

---

# **cortexpy Documentation**

***Release 0.46.5***

**Warren Kretzschmar and Kiran Garimella**

**Feb 09, 2020**



---

## Contents

---

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Overview of Cortexpy</b>   | <b>1</b>  |
| 1.1      | Audience . . . . .  | 1         |
| 1.2      | Free software . . . . .   | 1         |
| 1.3      | Installation . . . . .  | 2         |
| 1.4      | Documentation . . . . .   | 2         |
| 1.5      | Citing cortexpy . . . . .   | 2         |
| 1.6      | Bugs . . . . .  | 2         |
| 1.7      | Development . . . . .   | 2         |
| <b>2</b> | <b>Tutorial</b>   | <b>5</b>  |
| 2.1      | Using the python API to filter Cortex graphs . . . . .                | 5         |
| <b>3</b> | <b>On link-informed graph traversal</b>                               | <b>7</b>  |
| 3.1      | Link-informed graph traversal in <code>cortexpy</code> . . . . .      | 7         |
| <b>4</b> | <b>API reference</b>  | <b>9</b>  |
| 4.1      | Random access of Cortex graphs . . . . .                              | 9         |
| 4.2      | Cortex graph headers . . . . .  | 9         |
| 4.3      | Cortex kmers . . . . .  | 10        |
| 4.4      | Link-informed graph traversal . . . . .                               | 10        |
| 4.5      | Representing Cortex graphs as <code>nx.Graph</code> objects . . . . . | 11        |
| 4.6      | Interacting with graphs . . . . .                                     | 12        |
| 4.7      | Utility functions . . . . .   | 13        |
| <b>5</b> | <b>License</b>  | <b>15</b> |
| <b>6</b> | <b>Indices and tables</b>   | <b>19</b> |
|          | <b>Python Module Index</b>  | <b>21</b> |
|          | <b>Index</b>  | <b>23</b> |



# CHAPTER 1

---

## Overview of Cortexpy

---

|         |  |
|---------|--|
| tests   |  |
| package |  |
| docs    |  |

Cortexpy is a Python package for sequence analysis using linked and colored De Bruijn graphs such as the ones created by [Cortex](#) and [Mccortex](#). This project aims to mirror many of the features contained in [CortexJDK](#).

Cortexpy also comes with a command-line tool for basic inspection and manipulation of Cortex graphs with and without links.

## 1.1 Audience

The audience of cortexpy is researchers working with colored De Bruijn graphs and link information in [Cortex](#) and [Mccortex](#) format.

## 1.2 Free software

Cortexpy is free software; you can redistribute it and/or modify it under the terms of the Apache License version 2.0. Contributions are welcome. Please join us on [GitHub](#).

## 1.3 Installation

```
pip install cortexpy
```

## 1.4 Documentation

For more information, please see cortexpy [documentation](#).

## 1.5 Citing cortexpy

If you use cortexpy in your work, please consider citing:

Akhter, Shirin, Warren W. Kretzschmar, Veronika Nordal, Nicolas Delhomme, Nathaniel R. Street, Ove Nilsson, Olof Emanuelsson, and Jens F. Sundström. “Integrative analysis of three RNA sequencing methods identifies mutually exclusive exons of MADS-box isoforms during early bud development in *Picea abies*.” *Frontiers in Plant Science* 9 (2018). <https://doi.org/10.3389/fpls.2018.01625>

## 1.6 Bugs

This code is maintained by Warren Kretzschmar <[winni@warrenwk.com](mailto:winni@warrenwk.com)>. For bugs, please raise a [GitHub issue](#).

## 1.7 Development

1. Install [conda](#).
2. Download mccortex for testing:

```
conda env create -f environment.yml -n my-dev-environment
```

3. Activate development environment:

```
conda activate my-dev-environment
```

4. Install remaining development tools:

```
pip3 install -r requirements.txt
```

All remaining commands in the development section need to be run in an activated conda dev environment.

### 1.7.1 Tests

```
make test
```

### 1.7.2 Deploy new cortexpy version to pypi

Requires access credentials for pypi.

```
make deploy
```

### 1.7.3 Building the docs

The documentation is automatically built by read-the-docs on push to master. To build the documentation manually:

```
# install sphinx dependencies
pip install -r docs/requirements.txt

make docs
```





The cortexpy package consists of a python API and a command-line tool for working with Cortex graphs. Below, we start by looking at how to use the python API to perform an example workflow.

