata>
Path to JSON file containing user metadata.

--assembly <assembly>
Name of genome assembly (e.g. hg19, mm10)

-c1, --chrom1 <chrom1>
chrom1 field number (one-based) [required]

-p1, --pos1 <pos1>
pos1 field number (one-based) [required]

-c2, --chrom2 <chrom2>
chrom2 field number (one-based) [required]

-p2, --pos2 <pos2>
pos2 field number (one-based) [required]

--chunksize <chunksize>
Number of input lines to load at a time

-0, --zero-based
Positions are zero-based [default: False]

--comment-char <comment_char>
Comment character that indicates lines to ignore. [default: #]

-N, --no-symmetric-upper
Create a complete square matrix without implicit symmetry. This allows for distinct upper- and lower-triangle values

--input-copy-status <input_copy_status>
Copy status of input data when using symmetric-upper storage. | *unique*: Incoming data comes from a unique half of a symmetric map, regardless of how the coordinates of a pair are ordered. *duplex*: Incoming data contains upper- and lower-triangle duplicates. All input records that map to the lower triangle will be discarded! | If you wish to treat lower- and upper-triangle input data as distinct, use the `--no-symmetric-upper` option. [default: unique]

--field <field>
Specify quantitative input fields to aggregate into value columns using the syntax `--field <field-name>=<field-number>`. Optionally, append `:` followed by `dtype=<dtype>` to specify the data type (e.g. float), and/or `agg=<agg>` to specify an aggregation function different from sum (e.g. mean). Field numbers are 1-based. Passing 'count' as the target name will override the default behavior of storing pair counts. Repeat the `--field` option for each additional field.

--temp-dir <temp_dir>
Create temporary files in a specified directory. Pass - to use the platform default temp dir.

--no-delete-temp
Do not delete temporary files when finished.

--max-merge <max_merge>
Maximum number of chunks to merge before invoking recursive merging [default: 200]

--storage-options <storage_options>
Options to modify the data filter pipeline. Provide as a comma-separated list of key-value pairs of the form 'k1=v1,k2=v2,...'. See http://docs.h5py.org/en/stable/high/dataset.html#filter-pipeline for more details.

6.4 cooler cload pairix

Bin a pairix-indexed contact list file.

BINS : One of the following

<TEXT:INTEGER> : 1. Path to a chromsizes file, 2. Bin size in bp

<TEXT> : Path to BED file defining the genomic bin segmentation.

PAIRS_PATH : Path to contacts (i.e. read pairs) file.

COOL_PATH : Output COOL file path or URI.

See also: 'cooler csort' to sort and index a contact list file

Pairix on GitHub: <https://github.com/4dn-dcic/pairix>.

```
cooler cload pairix [OPTIONS] BINS PAIRS_PATH COOL_PATH
```

Arguments

BINS
Required argument

PAIRS_PATH
Required argument

COOL_PATH
Required argument

Options

--metadata <metadata>
Path to JSON file containing user metadata.

--assembly <assembly>
Name of genome assembly (e.g. hg19, mm10)

-p, --nproc <nproc>
Number of processes to split the work between. [default: 8]

-0, --zero-based
Positions are zero-based [default: False]

-s, --max-split <max_split>
Divide the pairs from each chromosome into at most this many chunks. Smaller chromosomes will be split less frequently or not at all. Increase ths value if large chromosomes dominate the workload on multiple processors. [default: 2]

6.5 cooler cload tabix

Bin a tabix-indexed contact list file.

BINS : One of the following

<TEXT:INTEGER> : 1. Path to a chromsizes file, 2. Bin size in bp

<TEXT> : Path to BED file defining the genomic bin segmentation.

PAIRS_PATH : Path to contacts (i.e. read pairs) file.

COOL_PATH : Output COOL file path or URI.

See also: 'cooler csort' to sort and index a contact list file

Tabix manpage: <http://www.htslib.org/doc/tabix.html>.

```
cooler cload tabix [OPTIONS] BINS PAIRS_PATH COOL_PATH
```

Arguments

BINS
Required argument

PAIRS_PATH
Required argument

COOL_PATH
Required argument

Options

--metadata <metadata>
Path to JSON file containing user metadata.

--assembly <assembly>
Name of genome assembly (e.g. hg19, mm10)

-p, --nproc <nproc>
Number of processes to split the work between. [default: 8]

-c2, --chrom2 <chrom2>
chrom2 field number (one-based)

-p2, --pos2 <pos2>
pos2 field number (one-based)

-0, --zero-based
Positions are zero-based [default: False]

-s, --max-split <max_split>
Divide the pairs from each chromosome into at most this many chunks. Smaller chromosomes will be split less frequently or not at all. Increase ths value if large chromosomes dominate the workload on multiple processors. [default: 2]

6.6 cooler cload hiclib

Bin a hiclib HDF5 contact list (frag) file.

BINS : One of the following

<TEXT:INTEGER> : 1. Path to a chromsizes file, 2. Bin size in bp

<TEXT> : Path to BED file defining the genomic bin segmentation.

