gedlibpy Documentation

Release 1.0

Natacha Lambert

Aug 23, 2019

CONTENTS:

This module allow to use a C++ library for edit distance between graphs (GedLib) with Python.

Before using, please read the first section to ensure a good start with the library. Then, you can follow some examples or informations about each method.

CHAPTER

ONE

HOW TO INSTALL THIS LIBRARY

Please Read https://dbblumenthal.github.io/gedlib/ before using Python code. You can also find this module documentation in documentation/build/html folder.

Make sure you have numpy installed (and Cython if you have to recompile the library). You can use pip for this.

1.1 Running the script

After donwloading the entire folder, you can run test.py to ensure the library works.

For your code, you have to make two imports:

```
import librariesImport
import gedlibpy
```

You can call each function in the library with this. You can't move any folder or files on the library, please make sure that the architecture remains the same.

This library is compiled for Python3 only. If you want to use it with Python 2, you have to recompile it with setup.py. You have to use this command on your favorite shell:

```
python setup.py build_ext --inplace
```

After this step, you can use the same lines as Python3 for import, it will be ok. Check the documentation inside the documentation/build/html folder before using function. You can also copy the tests examples for basic use.

1.2 A problem with the library ?

If the library isn't found, you can recompile the Python library because your Linux is different to mine. Please delete gedlibpy.so, gedlibpy.cpp and build folder. Then use this command on a linux shell

```
python3 setup.py build_ext --inplace
```

You can make it with Python 2 but make sure you use the same version with your code and the compilation.

If it's doesn't work, maybe the version of GedLib or another library can be a problem. If it is, you can re-install GedLib for your computer. You can download it on this git : https://dbblumenthal.github.io/gedlib/

You have to install Gedlib with the Python installer after that. Just call:

```
python3 install.py
```

Make the links like indicate on the documentation. Use the same architecture like this library, but just change the .so and folders with your installation. You can recompile the Python library with setup command, after that.

If you use Mac OS, you have to follow all this part, and install the external libraries with this command:

```
install_name_tool -change <mylib> <path>/<to>/<mylib> <myexec>
```

For an example, you have to write these lines:

```
install_name_tool -change libdoublefann.2.dylib lib/fann/libdoublefann.2.dylib␣
↪gedlibpy.so
install_name_tool -change libsvm.so lib/libsvm.3.22/libsvm.so gedlibpy.so
install_name_tool -change libnomad.so lib/nomad/libnomad.so gedlibpy.so
install_name_tool -change libsgtelib.so lib/nomad/libsgtelib.so gedlibpy.so
```

The name of the library gedlibpy can be different if you use Python 3.

If your problem is still here, you can contact me on : natacha.lambert@unicaen.fr

1.3 How to use this library

This library allow to compute edit distance between two graphs. You have to follow these steps to use it :

- Add your graphs (GXL files, NX Structures or your structure, make sure that the internal type is the same)
- Choose your cost function
- Init your environnment (After that, the cost function and your graphs can't be modified)
- Choose your method computation
- Run the computation with the IDs of the two graphs. You can have the ID when you add the graph or with some functions
- Find the result with differents functions (NodeMap, edit distance, etc)

Here is an example of code with GXL graphs:

```
gedlibpy.load_GXL_graphs('include/gedlib-master/data/datasets/Mutagenicity/data/',
↪'collections/MUTA_10.xml')
listID = gedlibpy.get_all_graph_ids()
gedlibpy.set_edit_cost("CHEM_1")
gedlibpy.init()
gedlibpy.set_method("IPFP", "")
gedlibpy.init_method()
g = listID[0]
h = listID[1]

gedlibpy.run_method(g,h)

print("Node Map : ", gedlibpy.get_node_map(g,h))
print ("Upper Bound = " + str(gedlibpy.get_upper_bound(g,h)) + ", Lower Bound = " +␣
↪str(gedlibpy.get_lower_bound(g,h)) + ", Runtime = " + str(gedlibpy.get_
↪runtime(g,h)))
```

Please read the documentation for more examples and functions.

1.4 An advice if you don't code in a shell

Python library don't indicate each C++ error. If you have a restart causing by an error in your code, please use on a linux shell for having C++ errors.

CHAPTER

TWO

HOW TO ADD YOUR OWN EDITCOST CLASS

When you choose your cost function, you can decide some parameters to personalize the function. But if you have some graphs which its type doesn't correpond to the choices, you can create your edit cost function.

