Projeto #2: Min Weight Cut Randomized Algoritmos Avançados

João Bernardo Tavares Farias
joaobernardo0@ua.pt

***Abstract*—This document was produced as a result of a study about a Randomized Algorithm used to solve the Minimum Weight Cut Problem. It is performed a computational complexity analysis, using the execution time and the number of basic operations carried out. The formal analysis is also compared with the experimental.**

***Index Terms*—random, karger's, algorithm, cut**

I. Introduction

This document was written as a result of the **Project #2** of Advanced Algorithms course and aims to explain the Minimum Weight Cut Problem, what is a random algorithm and how is it used to solve this problem. It is also explained some key concepts about **Graphs** and other important aspects.

II. Graphs

A. Graph explanation

A Graph is a non-linear data structure consisting of vertices (V) and edges (E). The graph is denoted by **G(E, V)**. Graphs are important to solve real-life problems, represent networks, routes and are present in many applications, like social networks, where **each person** represents a **node**.

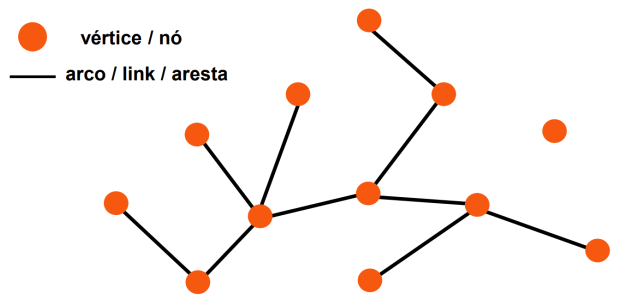

Fig. 1. Graph

B. Ways of representing a graph

A Graph can be represented in different ways, but there are two major representations:

- **Adjacency List**: an array consisting of the address of all the linked lists. The first node of linked list represents the vertex and the remaining lists connected to this node represents the vertices to which this node is connected.
- **Adjacency Matrix**: 2D array of size V x V, where V represents the number of vertices. The slot of adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j.

To undirected graphs the adjacency matrix is symmetric. It can be also used to represent weight graphs **adj[i][j] = w** where **W** represents the weight between edges **i** and **j**.

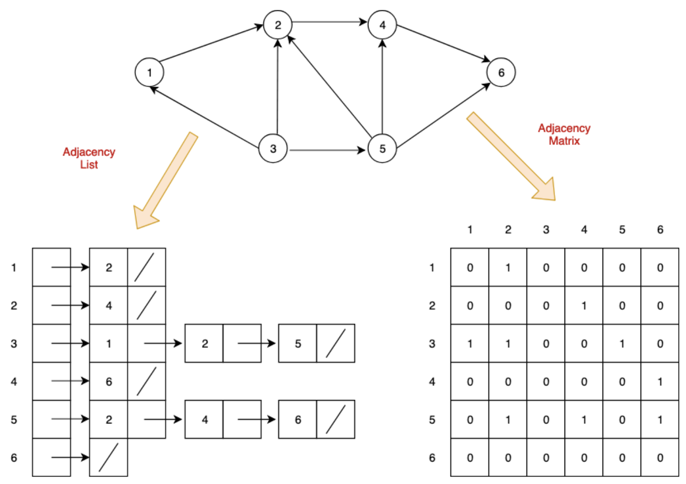

Fig. 2. Adjacency list vs Adjacency matrix

The picture above shows a directed graph represented in different formats.

C. Connected graph

A Graph is connected if there is a path from any point to any other point in the graph. The picture below helps to understand this concept.

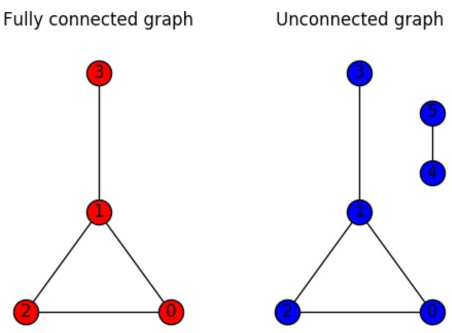

Fig. 3. Connected Graph vs Disconnected Graph

III. Minimum Weight Cut Problem

This is an algorithmic problem where there is an undirected graph **G(V, E)** and the goal is to find a cut where the nodes stay divided into **two** non empty subsets (**S** and **T**), where the weight of crossed edges are as small as possible. This problem be solved in polynomial time by the **Stoer-Wagner** algorithm, but in this work it was used the **Karger's algorithm** once it is a randomized algorithm.

