# bg Documentation

***Release 1.10***

**Sergey Aganezov**

**Sep 28, 2018**

# Contents

**BG** is a python based package that provides a comprehensive implementation of comparative genomics combinatorial object named breakpoint graph [1].

Code is written with the philosophy of TDD and requires Python v3.3+ for correct work.

The package is created and maintained by Sergey Aganezov, Ph.D. Candidate at the Department of Mathematics & Computational Biology institute (CBI), George Washington University (GWU), Washington, DC, USA.

Author is very grateful for thoughtful and dedicated leadership of Dr. Max A. Alekseyev, Associate Professor at Department of Mathematics & CBI, GWU.

CHAPTER 1

# Installation

Package is distributed by pypi online repository of software for the Python programming language.

To install execute the following simple command

```
>>> pip install bg
```

CHAPTER 2

Contents:

## 2.1 Contributing

This page will show the basic principals, that are used during the development of this package.

The project is hosted on the github.

### 2.1.1 Test Driven Development

Whole project is written with a test-driven development paradigm. This is especially important, since this project provides an implementation of a complex combinatorial object, which must be reliable in use during research projects.

Project uses unittest framework for implementing TDD paradigm.

### 2.1.2 Issues reporting

Any found bugs, miss-citations, mistakes in documentation, questions, etc. shall be reported to the issue-tracking system, powered by github.

### 2.1.3 Code incorporation

There are several rules for new code to be incorporated into this library:

1. All code has to written using the Sphinx style
2. All code must be covered by tests
3. All algorithms and data structures code must have proper citations

## 2.2 API documentation

### 2.2.1 grimm.py

**class** bg.grimm.**GRIMMReader**

Bases: object

Class providing a staticmethod based implementation of reading GRIMM formatted data file-like object and obtain a *bg.breakpoint_graph.BreakpointGraph* instance.

There are no private methods implementations for all public methods so inheritance shall be performed with caution. For now GRIMM format is a bit simplified and straightened from the version provided at http://grimm.ucsd.edu/GRIMM/grimm_instr.html

Supported GRIMM format:

1. all strings are stripped from both sides for tabs, spaces, etc. Below when said "string", stripped string is assumed
2. `genome declaration` is specified on a string that starts with >
   (a) `genome name` is everything, that follows > sign
3. all input data before the next genome declaration (or EOF) will be attributed to this genome by its `genome name`
4. a data string (containing information about gene orders) is a string that is not a genome declaration, comment, empty string
   (a) every new genomic fragments (chromosome/scaffold/contig/etc) must be specified on a new string
   (b) every data string must contain a `$` (for linear case) or `@` (for circular case) gene order terminator, that indicates the end of current genomic fragment
   (c) everything after the gene order terminator is ignored
   (d) if no gene order before gene order terminator is specified an error would be raised
   (e) **gene order:**
      i. gene order is a sequence of space separated block name strings with optional orientation declaration
      ii. **block can be described by a regular expression `^((-|\+).+$)|([^-\+]+$)` and viewed as follows:** if the sign (+ or −) is present as a first character, then it must be followed by a nonempty block name string if sign is not present, everything is assumed to be a block name, and + orientation is assigned to it automatically
5. comment string starts with # sign and is ignored during data processing

Main operations:

- *GRIMMReader.is_genome_declaration_string()*: checks if supplied string after stripping corresponds to `genome declaration`
- *GRIMMReader.is_comment_string()*: checks if supplied string after stripping corresponds to comment and shall thus be ignored in data processing
- *GRIMMReader.parse_genome_declaration_string()*: parses a string marked as `genome declaration` and returns a corresponding genome name
- *GRIMMReader.parse_data_string()*: parses a string assumed to contain gene order data, retrieving information about fragment type, gene order, blocks names and their orientation

- *GRIMMReader.get_edges_from_parsed_data()*: taking into account fragment type (circular|linear) and retrieved gene order information translates adjacencies between blocks into edges for addition to the *bg.breakpoint_graph.BreakpointGraph*
- *GRIMMReader.get_breakpoint_graph()*: taking a file-like object transforms supplied gene order data into the language of BreakpointGraph

**static _GRIMMReader__assign_vertex_pair**(*block*)
Assigns usual BreakpointGraph type vertices to supplied block.

Vertices are labeled as "block_name" + "h" and "block_name" + "t" according to blocks orientation.

**Parameters** **block** (`(str, str)`) – information about a genomic block to create a pair of vertices for in a format of ( + | −, block_name)

**Returns** a pair of vertices labeled according to supplied blocks name (respecting blocks orientation)

**Return type** `(str, str)`

**static get_breakpoint_graph**(*stream*, *merge_edges=True*)
Taking a file-like object transforms supplied gene order data into the language of

**Parameters**

- **merge_edges** (`bool`) – a flag that indicates if parallel edges in produced breakpoint graph shall be merged or not
- **stream** (`iterable` ver `str`) – any iterable object where each iteration produces a `str` object

**Returns** an instance of a BreakpointGraph that contains information about adjacencies in genome specified in GRIMM formatted input

**Return type** *bg.breakpoint_graph.BreakpointGraph*

**static get_edges_from_parsed_data**(*parsed_data*)
Taking into account fragment type (circular|linear) and retrieved gene order information translates adjacencies between blocks into edges for addition to the *bg.breakpoint_graph.BreakpointGraph*

In case supplied fragment is linear (`$`) special artificial vertices (with `___infinity` suffix) are introduced to denote fragment extremities

**Parameters** **parsed_data** (`tuple(str, list((str, str), ...))`) – (`$` | `@`, [(+ | −, block_name),...]) formatted data about fragment type and ordered list of oriented blocks

**Returns** a list of vertices pairs that would correspond to edges in *bg.breakpoint_graph.BreakpointGraph*

**Return type** `list((str, str), ...)`

**static is_comment_string**(*data_string*)
Checks if supplied string after stripping corresponds to comment and shall thus be ignored in data processing

**Parameters** **data_string** (`str`) – a string to check if it is a pure comment string

**Returns** a flag indicating if supplied string is a pure comment string

**Return type** `Boolean`

**static is_genome_declaration_string**(*data_string*)
Checks if supplied string after stripping corresponds to `genome declaration`

**Parameters** **data_string** (`str`) – a string to check genome name declaration in

**Returns** a flag indicating if supplied string corresponds to genome name declaration

**Return type** Boolean

**static parse_data_string**(*data_string*)

Parses a string assumed to contain gene order data, retrieving information about fragment type, gene order, blocks names and their orientation

First checks if gene order termination signs are present. Selects the earliest one. Checks that information preceding is not empty and contains gene order. Generates results structure by retrieving information about fragment type, blocks names and orientations.

**NOTE:** comment signs do not work in data strings. Rather use the fact that after first gene order termination sign everything is ignored for processing

**Parameters** **data_string** (str) – a string to retrieve gene order information from

**Returns** ($ | @, [(+ | −, block_name),...]) formatted structure corresponding to gene order in supplied data string and containing fragments type

**Return type** tuple(str, list((str, str), ...))

**static parse_genome_declaration_string**(*data_string*)

Parses a string marked as genome declaration and returns a corresponding *bg.genome.BGGenome*

**Parameters** **data_string** (str) – a string to retrieve genome name from

**Returns** genome name from supplied genome declaration string

**Return type** *bg.genome.BGGenome*

### 2.2.2 breakpoint_graph.py

**class** bg.breakpoint_graph.**BreakpointGraph**(*graph=None*)

Bases: object

Class providing implementation of breakpoint graph data structure and most utilized operations on it.

*BreakpointGraph* anticipates to work with bg.vertex.BGVertex, *bg.edge.BGEdge* and *bg.multicolor.Multicolor* classes instances, but is not limited to them. Extreme caution has to be assumed when working with non-expected classes.

The engine of graph information storage, low-level algorithms implementation is powered by NetworkX package MultiGraph data structure. This class provides a smart wrapping around it to perform most useful, from combinatorial bioinformatics stand point, operations and manipulations.

