hictkpy Documentation

Roberto Rossini

May 13, 2024

CONTENTS

1 Installation 2

2 Quickstart 3

3 Creating .cool and .hic files 6

4 Python API Reference 9

Python Module Index 13

Index 14

hictkpy provides Python bindings to hictk, a blazing fast toolkit to work with .hic and .cool files.

Installation

Python bindings for hictk can be installed using pip or conda. See *here* for more details.

How to cite this project?

Please use the following BibTeX template to cite hictkpy in scientific discourse:

```
@article {hictk,
        author = {Roberto Rossini and Jonas Paulsen},
        title = {hictk: blazing fast toolkit to work with .hic and .cool files},
        elocation-id = {2023.11.26.568707},
        year = {2023},
        doi = {10.1101/2023.11.26.568707},
        publisher = {Cold Spring Harbor Laboratory},
        URL = {https://www.biorxiv.org/content/early/2023/11/27/2023.11.26.568707},
        eprint = {https://www.biorxiv.org/content/early/2023/11/27/2023.11.26.568707.
↪full.pdf},
        journal = {bioRxiv}
}
```

CHAPTER

ONE

INSTALLATION

hictkpy can be installed in various ways.

1.1 PIP

```
pip install hictkpy
```

1.2 Conda (bioconda)

```
conda install -c conda-forge -c bioconda hictkpy
```

1.3 From source

```
pip install 'git+https://github.com/paulsengroup/hictkpy.git@main'
```

Note that this will install hictk's build dependencies under `~/.conan2`, if you don't need Conan for other purposes feel free to delete this `~/.conan2` after installing hictkpy from git.

CHAPTER

TWO

QUICKSTART

hictkpy provides Python bindings for hictk through pybind11.

`hictk.File()` can open .cool and .hic files and allows retrieval of interactions as well as file metadata.

The example use file 4DNFIOTPSS3L.hic, which can be downloaded from here.

2.1 Opening files

```
In [1]: import hictkpy as htk

# .mcool and .cool files work as well
In [2]: f = htk.File("4DNFIOTPSS3L.hic", 10_000)

In [3]: f.path()
Out[3]: '4DNFIOTPSS3L.hic'
```

2.2 Reading file metadata

```
In [4]: f.bin_size()
Out[4]: 10000

In [5]: f.chromosomes()
Out[5]:
{'2L': 23513712,
 '2R': 25286936,
 '3L': 28110227,
 '3R': 32079331,
 '4': 1348131,
 'X': 23542271,
 'Y': 3667352}

In [6]: f.attributes()
Out[6]:
{'bin_size': 10000,
 'format': 'HIC',
 'format_version': 8,
 'assembly': '/var/lib/cwl/stgb25a903a-ebb6-4a56-bf3f-90bd84a40bf4/4DNFIBEEN92C.chrom.
 ↪sizes',
 'format-url': 'https://github.com/aidenlab/hic-format',
 'nbins': 13758,
 'nchroms': 8}
```

2.3 Fetch interactions

Interactions can be fetched by calling the *hictkpy.File.fetch()* method on *hictkpy.File()* objects.

hictkpy.File.fetch() returns *hictkpy.PixelSelector()* objects, which are very cheap to create.

```
# Fetch all interactions (genome-wide query) in COO format (row, column, count)
In [7]: sel = f.fetch()

# Fetch all interactions (genome-wide query) in bedgraph2 format
In [8]: sel = f.fetch(join=True)

# Fetch KR-normalized interactions
In [9]: sel = f.fetch(normalization="KR")

# Fetch interactions for a region of interest
In [9]: sel = f.fetch("2L:10,000,000-20,000,000")

In [10]: sel = f.fetch("2L:10,000,000-20,000,000", "X")

In [11]: sel.nnz()
Out[11]: 2247057

In [12]: sel.sum()
Out[12]: 7163361
```

2.3.1 Fetching interactions as pandas DataFrames

```
In [13]: sel = f.fetch("2L:10,000,000-20,000,000", join=True)

In [14]: sel.to_df()
Out[14]:
       chrom1    start1      end1 chrom2    start2      end2  count
0          2L  10000000  10010000     2L  10000000  10010000   6759
1          2L  10000000  10010000     2L  10010000  10020000   3241
2          2L  10000000  10010000     2L  10020000  10030000    760
3          2L  10000000  10010000     2L  10030000  10040000    454
4          2L  10000000  10010000     2L  10040000  10050000    289
...       ...       ...       ...    ...       ...       ...    ...
339036     2L  19970000  19980000     2L  19980000  19990000    407
339037     2L  19970000  19980000     2L  19990000  20000000    221
339038     2L  19980000  19990000     2L  19980000  19990000    391
339039     2L  19980000  19990000     2L  19990000  20000000    252
339040     2L  19990000  20000000     2L  19990000  20000000    266

[339041 rows x 7 columns]
```

