Université Libre de Bruxelles

INFO-F413 - Data structures & algorithms

Karger's algorithm & FastCut

DUDZIAK Thomas : 000542286

October 31, 2022

Contents

1 Introduction

Mincut problem is a well-known and very useful computer science problem. Indeed, solving mincut problem has a myriad of real-world applications such as information retrieval, networks, and many more [2, p. 2]. Thus, there is a concrete need for algorithms to solve this problem.

A popular algorithm to solve that problem is the Karger's randomized algorithm. An improvement to Karger's algorithm, called "FastCut" (or "FastMinCut"), exists. In this report, both algorithms are analyzed, implemented and are compared between each other.

2 Some mathematical recalls

2.1 Graph

A graph is a pair $G = (V, E)$ where V is the set of vertices. $E \subseteq [V]^2$ is the set of edges and the elements of E are 2-element subsets of V. A graph can be drawn by representing the set of vertices as points and the edges as lines joining its two endpoints [1, p. 2]. A loop in a graph is an edge where both endpoints are the same [1, p. 28]. If a graph has no loop and only one edge between every distinct pair of vertices (no multiple edges), then, it is called a simple graph.

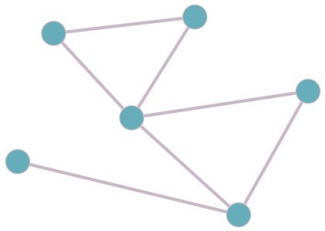

Figure 1: Example of simple graph.

2.2 Multigraph

A multigraph G is a type of graph where loops and multiple edges are allowed. In this report, everytime we will mention a graph, we will implicitely mean multigraph.

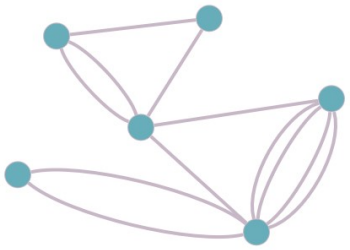

Figure 2: Example of multigraph.

2.3 Neighbours

In a given graph G, a neighbour of a vertex v is a vertex v_2 that is linked to v by any edge e. The set of neighbours of v is denoted $N_G(v)$. When the context is non-ambiguous between several graphs, then, we can omit the graph in the notation and write the set of neighbours as $N(v)$ [1, p. 3].

2.4 Complete graph

Let G be a graph. If all the vertices of G are connected with every other vertex of G, then the graph is said to be complete. When such a graph has n vertices, it is called a K_n [1, p. 3].

3 Cut and minimum cut problem

In this section, we define what is a cut and what is the minimum cut problem in an unweighted graph.

3.1 Cut in a graph

A cut (A, B) of a graph G is a partition of vertices of G into two subsets A and B where $A, B \neq \emptyset$. To determine whether an edge (u, v) is part of the cut (A, B), either u or v is in A and the other is in B. The size of a cut is the amount of edges that belongs to the cut (A, B) (this is true for unweighted graphs; otherwise, we sum the weights of the edges that belong to the cut) [2, p. 2].

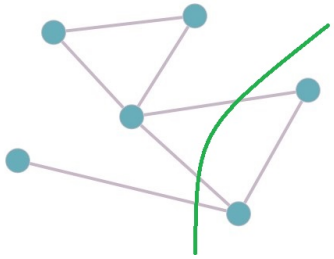

Figure 3: Example of a cut of size 3.

3.2 Minimum cut problem

A minimum cut in a graph G is simply the cut whose size is minimal within G. More formally, given a graph G, we want to find a partition (A, B) of vertices (where $A, B \neq \emptyset$) such that the number of edges connecting those two sets A and B is minimal. This problem is an NP-Complete problem [2, p. 2].

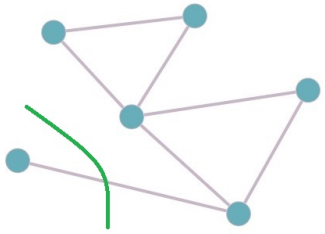

Figure 4: Minimum cut (cut of size 1).

4 Contraction & Karger's algorithm

In the present section, an algorithm which is based on edge-contractions to solve the minimum cut problem will be presented.

