CS 6385 Project 2

bxl190001

November 2022

Contents

1 Introduction **2**
- 1.1 Problem . . . 2
  - 1.1.1 Objective . . . 2
  - 1.1.2 Constraints . . . 2
- 1.2 Control Flow Graph . . . 2
  - 1.2.1 Main . . . 2
  - 1.2.2 Algorithm 1 (Greedy) . . . 2
  - 1.2.3 Algorithm 2 (Layers) . . . 2

2 Algorithms **3**
- 2.1 Greedy . . . 3
  - 2.1.1 Description . . . 3
  - 2.1.2 Pseudocode . . . 3
- 2.2 Proof . . . 3
  - 2.2.1 Diameter . . . 3
  - 2.2.2 Minimum Degree . . . 4
- 2.3 Layers . . . 4
  - 2.3.1 Description . . . 4
  - 2.3.2 Pseudocode . . . 5
- 2.4 Proof . . . 5
  - 2.4.1 Diameter . . . 5
  - 2.4.2 Minimum Degree . . . 5
- 2.5 Randomization . . . 5

3 Results **6**

4 Conclusion **8**

5 Instructions **8**

6 Appendix **8**
- 6.1 Source Code . . . 8
- 6.2 Randomly Generated Points (n=15) . . . 12

1 Introduction

In this project, we use heuristic algorithms to solve a network topology design problem.

1.1 Problem

1.1.1 Objective

Minimize the sum of the geometric length of edges given constraints.
$min \sum_{e \in E} l(e)$

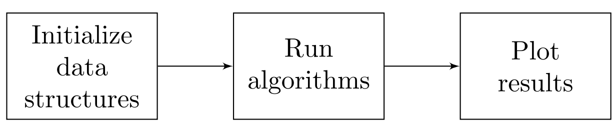
1.1.2 Constraints

The minimum degree of all nodes is 3.
$\forall n \in N \; degree(n) \geq 3$

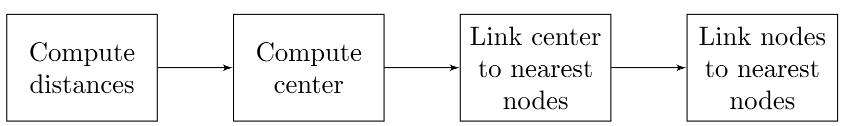
The maximum diameter of the graph is 4.
$diam(G) \leq 4$

1.2 Control Flow Graph

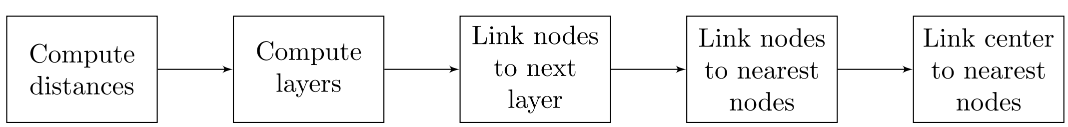
1.2.1 Main



1.2.2 Algorithm 1 (Greedy)



1.2.3 Algorithm 2 (Layers)



2 Algorithms

2.1 Greedy

2.1.1 Description

1. Let N be the node set, $m \leq |N| - 1$ be the minimum degree, and $4 \leq d \leq |N|$ be the diameter.

2. Precompute the distances between all points.

3. Find the point (n_c) that has the minimum sum of link lengths to the $n - m - 1$ nearest nodes and make the links. (diameter)

4. Sort the points by largest minimum link length sum for m links (N_s).

5. Make L links from each node $n_i \in N_s$ to the nearest nodes in $N - n_i - adj(n_i)$ if the number of links of the node is $m - L$. (degree)

2.1.2 Pseudocode

```
greedy(nodes, min_deg)
    n_c, nearest = argmin_dist(nodes, n - min - 1)
    add_edges(n_c, nearest)

    argsort_dist(nodes, min_deg)
    for node in nodes:
        n = min_deg - adj(node)

        if n <= 0:
            continue

        nearest = min_dist(nodes - node, node, n)
        add_edges(node, nearest)
```

2.2 Proof

2.2.1 Diameter

$n - 1 \leq min(G) + max(G) \Rightarrow diam(G) \leq 4$

Let v_m be the vertex with $max(G)$ edges.
Let V_a be the set of vertices that are adjacent to v_m including itself and $V_f = V - V_a$.
Let d be a metric that counts the number of nodes in a path.