Class carries following attributes carrying information about graphs structure:

- BreakpointGraph.bg: instance of NetworkX MultiGraph class

Main operations:

- *BreakpointGraph.add_bgedge()*: adds an instance of *bg.edge.BGEdge* to the current *BreakpointGraph*
- *BreakpointGraph.add_edge()*: adds a new *bg.edge.BGEdge*, constructed from a pair of supplied vertices instances and *bg.multicolor.Multicolor* object, to the current *BreakpointGraph*
- *BreakpointGraph.get_vertex_by_name()*: returns a bg.vertex.BGVertex instance by provided name argument

- *BreakpointGraph.get_edge_by_two_vertices()*: returns a first edge (order is determined by `key` NetworkX MultiGraph edge attribute) between two supplied `bg.vertex.BGVertex`
- *BreakpointGraph.get_edges_by_vertex()*: returns a generator yielding *bg.edge.BGEdge*
- *BreakpointGraph.edges_between_two_vertices()*: returns a generator yielding *bg.edge.BGEdge* between two supplied vertices
- *BreakpointGraph.connected_components_subgraphs()*: returns a generator of *BreakpointGraph* object, that represent connected components of a current *BreakpointGraph* object, deep copying(by default) all information of current *BreakpointGraph*
- *BreakpointGraph.delete_edge()*: deletes and edge from perspective of multi-color substitution of supplied vertices
- *BreakpointGraph.delete_bgedge()*: deletes a supplied *bg.edge.BGEdge* instance from perspective of substituting multi-colors.
- *BreakpointGraph.split_edge()*: deletes a supplied *bg.multicolor.Multicolor* instance in identifies edge from two supplied vertices.
- *BreakpointGraph.split_bgedge()*: splits a *bg.edge.BGEdge* with respect to provided guidance
- *BreakpointGraph.split_all_edges_between_two_vertices()*: splits all edges between two supplied vertives with respect to provided guidance.
- *BreakpointGraph.split_all_edges()*: splits all edge in *BreakpointGraph* with respect to provided guidance.
- *BreakpointGraph.delete_all_edges_between_two_vertices()*: deletes all edges between two given vertices, by plain deleting them from MultiGraph underling structure.
- *BreakpointGraph.merge_all_edges_between_two_vertices()*: merges all edge between two given vertices creating a single edge containing information about multi-colors in respective edges.
- *BreakpointGraph.merge_all_edges()*: merges all edges in current *BreakpointGraph*.
- *BreakpointGraph.merge()*: merges two *BreakpointGraph* instances with respect to vertices, edges, and multicolors.
- *BreakpointGraph.update()*: updates information in current *BreakpointGraph* instance by adding new *bg.edge.BGEdge* instances form supplied *BreakpointGraph*.

**_BreakpointGraph__add_bgedge**(*bgedge*, *merge=True*)

Adds supplied *bg.edge.BGEdge* object to current instance of *BreakpointGraph*.

Checks that vertices in supplied *bg.edge.BGEdge* instance actually are present in current *BreakpointGraph* if **merge** option of provided. Otherwise a new edge is added to the current *BreakpointGraph*.

**Parameters**

- **bgedge** (*bg.edge.BGEdge*) – instance of *bg.edge.BGEdge* infromation form which is to be added to current *BreakpointGraph*
- **merge** (`Boolean`) – a flag to merge supplied information from multi-color perspective into a first existing edge between two supplied vertices

**Returns** `None`, performs inplace changes

**_BreakpointGraph__delete_all_bgedges_between_two_vertices**(*vertex1*, *vertex2*)

Deletes all edges between two supplied vertices

**Parameters**

- **vertex1** (any python hashable object. `bg.vertex.BGVertex` is expected) – a first out of two vertices edges between which are to be deleted
- **vertex2** (any python hashable object. `bg.vertex.BGVertex` is expected) – a second out of two vertices edges between which are to be deleted

**Returns** `None`, performs inplace changes

**_BreakpointGraph__delete_bgedge**(*bgedge*, *key=None*, *keep_vertices=False*)

Deletes a supplied *bg.edge.BGEdge* from a perspective of multi-color substitution. If unique identifier `key` is not provided, most similar (from perspective of *bg.multicolor.Multicolor.similarity_score()* result) edge between respective vertices is chosen for change.

If no unique identifier for edge to be changed is specified, edge to be updated is determined by iterating over all edges between vertices in supplied *bg.edge.BGEdge* instance and the edge with most similarity score to supplied one is chosen. Once the edge to be substituted from is determined, substitution if performed form a perspective of *bg.multicolor.Multicolor* substitution. If after substitution the remaining multicolor of respective edge is empty, such edge is deleted form a perspective of MultiGraph edge deletion.

**Parameters**

- **bgedge** (*bg.edge.BGEdge*) – an edge to be deleted from a perspective of multi-color substitution
- **key** – unique identifier of existing edges in current *BreakpointGraph* instance to be changed

**Type** any python object. `int` is expected.

**Returns** `None`, performed inplace changes.

**_BreakpointGraph__edges**(*nbunch=None*, *keys=False*)

Iterates over edges in current *BreakpointGraph* instance.

Returns a generator over the edges in current *BreakpointGraph* instance producing instances of *bg.edge.BGEdge* instances wrapping around information in underlying MultiGraph object.

**Parameters**

- **nbunch** – a vertex to iterate over edges outgoing from, if not provided,iteration over all edges is performed.
- **keys** (`Boolean`) – a flag to indicate if information about unique edge's ids has to be returned alongside with edge

**Returns** generator over edges in current *BreakpointGraph*

**Return type** `generator`

**_BreakpointGraph__edges_between_two_vertices**(*vertex1*, *vertex2*, *keys=False*)

Iterates over edges between two supplied vertices in current *BreakpointGraph*

Checks that both supplied vertices are present in current breakpoint graph and then yield all edges that are located between two supplied vertices. If keys option is specified, then not just edges are yielded, but rather pairs (edge, edge_id) are yielded

**Parameters**

- **vertex1** (any hashable object, `bg.vertex.BGVertex` is expected) – a first vertex out of two, edges of interest are incident to

- **vertex2** (any hashable object, `bg.vertex.BGVertex` is expected) – a second vertex out of two, edges of interest are incident to
- **keys** (`Boolean`) – a flag to indicate if information about unique edge's ids has to be returned alongside with edge

**Returns** generator over edges (tuples `edge, edge_id` if keys specified) between two supplied vertices in current *BreakpointGraph* wrapped in `bg.vertex.BGVertex`

**Return type** `generator`

**_BreakpointGraph__get_edge_by_two_vertices**(*vertex1*, *vertex2*, *key=None*)

Returns an instance of `bg.edge.BBGEdge` edge between to supplied vertices (if `key` is supplied, returns a `bg.edge.BBGEdge` instance about specified edge).

Checks that both specified vertices are in current *BreakpointGraph* and then depending on `key` argument, creates a new `bg.edge.BBGEdge` instance and incorporates respective multi-color information into it.

**Parameters**

- **vertex1** (*any hashable object*) – first vertex instance out of two in current *BreakpointGraph*
- **vertex2** (*any hashable object*) – second vertex instance out of two in current *BreakpointGraph*
- **key** (any python object. `None` or `int` is expected) – unique identifier of edge of interested to be retrieved from current *BreakpointGraph*

**Returns** edge between two specified edges respecting a `key` argument.

**Return type** *bg.edge.BGEdge*

**_BreakpointGraph__get_edges_by_vertex**(*vertex*, *keys=False*)

Iterates over edges that are incident to supplied vertex argument in current *BreakpointGraph*

Checks that the supplied vertex argument exists in underlying MultiGraph object as a vertex, then iterates over all edges that are incident to it. Wraps each yielded object into *bg.edge.BGEdge* object.

**Parameters**

- **vertex** (any hashable object. `bg.vertex.BGVertex` object is expected.) – a vertex object in current *BreakpointGraph* object
- **keys** (`Boolean`) – a flag to indicate if information about unique edge's ids has to be returned alongside with edge

**Returns** generator over edges (tuples `edge, edge_id` if keys specified) in current *BreakpointGraph* wrapped in `bg.vertex.BGEVertex`

**Return type** `generator`

**_BreakpointGraph__get_vertex_by_name**(*vertex_name*)

Obtains a vertex object by supplied label

Returns a `bg.vertex.BGVertex` or its subclass instance

**Parameters** **vertex_name** (any hashable python object. `str` expected.) – a vertex label it is identified by.

**Returns** vertex with supplied label if present in current *BreakpointGraph*, `None` otherwise

**_BreakpointGraph__merge_all_bgedges_between_two_vertices**(*vertex1*, *vertex2*)

Merges all edge between two supplied vertices into a single edge from a perspective of multi-color merging.

