Nikodem Seidel - 148269
Mateusz Małecki - 148265

1. Eulerian Circuit

In this exercise, we are using python generated connected graphs where every vertex has an even degree since only this approach makes sure that there exists at least one Eulerian circuit. Using completely random connected graphs, the chance of getting one with Eulerian circuit is extremely low for big databases and too much time would be needed to find a suitable graph.

For our algorithm, we chose neighborhood matrix representation for several reasons. Since we use undirected graphs and our algorithm deletes edge after "going through it" removal operation is the easiest and what's more important the fastest in the matrix representation. Its complexity is just O(1). Searching for a next neighbor is also really fast - its complexity is linear O(n), where n is the number of all nodes. The other benefit is the fact that operations on matrices are faster than operations on structures. The other representation that can compete with the NBH matrix is the list of incidents. This solution would be found more efficient in the case of undirected graphs because the complexity of operation of deleting used edges would be O(1), and finding neighbor in such representation would also have complexity O(1), but since we use undirected graphs, the complexity of removal process changes to O(n) where n is a number of nodes. Those two representations are pretty close in comparison, but by reason of faster operation on matrices than on custom structures, the NBH matrix occurs to perform better in our experiment. The only disadvantage of our approach is the waste of memory i.e. in places where connections are not existing, our structure keeps 4-byte size integer "0", instead of simply not keeping this information.

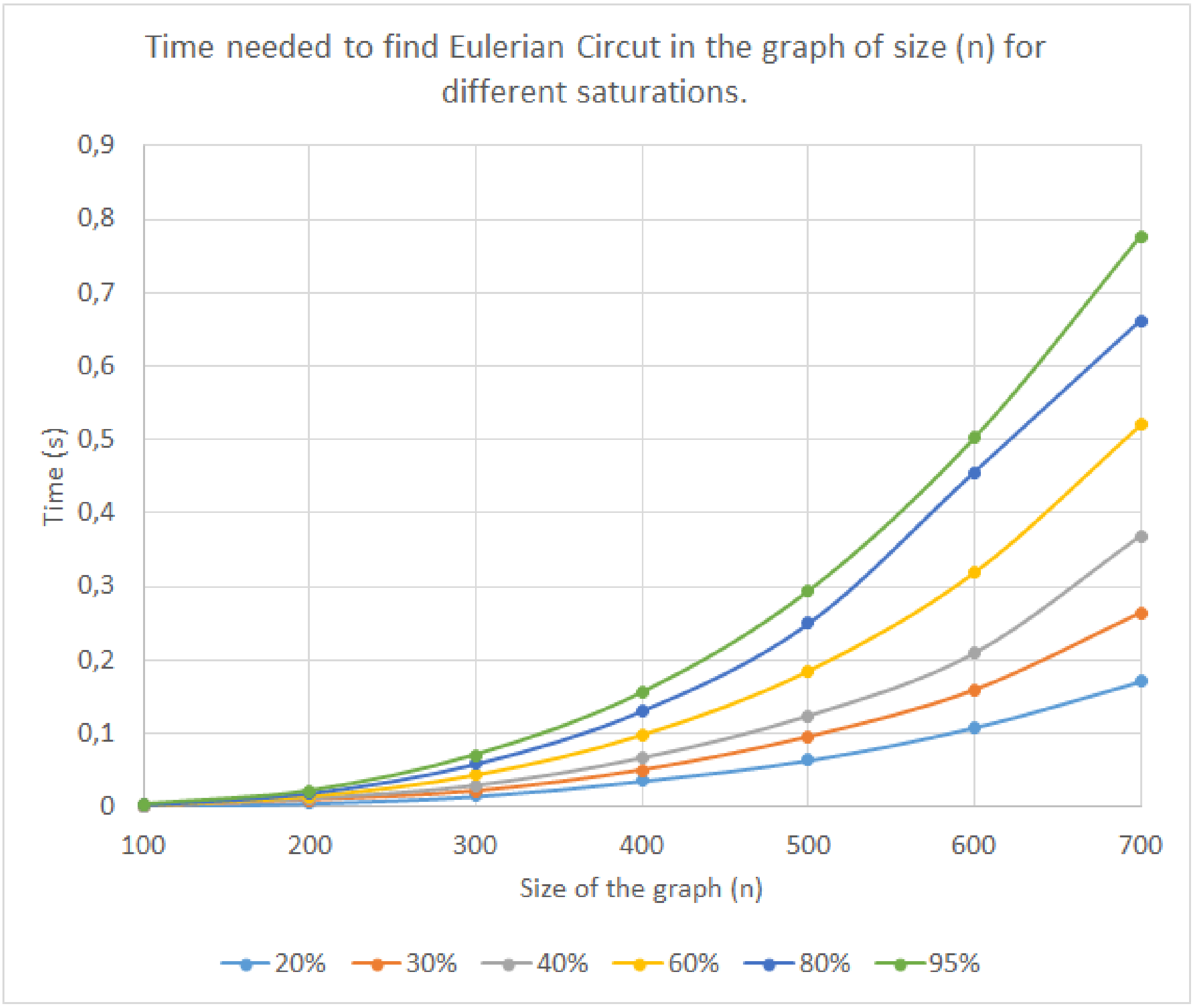
When it comes to the incident matrix, it would perform purely in finding connected nodes, in the deletion, and would waste even more memory.