4.1 Contraction

Contraction performed on an edge e consists of merging the two endpoints v_1, v_2 of e as an unique vertex whose neighbours are the neighbours of v_1 as well as the neighbours of v_2. More formally, if we contract the edge $e = (v_1, v_2)$, we will replace v_1 and v_2 by a new vertex v_{new} where the set of edges of v_{new} will be $(v_{new}, w)\ \forall\ w \in N(v_1) \cup N(v_2)$. As for notation purposes, the graph resulting from the contraction of the edges in the edge set E in the graph G is denoted by G/E [2, p. 9].

Note that contracting a loop has no effect on the graph. Indeed, both endpoints are from the same vertex and nothing changes after a contraction except that the loop is removed. This is why we can omit loops at graph creation to gain time.

4.2 A Monte-Carlo randomized approach

An algorithm is needed to solve the MinCut problem. The approach we will use is using a randomized algorithm. When using randomization in algorithms, we have two main "classes" : Monte-Carlo and Las Vegas algorithms. While the running time varies from execution to another execution in Las Vegas algorithms, Monte-Carlo algorithms have the same running time but the output may be incorrect with a probability p. The algorithms we will discuss in this paper are of Monte-Carlo type.

The advantage of using a randomized approach in this case resides in the simplicity of the algorithm that will be presented. If a known deterministic algorithm would have been used, it would be way more complicated.

4.3 Karger's algorithm

Karger's randomized algorithm to solve the minimum cut problem is based on the contraction of edges 4.1. Starting with a multigraph $G = (V, E)$, pick an edge $e = (u, v)$ at random and contract it. Repeat the operation until there are two vertices remaining. The only cut of the graph is the minimum with probability $p \geq \binom{n}{2}^{-1}$ [2, p. 10].

The algorithm is extremely short and straightforward. We can define it as the following procedure where $t = 2$ [2, p. 10] :

Algorithm 1 Karger's contraction algorithm.

Input: Multigraph G, integer t
Output: Minimum cut of G

repeat
  Choose an edge $e = (u, v)$ at random in G
  $G \leftarrow G/\{e\}$
until G has t vertices

return G

Corollary 1. *Contraction algorithm can be implemented in $\mathcal{O}(n^2)$ time [2, p. 13].*

5 Improvement with FastCut

In this section, an improvement to Karger's contraction algorithm will be presented. The aim is to improve the success probability of the algorithm to find the cut of minimum size.

5.1 Fastcut algorithm

Instead of contracting until there are two vertices remaining, the graph is divided in two contraction sequences until there are $t = \lceil 1 + \frac{n}{\sqrt{2}} \rceil$ vertices. After that, the algorithm is executed recursively on the two graphs created by the contraction sequences. Only when a graph has ≤ 6 vertices remaining, then the mincut is computed by bruteforce (note that the call to the "Contract" procedure is a call to the algorithm describer in section 4.3).

Algorithm 2 Karger-Stein (FastCut) algorithm.

Input: Multigraph G
Output: Minimum cut of G

if $|V| \leq 6$ **then**
  return Minimum cut of G by bruteforce
end if

$t \leftarrow \lceil 1 + \frac{n}{\sqrt{2}} \rceil$
$G_1 \leftarrow$ Contract(G, t)
$G_2 \leftarrow$ Contract(G, t)

return $min\ \{FastCut(G_1), FastCut(G_2)\}$

Theorem 1. *The running time of FastCut is $\mathcal{O}(n^2 log(n))$ [2, p. 15].*

Proof. Let G be a graph. The algorithm is recursive and one step of the recursion is done by performing two contraction sequences of G until there are $t = \lceil 1 + \frac{n}{\sqrt{2}} \rceil$ and then, performing a recursive call. As shown in Corollary 1, the contraction algorithm runs in $\mathcal{O}(n^2)$ time. Therefore, we can model the running time $T(n)$ by the following recurrence relation :

$$T(n) = 2 * (n^2 + T(\lceil 1 + \frac{n}{\sqrt{2}}))$$

Which is solved by :

$$T(n) = \mathcal{O}(n^2 log(n))$$

□

5.2 Probability improvement

We clearly see that the FastCut algorithm has no gain in running time but the gain resides in the increasing probability compared to the contraction algorithm. Indeed, if we denote p_c the success probability of the contraction algorithm, we know that $p_c \geq \binom{n}{2}^{-1} = \frac{2}{n(n-1)}$. The success probability p_f of FastCut is bounded by $\Omega(1/log(n))$. Since the probability of FastCut is much higher, the amount of runs needed to ensure the probability is high enough to get the correct output is way lower than for the contraction algorithm.

