graph_lab

October 2, 2022

1 Graph Lab

1.1 Header information:

- Author #1: Luigi Quattrociocchi (quattrl@mcmaster.ca)
- Author #2: Dennis Fong (fongd1@mcmaster.ca)
- Gitlab URL: http://gitlab.cas.mcmaster.ca/quattrl/l1-graph-lab
- Avenue to Learn group name: Graph 42

1.2 Week 1

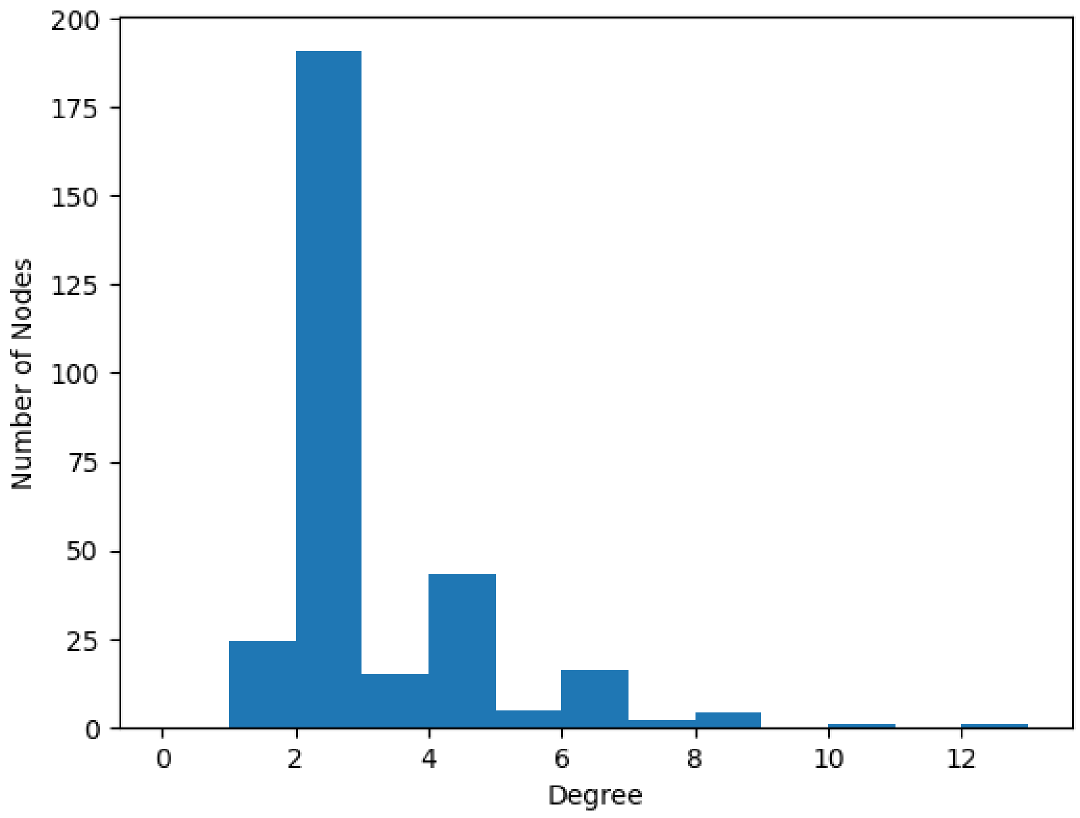
[1]:
```python
# Build a graph from the csv files
from graphlib.builders import TubemapCSVBuilder

tubemap_builder = TubemapCSVBuilder("_dataset/london.stations.csv", "_dataset/
 ↪london.connections.csv", "_dataset/london.lines.csv")
tubemap_graph = tubemap_builder.build()
```

[2]:
```python
# Compute some metrics about the graph
from graphlib.metrics import NumberOfNodesMetric, NumberOfEdgesMetric,␣
 ↪DegreeMetric

print(f"Number of nodes = {NumberOfNodesMetric(tubemap_graph)()}")
print(f"Number of edges = {NumberOfEdgesMetric(tubemap_graph)()}")

node_degrees = [DegreeMetric(tubemap_graph, x)() for x in tubemap_graph.adj]
avg_degree = sum(node_degrees) / len(node_degrees)
print(f"Average node degree = {avg_degree}")
```

```
Number of nodes = 302
Number of edges = 406
Average node degree = 2.6887417218543046
```

[3]:
```python
# Draw the distribution of node degrees
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator

N = max(node_degrees) + 1
fig, ax = plt.subplots(1, 1)
```

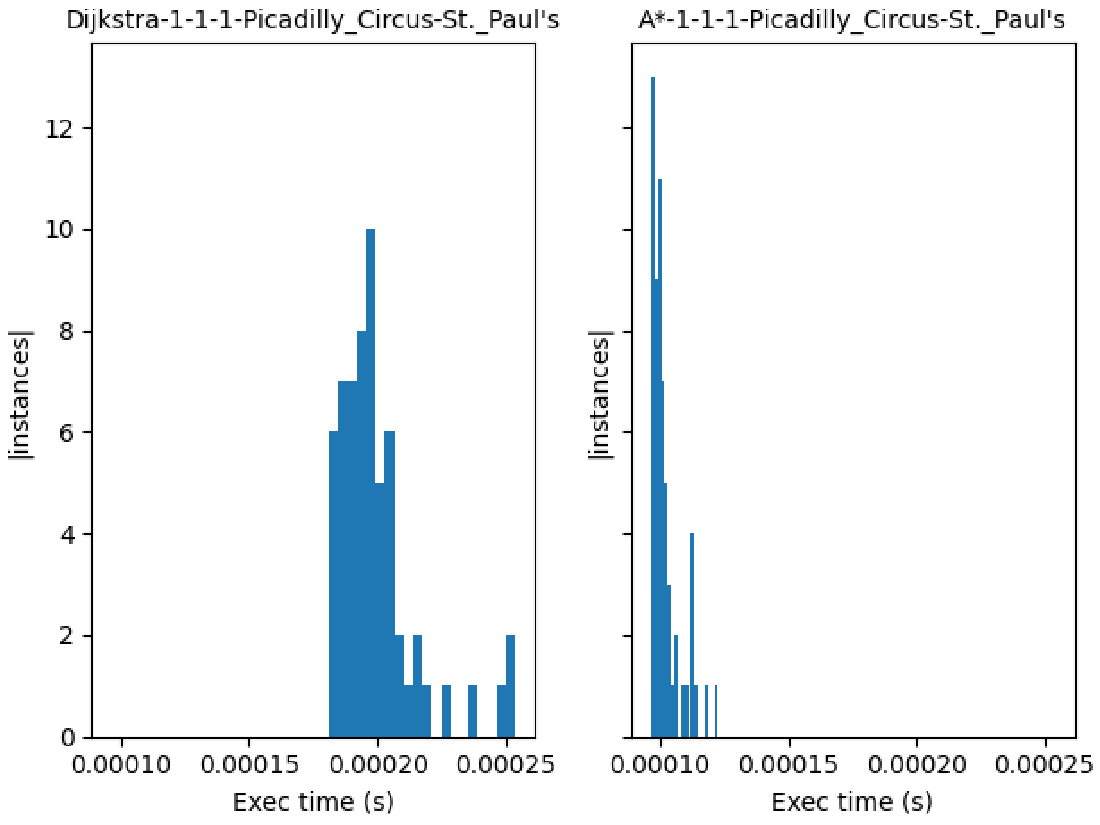
```
N, bins, patches = ax.hist(node_degrees, bins=N, range=(0, N))
ax.yaxis.set_major_locator(MaxNLocator(integer=True))
_ = plt.xlabel("Degree")
_ = plt.ylabel("Number of Nodes")
```

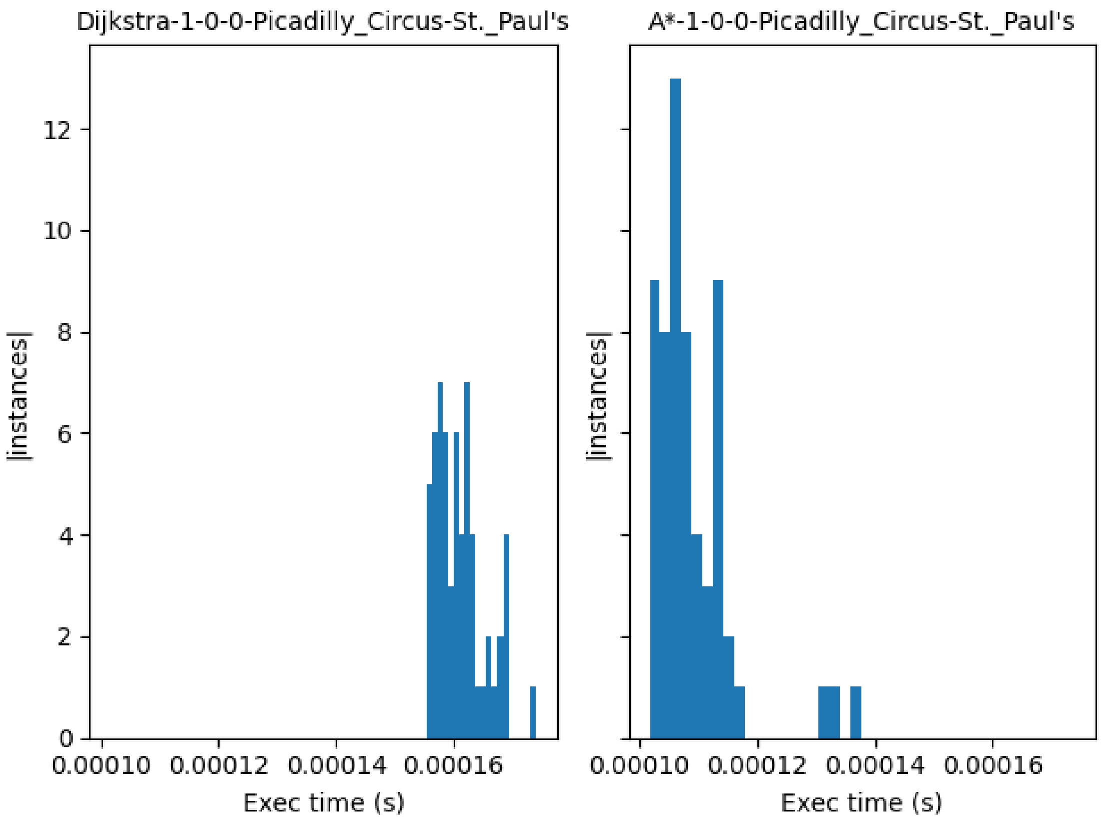


[4]:
```
# Run and visualize benchmark
BENCHMARK_FILENAME = "outputs/pyperf_measurements.json"

# Run the benchmark only if a bencmkark results file outputted by pyperf does␣
 ↪not exist
import os
if not os.path.exists(BENCHMARK_FILENAME):
    print("Running benchmark. This may take some time, go get a coffee or␣
 ↪something...")
    !pipenv run python benchmark.py -o $BENCHMARK_FILENAME

# Print KPIs
from benchmark import london_tubemap_cases
for case in london_tubemap_cases():
    _p = case.func(*case.args)
    print(f'Case "{case.name}":')
    print(f"KPI 1: Edges evaluated  = {case.metric._edges_counter}")
    print(f"KPI 2: Nodes evaluated  = {case.metric._nodes_counter}")
    print(f"KPI 3: Edge relaxations = {case.metric._relaxation_counter}")
    print("=" * 60)

# Draw the results
from pyperf import BenchmarkSuite
benchmarks = BenchmarkSuite.load(BENCHMARK_FILENAME).get_benchmarks()
N = len(benchmarks)
for i in range(N//2):
    fig, axs = plt.subplots(1, 2, sharex=True, sharey=True)
    for j in range(2):
        bench = benchmarks[i*2 + j]
        ax = axs[j]
        ax.set_title(bench.get_name(), fontsize=10)
        ax.set(xlabel="Exec time (s)", ylabel="|instances|")
        ax.hist(bench.get_values(), 20)
    fig.tight_layout()
```

