Sabancı University

Fall 2019

CS301 - Algorithms

Group 4 - Project Report

Hamiltonian Path

Yiğit Tekinalp, Ege Toker Dinç, Ataberk Özbay, Ceren Anıl, Uğurcan Açıkalın

PROBLEM DESCRIPTION

Hamiltonian Path in undirected graphs is a graph that have a path which passes through every vertex exactly one time.

An n-node undirected graph G(V,E) where V is the set of vertices V = $\{v_1, v_2, \ldots, v_m\}$ and E is the set of edges.

$\{v_1, v_2, \ldots, v_m\} = \{1, 2, \ldots, m\}$ and $\{(v_1, v_2),(v_2, v_3), \ldots ,(v_{m-1}, v_m)\} \subseteq E$.

We can see the Hamiltonian Path problem in everyday life, especially where optimization problems are encountered. Such as GPS or Google Maps, finding the optimum root from one destination to another. Hamiltonian Path represents the efficiency of including every vertex every root.[1]

In this Project, Hamiltonian Path will be examined, therefore our problem is expressed as follows.

On an n-node undirected graph G(V,E) with node set V and edge set E, is there a simple path of edge in G that contains every node in V, and thus contains exactly n-1 edge?

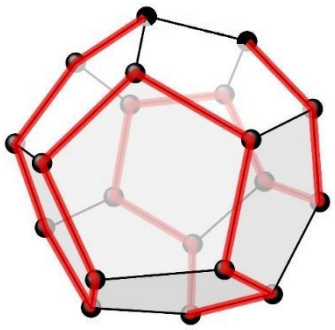

Fig. 1. An example of Hamiltonian Path

[1] Kaundal, K. (2017). Applications of Graph Theory in Everyday Life and Technology. Imperial Journal of Interdisciplinary Research, 3(3), 892–894. Retrieved from https://pdfs.semanticscholar.org/7aee/9da1025ea81e2fcfccaa53393b9b9ab76d05.pdf

In order to prove that Hamiltonian Path is NP-complete, first we have to prove the following.

1)The problem itself is in NP.

2) The problem can be reducible in polynomial time to a NP-complete problem.

1) In the given graph G, we can solve Hamiltonian Path by non-deterministically selecting edges from G that are to be included in the path. Then we traverse the path and make sure that we visit each vertex exactly once. This can be done in polynomial time. So, Hamiltonian Path problem is in NP class.[2]

2) To prove that Hamiltonian Path is NP-complete, we will make a reduction from Hamiltonian Cycle to Hamiltonian Path in polynomial time.

Given a graph G = (V, E) we construct a graph G' such that G contains a Hamiltonian cycle if and only if G' contains a Hamiltonian path. This is done by choosing an arbitrary vertex u in G and adding a copy, u', of it together with all its edges. Then add vertices v and v' to the graph and connect v with u and v' with u';

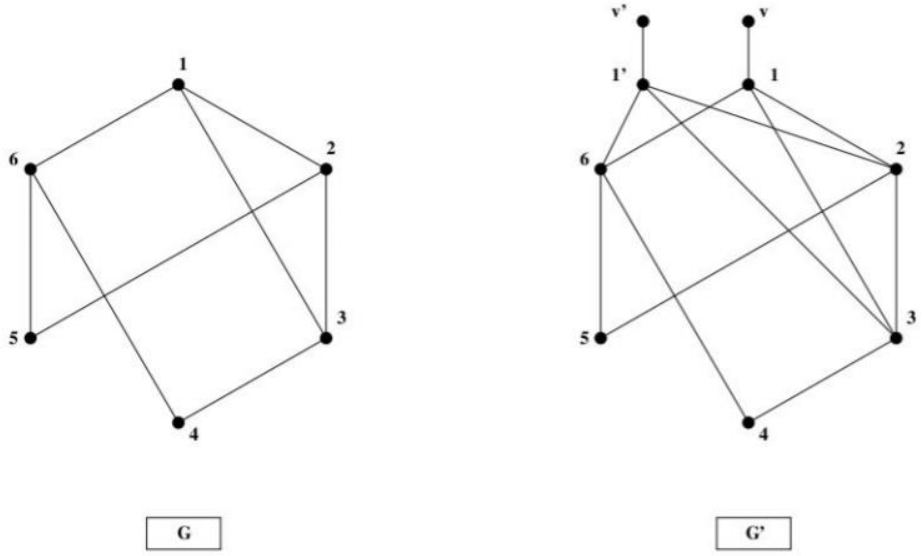

Fig. 2. A graph G and the Hamiltonian Path reduced graph G'