## 2.1 Using the python API to filter Cortex graphs

### 2.1.1 Building Cortex files

Let's start by creating two Cortex files to work with. At present, cortexpy does not provide a way to easily create a Cortex file, so we will instead use `Mccortex`. Mccortex can be compiled from source or installed using `bioconda`.

Let's start by creating two FASTA files from which to create two Cortex files:

```
echo -e '>1\nAAAAAA' > file1.fasta
echo -e '>1\nCCCCC' > file2.fasta
```

We now have two FASTA files each containing a single sequence. We can now build a Cortex graph from each file. We choose to use a kmer-size of 5:

```
mccortex 5 build --sort -k 5 --sample file1 -1 file1.fasta file1.ctx
mccortex 5 build --sort -k 5 --sample file2 -1 file2.fasta file2.ctx
```

We now have two cortex files: `file1.ctx` and `file2.ctx`. As the Cortex format represents colored De Bruijn graphs, we could have stored the information from the two FASTA files in a single graph as two separate colors. However, we are creating two files in order to demonstrate the cortexpy API later on.

We can check what kmers are stored in each graph using the `cortexpy` command-line tool:

```
> cortexpy view graph file1.ctx
AAAAA 1 .....

> cortexpy view graph file2.ctx
CCCCC 1 .....
```

This output tells us that each graph consists of a single kmer with coverage 1.

## 2.1.2 Inspecting Cortex graphs in Python

Cortexpy offers many ways to inspect Cortex files. Much of that functionality is available through the `RandomAccess` class. Let us start by loading a Cortex file inside python:

```
>>> from cortexpy.graph.parser.random_access import RandomAccess
>>> # make sure to open the cortex graph in binary mode
>>> ra = RandomAccess(open('file1.ctx', 'rb'))
```

We can now interrogate the `ra` object. Let's see what the header size of the Cortex file is:

```
>>> ra.header.kmer_size
5
```

Let's check if the kmer `AAAAA` exists in the graph and retrieve it:

```
>>> 'AAAAA' in ra
True
>>> ra['AAAAA']
Kmer(_kmer_data=KmerData(_data=b'\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00
↳', kmer_size=5, num_colors=1, _kmer='AAAAA', _coverage=None, _edges=None), num_
↳colors=1, kmer_size=5, _revcomp=None)
```

We can see that the returned kmer object contains information on the kmer size (5) and the number of colors stored in the kmer (1).

Now let's put it all together and search both graphs that we created while *Building Cortex files* for our kmer of interest, `AAAAA`:

Listing 1: search.py

```
from cortexpy.graph.parser.random_access import RandomAccess

for graph in ['file1.ctx', 'file2.ctx']:
    # make sure to open the cortex graph in binary mode
    with open(graph, 'rb') as fh:
        ra = RandomAccess(fh)

        # let's see if our favorite kmer is in the graph
        if 'AAAAA' in ra:
            print(f'AAAAA exists in {graph}!')
```

This is what we see if we run this code from the command line:

```
> python3 search.py
AAAAA exists in file1.ctx!
```

---

## On link-informed graph traversal

---

The all-simple-paths algorithm as implemented in `nx.all_simple_paths()` inside the `networkx` (abbreviated below as `nx`) package version 2.2 uses a depth-first traversal scheme to find all possible paths from a start node to one or more end nodes<sup>1</sup>.

For example, let nodes A-F represent unitigs in a De Bruijn graph created from sequencing reads of transcripts:

An all-simple-paths traversal starting at A will return the paths ABDE, ABDF, ACDE, and ACDF. However, what if the sequenced reads that were used to create this graph only originated from two paths: ABDE and ACDF? Can some of these sequencing reads be used to restrict the paths returned by the all-simple-paths algorithm?

`Mccortex` provides a data structure called “links” for annotating De Bruijn graphs in `Cortex` format. In the example above, links can be used to store information on a read that covers both the A → B and D → E junctions. `Cortexpy` can use these links to performed a “link-informed” (that is, a restricted) all-simple-paths traversal.

