README.md

Resilionator

Resilionator is a small and portable Python application for identifying and mitigating potential weak points in networks. The application is built with Python's standard GUI toolkit Tkinter and NetworkX. The tool is intended primarily for university lecturers or students, but also for small and medium-sized companies and households.

User Documentation

Installation

Resilionator is available for Windows, macOS and Linux (64-bit operating system, x64-based processor). Download the appropriate file for your operating system and follow the steps described above to run the application.

Windows Download the file located in `windows/Resilionator.zip`. Unpack it and then double click on the file in order to run it.

maOS Download the file located in `macOS/Resilionator.zip`. Unpack it and then double click on the file in order to run it.

Linux (Ubuntu) Download the file located in `linux/Resilionator.zip`. Unpack it and then open the terminal and drag and drop the file into the terminal window then hit enter.

Usage

Resilionator offers various functionalities, which are described below:

Quickmenu (Top) The quick menu allows quick access to node/edge creation, creating a test graph or deleting a graph. There is also the option to change the layout of the graph.

Menu: Graph

- Show graph: Show the current graph.
- Create random graph: Creates a random graph for quick testing.
- Import graph: Import a graph from a `.txt`, `.gml` or `.graphml` file. Text file syntax has to be NetworkX compatile. For more details please consult file format.
- Export graph: Export the graph into a `.txt`, `.gml` or `.graphml` file.
- Save graph as image: Save the current graph as `.png or .jpg`.
- Remove graph: Remove the current graph.
- Node
  - Add node: Add a new node to the graph.
  - Remove node: Remove a node from the graph.
- Edge
  - Add edge: Add a new edge to the graph. The edge endpoints will be created automatically if the do not exist yet.
  - Remove edge: Remove an edge from the graph.

Menu: Analysis

- Connectivity
  - Node connectivity: Check if the current graph is still connected after removing a specific node. This action is peformed for all nodes in the graph.
  - Edge connectivity: Check if the current graph is still connected after removing a specific edge. This action is peformed for all edges in the graph.
- Augmentation

- K-Node augmentation: Make the current graph resilient against one or two node failures.
- K-Edge augmentation: Make the current graph resilient against one or two edge failures.

Menu: Routing

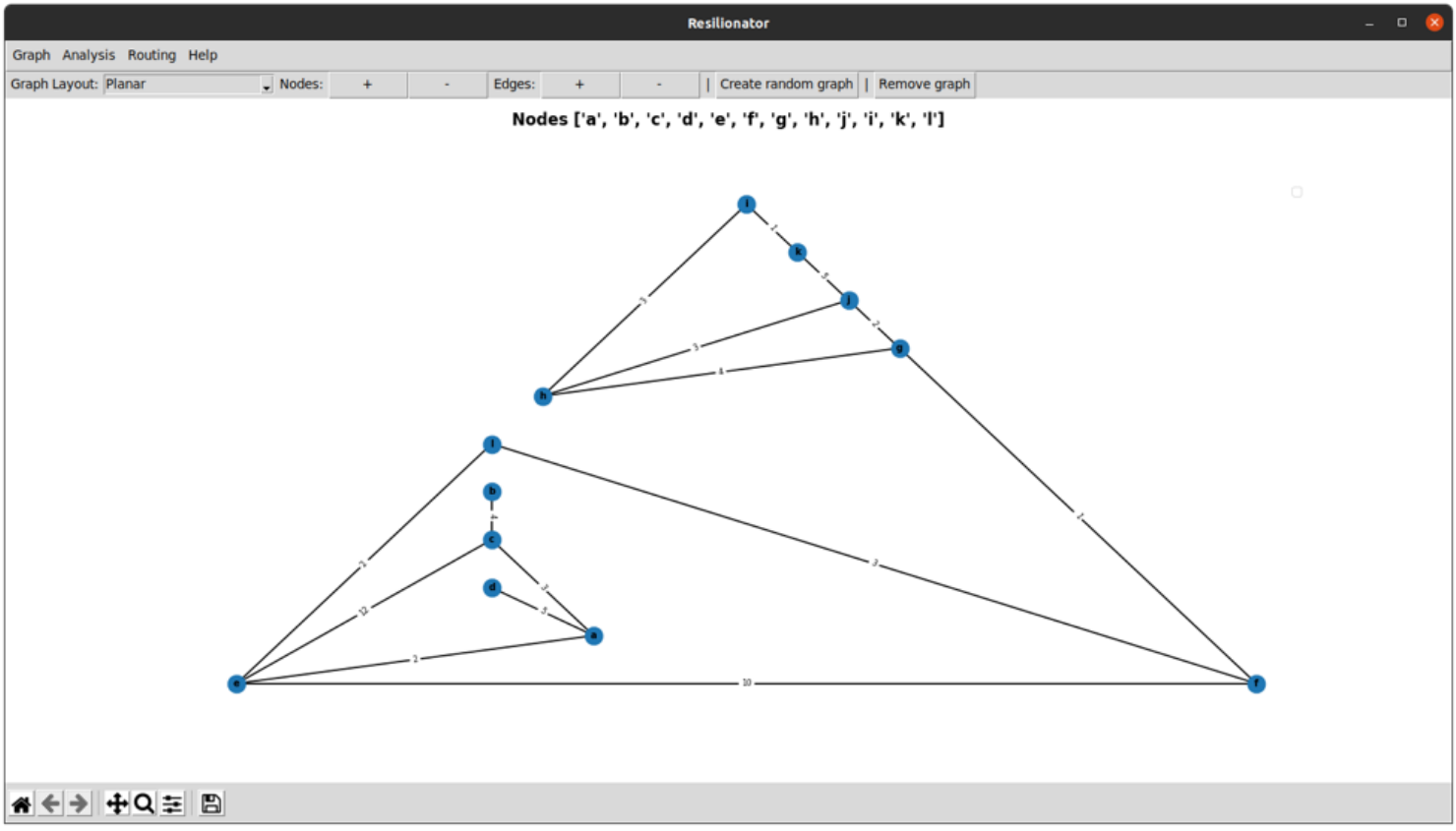
- Dijkstra - Original: Dijkstra shortest path algorithm. A `source` and `target` node need to be specified. Additionally nodes can be excluded from the path finding process.
- Dijkstra - Recalulated distances: Dijkstra shortest path algortihm, however distances are recalculated. This prevents the algorithm to get stuck if nodes become unavailable during the routing process.
- Custom Routing: The user can specify his own simple routing process by providing a priority list.
  - Priority list: A `.txt` file containing the nodes of the graph with their neighbors ordered after their priority. The file doesn't have to contain all nodes or neighbors of a node. However, note that the routing algorithm will ignore nodes or neighbors that are not explicitly listed.
  - Syntax: `<node>{<neighbor 1>,<neighbor 2>,...,<neighbor n>}`.