For this, you have to write it in C++.

2.1 C++ side

You class must inherit to EditCost class, which is an asbtract class. You can find it here : include/gedlib-master/src/edit_costs

You can inspire you to the others to understand how to use it. You have to override these functions :

- virtual double node_ins_cost_fun(const UserNodeLabel & node_label) const final;
- virtual double node_del_cost_fun(const UserNodeLabel & node_label) const final;
- virtual double node_rel_cost_fun(const UserNodeLabel & node_label_1, const UserNodeLabel & node_label_2) const final;
- virtual double edge_ins_cost_fun(const UserEdgeLabel & edge_label) const final;
- virtual double edge_del_cost_fun(const UserEdgeLabel & edge_label) const final;
- virtual double edge_rel_cost_fun(const UserEdgeLabel & edge_label_1, const UserEdgeLabel & edge_label_2) const final;

You can add some attributes for parameters use or more functions, but these are unavoidable.

When your class is ready, please go to the C++ Bind here : src/GedLibBind.cpp . The function is :

> void setPersonalEditCost(std::vector<double> editCostConstants){env.set_edit_costs(Your EditCost Class(editCostConstants));}

You have just to initialize your class. Parameters aren't mandatory, empty by default. If your class doesn't have one, you can skip this. After that, you have to recompile the project.

2.2 Python side

For this, use setup.py with this command in a linux shell:

```
python3 setup.py build_ext --inplace
```

You can also make it in Python 2.

Now you can use your edit cost function with the Python function set_personal_edit_cost(edit_cost_constant).

If you want more informations on C++, you can check the documentation of the original library here : https://github.com/dbblumenthal/gedlib

CHAPTER

THREE

EXAMPLES

Before using each example, please make sure to put these lines on the beginnig of your code :

```
import librariesImport
import gedlibpy
```

Use your path to access it, without changing the library architecture. After that, you are ready to use the library.

When you want to make new computation, please use this function :

```
gedlibpy.restart_env()
```

All the graphs and results will be delete so make sure you don't need it.

3.1 Classique case with GXL graphs

```
gedlibpy.load_GXL_graphs('include/gedlib-master/data/datasets/Mutagenicity/data/',
↪'collections/MUTA_10.xml')
listID = gedlibpy.get_all_graph_ids()
gedlibpy.set_edit_cost("CHEM_1")

gedlibpy.init()

gedlibpy.set_method("IPFP", "")
gedlibpy.init_method()

g = listID[0]
h = listID[1]

gedlibpy.run_method(g,h)

print("Node Map : ", gedlibpy.get_node_map(g,h))
print ("Upper Bound = " + str(gedlibpy.get_upper_bound(g,h)) + ", Lower Bound = " +␣
↪str(gedlibpy.get_lower_bound(g,h)) + ", Runtime = " + str(gedlibpy.get_
↪runtime(g,h)))
```

You can also use this function :

```
compute_edit_distance_on_GXl_graphs(path_folder, path_XML, edit_cost, method, options=
↪"", init_option = "EAGER_WITHOUT_SHUFFLED_COPIES")
```

This function compute all edit distance between all graphs, even itself. You can see the result with some functions and graphs IDs. Please see the documentation of the function for more informations.

3.2 Classique case with NX graphs

```
for graph in dataset :
  gedlibpy.add_nx_graph(graph, classe)
listID = gedlibpy.get_all_graph_ids()
gedlibpy.set_edit_cost("CHEM_1")

gedlibpy.init()

gedlibpy.set_method("IPFP", "")
gedlibpy.init_method()

g = listID[0]
h = listID[1]

gedlibpy.run_method(g,h)

print("Node Map : ", gedlibpy.get_node_map(g,h))
print ("Upper Bound = " + str(gedlibpy.get_upper_bound(g,h)) + ", Lower Bound = " +␣
↪str(gedlibpy.get_lower_bound(g,h)) + ", Runtime = " + str(gedlibpy.get_
↪runtime(g,h)))
```

You can also use this function :

```
compute_edit_distance_on_nx_graphs(dataset, classes, edit_cost, method, options, init_
↪option = "EAGER_WITHOUT_SHUFFLED_COPIES")
```

This function compute all edit distance between all graphs, even itself. You can see the result in the return and with some functions and graphs IDs. Please see the documentation of the function for more informations.