**Parameters**

- **vertex1** (any python hashable object. `bg.vertex.BGVertex` is expected) – a first out of two vertices edges between which are to be merged together
- **vertex2** (any python hashable object. `bg.vertex.BGVertex` is expected) – a second out of two vertices edges between which are to be merged together

**Returns** `None`, performs inplace changes

**_BreakpointGraph__split_all_edges_between_two_vertices**(*vertex1*, *vertex2*, *guidance=None*, *sorted_guidance=False*, *account_for_colors_multiplicity_in_guidance=T*

Splits all edges between two supplied vertices in current *BreakpointGraph* instance with respect to the provided guidance.

Iterates over all edges between two supplied vertices and splits each one of them with respect to the guidance.

**Parameters**

- **vertex1** (any python hashable object. `bg.vertex.BGVertex` is expected) – a first out of two vertices edges between which are to be split
- **vertex2** (any python hashable object. `bg.vertex.BGVertex` is expected) – a second out of two vertices edges between which are to be split
- **guidance** (*iterable where each entry is iterable with colors entries*) – a guidance for underlying *bg.multicolor.Multicolor* objects to be split

**Returns** `None`, performs inplace changes

**_BreakpointGraph__split_bgedge**(*bgedge*, *guidance=None*, *sorted_guidance=False*, *account_for_colors_multiplicity_in_guidance=True*, *key=None*)

Splits a *bg.edge.BGEdge* in current *BreakpointGraph* most similar to supplied one (if no unique identifier `key` is provided) with respect to supplied guidance.

If no unique identifier for edge to be changed is specified, edge to be split is determined by iterating over all edges between vertices in supplied *bg.edge.BGEdge* instance and the edge with most similarity score to supplied one is chosen. Once the edge to be split is determined, split if performed form a perspective of *bg.multicolor.Multicolor* split. The originally detected edge is deleted, and new edges containing information about multi-colors after splitting, are added to the current *BreakpointGraph*.

**Parameters**

- **bgedge** (*bg.edge.BGEdge*) – an edge to find most "similar to" among existing edges for a split
- **guidance** (*iterable where each entry is iterable with colors entries*) – a guidance for underlying *bg.multicolor.Multicolor* object to be split
- **duplication_splitting** (`Boolean`) – flag (**not** currently implemented) for a splitting of color-based splitting to take into account multiplicity of respective colors

- **key** (any python object. `int` is expected) – unique identifier of edge to be split

**Returns** `None`, performs inplace changes

**_BreakpointGraph__update**(*breakpoint_graph*, *merge_edges=False*)
Updates a current :class‘BreakpointGraph‘ object with information from a supplied :class‘BreakpointGraph‘ instance.

Depending of a `merge_edges` flag, while updating of a current :class‘BreakpointGraph‘ object is occuring, edges between similar vertices can be merged to already existing ones.

**Parameters**

- **breakpoint_graph** (*:class`BreakpointGraph`*) – a breakpoint graph to extract information from, which will be then added to the current
- **merge_edges** (`Boolean`) – flag to indicate if edges to be added to current :class‘BreakpointGraph‘ object are to be merged to already existing ones

**Returns** `None`, performs inplace changes

**__init__**(*graph=None*)
Initialization of a *BreakpointGraph* object.

**Parameters** **graph** (*instance of NetworkX MultiGraph is expected.*) – is supplied, *BreakpointGraph* is initialized with supplied or brand new (empty) instance of NetworkX MultiGraph.

**add_bgedge**(*bgedge*, *merge=True*)
Adds supplied *bg.edge.BGEdge* object to current instance of *BreakpointGraph*.

Proxies a call to *BreakpointGraph._BreakpointGraph__add_bgedge()* method.

**Parameters**

- **bgedge** (*bg.edge.BGEdge*) – instance of *bg.edge.BGEdge* infromation form which is to be added to current *BreakpointGraph*
- **merge** (`Boolean`) – a flag to merge supplied information from multi-color perspective into a first existing edge between two supplied vertices

**Returns** `None`, performs inplace changes

**add_edge**(*vertex1*, *vertex2*, *multicolor*, *merge=True*, *data=None*)
Creates a new *bg.edge.BGEdge* object from supplied information and adds it to current instance of *BreakpointGraph*.

Proxies a call to *BreakpointGraph._BreakpointGraph__add_bgedge()* method.

**Parameters**

- **vertex1** (*any hashable object*) – first vertex instance out of two in current *BreakpointGraph*
- **vertex2** (*any hashable object*) – second vertex instance out of two in current *BreakpointGraph*
- **multicolor** (*bg.multicolor.Multicolor*) – an information about multi-colors of added edge
- **merge** (`Boolean`) – a flag to merge supplied information from multi-color perspective into a first existing edge between two supplied vertices

**Returns** `None`, performs inplace changes

**apply_kbreak**(*kbreak*, *merge=True*)

Check validity of supplied k-break and then applies it to current *BreakpointGraph*

Only *bg.kbreak.KBreak* (or its heirs) instances are allowed as `kbreak` argument. KBreak must correspond to the valid kbreak and, since some changes to its internals might have been done since its creation, a validity check in terms of starting/resulting edges is performed. All vertices in supplied KBreak (except for paired infinity vertices) must be present in current *BreakpointGraph*. For all supplied pairs of vertices (except for paired infinity vertices), there must be edges between such pairs of vertices, at least one of which must contain a multicolor matching a multicolor of supplied kbreak.

Edges of specified in kbreak multicolor are deleted between supplied pairs of vertices in kbreak.start_edges (except for paired infinity vertices). New edges of specified in kbreak multicolor are added between all pairs of vertices in kbreak.result_edges (except for paired infinity vertices). If after the kbreak application there is an infinity vertex, that now has no edges incident to it, it is deleted form the current *BreakpointGraph*.

**Parameters**

- **kbreak** (*bg.kbreak.KBreak*) – a k-break to be applied to current *BreakpointGraph*
- **merge** (`Boolean`) – a flag to indicate on how edges, that will be created by a k-break, will be added to current *BreakpointGraph*

**Returns** nothing, performs inplace changes

**Return type** `None`

**Raises** `ValueError`, `TypeError`

**connected_components_subgraphs**(*copy=True*)

Iterates over connected components in current *BreakpointGraph* object, and yields new instances of *BreakpointGraph* with respective information deep-copied by default (week reference is possible of specified in method call).

**Parameters** **copy** (`Boolean`) – a flag to signal if graph information has to be deep copied while producing new *BreakpointGraph* instances, of just reference to respective data has to be made.

**Returns** generator over connected components in current *BreakpointGraph* wrapping respective connected components into new *BreakpointGraph* objects.

**Return type** `generator`

**delete_all_edges_between_two_vertices**(*vertex1*, *vertex2*)

Deletes all edges between two supplied vertices

Proxies a call to *BreakpointGraph._BreakpointGraph__delete_all_bgedges_between_two_vertice* method.

**Parameters**

- **vertex1** (any python hashable object. `bg.vertex.BGVertex` is expected) – a first out of two vertices edges between which are to be deleted
- **vertex2** (any python hashable object. `bg.vertex.BGVertex` is expected) – a second out of two vertices edges between which are to be deleted

**Returns** `None`, performs inplace changes

**delete_bgedge**(*bgedge*, *key=None*)

Deletes a supplied *bg.edge.BGEdge* from a perspective of multi-color substitution. If unique identifier `key` is not provided, most similar (from perspective of *bg.multicolor.Multicolor.similarity_score()* result) edge between respective vertices is chosen for change.

Proxies a call to $BreakpointGraph._BreakpointGraph_{__delete_bgedge}$ method.

