EE2T21 Data Communications Networking - 2024
Bonus Assignment # 2

Names: Sjoerd Terlouw Student ID: 5852455

1 Abstract

Implementation of the Dijkstra algorithm in Python and performance analysis using the ER random graph.

2 Example

The following plot gives the shortest path (indicated in red) of a certain graph G using the Dijkstra algorithm. The plot itself is done using the NetworkX python library. The inputs are the edges between the nodes with their weights and the starting and ending node. In the case of figure 1 the starting node is 0 and the ending node is 1. The exact format of the input can be found in the code, found in section 5.

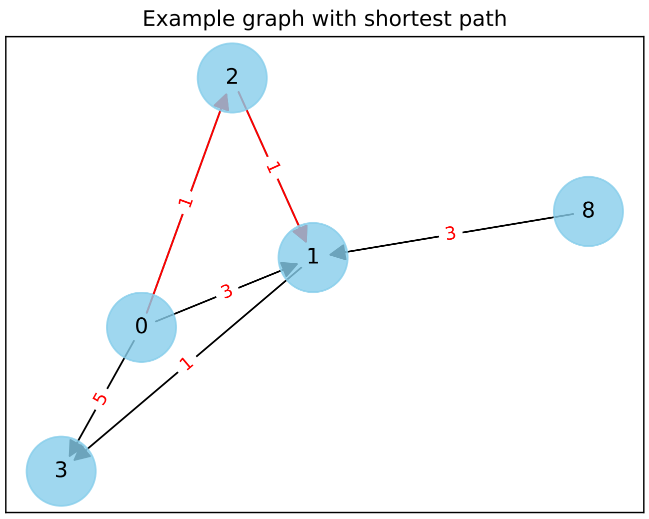

Figure 1: Visualization of the shortest path found by the Dijkstra algorithm - output of Dijkstra.run_example()

3 Erdős Rényi random graph

The random graphs are made using the Erdős Rényi method. It takes k edges from all possible edges given the node length. The edges are always taken to be bidirectional. An example graph generated using the Erdős Rényi method is given in the next figure. Not all nodes are shown in the figure, since not all nodes are connected.

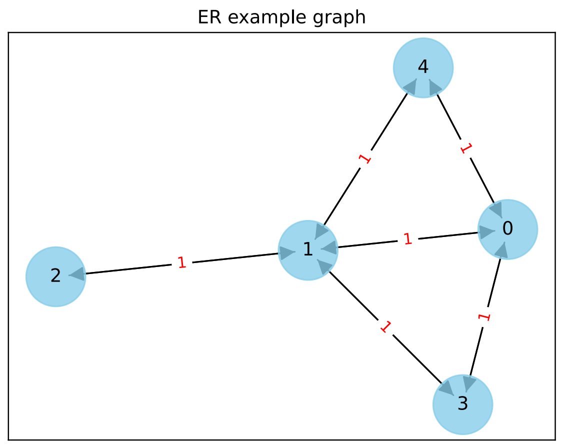

Figure 2: Visualization of a ER random graph with $k = 6$ and $N = 5$ - output graph of Dijkstra.ER()

4 Performance analysis

The performance analysis is done by taking the ER graphs for different sample sizes with the same connectivity. The time complexity is $O(N^2)$ as can be clearly seen in the plot. This is because the implementation is not done using a priority cue.

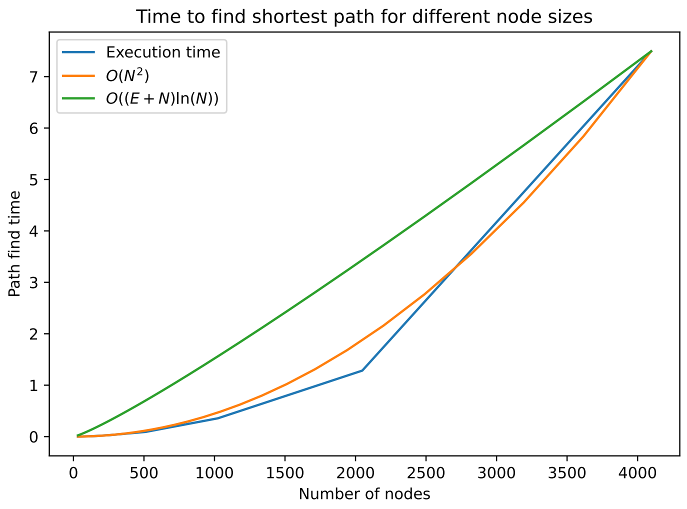

Figure 3: Time complexity of the Dijkstra implementation - output of Dijkstra.performance_analysis()

