运筹学实验报告-2

姓名：李奕萱学号：PB22000161

2024 年 12 月 3 日

1 问题背景

在图论中，最短路径问题是最基本也是最重要的问题之一，广泛应用于通信网络、交通调度、地图导航等领域。对于给定的加权图，最短路径问题旨在寻找从源点到目标点的路径，使得路径上的总权重最小。为了求解最短路径问题，许多算法被提出，其中最常用的算法包括 Dijkstra 算法和线性规划（LP）方法。Dijkstra 算法能够在加权图中高效地求解单源最短路径问题，但在某些情况下，尤其是在图的规模较大时，运行时间可能变得较长。与此相对，LP 方法通过建立优化模型来求解最短路径问题，尽管它的计算时间在大规模问题中可能会较长，但可以处理更复杂的约束条件。

本实验通过对比 Dijkstra 算法与 LP 方法在求解最短路径问题中的表现，探索不同规模图的求解效率，并通过实验结果分析它们的优缺点。

2 算法介绍

2.1 随机产生图

```python
def generate_G(nodenum,p):
    for i in range(nodenum):
        for j in range(i+1,nodenum,1):
            p_num=random.random()
            if(p_num<p):
                G.add_edge(i,j)
                G.edges[i,j]['weight']=random.randint(1,10)
    return G
```

本实验中，使用 *networkx package* 生成 $ER(n,p)$ 图，图中共 n 个点，每条边以概率 p 连接。

- 若 $p < \frac{(1-\epsilon)\ln n}{n}$，则图中几乎必有一个孤立点。此时会自动删除孤立点，n 减少。

- 若 $p > \frac{(1+\epsilon)\ln n}{n}$，则图几乎必然连通。

为了确保图的连通性，本实验中选择 $p = 0.8$，并为每条生成的边随机赋予一个距离，取值范围为 $1-10$ 的整数。

2.2 dijksta 算法实现

```python
def dijksta(graph,start):
    queue=[]
    heapq.heappush(queue,(0,start)) #distance node

    distances = {node:math.inf for node in graph}
    distances[start] = 0

    while queue:
        current_distance,current_node=heapq.heappop(queue)

        #if current_distance>known_distance skip
        if current_distance>distances[current_node]:
            continue
        #print(current_node,graph[current_node])
        for neighbor,value in graph[current_node].items():
            weight=value['weight']
            #print(graph[current_node].items())
            distance=current_distance+weight

            if distance<distances[neighbor]:
                distances[neighbor]=distance
                heapq.heappush(queue,(distance,neighbor))
    return distances
```

Dijkstra 算法是一种贪心算法，用于解决单源最短路径问题。我们使用 ‘heapq‘库实现 Dijkstra 算法，其步骤如下：

1. 初始化每个点到 start 的最短距离为 inf，其中 start 设置为 0
2. 将 start 加入队列
3. 进入循环：1. 将 *current_node* 设置为队列中最小点并踢出，2. 遍历 *current_node* 邻边并将相邻点加入队列，同时进行松弛操作。

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```


2.3 LP 算法实现

LP 算法通过将最短路径问题建模为线性规划问题来求解。使用 'copt'或 'gurobi'库来实现 LP 求解。具体约束可以如下表示：

$$\text{Minimize } \sum_{i,j} c_{ij} x_{ij}$$

subject to

$$\sum_j x_{ij} = 1, \quad \sum_i x_{ij} = 1, \quad x_{ij} \in \{0, 1\}$$

其中 x_{ij} 表示是否通过边 (i, j)，c_{ij} 为该边的权重。

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in E} c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{(s,j)\in E} x_{sj} - \textstyle\sum_{(j,s)\in E} x_{js} = 1 \\
& \sum_{(k,j)\in E} x_{kj} - \sum_{(i,k)\in E} x_{ik} = 0, \quad \forall k \in V - \{s, t\} \\
& \sum_{(t,i)\in E} x_{ti} - \sum_{(i,t)\in E} x_{it} = -1 \\
& x_{ij} \geq 0, \forall (i, j) \in E
\end{aligned}$$

2.4 连通性

LP 算法中通过输出 not found 报告该点与源点不连通。

dijksta 算法通过比较最短路径的长度判断是否连通，若有边为正无穷则不连通。

2.5 main 函数

```python
if __name__=="__main__":

    '''wb = openpyxl.Workbook()⋯

    nodenum=10
    p=0.8

    G=nx.Graph()
    G=generate_G(nodenum,p)
    start=list(G.nodes)[0]
    if is_welldefine.is_positive(G):

        starttime=time.time()
        distance={}
        for i in G.nodes:
            distance[i]=lp.shortest_path_lp(G, start, i)
        endtime=time.time()
        print(distance,endtime-starttime)


        print(G)
        #print(start)
        starttime=time.time()
        distances=dijksta.dijksta(G,start)
        endtime=time.time()
        if all(distances[node] != math.inf for node in G):
            print(distances,endtime-starttime)
        else:
            print("is not liantong")
```