**Parameters**

- **bgedge** (*bg.edge.BGEdge*) – an edge to be deleted from a perspective of multi-color substitution
- **key** – unique identifier of existing edges in current *BreakpointGraph* instance to be changed

**Type** any python object. `int` is expected.

**Returns** `None`, performed inplace changes.

**delete_edge**(*vertex1*, *vertex2*, *multicolor*, *key=None*)

Creates a new *bg.edge.BGEdge* instance from supplied information and deletes it from a perspective of multi-color substitution. If unique identifier `key` is not provided, most similar (from perspective of *bg.multicolor.Multicolor.similarity_score()* result) edge between respective vertices is chosen for change.

Proxies a call to $BreakpointGraph._BreakpointGraph_{__delete_bgedge}$ method.

**Parameters**

- **vertex1** (any python hashable object. `bg.vertex.BGVertex` is expected) – a first vertex out of two the edge to be deleted is incident to
- **vertex2** (any python hashable object. `bg.vertex.BGVertex` is expected) – a second vertex out of two the edge to be deleted is incident to
- **multicolor** (*bg.multicolor.Multicolor*) – a multi-color to find most suitable edge to be deleted
- **key** – unique identifier of existing edges in current *BreakpointGraph* instance to be changed

**Type** any python object. `int` is expected.

**Returns** `None`, performed inplace changes.

**edges**(*nbunch=None*, *keys=False*)

Iterates over edges in current *BreakpointGraph* instance.

Proxies a call to *BreakpointGraph._BreakpointGraph__edges()*.

**Parameters**

- **nbunch** – a vertex to iterate over edges outgoing from, if not provided,iteration over all edges is performed.
- **keys** (`Boolean`) – a flag to indicate if information about unique edge's ids has to be returned alongside with edge

**Returns** generator over edges in current *BreakpointGraph*

**Return type** `generator`

**edges_between_two_vertices**(*vertex1*, *vertex2*, *keys=False*)

Iterates over edges between two supplied vertices in current *BreakpointGraph*

Proxies a call to `Breakpoint._Breakpoint__edges_between_two_vertices()` method.

**Parameters**

- **vertex1** (any hashable object, `bg.vertex.BGVertex` is expected) – a first vertex out of two, edges of interest are incident to

- **vertex2** (any hashable object, `bg.vertex.BGVertex` is expected) – a second vertex out of two, edges of interest are incident to
- **keys** (`Boolean`) – a flag to indicate if information about unique edge's ids has to be returned alongside with edge

**Returns** generator over edges (tuples `edge, edge_id` if keys specified) between two supplied vertices in current *BreakpointGraph* wrapped in `bg.vertex.BGVertex`

**Return type** `generator`

**classmethod from_json**(*data*, *genomes_data=None*, *genomes_deserialization_required=True*, *merge=False*)

A JSON deserialization operation, that recovers a breakpoint graph from its JSON representation

as information about genomes, that are encoded in breakpoint graph might be available somewhere else, but not the json object, there is an option to provide it and omit encoding information about genomes.

**get_edge_by_two_vertices**(*vertex1*, *vertex2*, *key=None*)

Returns an instance of `bg.edge.BBGEdge` edge between to supplied vertices (if `key` is supplied, returns a `bg.edge.BBGEdge` instance about specified edge).

Proxies a call to *BreakpointGraph._BreakpointGraph__get_edge_by_two_vertices()*.

**Parameters**

- **vertex1** (*any hashable object*) – first vertex instance out of two in current *BreakpointGraph*
- **vertex2** (*any hashable object*) – second vertex instance out of two in current *BreakpointGraph*
- **key** (any python object. `None` or `int` is expected) – unique identifier of edge of interested to be retrieved from current *BreakpointGraph*

**Returns** edge between two specified edges respecting a `key` argument.

**Return type** *bg.edge.BGEdge*

**get_edges_by_vertex**(*vertex*, *keys=False*)

Iterates over edges that are incident to supplied vertex argument in current *BreakpointGraph*

Proxies a call to `Breakpoint._Breakpoint__get_edges_by_vertex()` method.

**Parameters**

- **vertex** (any hashable object. `bg.vertex.BGVertex` object is expected.) – a vertex object in current *BreakpointGraph* object
- **keys** (`Boolean`) – a flag to indicate if information about unique edge's ids has to be returned alongside with edge

**Returns** generator over edges (tuples `edge, edge_id` if keys specified) in current *BreakpointGraph* wrapped in `bg.vertex.BGEVertex`

**Return type** `generator`

**get_vertex_by_name**(*vertex_name*)

Obtains a vertex object by supplied label

Proxies a call to *BreakpointGraph._BreakpointGraph__get_vertex_by_name()*.

**Parameters** **vertex_name** (any hashable python object. `str` expected.) – a vertex label it is identified by.

**Returns** vertex with supplied label if present in current *BreakpointGraph*, `None` otherwise

**Return type** *bg.vertices.BGVertex* or None

**classmethod merge**(*breakpoint_graph1*, *breakpoint_graph2*, *merge_edges=False*)

Merges two given instances of :class‘BreakpointGraph‘ into a new one, that gather all available information from both supplied objects.

Depending of a merge_edges flag, while merging of two dat structures is occurring, edges between similar vertices can be merged during the creation of a result :class‘BreakpointGraph‘ obejct.

Accounts for subclassing.

**Parameters**

- **breakpoint_graph1** (*:class`BreakpointGraph`*) – a first out of two :class‘BreakpointGraph‘ instances to gather information from
- **breakpoint_graph2** (*:class`BreakpointGraph`*) – a second out of two :class‘BreakpointGraph‘ instances to gather information from
- **merge_edges** (Boolean) – flag to indicate if edges in a new merged :class‘BreakpointGraph‘ object has to be merged between same vertices, or if splitting from supplied graphs shall be preserved.

**Returns** a new breakpoint graph object that contains all information gathered from both supplied breakpoint graphs

**Return type** :class‘BreakpointGraph‘

**merge_all_edges**()

Merges all edges in a current :class‘BreakpointGraph‘ instance between same pairs of vertices into a single edge from a perspective of multi-color merging.

Iterates over all possible pairs of vertices in current *BreakpointGraph* and merges all edges between respective pairs.

**Returns** None, performs inplace changes

**merge_all_edges_between_two_vertices**(*vertex1*, *vertex2*)

Merges all edge between two supplied vertices into a single edge from a perspective of multi-color merging.

Proxies a call to *BreakpointGraph._BreakpointGraph__merge_all_bgedges_between_two_verti*

**Parameters**

- **vertex1** (any python hashable object. bg.vertex.BGVertex is expected) – a first out of two vertices edges between which are to be merged together
- **vertex2** (any python hashable object. bg.vertex.BGVertex is expected) – a second out of two vertices edges between which are to be merged together

**Returns** None, performs inplace changes

**nodes**()

Iterates over nodes in current *BreakpointGraph* instance.

**Returns** generator over nodes (vertices) in current *BreakpointGraph* instance.

**Return type** generator

**split_all_edges**(*guidance=None*, *sorted_guidance=False*, *account_for_colors_multiplicity_in_guidance=True*)

Splits all edge in current *BreakpointGraph* instance with respect to the provided guidance.

Iterate over all possible distinct pairs of vertices in current *BreakpointGraph* instance and splits all edges between such pairs with respect to provided guidance.

**Parameters** **guidance** (*iterable where each entry is iterable with colors entries*) – a guidance for underlying *bg.multicolor.Multicolor* objects to be split

**Returns** None, performs inplace changes

**split_all_edges_between_two_vertices**(*vertex1*, *vertex2*, *guidance=None*, *sorted_guidance=False*, *account_for_colors_multiplicity_in_guidance=True*)

Splits all edges between two supplied vertices in current *BreakpointGraph* instance with respect to the provided guidance.

Proxies a call to *BreakpointGraph._BreakpointGraph__split_all_edges_between_two_vertices()* method.