2.3.2 Fetching interactions as scipy.sparse.coo_matrix

```
In [15]: sel = f.fetch("2L:10,000,000-20,000,000", join=True)

In [16]: sel.to_coo()
Out[16]:
<1000x1000 sparse matrix of type '<class 'numpy.int32'>'
        with 339041 stored elements in COOrdinate format>
```

2.3.3 Fetching interactions as numpy NDarray

```
In [17]: sel = f.fetch("2L:10,000,000-20,000,000", join=True)

In [18]: m = sel.to_numpy()

In [19]: import matplotlib.pyplot as plt

In [20]: from matplotlib.colors import LogNorm

In [21]: plt.imshow(m, norm=LogNorm())

In [22]: plt.show()
```

CHAPTER

THREE

CREATING .COOL AND .HIC FILES

hictkpy supports creating .cool and .hic files from pre-binned interactions in COO or BedGraph2 format.

The example use file 4DNFIOTPSS3L.hic, which can be downloaded from here.

3.1 Preparation

The first step consists of converting interactions from `4DNFIOTPSS3L.hic` to bedGraph2 format. This can be achieved using `hictk dump`

```
user@dev:/tmp$ hictk dump --join 4DNFIOTPSS3L.hic --resolution 50000 > pixels.bg2

user@dev:/tmp$ head pixels.bg2

  2L  0       50000   2L      0       50000   30211
  2L  0       50000   2L      50000   100000  13454
  2L  0       50000   2L      100000  150000  2560
  2L  0       50000   2L      150000  200000  911
  2L  0       50000   2L      200000  250000  753
  2L  0       50000   2L      250000  300000  846
  2L  0       50000   2L      300000  350000  530
  2L  0       50000   2L      350000  400000  378
  2L  0       50000   2L      400000  450000  630
  2L  0       50000   2L      450000  500000  756
```

Next, we also generate the list of chromosomes.

```
user@dev:/tmp$ hictk dump -t chroms 4DNFIOTPSS3L.hic > chrom.sizes

user@dev:/tmp$ head chrom.sizes.bg2

  2L  23513712
  2R  25286936
  3L  28110227
  3R  32079331
  4   1348131
  X   23542271
  Y   3667352
```

3.2 Ingesting interactions in a .cool file

```
In [1]: import hictkpy as htk

In [2]: import pandas as pd

# Create a dictionary mapping chromosome names to chromosome sizes
In [3]: chroms = pd.read_table("chrom.sizes", names=["name", "length"])
    ...            .set_index("name")["length"]
    ...            .to_dict()

In [4]: chroms
Out[4]:
{'2L': 23513712,
 '2R': 25286936,
 '3L': 28110227,
 '3R': 32079331,
 '4': 1348131,
 'X': 23542271,
 'Y': 3667352}

# Initialize an empty .cool file
In [5]: f = htk.cooler.FileWriter("out.cool", chroms, resolution=50_000)

In [6]: cols = ["chrom1", "start1", "end1",
    ...         "chrom2", "start2", "end2",
    ...         "count"]

# Loop over chunks of interactions and progressively add them to "out.cool"
In [7]: for df in pd.read_table("pixels.bg2", names=cols, chunksize=1_000_000):
  ...:      f.add_pixels(df)
  ...:

# Important! If you forget to call f.finalize() the resulting .cool file will be empty
In [8]: f.finalize()

# Check that the resulting file has some interactions
In [9]: htk.File("out.cool").attributes()["nnz"]
Out[9]: 3118456
```

3.3 Ingesting interactions in a .hic file

Follow the same step as in the previous section and replace `htk.cooler.File` with `htk.hic.File`.