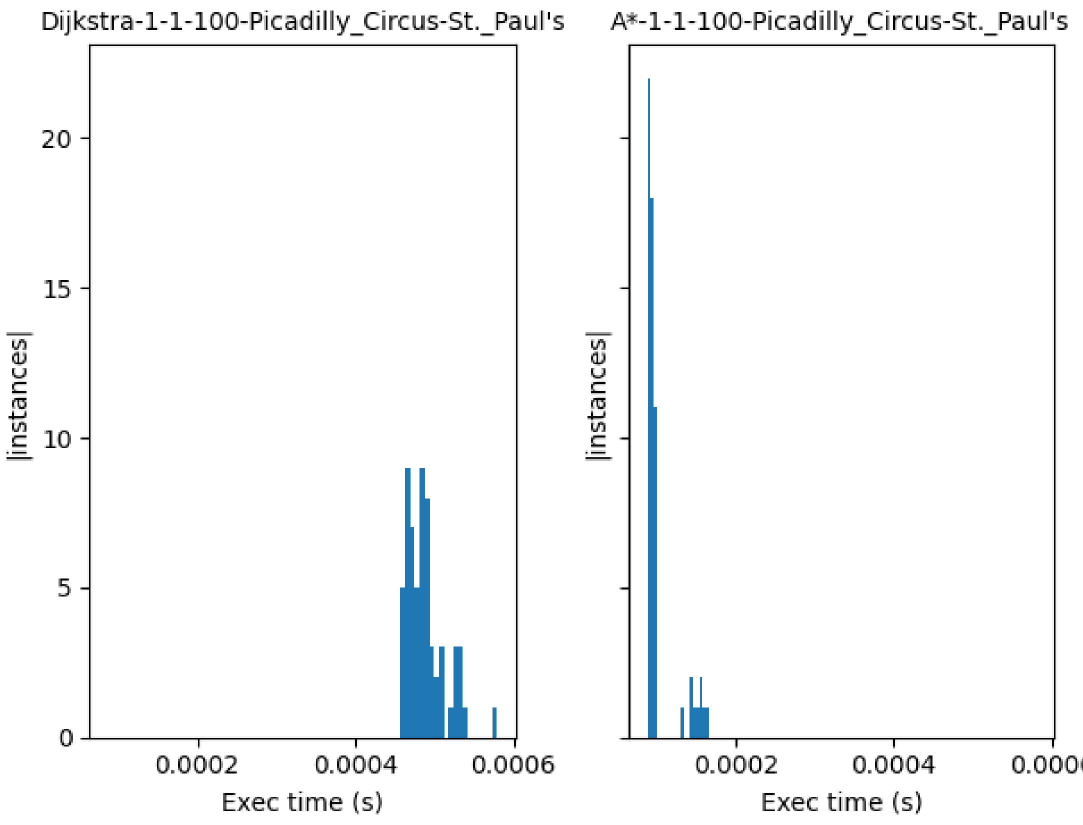
```
Case "Dijkstra-1-1-1-Picadilly_Circus-St._Paul's":
KPI 1: Edges evaluated  = 168
KPI 2: Nodes evaluated  = 40
KPI 3: Edge relaxations = 59
============================================================
Case "A*-1-1-1-Picadilly_Circus-St._Paul's":
KPI 1: Edges evaluated  = 53
KPI 2: Nodes evaluated  = 12
KPI 3: Edge relaxations = 24
============================================================
Case "Dijkstra-1-0-0-Picadilly_Circus-St._Paul's":
KPI 1: Edges evaluated  = 145
KPI 2: Nodes evaluated  = 36
KPI 3: Edge relaxations = 55
============================================================
Case "A*-1-0-0-Picadilly_Circus-St._Paul's":
KPI 1: Edges evaluated  = 57
KPI 2: Nodes evaluated  = 13
KPI 3: Edge relaxations = 22
============================================================
Case "Dijkstra-1-1-100-Picadilly_Circus-St._Paul's":
KPI 1: Edges evaluated  = 294
KPI 2: Nodes evaluated  = 91
KPI 3: Edge relaxations = 134
============================================================
Case "A*-1-1-100-Picadilly_Circus-St._Paul's":
KPI 1: Edges evaluated  = 31
KPI 2: Nodes evaluated  = 7
KPI 3: Edge relaxations = 18
============================================================
Case "Dijkstra-1-1-1-Hammersmith-Stratford":
KPI 1: Edges evaluated  = 680
KPI 2: Nodes evaluated  = 219
KPI 3: Edge relaxations = 234
============================================================
Case "A*-1-1-1-Hammersmith-Stratford":
KPI 1: Edges evaluated  = 79
KPI 2: Nodes evaluated  = 20
KPI 3: Edge relaxations = 47
============================================================
Case "Dijkstra-1-0-0-Hammersmith-Stratford":
KPI 1: Edges evaluated  = 620
KPI 2: Nodes evaluated  = 206
KPI 3: Edge relaxations = 221
============================================================
Case "A*-1-0-0-Hammersmith-Stratford":
KPI 1: Edges evaluated  = 79
KPI 2: Nodes evaluated  = 20
KPI 3: Edge relaxations = 46
============================================================
Case "Dijkstra-1-1-100-Hammersmith-Stratford":
KPI 1: Edges evaluated  = 792
KPI 2: Nodes evaluated  = 234
KPI 3: Edge relaxations = 265
============================================================
Case "A*-1-1-100-Hammersmith-Stratford":
KPI 1: Edges evaluated  = 102
KPI 2: Nodes evaluated  = 22
KPI 3: Edge relaxations = 51
============================================================
Case "Dijkstra-1-1-1-Ealing_Broadway-Upminster":
KPI 1: Edges evaluated  = 887
KPI 2: Nodes evaluated  = 317
KPI 3: Edge relaxations = 316
============================================================
Case "A*-1-1-1-Ealing_Broadway-Upminster":
KPI 1: Edges evaluated  = 157
KPI 2: Nodes evaluated  = 45
KPI 3: Edge relaxations = 71
============================================================
Case "Dijkstra-1-0-0-Ealing_Broadway-Upminster":
KPI 1: Edges evaluated  = 831
KPI 2: Nodes evaluated  = 306
KPI 3: Edge relaxations = 305
============================================================
Case "A*-1-0-0-Ealing_Broadway-Upminster":
```

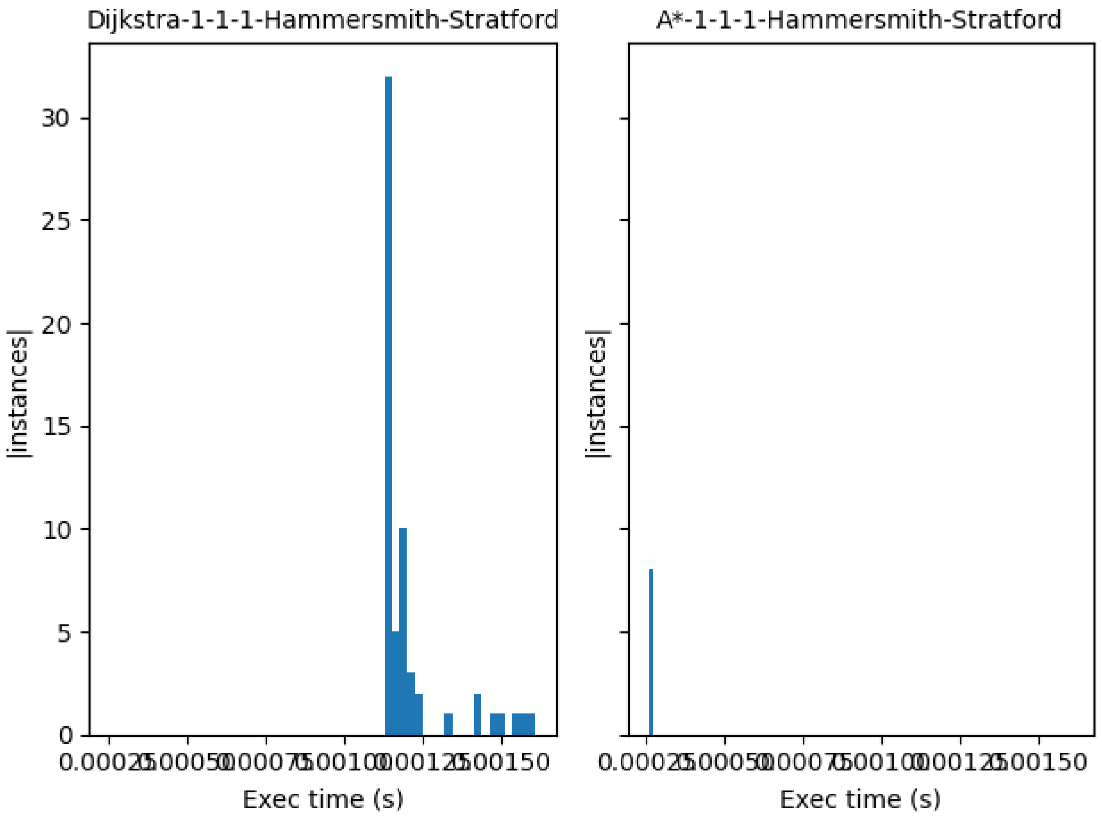
```
KPI 1: Edges evaluated  = 157
KPI 2: Nodes evaluated  = 45
KPI 3: Edge relaxations = 71
============================================================
Case "Dijkstra-1-1-100-Ealing_Broadway-Upminster":
KPI 1: Edges evaluated  = 722
KPI 2: Nodes evaluated  = 236
KPI 3: Edge relaxations = 270
============================================================
Case "A*-1-1-100-Ealing_Broadway-Upminster":
KPI 1: Edges evaluated  = 127
KPI 2: Nodes evaluated  = 46
KPI 3: Edge relaxations = 70
============================================================
```