### 3.1 Link-informed graph traversal in `cortexpy`

#### 3.1.1 `Cortexpy` uses `networkx` algorithms

`Cortexpy` represents `Cortex` graphs as `nx.DiGraph` objects<sup>2</sup>. This allows the easy application of `networkx` algorithms to `Cortex` graphs. `cortexpy` achieves link-informed traversal by wrapping a `Cortex` graph in a `LinkedGraphTraverser` object, which modifies the behavior of the `__getitem__()` method. To understand why this works, let us first take a look at the all-simple-paths algorithm:

```

1 def _all_simple_paths_graph(G, source, targets, cutoff):
2     """From networkx.algorithms.simple_paths"""
3     visited = collections.OrderedDict.fromkeys([source])
4     stack = [iter(G[source])]
5     while stack:
6         children = stack[-1]
```

(continues on next page)

<sup>1</sup> `Cortexpy` currently uses a copy of `nx.all_simple_paths()` named `_all_simple_paths_graph()`.

<sup>2</sup> The implementation is not perfect and could use some improvement.

(continued from previous page)

```

7      child = next(children, None)
8      if child is None:
9          stack.pop()
10         visited.popitem()
11     elif len(visited) < cutoff:
12         if child in visited:
13             continue
14         if child in targets:
15             yield list(visited) + [child]
16         visited[child] = None
17         if targets - set(visited.keys()): # expand stack until find all targets
18             stack.append(iter(G[child]))
19         else:
20             visited.popitem() # maybe other ways to child
21     else: # len(visited) == cutoff:
22         for target in (targets & (set(children) | {child})) - set(visited.
→keys()):
23             yield list(visited) + [target]
24         stack.pop()
25         visited.popitem()

```

The key line here is the highlighted line 18. This is the line that appends an iterator of a node's successors to the stack of nodes to visit. The algorithm asks the graph object *G* for the successor nodes of *child* by calling the `__getitem__()` method of *G*:

```
G[child]
```

This means that we can restrict the paths returned by `_all_simple_paths_graph()` by restricting the successor nodes returned by *G*.

### 3.1.2 `LinkedGraphTraverser` restricts simple paths using links

The `__getitem__()` method of `LinkedGraphTraverser` restricts the returned successors using the following rules:

1. If no link information exists for the query node, then return all successors.
2. Otherwise, return only successors that are consistent with the links encountered on the path from start to query node.
3. If the query node is annotated with links, pick up all links.
4. For each successor node, only retain links that are consistent with the path taken from the start to this successor node.
5. For each successor node, drop links that are no longer relevant to the successor node (i.e. links that have expired)

## 4.1 Random access of Cortex graphs

This module contains classes for inspecting Cortex graphs with random access to their kmers.

**class** `cortexpy.graph.parser.random_access.RandomAccess` (*graph\_handle*,  
*kmer\_cache\_size=None*)

Provide fast k-mer access to Cortex graph in log(n) time (n = number of kmers in graph)

**\_\_getitem\_\_** (*lexlo\_string*)

Return kmer associated with kmer string

No check is performed to make sure that the input string is a lexicographically-lowest kmer string. Use `get_kmer_for_string()` in order to convert a kmer string to its lexlo form before retrieving it from the cortex object.

**\_\_iter\_\_** ()

Iterate over kmer strings in graph in order stored in graph

**get\_kmer\_for\_string** (*string*)

Will compute the revcomp of kmer string before getting a kmer

**items** ()

Iterate over kmer strings and kmers in graph in order stored in graph

**values** ()

Iterate over kmers in cortex graph

## 4.2 Cortex graph headers

This module contains classes for parsing and representing a Cortex file header

```
class cortexpy.graph.parser.header.Header (version=6, kmer_size=1,  
                                           kmer_container_size=None,  
                                           num_colors=None, mean_read_lengths=None,  
                                           total_sequences=None, sample_names=None,  
                                           error_rates=None,  
                                           color_info_blocks=NOTHING)
```

Cortex header object

This object allows access to header information contained in a cortex file

```
classmethod from_stream (stream)  
    Extract a cortex header from a file handle
```

## 4.3 Cortex kmers

This module provides classes and functions for working with Cortex kmers.

```
class cortexpy.graph.parser.kmer.Kmer (kmer_data, num_colors, kmer_size, revcomp=None)  
    Represents a Cortex kmer
```