Or this function :

```
compute_ged_on_two_graphs(g1,g2, edit_cost, method, options, init_option = "EAGER_
↪WITHOUT_SHUFFLED_COPIES")
```

This function allow to compute the edit distance just for two graphs. Please see the documentation of the function for more informations.

3.3 Add a graph from scratch

```
currentID = gedlibpy.add_graph()
gedlibpy.add_node(currentID, "_1", {"chem" : "C"})
gedlibpy.add_node(currentID, "_2", {"chem" : "O"})
gedlibpy.add_edge(currentID,"_1", "_2",  {"valence": "1"} )
```

Please make sure as the type are the same (string for Ids and a dictionnary for labels). If you want a symmetrical graph, you can use this function to ensure the symmetry :

```
add_symmetrical_edge(graph_id, tail, head, edge_label)
```

If you have a Nx structure, you can use directly this function :

```
add_nx_graph(g, classe, ignore_duplicates=True)
```

Even if you have another structure, you can use this function :

```
add_random_graph(name, classe, list_of_nodes, list_of_edges, ignore_duplicates=True)
```

Please read the documentation before using and respect the types.

3.4 Median computation

An example is available in the Median_Example folder. It contains the necessary to compute a median graph. You can launch xp-letter-gbr.py to compute median graph on all letters in the dataset, or median.py for le letter Z.

To summarize the use, you can follow this example :

```
import pygraph #Available with the median example
from median import draw_Letter_graph, compute_median, compute_median_set

gedlibpy.load_GXL_graphs('../include/gedlib-master/data/datasets/Letter/HIGH/', '../
↪include/gedlib-master/data/collections/Letter_Z.xml')
gedlibpy.set_edit_cost("LETTER")
gedlibpy.init()
gedlibpy.set_method("IPFP", "")
gedlibpy.init_method()
listID = gedlibpy.get_all_graph_ids()

dataset,my_y = pygraph.utils.graphfiles.loadDataset("../include/gedlib-master/data/
↪datasets/Letter/HIGH/Letter_Z.cxl")
median, sod, sods_path,set_median = compute_
↪median(gedlibpy,listID,dataset,verbose=True)
draw_Letter_graph(median)
```

Please use the function in the median.py code to simplify your use. You can adapt this example to your case. Also, some function in the PythonGedLib module can make the work easier. Ask Benoît Gauzere if you want more information.

3.5 Hungarian algorithm

3.5.1 LSAPE

```
result = gedlibpy.hungarian_LSAPE(matrixCost)
print("Rho = ", result[0], " Varrho = ", result[1], " u = ", result[2], " v = ",␣
↪result[3])
```

3.5.2 LSAP

```
result = gedlibpy.hungarian_LSAP(matrixCost)
print("Rho = ", result[0], " u = ", result[1], " v = ", result[2], " Varrho = ",␣
↪result[3])
```

CHAPTER

FOUR

PYTHON GEDLIB MODULE

This module allow to use a C++ library for edit distance between graphs (GedLib) with Python.

4.1 Authors

David Blumenthal Natacha Lambert

4.2 Classes & Functions

exception gedlibpy.**EditCostError**
Class for Edit Cost Error. Raise an error if an edit cost function doesn't exist in the library (not in list_of_edit_cost_options).

Attribute message The message to print when an error is detected.

exception gedlibpy.**Error**
Class for error's management. This one is general.

exception gedlibpy.**InitError**
Class for Init Error. Raise an error if an init option doesn't exist in the library (not in list_of_init_options).

Attribute message The message to print when an error is detected.

exception gedlibpy.**MethodError**
Class for Method Error. Raise an error if a computation method doesn't exist in the library (not in list_of_method_options).

Attribute message The message to print when an error is detected.

gedlibpy.**add_edge**()
Adds an edge on a graph selected by its ID.

Parameters

- **graph_id** (*size_t*) – The ID of the wanted graph
- **tail** (*string*) – The ID of the tail node for the new edge
- **head** (*string*) – The ID of the head node for the new edge
- **edge_label** (*dict{string : string}*) – The label of the new edge
- **ignore_duplicates** (*bool*) – If True, duplicate edges are ignored, otherwise it's raise an error if an existing edge is added. True by default

See also:

add_graph(), add_node(), add_symmetrical_edge()

Note: You can also use this function after initialization, but only on a newly added graph. Call init() after you're finished your modifications.

---

gedlibpy.**add_graph**()

Adds a empty graph on the environment, with its name and its class. Nodes and edges will be add in a second time.