PAIRS_PATH : Path to contacts (i.e. read pairs) file.

COOL_PATH : Output COOL file path or URI.

hiclib on BitBucket: <https://bitbucket.org/mirnylab/hiclib>.

```
cooler cload hiclib [OPTIONS] BINS PAIRS_PATH COOL_PATH
```

Arguments

BINS
Required argument

PAIRS_PATH
Required argument

COOL_PATH
Required argument

Options

--metadata <metadata>
Path to JSON file containing user metadata.

--assembly <assembly>
Name of genome assembly (e.g. hg19, mm10)

-c, --chunksize <chunksize>
Control the number of pixels handled by each worker process at a time. [default: 100000000]

6.7 cooler load

Create a cooler from a pre-binned matrix.

BINS_PATH : One of the following

<TEXT:INTEGER> : 1. Path to a chromsizes file, 2. Bin size in bp

<TEXT> : Path to BED file defining the genomic bin segmentation.

PIXELS_PATH : Text file containing nonzero pixel values. May be gzipped. Pass '-' to use stdin.

COOL_PATH : Output COOL file path or URI.

Notes

Two input format options (tab-delimited). Input pixel file may be compressed.

COO: COO-rdinate sparse matrix format (a.k.a. ijv triple). 3 columns: "bin1_id, bin2_id, count",

BG2: 2D version of the bedGraph format. 7 columns: "chrom1, start1, end1, chrom2, start2, end2, count"

Examples

cooler load -f bg2 <chrom.sizes>:<binsize> in.bg2.gz out.cool

```
cooler load [OPTIONS] BINS_PATH PIXELS_PATH COOL_PATH
```

Arguments

BINS_PATH
Required argument

PIXELS_PATH
Required argument

COOL_PATH
Required argument

Options

-f, --format <format>
'coo' refers to a tab-delimited sparse triplet file (bin1, bin2, count). 'bg2' refers to a 2D bedGraph-like file (chrom1, start1, end1, chrom2, start2, end2, count). [required]

--metadata <metadata>
Path to JSON file containing user metadata.

--assembly <assembly>
Name of genome assembly (e.g. hg19, mm10)

--field <field>
Add supplemental value fields or override default field numbers for the specified format. Specify quantitative input fields to aggregate into value columns using the syntax --field <field-name>=<field-number>. Optionally, append : followed by dtype=<dtype> to specify the data type (e.g. float). Field numbers are 1-based. Repeat the --field option for each additional field.

-c, --chunksize <chunksize>
Size (in number of lines/records) of data chunks to read and process from the input file at a time. These chunks will be saved as temporary partial Coolers and merged at the end. Also specifies the size of the buffer during the merge step.

--count-as-float
Store the 'count' column as floating point values instead of as integers. Can also be specified using the *–field* option.

--one-based
Pass this flag if the bin IDs listed in a COO file are one-based instead of zero-based.

--comment-char <comment_char>
Comment character that indicates lines to ignore. [default: #]

-N, --no-symmetric-upper
Create a complete square matrix without implicit symmetry. This allows for distinct upper- and lower-triangle values

--input-copy-status <input_copy_status>
Copy status of input data when using symmetric-upper storage. | *unique*: Incoming data comes from a unique half of a symmetric matrix, regardless of how element coordinates are ordered. Execution will be aborted if duplicates are detected. *duplex*: Incoming data contains upper- and lower-triangle duplicates. All lower-triangle input elements will be discarded! | If you wish to treat lower- and upper-triangle input data as distinct, use the --no-symmetric-upper option instead. [default: unique]

--storage-options <storage_options>
Options to modify the data filter pipeline. Provide as a comma-separated list of key-value pairs of the form 'k1=v1,k2=v2,...'. See http://docs.h5py.org/en/stable/high/dataset.html#filter-pipeline for more details.

6.8 cooler merge

Merge multiple coolers with identical axes.

OUT_PATH : Output file path or URI.

IN_PATHS : Input file paths or URIs of coolers to merge.

Notes

Data columns merged:

pixels/bin1_id, pixels/bin2_id, pixels/<value columns>

Data columns preserved:

chroms/name, chroms/length bins/chrom, bins/start, bins/end

Additional columns in the the input files are not transferred to the output.

```
cooler merge [OPTIONS] OUT_PATH [IN_PATHS]...
```

Arguments

OUT_PATH
Required argument

IN_PATHS
Optional argument(s)

Options

-c, --chunksize <chunksize>
Size of the merge buffer in number of pixel table rows. [default: 20000000]

--field <field>
Specify the names of value columns to merge as '<name>'. Repeat the *–field* option for each one. Use '<name>,dtype=<dtype>' to specify the dtype. Include ',agg=<agg>' to specify an aggregation function different from 'sum'.

6.9 cooler coarsen

Coarsen a cooler to a lower resolution.

Works by pooling *k*-by-*k* neighborhoods of pixels and aggregating. Each chromosomal block is coarsened individually.