[2] Karlander, J. (n.d.). Retrieved from http://www.csc.kth.se/utbildning/kth/kurser/DD2354/algokomp10/Ovningar/Exercise6_Sol.pdf

Suppose first that G contains a Hamiltonian cycle. Then we get a Hamiltonian path in G' if we start in v, follow the cycle that we got from G back to u' instead of u and finally end in v'. For example, consider the left graph, G, in Figure 1 which contains the Hamiltonian cycle 1, 2, 5, 6, 4, 3, 1. In G' this corresponds to the path v, 1, 2, 5, 6, 4, 3, 1', v'. Conversely, suppose G' contains a Hamiltonian path. In that case, the path must necessarily have endpoints in v and v'. This path can be transformed to a cycle in G. Namely, if we disregard v and v', the path must have endpoints in u and u' and if we remove u' we get a cycle in G if we close the path back to u instead of u'. The construction won't work when G is a single edge, so this has to be taken care of as a special case. Hence, we have shown that G contains a Hamiltonian cycle if and only if G' contains a Hamiltonian path, which concludes the proof that **Hamiltonian Path is NP-complete.**[3]

ALGORITHM DESCRIPTION

Since, Hamiltonian Path is a NP-Complete problem, there is not an exact algorithm that solves Hamiltonian Path problem in polynomial time.

There is a solution that solves Hamiltonian Cycle problem in NP time by Sharon Christine. We use this algorithm as a base and modify it with greedy approach in our $O(n^3)$ heuristic Hamiltonian Path solution.

In this project, we will cover and analyze the performance of the greedy search algorithm for Hamiltonian Path.

The algorithm does the following step by step:

1. Firstly, it takes the first node as the starting point.
2. Then, it checks for all remaining vertices connected to this node.
3. If it is connected, add that to the path.
4. Then, it checks for the next node by repeating Step 2 and Step3.

ALGORITHM ANALYSIS

The algorithm we based on finds a Hamiltonian Cycle if there exist. We modified our code as finding a Hamiltonian Path. We compare our result with the DFS and backtracking algorithm which is O(N!) and analyze the success rate in this way in the experimental analysis part of this report.

[3] Karlander, J. (n.d.).

We calculate the worst-case running time as:

```
57 void displayCycle() {
58    cout<<"Path: ";
59
60    for (int i = 0; i < NODE; i++)
61       cout << path[i] << " ";
62    cout << endl;
63   // cout << path[0] << endl;      //print the first vertex again
64 }
65
66 bool isValid(int v, int k) {
67    if (graph [path[k-1]][v] == 0)   //if there is no edge
68       return false;
69
70    for (int i = 0; i < k; i++)    //if ... already taken, skip that
71       if (path[i] == v)
72          return false;
73    return true;
74 }
75
76 bool cycleFound(int k) {
77    if (k == NODE) {              //when all vertices are in the path
78          return true;
79    }
80    for (int v = 1; v < NODE; v++) {       //for all vertices except starting point
81       if (isValid(v,k)) {                 //if possible to add v in the path
82          path[k] = v;
83          break;
84       }
85       path[k] = -1;               //when k vertex will not ... the solution
86    }
87    if(path[k] == -1)
88       return false;
89
90    if (cycleFound (k+1) == true)
91       return true;
92
93    return false;
```

O(n)

O(n)

$O(n^2)$

O(n)

$O(n^3)$

```
bool hamiltonianCycle() {
   for (int i = 0; i < NODE; i++)
      path[i] = -1;
   path[0] = 0; //first vertex as 0

   if ( cycleFound(1) == false ) { // if there
      cout << "Solution does not exist"<<endl;
      return false;
   }

   displayCycle();
   return true;
}
```

O(n)

Total running time: $O(n) + O(n) + O(n^3) + O(n) = O(n^3)$

EXPERIMENTAL ANALYSIS

For experimental analysis, we randomly generated graphs that has 5,10,15,20,25,40 vertices. Our algorithm is considered successful if it correctly identifies whether the graph has a Hamiltonian Path or not, and it is considered a failure if it true rejects a graph with a HP.

