380CT - Asis Rai

May 14, 2018

Name: Asis Rai
Student ID: 6528683
Coursework: 380CT Assignment
Module: Theoretical Aspects of Computer Science
Group Members: Asis Rai, Jubad Miah, Rakshak Nalukurthi & Muhammed Khan **[Shared Code and Pseudocode]**
GitHub Repository: https://github.coventry.ac.uk/raia10/380CT-CourseworkLateTeam

1 Notation

Let G be a directed graph, and let V be nodes of the graph G, $(V_1, V_2, \dots, V_v)$ and E being the edges of G.

Hamilton Cycle is denoted by HC.

Let P be *Path* being a list, given by $P = E_1 \wedge E_2 \wedge \cdots \wedge E_n$

Every *clause* in the path, E_n has the form $A \wedge B$, where A is the current node and B is the next node.

Number of nodes in the graph is be denoted by N.

$Degree of connectivity$ between nodes is denoted by C.

Distance between nodes is denoted by D.

2 Definition Of The Problem

The Hamiltonian Cycle problem is a problem of finding out if a Hamiltonian path which is a path in directed graph, in which a path visits every vertex of the graphs only once or a Hamiltonian cycle exists in a directed graph. (Johnson, 1979)

3 Complexity Class Membership

3.0.1 Decisional Hamiltonian Cycle Problem

The decision problem takes input (G, V, E) and asks if there is a HC starting at V and ending at E, returns **YES** if there is a HC, **No** otherwise. (Johnson, 1979)

Hamilton Cycle Decisionional Problem in a directed path is **NP-Complete**. It is **NP-Complete** because DHP decision problem can be solved by a non-deterministic Turing Machine in polynomial Time. (West, 2017)

3.0.2 Computational/Search Hamiltonian Cycle

If directed graph G has a HC, then return P to find the Hamiltonian Cycle. (www.ici.usi.edu, n.d.)

Computational/ Search Hamiltonian Cycle problem is **NP-Complete**. It is **NP-Complete** because it is a problem that can be solved by a non-deterministic Turning Machine in a polynomial Time. e.g. Finding the shortest P Hamiltonian cycle in a given G. (West, 2017)

3.0.3 Optimization Hamiltonian Cycle

Find the HC, P with the shortest D by minising number of non-hamiltonian instances. (Johnson, 1979)

Optimization Hamiltonian Cycle is **NP-Hard** because Optimization probelms can only be **HP-Hard** as the task is to minimize D of nodes in the Hamiltonian P. (Jillian Beardwood, 2008)

4 Testing Methodology

Exhaustive Search: Average time increases for instances with increasing number of N.

Greedy Heuristic: Average time increases for instances with increasing number of N.

Meta-Heuristic: * Quality of approximation is calculated with increasing P, with the shortest D. * Average time increases for instances with increasing number of N.

NOTE: Since the code was not implemented by the team because of coursework deadlines leading to less time for the code to be developed, 3 algorithms will be compared with their O notation and their Big O notation instead. With suggestions on what kind of situation they will be best translated into. The accuracy and runtime performances of the three algorithms cannot be tested with inccreasing graph nodes and probablity of connection because the team could not develop the code on time.

4.1 Random Graph

What is a Random Graph?

Random Graph is a graph in which properties, vertices, edges and connections between them are determined in random way. (Bollobas, 1985)

How to generate a Random Graph?