COOL_PATH : Path to a COOL file or Cooler URI.

```
cooler coarsen [OPTIONS] COOL_PATH
```

Arguments

COOL_PATH
Required argument

Options

-k, --factor <factor>
Gridding factor. The contact matrix is coarsegrained by grouping each chromosomal contact block into k-by-k element tiles [default: 2]

-n, -p, --nproc <nproc>
Number of processes to use for batch processing chunks of pixels [default: 1, i.e. no process pool]

-c, --chunksize <chunksize>
Number of pixels allocated to each process [default: 10000000]

--field <field>
Specify the names of value columns to merge as '<name>'. Repeat the *–field* option for each one. Use '<name>,dtype=<dtype>' to specify the dtype. Include ',agg=<agg>' to specify an aggregation function different from 'sum'.

-o, --out <out>
Output file or URI [required]

6.10 cooler zoomify

Generate a multi-resolution cooler file by coarsening.

COOL_PATH : Path to a COOL file or Cooler URI.

```
cooler zoomify [OPTIONS] COOL_PATH
```

Arguments

COOL_PATH
Required argument

Options

-n, -p, --nproc <nproc>
Number of processes to use for batch processing chunks of pixels [default: 1, i.e. no process pool]

-c, --chunksize <chunksize>
Number of pixels allocated to each process [default: 10000000]

-r, --resolutions <resolutions>
Comma-separated list of target resolutions. Use suffixes B or N to specify a progression: B for binary (geometric steps of factor 2), N for nice (geometric steps of factor 10 interleaved with steps of 2 and 5). Examples: 1000B=1000,2000,4000,8000,... 1000N=1000,2000,5000,10000,... 5000N=5000,10000,25000,50000,... 4DN is an alias for 1000,2000,5000N [default: B]

--balance
Apply balancing to each zoom level. Off by default.

--balance-args <balance_args>
Additional arguments to pass to cooler balance

-i, --base-uri <base_uri>
One or more additional base coolers to aggregate from, if needed.

-o, --out <out>
Output file or URI

--field <field>
Specify the names of value columns to merge as '<name>'. Repeat the *–field* option for each one. Use '<name>:dtype=<dtype>' to specify the dtype. Include ',agg=<agg>' to specify an aggregation function different from 'sum'.

--legacy
Use the legacy layout of integer-labeled zoom levels.

6.11 cooler balance

Out-of-core matrix balancing.

Matrix must be symmetric. See the help for various filtering options to mask out poorly mapped bins.

COOL_PATH : Path to a COOL file.

```
cooler balance [OPTIONS] COOL_PATH
```

Arguments

COOL_PATH
Required argument

Options

-p, --nproc <nproc>
Number of processes to split the work between. [default: 8]

-c, --chunksize <chunksize>
Control the number of pixels handled by each worker process at a time. [default: 10000000]

--mad-max <mad_max>
Ignore bins from the contact matrix using the 'MAD-max' filter: bins whose log marginal sum is less than `mad-max` median absolute deviations below the median log marginal sum of all the bins in the same chromosome. [default: 5]

--min-nnz <min_nnz>
Ignore bins from the contact matrix whose marginal number of nonzeros is less than this number. [default: 10]

--min-count <min_count>
Ignore bins from the contact matrix whose marginal count is less than this number. [default: 0]

--blacklist <blacklist>
Path to a 3-column BED file containing genomic regions to mask out during the balancing procedure, e.g. sequence gaps or regions of poor mappability.

--ignore-diags <ignore_diags>
Number of diagonals of the contact matrix to ignore, including the main diagonal. Examples: 0 ignores nothing, 1 ignores the main diagonal, 2 ignores diagonals (-1, 0, 1), etc. [default: 2]

--ignore-dist <ignore_dist>
Distance in bp to ignore.

--tol <tol>
Threshold value of variance of the marginals for the algorithm to converge. [default: 1e-05]

--max-iters <max_iters>
Maximum number of iterations to perform if convergence is not achieved. [default: 200]

--cis-only
Calculate weights against intra-chromosomal data only instead of genome-wide.

--trans-only
Calculate weights against inter-chromosomal data only instead of genome-wide.

--name <name>
Name of column to write to. [default: weight]

-f, --force
Overwrite the target dataset, 'weight', if it already exists.

--check
Check whether a data column 'weight' already exists.

--stdout
Print weight column to stdout instead of saving to file.

--convergence-policy <convergence_policy>
What to do with weights when balancing doesn't converge in max_iters. 'store_final': Store the final result, regardless of whether the iterations converge to the specified tolerance; 'store_nan': Store a vector of NaN values to indicate that the matrix failed to converge; 'discard': Store nothing and exit gracefully; 'error': Abort with non-zero exit status. [default: store_final]

6.12 cooler info

Display a cooler's info and metadata.

COOL_PATH : Path to a COOL file or cooler URI.

```
cooler info [OPTIONS] COOL_PATH
```

Arguments

COOL_PATH
Required argument

Options

-f, --field <field>
Print the value of a specific info field.

-m, --metadata
Print the user metadata in JSON format.