There is no chance for a false accept, though, because the length of a Hamiltonian Path must be equal to the number of nodes. If the algorithm finds such a path, then it means it already succeeded. If not, a simple check suffices to avoid false accepts.

Success Rates

Number of random graphs	5 Vertices	10 Vertices	15 Vertices	20 Vertices	25 Vertices	40 Vertices
50	0.52	0.46	0.22	0.32	0.18	0.26
100	0.63	0.44	0.26	0.28	0.27	0.28
150	0.68	0.39	0.26	0.26	0.28	0.29
200	0.69	0.39	0.24	0.26	0.28	0.31
250	0.67	0.38	0.28	0.27	0.28	0.32
500	0.66	0.33	0.3	0.3	0.28	0.32
1000	0.67	0.33	0.3	0.3	0.29	0.33

Fig.3. Success rate table of randomly created graphs with different vertices

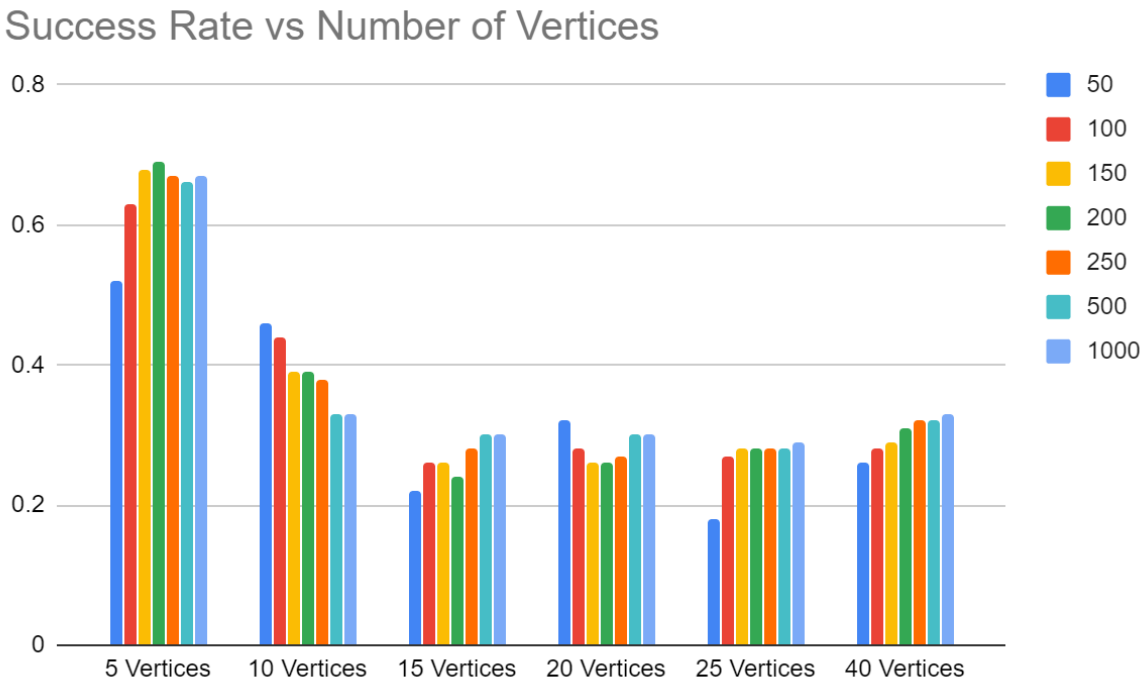

Fig.5. Success rate vs number of vertices graph

To generate random graphs, we used the method of repeatedly adding a new node and connecting that node to a certain number of old nodes. We used a probability of 50% to determine if we will connect the new node to a given old node for all nodes. This way, we managed to keep the edge density of our graph roughly constant. We never changed that %50 probability, because if we compare our results with more connected or less connected graphs, there would be significantly more or less Hamiltonian paths in the graphs, and that would destabilize and change the solutions given by our algorithm.