**Parameters**

- **vertex1** (any python hashable object. bg.vertex.BGVertex is expected) – a first out of two vertices edges between which are to be split
- **vertex2** (any python hashable object. bg.vertex.BGVertex is expected) – a second out of two vertices edges between which are to be split
- **guidance** (*iterable where each entry is iterable with colors entries*) – a guidance for underlying *bg.multicolor.Multicolor* objects to be split

**Returns** None, performs inplace changes

**split_bgedge**(*bgedge*, *guidance=None*, *sorted_guidance=False*, *account_for_colors_multiplicity_in_guidance=True*, *key=None*)

Splits a *bg.edge.BGEdge* in current *BreakpointGraph* most similar to supplied one (if no unique identifier key is provided) with respect to supplied guidance.

Proxies a call to *BreakpointGraph._BreakpointGraph__split_bgedge()* method.

**Parameters**

- **bgedge** (*bg.edge.BGEdge*) – an edge to find most "similar to" among existing edges for a split
- **guidance** (*iterable where each entry is iterable with colors entries*) – a guidance for underlying *bg.multicolor.Multicolor* object to be split
- **duplication_splitting** (Boolean) – flag (**not** currently implemented) for a splitting of color-based splitting to take into account multiplicity of respective colors
- **key** (any python object. int is expected) – unique identifier of edge to be split

**Returns** None, performs inplace changes

**split_edge**(*vertex1*, *vertex2*, *multicolor*, *guidance=None*, *sorted_guidance=False*, *account_for_colors_multiplicity_in_guidance=True*, *key=None*)

Splits an edge in current *BreakpointGraph* most similar to supplied data (if no unique identifier key is provided) with respect to supplied guidance.

Proxies a call to *BreakpointGraph._BreakpointGraph__split_bgedge()* method.

**Parameters**

- **vertex1** (any python hashable object. `bg.vertex.BGVertex` is expected) – a first vertex out of two the edge to be split is incident to
- **vertex2** (any python hashable object. `bg.vertex.BGVertex` is expected) – a second vertex out of two the edge to be split is incident to
- **multicolor** (*bg.multicolor.Multicolor*) – a multi-color to find most suitable edge to be split
- **duplication_splitting** (`Boolean`) – flag (**not** currently implemented) for a splitting of color-based splitting to take into account multiplicity of respective colors
- **key** (any python object. `int` is expected) – unique identifier of edge to be split

**Returns** `None`, performs inplace changes

**to_json**(*schema_info=True*)
JSON serialization method that account for all information-wise important part of breakpoint graph

**update**(*breakpoint_graph*, *merge_edges=False*)
Updates a current :class‘BreakpointGraph‘ object with information from a supplied :class‘BreakpointGraph‘ instance.

Proxoes a call to *BreakpointGraph._BreakpointGraph__update()* method.

**Parameters**

- **breakpoint_graph** (*BreakpointGraph*) – a breakpoint graph to extract information from, which will be then added to the current
- **merge_edges** (`Boolean`) – flag to indicate if edges to be added to current :class‘BreakpointGraph‘ object are to be merged to already existing ones

**Returns** `None`, performs inplace changes

### 2.2.3 tree.py

**class** `bg.tree.`**BGTree**(*newick=None*, *newick_format=1*, *dist=1*, *leaf_wrapper=<class 'bg.genome.BGGenome'>*)
Bases: `object`

Class that is designed to store information about phylogenetic information and relations between multiple genomes

Class utilizes a ete3.Tree object as an internal storage This tree can store information about:

- edge lengths
- tree topology

**_BGTree__get_node_by_name**(*name*)
Returns a first TreeNode object, which name matches the specified argument

**Raises** ValueError (if no node with specified name is present in the tree)

**_BGTree__get_v_tree_consistent_leaf_based_hashable_multicolors**()
Internally used method, that recalculates VTree-consistent sets of leaves in the current tree

**_BGTree__has_edge**(*node1_name*, *node2_name*, *account_for_direction=True*)
Returns a boolean flag, telling if a tree has an edge with two nodes, specified by their names as arguments

If a account_for_direction is specified as True, the order of specified node names has to relate to parent - child relation, otherwise both possibilities are checked

**_BGTree__has_node**(*name*)
Check is the current Tree has a node, matching by name to the specified argument

**_BGTree__update_consistent_multicolors**()
Internally used method, that recalculates T-consistent / VT-consistent multicolors for current tree topology

**_BGTree__vertex_is_leaf**(*node_name*)
Checks if a node specified by its name as an argument is a leaf in the current Tree

**Raises** ValueError (if no node with specified name is present in the tree)

**__init__**(*newick=None*, *newick_format=1*, *dist=1*, *leaf_wrapper=<class 'bg.genome.BGGenome'>*)
Initialize self. See help(type(self)) for accurate signature.

**add_edge**(*node1_name*, *node2_name*, *edge_length=1*)
Adds a new edge to the current tree with specified characteristics

Forbids addition of an edge, if a parent node is not present Forbids addition of an edge, if a child node already exists

**Parameters**

- **node1_name** – name of the parent node, to which an edge shall be added
- **node2_name** – name of newly added child node
- **edge_length** – a length of specified edge

**Returns** nothing, inplace changes

**Raises** ValueError (if parent node IS NOT present in the tree, or child node IS already present in the tree)

**append**(*node_name*, *tree*, *copy=False*)
Append a specified tree (represented by a root TreeNode element) to the node, specified by its name

**Parameters** **copy** (*Boolean*) – a flag denoting if the appended tree has to be added as is, or is the deepcopy of it has to be used

**Raises** ValueError (if no node with a specified name, to which the specified tree has to be appended, is present in the current tree)

**bgedge_is_tree_consistent**(*bgedge*)
Checks is supplied BGEdge (from the perspective of its multicolor is T-consistent)

**bgedge_is_vtree_consistent**(*bgedge*)
Checks is supplied BGEdge (from the perspective of its multicolor is VT-consistent)

**edges**()

**Returns** iterator over edges in current tree.

**Return type** iterator

**get_distance**(*node1_name*, *node2_name*)
Returns a length of an edge / path, if exists, from the current tree

**Parameters**

- **node1_name** – a first node name in current tree
- **node2_name** – a second node name in current tree

**Returns** a length of specified by a pair of vertices edge / path

**Return type** *Number*

**Raises** ValueError, if requested a length of an edge, that is not present in current tree

**get_node_by_name**(*name*)
Proxies the call to the __get_node_by_name method

**get_tree_consistent_multicolors**()
Returns a copy of the list of T-consistent multicolors from current tree

**get_vtree_consistent_multicolors**()
Returns a copy of the list of VT-consistent multicolors from current tree

**has_edge**(*node1_name*, *node2_name*, *account_for_direction=True*)
Proxies a call to the __has_edge method

**has_node**(*name*)
Proxies a call to __has_node method

**multicolor_is_tree_consistent**(*multicolor*)
Checks is supplied multicolor is T-consistent

**multicolor_is_vtree_consistent**(*multicolor*)
Checks is supplied multicolor is VT-consistent

**nodes**()
Proxies iteration to the underlying Tree.iter_descendants iterator, but first yielding a root element

**Returns** iterator over all descendants of a root, starting with a root, in current tree

**Return type** iterator

**root**
A property based call for the root pointer in current tree

**tree_consistent_multicolors**
Property based getter, that checks for consistency in terms of precomputed T-consistent multicolors, recomputes all consistent multicolors if tree topology has changed and returns internally stored list of T-consistent multicolors

**tree_consistent_multicolors_set**
Property based getter, that checks for consistency in terms of precomputed T-consistent multicolors, recomputes all consistent multicolors if tree topology has changed and returns internally stored set of hashable representation of T-consistent multicolors

**vtree_consistent_multicolors**
Property based getter, that checks for consistency in terms of precomputed VT-consistent multicolors, recomputes all consistent multicolors if tree topology has changed and returns internally stored list of VT-consistent multicolors

**vtree_consistent_multicolors_set**
Property based getter, that checks for consistency in terms of precomputed VT-consistent multicolors, recomputes all consistent multicolors if tree topology has changed and returns internally stored set of hashable representation of VT-consistent multicolors

### 2.2.4 kbreak.py

**class** bg.kbreak.**KBreak**(*start_edges*, *result_edges*, *multicolor*, *data=None*)
Bases: object

A generic object that can represent any k-break ( k>= 2)

A notion of k-break arises from the bioinformatics combinatorial object BreakpointGraph and is first mentioned in http://home.gwu.edu/~maxal/ap_tcs08.pdf A generic k-break operates on k specified edges of spisific multicolor and replaces them with another set of k edges with the same multicolor on the same set of vertices in way, that the degree of vertices is kept intact.