-o, --out <out>
Output file (defaults to stdout)

6.13 cooler dump

Dump a cooler's data to a text stream.

COOL_PATH : Path to COOL file or cooler URI.

```
cooler dump [OPTIONS] COOL_PATH
```

Arguments

COOL_PATH
Required argument

Options

-t, --table <table>
Which table to dump. Choosing 'chroms' or 'bins' will cause all pixel-related options to be ignored. Note that for coolers stored in symmetric-upper mode, 'pixels' only holds the upper triangle values of the matrix. [default: pixels]

-c, --columns <columns>
Restrict output to a subset of columns, provided as a comma-separated list.

-H, --header
Print the header of column names as the first row. [default: False]

--na-rep <na_rep>
Missing data representation. Default is empty ''.

--float-format <float_format>
Format string for floating point numbers (e.g. '.12g', '03.2f'). [default: g]

-r, --range <range>
The coordinates of a genomic region shown along the row dimension, in UCSC-style notation. (Example: chr1:10,000,000-11,000,000). If omitted, the entire contact matrix is printed.

-r2, --range2 <range2>
The coordinates of a genomic region shown along the column dimension. If omitted, the column range is the same as the row range.

-m, --matrix
For coolers stored in symmetric-upper mode, ensure any empty areas of the genomic query window are populated by generating the lower-triangular pixels. [default: False]

-b, --balanced, --no-balance
Apply balancing weights to data. This will print an extra column called *balanced* [default: False]

--join
Print the full chromosome bin coordinates instead of bin IDs. This will replace the *bin1_id* column with *chrom1*, *start1*, and *end1*, and the *bin2_id* column with *chrom2*, *start2* and *end2*. [default: False]

--annotate <annotate>
Join additional columns from the bin table against the pixels. Provide a comma separated list of column names (no spaces). The merged columns will be suffixed by '1' and '2' accordingly.

--one-based-ids
Print bin IDs as one-based rather than zero-based.

--one-based-starts
Print start coordinates as one-based rather than zero-based.

-k, --chunksize <chunksize>
Sets the amount of pixel data loaded from disk at one time. Can affect the performance of joins on high resolution datasets. Default is to load as many rows as there are bins.

-o, --out <out>
Output text file If .gz extension is detected, file is written using zlib. Default behavior is to stream to stdout.

6.14 cooler show

Display and browse a cooler in matplotlib.

COOL_PATH : Path to a COOL file or Cooler URI.

RANGE : The coordinates of the genomic region to display, in UCSC notation. Example: chr1:10,000,000-11,000,000

```
cooler show [OPTIONS] COOL_PATH RANGE
```

Arguments

COOL_PATH
Required argument

RANGE
Required argument

Options

-r2, --range2 <range2>
The coordinates of a genomic region shown along the column dimension. If omitted, the column range is the same as the row range. Use to display asymmetric matrices or trans interactions.

-b, --balanced
Show the balanced contact matrix. If not provided, display the unbalanced counts.

-o, --out <out>
Save the image of the contact matrix to a file. If not specified, the matrix is displayed in an interactive window. The figure format is deduced from the extension of the file, the supported formats are png, jpg, svg, pdf, ps and eps.

--dpi <dpi>
The DPI of the figure, if saving to a file

-s, --scale <scale>
Scale transformation of the colormap: linear, log2 or log10. Default is log10.

-f, --force
Force display very large matrices (>=10^8 pixels). Use at your own risk as it may cause performance issues.

--zmin <zmin>
The minimal value of the color scale. Units must match those of the colormap scale. To provide a negative value use a equal sign and quotes, e.g. -zmin='-0.5'

--zmax <zmax>
The maximal value of the color scale. Units must match those of the colormap scale. To provide a negative value use a equal sign and quotes, e.g. -zmax='-0.5'

--cmap <cmap>
The colormap used to display the contact matrix. See the full list at http://matplotlib.org/examples/color/colormaps_reference.html

--field <field>
Pixel values to display. [default: count]

6.15 cooler tree

Display a file's data hierarchy.

```
cooler tree [OPTIONS] URI
```

Arguments

URI
Required argument

Options

-L, --level <level>

6.16 cooler attrs

Display a file's attribute hierarchy.

```
cooler attrs [OPTIONS] URI
```

Arguments

URI
Required argument

Options

-L, --level <level>

6.17 cooler ls

List all coolers inside a file.

```
cooler ls [OPTIONS] COOL_PATH
```

Arguments

COOL_PATH
Required argument

Options

-l, --long
Long listing format

6.18 cooler cp

Copy a cooler from one file to another or within the same file.

See also: h5copy, h5repack tools from HDF5 suite.

```
cooler cp [OPTIONS] SRC_URI DST_URI
```

Arguments

SRC_URI
Required argument

DST_URI
Required argument

Options

-w, --overwrite
Truncate and replace destination file if it already exists.

6.19 cooler mv

Rename a cooler within the same file.

```
cooler mv [OPTIONS] SRC_URI DST_URI
```

Arguments

SRC_URI
Required argument

DST_URI
Required argument

Options

-w, --overwrite
Truncate and replace destination file if it already exists.