An intuitive explanation with limits can show us that, according to our graph generating algorithms' working method if we have an infinite number of nodes, the probability of having at least one node with no (or relatively really close to no) connection to the other nodes increases, which means, when randomly generating a graph, the probability of having a Hamiltonian path in it gets lowered and lowered. However, our algorithm never fails if the given graph has no Hamiltonian path, It can always say there is no Hamiltonian path if there is none. Therefore, when the number of graphs with no Hamiltonian paths increase, our algorithms success rate is only determined by the cases with the graphs with Hamiltonian Paths. When the number of vertices increase, this determining factor will start to get less significant, and will cause the success rate to approximately converge to a ratio.

In the above graph, we can see that success rate first decreases, then stabilizes at around 0.3. We reckon that this is caused by the situation explained above.

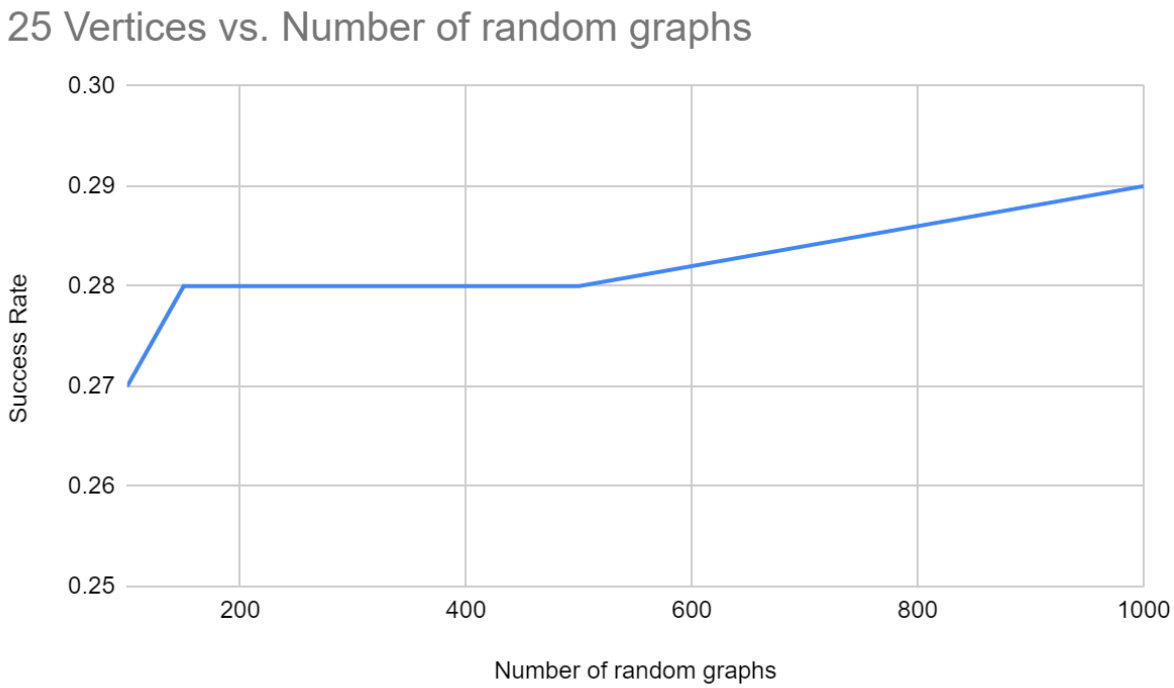

Fig.4. Success rate of 25 vertices

Running Time Experimental Analysis

We use some functions in order to calculate some statistics such as sample mean, standard deviation, variance to experimentally analyze the running time of our algorithm.

```
float sum = 0, avg, var = 0, stdev = 0, tvalue90 = 1.660, tvalue95 = 1.984;

for (int i = 0; i < NUM; i++)
{
    sum += runtimes[i];
}

avg = sum/NUM;

for (int i = 0; i < NUM; i++)
{
    var += (runtimes[i] - avg)*(runtimes[i] - avg);
}

var = var/NUM;
stdev = sqrt(var);

float stderror = stdev/sqrt(NUM);
float confint90_1 = avg + tvalue90*stderror, confint90_2 = avg - tvalue90*stderror;
float confint95_1 = avg + tvalue95*stderror, confint95_2 = avg - tvalue95*stderror;

cout << endl << "mean = " << avg << "  stdev = " << stdev
    << "  stderror = " << stderror << "  90ci = " << confint90_1 << " - " << confint90_2
    << "  95ci = " << confint95_1 << " - " << confint95_2 << endl;
```

$$\sigma^2 = \frac{\sum(x-\mu)^2}{N}$$

$$s = \sqrt{\frac{\sum(x-\bar{x})^2}{n-1}}$$

$$SE = \frac{\sigma}{\sqrt{n}}$$

We used 90% and 95% confidence level internals in order to calculate the true means.