The number of vertices in V_f is less than the minimum degree.

$$\begin{aligned}
n-1 &\leq min(G)+max(G) && (assumption)\\
n-max(G)-1 &\leq min(G)\\
V_f &= V-V_a && (definition)\\
|V_f| &= |V-V_a|\\
&= |V|-|V_a| && (V_a \subseteq V)\\
&= n-max(G)-1 && (definition)\\
&\leq min(G)
\end{aligned}$$

There must be at least 1 edge from all vertices in V_f to V_a because of the minimum degree constraint and an edge from V_a to v_m by definition. The maximum distance from any vertex in V_f to v_m is 2 and the maximum distance between any two vertices is then 4.

$$\begin{aligned}
\forall v \in V_f\ adj(v) \cap (V_f - v) &\leq |V_f|-1\\
&\leq min(G)-1\\
\forall v \in V_f\ \exists v_a \in V_a\ (v, v_a) &\in E && (counting)\\
\forall v \in V_f\ d(v, v_m) &= 2\\
\forall v \in V\ d(v, v_m) &\leq 2\\
\forall v_1, v_2 \in V\ d(v_1, v_2) &\leq d(v_1, v_m) + d(v_m, v_2) && (metric)\\
diam(G) &\leq 4
\end{aligned}$$

The center is the vertex with maximum degree of at least $n-m-1$, so the diameter constraint is satisfied assuming $n-m-1 \geq m$. Otherwise if $n \leq 2m$, the vertex of maximum degree has a degree equal to the minimum degree and $max(G)+min(G) = 2m \geq n-1$ and the diameter constraint is still satisfied.

2.2.2 Minimum Degree

At least m links are constructed for each node.

2.3 Layers

2.3.1 Description

1. Let N be the node set, $m \leq |N|-1$ be the minimum degree, and $2 \leq d \leq |N|$ be the maximum diameter.

2. Compute the geometric center and map it to the closest point n_c.

3. Compute the distances between all points. Set r as the maximum distance for n_c.

4. The number of layers $L = \lfloor d/2 \rfloor + 1$, where layer L only contains n_c.

5. Sort the points by distance to the center and equally partition the sorted points by the number of layers.

6. Link every node in layer l to the closest node in the next layer $l+1$ and the nearest nodes so that the degree is m. (diameter)

7. Link n_c to the nearest nodes so that its degree is m. (degree)

2.3.2 Pseudocode

```
layers(nodes, min_deg, diameter)
    center = sum(nodes) / len(nodes)
    n_c = argmin_dist(center, nodes)
    lengths = getDistances(nodes)
    L = d // 2 + 1
    sorted = sort(lengths[center])

    width = len(nodes) / L

    for i in (L - 1)..1:
        lower = i * width
        upper = lower + width

        for node in sorted[lower:upper]:
            next = argmin_dist(node, sorted[upper:upper + width])
            nearest = argmin(lengths[node], min_deg - 1)
            add_edges(node, nearest + next)

    nearest = argmin(lengths[center], min_deg)
    add_edges(center, nearest)
```

2.4 Proof

2.4.1 Diameter

There is at most $\lfloor d/2 \rfloor$ layers. Each layer has a link to the next layer, where the last layer is the center. Thus the maximum distance between any two nodes is $\lfloor d/2 \rfloor + \lfloor d/2 \rfloor \leq d$

2.4.2 Minimum Degree

At least m links are constructed for each node.

2.5 Randomization

```
tests = np.random.random((NUM_TESTS, NUM_POINTS, NUM_DIM))
```

The random input is generated by using np.random.random with a range for each coordinate from 0.0 to 1.0. The input used to generate the graphs for the results is in the appendix.

