INFO-F-413 - Randomized algorithms

Assignment 2: Karger's Algorithm

Louis Devroye - 523920

December 23, 2024

Index

Abstract

This report compares the performance of the Contract and FastCut algorithms under the same time budget, with an emphasis on verifying **Theorem 2** (A.1) experimentally. Implementations of both algorithms were created in Python, and tests were conducted on a variety of graph families with meaningful plots illustrating the results.

1 Introduction

The Contract and FastCut algorithms are designed for graph-based problems, focusing on success probabilities in minimizing cuts. This report outlines their implementations, experimental results, and an evaluation of their performance under identical constraints.

2 Implementation Details

Both algorithms were implemented in Python. The Contract algorithm relies on randomized edge contractions to minimize cuts, while the FastCut algorithm optimizes the process by incorporating additional heuristics.

2.1 Graph

I made a Graph implementation to have an easy acces to the edges, a well controlled contraction method and known attributs. It is made of a dictionnary for the edges. each key is the id of a vertex and the values are the vertices that are linked to this vertex. Thus there is no need for a "nodes" list since all the informations in edges are enough.
This graph can be a multigraph (and most of the time it will be when contracting).
the contraction method is used on two vertices. they **must** be neighbors !

2.2 Algorithms

The algorithms and the pseudo random graph generation are implemented in a 'Solver' static class. This is because I don't want to use an attribute graph, otherwise it would get change during the computations, leading to unreliable results.

2.3 Graph generation

the method used to create a pseudo random graph make sure that the graph is connected by doing a Depth First Search. It also adds edges even if the probability is 0% (otherwise there would be no point in analysing a 0% graph). With theses constraints, the randomness is limited and thus the resulting graphs are 'interesting' to analyze.

2.3.1 Karger's Algorithm

The karger's algortihm is fairly simple to implement, we just contract random edges until we have only two remaining vertices. It uses, the 'contract_until' method since it does the same but for 't' vertices instead of 2.

2.3.2 FastCut

The fast cut algortihm is described as recursive, in an attempt to make python a little less slow. I made a simple stack remembering "only" the sub graphs. All the implementation is mostly directly coding each point described in the given pseudo algorithm.

3 Experimental setup

The experiments were conducted on the following graph types:

- Sparse Graphs (with probability of an edge $< 5\%$)
- 'Normal' graphs (with probability of an edge $\approx 40\%$)
- Dense Graphs (with probability of an edge $> 70\%$)
- Complete Graphs (probability of an edge $= 100\%$)

Note : Planar Graphs could have been an interesting add due to their particular limitations.

Due to time restriction all the tests will be done over a sample of $[10, 100]$ vertices' graphs. With 50 iterations for each sample (to avoid edge cases). I also used a seed (523920) for the randomness on the tests to avoid inconsistencies.

Time budget: The time allocated for each iteration is 18 seconds. That also explains the ceiling bound at '18 second' on the computation time plots.

For the A.1 I could not make a link between the results and the Theoretical part to prove that it holds. Allowing a smaller time out or running the algorithms on bigger graphs might show some correlation.

3.1 Sparse

- error rate (1) : The error rate goes to 0 as the graph grows in vertices.
- computation time (2) : FastCut takes exponentially more time but it is still very fast.
- minimal cut (3): FastCut finds smaller cuts in comparison to Karger's algortihm that grows faster.