50t RUN PER INPUT SIZE

Size	Mean Time(s)	Standard Deviation	Standard Error	%90 - CL	%95 - CL
50	2.00E-05	0.00014	1.98E-05	0.0000528663 - - 1.28663e-005	5.92812e-005 - - 1.92812e-005
250	0.00132	0.000524976	7.42E-05	0.00150324 - 0.00125676	0.0015273 - 0.0012327
375	0.00546	0.00585221	0.000827628	0.00683386 - 0.00408614	0.00710201 - 0.00381799
500	0.0122	0.00206882	0.000292575	0.0126857 - 0.0117143	0.0127805 - 0.0116195
750	0.04166	0.00636431	0.000900049	0.0431541 - 0.0401659	0.0434457 - 0.0398743
1000	0.0957	0.00440568	0.000623057	0.0967343 - 0.0946657	0.0969362 - 0.0944639

Fig.6. 50 Run for 50,250,375,500,750,1000 vertices and Mean times, Standard Deviation, Standard Error, Confidence Levels

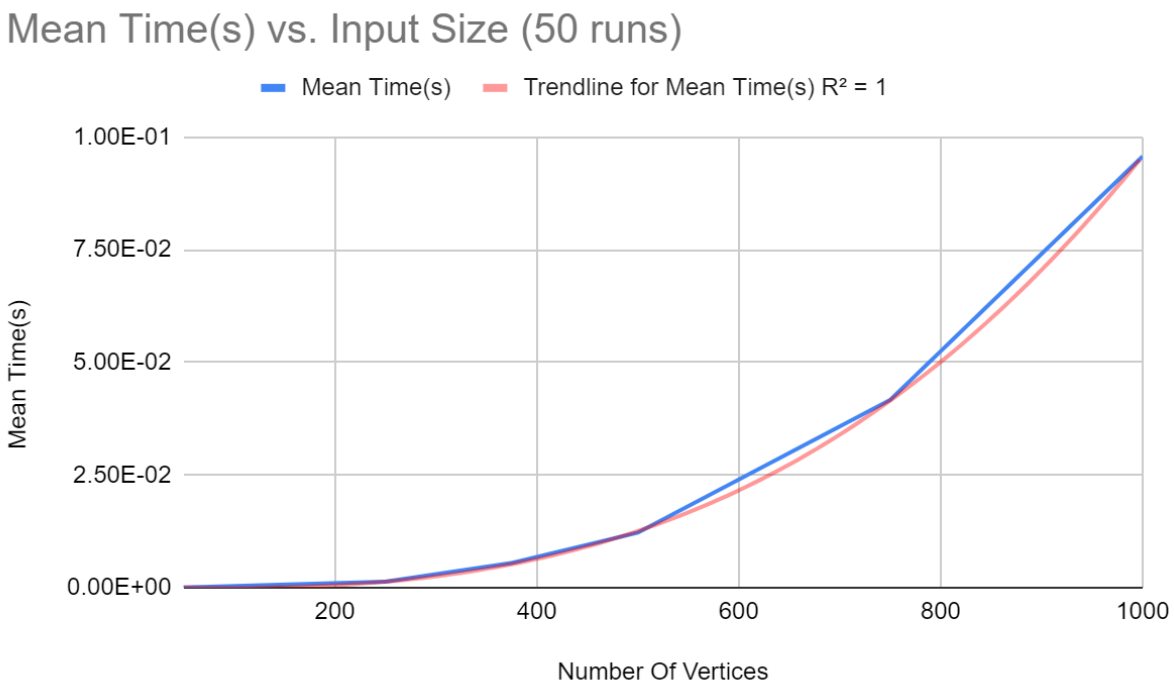

Fig.7. Running time graph of 50 run according to number of vertices

100 RUN PER INPUT SIZE

Size	Mean Time(s)	Standard Deviation	Standard Error	%90 - CL	%95 - CL
50	1.50E-04	0.00149248	1.49E-04	0.000397752 - -9.77518e-005	0.000446108 - -0.000146108
250	0.00286	0.00563741	5.64E-04	0.00379581 - 0.00192419	0.00397846 - 0.00174154
375	0.00658	0.00403034	0.000403034	0.00724903 - 0.00591096	0.00737962 - 0.00578038
500	0.0151	0.00319531	0.000319531	0.0156304 - 0.0145696	0.015734 - 0.0144661
750	0.05064	0.00411709	0.000411709	0.0513234 - 0.0499566	0.0514568 - 0.0498232
1000	0.11628	0.00367173	0.000367173	0.11689 - 0.115671	0.117008 - 0.115552

