Topics in Parameterized Algorithms   
Mai Yaacovi, Shachaf Katz

Contents

[Introduction 2](#_Toc72317390)

[Feedback vertex cover Problem 2](#_Toc72317391)

[Algorithms 2](#_Toc72317392)

[1. Brute force algorithm 2](#_Toc72317393)

[Steps 2](#_Toc72317394)

[Run time 2](#_Toc72317395)

[Reduction Rules 3](#_Toc72317396)

[2. Bounded search tree 3](#_Toc72317397)

[Steps 3](#_Toc72317398)

[Run time 3](#_Toc72317399)

[3. Randomized 3](#_Toc72317400)

[Steps 3](#_Toc72317401)

[Runtime 3](#_Toc72317402)

[Run-time graphs 4](#_Toc72317403)

[Improvements 5](#_Toc72317404)

[Brute force 5](#_Toc72317405)

[Description 5](#_Toc72317406)

[Run time 5](#_Toc72317407)

[Results 5](#_Toc72317408)

[Running simultaneously 6](#_Toc72317409)

[Description 6](#_Toc72317410)

[Run-time graphs 6](#_Toc72317411)

[Half-cycles 6](#_Toc72317412)

[Third Cycles 7](#_Toc72317413)

[All cycles 9](#_Toc72317414)

[References 9](#_Toc72317415)

# 

## Introduction

In this project we will discuss different algorithms for solving the feedback vertex cover problem.  
We will present the problem and 3 algorithms that we implemented, including brute force algorithm, bounded search tree and randomized algorithms as described in the book parameterized-algorithms [1](#_1._parameterized-algorithms_-)   
Followed by run time for different inputs and algorithm improvements.

## Feedback vertex cover Problem

Given a multi graph G=(V,E) {\displaystyle G=(V,E)}and a positive integer {\displaystyle k}k.  
Is there a subset X{\displaystyle X\subseteq V} with  {\displaystyle |X|\leq k}such that in the graph G\X{\displaystyle G} there are no cycles.

## Algorithms

### 1. Brute force algorithm

Input – Graph G; non-negative integer k.  
Output – Feedback vertex cover from size K if exists, Null otherwise

#### Steps

1. For each combination for – remove this combination from the graph and check if the new graph consists of cycle – if no return this combination

#### Run time

1. Number of iterations
2. In each iteration check if the graph consists of a cycle – O(n(n+m))

**In total –**

In regard to the following algorithms, we will use these reduction rules.

### Reduction Rules

1. If there is a loop at vertex , delete from the graph and decrease by 1.
2. If there is an edge of multiplicity larger than 2, reduce its multiplicity to 2.
3. If there is a vertex of degree at most 1, delete .
4. If there is a vertex of degree 2, delete and connect its two neighbors by a new edge.
5. If , terminate the algorithm and conclude there is no solution for the instance

### 2. Bounded search tree

Input – Graph G; non-negative integer k.  
Output – Feedback vertex cover from size K if exists, Null otherwise.

#### Steps

1. Run reductions 1-5 and receive reduced instance where and .
2. If we find that there is no solution, return no. Else if is empty, return – the set of all nodes removed in reduction (1). Else, continue.
3. Sort nodes of the graph by their degree in descending order.
4. Get first vertices with the largest degrees.
5. For each node chosen, recursively call the algorithm with .
6. If one of the recursive calls returns a solution , conclude that is a solution for . Else return that there is no solution.

#### Run time

### 3. Randomized

Input – Graph G; non-negative integer k.  
Output – Feedback vertex cover from size K if exists, Null otherwise.

#### Steps

1. Run reductions 1-5 and receive reduced instance where and .
2. If we find that there is no solution, return no. Else if is empty, return – the set of all nodes removed in reduction (1). Else, continue.
3. Choose an edge in uniformly at random and a node of that edge independently and uniformly at random.
4. Recursively call the algorithm with .
5. If the recursive call returns no, return no. Else if the recursive call returns a solution , return .
6. Repeat independently times.

#### Runtime

### Run-time graphs

We now present the runtime graphs we received when running the algorithms on different inputs.