3.4 Tips and tricks

When loading interactions into a .cool or .hic file, interactions are initially stored in a temporary file. When loading a large number of interactions, this temporary file can grow to be quite large. When this is the case, it is wise to pass a custom temporary folder where temporary files will be created:

```
In [1]: f = htk.cooler.FileWriter("out.cool", chroms, resolution=50_000, tmpdir="/var/
 ↪tmp/hictk")
```

When ingesting interactions in a .hic file, performance can be improved by using multiple threads:

```
In [1]: f = htk.hic.FileWriter("out.hic", chroms, resolution=50_000, n_threads=8)
```

When memory allows it, it is possible to bypass temporary files by specifying a very large chunk size and ingesting all interactions at once. This can significantly speed up file creation.

```
# Initialize an empty .cool file

In [1]: cols = ["chrom1", "start1", "end1",
    ...         "chrom2", "start2", "end2",
    ...         "count"]

In [2]: df = pd.read_table("pixels.bg2", names=cols)

In [3]: f = htk.cooler.FileWriter("out.cool", chroms, resolution=50_000, chunk_
 ↪size=len(df) + 1)

In [4]: f.add_pixels(df)

In [5]: f.finalize()
```

CHAPTER

FOUR

PYTHON API REFERENCE

hictkpy API is structured as follows:

4.1 Generic API

hictkpy.**is_cooler**(*path: str*) → bool

Test whether path points to a cooler file.

hictkpy.**is_mcool_file**(*path: str*) → bool

Test whether path points to a .mcool file.

hictkpy.**is_scool_file**(*path: str*) → bool

Test whether path points to a .scool file.

hictkpy.**is_hic**(*path: str*) → bool

Test whether path points to a .hic file.

class hictkpy.**MultiResFile**

Class representing a file handle to a .hic or .mcool file

__init__

Open a multi-resolution Cooler file (.mcool).

chromosomes

Get chromosomes sizes as a dictionary mapping names to sizes.

path

Get the file path.

resolutions

Get the list of available resolutions.

class hictkpy.**File**

Class representing a file handle to a .cool or .hic file.

__init__

Construct a file object to a .hic, .cool or .mcool file given the file path and resolution. Resolution is ignored when opening single-resolution Cooler files.

attributes

Get file attributes as a dictionary.

avail_normalizations

Get the list of available normalizations.

bins

Get bins as a pandas DataFrame.

chromosomes
Get chromosomes sizes as a dictionary mapping names to sizes.

fetch
Fetch interactions overlapping a region of interest.

has_normalization
Check whether a given normalization is available.

is_cooler
Test whether file is in .cool format.

is_hic
Test whether file is in .hic format.

nbins
Get the total number of bins.

nchroms
Get the total number of chromosomes.

path
Return the file path.

resolution
Get the bin size in bp.

uri
Return the file URI.

weights
Fetch the balancing weights for the given normalization method.

class hictkpy.**PixelSelector**
Class representing pixels overlapping with the given genomic intervals.

coord1
Get query coordinates for the first dimension.

coord2
Get query coordinates for the second dimension.

nnz
Get the number of non-zero entries for the current pixel selection.

sum
Get the total number of interactions for the current pixel selection.

to_coo
Retrieve interactions as a scipy.sparse.coo_matrix.

to_df
Retrieve interactions as a pandas DataFrame.

to_numpy
Retrieve interactions as a numpy 2D matrix.

4.2 Cooler API

class hictkpy.cooler.**SingleCellFile**

Class representing a file handle to a .scool file.

__init__

Open a single-cell Cooler file (.scool).

attributes

Get file attributes as a dictionary.

bins

Get bins as a pandas DataFrame.

cells

Get the list of available cells.

chromosomes

Get chromosomes sizes as a dictionary mapping names to sizes.

path

Get the file path.

resolution

Get the bin size in bp.

class hictkpy.cooler.**FileWriter**

Class representing a file handle to create .cool files.

__init__

Open a .cool file for writing.

add_pixels

Add pixels from a pandas DataFrame containing pixels in COO or BG2 format (i.e. either with columns=[bin1_id, bin2_id, count] or with columns=[chrom1, start1, end1, chrom2, start2, end2, count].

chromosomes

Get chromosomes sizes as a dictionary mapping names to sizes.

finalize

Write interactions to file.

path

Get the file path.

resolutions

Get the resolution in bp.

4.3 Hi-C API

class hictkpy.hic.**FileWriter**

Class representing a file handle to create .hic files.

__init__

Overloaded function.

1. `__init__(self, path: str, chromosomes: dict, resolution: int, assembly: str = 'unknown', n_threads: int = 1, chunk_size: int = 10000000, tmpdir: str = '/tmp', compression_lvl: int = 9, skip_all_vs_all_matrix: bool = False) -> None`

Open a .hic file for writing.