6.20 cooler ln

Create a hard link to a cooler (rather than a true copy) in the same file. Also supports soft links (in the same file) or external links (different files).

```
cooler ln [OPTIONS] SRC_URI DST_URI
```

Arguments

SRC_URI
Required argument

DST_URI
Required argument

Options

-w, --overwrite
Truncate and replace destination file if it already exists.

-s, --soft
Creates a soft link rather than a hard link if the source and destination file are the same. Otherwise, creates an external link. This type of link uses a path rather than a pointer.

6.21 cooler makebins

Generate fixed-width genomic bins.

Output a genome segmentation at a fixed resolution as a BED file.

CHROMSIZES_PATH : UCSC-like chromsizes file, with chromosomes in desired order.

BINSIZE : Resolution (bin size) in base pairs <int>.

```
cooler makebins [OPTIONS] CHROMSIZES_PATH BINSIZE
```

Arguments

CHROMSIZES_PATH
Required argument

BINSIZE
Required argument

Options

-o, --out <out>
Output file (defaults to stdout)

-H, --header
Print the header of column names as the first row. [default: False]

-i, --rel-ids <rel_ids>
Include a column of relative bin IDs for each chromosome. Choose whether to report them as 0- or 1-based.

6.22 cooler digest

Generate fragment-delimited genomic bins.

Output a genome segmentation of restriction fragments as a BED file.

CHROMSIZES_PATH : UCSC-like chromsizes file, with chromosomes in desired order.

FASTA_PATH : Genome assembly FASTA file or folder containing FASTA files (uncompressed).

ENZYME : Name of restriction enzyme

```
cooler digest [OPTIONS] CHROMSIZES_PATH FASTA_PATH ENZYME
```

Arguments

CHROMSIZES_PATH
Required argument

FASTA_PATH
Required argument

ENZYME
Required argument

Options

-o, --out <out>
Output file (defaults to stdout)

-H, --header
Print the header of column names as the first row. [default: False]

-i, --rel-ids <rel_ids>
Include a column of relative bin IDs for each chromosome. Choose whether to report them as 0- or 1-based.

6.23 cooler csort

Sort and index a contact list.

Order the mates of each pair record so that all contacts are upper triangular with respect to the chromosome ordering given by the chromosomes file, sort contacts by genomic location, and index the resulting file.

PAIRS_PATH : Contacts (i.e. read pairs) text file, optionally compressed.

CHROMOSOMES_PATH : File listing desired chromosomes in the desired order. May be tab-delimited, e.g. a UCSC-style chromsizes file. Contacts mapping to other chromosomes will be discarded.

Notes

- csort can also be used to sort and index a text representation of
  a contact *matrix* in bedGraph-like format. In this case, substitute
  pos1 and *pos2* with *start1* and *start2*, respectively.
- Requires Unix tools: sort, bgzip + tabix or pairix.

If indexing with Tabix, the output file will have the following properties:

- Upper triangular: the read pairs on each row are assigned to side 1 or 2
  in such a way that (chrom1, pos1) is always "less than" (chrom2, pos2)
- Rows are lexicographically sorted by chrom1, pos1, chrom2, pos2;
  i.e. "positionally sorted"
- Compressed with bgzip [*]
- Indexed using Tabix [*] on chrom1 and pos1.

If indexing with Pairix, the output file will have the following properties:

- Upper triangular: the read pairs on each row are assigned to side 1 or 2
  in such a way that (chrom1, pos1) is always "less than" (chrom2, pos2)
- Rows are lexicographically sorted by chrom1, chrom2, pos1, pos2; i.e.
  "block sorted"
- Compressed with bgzip [*]
- Indexed using Pairix [+] on chrom1, chrom2 and pos1.

[*] Tabix manpage: <http://www.htslib.org/doc/tabix.html>.
[+] Pairix on Github: <https://github.com/4dn-dcic/pairix>

```
cooler csort [OPTIONS] PAIRS_PATH CHROMOSOMES_PATH
```

Arguments

PAIRS_PATH
Required argument

CHROMOSOMES_PATH
Required argument

Options

-c1, --chrom1 <chrom1>
chrom1 field number in the input file (starting from 1) [required]

-c2, --chrom2 <chrom2>
chrom2 field number [required]

-p1, --pos1 <pos1>
pos1 field number [required]

-p2, --pos2 <pos2>
pos2 field number [required]

-i, --index <index>
Select the preset sort and indexing options [default: pairix]

--flip-only
Only flip mates; no sorting or indexing. Write to stdout. [default: False]

-p, --nproc <nproc>
Number of processors [default: 8]

-0, --zero-based
Read positions are zero-based [default: False]

--sep <sep>
Data delimiter in the input file [default: t]

--comment-char <comment_char>
Comment character to skip header [default: #]

--sort-options <sort_options>
Quoted list of additional options to *sort* command

-o, --out <out>
Output gzip file

-s1, --strand1 <strand1>
strand1 field number (deprecated)