## Improvements

### Brute force

We added the following step in brute force algorithm (marked in blue)

#### Description

1. Delete nodes from degree 0 or 1 .
2. For each combination for – remove this combination from the graph and check if the new graph consists of cycle – if no return this combination

Nodes from degree 0 or 1 – cannot be part of any cycle – therefor removing them will:

* Decrease number of nodes – which will affect dramatically the run-time.
* Won’t affect the correctness of the solution – because those nodes will not be part of minimal feedback vertex cover.

#### Run time

1. Removing nodes from degree 0 or 1 can be done in O(n+m)

Therefore, in total the run-time was not affected and is still

Since this step will decrease number of nodes in some inputs, the run-time will be improved as we can see in the example below.

#### Results

Input- Graph G with n nodes (increasing as can be seen in the graph) and k equals to 8.

### Running simultaneously

#### Description

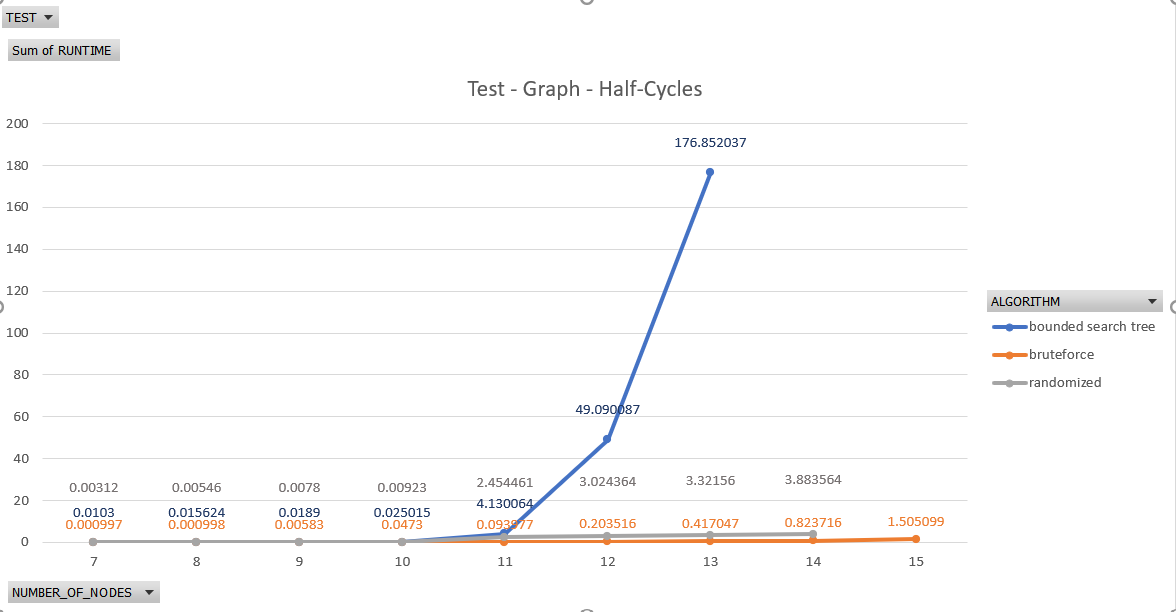
When running the algorithms, we were surprised to see that there were inputs in which the brute force algorithm finished running sooner than the randomized and bounded search tree algorithms. We also noticed that in general, each algorithm gave us different runtimes and in each input there was always one algorithm that ran faster than the others (whether that means finding a solution or finding that there is no solution).   
The specific problem we faced is that we would run one of the algorithms on a certain input and have to wait several hours for it to finish calculating, where if we ran a different algorithm on the same input, it would finish in a matter of minutes or even seconds.   
In order to overcome this problem, we decided to create a thread for each algorithm and for each input, run all the threads simultaneously. Once one of the threads finished running the algorithm and came to a conclusion (either solution or no solution) it would terminate the other threads.   
This way, we would always receive the best runtime possible out of all three algorithms for each input we used.