Priority list example

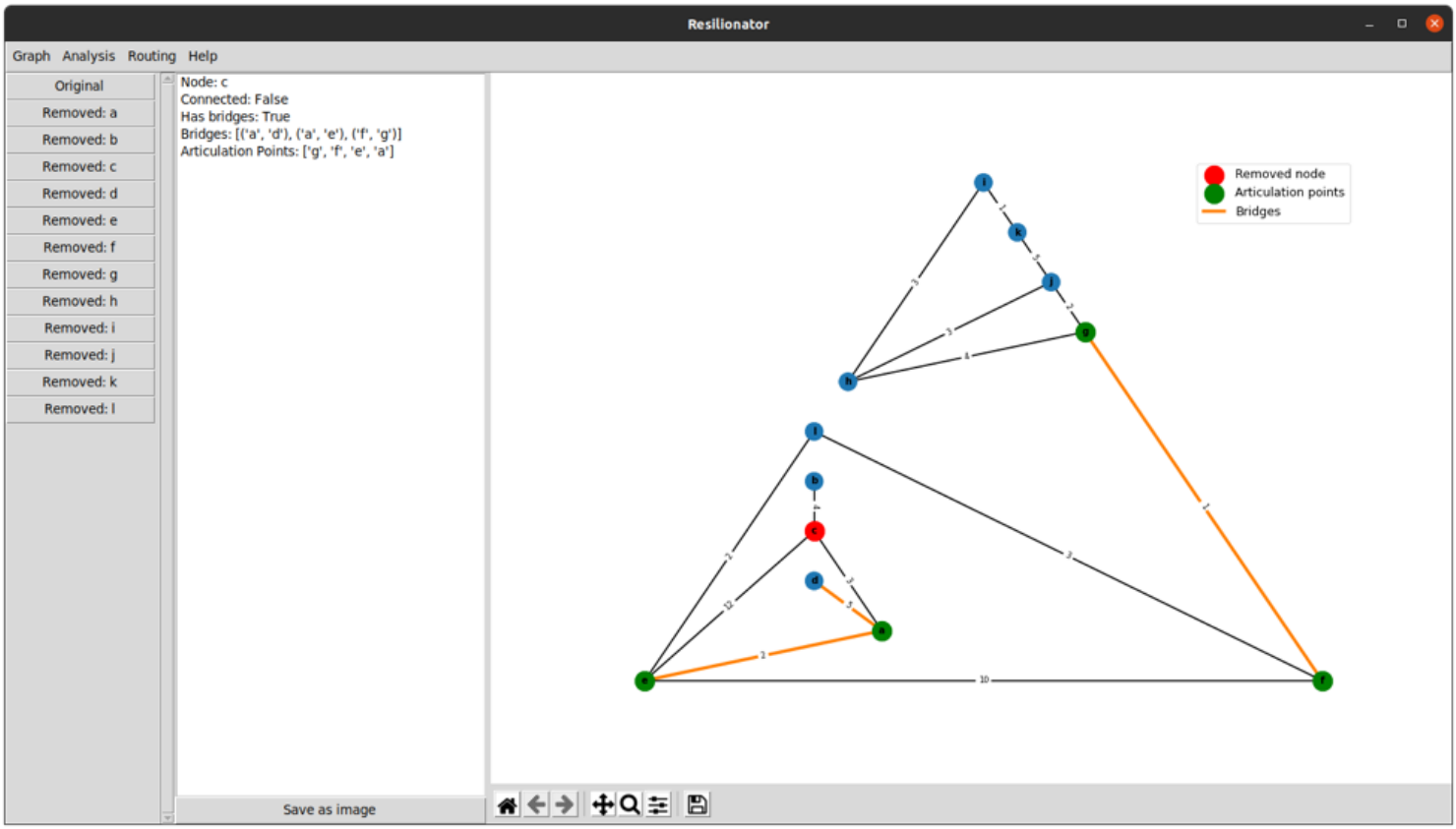
```
b{c}
c{e,a}
e{l,f,c,a}
a{c,d,e}
l{f,e}
f{g,l,e}
g{j,h}
j{h,k}
k{i}
```