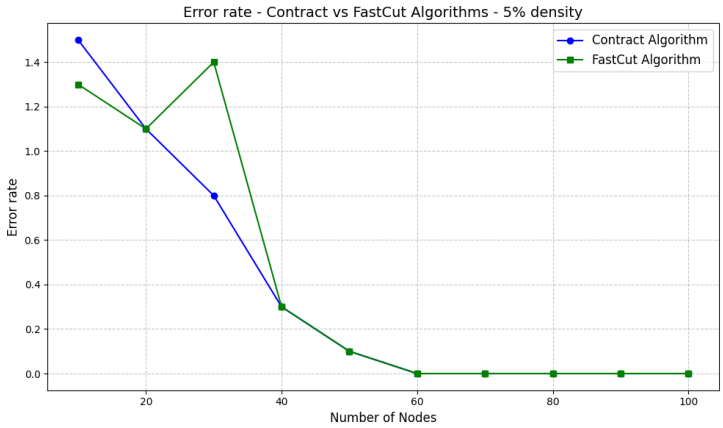

Figure 1: Error rate, Sparse graph

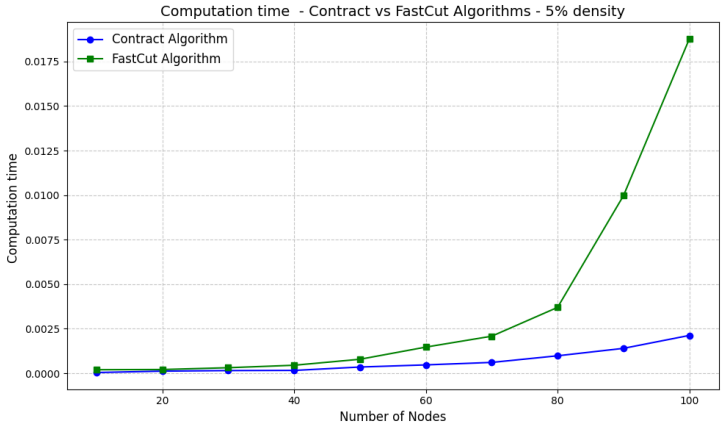

Figure 2: Computation time, Sparse graph

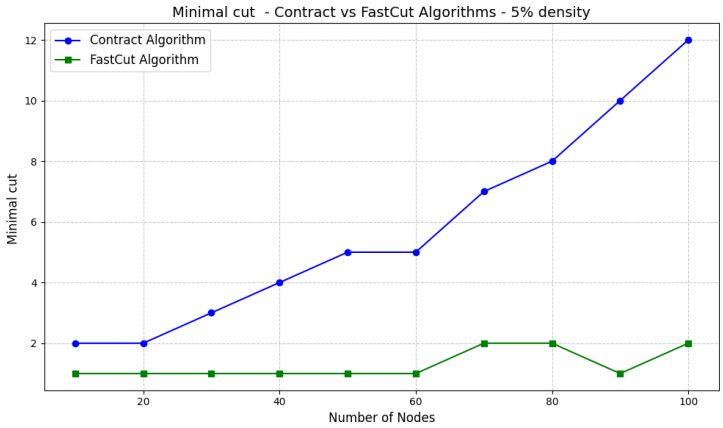

Figure 3: Minimal cut, Sparse graph

3.2 'Normal'

- error rate (4): The error rate is from now on, always at 0.
- computation time (5): The computation time of Karger's algortihm is very much faster than Fastcut. (even when being bounded at 18 seconds).
- minimal cut (6):FastCut's minimal cut grow slowly compared to Karger's algorithm.

