

ALCPack References

Release 1.0.1

A python package to create an edge between any two given nodes in a simple, connected, and undirected graph via a sequence of local complementation operations.

Amit Kumar Pal

July, 2019

Contents

1	Background	2
1.1	Graphs	2
1.2	Simple paths	2
1.3	Local complementation	2
2	ALCPack: Installation	3
3	ALCPack: Description	4
4	Examples	5
4.1	Example 1	5
4.2	Example 2	6

1 Background

We first define and discuss the terminology corresponding to different aspects of graph which will help putting the purpose of the developed package into perspective.

1.1 Graphs

A graph $G(V, E)$ is a set of *nodes*, denoted by V , which are connected to each other by a set of *edges*, represented by E . Let us assume the cardinality of V to be N , and the nodes are labelled as $1, 2, \dots, N-1, N$. An edge connecting the nodes i and j is denoted by (i, j) ($i \neq j, i, j \in V$). We consider *simple*, *undirected*, and *connected* graphs only. A simple graph is one where self-connection, i.e., a node connecting to itself by an edge, and the existence of multiple edges connecting a pair of nodes are prohibited. A simple graph is connected if for each pair of sites $\{i, j\} \in \mathcal{V}$, there exists at least a path P_{ij} between i and j , constituted of a set of links $\{(k, l)\} \in E$ with $k, l \in V$. Also, in an undirected graph, the links (i, j) and (j, i) are equivalent. The neighbourhood of a node i in G is denoted by $\mathcal{N}_i \subset V$, which is the set of nodes $\{j\}$ that are directly connected to i by links, i.e., $(i, j) \in E \forall j \in \mathcal{N}_i$.

1.2 Simple paths

A *simple* path P_{ab} connecting the *source* node a to the *target* node b is a sequence of nodes in the graph, given by

$$P_{ab} = [a \equiv 1, 2, \dots, n \equiv b], \quad (1)$$

with the link $(i, i+1) \in E, i = 1, \dots, n-1$, and $i \neq j \forall i, j \in P_{ab}$. The length of the path P_{ab} is the number of links $l = n-1$ traversed while going from a to b along the path (see Fig. 1(a) for an example). We denote a simple path between a and b having length l as $P_{ab}^{(l)}$. There can be more than one simple paths of different or same lengths between two nodes a and b in a graph. The *shortest path* is the simple path between a and b having the minimal length $l = l_{min}$, where the minimization is taken over all possible simple paths between a and b . There can be more than one shortest paths between a specific pair of nodes (see Fig. 1(b)).

Distance. The *distance* between two nodes i and j is specific to a chosen path P_{ab} of the form in Eq. (1), and is measured by the number of links between i and j along P_{ab} . It can be represented by $d_{(i,j)} = j - i, j > i$ can be assumed without any loss of generality. It is easy to see that for any path P_{ab} , $d_{(a,b)} = l = n-1$.

Classification. The set of all possible simple paths between any two given nodes a and b in a graph G can be divided into two categories, \mathcal{C}_1 and \mathcal{C}_2 . A simple path P_{ab} belongs to the category \mathcal{C}_1 iff for any two qubits i and j on P_{ab} , $(i, j) \notin E \forall i, j \in P_{ab}$, with $i+1 < j \leq n$. Any simple path for which a link $(i, j) \in E$, with $i, j \in P_{ab}$ and $i+1 < j \leq n$, belongs to \mathcal{C}_2 . By definition, $\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset$, while $\mathcal{C}_1 \cup \mathcal{C}_2$ constitutes the complete set of simple paths between a and b . All shortest paths between two given nodes a and b in a simple, connected, and undirected graph belongs to \mathcal{C}_1 . However, note that not all paths in \mathcal{C}_1 are shortest paths. See Fig. 1(c) for examples.