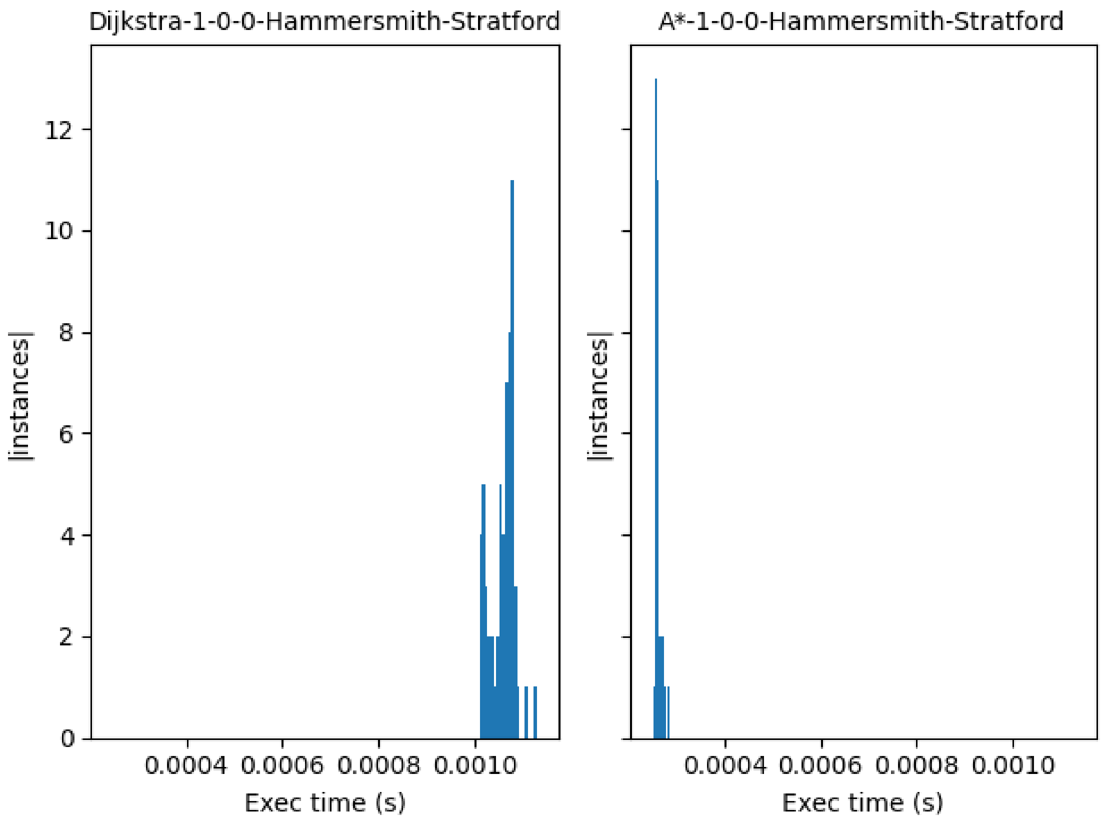


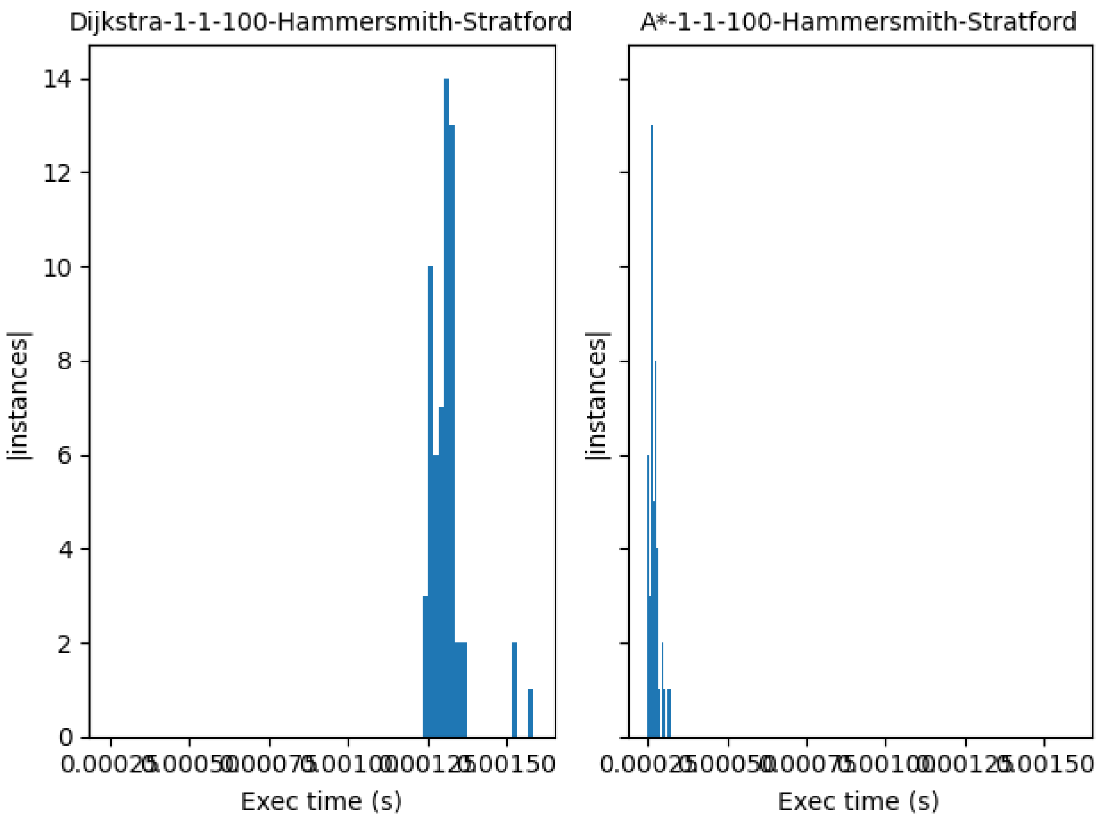


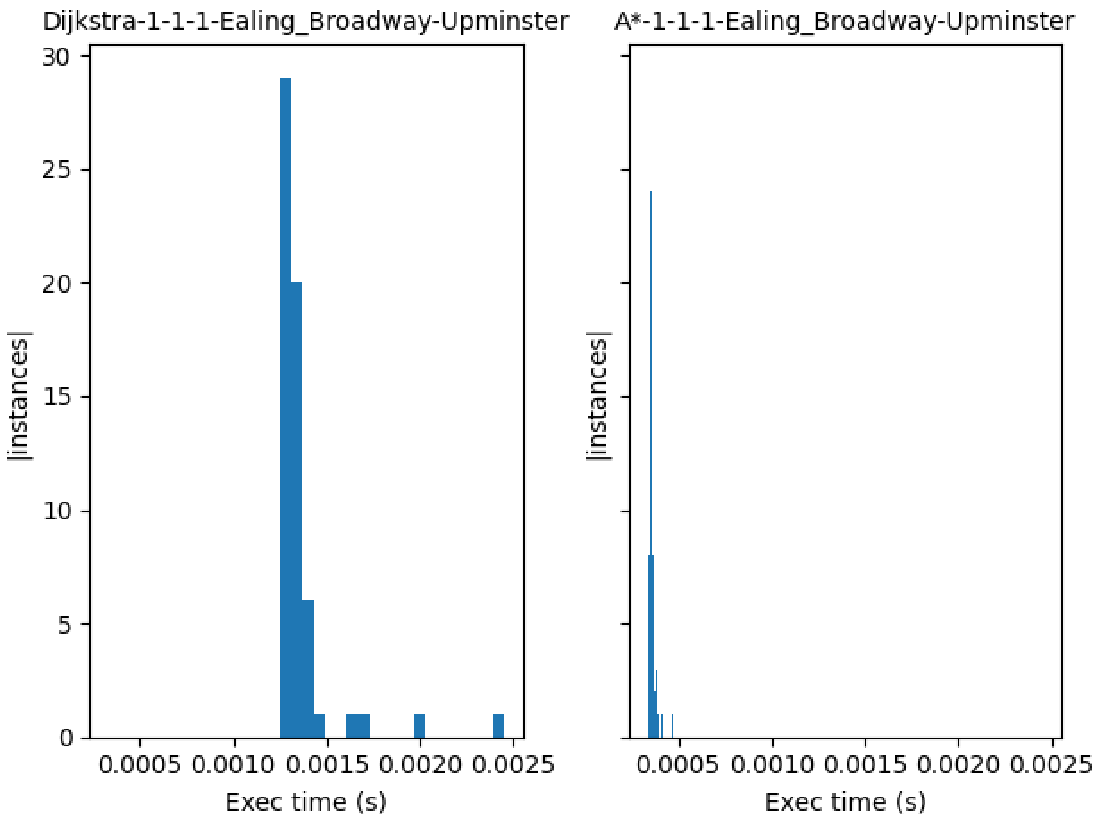
Dijkstra-1-0-0-Picadilly_Circus-St._Paul's
A*-1-0-0-Picadilly_Circus-St._Paul's
|instances|
0
2
4
6
8
10
12
0.00010
0.00012
0.00014
0.00016
Exec time (s)

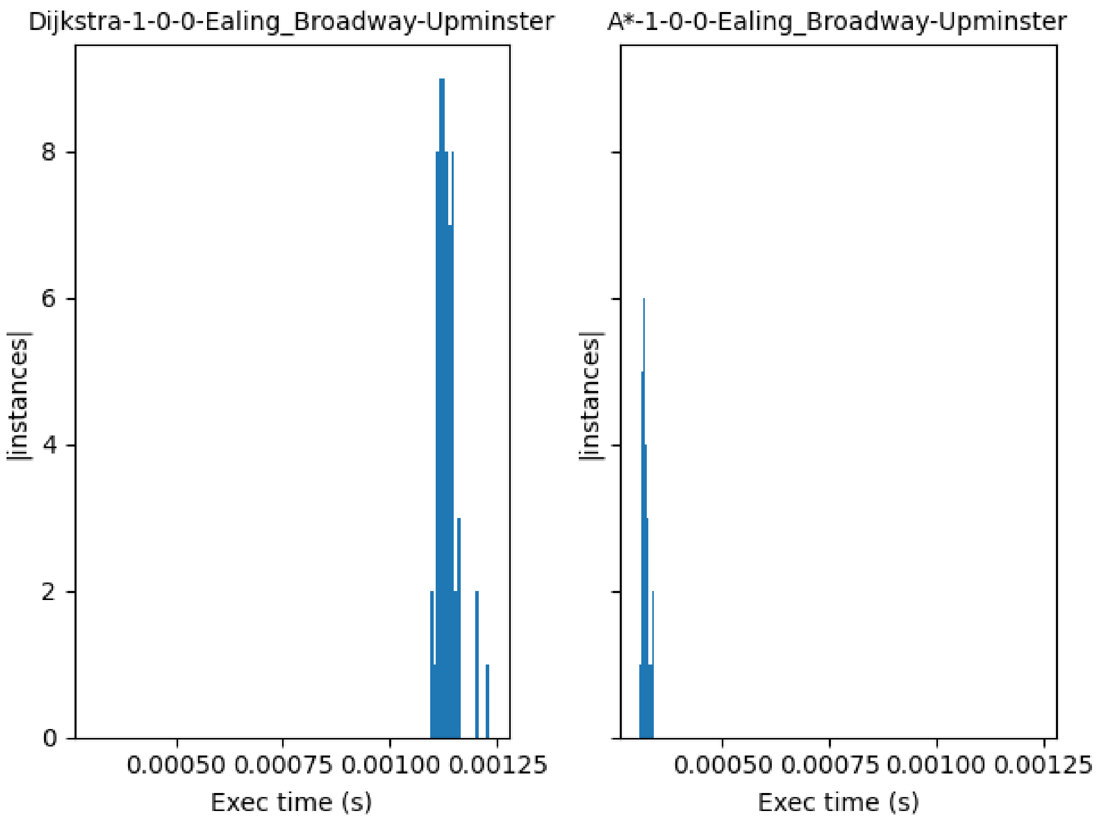

Dijkstra-1-1-100-Picadilly_Circus-St._Paul's
A*-1-1-100-Picadilly_Circus-St._Paul's
|instances|
|instances|
0
5
10
15
20
0.0002
0.0004
0.0006
0.0002
0.0004
0.0006
Exec time (s)
Exec time (s)