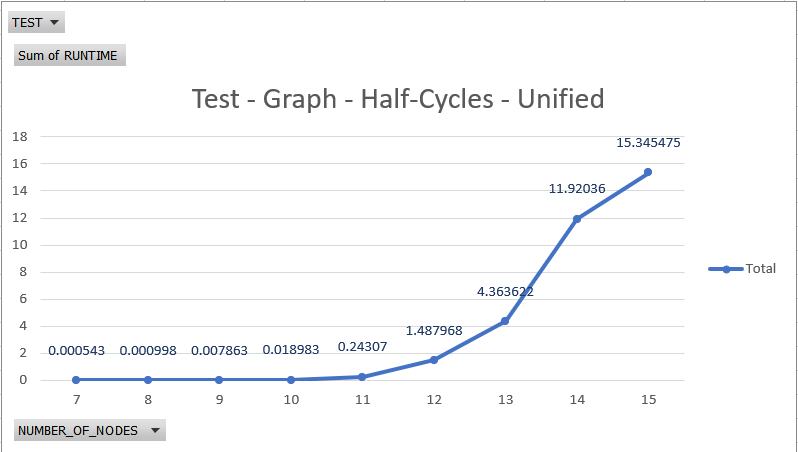
#### Run-time graphs

We now present the runtime graphs we received for each thread (each running a different algorithm) on the same input.

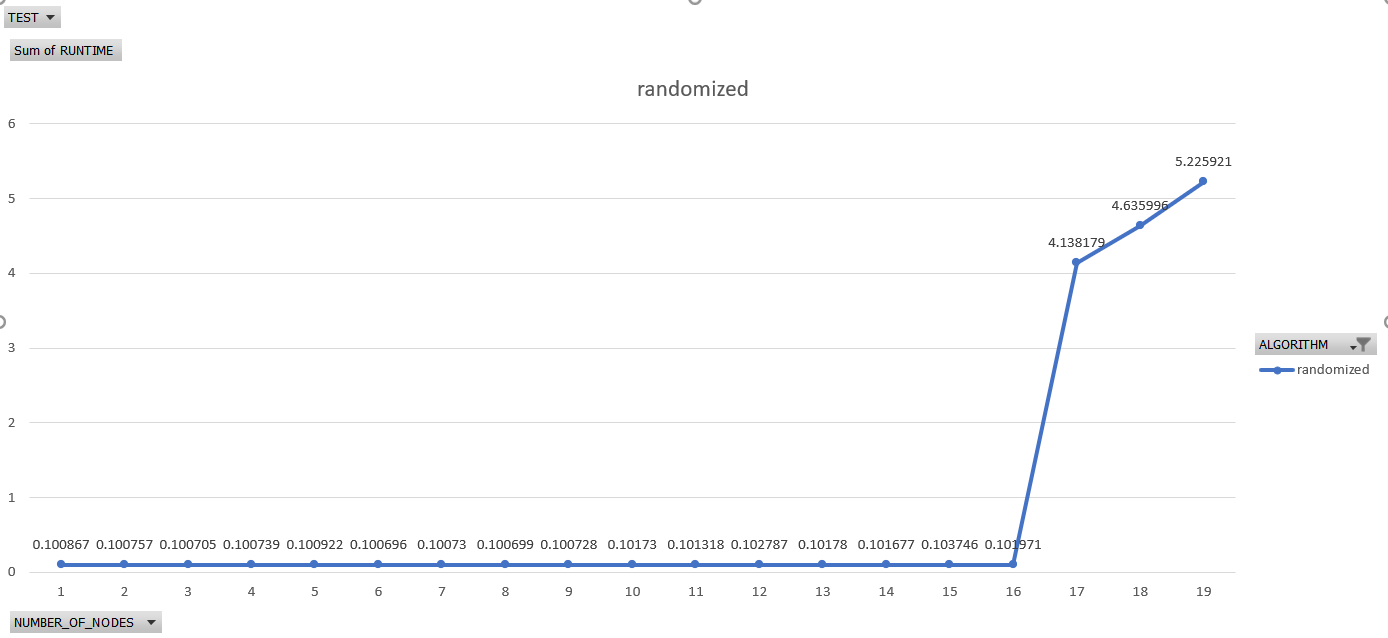
//Why do we think this happened?

