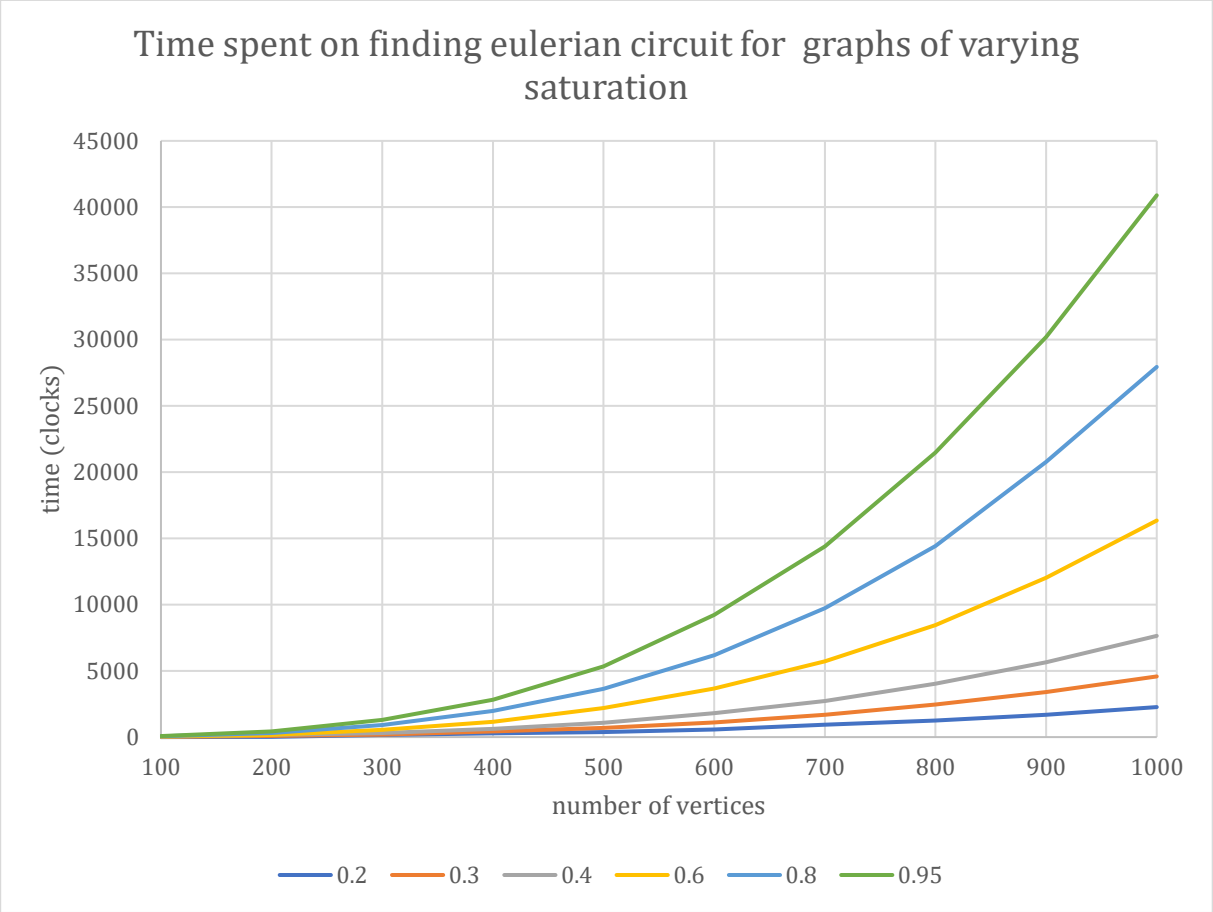
I Eulerian Cycle

An Eulerian circuit or an Eulerian tour is a trail which starts and ends at the same vertex, visiting each edge of the graph exactly once. In order for an undirected graph to be Eulerian, degrees of all vertices have to be even – for this to happen in a randomly generated connected graph is highly unlikely – so in order to find an Eulerian circuit and draw conclusions about the complexity of the circuit-finding algorithm, one has to be sure that the graph satisfies the given requirements (even degree of vertices and connectivity).

Generating the afore mentioned graph is done in the following way: first a path made up of edges from the i-th to the (i+1)-th vertex is established (0-1;1-2; … ;(V-2) – (V -1) (V = number of vertices)) – this first step ensures that the graph is connected. Next the vertices are divided into 2 subsets – of even and odd degrees. Initially the set of even-degreed vertices contains all of the bridges, while the set of odd-degreed vertices (the size of which is perpetually two) consists of the endpoints of the established Eulerian trail. For the algorithm to bring forth a graph with a given saturation – one has to increase the number of edges up to the point where the given threshold is reached. This is done by randomly choosing one vertex from each of the subsets, checking if there already exists an edge – if not – a direct connection is established and the previously odd-degreed vertex is now of even degree while the opposite is true for the even-degreed vertex. After the goal saturation is reached – the remaining two vertices from the odd subset are connected which transforms the Eulerian path into an Eulerian circuit.