Screenshots

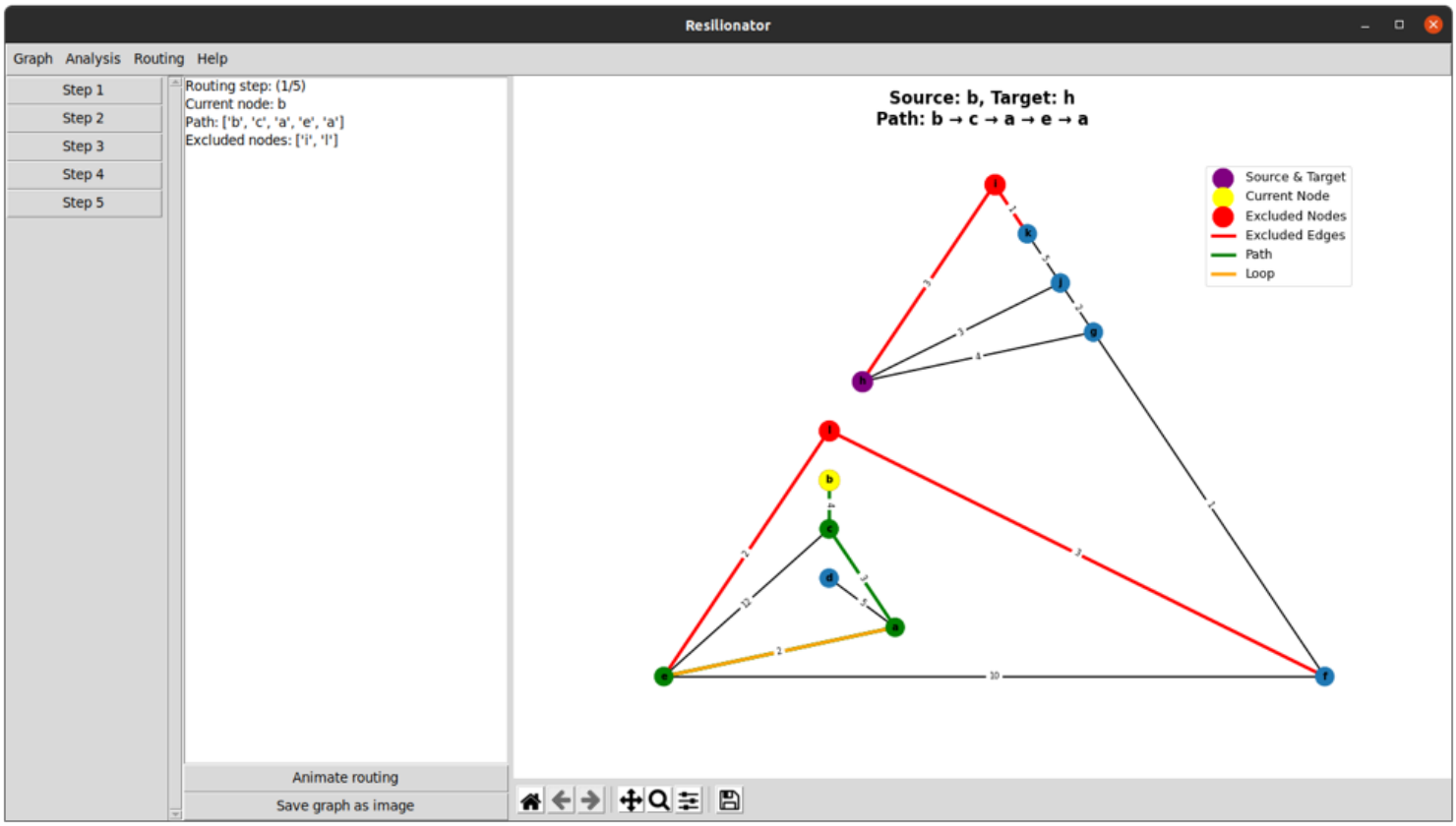
Homescreen



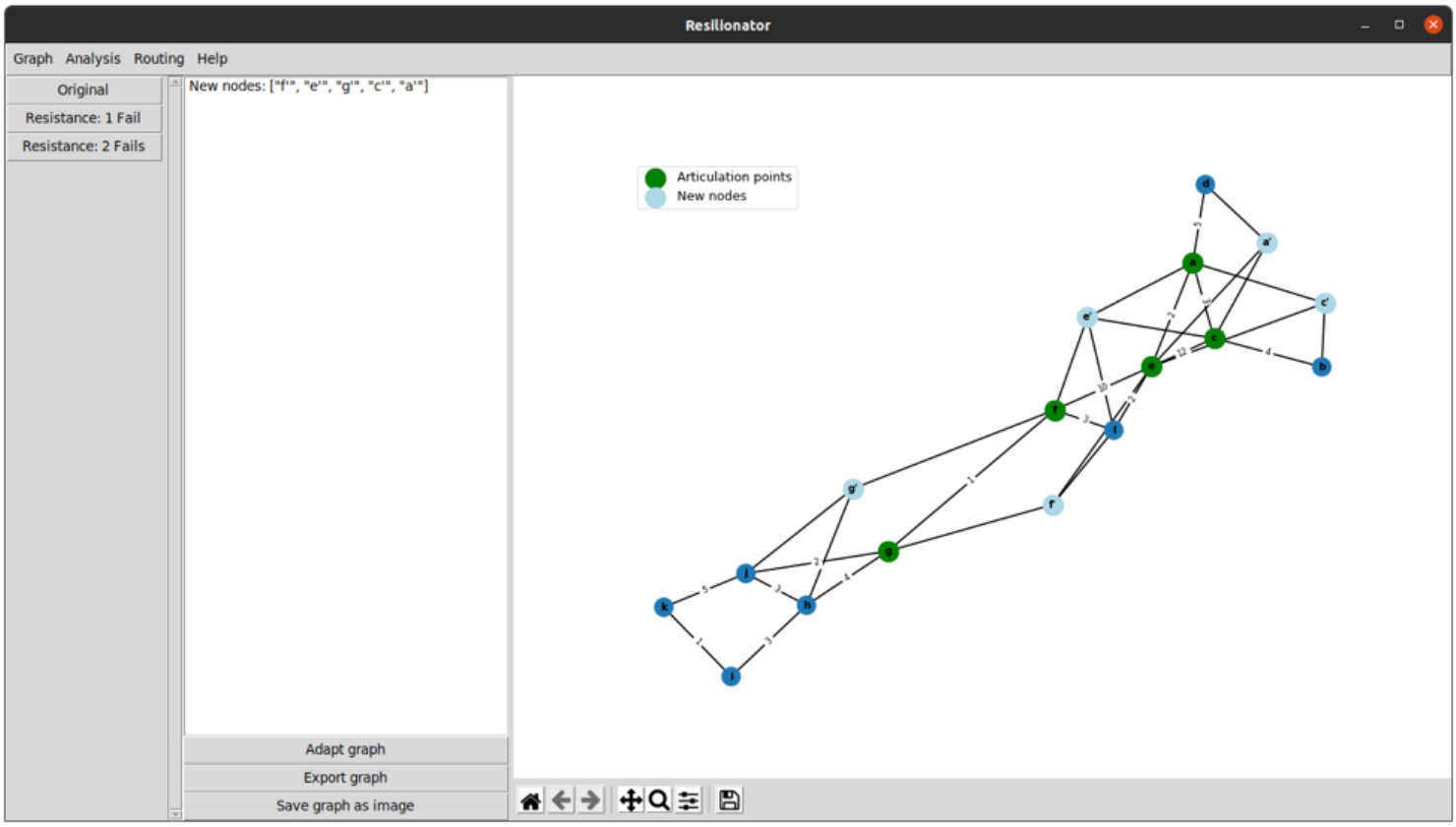
Node Connectivity



Dijkstra Routing



Failure Resistance



Resilionator Project Website

Link to the website: https://www.netidee.at/resilionator

The project Resilionator is funded by netidee.