1.3 Local complementation

The local complementation (LC) operation with respect to a node i , denoted by $\tau_i(\cdot)$, on a graph G deletes all the links $\{(j, k)\}$ if $j, k \in \mathcal{N}_i$, and $(j, k) \in E$, and creates all the links $\{(j, k)\}$ if $j, k \in \mathcal{N}_i$, and $(j, k) \notin E$. A

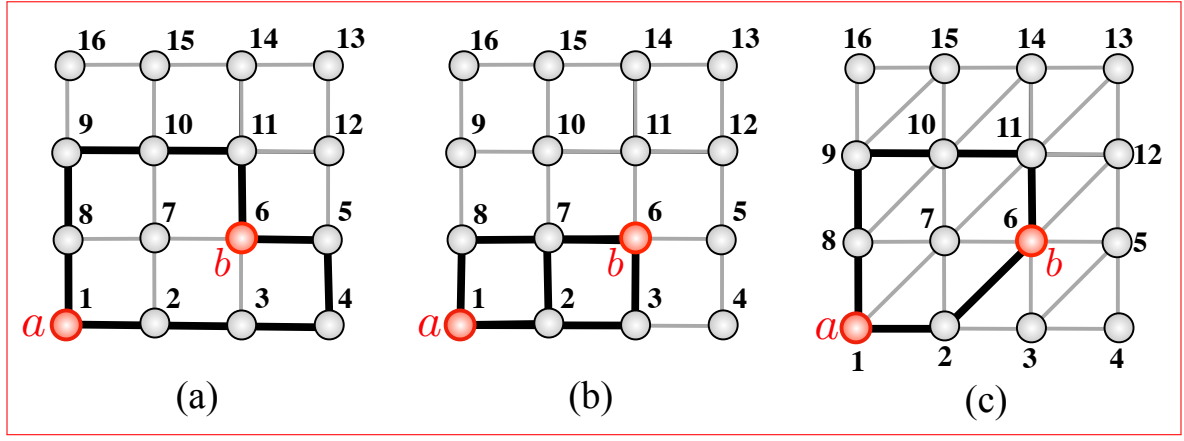


Figure 1: (Color online). (a) A square graph G_S of size $N = 16$, composed of a set of nodes $V = \{1, 2, \dots, 16\}$, as an example of a simple, connected, and undirected graph. We focus on the node-pair $\{a, b\} \equiv \{1, 6\}$, where $(a, b) \notin E$. The neighborhoods of the nodes a and b are given by $\mathcal{N}_a = \{2, 8\}$ and $\mathcal{N}_b = \{3, 5, 7, 11\}$. Examples of simple paths connecting the node-pair $\{1, 6\}$ are $P_{ab}^{(l=5)} = [a \equiv 1, 8, 9, 10, 11, 6 \equiv b]$ and $P_{ab}^{(l=5)} = [a \equiv 1, 2, 3, 4, 5, 6 \equiv b]$, where both paths have length $l = 5$. (b) The set of shortest paths between the nodes $\{1, 6\}$ in G_S have cardinality 3, and is given by $\{P_{ab}^{(l=3)}\} = \{[1, 2, 3, 6], [1, 8, 7, 6], [1, 2, 7, 6]\}$. (c) In this graph G , the simple path $P_{ab}^{l=5} \equiv [a \equiv 1, 8, 9, 10, 11, 6 \equiv b]$ belongs to \mathcal{C}_1 due to the existence of the link $(8, 10)$, while the path $P_{ab}^{l=2} \equiv [a \equiv 1, 2, 6 \equiv b]$ is a category 1 path.

sequence of LC operations on n nodes denoted by $\mathbf{m} \equiv \{1, 2, \dots, n\}$ of a graph results in a graph transformation, and the corresponding operation is denoted by

$$\tau_{\mathbf{m}} = \tau_{n/n-1/\dots/1}(\cdot) = \tau_n \circ \tau_{n-1} \circ \dots \circ \tau_1(\cdot), \quad (2)$$

where the LC operation is performed on the node 1 first, and then according to the sequence $\{1, 2, \dots, n\}$. See Fig. 2 for examples.

For a given simple, connected, and undirected graph $G(V, E)$, and two specific qubits a and b such that the link $(a, b) \notin E$, using the LC operations w.r.t. the members in a set of selected nodes in G , a link between two chosen qubits a and b can be created. These sets of nodes have to be chosen as the nodes on simple paths that connect a and b , and belong to \mathcal{C}_1 , as stated in the following theorem (see [1] for a proof).

Theorem. For a simple path $P_{ab} \in \mathcal{C}_1$ of the form given in Eq. (1) between a pair of nodes $a \equiv 1$ and $b \equiv n$ in a graph G , a sequence of local complementation operations on the nodes $\{2, \dots, n-1\}$ always creates a link between the nodes $a \equiv 1$ and $b \equiv n$, when the local complementation operations are performed on the nodes in the same order as they are in the sequence P_{ab} .