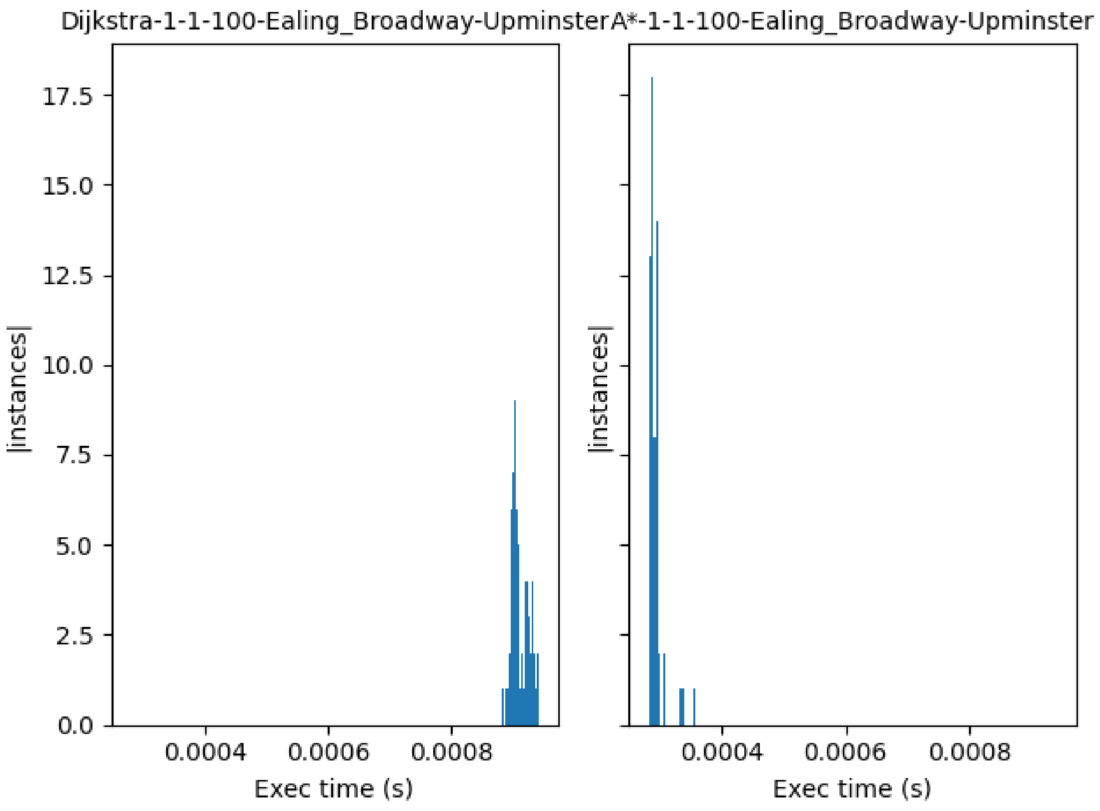

Dijkstra-1-1-1-Hammersmith-Stratford
A*-1-1-1-Hammersmith-Stratford
|instances|
|instances|
Exec time (s)
Exec time (s)
0.00025
0.00050
0.00075
0.00100
0.00125
0.00150
0
5
10
15
20
25
30


Dijkstra-1-0-0-Hammersmith-Stratford
A*-1-0-0-Hammersmith-Stratford
|instances|
|instances|
0
2
4
6
8
10
12
0.0004
0.0006
0.0008
0.0010
0.0004
0.0006
0.0008
0.0010
Exec time (s)
Exec time (s)


Dijkstra-1-1-100-Hammersmith-Stratford
A*-1-1-100-Hammersmith-Stratford
|instances|
Exec time (s)
0
2
4
6
8
10
12
14
0.00025
0.00050
0.00075
0.00100
0.00125
0.00150


Dijkstra-1-1-1-Ealing_Broadway-Upminster
A*-1-1-1-Ealing_Broadway-Upminster
|instances|
Exec time (s)
0
5
10
15
20
25
30
0.0005
0.0010
0.0015
0.0020
0.0025


Dijkstra-1-0-0-Ealing_Broadway-Upminster
A*-1-0-0-Ealing_Broadway-Upminster
|instances|
|instances|
0
2
4
6
8
0.00050 0.00075 0.00100 0.00125
0.00050 0.00075 0.00100 0.00125
Exec time (s)
Exec time (s)

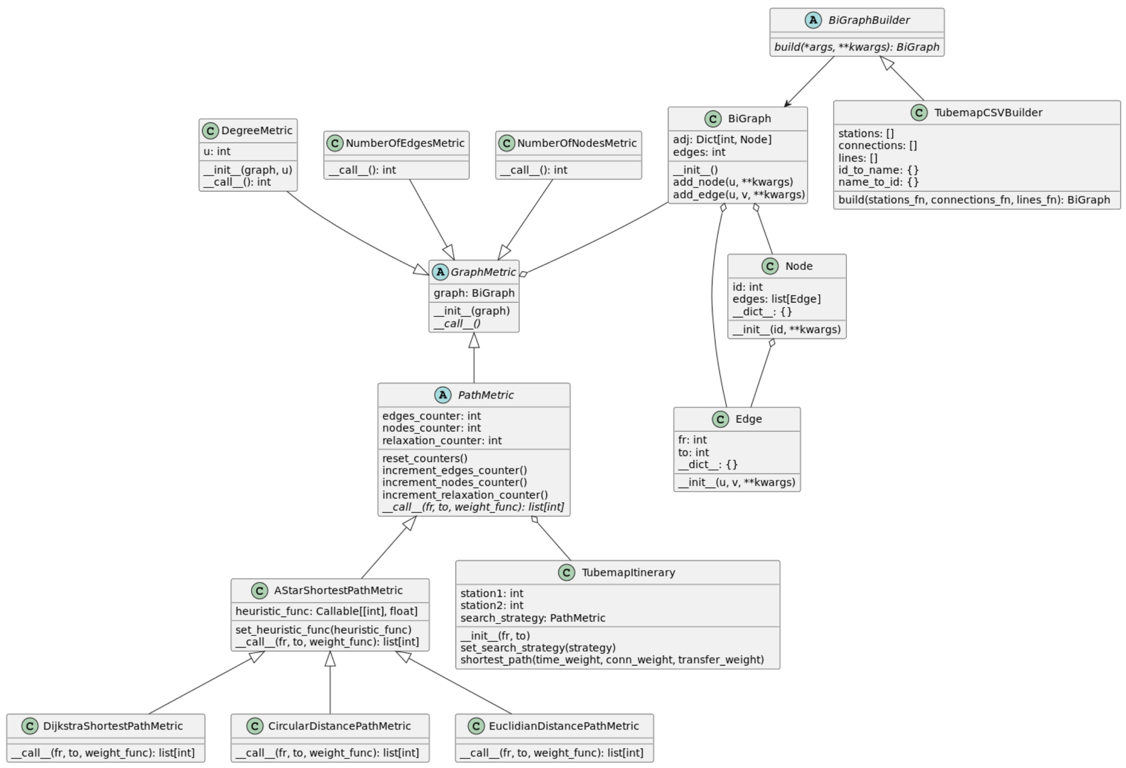


1.2.1 Benchmark results analysis

It's clear to see that the A* algorithm has lower average execution times and lower KPIs in every category for every test case (see `benchmark.py` for test cases). This correlation makes sense: if a searching algorithm inspects fewer nodes, edges, and performs less relaxations, then does less work and will finish faster.

An important note to make here is the heuristic function used by the particular version of A* benchmarked. The heuristic function used is the Great-circle distance between the latitude and longitude points of the current station and the target station. This is a (mostly) accurate way to measure the distance between two points given their positions on the earth, and for points this close together we could even have used the euclidian distance between the points (essentially projecting the points onto a plane). For the london tubemap data, neither of these particular heuristics are necessarily admissible, and we can see that A* does not always return the “shortest” path by weight. This contributes to the margin of victory of A* over Dijkstra.

1.2.2 Class diagram



Notes: - *args and **kwargs are ways to pass a variable number of arguments into a function in python

1.2.3 Justification of design choices

A couple important design decisions were made this week. Most notably: the separation of "builders" and "graphs", the structure and usage of "graph metrics", and the tubemap itinerary.

The notion of a `BiGraphBuilder` (an abstract base class) was introduced to generalize the production/building of some kind of graph. It is important that this functionality is distinct from the graph class itself to reduce the coupling between the format of the input and the graph, while maintaining cohesion.

The `BiGraphBuilder` has an abstract method called `build`, which takes some (arbitrary) arguments and returns a graph, created from those arguments. In our case, a concrete implemenation of `BiGraphBuilder` called `TubemapCSVBuilder` takes three filenames of .csv files of stations, connections, and lines, and returns a bidirectional graph (`BiGraph`) class from the contents of those files. If a new kind of representation of a graph comes in some other form (different file format, input stream, existing graph, etc), a new subclass of `BiGraphBuilder` can be created. This is similar to the factory design pattern, however having a dedicated builder class is more useful because additional information (perhaps a byproduct of the input used to build the graph) can be stored inside the class, separate from the graph.

The graph metrics, like the graph builders, are entirely distinct from the graph class itself. This way, the SRP (single responsibility principle) is satisfied. Our library has an abstract base class

GraphMetric which has an abstract __call__ method. Each “metric” (any information that a user may want to extract from their graph) is implemented as a subclass of GraphMetric. To extract a metric, a user should instantiate the subclass of the desired metric and “call” that object. The design of this mechanism is intentional and important for satifying the Open/Closed principle; When a library writer want to add a new metric (extending functionality) without modifying (and potentially breaking) existing code, they should create a new subclass of GraphMetric.

The TubemapItinerary class is yet another practical application of a design pattern in our library. The class uses the strategy pattern to compute an optimal path between two stations. The itinerary class is constructed with the IDs of the starting and target nodes, and has a method set_search_strategy which takes a callable object (more specifically, a PathMetric, which is an abstract subclass of GraphMetric). The PathMetric dependency is injected into the itinerary class when it is set, and is called upon later when the itinerary’s shortest_path method is called. The itinerary class was written in this way for SRP purposes: TubemapItinerary does not care about the graph or search strategy implementations, only that it can call back to something when it needs to compute the shortest path. PathMetric subclasses must (and do) satisfy the Liskov substitution principle for this to work. TubemapItinerary can call any PathMetric which implements the abstract __call__ operator with the correct arguments.

1.2.4 How we split the work and who did what?

The design and implementation of the library was done by both partners together in person during the first week. We agreed that it is beneficial to work together on one computer in the early stages, so that our rapid prototyping isn’t slowed down by version control. On the weekend after the first week, we split up the final tasks; Dennis drew the class diagram and Luigi wrote the benchmark, then we checked each others work.

