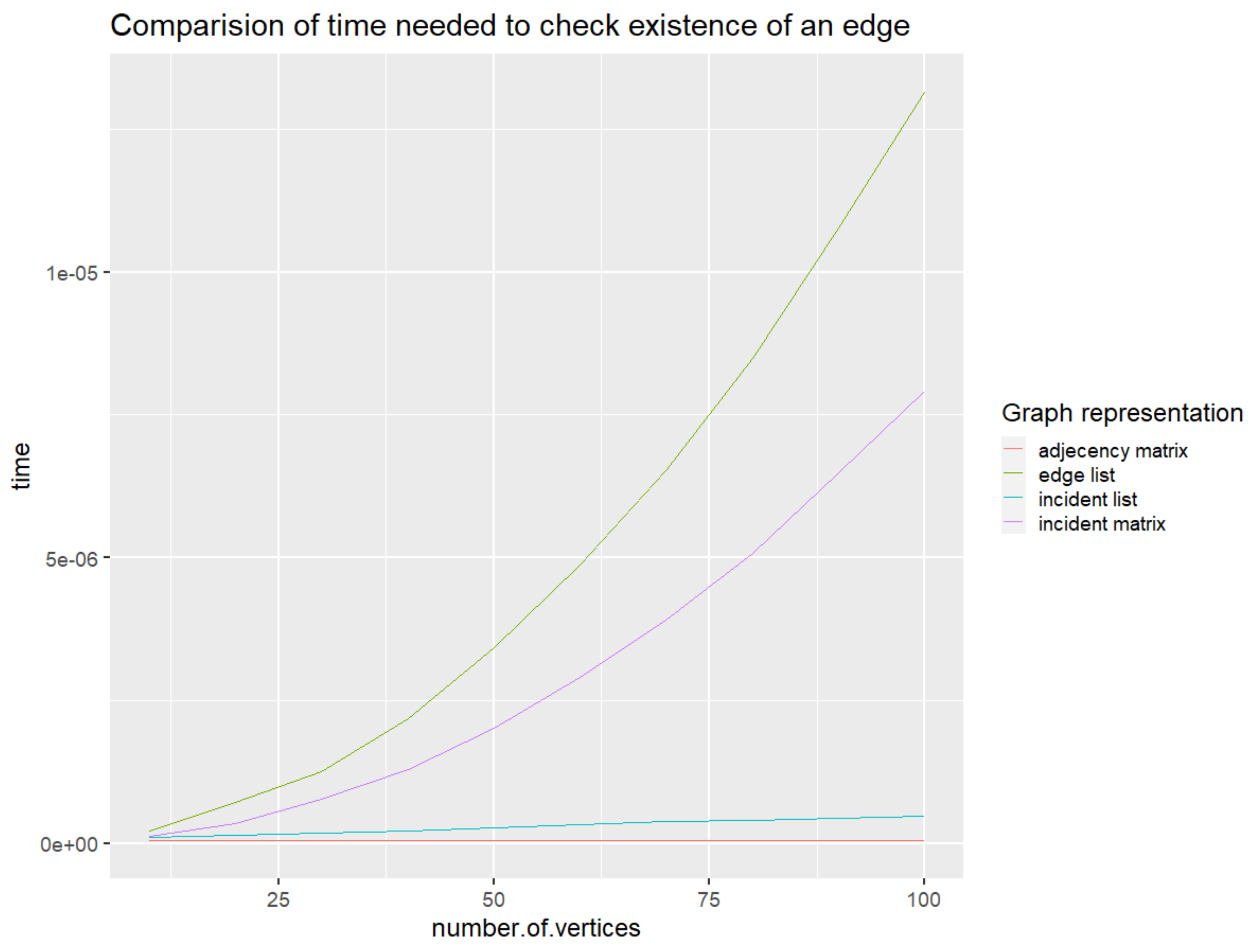
Report 3 - Graph Algorithms
Bogna Kilanowska 148252

The time of obtaining information about the existence of edge between a pair of random vertices in different graph representations

Data:
The average time of checking the existence of the edge in the following graph representations was measured for random generated graphs consisting of 10, 20, …, 100 vertices. The saturation of the graphs was 0,6.

Chart:



Conclusions:
Complexity of checking the existence of an edge (a,b) in the graph represented by an adjacency (vertex) matrix is constant and equals O(1). It is because, we check if arr[a][b] == 1.

This procedure lasts a bit longer for an incident list. For this graph representation, we must check if in row a exists b. In this case the time complexity of checking the existence of an edge equals O(n), where n - number of vertices.