2 ALCPack: Installation

The Adaptive Local Complementation Package, or ALCPack, works for [Python 3.0](#) or more recent releases, and is created using [NetworkX](#). In order to build the ALCPack for installation via pip, one requires

- **Setuptools:** Python package development process library.
- **Wheel:** For creating .whl file directly installable via pip
- **NetworkX:** A python package for analysing complex networks

Before building the ALCPack,

1. Check installation of **python 3.0** (or higher), and **pip**.

```
$ python -V
$ python -m pip --version
```

2. Install **Setuptools** and **Wheel**.

```
$ sudo python -m pip install --upgrade pip setuptools wheel
```

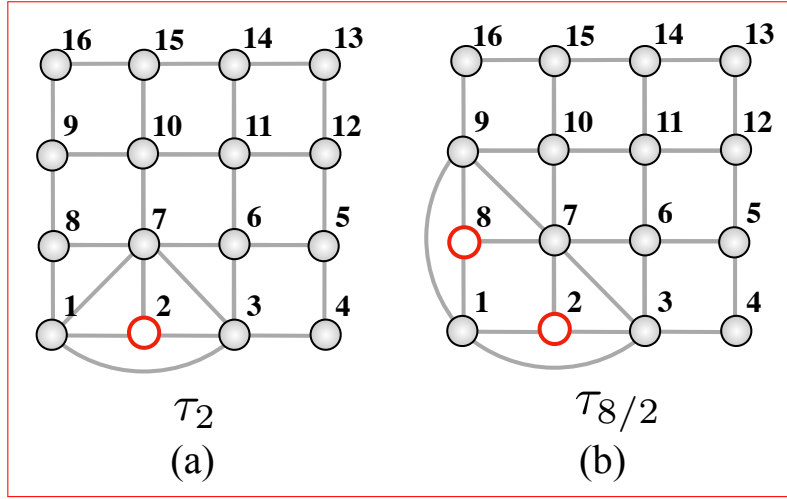


Figure 2: (Color online). Local complementation operation τ_2 with respect to node ‘2’ (a) on the square graph G_S , and then τ_8 with respect to node ‘8’ on the graph $\tau_2(G_S)$ (b). The operation τ_2 on G_S creates the links (1, 7), (3, 7), and (1, 3), and the operation τ_8 on $\tau_2(G_S)$ creates the links (1, 9) and (7, 9) while deleting the link (1, 7).

3. Install the current release of **NetworkX**

```
$ sudo python -m pip install networkx
```

In order to install ALCPack in a personal computer,

1. Download ALCPack from the hyperlink <https://github.com/amitkpal/alcpack/archive/master.zip> and unzip the bundle in a folder named ‘alcpack-master’.
2. Open a terminal in folder ‘alcpack-master’.
3. Run the following commands:

```
$ python setup.py bdist_wheel
$ python -m pip install dist/alcpack-1.1-py3-none-any.whl
```

This should install ALCPack in a personal computer. Alternatively, if one chooses not to build the package, it can simply be installed via pip.

```
$ python -m pip install alcpack
```

3 ALCPack: Description

ALCPack provides a sets of functions based on the discussion in Sec. 1 (see also [1]). The package provides three functions:

1. `local_complementation(G, target)`

Performs the local complementation operation on a chosen node in a simple, connected, undirected graph

Parameters: `G` NetworkX graph

`target` node

chosen node w.r.t. which the local complementation operation is performed

Returns: `H` NetworkX graph

graph obtained from G via local complementation operation on the node ‘target’

2. `path_category(G, path)`

Determines the category of a chosen simple path, and distils a category 1 path from the chosen path

Parameters: `G` NetworkX graph

`path` list of nodes

represents the simple path between a source node and a target node

source: first node on the path, target: last node on the path

Returns: `n` integer

value of `n` is 1 (2) for a category 1 (2) path

`newpath` list of nodes

distilled category 1 path (originally chosen path) if `n = 2` (1)

3. `alc_function(G, path)`

Adaptive local complementation function: Performs local complementation operation on all nodes in a chosen path, if the path is of category 1, or on all nodes on a distilled path of category 1 obtained from the chosen path of category 2

Parameters: `G` NetworkX graph

`path` list of nodes

represents the simple path between a source node and a target node

Returns: `H` NetworkX graph

graph in which a link between the source node and the target node exists

4 Examples

We now consider two specific examples.