A. Graphs generated

To solve this problem, the graphs were randomly generated, as it was in the first assignment. It was generated graphs with the number of nodes equals to **4** until the number of nodes equals to **456** nodes. For each of these graphs the number of edges were 12.5%, 25%, 50% and 75% of the number of nodes. In the end, there were tested more than 1850 graphs. These graphs has the same positions and the same edge weights as the ones used in the first assignment.

B. Professor Graphs

The teacher provides three files with graphs that apparently can be used in this problem: **SWtinyG.txt**, **SWmediumG.txt** and **SWlargeG.txt**. After the graph build, it is possible to understand that the medium graph doesn't provide a connect graph, so in this problem it can't also be used. In this algorithm, it was not also possible to use the large graph, once this one has more than **7.5** million edges, **1** million nodes and this algorithm is executed until it reaches just two nodes. The tiny graph is processed and the information is recorded in a file. More information is provided in **Complexity Analysis** section.

IV. Karger's Algorithm

Karger's algorithm is a randomized algorithm whose execution time is deterministic. This algorithm presents two main concepts:

- **Supernode**: group of nodes
- **Superedge**: all edges between a pair of nodes

Using this algorithm in the current problem, we start with all nodes being their **supernode** and each **superedge** contains only a single edge. The main goal of this algorithm is to pick a random edge (among all edges), merge its connections, and repeat the process until there are only two supernodes left. These supernodes will define the cut.
It is important to say that this algorithm not always return the correct solution, but it is faster comparing with brute force, that **always** found the correct solution, but can take too much time. In the following example you can see the pseudo-code used to solve this problem.

When two nodes are merged the connections are kept, except when the merge produces redundant edges. In that case, they are removed. Also, if there are multiple edges between the merged nodes, they are removed, too.

Algorithm 1 Karger's Algorithm

```
1: $g \leftarrow Graph(nodes, edges)$
2: **while** $g.numberNodes > 2$ **do**
3:     $e \leftarrow choice(edges)$
4:     $g.contract(e)$
5: **end while**
6: $cut \leftarrow 0$
7: **for** $i \leftarrow g.edges$ **do**
8:     $cut \leftarrow cut + g.weight$
9: **end for**
10: **return** cut
```

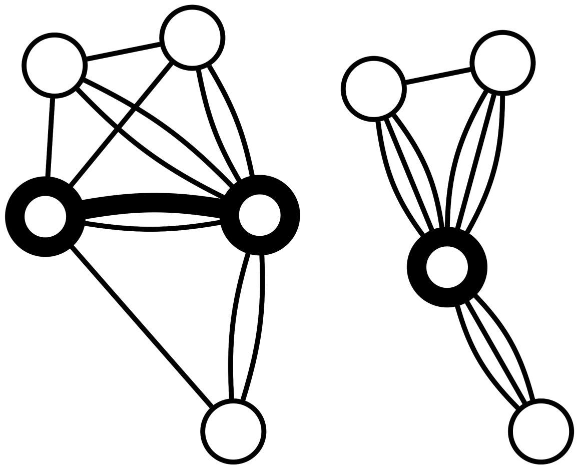
Fig. 4. Contraction of a node

A. Probability of Failure and Success

In this context the probabilities of success and failure are defined by the algorithm correctness when returning the solution. If the algorithm is run 100 times and in 99 of this times it returns the correct solution, the algorithm has 99% of success and 1% of failure.
In this case the probability of returning a min-cut is defined by the following formula $Psucc \geq \frac{1}{\binom{n}{2}}$ and on the other hand the probability of **failure** is $Pfail \leq 1 - \frac{1}{\binom{n}{2}}$.

It is correct to assume that:

- $1 - x = e^{-x}$
- $\binom{n}{2} = \frac{n(n-1)}{2} <= n^2$

Instead of running this algorithm just one time, if we run it **k** times, the probability of failure is

$$Pfail \leq (1 - \frac{1}{\binom{n}{2}})^k$$

where **n** represents the number of nodes.