Parameters

- **name** (*string*) – The name of the new graph, an empty string by default
- **classe** (*string*) – The class of the new graph, an empty string by default

Returns The ID of the newly graphe

Return type size_t

Note: You can call this function without parameters. You can also use this function after initialization, call init() after you're finished your modifications.

---

gedlibpy.**add_node**()

Adds a node on a graph selected by its ID. A ID and a label for the node is required.

Parameters

- **graph_id** (*size_t*) – The ID of the wanted graph
- **node_id** (*string*) – The ID of the new node
- **node_label** (*dict{string : string}*) – The label of the new node

See also:

add_graph(), add_edge(), add_symmetrical_edge()

Note: You can also use this function after initialization, but only on a newly added graph. Call init() after you're finished your modifications.

---

gedlibpy.**add_nx_graph**()

Add a Graph (made by networkx) on the environment. Be careful to respect the same format as GXL graphs for labelling nodes and edges.

Parameters

- **g** (*networkx.graph*) – The graph to add (networkx graph)
- **ignore_duplicates** (*bool*) – If True, duplicate edges are ignored, otherwise it's raise an error if an existing edge is added. True by default

Returns The ID of the newly added graphe

Return type size_t

Note: The NX graph must respect the GXL structure. Please see how a GXL graph is construct.

---

gedlibpy.**add_random_graph**()

Add a Graph (not GXL) on the environment. Be careful to respect the same format as GXL graphs for labelling nodes and edges.

Parameters

- **name** (*string*) – The name of the graph to add, can be an empty string
- **classe** (*string*) – The classe of the graph to add, can be an empty string
- **list_of_nodes** (*list[tuple(size_t, dict{string : string})]*) – The list of nodes to add
- **list_of_edges** (*list[tuple(tuple(size_t,size_t), dict{string : string})]*) – The list of edges to add
- **ignore_duplicates** (*bool*) – If True, duplicate edges are ignored, otherwise it's raise an error if an existing edge is added. True by default

Returns The ID of the newly added graphe

Return type size_t

Note: The graph must respect the GXL structure. Please see how a GXL graph is construct.

---

gedlibpy.**add_symmetrical_edge**()

Adds a symmetrical edge on a graph selected by its ID.

Parameters

- **graph_id** (*size_t*) – The ID of the wanted graph
- **tail** (*string*) – The ID of the tail node for the new edge
- **head** (*string*) – The ID of the head node for the new edge
- **edge_label** (*dict{string : string}*) – The label of the new edge

See also:

add_graph(), add_node(), add_edge()

Note: You can also use this function after initialization, but only on a newly added graph. Call init() after you're finished your modifications.

---

gedlibpy.**clear_graph**()

Deletes a graph, selected by its ID, to the environment.

Parameters **graph_id** (*size_t*) – The ID of the wanted graph

---

Note: Call init() after you're finished your modifications.

---

gedlibpy.**compute_edit_distance_on_GXl_graphs**()

Computes all the edit distance between each GXL graphs on the folder and the XMl file.

Parameters

- **path_folder** (*string*) – The folder's path which contains GXL graphs
- **path_XML** (*string*) – The XML's path which indicates which graphes you want to load

- **edit_cost** (*string*) – The name of the edit cost function
- **method** (*string*) – The name of the computation method
- **options** (*string*) – The options of the method (like bash options), an empty string by default
- **init_option** (*string*) – The name of the init option, "EAGER_WITHOUT_SHUFFLED_COPIES" by default

Returns The list of the first and last-1 ID of graphs

Return type tuple(size_t, size_t)

See also:

list_of_edit_cost_options, list_of_method_options, list_of_init_options

---

Note: Make sure each parameter exists with your architecture and these lists : list_of_edit_cost_options, list_of_method_options, list_of_init_options.

---

gedlibpy.**compute_edit_distance_on_nx_graphs**()

Computes all the edit distance between each NX graphs on the dataset.

Parameters

- **dataset** (*list[networksx.graph]*) – The list of graphs to add and compute
- **classes** (*string*) – The classe of all the graph, can be an empty string
- **edit_cost** (*string*) – The name of the edit cost function
- **method** (*string*) – The name of the computation method
- **options** (*string*) – The options of the method (like bash options), an empty string by default
- **init_option** (*string*) – The name of the init option, "EAGER_WITHOUT_SHUFFLED_COPIES" by default

Returns Two matrix, the first with edit distances between graphs and the second the nodeMap between graphs. The result between g and h is one the [g][h] coordinates.