This class wraps a kmer data object with attributes and methods for inspecting and manipulating the underlying kmer data object.

```
increment_color_coverage (color)  
    Increment the coverage of a color by one
```

```
class cortexpy.graph.parser.kmer.StringKmerConverter (kmer_size)  
    Converts kmer strings to various binary representations
```

```
to_uints (kmer_string)  
    Converts kmer_string to big-endian uint64 array
```

```
connect_kmers (first, second, color, identical_kmer_check=True)  
    Connect two kmers
```

```
disconnect_kmers (first, second, colors)  
    Disconnect two kmers
```

```
find_all_neighbors (first, second)  
    Return kmers and letters to get from first kmer to second
```

## 4.4 Link-informed graph traversal

This module provides classes for parsing Mccortex link files and for traversing graphs while using links.

```
class cortexpy.links.LinkOrientation  
    An enumeration.
```

```
class cortexpy.links.LinkWalker (links, junctions)  
    Manages the loading and walking of links for kmers
```

```
choose_branch (base)  
    Choose a branch and advance all links. Keep only links consistent with branch.
```

```
load_kmer (kmer)  
    Load the link group for a kmer in the orientation of the kmer.
```

```

next_junction_bases ()
    Returns the the bases of the branches that can be chosen.

class cortexpy.links.LinkedGraphTraverser (graph, walkers=NOTHING)
    Adapter for linked walkers to be able to work with nx.all_simple_paths()

class cortexpy.links.UnitigLinkWalker (link_walker, unitigs, kmer_size, current_unitig)
    Traverses a unitig graph with links

    choose (successor)
        Register the choice of a successor and advance

    link_successors ()
        Only returns unitigs based on link information

    successors ()
        Returns nodes from links or all available junctions if no link info exists

class cortexpy.links.LinkedGraphTraverser (graph, walkers=NOTHING)
    Adapter for linked walkers to be able to work with nx.all_simple_paths()

    __getitem__ (item)
        Get the children of item according to the walker object associated with item

        Warning: This scheme only works with depth-first search.

```

## 4.5 Representing Cortex graphs as `nx.Graph` objects

This module contains classes for representing Cortex graphs as objects that are compatible with `networkx` algorithms.

todo: Simplify the Graph implementations

```

class cortexpy.graph.cortex.ConsistentCortexDiGraph (kmer_mapping=NOTHING,
                                                    graph=NOTHING)

```

Graph that stores kmer strings that are consistent with each other

```

class cortexpy.graph.cortex.CortexDiGraph (kmer_mapping=NOTHING,
                                             graph=NOTHING)

```

Stores cortex k-mers and conforms to parts of the interface of `nx.MultiDiGraph`

```
add_edge (first, second, *, key)
```

Note: edges can only be added to existing nodes

```
nbunch_iter (nbunch=None)
```

Return an iterator over nodes contained in *nbunch* that are also in the graph.

This code has been copied from `networkx`.

The nodes in *nbunch* are checked for membership in the graph and if not are silently ignored.

**Parameters** *nbunch* (*single node, container, or all nodes (default=all nodes)*) – The view will only report edges incident to these nodes.

**Returns** *niter* – An iterator over nodes in *nbunch* that are also in the graph. If *nbunch* is `None`, iterate over all nodes in the graph.

**Return type** iterator

**Raises** `NetworkXError` – If *nbunch* is not a node or sequence of nodes. If a node in *nbunch* is not hashable.

**See also:**`Graph.__iter__()`**Notes**

When nbunch is an iterator, the returned iterator yields values directly from nbunch, becoming exhausted when nbunch is exhausted.

To test whether nbunch is a single node, one can use “if nbunch in self:”, even after processing with this routine.

If nbunch is not a node or a (possibly empty) sequence/iterator or None, a `NetworkXError` is raised. Also, if any object in nbunch is not hashable, a `NetworkXError` is raised.