Initialization of the instance of *KBreak* is performed with a validity check of supplied data, which must comply with the definition of k-break.

Class carries following attributes carrying information about k-break structure:

- `KBreak.start_edges`: a list of edges (in terms of paired vertices) that are to be removed by current *KBreak*
- `KBreak.result_edges`: a list of edges (in terms of paired vertices) that are to be created by current *KBreak*
- `KBreak.multicolor`: a *bg.multicolor.Multicolor* instance, that specifies the multicolor of edges that are to be removed / created by current *KBreak*

Main operations:

- *KBreak.valid_kbreak_matchings()*: a method that checks if provided sets of started / resulted edges comply with the notions ob k-break definition

**__init__**(*start_edges*, *result_edges*, *multicolor*, *data=None*)
Initialization of *KBreak* object.

The initialization process consists of multiple checks, before any assignment and initialization itself is performed.

First checks the fact, that information about start / result edges is supplied in form of paired vertices. Then check is performed to make sure, that degrees of vertices, that current *KBreak* operates on, is preserved.

**Parameters**

- **start_edges** (`list(tuple(vertex, vertex), ...)`) – a list of pairs of vertices, that specifies where edges shall be removed by current *KBreak*
- **result_edges** (`list(tuple(vertex, vertex), ...)`) – a list of pairs of vertices, that specifies where edges shall be created by current *KBreak*
- **multicolor** (*bg.multicolor.Multicolor*) – a multicolor, that specifies which edges between specified pairs of vertices are to be removed / created

**Returns** a new instance of `Kbreak`

**Return type** *KBreak*

**Raises** `ValueError`

**static valid_kbreak_matchings**(*start_edges*, *result_edges*)
A staticmethod check implementation that makes sure that degrees of vertices, that are affected by current *KBreak*

By the notion of k-break, it shall keep the degree of vertices in *bg.breakpoint_graph.BreakpointGraph* the same, after its application. By utilizing the Counter class, such check is performed, as the number the vertex is mentioned corresponds to its degree.

**Parameters**

- **start_edges** (`list(tuple(vertex, vertex), ...)`) – a list of pairs of vertices, that specifies where edges shall be removed by *KBreak*
- **result_edges** (`list(tuple(vertex, vertex), ...)`) – a list of pairs of vertices, that specifies where edges shall be created by *KBreak*

**Returns** a flag indicating if the degree of vertices are equal in start / result edges, targeted by *KBreak*

**Return type** Boolean

### 2.2.5 multicolor.py

**class** bg.multicolor.**Multicolor**(**args*)

Bases: object

Class providing implementation of multi-color notion for edges in *bg.breakpoint_graph.BreakpointGraph*.

Multi-color is a specific property of edges in Breakpoint Graph combinatorial object which represents similar adjacencies between genomic material in multiple genomes.

This class supports the following attributes, that carry information colors and their multiplicity of edges in *bg.breakpoint_graph.BreakpointGraph*.

- Multicolor.multicolors: a python Counter object which contains information about colors and their multiplicity for a given *Multicolor* instance
- *Multicolor.colors*: a property attribute providing a set of colors in Multicolor.multicolors attribute, hiding information about colors multiplicity

Main operations:

- +, −, +=, −=, ==, >, >=, <, <=
- *Multicolor.update()*: updates information in Multicolor.multicolors attribute of respective instance
- *Multicolor.merge()*: creates a new *Multicolor* object out of a list of provided *Multicolor* objects, gathering respective information about colors and their multiplicity
- *Multicolor.left_merge()*: updates respective *Multicolor* instance with information from supplied *Multicolor* object
- *Multicolor.delete()*: reduces information in respective instance Multicolor.multicolors attribute by iterating over supplied data
- *Multicolor.similarity_score()* computes how similar two supplied *Multicolor* object are
- *Multicolor.split_colors()* produces several new instances of *Multicolor* object by splitting information about colors by using provided guidance iterable set-like object

**_Multicolor__delete**(*multicolor*)

Reduces information *Multicolor* attribute by iterating over supplied colors data.

In case supplied argument is a *Multicolor* instance, multi-color specific information to de deleted is set to its Multicolor.multicolors. In other cases multi-color specific information to de deleted is obtained from iterating over the argument.

Colors and their multiplicity is reduces with a help of − method of python Counter object.

**Parameters** **multicolor** (any iterable with colors object as entries or *Multicolor*) – information about colors to be deleted from *Multicolor* object

**Returns** None, performs inplace changes

**static _Multicolor__left_merge**(*multicolor1*, *multicolor2*)

Updates first supplied *Multicolor* instance with information from second supplied *Multicolor* instance.

First supplied instances attribute `Multicolor.multicolors` is updated with a help of + method of python Counter object.

**Parameters**

- **multicolor1** (*Multicolor*) – instance to update information in
- **multicolor2** (*Multicolor*) – instance to use information for update from

**Returns** updated first supplied *Multicolor* instance

**Return type** *Multicolor*

**classmethod _Multicolor__merge**(**multicolors*)

Produces a new *Multicolor* object resulting from gathering information from all supplied *Multicolor* instances.

New *Multicolor* is created and its `Multicolor.multicolors` attribute is updated with similar attributes of supplied *Multicolor* objects.

Accounts for subclassing.

**Parameters** **multicolors** (*Multicolor*) – variable number of *Multicolor* objects

**Returns** object containing gathered information from all supplied arguments

**Return type** *Multicolor*

**__add__**(*other*)

Implementation of + operation for *Multicolor*

Invokes a private *Multicolor._Multicolor__merge()* method to implement addition of two *Multicolor* instances.

**Parameters** **other** (*Multicolor*) – object, whose multi-color information has to be added to current one

**Returns** new *Multicolor* object, colors in which and their multiplicity result from addition of current `Multicolor.multicolors` and supplied `Multicolor.multicolors`

**Return type** *Multicolor*

**Raises** `TypeError`, if not *Multicolor* instance is provided

**__eq__**(*other*)

Implementation of == operation for *Multicolor*

Two *Multicolor* objects are called to be equal if colors that both of them contain and respective colors multiplicity are equal. *Multicolor* instance never equals to non-*Multicolor* object. Performs `Multicolor.multicolors` comparison with a help of == method of python Counter object.

**Parameters** **other** (*any python object*) – an object to compare to

**Returns** a flag of equality between current *Multicolor* instance and supplied object

**Return type** `Boolean`

**__ge__**(*other*)

Implementation of ">=" operation for *Multicolor*

One *Multicolor* instance is said to be "greater than" the other *Multicolor* instance, if it contains greater or equal number of colors, as the other *Multicolor* object does, and multiplicity of all of them is greater or equal than in the other multicolor. *Multicolor* instance is never less, than non-*Multicolor* object.

**Parameters** **other** (*any python object*) – an object to compare to

**Returns** a flag if current *Multicolor* object is greater or equal than supplied object

**Return type** Boolean

**__gt__**(*other*)
Implementation of > operation for *Multicolor*

One *Multicolor* instance is said to be "greater than" the other *Multicolor* instance, if it contains greater os equal number of colors, as the other *Multicolor* object does, and multiplicity of all of them is greater or equal than in the other multicolor, and at least one color has multiplicity greater, than in the other multicolor. *Multicolor* instance is never less, than non-*Multicolor* object.

**Parameters** **other** (*any python object*) – an object to compare to

**Returns** a flag if current *Multicolor* object is less than supplied object

**Return type** Boolean

**__iadd__**(*other*)
Implementation of += operation for *Multicolor*

Invokes a private *Multicolor._Multicolor__merge()* method to implement addition of two *Multicolor* instances.

**Parameters** **other** (*Multicolor*) – object, whose multi-color information has to be added to current one

**Returns** new *Multicolor* object, colors in which and their multiplicity result from addition of current Multicolor.multicolors and supplied Multicolor.multicolors

**Return type** *Multicolor*

**Raises** TypeError, if not *Multicolor* instance is provided

**__init__**(**args*)
Initialization of *Multicolor* object.

Initialization is performed by supplied variable number of colors, that respective *Multicolor* object must contain information about Multiplicity of each color is determined by the number of times it occurs as argument in initialization process.