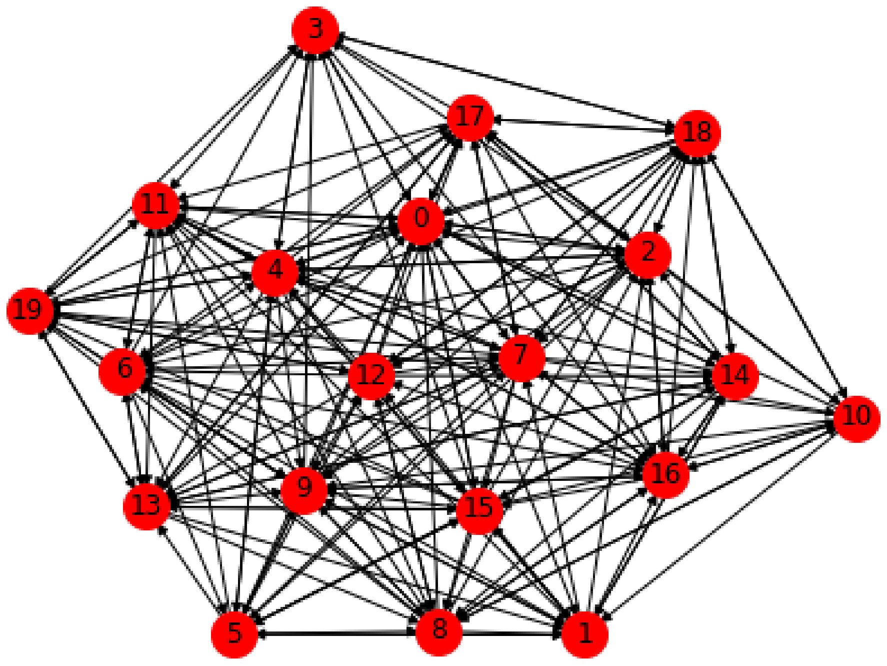
To generate a Random Graph, V is inputted from $(V_1, V_2, \ldots, V_v)$. Inputting number of nodes N, will generate a random number of edges and nodes.

GenerateGraphRandomly Function randomly generates a graph. N number of nodes, is given as vertex, it then creates the random graph, a new directed graph is created after that with the edges added into the new directed graph.

4.2 Random Graph Generation - Code

```
In [2]: """Imported Modules"""
        import math
        import random
        import networkx as nx
        import matplotlib.pyplot as plt
        import sys
```

```python
"""Base Class Definition"""
class DirectedHamilton:
    #Variables for results #
    times = []
    start = None
    end = None

    #Setting Parameters #
    # Number of nodes, strategy #
    vertex = None
    structure = None
    graph = [[0 for i in range(20)] for j in range(20)]

    def __init__(self, n, g):
        self.vertex = n
        self.graph = g

"""Random Graph Generation"""
def GenerateGraphRandomly(self, n):
    # Generate a random directed graph of size n #
    # Probability between connectivity between nodes is defined #
    # 50% probability set, change it to whatever you want #
    G = nx.gnp_random_graph(n, 0.50, directed = True)
    # Setting directed graph which is empty #
    S = nx.DiGraph()

    # This will get the edgdes from the random graph #
    # The edges will be used to connect to the new one #
    S.add_edges_from(G.edges())

    # Creating a list which stores value of nodes #
    List = list(G.nodes())
    # Shuffle the list so that it's random #
    random.shuffle( List )

    # This wil make sure that there is a cycle in the graph #
    for i in range(len(List) - 1):
        # Adding the paths, so that every node is connected #
        S.add_path([List[i], List[i + 1]])

    # Making sure that final node is connected to final node #
    # This makes sure that the graph has a HC #
    S.add_path([List[-1], List[0]])

    # Setting up edges to be connected between nodes #
    # Default colour is black, therefore the colour chosen is black #
    edges = [edge for edge in S.edges()]
    # Setting the layout for the graph, through the module imported #
    # Force-directed graph drawing #
    # Kamada & Kawai (1989) #
    position = nx.kamada_kawai_layout(S)

    # Connected the nodes to the graph #
    # Setting the layout as 'Ocean' for the graph #
    # Setting the nodes colour to Red #
    nx.draw_networkx_nodes(S, position, cmap = plt.get_cmap('ocean'),
                           node_color = "Red", node_size = 400)

    # Set the number labels on the nodes, in the graph #
    nx.draw_networkx_labels(S, position)

    # Set the egdes with arrows in graph #
    nx.draw_networkx_edges(S, position, edgelist = edges, arrows = True)
    limits=plt.axis('off')

    return S

# Initialize GenerateGraphRandomly function #
DirectedHamilton.GenerateGraphRandomly = GenerateGraphRandomly

"""Giving values & Output for the graph generation"""

# Give number of nodes to start with #
vertex = 20

# Give maximum size of the layout #
plt.figure(figsize=(8, 6))
rand = DirectedHamilton(vertex, None)

# Calling the function to randomly generate graph #
graph = rand.GenerateGraphRandomly(vertex)
```



Generation of Hamiltonian graphs (yes-instances of DHCP)

The code is written to make sure that every node is connected with it's next node, leading connectivity to the final node to make sure that there is a connectivity between them and also to make sure that there is a Hamiltonian Cycle in the graph.