This method was copied from Networkx version 2.1 and then modified

```
class cortexpy.graph.cortex.CortexGraphMapping(ra_parser,    exclusion_set=NOTHING,
                                              new_kmers=NOTHING,
                                              n_duplicates=0)
```

Create a dict-like kmer mapping from a RandomAccess parser (ra\_parser)

The exclusion set tracks kmers deleted from the ra\_parser. The new\_kmers track kmers that have been added to the mapping. Kmers that exist in both new\_kmers and ra\_parser are considered overwritten. The kmers in new\_kmers have precedence.

```
connect_kmers(first, second, color, identical_kmer_check=True)
```

Connect two kmers

```
disconnect_kmers(first, second, colors)
```

Disconnect two kmers

```
cortexpy.graph.cortex.build_cortex_graph(* , sample_names, kmer_size, num_colors, colors,
                                          kmer_generator=None, kmer_mapping=None)
```

Colored de Bruijn graph constructor

```
cortexpy.graph.cortex.get_canonical_edge(first, second)
```

Get canonical edge.

Canonical edges are between lexlo kmers and are ordered lexicographically

Return canonical edge, if the first and second nodes were lexlo

## 4.6 Interacting with graphs

This module contains classes and functions for inspecting, manipulating, and traversing graphs

```
class cortexpy.graph.interactor.SeedKmerStringIterator(seed_kmer_strings,    un-
                                                         seen_lexlo_kmer_strings,
                                                         seen_lexlo_kmer_strings=NOTHING)
```

Iterates seeds and their lexlo representations that exist in the supplied all\_kmers:

```
>>> list(SeedKmerStringIterator.from_all_kmer_strings_and_seeds(['AAC'], ['GTT']))
[('GTT', 'AAC')]
```

Kmers that are not in the seed list are return after that:



```
>>> list(SeedKmerStringIterator.from_all_kmer_strings_and_seeds(['AAA', 'AAC'], [
↳ 'GTT']))
[('GTT', 'AAC'), ('AAA', 'AAA')]
```

Seeds that do not exist in the all\_kmers are not returned.

```
>>> list(SeedKmerStringIterator.from_all_kmer_strings_and_seeds([], ['CCC']))
[]
```

Returned kmers from all\_kmers list are returned in order.

```
>>> list(SeedKmerStringIterator.from_all_kmer_strings_and_seeds(['AAA', 'AAG',
↳ 'AAC'], []))
[('AAA', 'AAA'), ('AAG', 'AAG'), ('AAC', 'AAC')]
```

`cortexpy.graph.interactor.edge_nodes_of(graph)`

Find all edge nodes of a graph

Second return value is direction of edge.

`cortexpy.graph.interactor.make_copy_of_color_for_kmer_graph(graph, color, include_self_refs=False)`

Makes a copy of graph, but only copies over links with key=color. Only copies over nodes that are linked by a link with key=color.

## 4.7 Utility functions

This module contains utility functions that are used inside cortexpy. These functions may also be useful outside of cortexpy.

`cortexpy.utils.kmerize_contig(contig, kmer_size)`

Return generator of kmers in contig

The returned kmers are not lexicographically lowest.

```
>>> list(kmerize_contig('ATTT', 3))
['ATT', 'TTT']
```

`cortexpy.utils.kmerize_fasta(fasta, kmer_size)`

Return generator to all kmers in fasta

`cortexpy.utils.lexlo`

Return lexicographically lowest version of a kmer string and its reverse complement

The reverse complement of a kmer string is generated and the lexicographically-lowest kmer string is returned.

```
>>> lexlo('AAA')
'AAA'
```

```
>>> lexlo('TTT')
'AAA'
```



## CHAPTER 5

---

### License

---

Cortexpy is distributed under the Apache License version 2.0:

Apache License  
Version 2.0, January 2004  
<http://www.apache.org/licenses/>

#### TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

##### 1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation,

(continues on next page)

(continued from previous page)

and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the

(continues on next page)

(continued from previous page)

Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

- (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
- (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
- (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
- (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

- 5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
- 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
- 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or

(continues on next page)

(continued from previous page)

|  |  |
|--|--|
| <p>implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.</p> <p>8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.</p> <p>9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.</p> <p>END OF TERMS AND CONDITIONS</p> <p>APPENDIX: How to apply the Apache License to your work.</p> <p>To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[ ]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.</p> <p>Copyright [yyyy] [name of copyright owner]</p> <p>Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at</p> <p><a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a></p> <p>Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.</p> |  |
|--|--|