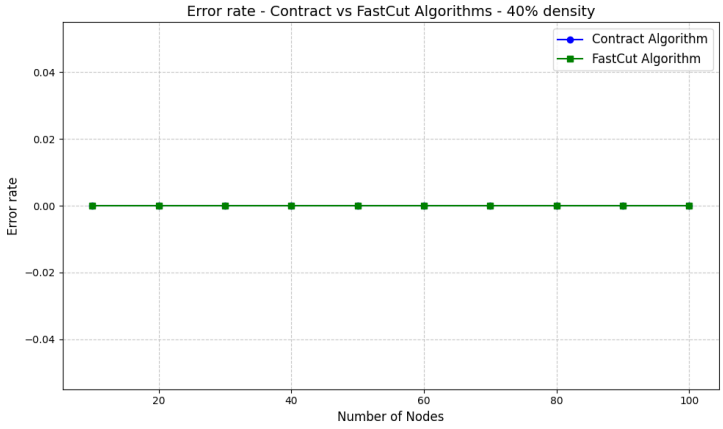

Figure 4: Error rate, 'Normal' graph

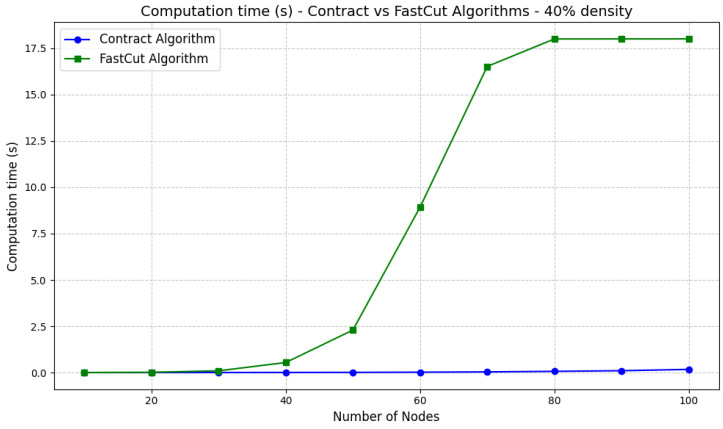

Figure 5: Computation time, 'Normal' graph

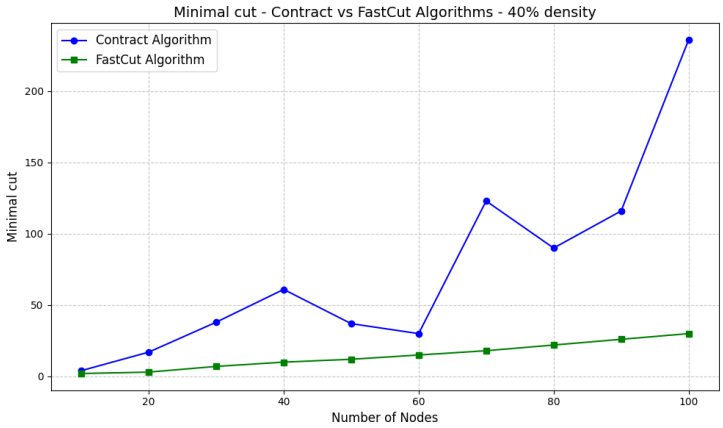

Figure 6: Minimal cut, 'Normal' graph

3.3 Dense

- error rate (7): Still and always 0
- computation time (8): The bounding time is reached faster for FastCut (at 55-60 nodes). Karger's starts to show a faster grow.
- minimal cut (9): Both minimal cuts are growing faster.

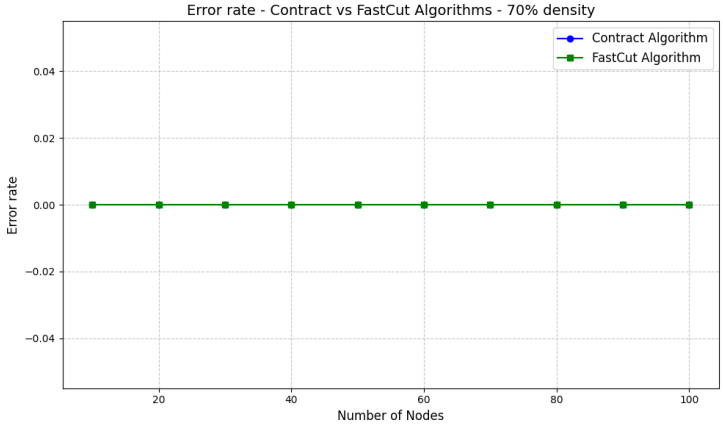

Figure 7: Error rate, Dense graph

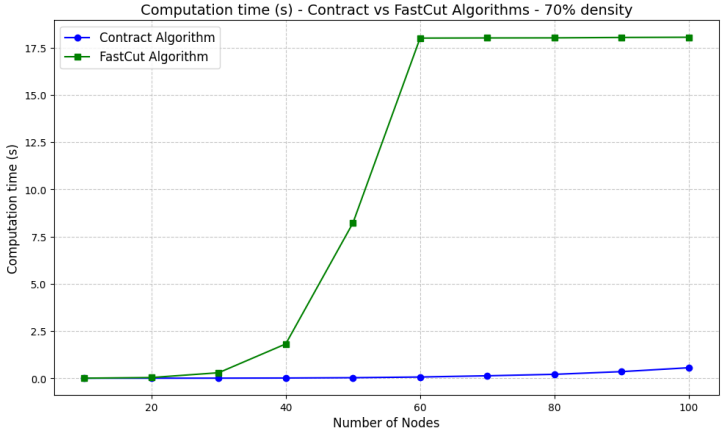

Figure 8: Computation time, Dense graph

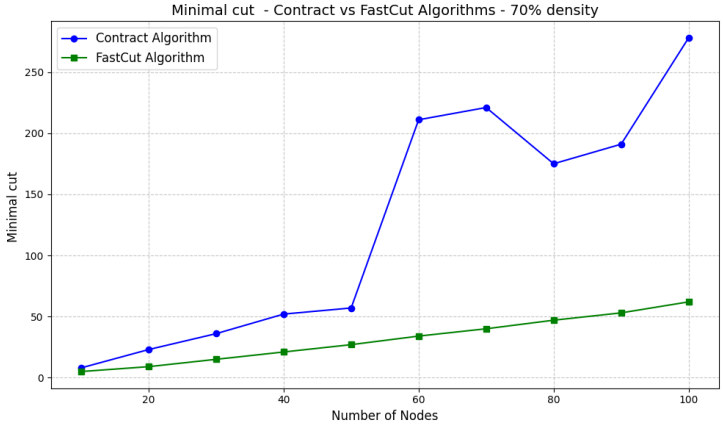

Figure 9: Minimal cut, Dense graph

3.4 Complete

- error rate (10): Always 0%
- computation time (11): FastCut reaches the time bound even faster (at ≈ 52 nodes) and Karger's grow a bit faster.
- minimal cut (12): The minimal cut of FastCut algorithm starts to grow fast considering the time bound but is still lower than the Karger's algorithm that grows very fast.