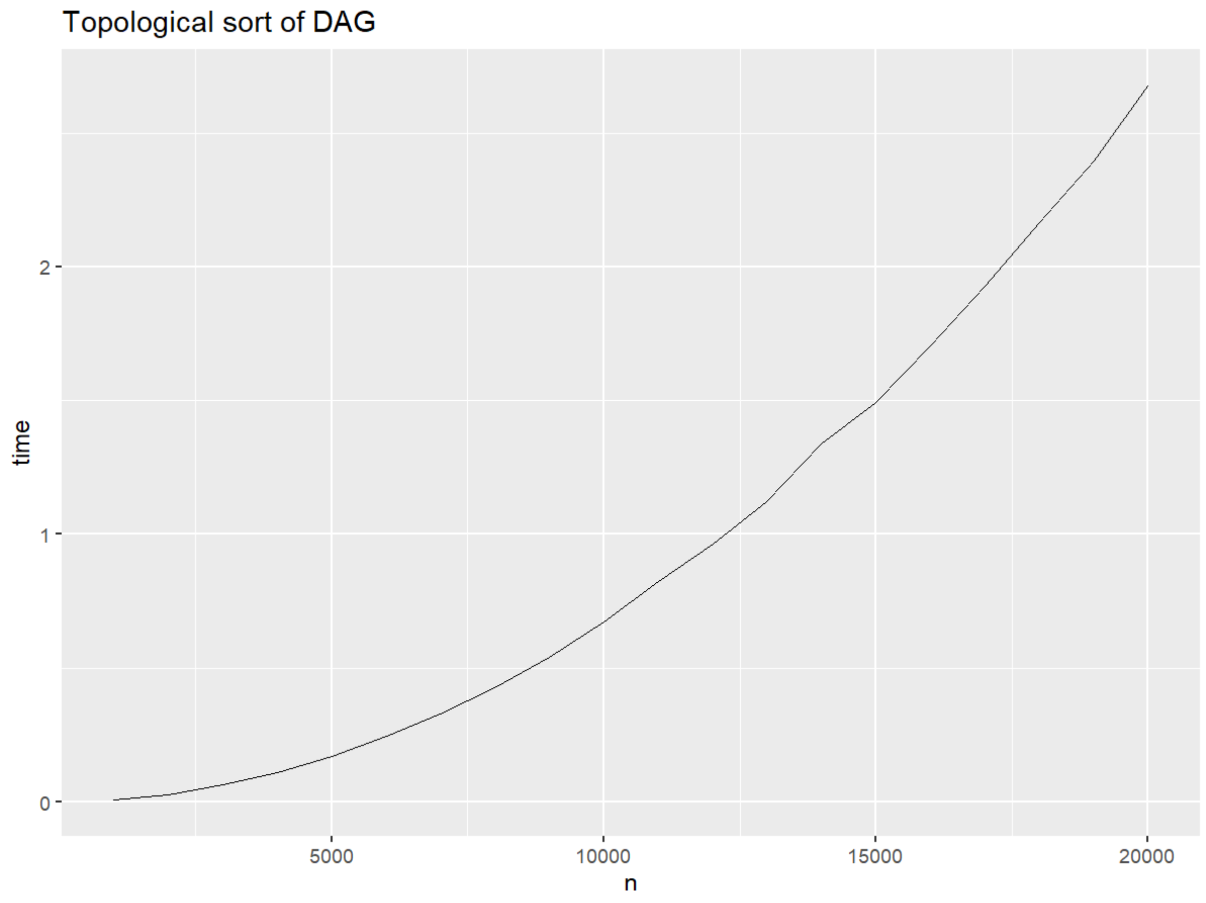
To check in the incident matrix if an edge exists, we must check if in row a and row b exists a column, for which in each of this row, the value will be 1. So the time complexity of checking the existence of an edge depends on the number of edges and equals O(e).

The worst time complexity was obtained for an edge list. For this graph representation we also must check each edge. The time complexity of checking the existence of an edge in the graph depends on the number of edges.

Time of sorting a Directed Acyclic Graph using Topological Sort algorithm

<u>Data:</u>
The Topological Sort algorithm was applied to DAGs represented by an Incident List. Number of vertices n = 1000, 2000, …, 20000, saturation of the graph = 0,3.

<u>Chart:</u>



Comparison of the Incident List representation to others representations:

	Incident List	Vertex Matrix	Edge List	Edge matrix
Storage	O(n + e)	O(n^2)	O(e)	O(n * e)
Edge exists check	O(n)	O(1)	O(e)	O(e)
Getting a list of adjacent vertices	O(1)	O(n)	O(e)	O(e)

n - number of vertices e - number of edges

Justification of the choice of the representation:
To sort the graph topologically, the algorithm needs access to the list of all adjacent vertices to the current vertex. Getting this information from the List of Incidents is very fast, it is just needed to check the proper row. In comparison, getting the list of neighbourhood vertices from the Vertex Matrix costs n operations, for Edge List it might be e operations, just like for an Edge Matrix. Moreover, the Incident List requires less space in memory than Vertex Matrix or Edge Matrix, which allowed me to perform experiments on bigger graphs (up to 20000 vertices).

Conclusions:
Performing the topological sort algorithm on the Incident List representation is pretty efficient in my opinion. The algorithm saves time at each iteration, when it demands access to the adjacent vertices list. Finally the time complexity of the algorithm totals O(n + e).

Conclusions

Each of the graph representations has its pros and cons. The main advantage of using the Vertex/Adjacency Matrix representation is a quick check of the existence of the edge. On the other hand, Vertex Matrix occupies a lot of space in memory.

Incident List takes less storage, but checking the existence of the edge takes a bit longer than for the Adjacency Matrix. Easy access to the list of vertex's neighbours makes this representation the best choice if we want to apply the Topological Ordering algorithm.

Operations performed on Edge List and Edge Matrix take much longer. However, both of them have one big advantage: in these representations we also store the information about the names of the edges. This property might be very helpful if for example we need to represent a map of the city with names of all the streets.