Using the formulas that were presented above, it is possible to assume that:

$$Pfail \leq (1 - \frac{1}{n^2})^k \equiv (e^{-\frac{1}{n^2}})^k$$

If we run this algorithm n^2 times, we reach the following formula:

$$Pfail \leq (e^{-\frac{1}{n^2}})^{n^2} \leq (e^{-\frac{n^2}{n^2}}) \leq e^{-1} \leq \frac{1}{e} \quad (1)$$

So, the probability of failure is

$$Pfail \leq \frac{1}{e}$$

.

It is possible to go even further, running the algorithm $n^2 \log(n)$.

$$Pfail \leq (e^{-\frac{1}{n^2}})^{n^2 \log(n)} \leq (e^{-\frac{n^2 \log(n)}{n^2}}) \leq e^{-\log(n)} \leq n^{-1} \quad (2)$$

In this case the probability of failure is going to be:

$$Pfail \leq \frac{1}{n}$$

where n represents the number of nodes.

So, after this explanation, it is possible to conclude that the more this algorithm is executed the lower is the **Probability of failure**. On the other hand,

$$Pfailure = 1 - Psuccess$$

so the probability of success is going to be high.

V. Complexity analysis

Usually, random algorithms takes less execution time, but not always return the correct solution. Karger's algorithm experiments run time of $\Theta(n^2)$, since each merge operation takes $\Theta(n)$ time (going through at most $\Theta(n)$ edges and vertices), and there are **n - 2** merges until there are 2 supernodes left.

A. Number of Basic Operations

The number of basic operations directly depends of the number of nodes, once each time an edge is chosen, the number of nodes decreases **one** and this proccess corresponds to a basic operation. The **Karger's Algorithm** is executed until the number of nodes is equals to **2**. Thus, the number of basic operations is directly proportional to the number of nodes. For example, a graph has 100 nodes, so the number of basic operations is equals to 98. The following figure illustrates this relation:

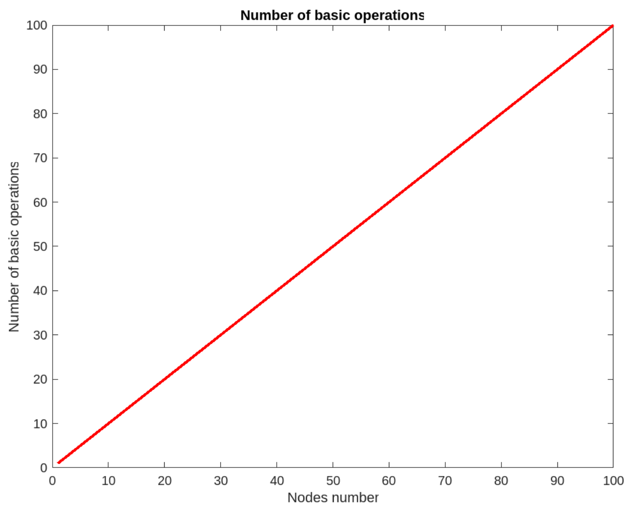

Fig. 5. Number of basic operations

B. Execution Time

The execution time is directly related with the number of nodes and the number of edges of the graph mostly with the last one.