The last is the edge list. This representation would have the worst possible complexity of finding connections of a given node, and deletion would highly depend on implementation.

To sum up, in our experiment with undirected graphs the bigger is the saturation of the graph, the more worthy it is to use the NBH matrix over the list of incidents, but if we would use directed graphs, the list of incidents would be unbeatable.

Bellow, we can see the results of our experiment:

Nikodem Seidel - 148269
Mateusz Małecki - 148265



The algorithm picks a random node, and checks the existence of its neighbors. If any neighbor exists (which is ensured by the completeness of the graph), the algorithm picks the first found one and goes to the given (by the edge connection) node. This operation has time complexity O(n), where n is equal to a number of nodes. Then erase the edge in graph representation (time complexity of the operation O(1)) and search for the next connection (O(n)). Such a cycle is repeated n*n times, which gives us whole time complexity equal to O(n^3).
Given chart clearly shows that the biggest difference in computation time is made by the change of graph saturation. This is exactly what we expected since our algorithm has the time complexity equal to O(e*n) where e is a number of edges. By reason of $e = \frac{n*(n-1)}{2}$ where n is the number of nodes, it is logical that our plots are parabolic-shaped. In conclusion, we can assume that the time complexity of the algorithm is equal to O(e * n) = O(n^3).

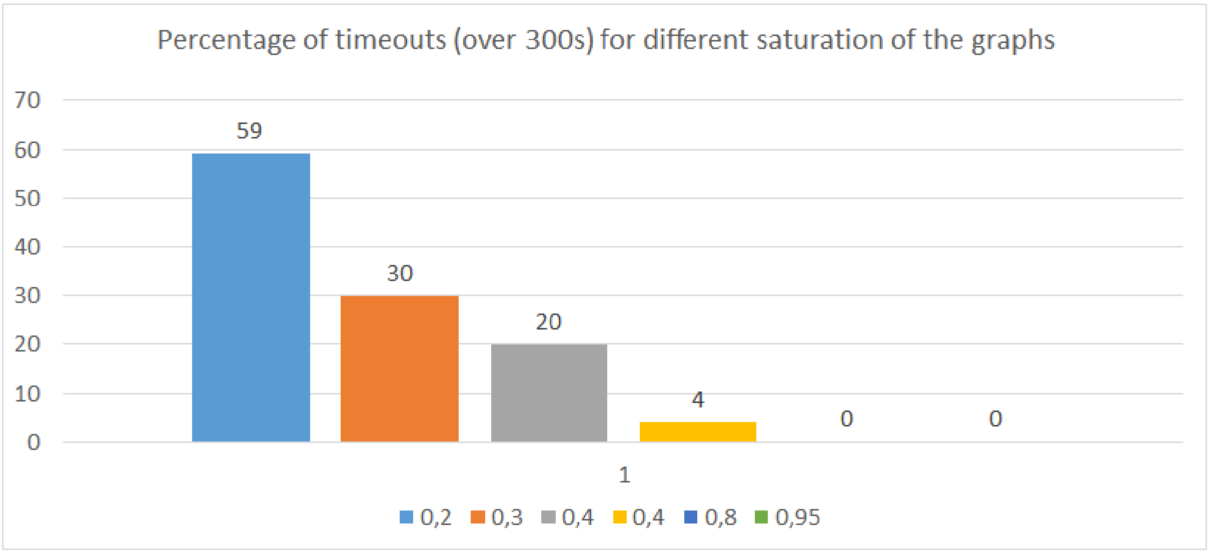
Nikodem Seidel - 148269
Mateusz Małecki - 148265

2. Hamiltonian Cycle

To provide all needed computation we’ve created 600 independent graphs - 6 different saturation levels * 10 measure points * 10 trials (to get more accurate results). As in the previous case data has been made in python language and chosen representation was Neighborhood Matrix. Partially for the same reasons as in exercise 1, but the main goal was to get information about the existence of the edge between two specific vertices as soon as possible.