3 Results

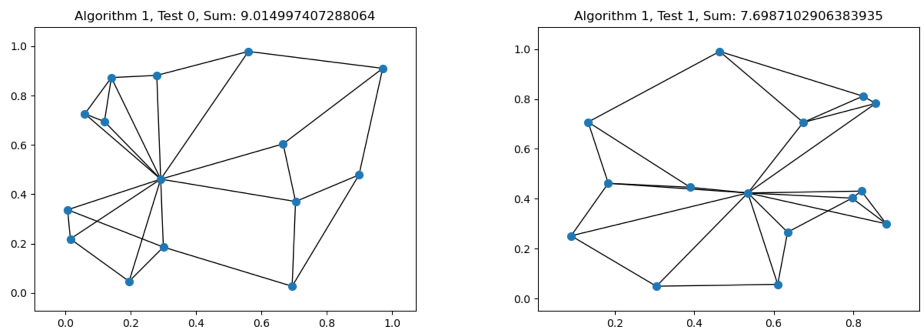


Figure 1: Graphs generated by Algorithm 1 for 15 points

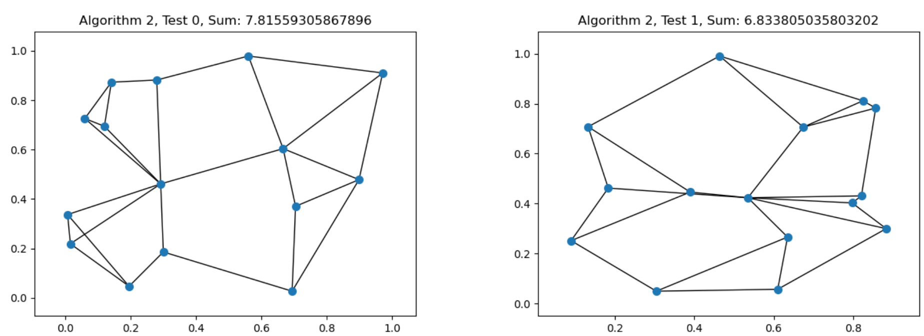


Figure 2: Graphs generated by Algorithm 2 for 15 points

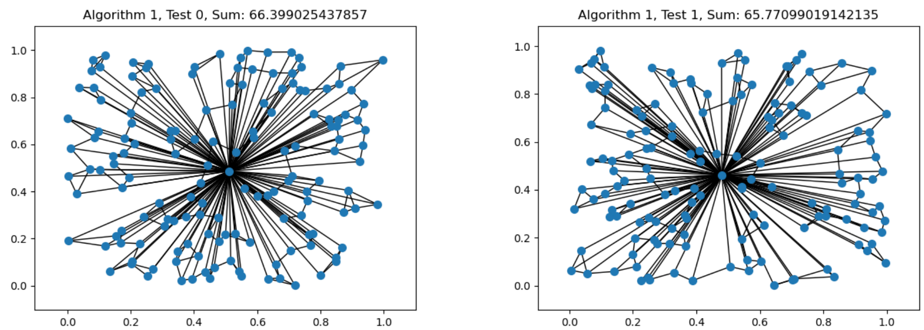


Figure 3: Graphs generated by Algorithm 1 for 150 points

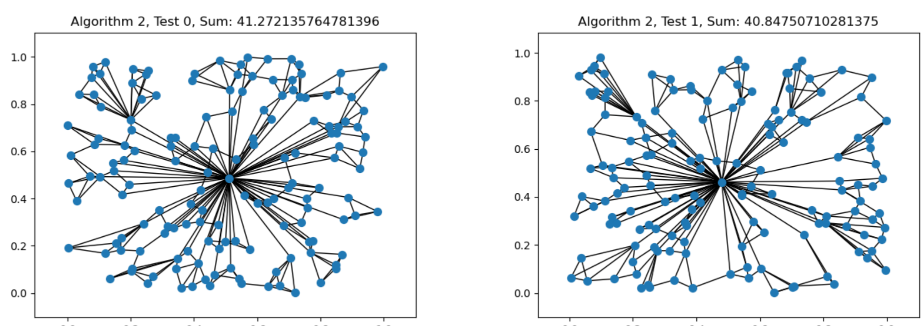


Figure 4: Graphs generated by Algorithm 2 for 150 points

4 Conclusion

One of the algorithms is not always better. The first algorithm (greedy) has a lower average sum of link lengths on all tests when the number of points n is in the range $4 \leq n \leq 10$. For $n > 10$, the second algorithm (layers) has a lower average sum of link lengths.