**Parameters** **args** (*any hashable python object*) – variable number of colors to contain information about

**Returns** a new instance of *Multicolor*

**Return type** *Multicolor*

**__isub__**(*other*)
Implementation of − operation for *Multicolor*

Updates current *Multicolor* instance by updating its Multicolor.multicolors attribute information by deleting multi-colors in supplied Multicolor.multicolors attribute. Utilizes − method of python Counter

**Parameters** **other** (*Multicolor*) – object, whose multi-color information to subtract form current one

**Returns** updated current *Multicolor* object

**Return type** *Multicolor*

**Raises** TypeError, if not *Multicolor* instance is supplied

**__le__**(*other*)
Implementation of "<=" operation for *Multicolor*

One *Multicolor* instance is said to be "less or equal than" the other *Multicolor* instance, if it contains less or equal number colors, as the other *Multicolor* object does, and multiplicity of all of them is less or equal than in the other multicolor. *Multicolor* instance is never less or equal, than non-*Multicolor* object.

**Parameters** **other** (*any python object*) – an object to compare to

**Returns** a flag if current *Multicolor* object is less or equal than supplied object

**Return type** Boolean

**__lt__**(*other*)
Implementation of < operation for *Multicolor*

One *Multicolor* instance is said to be "less than" the other *Multicolor* instance, if it contains less or equal number of colors colors, as the other *Multicolor* object does, and multiplicity of all of them is less or equal than in the other multicolor, and at least one color has multiplicity less, than in the other multicolor. *Multicolor* instance is never less, than non-*Multicolor* object.

**Parameters** **other** (*any python object*) – an object to compare to

**Returns** a flag if current *Multicolor* object is less than supplied object

**Return type** Boolean

**__mul__**(*other*)
Multicolor can be multiplied by a number and it multiplies multiplicity of each present color respectively

**Parameters** **other** – an integer multiplier

**Returns** a new multicolor object resulted from multiplying each colors multiplicity by the multiplier

**__sub__**(*other*)
Implementation of − operation for *Multicolor*

Creates a new *Multicolor* instance by cloning current *Multicolor* object and updating its `Multicolor.multicolors` attribute information by deleting multi-colors in supplied *Multicolor* object.

**Parameters** **other** (*Multicolor*) – object, whose multi-color information to subtract form current one

**Returns** new *Multicolor* object, colors in which and their multiplicity result from subtracting of current `Multicolor.multicolors` and supplied `Multicolor.multicolors` attributes.

**Return type** *Multicolor*

**Raises** `TypeError`, if not *Multicolor* instance is supplied

**colors**
Implements an "attribute" like object to access information about colors only, hiding information about their multiplicity.

Creates a fresh set object every time is accessed.

**Returns** all colors that current *Multicolor* object contains information about.

**Return type** set

**delete**(*multicolor*)
Reduces information *Multicolor* attribute by iterating over supplied colors data.

Works as proxy to respective call to private static method *Multicolor._Multicolor__delete()* for purposes of inheritance compatibility.

**Parameters** **multicolor** (any iterable with colors object as entries or *Multicolor*) – information about colors to be deleted from *Multicolor* object

**Returns** None, performs inplace changes

**hashable_representation**
For a sake of speed check for multicolor presence, each multicolor has a deterministic hashable representation

**intersect**(*other*)
Computes the multiset intersection, between the current Multicolor and the supplied Multicolor

**Parameters** **other** – another Multicolor object to compute a multiset intersection with

**Returns**

**Raises** **TypeError** – an intersection can be computed only between two Multicolor objects

**classmethod left_merge**(*multicolor1*, *multicolor2*)
Updates first supplied *Multicolor* instance with information from second supplied *Multicolor* instance.

Works as proxy to respective call to private static method *Multicolor._Multicolor__left_merge()* for purposes of inheritance compatibility.

Accounts for subclassing.

**Parameters**

- **multicolor1** (*Multicolor*) – instance to update information in
- **multicolor2** (*Multicolor*) – instance to use information for update from

**Returns** updated first supplied *Multicolor* instance

**Return type** *Multicolor*

**classmethod merge**(**multicolors*)
Produces a new *Multicolor* object resulting from gathering information from all supplied *Multicolor* instances.

Works as proxy to respective call to private static method *Multicolor._Multicolor__merge()* for purposes of inheritance compatibility.

**Parameters** **multicolors** (*Multicolor*) – variable number of *Multicolor* objects

**Returns** object containing gathered information from all supplied arguments

**Return type** *Multicolor*

**static similarity_score**(*multicolor1*, *multicolor2*)
Computes how similar two *Multicolor* objects are from perspective of information, that they contain.

Two multicolors are called to be similar if they contain same colors (at least one). Multiplicity of colors is taken into account as well.

**Parameters**

- **multicolor1** (*Multicolor*) – first out of two multi-colors to compute similarity between

- **multicolor2** (*Multicolor*) – second out of two multi-colors to compute similarity between

**Returns** the similarity score between two supplied *Multicolor* object

**Return type** int

**classmethod split_colors**(*multicolor*, *guidance=None*, *sorted_guidance=False*, *account_for_color_multiplicity_in_guidance=True*)

Produces several new instances of *Multicolor* object by splitting information about colors by using provided guidance iterable set-like object.

Guidance is an iterable type of object where each entry has information about groups of colors that has to be separated for current Multicolor.multicolors chunk. If no Guidance is provided, single-color guidance of Multicolor.multicolors is created. Guidance object is first reversed sorted to iterate over it from larges color set to the smallest one, as small color sets might be subsets of bigger ones, and shall be utilized only if bigger sets didn't help in separating.

During the first iteration over the guidance information all subsets of Multicolor.multicolors that equal to entries of guidance are recorded. During second iteration over remaining of the guidance information, if colors in Multicolor.multicolors form subsets of guidance entries, such instances are recorded. After this two iterations, the rest of Multicolor.multicolors is recorded as non-tackled and is recorded on its own.

Multiplicity of all separated colors in respective chunks is preserved.

Accounts for subclassing.

**Parameters**

- **multicolor** (*Multicolor*) – an instance information about colors in which is to be split
- **guidance** (*iterable where each entry is iterable with colors entries*) – information how colors have to be split in current *Multicolor* object
- **sorted_guidance** – a flag, that indicates is sorting of provided guidance is in order

**Returns** a list of new *Multicolor* object colors information in which complies with guidance information

**Return type** list of *Multicolor* objects

**update**(**args*)

Updates information about colors and their multiplicity in respective *Multicolor* instance.

By iterating over supplied arguments each of which should represent a color object, updates information about colors and their multiplicity in current *Multicolor* instance.

**Parameters** **args** (*any hashable python object*) – variable number of colors to add to currently existing multi colors data

**Returns** None, performs inplace changes to Multicolor.multicolors attribute

### 2.2.6 edge.py

**class** bg.edge.**BGEdge**(*vertex1*, *vertex2*, *multicolor*, *data=None*)

Bases: object

A wrapper class for edges in *bg.breakpoint_graph.BreakpointGraph*

Is not stored on its own in the *bg.breakpoint_graph.BreakpointGraph*, but is rather can be supplied to work with and is returned if retrieval is performed. BGEdge represents an undirected edge, thus distinction

between BGEdge.vertex1 and BGEdge.vertex2 attributes is just from identities perspective, not from the order perspective.

This class supports th following attributes, that cary information about multi-color for this edge, as well as vertices, its is attached to:

- BGEdge.vertex1: a first vertex to be utilized in *bg.breakpoint_graph.BreakpointGraph*
- BGEdge.vertex2: a second vertex to be utilized in *bg.breakpoint_graph.BreakpointGraph*

Main operations:

- ==
- *BGEdge.merge()*: produces a new BGEdge with multi-color information being merged from them

**class BGEdgeJSONSchema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: marshmallow.schema.Schema

Marshmallow powered JSON schema used for serialization / deserialization of edge object

**static _BGEdge__vertex_json_id**(*vertex*)

A proxy property based access to vertices in current edge

When edge is serialized to JSON object, no explicit object for its vertices are created, but rather they are referenced by special vertex json_ids.