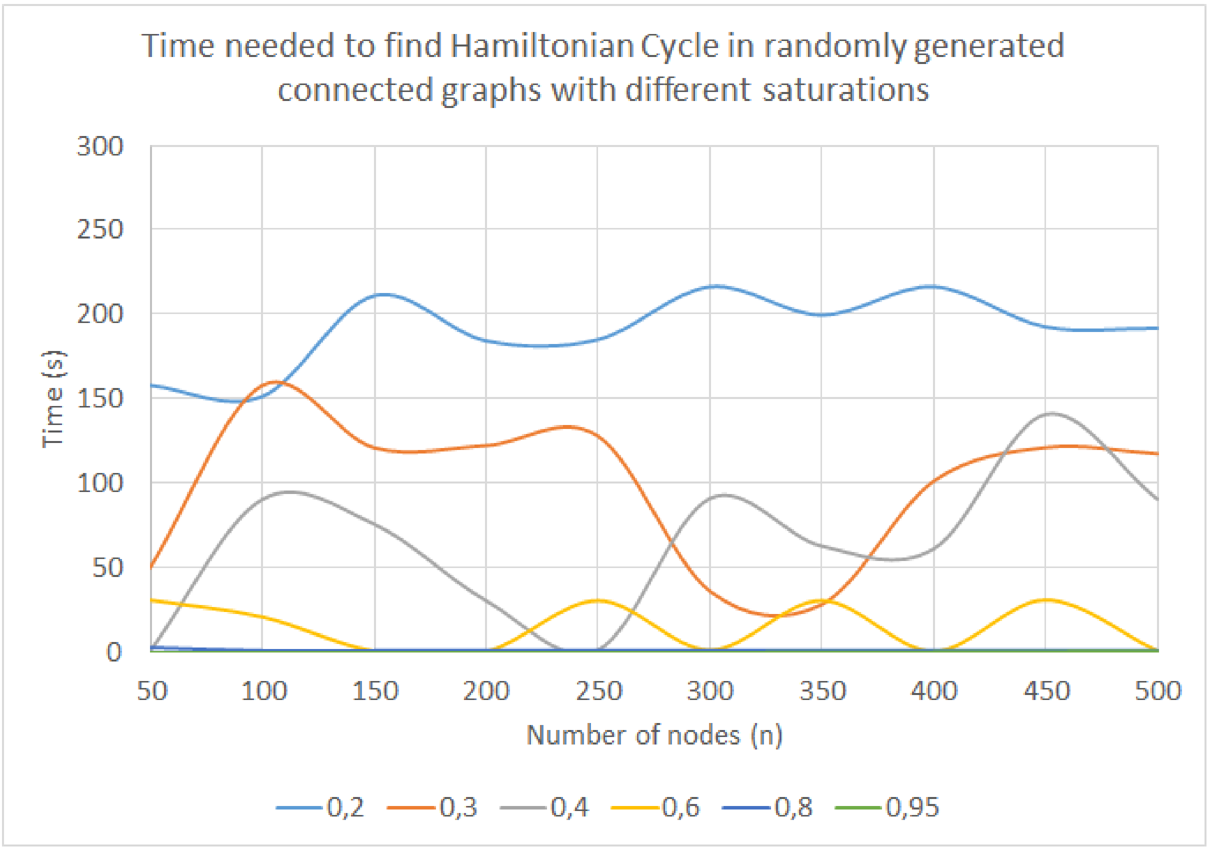
Hamiltonian Cycle is a problem that belongs to the NP-complete class. It means that there is not any known algorithm which has polynomial complexity. In the second exercise we have used a backtracking algorithm (complexity O(n!)). It checks nearly all possible permutations so we can say that it’s smart (because the algorithm omits every case which doesn’t hold some conditions) brute force approach. Because of the complexity (for example if a graph has 50 vertices algorithm can check even 50! cases (3*10e64), so after 5 minutes of searching algorithm stops and goes to the next graph.

The graph below represents how many times the computer didn’t find Hamiltonian Cycle in the time and skipped calculations.



We can clearly see that bigger saturation of the graph causes it to be much easier to find CH in the time limit. It is expected because greater number of edges in the graph increases the chance of existence of CH.

Nikodem Seidel - 148269
Mateusz Małecki - 148265



Unfortunately despite the long time of computation (about 12h) we have used too few trials (10) and the chart doesn't represent the expected exponential curves. The only observation which is visible here confirms that a smaller saturation level increases computation time. Also the computer has found a cycle for cases with saturation = 0.2 only when the graph had lucky arrangement.

Summary of the time and space complexity of considered algorithms.

Problem	time complexity	space complexity
Eulerian Circuit	$O(V^2) = O(E)$	$O(V^2) = O(E)$
Hamiltonian Cycle	$O(V!)$	$O(V)$

V - number of vertices
E - number of edges