The second algorithm performs better on larger problem sizes because links to the center are only made to the layer closest to the center, while the first algorithm makes links to the center to all nodes minus a constant. For small problem sizes, the constant factor for the second algorithm results in a smaller number of links to the center.

The experimental running time of both algorithms is $O(n^2)$, where n is the size of the problem. The running time is dominated by the computation of the distances between each node.

5 Instructions

The program uses python 3.10.8, numpy 1.23.4, matplotlib 3.6.2, and networkx 2.8.7 [1] and was tested on Asahi Linux aarch64.

To run the program, write the source code to a file and execute

$ python3 <filename>

where filename is the name of the file.

References

[1] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

6 Appendix

6.1 Source Code

```
#!/usr/bin/env python3
from enum import IntEnum
import math
import matplotlib.pyplot as plt
import networkx as nx
```

```
import numpy as np

MAX_DIAM = 4
MIN_DEGREE = 3
NUM_POINTS = 15
NUM_DIM = 2
NUM_TESTS = 5

class Algs(IntEnum):
    alg1 = 0
    alg2 = 1

def alg1(points, graph, min_deg=MIN_DEGREE, max_diam=MAX_DIAM):
    # min_deg < len(points)
    if min_deg >= len(points):
        return

    # max_diam >= 4 for (n - min_deg - 1) proof
    if max_diam < 4:
        return

    # compute lengths between each point
    lengths = np.sqrt(np.sum((points[:, None] - points[None, :]) ** 2, 2))

    # do not allow self-loops
    np.fill_diagonal(lengths, np.inf)

    # find point with min length to (n - min_deg - 1) nodes
    n = len(points) - min_deg - 1
    nearest = np.argpartition(lengths, n)[:, :n]
    indices = np.arange(len(points))[:, None]
    min_lengths = lengths[indices, nearest]
    min_length_sums = np.sum(min_lengths, 1)
    center = np.argmin(min_length_sums)

    # make link to center for (n - min_deg - 1) nearest nodes
    graph[nearest[center], center] = 1

    # add length sum
    sum = min_length_sums[center]

    # order by largest link length sum for m nearest nodes
    n = min_deg
    nearest = np.argpartition(lengths, n)[:, :n]
    min_lengths = lengths[indices, nearest]
    min_length_sums = np.sum(min_lengths, 1)
```

```
    sorted_points = np.argsort(min_length_sums)[::-1]

    # make links to nearest nodes
    for p in sorted_points:
        # get adjacent nodes
        edges = graph[:, p] > 0
        edges[center] = graph[p, center] > 0
        adj = np.flatnonzero(edges)

        # get n nearest points
        n = min_deg - len(adj)

        if n <= 0:
            continue

        lengths_copy = np.copy(lengths[p])
        lengths_copy[adj] = np.inf
        nearest = np.argpartition(lengths_copy, n)[:n]

        # add edges
        graph[p][nearest] = 1

        # add lengths
        sum += np.sum(lengths_copy[nearest])

    return sum

def alg2(points, graph, min_deg=MIN_DEGREE, max_diam=MAX_DIAM):
    # min_deg < len(points)
    if min_deg >= len(points):
        return

    # geometric center
    gcenter = np.mean(points, 0)

    # map to closest point
    center = np.argmin(np.sum((points - gcenter) ** 2, 1))

    # compute lengths between each point
    lengths = np.sqrt(np.sum((points[:, None] - points[None, :]) ** 2, 2))

    num_outer_layers = max_diam // 2

    # sort by decreasing distance