1.3 Week 2

[5]:
```python
# Subway Patrol Planning
from graphlib.metrics import TubemapPatrolMetric

import random
stations = random.sample(list(tubemap_graph.adj.keys()), 8)
print("Randomly generated stations:", stations)
print()

time, paths = TubemapPatrolMetric(tubemap_graph, stations)()
print("Minimal round trip time:", time)
print()
for path in paths:
    for edge in path:
        print(f"Connection {edge.fr} - {edge.to} (via line {edge.line}, {edge.
 ↪time} minutes)")
    print()
```

```
Randomly generated stations: [34, 128, 4, 73, 32, 279, 5, 200]

Minimal round trip time: 160
```

```
Connection 5 - 194 (via line 10, 3 minutes)
Connection 194 - 182 (via line 10, 2 minutes)
Connection 182 - 73 (via line 10, 3 minutes)
Connection 73 - 1 (via line 4, 2 minutes)
Connection 1 - 265 (via line 10, 3 minutes)
Connection 265 - 110 (via line 10, 2 minutes)
Connection 110 - 17 (via line 4, 1 minutes)
Connection 17 - 74 (via line 10, 3 minutes)
Connection 74 - 99 (via line 10, 2 minutes)
Connection 99 - 236 (via line 3, 1 minutes)
Connection 236 - 229 (via line 3, 2 minutes)
Connection 229 - 273 (via line 3, 2 minutes)
Connection 273 - 107 (via line 11, 2 minutes)
Connection 107 - 192 (via line 11, 2 minutes)
Connection 192 - 277 (via line 11, 2 minutes)
Connection 277 - 89 (via line 9, 1 minutes)
Connection 89 - 40 (via line 9, 3 minutes)
Connection 40 - 47 (via line 9, 2 minutes)
Connection 47 - 22 (via line 9, 2 minutes)
Connection 22 - 111 (via line 9, 2 minutes)
Connection 111 - 100 (via line 9, 4 minutes)
Connection 100 - 34 (via line 9, 3 minutes)

Connection 34 - 100 (via line 9, 3 minutes)
Connection 100 - 111 (via line 9, 4 minutes)
Connection 111 - 22 (via line 9, 2 minutes)
Connection 22 - 47 (via line 9, 2 minutes)
Connection 47 - 40 (via line 9, 2 minutes)
Connection 40 - 89 (via line 9, 3 minutes)
Connection 89 - 145 (via line 9, 2 minutes)
Connection 145 - 39 (via line 10, 5 minutes)
Connection 39 - 128 (via line 10, 2 minutes)

Connection 128 - 39 (via line 10, 2 minutes)
Connection 39 - 145 (via line 10, 5 minutes)
Connection 145 - 7 (via line 9, 2 minutes)
Connection 7 - 188 (via line 9, 3 minutes)
Connection 188 - 167 (via line 9, 1 minutes)
Connection 167 - 156 (via line 3, 2 minutes)
Connection 156 - 24 (via line 2, 3 minutes)
Connection 24 - 164 (via line 2, 2 minutes)
Connection 164 - 33 (via line 4, 1 minutes)
Connection 33 - 36 (via line 4, 2 minutes)
Connection 36 - 289 (via line 4, 2 minutes)
Connection 289 - 200 (via line 4, 2 minutes)

Connection 200 - 289 (via line 4, 2 minutes)
Connection 289 - 247 (via line 7, 3 minutes)
Connection 247 - 204 (via line 13, 2 minutes)
Connection 204 - 32 (via line 13, 2 minutes)

Connection 32 - 70 (via line 13, 2 minutes)
Connection 70 - 4 (via line 13, 2 minutes)

Connection 4 - 201 (via line 13, 2 minutes)
Connection 201 - 284 (via line 13, 2 minutes)
Connection 284 - 155 (via line 13, 2 minutes)
Connection 155 - 225 (via line 13, 2 minutes)
Connection 225 - 13 (via line 13, 2 minutes)
Connection 13 - 279 (via line 12, 4 minutes)

Connection 279 - 285 (via line 7, 2 minutes)
Connection 285 - 248 (via line 3, 2 minutes)
Connection 248 - 273 (via line 3, 2 minutes)
Connection 273 - 229 (via line 3, 2 minutes)
Connection 229 - 236 (via line 3, 2 minutes)
Connection 236 - 99 (via line 3, 1 minutes)
Connection 99 - 74 (via line 10, 2 minutes)
Connection 74 - 17 (via line 10, 3 minutes)
Connection 17 - 110 (via line 4, 1 minutes)
Connection 110 - 265 (via line 10, 2 minutes)
Connection 265 - 1 (via line 10, 3 minutes)
Connection 1 - 73 (via line 4, 2 minutes)

Connection 73 - 182 (via line 10, 3 minutes)
Connection 182 - 194 (via line 10, 2 minutes)
Connection 194 - 5 (via line 10, 3 minutes)
```

[6]:
```
# Tubemap transporation islands
from graphlib.metrics.tubemap_islands import TubemapIslandMetric

components, rep_nodes, zone_edges = TubemapIslandMetric(tubemap_graph)()
named_components = {}
component_names = {}

from math import log, ceil
from itertools import product
from string import ascii_uppercase
num_letters = ceil(log(len(components), 26))
all_names = product(ascii_uppercase, repeat=num_letters)
for rep_node, component in components.items():
    name = "".join(next(all_names))
    named_components[name] = component
    component_names[rep_node] = name

from collections import defaultdict
zone_components = defaultdict(list)

for name, stations in named_components.items():
    zone_components[tubemap_graph.adj[stations[0]].zone].append((name,␣
 ↪stations))
for zone, components in sorted(zone_components.items()):
    for name, stations in components:
        print(f"Island {name} (zone {tubemap_graph.adj[stations[0]].zone}):",
              list(map(lambda x: tubemap_builder.station_id_to_name[x],␣
 ↪stations)))

for zone_edge in zone_edges:
    print(f"Islands {component_names[rep_nodes[zone_edge.fr]]} (zone␣
 ↪{tubemap_graph.adj[zone_edge.fr].zone}),␣
 ↪{component_names[rep_nodes[zone_edge.to]]} (zone {tubemap_graph.
 ↪adj[zone_edge.to].zone})",
          f"connected by {tubemap_builder.station_id_to_name[zone_edge.fr]} -␣
 ↪{tubemap_builder.station_id_to_name[zone_edge.to]}",
          f"(on line {zone_edge.line} ({tubemap_builder.
 ↪line_id_to_name[zone_edge.line]}))")
```

```
Island AB (zone 1): ['Aldgate', 'Liverpool Street', 'Bank', "St. Paul's",
'Chancery Lane', 'Holborn', 'Tottenham Court Road', 'Oxford Circus', 'Picadilly
Circus', 'Charing Cross', 'Embankment', 'Waterloo', 'Lambeth North',
'Southwark', 'London Bridge', 'Borough', 'Westminster', "St. James's Park",
'Victoria', 'Sloane Square', 'South Kensington', 'Gloucester Road', 'High Street
Kensington', 'Knightsbridge', 'Hyde Park Corner', 'Green Park', 'Bond Street',
'Marble Arch', 'Lancaster Gate', 'Queensway', 'Baker Street', 'Marylebone',
'Edgware Road (B)', 'Paddington', 'Bayswater', 'Edgware Road (C)', "Regent's
Park", 'Great Portland Street', 'Euston Square', "King's Cross St. Pancras",
'Farringdon', 'Barbican', 'Moorgate', 'Old Street', 'Angel', 'Euston', 'Warren
Street', 'Goodge Street', 'Russell Square', 'Pimlico', 'Temple', 'Blackfriars',
'Mansion House', 'Cannon Street', 'Monument', 'Tower Hill', 'Aldgate East',
'Leicester Square', 'Covent Garden']
Island CO (zone 1): ['Tower Gateway']
Island AY (zone 1.5): ["Earl's Court"]
Island BG (zone 1.5): ['Elephant & Castle']
Island CJ (zone 1.5): ['Notting Hill Gate']
Island CR (zone 1.5): ['Vauxhall']
Island AC (zone 2): ['All Saints', 'Devons Road', 'Bow Church', 'Poplar',
'Blackwall', 'Westferry', 'Limehouse', 'Shadwell', 'Wapping', 'Rotherhithe',
'Canada Water', 'Surrey Quays', 'New Cross', 'New Cross Gate', 'Bermondsey',
'Canary Wharf', 'Heron Quays', 'South Quay', 'Crossharbour & London Arena',
'Mudchute', 'Island Gardens', 'West India Quay', 'Whitechapel', 'Stepney Green',
'Mile End', 'Bethnal Green', 'Bow Road', 'Shoreditch']
Island AG (zone 2): ['Arsenal', 'Finsbury Park', 'Highbury & Islington',
'Holloway Road', 'Caledonian Road']
Island AK (zone 2): ['Barons Court', 'Hammersmith', 'Ravenscourt Park',
'Stamford Brook', 'Goldhawk Road', "Shepherd's Bush (H)", 'Latimer Road',
'Ladbroke Grove', 'Westbourne Park', 'Royal Oak', 'West Kensington']
Island AN (zone 2): ['Belsize Park', 'Chalk Farm', 'Camden Town', 'Kentish
Town', 'Tufnell Park', 'Mornington Crescent']
Island AV (zone 2): ['Clapham Common', 'Clapham North', 'Stockwell', 'Oval',
'Kennington', 'Brixton']
Island BA (zone 2): ['East Acton', 'White City', "Shepherd's Bush (C)", 'Holland
Park']
Island BJ (zone 2): ['Finchley Road', 'Swiss Cottage', "St. John's Wood", 'West
Hampstead', 'Kilburn']
Island BK (zone 2): ['Fulham Broadway', 'Parsons Green', 'Putney Bridge', 'West
Brompton']
Island BX (zone 2): ['Kensal Green', "Queen's Park", 'Kilburn Park', 'Maida
Vale', 'Warwick Avenue']
Island BY (zone 2): ['Kensington (Olympia)']
Island AE (zone 2.5): ['Archway']
Island AS (zone 2.5): ['Bromley-By-Bow']
Island AW (zone 2.5): ['Clapham South']
Island BD (zone 2.5): ['East India']
Island BE (zone 2.5): ['East Putney']
Island BO (zone 2.5): ['Hampstead']
Island CD (zone 2.5): ['Manor House']
Island CI (zone 2.5): ['North Acton']
Island CP (zone 2.5): ['Turnham Green']
Island CT (zone 2.5): ['Willesden Green']
Island DF (zone 2.5): ['Pudding Mill Lane']
Island DG (zone 2.5): ['North Greenwich']
Island DH (zone 2.5): ['Cutty Sark', 'Greenwich', 'Deptford Bridge', 'Elverson
Road', 'Lewisham']
Island AA (zone 3): ['Acton Town', 'Chiswick Park', 'Ealing Common', 'Ealing
Broadway', 'West Acton', 'North Ealing', 'Park Royal', 'South Ealing',
'Northfields']
Island AH (zone 3): ['Balham', 'Tooting Bec', 'Tooting Broadway', 'Colliers
Wood']
Island AL (zone 3): ['Beckton', 'Gallions Reach', 'Cyprus', 'Beckton Park',
'Royal Albert', 'Prince Regent', 'Custom House', 'Royal Victoria', 'Canning
Town', 'West Ham', 'Plaistow', 'Upton Park', 'Stratford', 'Leyton']
Island AO (zone 3): ['Blackhorse Road', 'Tottenham Hale', 'Seven Sisters',
'Walthamstow Central']
Island AR (zone 3): ['Brent Cross', 'Golders Green']
Island AX (zone 3): ['Dollis Hill', 'Neasden']
Island BB (zone 3): ['East Finchley', 'Highgate']
Island BN (zone 3): ['Gunnersbury']
Island BP (zone 3): ['Hanger Lane']
Island BQ (zone 3): ['Harlesden', 'Stonebridge Park', 'Willesden Junction']
Island CM (zone 3): ['Southfields', 'Wimbledon Park', 'Wimbledon']
Island CQ (zone 3): ['Turnpike Lane', 'Wood Green']
Island AQ (zone 3.5): ['Bounds Green']
Island BC (zone 3.5): ['East Ham']
Island BU (zone 3.5): ['Hendon Central']
Island CA (zone 3.5): ['Kew Gardens']
Island CC (zone 3.5): ['Leytonstone']
Island CN (zone 3.5): ['South Wimbledon']
Island AD (zone 4): ['Alperton', 'Sudbury Town', 'Sudbury Hill']
Island AF (zone 4): ['Arnos Grove', 'Southgate']
Island AI (zone 4): ['Barking', 'Upney']
Island AP (zone 4): ['Boston Manor', 'Osterley', 'Hounslow East', 'Hounslow
Central']
Island AT (zone 4): ['Burnt Oak', 'Colindale']
Island BI (zone 4): ['Finchley Central', 'Mill Hill East', 'West Finchley',
'Woodside Park', 'Totteridge & Whetstone']
Island BL (zone 4): ['Gants Hill', 'Newbury Park', 'Redbridge', 'Wanstead']
Island BM (zone 4): ['Greenford', 'Perivale']
Island BZ (zone 4): ['Kenton', 'South Kenton', 'North Wembley', 'Wembley
Central']
Island CB (zone 4): ['Kingsbury', 'Queensbury', 'Wembley Park', 'Preston Road',
'Northwick Park']
Island CF (zone 4): ['Morden']
Island CK (zone 4): ['Richmond']
Island CL (zone 4): ['Snaresbrook', 'South Woodford', 'Woodford']
Island AJ (zone 5): ['Barkingside', 'Fairlop', 'Hainault', 'Grange Hill',
'Chigwell', 'Roding Valley']
Island AM (zone 5): ['Becontree', 'Dagenham Heathway', 'Dagenham East']
Island AU (zone 5): ['Canons Park', 'Stanmore']
Island AZ (zone 5): ['Eastcote', 'Rayners Lane', 'West Harrow', 'Harrow-on-the-
Hill', 'North Harrow', 'Pinner', 'South Harrow']
Island BF (zone 5): ['Edgware']
Island BR (zone 5): ['Harrow & Wealdston']
Island BW (zone 5): ['Hounslow West']
Island CG (zone 5): ['Northolt', 'South Ruislip', 'Ruislip Gardens']
Island DB (zone 5): ['High Barnet']
Island DC (zone 5): ['Cockfosters', 'Oakwood']
Island DE (zone 5): ['Buckhurst Hill']
Island BS (zone 5.5): ['Hatton Cross']
Island BH (zone 6): ['Elm Park', 'Hornchurch', 'Upminster Bridge', 'Upminster']
Island BT (zone 6): ['Heathrow Terminals 1, 2 & 3', 'Heathrow Terminal 4']
Island BV (zone 6): ['Hillingdon', 'Ickenham', 'Ruislip', 'Ruislip Manor',
'Uxbridge']
Island CH (zone 6): ['Northwood', 'Northwood Hills']
Island CS (zone 6): ['West Ruislip']
Island DD (zone 6): ['Epping', 'Theydon Bois', 'Debden', 'Loughton']
Island CE (zone 6.5): ['Moor Park']
Island CY (zone 7): ['Rickmansworth']
Island CZ (zone 7): ['Croxley']
Island CX (zone 8): ['Chorleywood']
Island DA (zone 8): ['Watford']
Island CW (zone 9): ['Chalfont & Latimer']
Island CU (zone 10): ['Amersham']
Island CV (zone 10): ['Chesham']
Islands AA (zone 3), CP (zone 2.5) connected by Acton Town - Turnham Green (on
line 10 (Piccadilly Line))
Islands AB (zone 1), AC (zone 2) connected by Aldgate East - Whitechapel (on
line 4 (District Line))
Islands AB (zone 1), AC (zone 2) connected by Aldgate East - Whitechapel (on
line 6 (Hammersmith & City Line))
Islands AD (zone 4), AA (zone 3) connected by Alperton - Park Royal (on line 10
(Piccadilly Line))
Islands AE (zone 2.5), BB (zone 3) connected by Archway - Highgate (on line 9
(Northern Line))
Islands AE (zone 2.5), AN (zone 2) connected by Archway - Tufnell Park (on line
9 (Northern Line))
Islands AF (zone 4), AQ (zone 3.5) connected by Arnos Grove - Bounds Green (on
line 10 (Piccadilly Line))
Islands AB (zone 1), BJ (zone 2) connected by Baker Street - St. John's Wood (on
line 7 (Jubilee Line))
Islands AB (zone 1), BJ (zone 2) connected by Baker Street - Finchley Road (on
line 8 (Metropolitan Line))
Islands AH (zone 3), AW (zone 2.5) connected by Balham - Clapham South (on line
9 (Northern Line))
Islands AB (zone 1), AC (zone 2) connected by Bank - Shadwell (on line 13
(Docklands Light Railway))
Islands AI (zone 4), BC (zone 3.5) connected by Barking - East Ham (on line 4
(District Line))
Islands AI (zone 4), BC (zone 3.5) connected by Barking - East Ham (on line 6
(Hammersmith & City Line))
Islands AJ (zone 5), BL (zone 4) connected by Barkingside - Newbury Park (on
line 2 (Central Line))
Islands AK (zone 2), AY (zone 1.5) connected by Barons Court - Earl's Court (on
line 10 (Piccadilly Line))
Islands AB (zone 1), CJ (zone 1.5) connected by Bayswater - Notting Hill Gate
(on line 3 (Circle Line))
Islands AB (zone 1), CJ (zone 1.5) connected by Bayswater - Notting Hill Gate
(on line 4 (District Line))
Islands AM (zone 5), AI (zone 4) connected by Becontree - Upney (on line 4
(District Line))
Islands AN (zone 2), BO (zone 2.5) connected by Belsize Park - Hampstead (on
line 9 (Northern Line))
Islands AC (zone 2), AB (zone 1) connected by Bethnal Green - Liverpool Street
(on line 2 (Central Line))
Islands AC (zone 2), BD (zone 2.5) connected by Blackwall - East India (on line
13 (Docklands Light Railway))
Islands AB (zone 1), BG (zone 1.5) connected by Borough - Elephant & Castle (on
line 9 (Northern Line))
Islands AP (zone 4), AA (zone 3) connected by Boston Manor - Northfields (on
line 10 (Piccadilly Line))
Islands AQ (zone 3.5), CQ (zone 3) connected by Bounds Green - Wood Green (on
line 10 (Piccadilly Line))
Islands AC (zone 2), DF (zone 2.5) connected by Bow Church - Pudding Mill Lane
(on line 13 (Docklands Light Railway))
Islands AC (zone 2), AS (zone 2.5) connected by Bow Road - Bromley-By-Bow (on
line 4 (District Line))
Islands AC (zone 2), AS (zone 2.5) connected by Bow Road - Bromley-By-Bow (on
line 6 (Hammersmith & City Line))
Islands AR (zone 3), BU (zone 3.5) connected by Brent Cross - Hendon Central (on
line 9 (Northern Line))
Islands AS (zone 2.5), AL (zone 3) connected by Bromley-By-Bow - West Ham (on
line 4 (District Line))
Islands AS (zone 2.5), AL (zone 3) connected by Bromley-By-Bow - West Ham (on
line 6 (Hammersmith & City Line))
Islands AT (zone 4), BF (zone 5) connected by Burnt Oak - Edgware (on line 9
(Northern Line))
Islands AG (zone 2), AB (zone 1) connected by Caledonian Road - King's Cross St.
Pancras (on line 10 (Piccadilly Line))
Islands AN (zone 2), AB (zone 1) connected by Camden Town - Euston (on line 9
(Northern Line))
Islands AC (zone 2), DG (zone 2.5) connected by Canary Wharf - North Greenwich
(on line 7 (Jubilee Line))
Islands AU (zone 5), CB (zone 4) connected by Canons Park - Queensbury (on line
7 (Jubilee Line))
Islands AA (zone 3), CP (zone 2.5) connected by Chiswick Park - Turnham Green
(on line 4 (District Line))
Islands AV (zone 2), AW (zone 2.5) connected by Clapham Common - Clapham South
(on line 9 (Northern Line))
Islands AT (zone 4), BU (zone 3.5) connected by Colindale - Hendon Central (on
line 9 (Northern Line))
Islands AH (zone 3), CN (zone 3.5) connected by Colliers Wood - South Wimbledon
(on line 9 (Northern Line))
Islands AM (zone 5), BH (zone 6) connected by Dagenham East - Elm Park (on line
4 (District Line))
Islands AX (zone 3), CT (zone 2.5) connected by Dollis Hill - Willesden Green
(on line 7 (Jubilee Line))
Islands AY (zone 1.5), AB (zone 1) connected by Earl's Court - Gloucester Road
(on line 4 (District Line))
Islands AY (zone 1.5), AB (zone 1) connected by Earl's Court - High Street
Kensington (on line 4 (District Line))
Islands AY (zone 1.5), BY (zone 2) connected by Earl's Court - Kensington
(Olympia) (on line 4 (District Line))
Islands AY (zone 1.5), BK (zone 2) connected by Earl's Court - West Brompton (on
```