Fig.8. 100 Run for 50,250,375,500,750,1000 vertices and Mean times, Standard Deviation, Standard Error, Confidence Levels

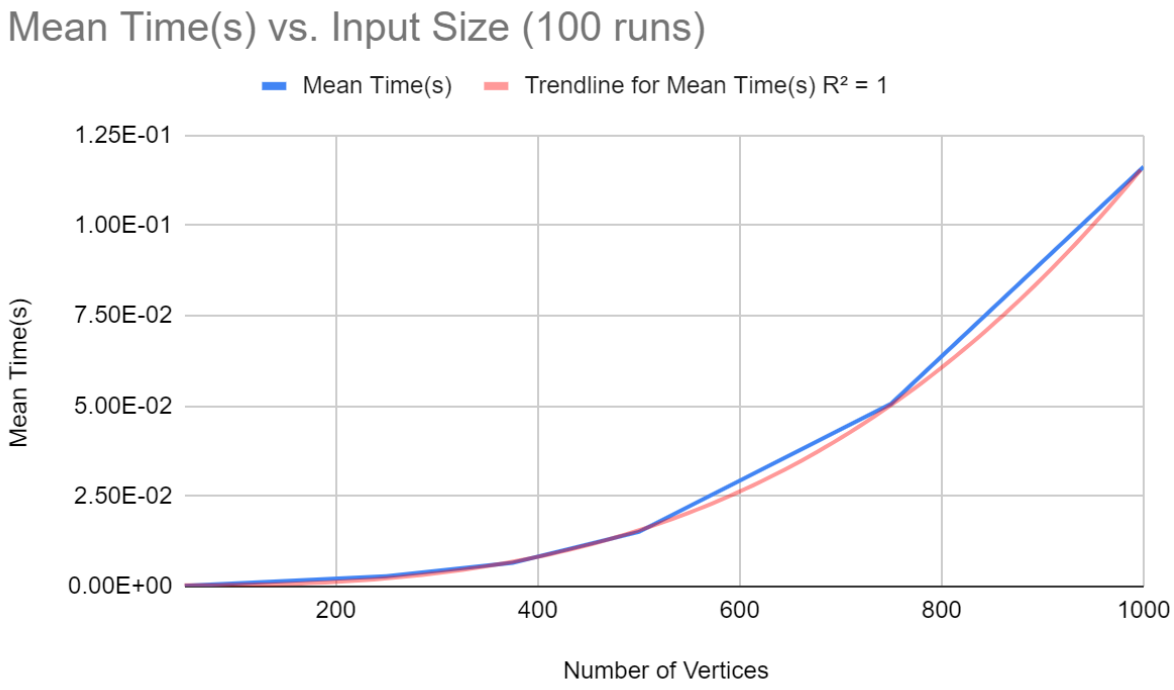

Fig.9. Running time graph of 100 run according to number of vertices

TESTING

For testing, we have used a black box method, and this time as our graphs, instead of checking for randomly generated graphs we checked for graphs that we have created, whose Hamiltonian paths were known. Then, we tried to determine some specific cases that our algorithm worked out better solu0.tions, or where it exclusively fails.

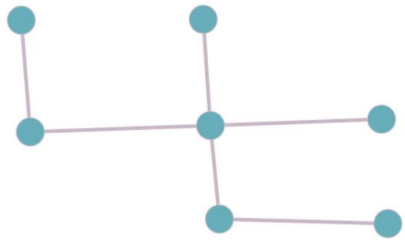

Figure 10. Test Graph 1

This was a simple example to show that our algorithm does not find Hamiltonian paths if there is none, the accuracy of our algorithm depends on the graphs with Hamiltonian paths, and we will see if our algorithm fails or succeeds to find them.