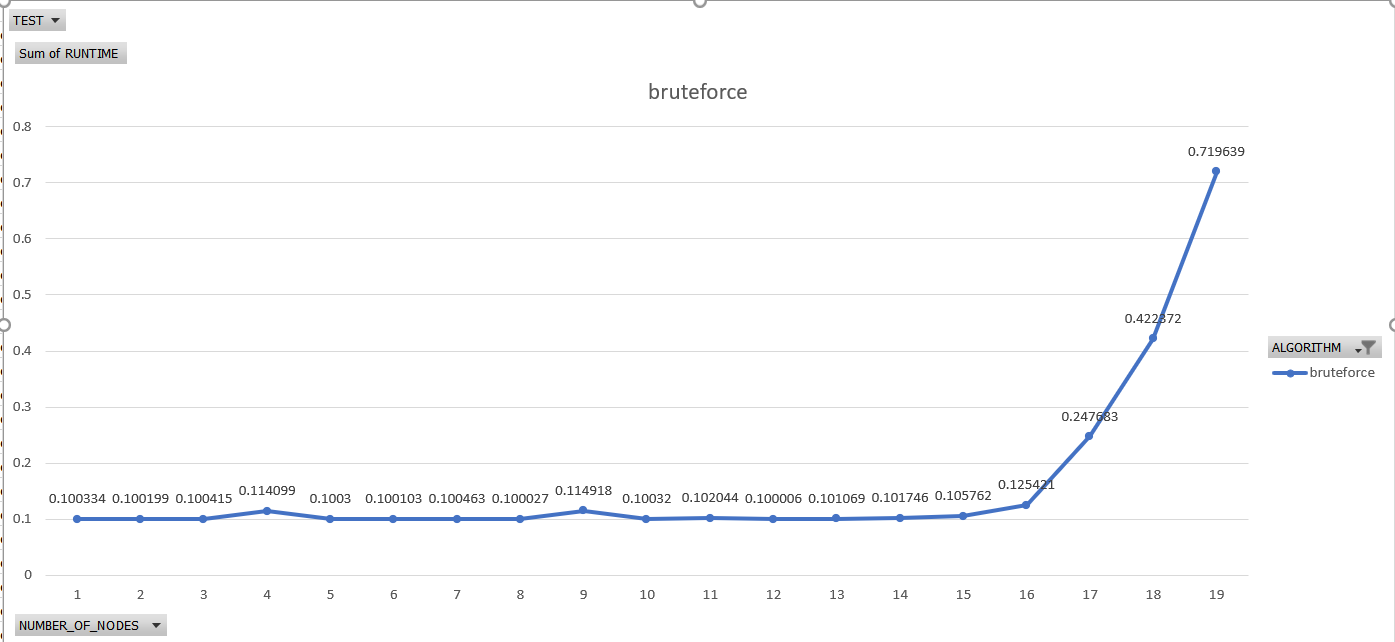
##### Half-cycles

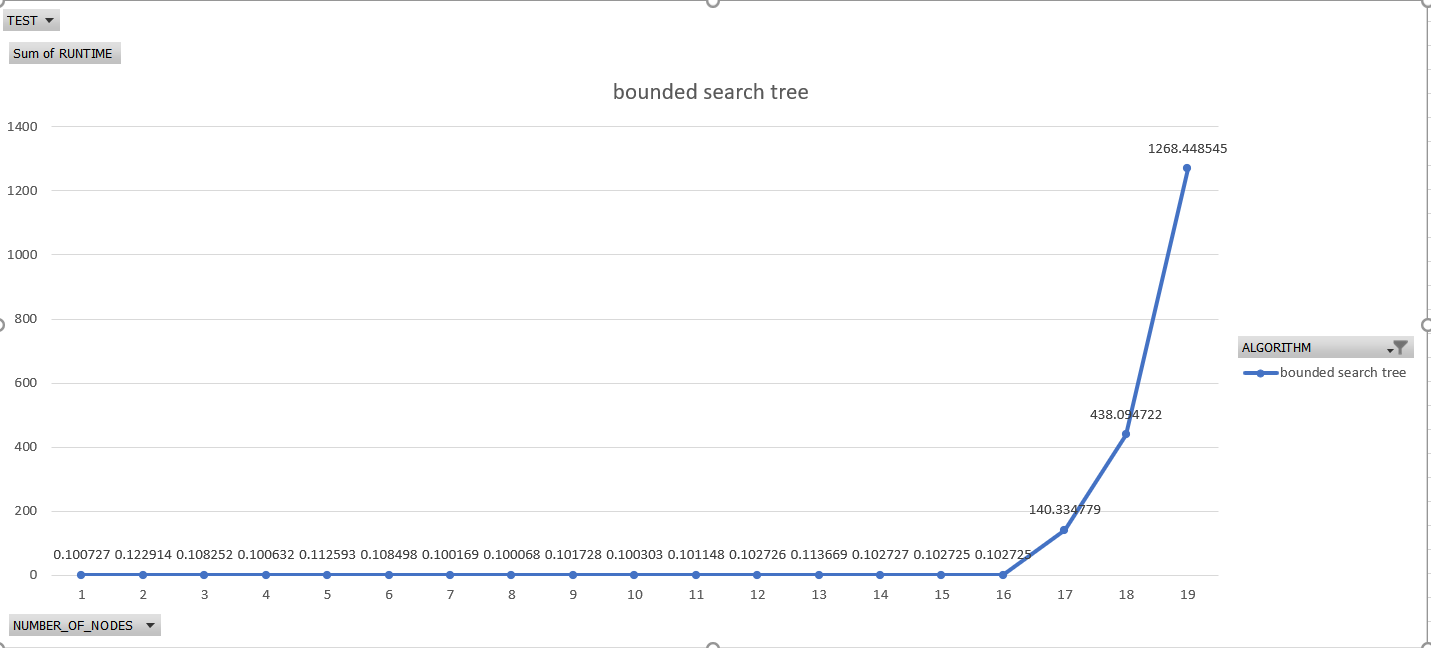


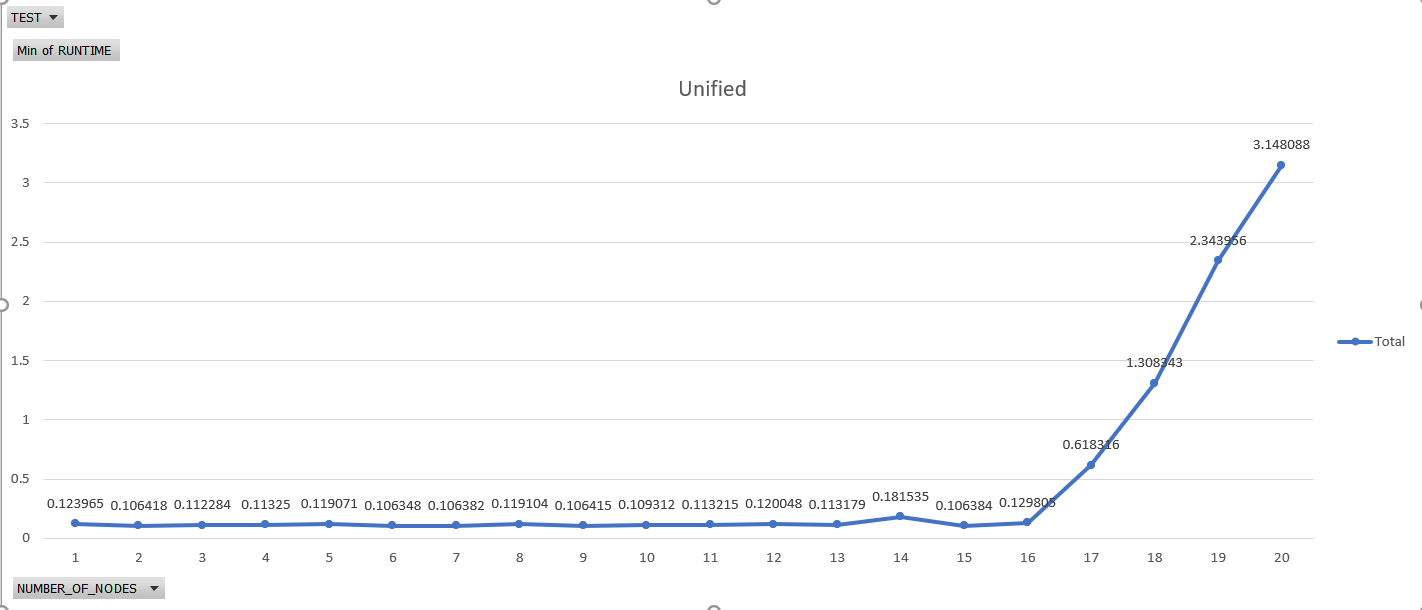


