POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATION

Institute of Computing Science

Report

BACKTRACKING ALGORITHMS

Adam Korba, 151962

Teacher

PhD. Grzegorz Pawlak

POZNAŃ 2022

Contents

Chapter 1

Introduction

This report covers topics of finding Eulerian circuits and Hamiltonian cycles in graphs. Many techniques were explored to achieve these tasks, including the backtracking algorithm. First of all, let me clarify some terms:

- Circuit - a sequence of vertices and edges where vertices may repeat but edges not.
- Eulerian circuit (EC) - a closed circuit - which means that it starts and ends at the same vertex
- Cycle - a sequence of vertices and edges where
- Hamiltonian cycle (CH) - a closed cycle - starts and ends in the same vertex

I used C++ as programming language to generate graphs, and find EC and CH in them. I used Python (in particular numpy library) to generate charts used in the analysis part.

Chapter 2

Exercises

2.1 Random graph generation

2.1.1 Graph containing eulerian circuit

A necessary and sufficient condition to have a graph with an EC is that all vertices have an even degree. Also, the graph has to be connected. I have taken a particular strategy to generate such graphs.

- Generate random hamiltonian cycle. (Resulting graph for sure is connected and all vertices have even degrees)
- Randomly find triplets of vertices such that none of them have an edge between each other. If you then create edges between them, for sure an even number of degrees will be added.
- repeat second step untill you get desired number of edges. If saturation of a graph is k it will be

$$|E| = |V| * (|V| - 1) * k/2$$

2.1.2 Graph containing Hamiltonian cycle

This one is similar but only the second part changes, now we don't need to find triplets of vertices without edges, it is enough to find pairs. This algorithm will still result in a connected graph with HC but is going to work much faster than the one generating eulerian graphs.

2.2 Exercise 1 - Euler circuit

2.2.1 Algorithm

To find and print the EC I choose Hierzholer's algorithm. It is a very fast and simple algorithm but it relies heavily on the speed that it takes to perform the following operations:

- finding neighborhoods of some vertices
- removing edges

I decided that the best graph representation for these operations is **Incidence List** because it allows for very fast retrieval of neighborhoods, and removing elements is as simple as removing them from the unsorted array. On the other hand, adjacency matrix would work well too, as it can remove edges in an instant but it is harder to find neighboring vertices and has bigger space complexity.

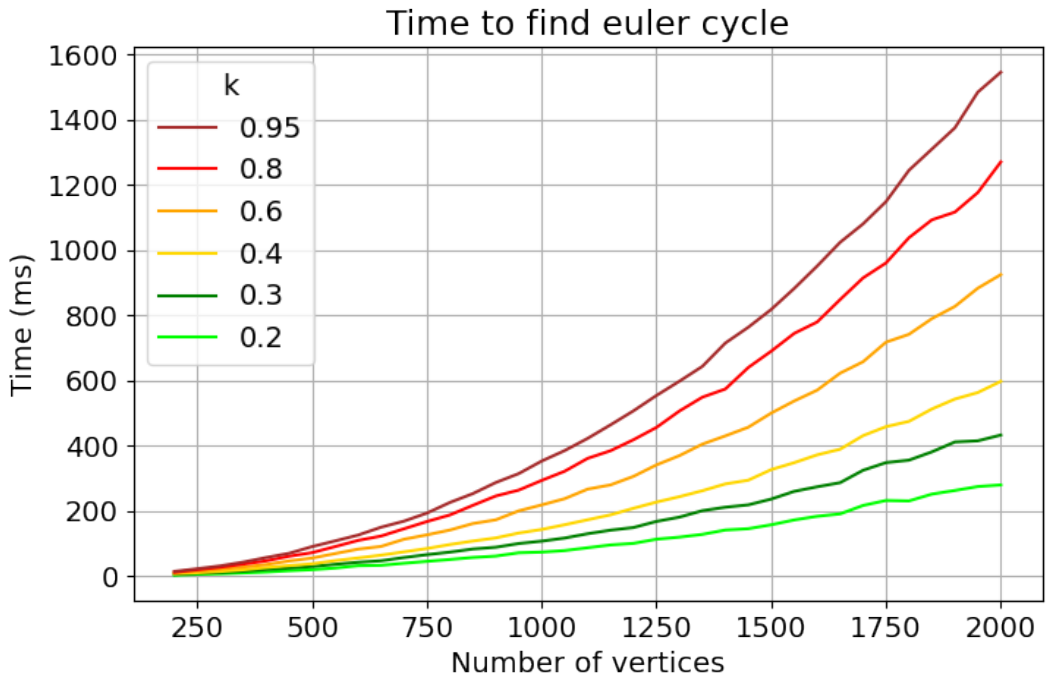

FIGURE 2.1: Results of experiment 1

2.2.2 Experiment

The experiment was run for different saturations of edges and numbers of elements. All graphs generated for tests contained an Euler circuit. Every data point was measured once but as you can see data is not very distorted, which means that randomness doesn't play a big role in the runtime of this algorithm. The algorithm depends mostly on a number of edges in a graph. It is harder to find an Euler circuit for larger saturations because they have more edges, and these circuits become longer. It is also interesting that results form some kind of a curve, the reason for that is the fact that on the x-axis we have a number of vertices, as you can see in the formula from section 2.1.1, a number of edges is approximately **square** of a number of vertices and multiplied by some constant k/2.

