Advanced Algorithms

Assignment 3: Minimum cut

June 14, 2022

Budai Matteo	2057217
Burke Jamie	2044062
Tlepin Sanjar	2041606

Contents

1 Introduction

For this assignment, we implemented and analyzed the performance of two algorithms for the min-cut problem for weighted graphs. The algorithms implemented are:

1. Stoer and Wagner's Deterministic Algorithm;
2. Karger and Stein's Randomized Algorithm.

2 Stoer and Wagner's Deterministic Algorithm

2.1 Data Structure

2.2 Implementation

2.3 Complexity

3 Karger and Stein's Randomized Algorithm

```
KARGER (G,k):
A = +∞
for i = 1 to k:
    t = RECURSIVE_CONTRACT(G)
        if t < min:
            min = t
return min

RECURSIVE_CONTRACT(G=(D,W)):
n= number of vertices in G
if n<=6:
    Gp= CONTRACT(G,2)
    return weight of the only edge (u,v) in Gp
t = n/√2+1
for i = 1 to 2:
    Gi = CONTRACT(G,t)
    wi = RECURSIVE_CONTRACT(Gi)
return min(w1,w2)

CONTRACT(G=(D,W),k):
n= number of vertices in G
for i = 1 to n-k:
    (u,v) = EDGE_SELECT(D,W)
    CONTRACT_EDGE(u,v)
return D,W

CONTRACT_EDGE(u,v):
D[u] = D[u]+D[v]-2W[u,v]
D[v] = 0
W[u,v] = W[v,u] = 0
for each vertex w ∈ V: except u and v:
    W[u,v] = W[u,w] + W[v,w]
    W[w,u] = W[w,u] + W[w,v]
    W[v,w] = W[w,v] = 0

EDGE_SELECT(D,W)
1. Choose u with probability proportional to D[u]
2. Once u is fixed, choose v with probability proportional to W[u,v]
3. return the edge (u,v)
```

This is a randomized algorithm for the computation of a graph. In the next subsections we explain how we have implemented the data structure and the functions of the algorithm.

3.1 Data Structure

For the implementation of the algortihm we used:

- **k**: is a constant ($log^2 n$) used by Karger to repeat Recursive-Contract k times and to obtain an error with probability less or equal to 1/n;
- **V**: is the list of nodes;
- **D**: is the list of the sum of the weights of each node;
- **W**: is the list of the graph with 3 parameters(node,node,weight). Each node is connected with the others and if they are not connected the third parameter is set to 0 otherwise is set to the correct weight.

3.2 Implementation

For the implementation of the algorithm we used these functions:

- **Karger(G,k)**:
  1. This is the main function of the algorithm where we set the timeout to 60 to limit the execution time of large instances;
  2. We start the time and we set the minimum cut to infinite;
  3. We iterate k times to obtain an error with probability less or equal to 1/n;
  4. If the time minus the starting time is greater than the timeout we break;
  5. We execute a copy of V,W and D and then we call the function Recursive-Contract;
  6. If we found a value less than our minimum we update it and we set the discovery time;
  7. In the end we print the Minimum Cut, the Total time and the discovery time.
- **Recursive-Contract(V, W, D)**: In this function we follow exactly the function above with our data structure;
- **Contract(s, V, W, D)**: Also in this function we follow the function above. The only difference is that when we select and contract an edge (u,v) then we remove from V the vertex v to respect the contraction;
- **Contract-Edge(u,v, W, D)**: Also in this case we follow the function above (We update D and W with the new values to execute the contraction of the edge) with our data structure;
- **Edge-Select(V1, D, W)**: Edge selecting randome select first node and look for connected other node, algorithm use:
  1. **Random-Select(C)**:
     (a) Build cumulative weights vector by input array of weights;
     (b) Set random value **r** in range (0, max value of weight);

(c) Run binary search to return node related to selected edge;

2. **binarySearch(array, x)**:

   (a) Special case of binary search according to inequality C[i - 1] $\leq$ r $<$ C[i];

   (b) Devide array in parts;

   (c) Check first value of right part, if random value higher, then go right;

   (d) If value lower check that previous element lower or equal;

   (e) Return ralated name of node.

3.3 Complexity

4 Results

4.1 Table with Min-Cut results

4.2 Graph of the Time Cost of the two Algorithms

4.3 Graph of the Time Cost compared to the Discovery Time of Karger and Stein Algorithm

4.4 Graph of the Time Cost compared to the Asymptotic Complexity of the two Algorithms

5 Conclusion