6 Comparing Karger's algorithm and FastCut

This section is dedicated on the comparison between the algorithms as well as their behaviour on different types and size of inputs.

6.1 What do we compare ?

Firstly, let us define **what** should be compared between the two algorithms. It is obvious that the running time can be compared between the two algorithms. But, it is important to notice that the two algorithms have a different success probability. Therefore, there is a need to compare that factor too since an algorithm which is slower but that has an higher probability **may** be more interesting due to the reduction of the number of times it needs to run to have a good answer. To compare the running times as well as the probability, we will use small graphs but also bigger graphs to illustrate how those two factors evolve. In other words, we will give a time budget of 60 seconds (except for much bigger graphs where that time budget will be increased) where we will calculate the amount of correct results over the total number of runs, and this, for both algorithms. Note that **all the graphs used for testing are provided with the code**.

6.2 Complete graphs

A first family of graphs that is interesting to use is the complete graphs K_n (see section 2.4). Indeed, complete graphs are interesting for our tests since we know the minimum cut as well as the evolution of the number of edges between the graph with $(n-1)$ and n vertices.

Lemma 1. *The graph K_n has $\frac{n*(n-1)}{2}$ edges.*

Proof. Let $G = (V, E)$ be a complete graph with n vertices. Every vertex is connected with every other vertex, thus, the degree of each vertex is $(n-1)$. The degree of each vertex counts for twice for the edge count, hence :

$$|E| = \frac{n * (n-1)}{2}$$

□

Lemma 2. *The (unweighted) graph K_n has $(n-1)$ as minimum cut.*

Proof. Let $G = (V, E)$ be a complete graph with n vertices. It is obvious that the minimum cut cannot be less than the minimal degree of G and the degree of every vertex of G is identical. Therefore, we can always partition the set V of vertices of G in two sets V_1 and V_2 such that either V_1 or V_2 contains only one vertex and the number edges crossing V_1 and V_2 is exactly $(n-1)$. □

Since we know the result of the minimum cut as well as the evolution of the number of edges for every value of n, we can compare the evolution of the running time and the success probability for growing graphs.

Graph	Budget time	Statistics			
		Contraction algorithm		FastCut algorithm	
		#Runs	Probability	#Runs	Probability
K_6	1 minute	2650650	0.64	1163607	1
K_{10}	1 minute	838547	0.572	38491	1
K_{20}	1 minute	64410	0.541	1500	1
K_{25}	1 minute	25569	0.538	456	1
K_{50}	2 minutes	2537	0.521	28	1
K_{75}	5 minutes	1031	0.51	9	1
K_{100}	10 minutes	626	0.538	3	1

We can clearly see that FastCut is slower than the classic algorithm by looking at the number of runs done by each algorithm for the same time budget. However, we observe that FastCut is more reliable due to its increased probability. But at some point, we see that FastCut is so slow that it might be worth to execute the contraction algorithm more times until having a good probability for the algorithm to give the good answer.

6.3 Multigraphs

In order to illustrate the success probability of both algorithms in multigraphs instead of simple complete graphs, 3 graphs have been created by drawing arbitrarily on paper. Those graphs are, of course, available as an appendix with the code.

Graph	Budget time	Statistics			
		Contraction algorithm		FastCut algorithm	
		#Runs	Probability	#Runs	Probability
Manual1.txt	1 minute	2707136	0.281	288989	0.994
Manual2.txt	1 minute	2713392	0.264	99642	0.999
triangulation_and_some_multi.txt	1 minute	2741157	0.33	41953	1

Even if the graphs are relatively small, we clearly see the same result as for the simple complete graphs. The number of runs of contraction is significantly higher but the probability of giving a right answer is much lower.

7 Code & implementation

As appendix to this report, implementations of the contraction algorithm (4) and FastCut (5) are available. In this section, we will explore the way those algorithms haver been implemented (technology stack, testing, ...) and how to use the provided code.

7.1 Technology stack

In order to implement the algorithms, a familiar but high-level programming language was ideal since it would bring a great abstraction level to the code for it to stay easily readable. Therefore, the choice has been made to go with C#. C# is a cross-platform, strongly typed, object-oriented, .NET language, close to Java syntactically speaking, which makes it a great choice.