1. 首先设置点数以及边生成概率
2. 生成 ER 图
3. 用 LP 方法对每个点求出其最短路径并记录时长
4. 用 dijksta 方法求出最短路径并记录时长
5. 注：因为当无法生成连通图时，有可能出现 0 点不存在情况，故我这里选 G 中第一个点为 $start$。

3 实验结果

3.1 时间与节点数的关系

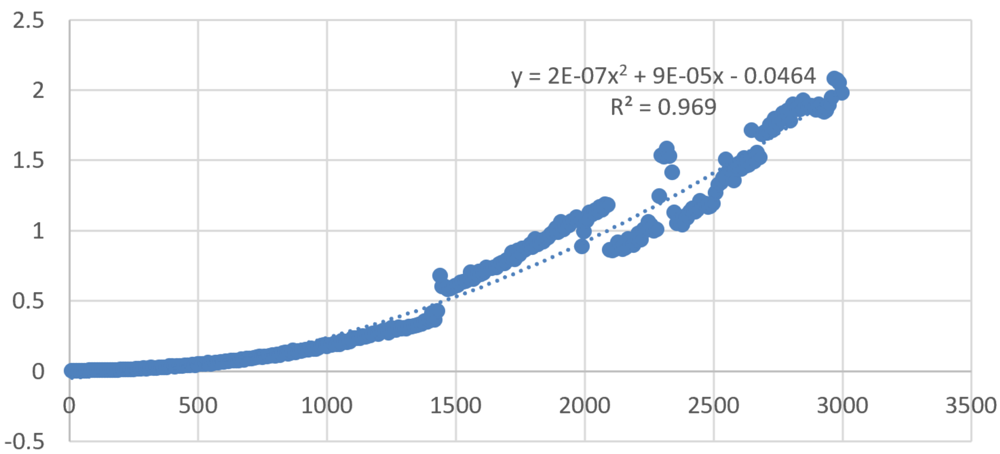

图 1: $time(s) - node$

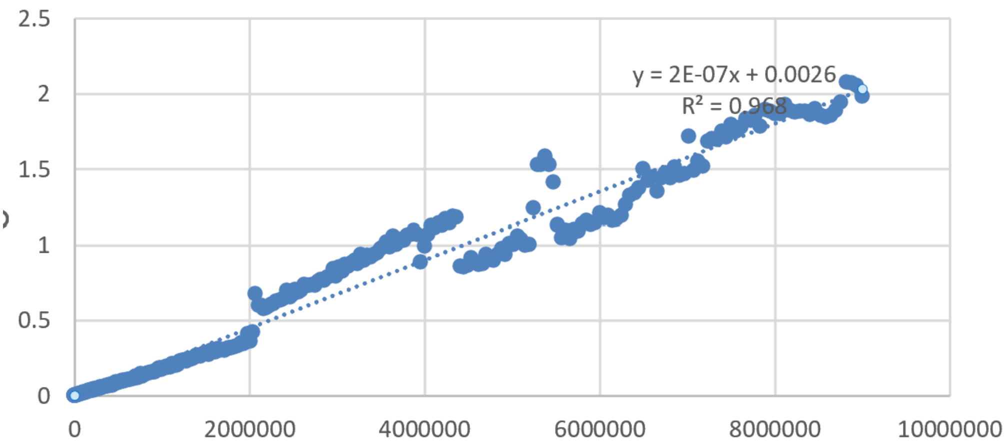

图 2: $time(s) - node^2$

我们对不同节点数的图进行实验，观察运行时间与节点数之间的关系。实验中将节点数从 10 开始，以 10 为步长跑到 3000，每步运行 10 次并计算平均时间，得到如上结果：

做拟合可知运行时间和节点数大致成二阶线性，可能因为边数的不稳定会导致一些突变。

3.2 与 LP 对比

为了比较 Dijkstra 算法和 LP 方法的性能，我们统计了在不同节点数下两种算法的运行时间。以下为部分实验结果（以 $n = 10, 50, 100, 200, 500$ 为例）：

n	LP	Dijksta
10	0.07209038734436035	0.0
50	0.8043262958526611	0.0
100	4.852661848068237	0.0
200	39.18767261505127	0.0
500	1126.7072744369507	0.04920601844787598

可以看出，Dijkstra 算法的效率明显优于 LP 方法，尤其在节点数较小时，Dijks tra 算法几乎可以立即求解最短路径，而 LP 方法则需要更多的计算时间。

4 总结

Dijksta 优点：

1. 高效性：在稠密图中，Dijkstra 算法能够快速求解最短路径。相比于其他如 Bellman-Ford 等算法，Dijkstra 不需要处理负权边，且在没有负权环的情况下，算法能够稳定地求解最短路径。
2. 广泛应用：Dijkstra 算法被广泛应用于网络路由、地图导航等需要快速计算最短路径的场景。

缺点：

1. 性能瓶颈：当节点数和边数增加时，Dijkstra 算法的运行时间增长较快，尤其在大规模图中，可能导致计算效率不高。
2. 不适用于负权边：Dijkstra 算法不能处理带有负权边的图，如果图中有负权边，则需要使用 Bellman-Ford 等算法。