The generated graph from the code has nodes and arrows connected to each nodes and to the last, this makes sure that the generated graph is a directed graph and it has a Hamilton Cycle inside it.

5 Solution Methods

5.1 Exhaustive search - Pseudocode

```
1. $G = (V, E)$
2. **FUNCTION** ExhaustiveSearch($P$,position)
3.     $HC$ = True;
4.     **FOR** each $V$ in $(V_1, V_2, \dots, V_n)$
5.         **IF** $V$ not in $P$
6.             $HC$ = False;
7.     **IF** $HC$:
```

```
8.          IF last $V$ is adjacent to first $N$ $P$ from $G$
9.              Return True
10.         ELSE
11.             Return False
12.  Else
13.      TRY different $V$ in $G$
14.         IF current $V$ and last $V$ are adjacent in $G$
15.             Return False
16.         ELSE
17.             Check if $V$ has been visited from $G$
18.         IF ExhaustiveSearch($P$,$P$+1) = True
19.             Return True
20.         ELSE
21.             Remove current $V$ if solution not found $P$[position] = 1
```

5.1.1 Exhaustive search - Discussion

The Exhaustive search in practice, goes through all possbilities to check whether there is a Hamiltonian cycle in the graph. If it cannot find a Hamilton cycle then the search keeps going until it has traversed through every path in the graph. This means that if a Hamiltonian path is found then the Exhaustive search returns the most optimal path. However, if there is not Hamiltonian Cycle, then False is returned. (Kreinovich, 1987)

If Exhaustive search was tested in practice, i believe the Exhaustive search would have exponential growth. This is because, the algorithm depends on the N of V given. If there are a large amounts of nodes V, the time to find an optimal solution would increase aswell, this means that the Exhaustive search will have to keep traversing around every single node in the graph. Therefore, in practice the algorithm depends on the number of nodes N inputted, if the size of the V is low then the algorithm would return an optimal path quickly however as the number of nodes N increases in the graph, the algorithms would take longer to find the optimal path and will have exponential growth.

Big O Nototation:

BigO Complexity: $O(n!)$

It is $O(n!)$ for this algorithm because every V is iterated and therefore will always be one remaining V, which gives notation O(n) and (n*(n-1)!) to find the optimal path. Therefore with every call, the worst factor is 0(n!) because every call reduces branch factor by 1.

5.2 Greedy Heuristic - Pseudocode

```
1. FUNCTION GreedyHeuristic()
2.     FirstNode$V$ $\leftarrow$ First position
3.     Path$P$ $\leftarrow$ []
4.     While locations to travel
5.         For neighbours, $V$ of FirstNode$V$ do
6.             If $V$ has greatest number of neighbours then
7.                 FirstNode$V$ $\leftarrow$ $V$
8.             Append FirstNode$V$ to Path$P$
9. End While
10. Return Path$P$
```

5.2.1 Greedy Heuristic - Discussion

The greedy algorithm traverses through the path by picking a random node and assining it as the startiting node, then it starts on the node to determine all the edges connected from the node, by traversing through one node at a time to find a path. It will traverse through the cheapest path, to find the cheapest fath.It will stop traversing and looking for paths when there are no nodes left to traverse to. It is an algorithm which tries to find a Hamiltonian path by visiting fewest number of nodes. (Stefan Voss, 1999)

Sometimes greedy algorithms fail to find the globally optimal solution because they do not consider all the nodes in the graph. The choice made by a greedy algorithm may depend on choices it has made so far, but it is not aware of future choices it could make. Therefore the path it returns is not always the optimal path. It is faster than Exhaustive but Exhaustive Search is more reliable.

Big O Nototation:

BigO Complexity: $O(N^2)$

Due to the fact that for each node, the near neighbour have to be located therefore giving the first n, computing distance between two nodes will get the factor 1 and it runs in $O(1)$. The travelling distance between one node to all the other cheapest node will give the factor of $O(n)$, making it ultimately $O(N^2)$.