4.1 Example 1

Here we consider the graph shown in Fig. 3(a). The ALCPack distils the \mathcal{C}_1 path [1, 9, 5] from the \mathcal{C}_2 path [1, 2, 9, 4, 5], and performs LC operation on node 9 to create a link between the nodes 1 and 5 (see Fig. 3(b)).

```
>>> # import networkx
>>> import networkx as nx
>>> # import adaptive local complementation package
>>> import alcpack as alc
>>> # lists of nodes and edges
>>> nodelist=list([1,2,3,4,5,6,7,8,9])
>>> edgelist=list([(1,2),(1,8),(1,9),(2,3),(2,9),(3,4),(3,9),(4,5),(4,9),(5,6),(5,9),(6,7),(6,9),(7,8),(7,9),(8,9)])
>>> # build the graph
>>> G=nx.Graph()
>>> G.add_nodes_from(nodelist)
>>> G.add_edges_from(edgelist)
>>> # chosen path belonging to  $\mathcal{C}_1$ 
>>> path=list([1,2,9,4,5])
>>> # determine category of the path and distil a category 1 path from a category 2 path
>>> pc=alc.path_category(G,path)
>>> print(pc)
>>> (2,[1,9,5])
>>> # category of the path
>>> print(pc[0])
>>> 2
>>> # distilled path of category 1
>>> print(pc[1])
>>> [1,9,5]
>>> # do local complementations w.r.t nodes on the new path
>>> newpath=pc[1]
>>> H=alc.alc_function(G,newpath)
>>> print(H.edges())
>>> [(1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 9), (2, 4), (2, 5), (2, 6), (2, 7), (2, 8), (2, 9), (3, 5), (3, 6), (3, 7),
(3, 8), (3, 9), (4, 6), (4, 7), (4, 8), (4, 9), (5, 7), (5, 8), (5, 9), (6, 8), (6, 9), (7, 9), (8, 9)]
```

The list of edges of the graph H is shown pictorially in Fig. 3(b).

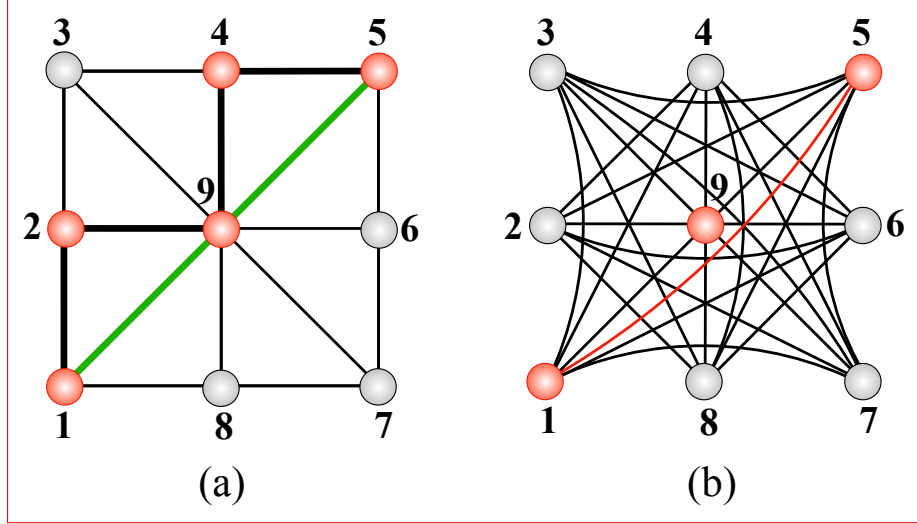


Figure 3: (Color online). (a) A graph G of size $N = 9$, composed of a set of nodes $V = \{1, 2, \dots, 9\}$, as an example of a simple, connected, and undirected graph. The path $[1, 2, 9, 4, 5]$ between the nodes 1 and 5 is a category 2 path, from which a category 1 path $[1, 9, 5]$ can be distilled via the ALCPack (see also [1]). (b) Application of the **alc_function** with graph G and the path $[1, 2, 9, 4, 5]$ as inputs creates a modified graph with the edge $(1, 5)$.