-s2, --strand2 <strand2>
strand2 field number (deprecated)

CHAPTER 7

Release notes

7.1 v0.8.10

Date : 2020-09-25

7.1.1 Bug fixes

- Fixed the new header parsing in `cooler cload pairs` to handle esoteric file stream implementations. Specifically `GzipFile` had stopped working. By @golobor

7.2 v0.8.9

Date : 2020-07-17

7.2.1 Enhancements

- Added single-cell cooler file flavor (.scool) (#201)

7.3 v0.8.8

Date : 2020-06-23

7.3.1 Maintenance

- Improved code coverage
- Added missing autodoc for cooler balance

- Dropped pysam and biopython as hard dependencies
- Officially sunsetting Python 2.7 support

7.3.2 Enhancements

- Added zoom progressions (#203)

7.3.3 Bug fixes

- Allow hashes in read IDs in cload pairs (#193)

7.4 v0.8.7

Date: 2020-01-12

7.4.1 Maintenance

- Code styling with black
- Add coverage reporting

7.4.2 Bug fixes

- Replace `json` with `simplejson` to deal with attrs stored as bytes

7.5 v0.8.6

Date: 2019-08-12

7.5.1 Maintenance

- Added contributing guidelines

7.5.2 Bug fixes

- Fixed a related regression that affected selection of the `chrom` column.

Post-release `v0.8.6.post0`: requirements files added to MANIFEST.in

7.6 v0.8.5

Date: 2019-04-08

7.6.1 Bug fixes

- Fixed a regression that prevented selection of bins excluding the `chrom` column.

7.7 v0.8.4

Date: 2019-04-04

7.7.1 Enhancements

- When creating coolers from unordered input, change the default temporary dir to be the same as the output file instead of the system tmp (pass '-' to use the system one). #150
- `cooler ls` and `list_coolers()` now output paths in natural order. #153
- New option in `cooler.matrix()` to handle divisive balancing weight vectors.

7.7.2 Bug fixes

- Restore function of `--count-as-float` option to `cooler load`
- Fixed partitioning issue sometimes causing some bins to get split during coarsen
- `rename_chroms()` will refresh cached chromosome names #147
- `Cooler.bins()` selector will always properly convert bins/chrom integer IDs to categorical chromosome names when the number of contigs is very large and therefore the HDF5 ENUM header is missing. Before this would only happen when explicitly requesting `convert_enum=True`.

7.8 v0.8.3

Date: 2019-02-11

7.8.1 Bug fixes

- Fixed import bug in `rename_chroms`
- `create_cooler` no longer requires a "count" column when specifying custom value columns

7.9 v0.8.2

Date: 2019-01-20

7.9.1 Enhancements

New options for `cooler dump` pixel output:

- `--matrix` option: Applies to symmetric-upper coolers; no-op for square coolers. Generates all lower triangular pixels necessary to fill the requested genomic query window. Without this option, `cooler dump` will only return the data explicity stored in the pixel table (i.e. upper triangle).
- `--one-based-ids` and `--one-based-starts` convenience options.

7.9.2 Bug fixes

- A bug was introduced into the matrix-as-pixels selector in 0.8.0 that also affected `cooler dump`. The behavior has been restored to that in 0.7.

7.10 v0.8.1

Date: 2019-01-02

7.10.1 Enhancements

- `cooler zoomify` command can take additional base resolutions as input.

7.10.2 Bug fixes

- Fixed regression that slowed down pre-processing during coarsen.
- Fixed missing import on handling bad URIs.
- Restore but deprecate `cooler.io.ls` for backwards compatibility.

7.11 v0.8.0

Date: 2018-12-31

This is a major release from 0.7 and includes an updated format version, and several API changes and deprecations.

7.11.1 Schema

- New schema version: v3
- Adds required `storage-mode` metadata attribute. Two possible values: `"symmetric-upper"` indicates a symmetric matrix encoded as upper triangle (previously the only storage mode); `"square"` indicates no special encoding (e.g. for non-symmetric matrices).

7.11.2 New features

- Support for **non-symmetric** matrices, e.g. RNA-DNA maps.
  - Create function accepts a boolean `symmetric_upper` option to set the storage mode. Default is `True`.
  - Creation commands also use `symmetric_upper` by default, which can be overridden with a flag.
- All main functionality exposed through top-level functions (create, merge, coarsen, zoomify, balance)
- New commands for generic file operations and file inspection.

7.11.3 API changes

- `cooler.annotate()` option `replace` now defaults to `False`.
- Submodule renaming. Old names are preserved as aliases but are deprecated.
  - `cooler.io` -> `cooler.create`.
  - `cooler.ice` -> `cooler.balance`.
- New top level public functions:
  - `cooler.create_cooler()`. Use instead of `cooler.io.create` and `cooler.io.create_from_unordered`.
  - `cooler.merge_coolers()`
  - `cooler.coarsen_cooler()`
  - `cooler.zoomify_cooler()`
  - `cooler.balance_cooler()`. Alias: `cooler.balance.iterative_correction()`.
- Refactored file operations available in `cooler.fileops`. See the API reference.

