MATH6005-Introduction to Python **Zhe GUAN**
2024-25 Student ID: 36516473
Dr. Michael Kenna-Allison Email: zg2u24@soton.ac.uk

Project Overview

A Python programme is provided to compute the shortest path around transport networks using the specified Dijkstra algorithm[1]. To increase portability, a custom class structure is designed. A Streamlit application[2] was built to provide a user-friendly interface showing computed paths with chosen start point and target point on any provided network in CSV or Excel format. Furthermore, the project conducts a comparative analysis with A* (A-star) algorithm [3] to evaluate performance between two algorithms.

All materials created by the author are available for review in the repository[4].

Algorithm Introduction

Dijkstra's algorithm is introduced by Edsger W. Dijkstra in 1956 as a greedy algorithm for finding the shortest paths from a single source vertex to all other vertices in a weighted graph[1]. It has been widely applied in many fields such as transportation networks, routing protocols, and navigation systems.

The specified Dijkstra algorithm in the project first initialises the starting node s as the only member of the visited node set $S = \{s\}$, with its distance label $Y(s) = 0$, and path $P(s) = \{-\}$. Then a cut of the network $\delta^+(S)$ is performed, which indicates all possible pairs that connect nodes inside S to nodes outside S.

For each pair $(i, j) \in \delta^+(S)$, $Y(i) + l_{ij}$ is calculated, where $Y(i)$ is the current distance to point i and l_{ij} is the length between point i inside S and point j outside S. Then minimum $Y(i) + l_{ij}$ is selected and j is added to the visited node set S with an updated distance label $Y(j)$ and path set P. The process is repeated until all nodes are added to the visited node set S.

Algorithm Implementation

A class `Dijkstra` is designed to store all functions and improve reusability and modularity.

The class `Dijkstra` contains six methods, including five protected methods `_input`, `_find_sigma_s`, `_find_closest_outpoints`, `_update_Pre_Y_S`, and `_find_P` and one public method which can be used by users `find_shortest_path`.

The protected methods are designed for internal use: `_input` processes and validates input data; `_find_sigma_s` identifies the set $\delta^+(S)$, `_find_closest_outpoints` calculates the shortest $Y(i) + l_{ij}$ and returns the pair(s) with shortest path. `_update_Pre_Y_S` updates the labels for Y and S, and `_find_P` reconstructs the full path P. A public method `find_shortest_path` as an interface for users to call these protected methods.

Algorithm Validation and Testing

Test 1: Simple network

A 5-node bi-directional network is validated for the algorithm.

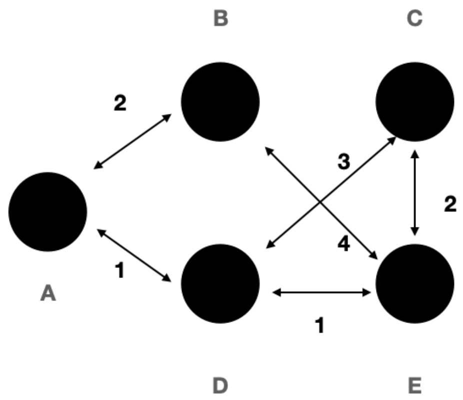

Figure 1: A simple map is tested to check coding sanity

All material is available on repository:

```
$ git clone git@github.com:phy-guanzh/Dijkstra_Astar_Algorithm.git
```

The shortest path from A to C:

```
$ python Dijkstra_main.py -f data/default.csv -s A -e C
```

Output:

```
Welcome to Dijktra Algorithm tool
Map below will be processed:
     A    B    C    D    E
A  NaN  2.0  NaN  1.0  NaN
B  2.0  NaN  NaN  NaN  4.0
C  NaN  NaN  NaN  3.0  2.0
D  1.0  NaN  3.0  NaN  1.0
E  NaN  4.0  2.0  1.0  NaN
[5 rows x 5 columns]

Start Point: A
End Point: C
Cost: 4.0 h,
Path: A->D->C or A->D->E->C
```

Two same-cost paths are printed in the output.

Test 2: European City Network

A specified simulated 19-node European network is tested for the algorithm.