Return type list[list[double]], list[list[list[tuple(size_t, size_t)]]]

See also:

list_of_edit_cost_options, list_of_method_options, list_of_init_options

---

Note: Make sure each parameter exists with your architecture and these lists : list_of_edit_cost_options, list_of_method_options, list_of_init_options. The structure of graphs must be similar as GXL.

---

gedlibpy.**compute_ged_on_two_graphs**()

Computes the edit distance between two NX graphs.

Parameters

- **g1** (*networksx.graph*) – The first graph to add and compute
- **g2** (*networksx.graph*) – The second graph to add and compute
- **edit_cost** (*string*) – The name of the edit cost function

- **method** (*string*) – The name of the computation method
- **options** (*string*) – The options of the method (like bash options), an empty string by default
- **init_option** (*string*) – The name of the init option, "EAGER_WITHOUT_SHUFFLED_COPIES" by default

Returns The edit distance between the two graphs and the nodeMap between them.

Return type double, list[tuple(size_t, size_t)]

See also:

list_of_edit_cost_options, list_of_method_options, list_of_init_options

---

Note: Make sure each parameter exists with your architecture and these lists : list_of_edit_cost_options, list_of_method_options, list_of_init_options. The structure of graphs must be similar as GXL.

---

gedlibpy.**encode_your_map**()

Encodes a string dictionnary to utf-8 for C++ functions

Parameters **map** (*dict{string : string}*) – The map to encode

Returns The encoded map

Return type dict{'b'string : 'b'string}

Note: This function is used for type connection.

---

gedlibpy.**get_all_graph_ids**()

Searchs all the IDs of the loaded graphs in the environment.

Returns The list of all graphs's Ids

Return type list[size_t]

Note: The last ID is equal to (number of graphs - 1). The order correspond to the loading order.

---

gedlibpy.**get_all_map**()

Returns a vector which contains the forward and the backward maps between nodes of the two indicated graphs.

Parameters

- **g** (*size_t*) – The Id of the first compared graph
- **h** (*size_t*) – The Id of the second compared graph

Returns The forward and backward maps to the adjacence matrix between nodes of the two graphs

Return type list[list[npy_uint32]]

See also:

run_method(), get_upper_bound(), get_lower_bound(), get_forward_map(), get_backward_map(), get_runtime(), quasimetric_cost()

Warning: run_method() between the same two graph must be called before this function.

Note: This function duplicates data so please don't use it. I also don't know how to connect the two map to reconstruct the adjacence matrix. Please come back when I know how it's work !

gedlibpy.**get_assignment_matrix**()

Returns the Assignment Matrix between two selected graphs g and h.

Parameters

- **g** (*size_t*) – The Id of the first compared graph
- **h** (*size_t*) – The Id of the second compared graph

Returns The Assignment Matrix between the two selected graph.

Return type list[list[int]]

See also:

run_method(), get_forward_map(), get_backward_map(), get_node_image(), get_node_pre_image(), get_node_map()

Warning: run_method() between the same two graph must be called before this function.

Note: This function creates datas so use it if necessary.

gedlibpy.**get_backward_map**()

Returns the backward map (or the half of the adjacence matrix) between nodes of the two indicated graphs.

Parameters

- **g** (*size_t*) – The Id of the first compared graph
- **h** (*size_t*) – The Id of the second compared graph

Returns The backward map to the adjacence matrix between nodes of the two graphs

Return type list[npy_uint32]

See also:

run_method(), get_upper_bound(), get_lower_bound(), get_forward_map(), get_runtime(), quasimetric_cost(), get_node_map(), get_assignment_matrix()

Warning: run_method() between the same two graph must be called before this function.

Note: I don't know how to connect the two map to reconstruct the adjacence matrix. Please come back when I know how it's work !

gedlibpy.**get_dummy_node**()

Returns the ID of a dummy node.

Returns The ID of the dummy node (18446744073709551614 for my computer, the hugest number possible)

Return type size_t

Note: A dummy node is used when a node isn't associated to an other node.

---

gedlibpy.**get_edit_cost_options**()
Searchs the differents edit cost functions and returns the result.

Returns The list of edit cost functions

Return type list[string]

Warning: This function is useless for an external use. Please use directly list_of_edit_cost_options.