    sorted = np.argsort(lengths[center])[::-1]
```

```
width = math.ceil((len(sorted) - 1) / num_outer_layers)

sum = 0

# do not allow self-loops
np.fill_diagonal(lengths, np.inf)

start = (len(sorted) - 1) - width * num_outer_layers

for i in range(num_outer_layers):
    lower = max(0, start + i * width)
    upper = lower + width
    cur_layer = sorted[lower:upper]
    next_layer = sorted[upper:upper + width]

    # make links to next layer and nearest nodes
    for p in cur_layer:
        # link to closest node in next layer
        next = sorted[upper + np.argmin(lengths[p][next_layer])]
        graph[p][next] = 1
        sum += np.sum(lengths[p][next])

        # get adjacent nodes
        edges = graph[:, p] > 0
        edges[next] = 1
        adj = np.flatnonzero(edges)

        # get n nearest points
        n = min_deg - len(adj)

        if n <= 0:
            continue

        # link to nearest nodes
        lengths_copy = np.copy(lengths[p])
        lengths_copy[adj] = np.inf
        nearest = np.argpartition(lengths_copy, n)[:n]

        graph[p][nearest] = 1

        sum += np.sum(lengths_copy[nearest])

# check number of links
adj = np.flatnonzero(graph[:, center] > 0)
n = min_deg - len(adj)
```

```
    if n > 0:
        # add edges from center
        lengths[adj] = np.inf
        nearest = np.argpartition(lengths[center], n)[:n]
        graph[center][nearest] = 1
        sum += np.sum(lengths[center][nearest])

    return sum

def plot(tests, graphs, sums):
    for alg in Algs:
        for i in range(len(graphs[alg])):
            pos = {p: point for p, point in enumerate(tests[i])}
            ax = plt.axes()
            G = nx.from_numpy_matrix(graphs[alg][i])
            nx.draw(G, pos, ax, node_size=50)
            ax.set_title(f'Algorithm {alg + 1}, Test {i}, Sum: {sums[alg][i]}')
            ax.tick_params(left=True, bottom=True,
                           labelleft=True, labelbottom=True)
            plt.axis("on")
            plt.savefig(f'alg{alg + 1}_graph{i}.png')
            plt.clf()

def main():
    tests = np.random.random((NUM_TESTS, NUM_POINTS, NUM_DIM))
    graphs = np.zeros((len(Algs), NUM_TESTS, NUM_POINTS, NUM_POINTS), np.uint8)
    sums = np.zeros((len(Algs), NUM_TESTS))
    for i in range(len(tests)):
        sums[Algs.alg1][i] = alg1(tests[i], graphs[Algs.alg1][i])
        sums[Algs.alg2][i] = alg2(tests[i], graphs[Algs.alg2][i])

    print(f'Mean: {np.mean(sums[Algs.alg1])}')
    print(f'Mean: {np.mean(sums[Algs.alg2])}')
    plot(tests, graphs, sums)

if __name__ == '__main__':
    main()
```

6.2 Randomly Generated Points (n=15)

```
[[[0.2923849  0.46134857]
  [0.8988376  0.47864835]
  [0.7048505  0.37069163]
  [0.12030117 0.69451434]
  [0.01619614 0.21913851]
  [0.66680748 0.60415674]
```

```
 [0.56066456 0.97887123]
 [0.69429922 0.02729669]
 [0.28038393 0.88178874]
 [0.9713258  0.91031686]
 [0.19576941 0.04709962]
 [0.00766924 0.33735613]
 [0.14152026 0.87288746]
 [0.06045157 0.72600492]
 [0.30048632 0.1860172 ]]

[[0.88288738 0.2997949 ]
 [0.79711413 0.4026676 ]
 [0.30485375 0.04950409]
 [0.46403163 0.99049421]
 [0.82440496 0.81138269]
 [0.39003484 0.44604892]
 [0.53496414 0.42335716]
 [0.61003214 0.05704884]
 [0.63480178 0.26615945]
 [0.13315384 0.70754364]
 [0.67419955 0.70609489]
 [0.09036003 0.25163582]
 [0.85646717 0.78229587]
 [0.18388287 0.46214109]
 [0.82045584 0.43079743]]
```