##### Third Cycles

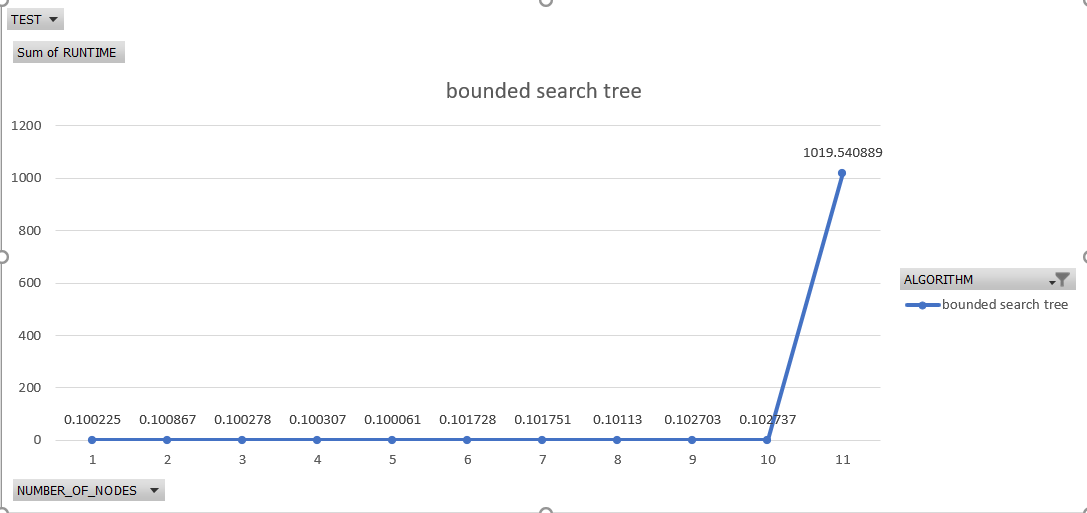


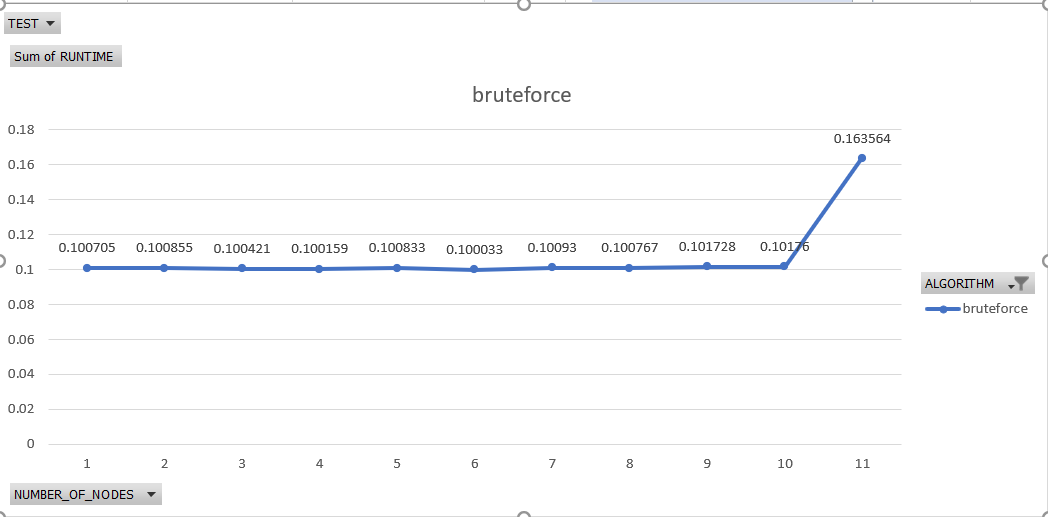


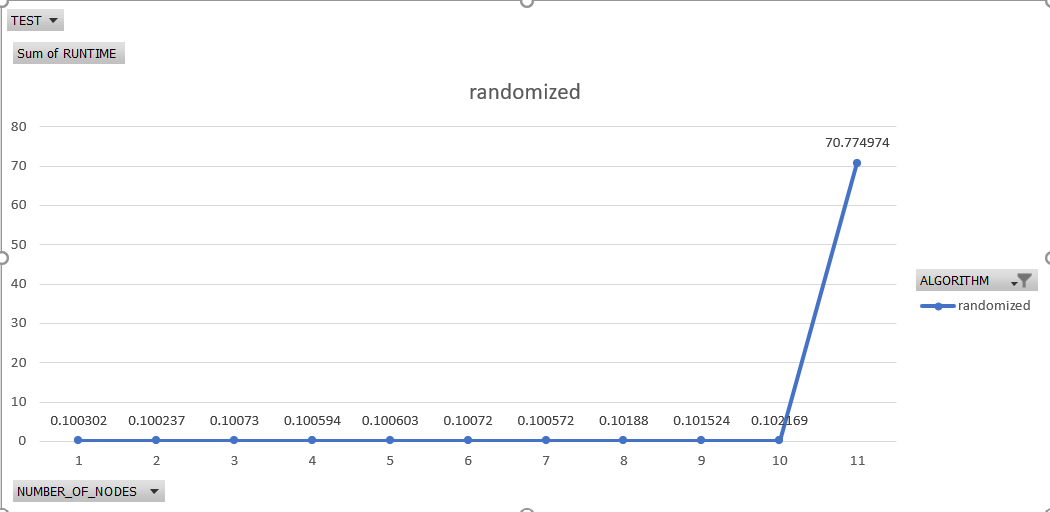