7.11.4 CLI changes

- Various output options added to `cooler info`, `cooler dump`, `cooler makebins` and `cooler digest`.
- Generic data and attribute hierarchy viewers `cooler tree` and `cooler attrs`.
- Generic `cp`, `mv` and `ln` convenience commands.
- New verbosity and process info options.

7.11.5 Maintenance

- Unit tests refactored and re-written for pytest.

7.12 v0.7.11

Date: 2018-08-17

- Genomic range parser supports humanized units (k/K(b), m/M(b), g/G(b))
- Experimental support for arbitrary aggregation operations in `cooler csort` (e.g. mean, median, max, min)

- Documentation updates

Bug fixes

- Fix newline handling for csort when p1 or p2 is last column.
- Fix `--count-as-float` regression in load/cload.

7.13 v0.7.10

Date: 2018-05-07

- Fix a shallow copy bug in validate pixels causing records to sometimes flip twice.
- Add ignore distance (bp) filter to cooler balance
- Start using shuffle filter by default

7.14 v0.7.9

Date: 2018-03-30

- Indexed pairs loading commands now provide option for 0- or 1-based positions (1-based by default). #115
- Fixed error introduced into cload pairix in last release.

7.15 v0.7.8

Date: 2018-03-18

7.15.1 Enhancements

- New `cooler cload pairs` command provides index-free loading of pairs.
- Changed name of `create_from_unsorted` to more correct `create_from_unordered`.

7.15.2 Bug fixes

- Fixed broken use of single-file temporary store in `create_from_unordered`.
- Added heuristic in pairix cload to prevent excessively large chunks. #92
- Added extra checks in `cload pairix` and `cload tabix`. #62, #75

7.16 v0.7.7

Date: 2018-03-16

7.16.1 Enhancements

- Implementation of unsorted (index-free) loading
  - `cooler.io.create_from_unsorted` takes an iterable of pixel dataframe chunks that need not be properly sorted.
  - Use input sanitization procedures for pairs `sanitize_records` and binned data `sanitize_pixels` to feed data to `create_from_unsorted`. #87 #108 #109
  - The `cooler load` command is now index-free: unsorted `COO` and `BG2` input data can be streamed in. #90. This will soon be implemented as an option for loading pairs as well.
- Prevent `cooler balance` command from exiting with non-zero status upon failed convergence using convergence error policies. #93
- Improve the `create` API to support pandas read_csv-style `columns` and `dtype` kwargs to add extra value columns or override default dtypes. #108
- Experimental implementation of trans-only balancing. #56

7.16.2 Bug fixes

- Fix argmax deprecation. #99

7.17 v0.7.6

Date: 2017-10-31

7.17.1 Enhancements

- Cooler zoomify with explicit resolutions
- Towards standardization of multicooler structure
- Support for loading 1-based COO triplet input files

7.17.2 Bug fixes

- Fixed issue of exceeding header limit with too many scaffolds. If header size is exceeded, chrom IDs are stored as raw integers instead of HDF5 enums. There should be no effect at the API level.
- Fixed issue of single-column chromosomes files not working in `cload`.
- Fixed edge case in performing joins when using both `as_pixels` and `join` options in the matrix selector.

Happy Halloween!

7.18 v0.7.5

Date: 2017-07-13

- Fix pandas issue affecting cases when loading single chromosomes
- Add transform options to higlass API

7.19 v0.7.4

Date: 2017-05-25

- Fix regression in automatic –balance option in cooler zoomify
- Fix special cases where cooler.io.create and append would not work with certain inputs

7.20 v0.7.3

Date: 2017-05-22

- Added function to print higlass zoom resolutions for a given genome and base resolution.

7.21 v0.7.2

Date: 2017-05-09

- Improve chunking and fix pickling issue with aggregating very large text datasets
- Restore zoom binsize metadata to higlass files

7.22 v0.7.1

Date: 2017-04-29

- `cooler load` command can now accept supplemental pixel fields and custom field numbers
- Fix parsing errors with unused pixel fields
- Eliminate hard dependence on dask to make pip installs simpler. Conda package will retain dask as a run time requirement.

7.23 v0.7.0

Date: 2017-04-27

7.23.1 New features

- New Cooler URIs: Full support for Cooler objects anywhere in the data hierarchy of a .cool file
- Experimental dask support via `cooler.contrib.dask`
- New explicit bin blacklist option for `cooler balance`
- Various new CLI tools:
  - `cooler list`
  - `cooler copy`
  - `cooler merge`

- `cooler csort` now produces Pairix files by default
- `cooler load` now accepts two types of matrix text input formats
  - 3-column sparse matrix
  - 7-column bg2.gz (2D bedGraph) indexed with Pairix (e.g. using csort)
- `cooler coarsegrain` renamed `cooler coarsen`
- Multi-resolution HiGlass input files can now be generated with the `cooler zoomify` command
- More flexible API functions to create and append columns to Coolers in `cooler.io`