Line chart 1

On the subject of the actual circuit-finding algorithm: after having generated a graph which satisfies the essential conditions it is redundant to check for the degree of vertices – because this is a given. Firstly one selects a vertex and simply chooses an edge which leads to a different vertex, subsequently deleting the given("used") edge from the structure storing the information (in the case of this project – the adjacency list – this will be elaborated on below), this approach decreases the given structure both edge- and vertex- wise, - vertices with all edges already explored don't get visited anymore, which leads to the structure being empty after the circuits are completed (subcircuits are inspected one by one, since after having complete one tour, another might still be present – but starting from a different vertex). Since all of the edges have to be visited(and fed into the adjacency list), this algorithm might seem to be of complexity O(V^2) since $E = \frac{V*(V-1)}{2} * saturation$, but one has to take into account the time it takes to delete vertices from the graph representation which is O(V) – but on average it's V/2 since the number of vertices decreases during the runtime, so the complexity is O(V^3), so the problem can be solved in polynomial time it's in class P.

It is also worth noting that one would expect the time consumptions for graphs for given saturations to be proportional – just like the number of edges, but looking at Line Chart 1, one clearly notices that this is not the case – since there are $E = \frac{V*(V-1)}{2} * saturation$ edges.

Graph representation

The representation of choice for the first task was the adjacency list, with many factors contributing to this decision: a key advantage to this structure is the possibility of efficient iteration through the edges connected to a particular vertex which is not guaranteed to the same extent by alternative representations – this is especially important since the algorithm basically finds an Eulerian circuit by traversing it. However, the quick iteration through edges comes with a drawback – removing an element from a list (list of edges connecting to a vertex is of complexity O(V-1) = O(V), V being the maximum number of vertices that are connected to a given node(vertex)), this is not the case for example in an adjacency matrix, where it would be O(1) (However, the adjacency matrix has another problem – finding the next neighbor to visit, which is linear, so the algorithm would still be O(V^3)). On the plus side – the adjacency list takes up less memory when representing sparse graphs – in an AM the space complexity is always V^2, while in the adjacency list it's O(E) which is technically O(V^2) – but as previously stated, in sparse graphs θ << O.