Note: Prefer the list_of_edit_cost_options attribute of this module.

---

gedlibpy.**get_forward_map**()
Returns the forward map (or the half of the adjacence matrix) between nodes of the two indicated graphs.

Parameters

- **g** (*size_t*) – The Id of the first compared graph
- **h** (*size_t*) – The Id of the second compared graph

Returns The forward map to the adjacence matrix between nodes of the two graphs

Return type list[npy_uint32]

See also:

run_method(), get_upper_bound(), get_lower_bound(), get_backward_map(), get_runtime(), quasimetric_cost(), get_node_map(), get_assignment_matrix()

Warning: run_method() between the same two graph must be called before this function.

Note: I don't know how to connect the two map to reconstruct the adjacence matrix. Please come back when I know how it's work !

---

gedlibpy.**get_graph_adjacence_matrix**()
Searchs and returns the adjacence list of a graph, selected by its ID.

Parameters **graph_id** (*size_t*) – The ID of the wanted graph

Returns The adjacence list of the selected graph

Return type list[list[size_t]]

See also:

get_graph_internal_id(), get_graph_num_nodes(), get_graph_num_edges(), get_original_node_ids(), get_graph_node_labels(), get_graph_edges()

Note: These functions allow to collect all the graph's informations.

gedlibpy.**get_graph_class**()
Returns the class of a graph with its ID.

Parameters **id** (*size_t*) – The ID of the wanted graph

Returns The class of the graph which correpond to the ID

Return type string

See also:

get_graph_class()

Note: An empty string can be a class.

gedlibpy.**get_graph_edges**()
Searchs and returns all the edges on a graph, selected by its ID.

Parameters **graph_id** (*size_t*) – The ID of the wanted graph

Returns The list of edges on the selected graph

Return type dict{tuple(size_t,size_t) : dict{string : string}}

Note: These functions allow to collect all the graph's informations.

gedlibpy.**get_graph_internal_id**()
Searchs and returns the internal Id of a graph, selected by its ID.

Parameters **graph_id** (*size_t*) – The ID of the wanted graph

Returns The internal ID of the selected graph

Return type size_t

See also:

get_graph_num_nodes(), get_graph_num_edges(), get_original_node_ids(), get_graph_node_labels(), get_graph_edges(), get_graph_adjacence_matrix()

Note: These functions allow to collect all the graph's informations.

gedlibpy.**get_graph_name**()
Returns the name of a graph with its ID.

Parameters **id** (*size_t*) – The ID of the wanted graph

Returns The name of the graph which correpond to the ID

Return type string

See also:

get_graph_class()

Note: An empty string can be a name.

gedlibpy.**get_graph_node_labels**()
Searchs and returns all the labels of nodes on a graph, selected by its ID.

Parameters **graph_id** (*size_t*) – The ID of the wanted graph

Returns The list of labels's nodes on the selected graph

Return type list[dict{string : string}]

See also:

get_graph_internal_id(), get_graph_num_nodes(), get_graph_num_edges(), get_original_node_ids(), get_graph_edges(), get_graph_adjacence_matrix()

Note: These functions allow to collect all the graph's informations.

gedlibpy.**get_graph_num_edges**()
Searchs and returns the number of edges on a graph, selected by its ID.

Parameters **graph_id** (*size_t*) – The ID of the wanted graph

Returns The number of edges on the selected graph

Return type size_t

See also:

get_graph_internal_id(), get_graph_num_nodes(), get_original_node_ids(), get_graph_node_labels(), get_graph_edges(), get_graph_adjacence_matrix()

Note: These functions allow to collect all the graph's informations.

gedlibpy.**get_graph_num_nodes**()
Searchs and returns the number of nodes on a graph, selected by its ID.

Parameters **graph_id** (*size_t*) – The ID of the wanted graph

Returns The number of nodes on the selected graph

Return type size_t

See also:

get_graph_internal_id(), get_graph_num_edges(), get_original_node_ids(), get_graph_node_labels(), get_graph_edges(), get_graph_adjacence_matrix()

Note: These functions allow to collect all the graph's informations.

gedlibpy.**get_init_options**()
Searchs the differents initialization parameters for the environment computation for graphs and returns the result.

Returns The list of options to initialize the computation environment

Return type list[string]

Warning: This function is useless for an external use. Please use directly list_of_init_options.

Note: Prefer the list_of_init_options attribute of this module.

gedlibpy.**get_init_time**()
Returns the initialization time.