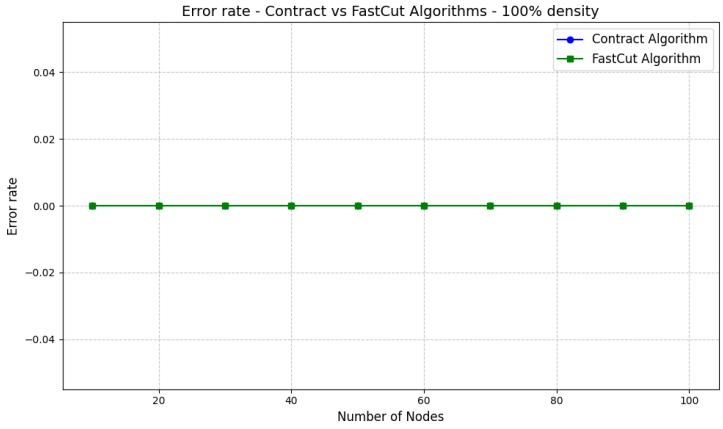

Figure 10: Error rate, Complete graph

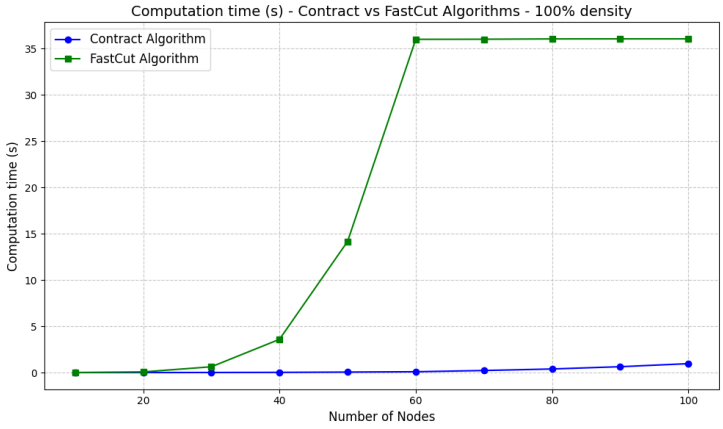

Figure 11: Computation time, Complete graph

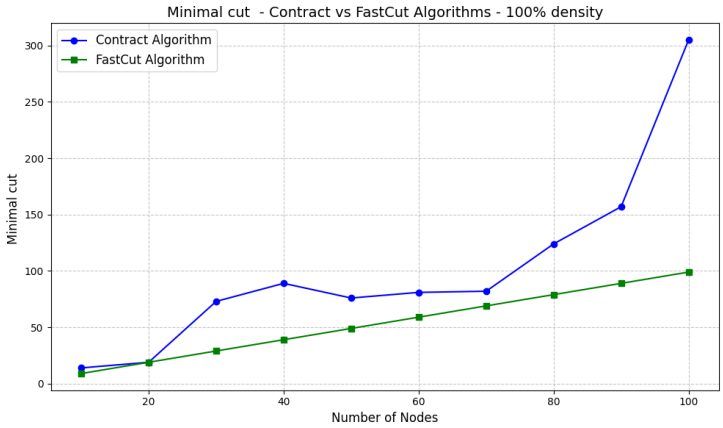

Figure 12: Minimal cut, Complete graph

4 Conclusion

In conclusion, my implementation of the FastCut algorithm gets a better minimal cut and the difference grows more and more as the size of the graph grows (apart for the complete graphs for whom the result it the same). However, the computation time of FastCut, surely due to its recursive nature, also grows (and very much very fast) as the size (number of vertices and edges) grows. This is made testing 100 vertices graphs with more than a 'normal' graph long.

An important note is that I only tested for 18 seconds bounding time. FastCut could have been giving smaller minimal cuts if it was bigger and the other way around. This other way would also have produced more errors as that if the algorithm doesn't have time to do one recursion step, the output is not acceptable.

We can conclude that there is room for fine-tuning between the size of a graph and the time allowed to run the FastCut algorithm such that it will give a better minimal cut than Karger's in an acceptable amount of time.

A Appendix

A.1 Theorem 2

The FastCut algorithm succeeds in finding a minimum cut with probability $\Omega(1/log(n))$. The success probability is therefore much higher that for the simpler Contract algorithm $(2/(n(n-1)))$.