2.3 Exercise 2 - Hamiltonian cycle

2.3.1 Algorithm

To find Hamiltonian cycles I used a backtracking algorithm, it has some pros and cons. It is an exhaustive algorithm so it will definitely find a hamiltonian cycle in a graph that contains such a cycle, however, it may take extremely long time to find a cycle because of the algorithm's worst-case complexity which is $\mathcal{O}(n!)$. In this algorithm, I once again used Incidence List because it will be helpful to find neighbors quickly.

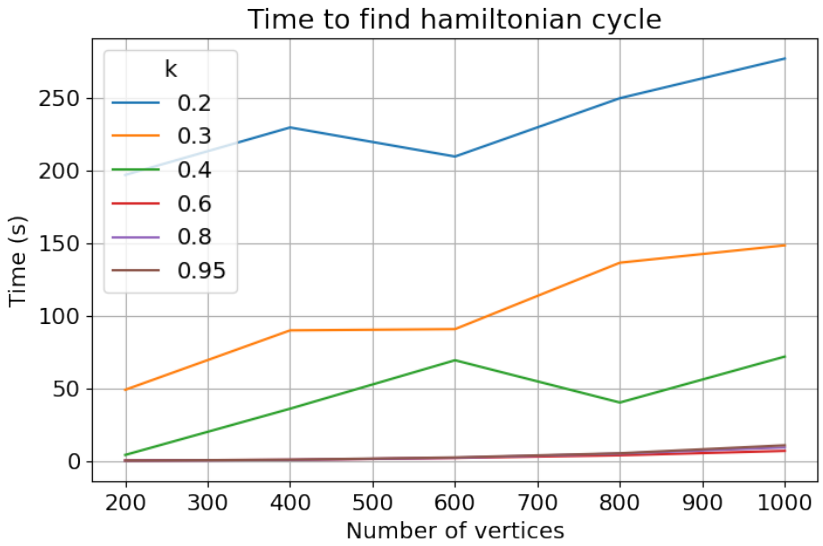

FIGURE 2.2: Results of experiment 2

2.3.2 Experiment

The algorithm works very slowly for certain graphs, that is why timeout was set on 300 seconds so the algorithm doesn't freeze on one particular test subject. I restricted myself to only 5 data points but I run algorithm 20 times with different graphs for every for all of them. Although it may not be clear at first algorithm execution time increases as the number of vertices increases. Another observation is that for smaller saturations algorithm worked longer. One of the reasons is that while generating a graph there is definitely **one** hamiltonian cycle, but if we add random edges there is a chance that we will accidentally create more cycles. So if there are more edges, thus more cycles algorithms finds one of them faster. In figure 2.3, there are only higher saturations plotted. As you can see the trend to increase as a number of edges increases is maintained. But the second one doesn't apply anymore. The algorithm worked the fastest for 0.6 then 0.8 and then 0.95.

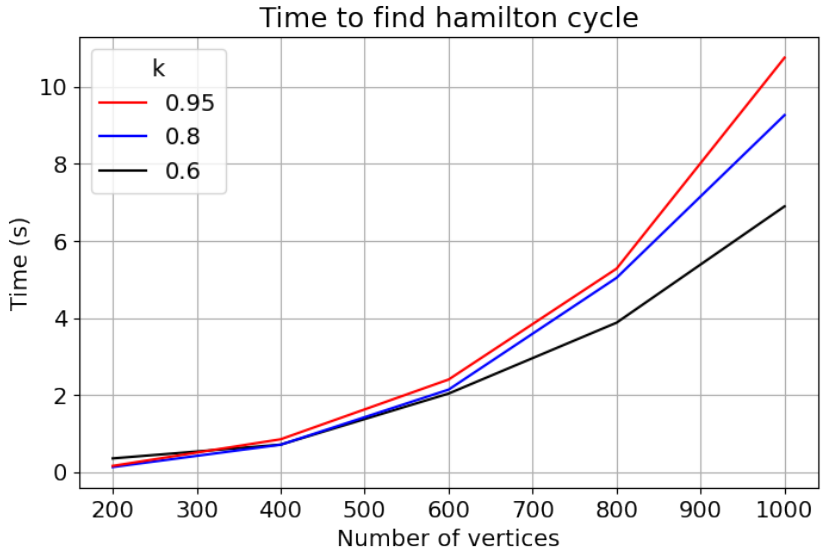

FIGURE 2.3: Experiment 2 zoom on bigger saturations

Chapter 3

Conclusions

Many real-world problems can be reduced to graph theory and finding Eulerian circuits and Hamiltonian cycles. That's why it is important to develop efficient algorithms. After performing experiments on these topics I came to certain conclusions.

It is easy to work with Euler circuits, starting with checking their existence, which is very easy: It just takes $|V|$ operations if we use the right graph representation. Finding such circuits is also not a hard problem, in fact, it belongs to **P** complexity class as it can be solved in polynomial time with a deterministic Turing Machine. The time complexity of Heirzholer's algorithm that solves it is $\mathcal{O}(|E|)$.

When it comes to finding Hamiltonian cycles it is not that easy. The problem of finding such cycles belongs to **NP-complete** class, so it is not possible to find solutions in polynomial time with machines we use today. It is easy to check our solutions. (We just iterate through the solution that our algorithm obtained and check if every vertex appears exactly once). The backtracking algorithm does its job (it solves the problem), but execution times vary a lot between different test subjects. This algorithm usually works well for high saturation (for reasons mentioned in 2.3.2, but there were certain cases where even for such graphs execution times reached 200 seconds, so it is very random. For lower saturation (like 0.2 or 0.4) this problem becomes even more present, so it is not recommended to run this algorithm for big graphs with such saturation.

Poznan University of Technology
Faculty of Computing and Telecommunication
Institute of Computing Science

Typeset using LaTeX.