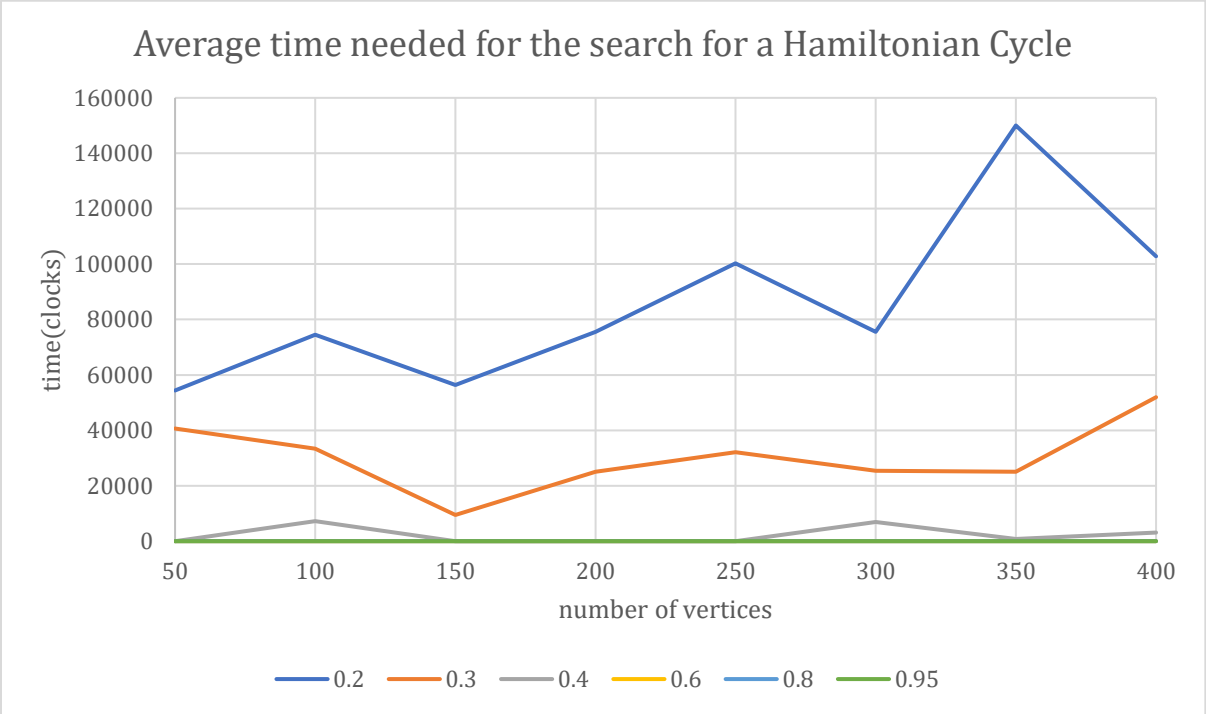
II Hamiltonian cycle

II a) Connected graph generation

In order to find a Hamiltonian circuit (a circuit in which every vertex is visited once with no repeats) one has to make sure that the graph is connected – else the notion of a cycle in such a structure is illogical, however in this task the connectivity of the graph is a given. The algorithm responsible for producing such a graph with a specified saturation performs in the following way: first a basic connected graph is generated: each vertex gets connected by an edge to a vertex with a lower index – in this way (1 is always connected to 0, 2 can be connected to 0 or 1, 3 to 0,1 or 2) ensuring that no vertex or a group of vertices form a disconnected subgraph. After the basic connected backbone gets generated – random edges get added to the graph up until a given saturation is reached.

II b) Finding Hamiltonian cycle

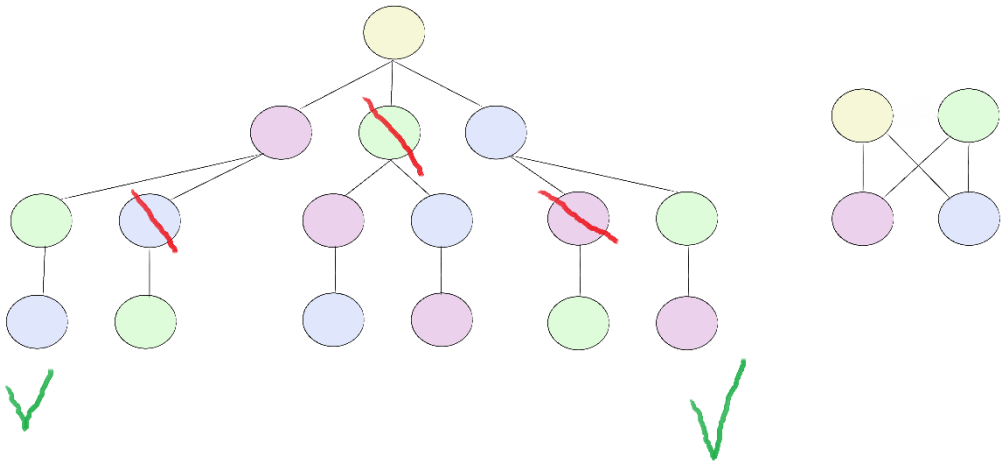
In order to find a Hamiltonian circuit, the algorithm traverses the graph, building the solution (circuit) one vertex at a time, by iterating through the adjacent vertices of a given vertex in a recursive manner – the maximum number of recursive function calls in a row is equal to V – 1. The backtracking algorithm tries to go over unvisited neighbors of vertices – if there are no left – that is the possibilities have been exhausted, the algorithm moves back to the previous vertex and keeps iterating through the other unvisited vertices adjacent to the given one – this procedure is at the core of the said algorithm, bearing resemblance to DFS. At the end – when a Hamiltonian path is found, it is checked if the last vertex is connected to the first one – if not – backtracking is performed (going one vertex back).



Line chart 2

One can see that for graphs with lower edge saturation the search takes up notably much more time – oftentimes the algorithm fails to discover a solution in the given timeframe – that is because a lot of backtracking has to be conducted due to the sparseness of the graph. The same is not true for graphs of higher edge density – judging by the search time little backtracking is needed, owing to the fact that vertices are closely interconnected, which leads to a higher probability of finding a circuit. This claim is supported by the results from the graphs of density 0.3 and 0.4 – in which less and less time is needed to conduct a search. The Hamiltonian Circuit algorithm is of time complexity O(n!) (specifically, O((n-1)!) – each vertex cannot have more neighbors than V-1, and in each subsequent step no vertex can be repeated which leaves V – number_of_visited_vertices – 1 possible edges that can be used to get to a new vertex. This is represented by the picture below – there are (n-1)! Leaves (possible solutions), but many branches get cut off early in the process which can (especially when the graph has a high edge saturation – lead to discovering a circuit

early on, just like in the picture: there are 2 circuits, but the first circuit was discovered without backtracking).



Picture 1 – (4-1)! "possible routes"