5 Code

The code is also provided seperately and is on github.

```python
# Dijkstra.py

import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import itertools
import time
import tqdm

def initialize(G, A):
    """
    Find all vertices and initialize distances to infinity

    Also checks if vertices are correct and fixes them if possible
    """
    # Find all vertices
    vertices = np.unique(G['start_node'] + G['end_node'])


    # Initialize distances
    d = [np.inf] * (np.max(vertices)+1)
    d[A] = 0

    pi = [[] for _ in range(np.max(vertices)+1)]

    return np.array(d), pi

def dijkstra(G, A, B):
    """
    Description
    -----------
    Implementation of the Dijkstra algorithm
    The nodes of G should be integers starting from 0

    Parameters
    ----------
    G : dictionary
        Contains the graph in the format
            G = {
                start_node = [0,1,2],
                end_node =   [2,3,4],
                weights =    [2,2,4]
                }
            Which indicates a graph with three weighted edges:
                - Edge going from node 0 to node 2 with weight 2
                - Edge going from node 1 to node 4 with weight 2
                - Edge going from node 2 to node 5 with weight 4
    A :
        Starting node
    B :
        Ending node

    Returns
    -------
    shortest_path : list
        Shortest path from A to B starting with node A and ending
         ↪  with node B
    """
    st = time.time()

    d, pi = initialize(G, A)
    Q = np.array([True] * (np.max(np.array(G['start_node'] +
     ↪  G['end_node']))+1))

    while Q.any() == True:
        u = np.argmin(np.where(Q, d, np.inf))
        Q[u] = False

        if u == B:
            et = time.time()
            return pi[u]+[u], (et-st)
        else:
            indices = np.where(G["start_node"] == u)[0]
            for index in indices:
                w = G["weights"][index]
                v = G["end_node"][index]
                if d[u] + w < d[v]:
                    d[v] = d[u] + w
                    (pi[v]).append(u)

def ER(N, k):
    """
    Description
    -----------
    Take k edges from all possible edges given the amount of nodes
     ↪  N
    """
    start_node = []
    end_node = []
    nodes = np.arange(N)

    pairs = np.array(list(itertools.combinations(nodes, 2)))

    if k > len(pairs):
        raise ValueError("Number of edges (k) exceeds maximum
         ↪  possible edges.")

    connected_pairs = (np.random.choice(len(pairs), k,
     ↪  replace=False))

    start_node = [pairs[idx][0] for idx in connected_pairs]
    end_node = [pairs[idx][1] for idx in connected_pairs]

    weights = [1] * len(start_node * 2)

    G = {"start_node" : start_node + end_node,
         "end_node" : end_node + start_node,
         "weights" : weights}

    return G

def plot_graph(G, pi=None, name="Graph", title = "Example graph
 ↪  with shortest path"):
    """
    Description
    -----------
    Plot the graph using networkX

    Parameters
    -----------
    G : dictionary
        See dijkstra() for format
    pi : list
        Shortest path as obtained by dijkstra()
    """
    G_nx = nx.DiGraph()
    start_node = G["start_node"]
    end_node = G["end_node"]
    weights = G["weights"]

    # Add edges to the graph with weights
    for i in range(len(start_node)):
        G_nx.add_edge(start_node[i], end_node[i],
         ↪  weight=weights[i])
    pos = nx.spring_layout(G_nx)  # positions for all nodes

    # Nodes with labels
    nx.draw_networkx_nodes(G_nx, pos, node_color='skyblue',
     ↪  node_size=1500, alpha=0.8)
    nx.draw_networkx_labels(G_nx, pos, font_size=12,
     ↪  font_family='sans-serif')

    # Edges
    nx.draw_networkx_edges(G_nx, pos, edgelist=G_nx.edges(),
     ↪  edge_color='black', arrows=True, arrowsize=20)

    if pi != None:
        for i in range(len(pi) - 1):
            u = pi[i]
            v = pi[i + 1]
            nx.draw_networkx_edges(G_nx, pos, edgelist=[(u, v)],
             ↪  edge_color='red', arrows=True, arrowsize=20)

    # Edge labels
    edge_labels = {(start_node[i], end_node[i]): weights[i] for i
     ↪  in range(len(start_node))}
    nx.draw_networkx_edge_labels(G_nx, pos,
     ↪  edge_labels=edge_labels, font_color='red')

    plt.title(title)
    if name == None:
        plt.show()
    else:
        plt.savefig(f"{name}.svg")

def run_example():
    """
    Find shortest path of an example graph and save visualization
    """
    start_node = [0,0,2,1,0,8]
    end_node =   [1,2,1,3,3,1]
    weights =    [3,1,1,1,5,3]

    G = {"start_node" : start_node,
        "end_node" : end_node,
        "weights" : weights}
    A = 0
    B = 1

    pi, _ = dijkstra(G,A,B)
    plot_graph(G, pi, name="Graph")
    print(pi)

def performance_analysis(start = 5, stop=13):
    """
    Description
    ------------
    Plot of time of execution for different values of N for the
     ↪  same k
    Values of N are given by N = [2**start, 2**(start+1), ...,
     ↪  2**end]
    """
    N = np.array([2**i for i in range(start, stop)])
    k = 20
    T = []


    for n in tqdm.tqdm(N):
        t = []
        for _ in range(0,20):
            G = ER(n, int((k*n)/2))
            # Only choose from connected nodes
            connected_vertices = np.unique(G['start_node'])
            start_stop_node = np.random.choice(connected_vertices,
             ↪  2)
            _, ct = dijkstra(G, start_stop_node[0],
             ↪  start_stop_node[1])
            t.append(ct)
        T.append(np.average(t))

    fig, ax = plt.subplots()
    ax.plot(N, T, label = 'Execution time')

    # Time complexities
    N = np.logspace(start, stop-1, num=int((stop-start)*5), base=2,
     ↪  dtype=int)
    ax.plot(N, (N**2/N[-1]**2)*T[-1], label = '$O(N^2)$')
    priority_complexity = ((k*N)/2)*np.log(N)
    ax.plot(N, priority_complexity/priority_complexity[-1] * T[-1],
     ↪  label = '$O((E+N)\ln(N))$')

    ax.set_xlabel("Number of nodes")
    ax.set_ylabel("Path find time")
    ax.set_title("Time to find shortest path for different node
     ↪  sizes")
    plt.legend()
    plt.tight_layout()
    plt.savefig(f"time_complexity.svg")


# G = ER(5,6)
# plot_graph(G, None, name="ER_example", title="ER example graph")

# performance_analysis()

# run_example()
```