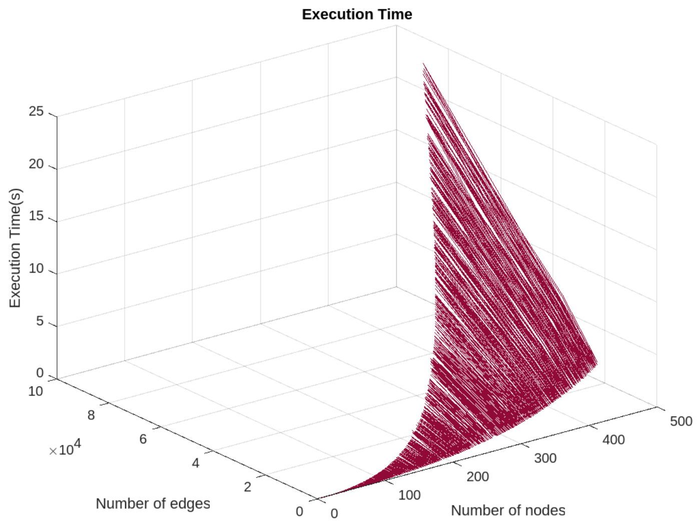

Fig. 6. Execution Time

The figure above represents **three** components: the execution time, the number of nodes of the graph and the number of edges. As the number of nodes increases, the execution time also increases, but the **number of edges** is the principal variable that makes the execution time become bigger. This is proved once, if there are two graphs and the first one has less number of nodes and more number of edges than the second one, the algorithm's execution time is bigger for the first graph than for the second graph.

C. Number of solutions and Configurations Tested

This algorithm returns the minimum cost associated to each graph and the cut edges. Thus, the algorithm only provides a single solution. But, as this algorithm can provide wrong solutions, each graph is tested **10** times and in the end the minimum value is written in a file as the best solution for that graph. According to the formula that were presented before, it is recommended to run the algorithm $n^2 * log(n)$ times to each graph, to reduce the **probability of failure**. This would take to much time and it wasn't done. In this problem, the probability of failure is defined by the following formula:

$$Pfail \leq (e^{-\frac{1}{n^2}})^{10} \leq e^{-\frac{10}{n^2}} \leq \frac{1}{e^{\frac{10}{n^2}}} \tag{3}$$

one more time, the **n** represents the number of nodes. Using an example, for n=100 the **Probability of failure** is the following:

$$Pfail \leq (e^{-\frac{1}{100^2}})^{10} \leq e^{-\frac{10}{100^2}} \leq \frac{1}{e^{\frac{1}{10}}} \leq 0.91 \tag{4}$$

When looking to this value, it is normal to assume that the algorithm isn't good and has less than 10% of accuracy. But what these values really mean is: the probability of returning **always** the correct value in the 10 executions of this algorithm is 10%. Using this implementation, the algorithm just needs to return the right solution one time and in the end it will pick the lower cost (the solution) as the final solution. Thus, it is possible to assume that this algorithm has a good ratio **execution time/algorithm correctness**.
This algorithm ensures that each edge is tested just one time. After it is chosen, it is removed from the list of nodes to test. Beyond this edge, other edges are also removed, for example the edges that produces loops.

D. Accuracy of the Obtained Solutions

In order to analyse the obtained results, a comparation with the results of exhaustive search was made, once this algorithm always find the solution, if exists.
Karger's algorithm finds the solution in about 30% of the cases, which matches the **probability of success** mentioned above. Thus, this algorithm doesn't find always the best solution. This is explained, insofar as the edges has different weights.

Acknowledgment

With this work it was possible to understand the randomized algorithms especially the Karger's Algorithm as well as its properties. Despite of Karger's Algorithm be used in weight graphs, it is also a good option to use in weight graphs. The behaviour of this algorithm with the growth of the number of nodes and the number of edges is also an aspect to highlight. To conclude, this algorithm is not a good one to use in Min Edge Cut Problem, once has a low **probability of success**.

References

[1] Peng Hui How, Virginia Williams, "Min Cut and Karger's Algorithm type, November 28, 2016 https://web.stanford.edu/class/archive/cs/cs161/cs161.1176/Lectures/CS161Lecture16.pdf
[2] Paul Learns Things, Karger's Algorithm: Procedure, https://www.youtube.com/watch?v=KqMGeNZuwfI.
[3] Wikipedia, "Karger's algorithm https://en.wikipedia.org/wiki/Karger%27salgorithm
[4] Wikipedia, 'Stoer–Wagner algorithm‘," https://en.wikipedia.org/wiki/Stoer%E2%80%93Wagner_algorithm.
[5] GeekForGeeks, "Karger's algorithm for Minimum Cut, https://www.geeksforgeeks.org/kargers-algorithm-for-minimum-cut-set-1-introduction-and-implementation/
[6] Top Coder, "COMPUTATIONAL COMPLEXITY PART ONE," https://www.topcoder.com/thrive/articles/Computational%20Complexity%20part%20one