2. `__init__(self, path: str, chromosomes: dict, resolutions: list[int], assembly: str = 'unknown', n_threads: int = 1, chunk_size: int = 10000000, tmpdir: str = '/tmp', compression_lvl: int = 9, skip_all_vs_all_matrix: bool = False) -> None`

Open a .hic file for writing.

add_pixels

Add pixels from a pandas DataFrame containing pixels in COO or BG2 format (i.e. either with columns=[bin1_id, bin2_id, count] or with columns=[chrom1, start1, end1, chrom2, start2, end2, count].

chromosomes

Get chromosomes sizes as a dictionary mapping names to sizes.

finalize

Write interactions to file.

path

Get the file path.

resolutions

Get the list of resolutions in bp.

PYTHON MODULE INDEX

h

hictkpy, 9
hictkpy.cooler, 11
hictkpy.hic, 11

INDEX

Symbols

__init__ (*hictkpy.File attribute*), 9
__init__ (*hictkpy.MultiResFile attribute*), 9
__init__ (*hictkpy.cooler.FileWriter attribute*), 11
__init__ (*hictkpy.cooler.SingleCellFile attribute*), 11
__init__ (*hictkpy.hic.FileWriter attribute*), 11

A

add_pixels (*hictkpy.cooler.FileWriter attribute*), 11
add_pixels (*hictkpy.hic.FileWriter attribute*), 12
attributes (*hictkpy.cooler.SingleCellFile attribute*), 11
attributes (*hictkpy.File attribute*), 9
avail_normalizations (*hictkpy.File attribute*), 9

B

bins (*hictkpy.cooler.SingleCellFile attribute*), 11
bins (*hictkpy.File attribute*), 9

C

cells (*hictkpy.cooler.SingleCellFile attribute*), 11
chromosomes (*hictkpy.cooler.FileWriter attribute*), 11
chromosomes (*hictkpy.cooler.SingleCellFile attribute*), 11
chromosomes (*hictkpy.File attribute*), 9
chromosomes (*hictkpy.hic.FileWriter attribute*), 12
chromosomes (*hictkpy.MultiResFile attribute*), 9
coord1 (*hictkpy.PixelSelector attribute*), 10
coord2 (*hictkpy.PixelSelector attribute*), 10

F

fetch (*hictkpy.File attribute*), 10
File (*class in hictkpy*), 9
FileWriter (*class in hictkpy.cooler*), 11
FileWriter (*class in hictkpy.hic*), 11
finalize (*hictkpy.cooler.FileWriter attribute*), 11
finalize (*hictkpy.hic.FileWriter attribute*), 12

H

has_normalization (*hictkpy.File attribute*), 10
hictkpy
    module, 9
hictkpy.cooler
    module, 11
hictkpy.hic
    module, 11

I

is_cooler (*hictkpy.File attribute*), 10
is_cooler() (*in module hictkpy*), 9
is_hic (*hictkpy.File attribute*), 10
is_hic() (*in module hictkpy*), 9
is_mcool_file() (*in module hictkpy*), 9
is_scool_file() (*in module hictkpy*), 9

M

module
    hictkpy, 9
    hictkpy.cooler, 11
    hictkpy.hic, 11
MultiResFile (*class in hictkpy*), 9

N

nbins (*hictkpy.File attribute*), 10
nchroms (*hictkpy.File attribute*), 10
nnz (*hictkpy.PixelSelector attribute*), 10

P

path (*hictkpy.cooler.FileWriter attribute*), 11
path (*hictkpy.cooler.SingleCellFile attribute*), 11
path (*hictkpy.File attribute*), 10
path (*hictkpy.hic.FileWriter attribute*), 12
path (*hictkpy.MultiResFile attribute*), 9
PixelSelector (*class in hictkpy*), 10

R

resolution (*hictkpy.cooler.SingleCellFile attribute*), 11
resolution (*hictkpy.File attribute*), 10
resolutions (*hictkpy.cooler.FileWriter attribute*), 11
resolutions (*hictkpy.hic.FileWriter attribute*), 12
resolutions (*hictkpy.MultiResFile attribute*), 9

S

SingleCellFile (*class in hictkpy.cooler*), 11
sum (*hictkpy.PixelSelector attribute*), 10

T

to_coo (*hictkpy.PixelSelector attribute*), 10
to_df (*hictkpy.PixelSelector attribute*), 10

to_numpy (*hictkpy.PixelSelector attribute*), 10

U

uri (*hictkpy.File attribute*), 10

W

weights (*hictkpy.File attribute*), 10