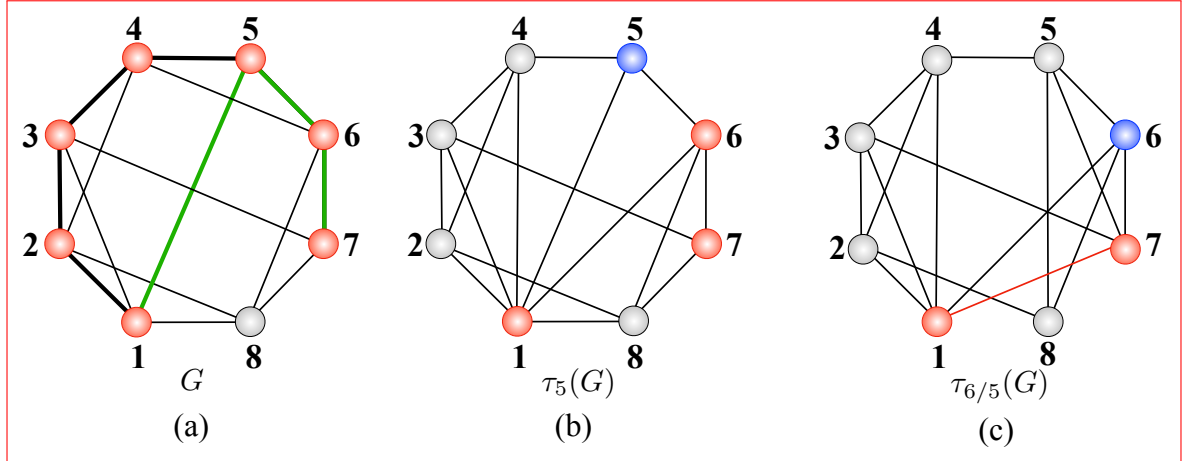


Figure 4: (Color online). (a) A graph G of size $N = 8$, composed of a set of nodes $V = \{1, 2, \dots, 8\}$, as an example of a simple, connected, and undirected graph. The path $[1, 2, 3, 4, 5, 6, 7]$ between the nodes 1 and 7 is a category 2 path, from which a category 1 path $[1, 5, 6, 7]$ can be distilled via the ALCPack (see also [1]). (b) Application of the **alc_function** with graph G and the path $[1, 2, 3, 4, 5, 6, 7]$ as inputs creates a modified graph with the edge $(1, 7)$.

4.2 Example 2

Next we consider the graph shown in Fig. 4(a). The ALCPack distills the C_1 path $[1, 5, 6, 7]$ from the C_2 path $[1, 2, 3, 4, 5, 6, 7]$, and performs LC operations on node 5 (the intermediate graph $\tau_2(G)$ is shown in Fig. 4(b)) and 6 to create a link between the nodes 1 and 5 (see graph $\tau_{6/5}(G)$ in Fig. 4(c)).

```

>>> # import networkx
>>> import networkx as nx
>>> # import adaptive local complementation package
>>> import alcpack as alc
>>> # lists of nodes and edges
>>> nodelist=list([1,2,3,4,5,6,7,8])
>>> edgelist=list([(1,2),(1,3),(1,5),(1,8),(2,3),(2,4),(2,8),(3,4),(3,7),(4,5),(4,6),(5,6),(6,7),(6,8),(7,8)])
>>> # build the graph
>>> G=nx.Graph()
>>> G.add_nodes_from(nodelist)
>>> G.add_edges_from(edgelist)
>>> # chosen path belonging to  $\mathcal{C}_1$ 
>>> path=list([1,2,3,4,5,6,7])
>>> # Adaptive local complementation on the chosen path
>>> H=alc.alc_function(G,path)
>>> print(H.edges())
>>> [(1, 2), (1, 3), (1, 4), (1, 6), (1, 7), (2, 3), (2, 4), (2, 8), (3, 4), (3, 7), (4, 5), (5, 6), (5, 7), (5, 8), (6, 7),
      (6, 8)]

```

References

- [1] D. Amaro, M. Müller, and A. K. Pal, arXiv:xxxx.xxxxx [quant-ph] (2019).