## CHAPTER 6

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





### C

`cortexpy.graph.cortex`, [11](#)  
`cortexpy.graph.interactor`, [12](#)  
`cortexpy.graph.parser.header`, [9](#)  
`cortexpy.graph.parser.kmer`, [10](#)  
`cortexpy.graph.parser.random_access`, [9](#)  
`cortexpy.links`, [10](#)  
`cortexpy.utils`, [13](#)



## Symbols

`__getitem__()` (cortexpy.graph.parser.random\_access.RandomAccess method), 9

`__getitem__()` (cortexpy.links.LinkedGraphTraverser method), 11

`__iter__()` (cortexpy.graph.parser.random\_access.RandomAccess method), 9

## A

`add_edge()` (cortexpy.graph.cortex.CortexDiGraph method), 11

## B

`build_cortex_graph()` (in module cortexpy.graph.cortex), 12

## C

`choose()` (cortexpy.links.UnitigLinkWalker method), 11

`choose_branch()` (cortexpy.links.LinkWalker method), 10

`connect_kmers()` (cortexpy.graph.cortex.CortexGraphMapping method), 12

`connect_kmers()` (in module cortexpy.graph.parser.kmer), 10

`ConsistentCortexDiGraph` (class in cortexpy.graph.cortex), 11

`CortexDiGraph` (class in cortexpy.graph.cortex), 11

`CortexGraphMapping` (class in cortexpy.graph.cortex), 12

`cortexpy.graph.cortex` (module), 11

`cortexpy.graph.interactor` (module), 12

`cortexpy.graph.parser.header` (module), 9

`cortexpy.graph.parser.kmer` (module), 10

`cortexpy.graph.parser.random_access` (module), 9

`cortexpy.links` (module), 10

`cortexpy.utils` (module), 13

## D

`disconnect_kmers()` (cortexpy.graph.cortex.CortexGraphMapping method), 12

`disconnect_kmers()` (in module cortexpy.graph.parser.kmer), 10

## E

`edge_nodes_of()` (in module cortexpy.graph.interactor), 13

## F

`find_all_neighbors()` (in module cortexpy.graph.parser.kmer), 10

`from_stream()` (cortexpy.graph.parser.header.Header class method), 10

## G

`get_canonical_edge()` (in module cortexpy.graph.cortex), 12

`get_kmer_for_string()` (cortexpy.graph.parser.random\_access.RandomAccess method), 9

## H

`Header` (class in cortexpy.graph.parser.header), 9

## I

`increment_color_coverage()` (cortexpy.graph.parser.kmer.Kmer method), 10

`items()` (cortexpy.graph.parser.random\_access.RandomAccess method), 9

## K

`Kmer` (class in cortexpy.graph.parser.kmer), 10

`kmerize_contig()` (in module `cortexpy.utils`), 13  
`kmerize_fasta()` (in module `cortexpy.utils`), 13

## L

`lexlo` (in module `cortexpy.utils`), 13  
`link_successors()` (`cortexpy.links.UnitigLinkWalker` method), 11  
`LinkedGraphTraverser` (class in `cortexpy.links`), 11  
`LinkOrientation` (class in `cortexpy.links`), 10  
`LinkWalker` (class in `cortexpy.links`), 10  
`load_kmer()` (`cortexpy.links.LinkWalker` method), 10

## M

`make_copy_of_color_for_kmer_graph()` (in module `cortexpy.graph.interactor`), 13

## N

`nbunch_iter()` (`cortexpy.graph.cortex.CortexDiGraph` method), 11  
`next_junction_bases()` (`cortexpy.links.LinkWalker` method), 10

## R

`RandomAccess` (class in `cortexpy.graph.parser.random_access`), 9

## S

`SeedKmerStringIterator` (class in `cortexpy.graph.interactor`), 12  
`StringKmerConverter` (class in `cortexpy.graph.parser.kmer`), 10  
`successors()` (`cortexpy.links.UnitigLinkWalker` method), 11

## T

`to_uints()` (`cortexpy.graph.parser.kmer.StringKmerConverter` method), 10

## U

`UnitigLinkWalker` (class in `cortexpy.links`), 11

## V

`values()` (`cortexpy.graph.parser.random_access.RandomAccess` method), 9