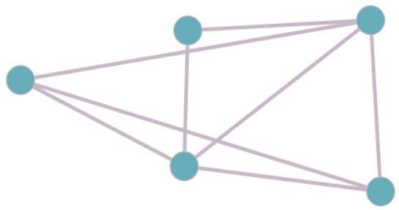

Figure 11. Test Graph 2

Graph 1 and 2's Hamiltonian path was successfully found by our algorithm. We can see that if the graph is closer to a complete graph, our algorithm has more success rate because of the greedy way it works. If the next node we are going to check has more connections to other nodes, there is a higher possibility for us not to fail.

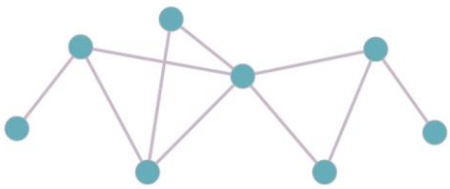

Figure 12. Test Graph 3

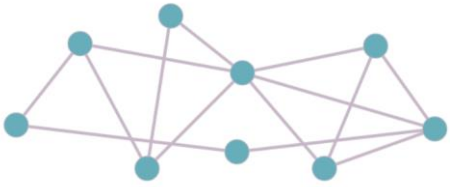

Figure 13. Test Graph 4

These two graphs' Hamiltonian paths were found or not in the condition of the starting node. If the starting node has the lowest degree, our algorithm works better, but if we start the algorithm from one of the middle nodes, with the way it works it has difficulties finding a Hamiltonian path and fails at the second graph.

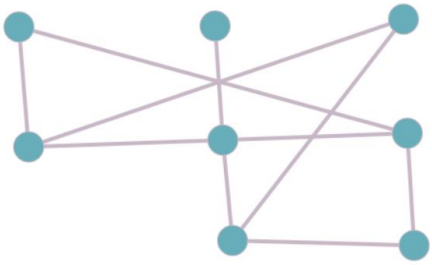

Figure 14. Test Graph 5

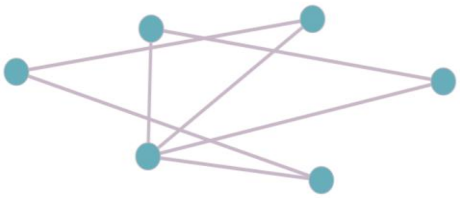

Figure 15. Test Graph 6

Our algorithm failed on determining if those graphs have a Hamiltonian path or not, the main reason for it that, those graphs does not have lots of different Hamiltonian path possibilities for a specific node as a starting point, and therefore can fail if the greedy step of our algorithm makes a non-optimal decision.

CONCLUSION

In conclusion, Hamiltonian Path is an NP-Complete problem, that is reduced from Hamiltonian Cycle. There is no possible algorithm to solve this problem in polynomial time if we want to have an exact solution to it. However, we can approach this problem in different ways to find less effective but more efficient solutions to it.

Our greedy Hamiltonian Path detection algorithm does not always find the Hamiltonian path. Its success rate is determined by some aspects such as how connected the graph is, and the number of vertices. Also, due to the fact that our algorithm works by trying to connect nodes one by one, and does not check the further steps while working on one, we found out and showed that the starting node is also really important for our algorithm to operate.

As the number of vertices increases, we have found out that the success rate of our algorithm converges to an approximate of 0.3, caused by the edge enrichment in larger graphs.

Also, as runtime, we checked the algorithm with randomly generated inputs, and compared the results with the results of our theoretical algorithm analysis experimentally. We have shown that the results we got from various inputs matched the trendline of n3, which was our initial theoretical result.

Then trying some specific graphs showed us that if we wanted to make another trade-off with efficiency and time, we can add a mechanism(such as Breadth-First-Search) to the algorithm which determines the starting node of our algorithm, making it start from the lowest degreed node, to make it work more accurately.

REFERENCES

1) Karlander, J. (n.d.). Retrieved from http://www.csc.kth.se/utbildning/kth/kurser/DD2354/algokomp10/Ovningar/Exercise6_Sol.pdf
2) Kaundal, K. (2017). Applications of Graph Theory in Everyday Life and Technology. Imperial Journal of Interdisciplinary Research, 3(3), 892–894. Retrieved from https://pdfs.semanticscholar.org/7aee/9da1025ea81e2fcfccaa53393b9b9ab76d05.pdf