7.2 Code structure

In C#, a solution contains one or several projects with one of the projects being the entry point of the program. The implementation is made such that the algorithms stand in one project while the data structures and miscellaneous classes are in another one. This allows for a simple, readable, maintainable architecture.

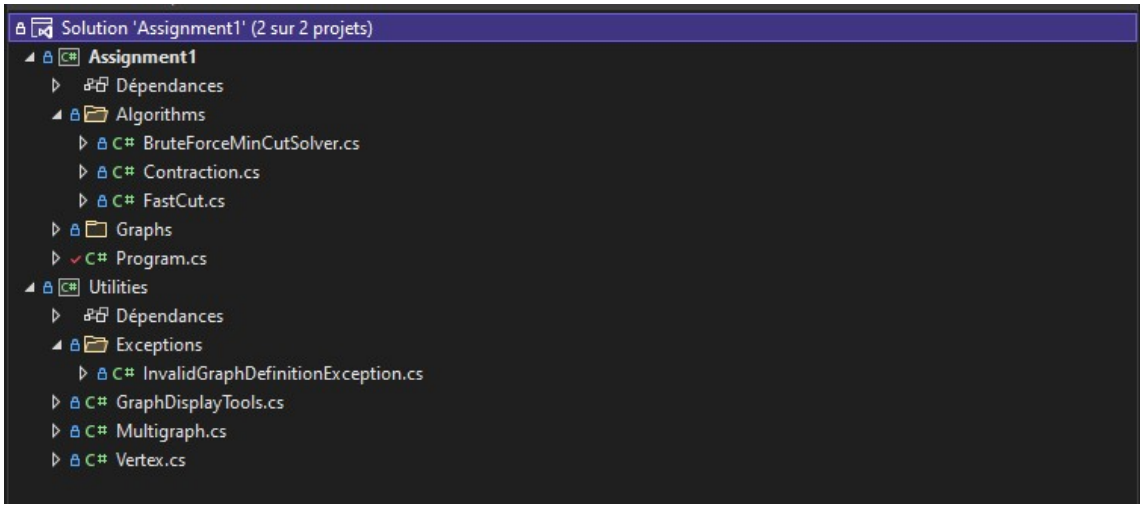

Figure 5: Architecture of the provided code.

7.3 Documentation

The goal of this report is not to document the code provided in the appendix since the whole code has been documented with .NET XML documentation in a precise manner throughout all the development phases. A README file is also available with the provided code for more information.

7.4 Program usage

The program executable can be found in */Assignment1/bin/Release/net6.0* directory. The program requires four arguments to execute properly. The usage is the following : "*./Assignment1.exe* {*path_to_graph.txt*} {*time_budget_seconds*} {*algorithm*} {*expected_answer*}", where *algorithm* is responsible for the execution of the desired algorithm :

Algorithm	Value of argument
Karger's contraction	1
Karger-Stein FastCut	2

If the end user wants to create a new textfile describing the graph $G = (V, E)$ to test the implementation, this can be done by creating a new text file with the following structure :

$|V|$
$|E|$
Edge1_Endpoint1 Edge1_Endpoint2
Edge2_Endpoint1 Edge2_Endpoint2
Edge3_Endpoint1 Edge3_Endpoint2
...

Note that there is a powershell script available to generate the complete graphs that have been used for the tests discussed in section 6.2.

8 Conclusion

After the comparison of two randomized algorithms, **Karger's contraction algorithm** and **Karger-Stein Fastcut algorithm**, we can clearly see a running time difference between those two algorithms. Indeed, the contraction algorithm is faster than Fastcut. However, Fastcut is a robust algorithm since its success probability is much higher than the success probability of the contraction algorithm !

We should also consider the size of the graph we are treating. Indeed, using FastCut is slower and there could be a point where being faster and doing more executions of the contraction algorithm might be more interesting. This aspect has not been discussed in details in this report.

Note that it is quite weird that FastCut has such a good accuracy in the tests realized during this project.

9 Bibliography

[1] R. Diestel. *Graph Theory*. Graduate Texts in Mathematics, 173. Springer Berlin Heidelberg, Berlin, Heidelberg, 5th ed. 2017. edition, 2017.

[2] D. R. Karger and C. Stein. A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, jul 1996.