**__eq__**(*other*)

Implementation of == operation for *BGEdge*

**Checks if current *BGEdge* instance comply in terms of vertices set with the supplied *BGEdge*, and then checks the ec**

*BGEdge* does not equal to non-*BGEdge* objects

**Parameters** **other** (*any python object*) – object to compare current *BGEdge* to

**Returns** flag of equality if current *BGEdge* object equals to the supplied one

**Return type** Boolean

**__init__**(*vertex1*, *vertex2*, *multicolor*, *data=None*)

Initialization of *BGEdge* object.

**Parameters**

- **vertex1** (*any hashable python object*) – vertex the edges is outgoing from
- **vertex2** (*any hashable python object*) – vertex the edges is ingoing to
- **multicolor** (*bg.multicolor.Multicolor*) – multicolor that this single edge shall posses

**Returns** a new instance of *BGEdge*

**Return type** *BGEdge*

**colors_json_ids**

A proxy property based access to vertices in current edge

When edge is serialized to JSON object, no explicit object for its multicolor is created, but rather all colors, taking into account their multiplicity, are referenced by their json_ids.

**classmethod from_json**(*data*, *json_schema_class=None*)
JSON deserialization method that retrieves edge instance from its json representation

If specific json schema is provided, it is utilized, and if not, a class specific is used

**json_schema_name**
When genome is serialized information about JSON schema of such serialization can be recorded, and this property provides access to it

**classmethod merge**(*edge1*, *edge2*)
Merges multi-color information from two supplied *BGEdge* instances into a new *BGEdge*

Since *BGEdge* represents an undirected edge, created edge's vertices are assign according to the order in first supplied edge.

Accounts for subclassing.

**Parameters**

- **edge1** – first out of two edge information from which is to be merged into a new one
- **edge2** – second out of two edge information from which is to be merged into a new one

**Returns** a new undirected with multi-color information merged from two supplied *BGEdge* objects

**Raises** ValueError

**to_json**(*schema_info=True*)
JSON serialization method that accounts for a possibility of field filtration and schema specification

**vertex1_json_id**
First vertex json id access

**vertex2_json_id**
Second vertex json id access

### 2.2.7 vertices.py

**class** bg.vertices.**BGVertex**(*name*)
Bases: object

An base class that represents a vertex (node) with all associated information in a breakpoint graph data structure

While class represents a base inheritance point for specific vertex implementations, it does implement most of business logic operations, that vertex shall support.

While different type of vertices are to be represented with different python classes, they all have a string representation, which mainly relies one the *name* attribute.

**class BGVertexJSONSchema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)
Bases: marshmallow.schema.Schema

**__eq__**(*other*)
Return self==value.

**__hash__**()
Return hash(self).

**__init__**(*name*)
Initialize self. See help(type(self)) for accurate signature.

**__ne__**(*other*)
Return self!=value.

**__str__**()
Return str(self).

**class** bg.vertices.**BlockVertex**(*name*, *mate_vertex=None*)
Bases: *bg.vertices.BGVertex*

This class represents a special type of breakpoint graph vertex that correspond to a generic block extremity (gene/ synteny block/ etc.)

**class BlockVertexJSONSchema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)
Bases: bg.vertices.BGVertexJSONSchema

JSON schema for this class is redefined to tune the *make_object* method, that shall return *BlockVertex* instance, rather than *BGVertex* one

**__init__**(*name*, *mate_vertex=None*)
Initialize self. See help(type(self)) for accurate signature.

**classmethod from_json**(*data*, *json_schema_class=None*)
This class overwrites the from_json method thus, making sure, that if *from_json* is called from this class instance, it will provide its JSON schema as a default one

**is_block_vertex**
This class implements a property check for vertex to belong to a class of vertices, that correspond to extremities of genomic blocks

**is_regular_vertex**
This class implements a property check for vertex to belong to class of regular vertices

**class** bg.vertices.**InfinityVertex**(*name*)
Bases: *bg.vertices.BGVertex*

This class represents a special type of breakpoint graph vertex that correspond to a generic extremity of genomic fragment (chromosome, scaffold, contig, etc.)

**class InfinityVertexJSONSchema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)
Bases: bg.vertices.BGVertexJSONSchema

JSON Schema for this class is redefined to tune the *make_object* method, that shall return *InfinityVertex* instance, rather than a *BGVertex* one

**__init__**(*name*)
Initialize self. See help(type(self)) for accurate signature.

**classmethod from_json**(*data*, *json_schema_class=None*)
This class overwrites the from_json method, thus making sure that if *from_json* is called from this class instance, it will provide its JSON schema as a default one

**is_infinity_vertex**
This class implements a property check for vertex to belong to a class of vertices, that correspond to standard extremities of genomic fragments

**is_irregular_vertex**
This class implements a property check for vertex to belong to a class of vertices, that correspond to extremities of genomic fragments

**name**
access to classic name attribute is hidden by this property

**class** bg.vertices.**TaggedBlockVertex**(*name*, *mate_vertex=None*)
Bases: *bg.vertices.BlockVertex*, *bg.vertices.TaggedVertex*

**class TaggedBlockVertexJSONSchema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)
Bases: bg.vertices.TaggedVertexJSONSchema, bg.vertices.BlockVertexJSONSchema

**class** bg.vertices.**TaggedInfinityVertex**(*name*)
Bases: *bg.vertices.InfinityVertex*, *bg.vertices.TaggedVertex*

**class TaggedInfinityVertexJSONSchema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)
Bases: bg.vertices.TaggedVertexJSONSchema, bg.vertices.InfinityVertexJSONSchema

**class** bg.vertices.**TaggedVertex**(*name*)
Bases: *bg.vertices.BGVertex*

**class TaggedVertexJSONSchema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)
Bases: bg.vertices.BGVertexJSONSchema

**__getattr__**(*item*)

**__init__**(*name*)
Initialize self. See help(type(self)) for accurate signature.

**add_tag**(*tag*, *value*)
as tags are kept in a sorted order, a bisection is a fastest way to identify a correct position of or a new tag to be added. An additional check is required to make sure w don't add duplicates

**classmethod from_json**(*data*, *json_schema_class=None*)
This class overwrites the from_json method, thus making sure that if *from_json* is called from this class instance, it will provide its JSON schema as a default one

**name**
access to classic name attribute is hidden by this property

**remove_tag**(*tag*, *value*, *silent_fail=False*)
we try to remove supplied pair tag – value, and if does not exist outcome depends on the silent_fail flag

### 2.2.8 genome.py

**class** bg.genome.**BGGenome**(*name*)
Bases: object

A class that represent a genome object for the breakpoint graph

For purposes of breakpoint graph no additional information about genome is needed, except its name, that is used in various algorithmic tasks (multicolor splitting, tree traversing, etc)

**class BGGenomeJSONSchema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: `marshmallow.schema.Schema`

a JSON schema powered by marshmallow library to serialize/deserialize genome object into/from JSON format

**__eq__**(*other*)

Two genomes a called equal if they are of same class and their hash values are equal to each other

**__hash__**()

Since for breakpoint graph purposes distinction between genomes is made purely by their name, hash value of genome is proxied to hash value of genomes name

**__init__**(*name*)

Initialize self. See help(type(self)) for accurate signature.

**__le__**(*other*)

Genomes are ordered according to lexicographical ordering of their names

**__lt__**(*other*)

Genomes are ordered according to lexicographical ordering of their names

**classmethod from_json**(*data*, *json_schema_class=None*)

JSON deserialization method that retrieves a genome instance from its json representation

If specific json schema is provided, it is utilized, and if not, a class specific is used

**json_id**

A genome is references multiple times, as for example in multicolor object, and such reference is done by genome unique json id.

**json_schema_name**

When genome is serialized information about JSON schema of such serialization can be recorded, and this property provides access to it

**to_json**(*schema_info=True*)

JSON serialization method that accounts for a possibility of field filtration and schema specification

### 2.2.9 distances.py

### 2.2.10 util.py

CHAPTER 3

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## b

# Index

## Symbols

## A



## B



## C



## D



## E



## F



## G



## H