Figure 2: A specified map showing networks among European cities

(1) The shortest path(s) from *London* to *Moscow*:

```
$ python Dijkstra_main.py -f data/network.csv -s London -e Moscow
```

Output:

```
**Welcome to Dijktra Algorithm tool**
Map below will be processed:
                   Arkhangelsk  Baku  Bergen  Budapest  Donetsk ...
Arkhangelsk                NaN   NaN     NaN       NaN      NaN ...
Baku                       NaN   NaN     NaN       NaN      2.0 ...
Bergen                     NaN   NaN     NaN       NaN      NaN ...
Budapest                   NaN   NaN     NaN       NaN      NaN ...
Donetsk                    NaN   2.0     NaN       NaN      NaN ...
Edinburgh                  NaN   NaN     2.0       NaN      NaN ...
...                        ...   ...     ...       ...      ...
Ulyanovsk                  4.0   6.0     NaN       NaN      2.0  ...

Start Point: London
End Point: Moscow
Time: 5.0 h,
Path: London->Hamburg->Budapest->Kyiv->Moscow
```

(2) The shortest paths from *Kyiv* to *Bergen*:

```
$ python Dijkstra_main.py -f data/network.csv -s Kyiv -e Bergen
```

Output:

```
...
Start Point: Kyiv
End Point: Bergen
Cost: 5.0 h,
Path: Kyiv->Stockholm->Bergen or Kyiv->Budapest->Hamburg->Bergen
```

(3) The shortest paths from *London* to *Murmansk*:

```
$ python Dijkstra_main.py -f data/network.csv -s London -e Murmansk
```

Output:

```
...
Start Point: London
End Point: Murmansk
Cost: 9.0 h,
Path: London->Hamburg->Stockholm->Oulu->Murmansk or \
London->Edinburgh->Bergen->Tromsø->Murmansk or \
London->Hamburg->Bergen->Tromsø->Murmansk or \
London->Hamburg->Stockholm->Oulu->Tromsø->Murmansk or \
London->Hamburg->Stockholm->Oulu->Arkhangelsk->Murmansk or \
London->Hamburg->Stockholm->Saint Petersburg->Arkhangelsk->Murmansk
```

(4) The shortest paths from *Ulyanovsk* to *Edinburgh*:

```
$ python Dijkstra_main.py -f data/network.csv -s Ulyanovsk -e Edinburgh
```

Output:

```
...
Start Point: Ulyanovsk
End Point: Edinburgh
Cost: 7.0 h,
Path: Ulyanovsk->Moscow->Kyiv->Budapest->Hamburg->Edinburgh or \
Ulyanovsk->Moscow->Kyiv->Budapest->Hamburg->London->Edinburgh
```

Test 3: Real China City Network

A real-time China city network among 279 cities in 2017[5] is tested. Data-cleaning and format-transforming steps are conducted.

The shortest path from *Beijing* to *Shanghai* in 2017:

```
$ python Dijkstra_main.py -f data/network_china/china_network.csv /
  -s Beijing -e Shanghai
```

```
**Welcome to Dijktra Algorithm tool**
Map below will be processed:
           Beijing  Tianjin  Shijiazhuang  Tangshan  Qinhuangdao  ...
origin
Ankang     13.700   13.597        11.523    14.473       15.797  ...
Anqing     12.317   11.284        10.910    12.161       13.485  ...
Anshan      6.422    5.935         9.015     4.971        3.538 ...
Anyang      5.128    5.109         2.926     6.092        7.416 ...
Baicheng   10.813   11.514        13.896    10.681        9.248
...           ...      ...           ...       ...          ...
Zhuzhou    15.489   15.004        13.494    15.880       17.205 ...
Zibo        4.432    3.184         4.283     3.952        5.276 ...
Zigong     19.251   19.317        17.075    20.456       21.780 ...
Ziyang     18.674   18.739        16.497    19.878       21.202 ...
Zunyi      20.339   19.854        18.377    20.731       22.055

[279 rows x 279 columns]

Start Point: Beijing
End Point: Shanghai
Time: 12.205 h,
Path: Beijing->Shanghai
```

Test 4: Real UK City Network

A dataset is used that describes all stations (2556 stations) in the UK with routes trains take between each station[6]. Data-cleaning and format-transforming steps are conducted. The station abbreviations have been converted to full names. The cost in the dataset is distance rather than time.

The shortest path from *Achanalt* to *Rogart*:

```
$ python Dijkstra_main.py -f data/network_uk/uk_network.csv /
  -s Achanalt -e Rogart -u km
```