line 4 (District Line))
Islands AY (zone 1.5), AK (zone 2) connected by Earl's Court - West Kensington (on line 4 (District Line))
Islands AY (zone 1.5), AB (zone 1) connected by Earl's Court - Gloucester Road (on line 10 (Piccadilly Line))
Islands AZ (zone 5), BV (zone 6) connected by Eastcote - Ruislip Manor (on line 8 (Metropolitan Line))
Islands AZ (zone 5), BV (zone 6) connected by Eastcote - Ruislip Manor (on line 10 (Piccadilly Line))
Islands BA (zone 2), CI (zone 2.5) connected by East Acton - North Acton (on line 2 (Central Line))
Islands BB (zone 3), BI (zone 4) connected by East Finchley - Finchley Central (on line 9 (Northern Line))
Islands BC (zone 3.5), AL (zone 3) connected by East Ham - Upton Park (on line 4 (District Line))
Islands BC (zone 3.5), AL (zone 3) connected by East Ham - Upton Park (on line 6 (Hammersmith & City Line))
Islands BE (zone 2.5), BK (zone 2) connected by East Putney - Putney Bridge (on line 4 (District Line))
Islands BE (zone 2.5), CM (zone 3) connected by East Putney - Southfields (on line 4 (District Line))
Islands BG (zone 1.5), AB (zone 1) connected by Elephant & Castle - Lambeth North (on line 1 (Bakerloo Line))
Islands BG (zone 1.5), AV (zone 2) connected by Elephant & Castle - Kennington (on line 9 (Northern Line))
Islands AB (zone 1), AN (zone 2) connected by Euston - Mornington Crescent (on line 9 (Northern Line))
Islands BJ (zone 2), CB (zone 4) connected by Finchley Road - Wembley Park (on line 8 (Metropolitan Line))
Islands AG (zone 2), CD (zone 2.5) connected by Finsbury Park - Manor House (on line 10 (Piccadilly Line))
Islands AG (zone 2), AO (zone 3) connected by Finsbury Park - Seven Sisters (on line 11 (Victoria Line))
Islands AR (zone 3), BO (zone 2.5) connected by Golders Green - Hampstead (on line 9 (Northern Line))
Islands BM (zone 4), CG (zone 5) connected by Greenford - Northolt (on line 2 (Central Line))
Islands BN (zone 3), CA (zone 3.5) connected by Gunnersbury - Kew Gardens (on line 4 (District Line))
Islands BN (zone 3), CP (zone 2.5) connected by Gunnersbury - Turnham Green (on line 4 (District Line))
Islands AK (zone 2), CP (zone 2.5) connected by Hammersmith - Turnham Green (on line 10 (Piccadilly Line))
Islands BP (zone 3), CI (zone 2.5) connected by Hanger Lane - North Acton (on line 2 (Central Line))
Islands BP (zone 3), BM (zone 4) connected by Hanger Lane - Perivale (on line 2 (Central Line))
Islands BR (zone 5), BZ (zone 4) connected by Harrow & Wealdston - Kenton (on