Returns The initialization time

Return type double

gedlibpy.**get_lower_bound**()

Returns the lower bound of the edit distance cost between two graphs g and h.

Parameters

- **g** (*size_t*) – The Id of the first compared graph
- **h** (*size_t*) – The Id of the second compared graph

Returns The lower bound of the edit distance cost

Return type double

See also:

run_method(), get_upper_bound(), get_forward_map(), get_backward_map(), get_runtime(), quasimetric_cost()

Warning: run_method() between the same two graph must be called before this function.

Note: This function can be ignored, because lower bound doesn't have a crucial utility.

gedlibpy.**get_method_options**()
Searchs the differents method for edit distance computation between graphs and returns the result.

Returns The list of method to compute the edit distance between graphs

Return type list[string]

Warning: This function is useless for an external use. Please use directly list_of_method_options.

Note: Prefer the list_of_method_options attribute of this module.

gedlibpy.**get_node_image**()
Returns the node's image in the adjacence matrix, if it exists.

Parameters

- **g** (*size_t*) – The Id of the first compared graph

- **h** (*size_t*) – The Id of the second compared graph
- **node_id** (*size_t*) – The ID of the node which you want to see the image

Returns The ID of the image node

Return type size_t

See also:

run_method(), get_forward_map(), get_backward_map(), get_node_pre_image(), get_node_map(), get_assignment_matrix()

Warning: run_method() between the same two graph must be called before this function.

Note: Use BackwardMap's Node to find its images ! You can also use get_forward_map() and get_backward_map().

gedlibpy.**get_node_map**()

Returns the Node Map, like C++ NodeMap.

Parameters

- **g** (*size_t*) – The Id of the first compared graph
- **h** (*size_t*) – The Id of the second compared graph

Returns The Node Map between the two selected graph.

Return type list[tuple(size_t, size_t)]

See also:

run_method(), get_forward_map(), get_backward_map(), get_node_image(), get_node_pre_image(), get_assignment_matrix()

Warning: run_method() between the same two graph must be called before this function.

Note: This function creates datas so use it if necessary, however you can understand how assignement works with this example.

gedlibpy.**get_node_pre_image**()

Returns the node's preimage in the adjacence matrix, if it exists.

Parameters

- **g** (*size_t*) – The Id of the first compared graph
- **h** (*size_t*) – The Id of the second compared graph
- **node_id** (*size_t*) – The ID of the node which you want to see the preimage

Returns The ID of the preimage node

Return type size_t

See also:

run_method(), get_forward_map(), get_backward_map(), get_node_image(), get_node_map(), get_assignment_matrix()

Warning: run_method() between the same two graph must be called before this function.

Note: Use ForwardMap's Node to find its images ! You can also use get_forward_map() and get_backward_map().

gedlibpy.**get_original_node_ids**()

Searchs and returns all th Ids of nodes on a graph, selected by its ID.

Parameters **graph_id** (*size_t*) – The ID of the wanted graph

Returns The list of IDs's nodes on the selected graph

Return type list[string]

Note: These functions allow to collect all the graph's informations.

gedlibpy.**get_runtime**()

Returns the runtime to compute the edit distance cost between two graphs g and h

Parameters

- **g** (*size_t*) – The Id of the first compared graph
- **h** (*size_t*) – The Id of the second compared graph

Returns The runtime of the computation of edit distance cost between the two selected graphs

Return type double

See also:

run_method(), get_upper_bound(), get_lower_bound(), get_forward_map(), get_backward_map(), quasimetric_cost()

Warning: run_method() between the same two graph must be called before this function.

Note: Python is a bit longer than C++ due to the functions's encapsulate.

gedlibpy.**get_upper_bound**()

Returns the upper bound of the edit distance cost between two graphs g and h.

Parameters

- **g** (*size_t*) – The Id of the first compared graph
- **h** (*size_t*) – The Id of the second compared graph

Returns The upper bound of the edit distance cost

Return type double

See also:

run_method(), get_lower_bound(), get_forward_map(), get_backward_map(), get_runtime(), quasimetric_cost()

Warning: run_method() between the same two graph must be called before this function.

Note: The upper bound is equivalent to the result of the pessimist edit distance cost. Methods are heuristics so the library can't compute the real perfect result because it's NP-Hard problem.

gedlibpy.**graph_ids**()
Searchs the first and last IDs of the loaded graphs in the environment.