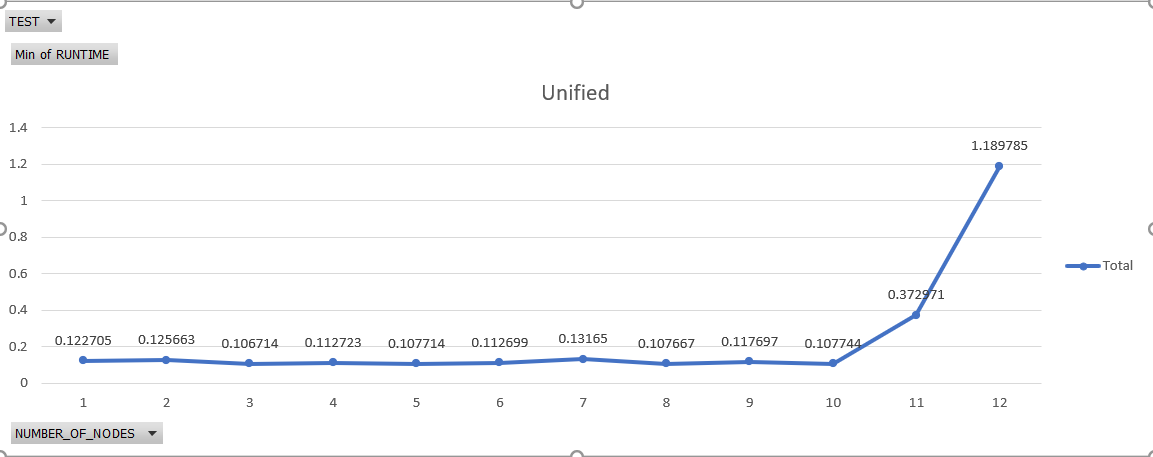


##### All cycles









## References

1. [parameterized-algorithms - Marek Cygan, Fedor V. Fomin, ukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michaª Pilipczuk and Saket Saurabh](http://parameterized-algorithms.mimuw.edu.pl/index.html)