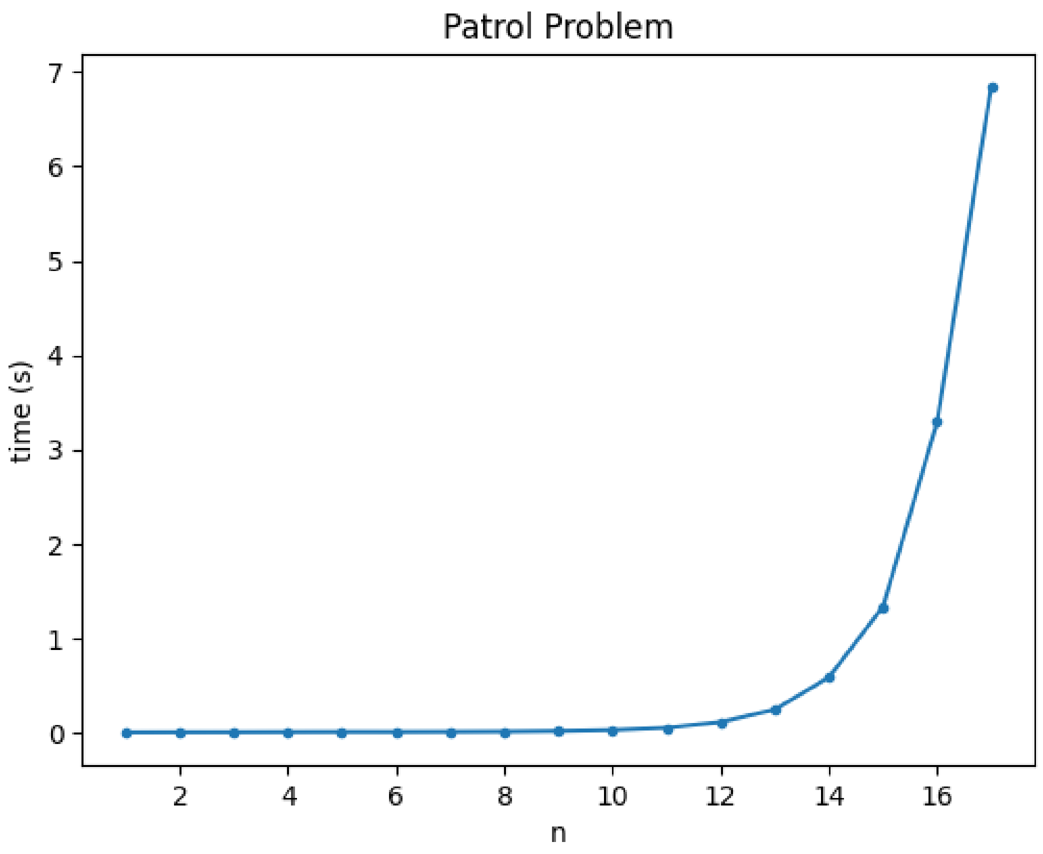
line 1 (Bakerloo Line))
Islands AZ (zone 5), CB (zone 4) connected by Harrow-on-the-Hill - Northwick Park (on line 8 (Metropolitan Line))
Islands BS (zone 5.5), BT (zone 6) connected by Hatton Cross - Heathrow Terminals 1, 2 & 3 (on line 10 (Piccadilly Line))
Islands BS (zone 5.5), BT (zone 6) connected by Hatton Cross - Heathrow Terminal 4 (on line 10 (Piccadilly Line))
Islands BS (zone 5.5), BW (zone 5) connected by Hatton Cross - Hounslow West (on line 10 (Piccadilly Line))
Islands AB (zone 1), CJ (zone 1.5) connected by High Street Kensington - Notting Hill Gate (on line 3 (Circle Line))
Islands AB (zone 1), CJ (zone 1.5) connected by High Street Kensington - Notting Hill Gate (on line 4 (District Line))
Islands AG (zone 2), AB (zone 1) connected by Highbury & Islington - King's Cross St. Pancras (on line 11 (Victoria Line))
Islands BA (zone 2), CJ (zone 1.5) connected by Holland Park - Notting Hill Gate (on line 2 (Central Line))
Islands AP (zone 4), BW (zone 5) connected by Hounslow Central - Hounslow West (on line 10 (Piccadilly Line))
Islands AV (zone 2), AB (zone 1) connected by Kennington - Waterloo (on line 9 (Northern Line))
Islands BX (zone 2), BQ (zone 3) connected by Kensal Green - Willesden Junction (on line 1 (Bakerloo Line))
Islands CA (zone 3.5), CK (zone 4) connected by Kew Gardens - Richmond (on line 4 (District Line))
Islands BJ (zone 2), CT (zone 2.5) connected by Kilburn - Willesden Green (on line 7 (Jubilee Line))
Islands AL (zone 3), CC (zone 3.5) connected by Leyton - Leytonstone (on line 2 (Central Line))
Islands CC (zone 3.5), CL (zone 4) connected by Leytonstone - Snaresbrook (on line 2 (Central Line))
Islands CC (zone 3.5), BL (zone 4) connected by Leytonstone - Wanstead (on line 2 (Central Line))
Islands CD (zone 2.5), CQ (zone 3) connected by Manor House - Turnpike Lane (on line 10 (Piccadilly Line))
Islands AC (zone 2), AL (zone 3) connected by Mile End - Stratford (on line 2 (Central Line))
Islands CE (zone 6.5), CH (zone 6) connected by Moor Park - Northwood (on line 8 (Metropolitan Line))
Islands CE (zone 6.5), CY (zone 7) connected by Moor Park - Rickmansworth (on line 8 (Metropolitan Line))
Islands CF (zone 4), CN (zone 3.5) connected by Morden - South Wimbledon (on line 9 (Northern Line))
Islands AX (zone 3), CB (zone 4) connected by Neasden - Wembley Park (on line 7 (Jubilee Line))
Islands CH (zone 6), AZ (zone 5) connected by Northwood Hills - Pinner (on line 8 (Metropolitan Line))
Islands CI (zone 2.5), AA (zone 3) connected by North Acton - West Acton (on

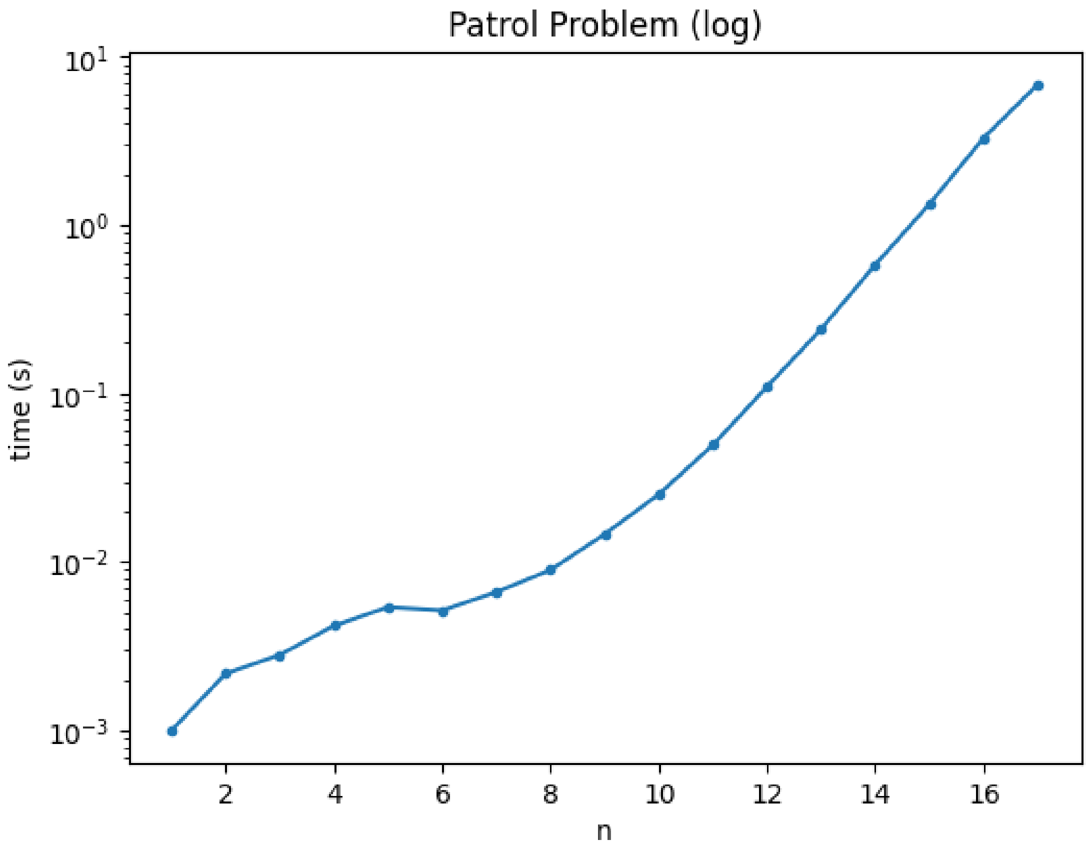
line 2 (Central Line))
Islands CJ (zone 1.5), AB (zone 1) connected by Notting Hill Gate - Queensway (on line 2 (Central Line))
Islands AB (zone 1), BX (zone 2) connected by Paddington - Warwick Avenue (on line 1 (Bakerloo Line))
Islands AB (zone 1), AK (zone 2) connected by Paddington - Royal Oak (on line 6 (Hammersmith & City Line))
Islands AB (zone 1), CR (zone 1.5) connected by Pimlico - Vauxhall (on line 11 (Victoria Line))
Islands AJ (zone 5), CL (zone 4) connected by Roding Valley - Woodford (on line 2 (Central Line))
Islands AC (zone 2), CO (zone 1) connected by Shadwell - Tower Gateway (on line 13 (Docklands Light Railway))
Islands AZ (zone 5), AD (zone 4) connected by South Harrow - Sudbury Hill (on line 10 (Piccadilly Line))
Islands AK (zone 2), CP (zone 2.5) connected by Stamford Brook - Turnham Green (on line 4 (District Line))
Islands AV (zone 2), CR (zone 1.5) connected by Stockwell - Vauxhall (on line 11 (Victoria Line))
Islands BQ (zone 3), BZ (zone 4) connected by Stonebridge Park - Wembley Central (on line 1 (Bakerloo Line))
Islands CU (zone 10), CW (zone 9) connected by Amersham - Chalfont & Latimer (on line 8 (Metropolitan Line))
Islands AC (zone 2), AB (zone 1) connected by Bermondsey - London Bridge (on line 7 (Jubilee Line))
Islands CW (zone 9), CV (zone 10) connected by Chalfont & Latimer - Chesham (on line 8 (Metropolitan Line))
Islands CW (zone 9), CX (zone 8) connected by Chalfont & Latimer - Chorleywood (on line 8 (Metropolitan Line))
Islands CX (zone 8), CY (zone 7) connected by Chorleywood - Rickmansworth (on line 8 (Metropolitan Line))
Islands CZ (zone 7), CE (zone 6.5) connected by Croxley - Moor Park (on line 8 (Metropolitan Line))
Islands CZ (zone 7), DA (zone 8) connected by Croxley - Watford (on line 8 (Metropolitan Line))
Islands CG (zone 5), CS (zone 6) connected by Ruislip Gardens - West Ruislip (on line 2 (Central Line))
Islands DB (zone 5), BI (zone 4) connected by High Barnet - Totteridge & Whetstone (on line 9 (Northern Line))
Islands DC (zone 5), AF (zone 4) connected by Oakwood - Southgate (on line 10 (Piccadilly Line))
Islands DE (zone 5), DD (zone 6) connected by Buckhurst Hill - Loughton (on line 2 (Central Line))
Islands DE (zone 5), CL (zone 4) connected by Buckhurst Hill - Woodford (on line 2 (Central Line))
Islands DF (zone 2.5), AL (zone 3) connected by Pudding Mill Lane - Stratford (on line 13 (Docklands Light Railway))
Islands AL (zone 3), BD (zone 2.5) connected by Canning Town - East India (on

```
line 13 (Docklands Light Railway))
Islands AL (zone 3), DG (zone 2.5) connected by Canning Town - North Greenwich
(on line 7 (Jubilee Line))
Islands DH (zone 2.5), AC (zone 2) connected by Cutty Sark - Island Gardens (on
line 13 (Docklands Light Railway))
```

[8]:
```
def patrol(tubemap_graph, n):
    stations = random.sample(list(tubemap_graph.adj.keys()), n)
    return TubemapPatrolMetric(tubemap_graph, stations)()

from time import perf_counter
x, y = [], []
for n in range(1, 18):
    t = perf_counter()
    patrol(tubemap_graph, n)
    t = perf_counter() - t
    print(n, t)
    x.append(n)
    y.append(t)

_, ax = plt.subplots(1, 1)
ax.set_title("Patrol Problem")
ax.set(xlabel="n", ylabel="time (s)")
_ = ax.plot(x, y, marker='.')
_, ax = plt.subplots(1, 1)
ax.set_title("Patrol Problem (log)")
ax.set(xlabel="n", ylabel="time (s)")
_ = ax.semilogy(x, y, marker='.')
```

```
1 0.0010053200000018628
2 0.0021871410000073865
3 0.0028201750000107495
4 0.004201613000020643
5 0.005418343999991748
6 0.005179190999996308
7 0.006654645999986997
8 0.008996130000014091
9 0.014665296999993416
10 0.025276988000001666
11 0.04965770000001157
12 0.10997884900001509
13 0.241907120999997
14 0.5831076990000099
15 1.333765698999997
16 3.2920550510000055
17 6.845901956000006
```


Patrol Problem
time (s)
n
0
1
2
3
4
5
6
7
2
4
6
8
10
12
14
16



1.3.1 Analysis of algorithms

Subway Patrol Planning The Subway Patrol Planning problem is a variation of the travelling salesman problem (TSP), which essentially boils down to solving for the minimum weight Hamiltonian cycle in a connected graph. This is a well known NP-hard problem in computer science. It can be solved non-optimally by heuristic algorithms in polynomial time, but can only be solved optimally in exponential time. We opted for a traditional dynamic programming approach which takes time proportional to $O(2^n)$ (reasoning below) (as opposed to the brute force algorithm, which checks every permutation of nodes in $O(n!)$ time) where n is the number of stations in the path.

