Python `strat` module

This repository is an implementation of parts of the following papers:

[1] J.C. Gómez-Larrañaga, F. González-Acuña, Wolfgang Heil. Classification of Simply-connected Trivalent 2-dimensional Stratifolds, Top. Proc. (2018)

[2] J.C. Gómez-Larrañaga, F. González-Acuña, Wolfgang Heil. Models of Simply-connected Trivalent 2-dimensional Stratifolds.

It is implemented using NetworkX 2.1 and Python 3.6.

This module has a 'stratifold graph' class (strat_graph), which was implemented as a subclass of the multigraph class from the python NetworkX module. This class has the following methods:

- `__init__(self,black=[],white=[],edges=None)`: the initializer of this class. `black` and `white` should be iterables containing the black and white vertices, respectively. `edges` is an optional argument, an iterable with edges in the following format: `(vertex1,vertex2[,label])`. If `label` is not given, it defaults to 1. If `edges` is given, all vertices involved should already belong to `white` or `black`.

- `addEdg(self,edges)`: a function for adding edges. `edges` must be an iterable with the edges in the following format:

$$(\texttt{white\_vertex,black\_vertex[,label]}).$$

  As before, if `label` is not given, the label of the edge defaults to 1.

- `addNod(self, black=[], white=[])`: a funcion for adding black or white vertices. Receives `black`, `white`, iterables containing the vertices to be added. If an existing vertex is added, nothing happens to that specific vertex.

- `black_vals(self)`: returns a dictionary where the keys are the black vertices and values are the sum of the labels of the edges incident to that node (which, in the case of a trivalent stratifold, must equal 3).

- `is_trivalent(self)`: returns True if the graph is trivalent, False otherwise.

- `white(self)`: returns a set object with the white vertices. `black(self)` does the same for black vertices.

- `copy(self)`: returns a copy of the `strat_graph` instance.

- `is_horned_tree(self)`: returns True if the graph is a horned tree (definition 3.3 from [?]), and False otherwise.

- `is_21_collapsible(self)`: if `self` is a 2,1-collapsible tree (defined before lemma 3.2 in [?]), returns the root of the tree. Otherwise, returns `None`.

- `subg(self,nodes)`: returns a subgraph from `self` with `nodes` as set of nodes, where an edge occurs if and only both vertices belong to `nodes`. All labels are preserved.

- `St_B(self)`: returns a list of the connected components of $St(B)$ (notation from [?]: $St(B)$ is the closed star of B, the set of vertices with degree 3), where each component is a `strat_graph` instance.

- `graph_stB(self)`: returns a list of the components of $G \setminus st(B)$, where (using notation from [?]) $st(B)$ is the open star of B.

- `is_simply_connected(self)`: implementation of theorem 3.6 from [?]. Returns True if self is the graph of a simply connected trivalent 2-stratifold.

- `O1(self,white_node,black_nodes1,black_nodes2)`: returns a copy of `self` where operation $O1$ has been performed on the vertex `white_node`. `black_nodes1`, `black_nodes2` are the iterables with the two groups of neighbours of `white_node` that will determine how $O1$ will be performed. The new vertices are named automatically (see below).

- `O1_1(self,node,other,node_other, black=None,white=None)`: performs operation $O1*$ if `self` and `other` are disjoint graphs. If new nodes `black`, `white` are not given, they are created automatically (see below). `node` and `node_other` are the nodes where $O1*$ will be performed. Returns a new `strat_graph` object.

- `O2(self,white_node,new_white=None,new_black=None)`: performs operation $O2$ on vertex `white_node`. If new vertices `new_white`, `new_black` are not given, they are created automatically (see below). Unlike `O1,O1_1`, this operation is performed in place; no new instance is created. If original graph should be saved, user must create a copy and then perform $O2$ on that copy.

- `draw(self)`: Function for drawing the graph. Calls the function draw from networkx, coloring black vertices black and white vertices gray. Intended for trivalent graphs, draws edges with label 2 wider than those with label 1.

Additionaly, the module has functions `b111(black=None,white=None)`, `b12(black=None,white=None)` for generating $b111$- and $b12$-trees. Each of these functions may receive the names of the vertices; for the names to be assigned correctly, it must receive both. If `black` or `white` are not received, names will be assigned automatically (see below).

Automatic name assignment: by default, white vertices are named with integers and black ones with letters (strings); this does not need to be always the case, it is just fur neatness. For this, the following generators are defined:

- `get_int(used=[])`: yields integers not in `used` starting from 0; for example, the generator returned by the call `get_int(used=[1,3]` yields the sequence $0, 2, 3, 4, 5 \dots$.

- `get_str(used=[],n=1)`: yields strings not in `used` in the following order:

$$a, b, \dots, z, aa, ab, \dots, zx, zy, zz, aaa, aab, \dots$$

The argument n is the starting length of the string.

The automatic naming of vertices in operations $O1, O1*, O2$ calls the functions above defined by passing as `used` the current set of vertices of `self`.

The previous functions is also useful when generating many $b - 111$ or $b - 12$ trees automatically (see last example in `examples.py`).