Output:

```
**Welcome to Dijktra Algorithm tool**
Map below will be processed:
                        Alexandra Palace  Achanalt  Aberdare  Altnabreac ...
source
Abbey Wood                           NaN       NaN       NaN         NaN  ...
Aber                                 NaN       NaN       NaN         NaN  ...
Abercynon                            NaN       NaN       NaN         NaN  ...
```

```
Aberdare                          NaN        NaN        0.0          NaN  ...
Aberdeen                          NaN        NaN        NaN          NaN  ...
...                               ...        ...        ...          ...  ...
Yoker                             NaN        NaN        NaN          NaN  ...
York                              NaN        NaN        NaN          NaN  ...
Yorton                            NaN        NaN        NaN          NaN  ...
Ystrad Mynach                     NaN        NaN        NaN          NaN  ...
Ystrad Rhondda                    NaN        NaN        NaN          NaN  ...

[2556 rows x 2556 columns]
Start Point: Achanalt
End Point: Rogart
Cost: 117.6 km,
Path: Achanalt->Lochluichart->Garve->Dingwall->Alness->Invergordon->Fearn->
      Tain->Ardgay->Culrain->Invershin->Lairg->Rogart
```

Algorithm and Application Improvement

A* Algorithm

While Dijkstra's algorithm is efficient and widely used for shortest-path problems, but it computes the shortest path from a source node to all other nodes, even if the goal is only to find the path to a single target node. This results in unnecessary computations. A* (A star) Algorithm[3] is developed by Peter Hart, Nils Nilsson, and Bertram Raphael in 1968 and applied a Heuristic Function on Dijkstra's algorithm to guide the search, making it more efficient as shown in Table1.

Table 1: Comparisons between Dijkstra's algorithm and A* algorithm

	Dijkstra's Algorithm	A* Algorithm
Objective	The shortest path from a source node to all other nodes.	The shortest path from a source node to a specific target node.
Cost Function	$g(n)$	$f(n) = g(n) + h(n)$
Heuristic Function	Not used.	Requires a heuristic $h(n)$ to estimate the cost from the current node to the target.
Efficiency	Explores all possible paths even those irrelevant to the target.	Explores fewer nodes by using the heuristic to guide toward the target.
Optimality	Guarantees the shortest path if all edge weights are non-negative.	Guarantees the shortest path if the heuristic $h(n)$ is admissible (never overestimates the cost).

The main framework for A* is the same as the framework for Dijkstra Algorithm, but the cost function `_find_closest_outpoints` would be $Y(i) + l_{ij} + h(j)$ where the heuristic function $h(j)$ is to estimate the cost from current node j to the end target.

In this project, the Euclidean Distance is the heuristic function:

$$\text{H(j, t)} \ = \sqrt{(\Delta_{jt} \text{ Latitude })^2 + (\Delta_{jt}\text{Longitude})^2} \tag{1}$$

where j is the current node we visit and e is the target point.

Several excutation time tests are conducted for testing two algorithms, as shown in Table2.

```
$  python Astar_main.py -f data/network.csv -w data/city_coordinates.csv \
-s 'Moscow' -e 'London'
$  python Astar_main.py -f data/network_uk/uk_network.csv \
-w data/network_uk/stations.csv -s 'Yeovil Junction' -e 'Yeovil Pen Mill'
```

Table 2: Execution Time Comparisons between Dijkstra's algorithm and A* algorithm

	Dijkstra's Algorithm	A* Algorithm
UK network dataset Achanalt ⇌ Rogart	7.7959 s	7.2403 s
UK network dataset Yeovil Junction ⇌ Yeovil Pen Mill	0.5641 s	0.3903 s
European City Network Murmansk ⇌ Madrid	0.0406 s	0.0392 s
European City Network London⇌ Moscow	0.0354 s	0.0321 s

Execution times may vary depending on the hardware and system configurations.

Streamlit Application

To increase portability, a Streamlit application[7] with Dijkstra Algorithm is provided that supports any network input. It can be run locally or online.

Summary

A Dijkstra algorithm project is designed using a custom class structure, and the algorithm is tested with several datasets. An improved A* algorithm is developed using Euclidean Distance as the heuristic input, and a Streamlit application is provided for improving portability.

References

[1] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[2] Streamlit Team. Streamlit github repository. `https://github.com/streamlit`, n.d. Accessed: 2024-12-09.

[3] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[4] Zhe Guan. Dijkstra and a* algorithm implementation. `https://github.com/phy-guanzh/Dijkstra_Astar_Algorithm/tree/main`, 2024. Accessed: 2024-12-09.

[5] Lin Ma and Yang Tang. The distributional impacts of transportation networks in china. *Journal of International Economics*, page 103873, 2024.

[6] Bill2Bill. UK Train Stations. https://www.kaggle.com/datasets/bill2bill/uk-train-stations/data, n.d. License: CC0: Public Domain, Accessed: 2024-12-08.

[7] Zhe Guan. Dijkstra and A* algorithm dashboard. https://dashboard-dijkstra-zhe.streamlit.app/, 2024. Accessed: 2024-12-09.