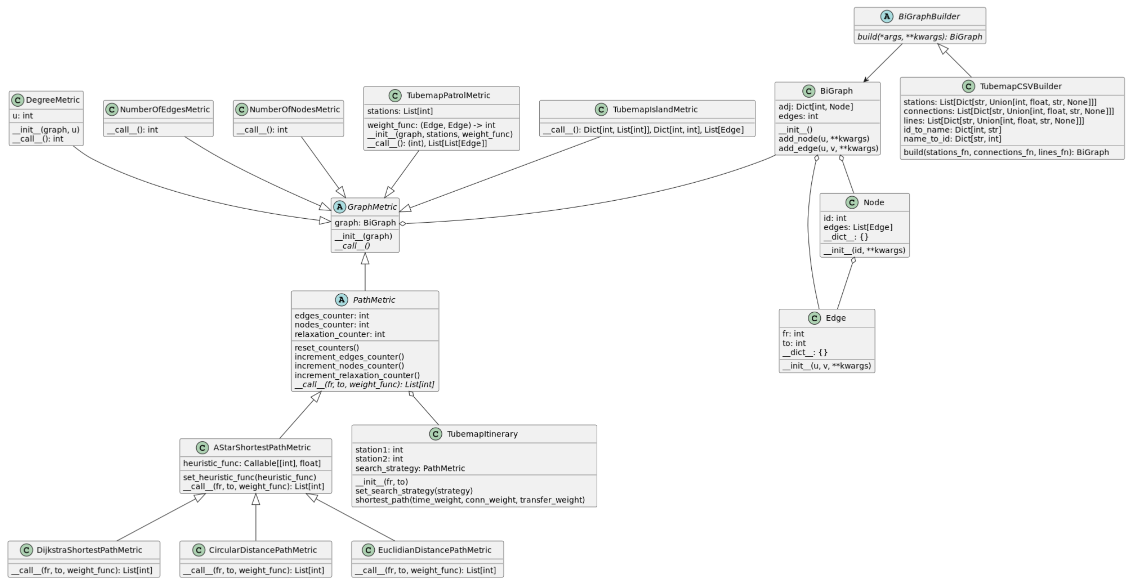
Our implementation of the TSP algorithm first computes a matrix of distances from every station to every other station. The details of this are not important, since the time required becomes insignificant compared to the rest of the algorithm, but we used our existing Dijktra's implementation for this. Next, we recursively compute the minimum cost to visit every station in some order, caching the result of every call to avoid recomputation.

Let $\text{tsp}(u, S)$ be the cost (time) of the trip so far for one recursive call where S is the set of unvisited stations, and u is the current station. Computing one value of $\text{tsp}(u, S)$ requires time proportional to the size of S. Since there are $\binom{n}{k}$ subsets of stations each with k stations, computing every recursive call requires time proportional to $\sum_{k=1}^{n} k\binom{n}{k} = n2^{n-1} = O(2^n)$. Finally, the path is reconstructed by backtracking through the recorded best edges in $O(n)$. The only downside of this algorithm over brute force is the memory usage; caching the results of the recursive calls can take up to $O(2^n)$ extra space. In most cases however, this tradeoff is very worth it to make.

Empirically, we can show that our algorithm has a run time proportional to 2^n by running our algorithm on inputs with different sizes of n and plotting the results (shown in the above cell). The first plot shows clear exponential growth, and the second semilog plot supports this as evident by the (nearly) straight line, with exception to the variance in runtimes of smaller times.

Urbanism Planning The Urbanism Planning problem involves finding the "transportation islands" (connected components that do not cross zone boundaries) of our tubemap graph. We boiled this problem down to finding all subgraphs (components) in a disconnected graph (forest), where the original graph is disconnected on every edge that crosses between zones. The algorithm we used to solve problem is based on depth-first search (DFS) and has an overall runtime complexity of $O(n)$, where n is the number of stations in the tubemap. To determine the connected components, we run a recursive DFS starting from every station in the graph. Each recursive call to DFS checks if the adjacent station is in the same zone, and only continues to recurse if the both the current station and the adjacent stations are in the same zone, effectively disconnecting the graph across zone boundaries. This allows us to identify the all the components within the same zone in a single search, (which creates a single transportation island). After performing every call to DFS, we reconnect the components (or "supernodes") by re-adding edges which connect stations in different zones. The reason why this algorithm has a runtime complexity of $O(n)$ is because our DFS doesn't revisit any station, i.e. every station is visited exactly once.

1.3.2 Updated class diagram


A BiGraphBuilder
build(*args, **kwargs): BiGraph
C DegreeMetric
u: int
__init__(graph, u)
__call__(): int
C NumberOfEdgesMetric
__call__(): int
C NumberOfNodesMetric
__call__(): int
C TubemapPatrolMetric
stations: List[int]
weight_func: (Edge, Edge) -> int
__init__(graph, stations, weight_func)
__call__(): (int), List[List[Edge]]
C TubemapIslandMetric
__call__(): Dict[int, List[int]], Dict[int, int], List[Edge]
C BiGraph
adj: Dict[int, Node]
edges: int
__init__()
add_node(u, **kwargs)
add_edge(u, v, **kwargs)
C TubemapCSVBuilder
stations: List[Dict[str, Union[int, float, str, None]]]
connections: List[Dict[str, Union[int, float, str, None]]]
lines: List[Dict[str, Union[int, float, str, None]]]
id_to_name: Dict[int, str]
name_to_id: Dict[str, int]
build(stations_fn, connections_fn, lines_fn): BiGraph
A GraphMetric
graph: BiGraph
__init__(graph)
__call__()
C Node
id: int
edges: List[Edge]
__dict__: {}
__init__(id, **kwargs)
A PathMetric
edges_counter: int
nodes_counter: int
relaxation_counter: int
reset_counters()
increment_edges_counter()
increment_nodes_counter()
increment_relaxation_counter()
__call__(fr, to, weight_func): List[int]
C Edge
fr: int
to: int
__dict__: {}
__init__(u, v, **kwargs)
C AStarShortestPathMetric
heuristic_func: Callable[[int], float]
set_heuristic_func(heuristic_func)
__call__(fr, to, weight_func): List[int]
C TubemapItinerary
station1: int
station2: int
search_strategy: PathMetric
__init__(fr, to)
set_search_strategy(strategy)
shortest_path(time_weight, conn_weight, transfer_weight)
C DijkstraShortestPathMetric
__call__(fr, to, weight_func): List[int]
C CircularDistancePathMetric
__call__(fr, to, weight_func): List[int]
C EuclidianDistancePathMetric
__call__(fr, to, weight_func): List[int]

1.3.3 Justification of design changes

During week 1, we intended to keep the SOLID principles in mind, and designed the library to adhere to these principles. However, initially our design did not follow the Liskov's Substitution princple in one particular class, namely the `TubemapCSVBuilder` class. During week two, we modified the `TubemapCSVBuilder` class to follow the Liskov's Substitution principle by moving the parameters of the `build` method into the `__init__` method. This way the `build` method is identical for all concrete subclasses of the `BiGraphBuilder` class.

Aside from this, no major design changes were necessary because we had the foresight to design our library in a modular way from the beginning. For instance, we already satisfied the open-closed (O in SOLID) principle; we were able to extend our design and implement new metrics (`TubemapPatrolMetric` and `TubemapIslandMetric`) without modifying existing classes. This was possible because of our `GraphMetric` abstract class which each new graph metric must inherit. In a related vein, the single-responsibility (S in SOLID) and dependency inversion (D in SOLID) principles are also satisfied by this design. Each concrete metric class has the single responsibility of computing its own metric, and users of our library can depend upon the `GraphMetric` class' interface rather than concrete metric implementation.
Incidentally, the `GraphMetric` class had already satisfied the interface segregation (I in SOLID) principle by only having one abstract method. This was because when creating the class we had the single-responsibility principle in mind, and so the single responsibility of each graph metric *is* the only abstract method.

1.4 Self Relfection (Dennis Fong)

Backward I have not done any hands on coding with graphs; the only exposure I had to it was learning about it within an algorithms course. However, I have never heard of the travelling salesman problem until now. I have only used design patterns and SOLID principles before.

Inward I think this work is very interesting, and a good challenge that pushes students out of their comfort zone. I liked how we are able to apply the knowledge we learned (presumably) last year on a practical example, being the London Tube map, since when we learned it we only analyzed the algorithm, without actually implementing it. However, I did not like how we had to dive straight into an example like this since we didnt have any prior experience, coding the algorithm and making it work proved to be quite difficult.

Outward I want people to notice the inclusion of concepts previously taught throughout the classes we made. At first, I questioned if the concepts we learned last year were of any real relevance, such as the SOLID principles, but after implementing them, I am able to see the usage and I hope other people looking at this report see the same.

Forward If I had the chance to do the project again, I would have tried to communicate with the TA/Professor more. A lot of sections were not very clear, and I tried to brute force through the issue, only to learn in the future through the announcements that some things were wrong, causing me to have to work backwards. Other than that, I am happy with the workflow I have achieved with my partner on this project.

1.5 Self Relfection (Luigi Quattrociocchi)

Backward Personally, I have a lot of experience with competitive programming starting from back in high school. Because of this, I am pretty familiar with graphs and pathfinding algorithms. On the other hand, I haven't worked on any projects large enough to warrant obsessing over design principles and organization.

Inward I feel proud of the work my partner and I produced within the past few weeks. In my opinion, we came up with a very modular and extensive library of graphing functionality. I particularly enjoyed implementing the algorithms because of my background in data structures and algorithms. It's something that I like to practice as a hobby in my free time, so it was no issue for me in this lab. I disliked the grunt work of creating UML diagrams and writing tests. I don't think that this opinion is uncommon among my peers, since these things are less exciting than implementations.

Outward One thing I'd like for people to notice about our work is the cleanliness of our git repository. In my opinion, being conscious of version control is a skill that every software engineer should have. My partner and I ensured that we pushed our changes as often as we made them in order to minimize conflicts and merges. Related to this, our folder structure/hierarchy is also clean and organized. Anyone can implement the algorithms, but enforcing organization helps ourselves and others to better understand the structure of our code.

Forward Given the chance to start over, I would have paid more attention to the amount of time each component of the lab should have taken. If the third week of the assignment was not cut, then our team would have struggled to finish in time. Although I can't control the amount of work I have to complete in a given week, I can manage my time better and not have to rush to complete milestones.