Returns The pair of the first and the last graphs Ids

Return type tuple(size_t, size_t)

Note: Prefer this function if you have huges structures with lots of graphs.

gedlibpy.**hungarian_LSAP**()
Applies the hungarian algorithm (LSAP) on a matrix Cost.

Parameters **matrix_cost** (*vector[vector[size_t]]*) – The matrix Cost

Returns The values of rho, varrho, u and v, in this order

Return type vector[vector[size_t]]

See also:

hungarian_LSAPE()

gedlibpy.**hungarian_LSAPE**()
Applies the hungarian algorithm (LSAPE) on a matrix Cost.

Parameters **matrix_cost** (*vector[vector[double]]*) – The matrix Cost

Returns The values of rho, varrho, u and v, in this order

Return type vector[vector[double]]

See also:

hungarian_LSAP()

gedlibpy.**init**()
Initializes the environment with the chosen edit cost function and graphs.

Parameters **init_option** (*string*) – The name of the init option, "EAGER_WITHOUT_SHUFFLED_COPIES" by default

See also:

list_of_init_options

Warning: No modification were allowed after initialization. Try to make sure your choices is correct. You can though clear or add a graph, but recall init() after that.

Note: Try to make sure the option exists with list_of_init_options or choose no options, raise an error otherwise.

gedlibpy.**init_method**()
Inits the environment with the set method.

See also:

set_method(), list_of_method_options

Note: Call this function after set the method. You can't launch computation or change the method after that.

gedlibpy.**is_initialized**()
Checks and returns if the computation environment is initialized or not.

Returns True if it's initialized, False otherwise

Return type bool

Note: This function exists for internals verifications but you can use it for your code.

gedlibpy.**load_GXL_graphs**()
Loads some GXL graphes on the environment which is in a same folder, and present in the XMLfile.

Parameters

- **path_folder** (*string*) – The folder's path which contains GXL graphs
- **path_XML** (*string*) – The XML's path which indicates which graphes you want to load

Note: You can call this function multiple times if you want, but not after an init call.

gedlibpy.**quasimetric_cost**()
Checks and returns if the edit costs are quasimetric.

Parameters

- **g** (*size_t*) – The Id of the first compared graph
- **h** (*size_t*) – The Id of the second compared graph

Returns True if it's verified, False otherwise

Return type bool

See also:

run_method(), get_upper_bound(), get_lower_bound(), get_forward_map(), get_backward_map(), get_runtime()

Warning: run_method() between the same two graph must be called before this function.

gedlibpy.**restart_env**()
Restarts the environment variable. All data related to it will be delete.

Warning: This function deletes all graphs, computations and more so make sure you don't need anymore your environment.

Note: You can now delete and add somes graphs after initialization so you can avoid this function.

gedlibpy.**run_method**()

Computes the edit distance between two graphs g and h, with the edit cost function and method computation selected.

Parameters

- **g** (*size_t*) – The Id of the first graph to compare
- **h** (*size_t*) – The Id of the second graph to compare

See also:

get_upper_bound(), get_lower_bound(), get_forward_map(), get_backward_map(), get_runtime(), quasimetric_cost()

Note: This function only compute the distance between two graphs, without returning a result. Use the differents function to see the result between the two graphs.

gedlibpy.**set_edit_cost**()

Sets an edit cost function to the environment, if its exists.

Parameters

- **edit_cost** (*string*) – The name of the edit cost function
- **edi_cost_constant** – The parameters you will add to the editCost, empty by default

See also:

list_of_edit_cost_options

Note: Try to make sure the edit cost function exists with list_of_edit_cost_options, raise an error otherwise.

gedlibpy.**set_method**()

Sets a computation method to the environment, if its exists.

Parameters

- **method** (*string*) – The name of the computation method
- **options** (*string*) – The options of the method (like bash options), an empty string by default

See also:

init_method(), list_of_method_options

Note: Try to make sure the edit cost function exists with list_of_method_options, raise an error otherwise. Call init_method() after your set.

gedlibpy.**set_personal_edit_cost**()

Sets an personal edit cost function to the environment.

Parameters **edit_cost_constant** (*list*) – The parameters you will add to the editCost, empty by default

See also:

list_of_edit_cost_options, set_edit_cost()

CHAPTER

FIVE

AUTHORS

- David Blumenthal for C++ module
- Natacha Lambert for Python module

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

g