API/CLI changes

- `cooler.io.create` signature changed; `chromsizes` argument is deprecated.
- `cooler csort` argument order changed

7.23.2 Bug fixes

- Chromosome name length restriction removed
- `Cooler.open` function now correctly opens the specific root group of the Cooler and behaves like a proper context manager in all cases

7.24 v0.6.6

Date: 2017-03-21

- Chromosome names longer than 32 chars are forbidden for now
- Improved pairix and tabix iterators, dropped need for slow first pass over contacts

7.25 v0.6.5

Date: 2017-03-18

- Fixed pairix aggregator to properly deal with autoflipping of pairs

7.26 v0.6.4

Date: 2017-03-17

- Migrated higlass multires aggregator to `cooler coarsegrain` command
- Fixed pairix aggregator to properly deal with autoflipping of pairs

7.27 v0.6.3

Date: 2017-02-22

- Merge PairixAggregator patch from Soo.
- Update repr string
- Return matrix scale factor in balance stats rather than the bias scale factor: #35.

7.28 v0.6.2

Date: 2017-02-12

Fixed regressions in

- cooler cload tabix/pairix failed on non-fixed sized bins
- cooler show

7.29 v0.6.1

Date: 2017-02-06

- This fixes stale build used in bdist_wheel packaging that broke 0.6.0. #29

7.30 v0.6.0

Date: 2017-02-03

7.30.1 Enhancements

- Dropped Python 3.3 support. Added 3.6 support.
- Added `contrib` subpackage containing utilities for higlass, including multires aggregation.
- Fixed various issues with synchronizing read/write multiprocessing with HDF5.
- Replacing prints with logging.
- Added sandboxed `tools` module to develop utilities for out-of-core algorithms using Coolers.

7.30.2 New features

- Cooler objects have additional convenience properties `chromsizes`, `chromnames`.
- New file introspection functions `ls` and `is_cooler` to support nested Cooler groups.
- Cooler initializer can accept a file path and path to Cooler group.
- `cload` accepts contact lists in hiclib-style HDF5 format, the legacy tabix-indexed format, and new pairix-indexed format.

7.30.3 API/CLI changes

- `create` only accepts a file path and optional group path instead of an open file object.
- `Cooler.matrix` selector now returns a balanced dense 2D NumPy array by default. Explicitly set `balance` to False to get raw counts and set `sparse` to True to get a `coo_matrix` as per old behavior.
- Command line parameters of `cload` changed significantly

7.30.4 Bug fixes

- Fixed bug in `csort` that led to incorrect triangularity of trans read pairs.

7.31 v0.5.3

Date: 2016-09-10

- Check for existence of required external tools in CLI
- Fixed `cooler show` incompatibility with older versions of matplotlib
- Fixed `cooler.annotate` to work on empty dataframe input
- Fixed broken pipe signals not getting suppressed on Python 2
- `cooler cload` raises a warning when bin file lists a contig missing from the contact list

7.32 v0.5.2

Date: 2016-08-26

- Fix bug in `cooler csort` parsing of chromsizes file.
- Workaround for two locale-related issues on Python 3. Only affects cases where a machine's locale is set to ASCII or Unices which use the ambiguous C or POSIX locales.
- Fix typo in setup.py and add pysam to dependencies.

7.33 v0.5.1

Date: 2016-08-24

- Bug fix in input parser to `cooler csort`
- Update triu reording awk template in `cooler csort`
- Rename `cooler binnify` to `cooler makebins`. Binnify sounds like "aggregate" which is what `cload` does.

7.34 v0.5.0

Date: 2016-08-24

- Most scripts ported over to a new command line interface using the Click framework with many updates.
- New `show` and `info` scripts.
- Updated Readme.
- Minor bug fixes.

7.35 v0.4.0

Date: 2016-08-18

7.35.1 Schema

- Updated file schema: v2
- `/bins/chroms` is now an enum instead of string column

7.35.2 API changes

- Table views are a bit more intuitive: selecting field names on table view objects returns a new view on the subset of columns.
- New API function: `cooler.annotate` for doing joins

7.35.3 New Features

- Support for nested Cooler "trees" at any depth in an HDF5 hierarchy
- Refactored `cooler.io` to provide "contact readers" that process different kinds of input (aggregate from a contact list, load from an existing matrix, etc.)
- Added new scripts for contact aggregation, loading, dumping and balancing

7.36 v0.3.0

Date: 2016-02-18

- 2D range selector `matrix()` now provides either rectangular data as coo_matrix or triangular data as a pixel table dataframe.
- Added binning support for any genome segmentation (i.e., fixed or variable bin width).
- Fixed issues with binning data from mapped read files.
- Genomic locus string parser now accepts ENSEMBL-style number-only chromosome names and FASTA-style sequence names containing pipes.

7.37 v0.2.1

Date: 2016-02-07

- Fixed bintable region fetcher

7.38 v0.2

Date: 2016-01-17

- First beta release

7.39 v0.1

Date: 2015-11-22

- Working initial prototype.
- genindex
- Glossary

Index

Symbols

A

B

C

U

Z