6 Meta-Heuristic - Tabu Search

```
1. FUNCTION TabuSearch()
2.     While $Tabu < Tabu_m $ do
3.         $Tabu \leftarrow Tabu + 1$
4.         Search Local Neighborhood
5.         Evaluate Seek Best Solutions
```

```
6.        **Update** Tabulist
7. **End while**
8. **Return** count of satisfied clauses
```

6.0.1 Tabu Search - Discussion

The Tabu Search algorithm maintains a tabu list of edges giving record of all the path visited. The algorithm begins with an initial *Path* P as a solution and then checks for other possible P solutions, comparing neighbouring solutions with the intitial solution. This is done by adding each solution in TabuList and evalutating the best one by removing the worst *Path* from the TabuList. The TabuList contains all the list of P visited, therfore aviding any illegal move and avioding groing around in circles.(Glover, 1986)

Big O Nototation:

BigO Complexity: $O(N^2)$

7 Special Cases

1. Singleton graph, where a graph which has just one V and no E.
2. Single V missing an E therefore not being able to connect to the neighbouring Vs'.
3. Complete Graph where each V is connected to another V in the graph.
4. Empty graph, it is a special case of the Singleton graph where the graph contains empty V and the V's have no edges.
5. Multiple V but zero E.

8 Conclusion

From the research, I can conclude that:

The Exhaustive Search: This algorithm is best when trying to solve the DHCP in a smaller size graph, with small number of nodes V, populated throughout the graph. If there are a large amounts of nodes V, the time to find an optimal solution would increase aswell, this means that the Exhaustive search will have to keep traversing around every single node in the graph. Therefore, the algorithm depends on the number of nodes V inputted, if the size of the V is low then the algorithm would return an optimal path quickly however as the number of nodes N increases in the graph, the algorithms would take longer to find the optimal path and will have exponential growth.

The Greedy Search: This algorithm is faster than Exhaustive search, this is self-explanatory since the BigO notation for greedy is $O(n^2)$ compared to $O(n!)$ of Exhaustive Search. The Exhaustive search is slower than the Greedy search but it is best for finding the best optimal path of a HC. The Greedy algorithm fail to find the optimal solution because it doesn't consider all the nodes in the graph. The choice made by a greedy algorithm depends on choices it has made so far and it is not aware of future choices it could make. Therefore, the path it returns is not always the optimal path. Therefore, Greedy Search is useful when a solution is not needed to be optimal but needed to be produced faster.

The Tabu Search: The Tabu Search is the most efficient of the three because unlike the other two, Exhaustive and Greedy, The Tabu Search stores every P visited in the through into it's list, the TabuList contains all the list of P visited, therefore avoiding any illegal move and avoiding going around in circles, therefore finding the most best optimal path possible.

9 Reflection

Overall, working on this project has been very beneficial to me. I have a new understanding of Big O complexity can help algorithms grow and improve. It was particularly beneficial understanding Big O in-depth because last year, I only had basic knowledge of it. Furthermore, i also gained new knowledge in Pseudocode. Going into depth this year, fully understand Pseudocode, made me realise how wrongly i wrote Pseudocode for programming last year.

Last year we were only taught the basics of graph generation, this project has helped me learn to create new graph generation like Random graph with a HC inside it. To find the HC, algorithms had to be researched as a group. This helped me gain transferable skills, particularly social skills. I could communicate and learn better, leading me to find out that i work better in a group than individually.

However, I also faced some huge difficulties while working on this project. One in particular, was time-management. I felt like i sometimes spent too long learning things that shouldn't have taken me that long, such as Jupiter Notebook. It was a big hassle because i spent a long time researching about it when i could have asked my group peers all on about it, which would have made me learn quicker. Other problems were understanding the problem and coding to solve the problem. It was a huge challenge balancing loads between doing this coursework, doing dissertation, revising for the phase test, understand and learning theoretical aspects of the course.

10 References

Bollobas, B., 1985. Random Graphs. First Printing edition ed. s.l.:Academic Pr; First Printing edition (September 1, 1985).

Glover, F., 1986. Tabu Search – Part 1. ORSA Journal on Computing, p. 190–206.

Jillian Beardwood, J. H. H. a. J. M. H., 2008. The shortest path through many points. Volume 55, Issue 4 ed. USA: s.n.

Johnson, M. R. G. a. D. S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. United States: W. H. Freeman and Company.

Kreinovich, V., 1987. Problems of Reducing the Exhaustive Search. Series 2, Volume 178 ed. s.l.:s.n.

Stefan Voss, S. M. I. H. C. R., 1999. Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization. First Edition ed. France : Kluwer Science + Business Media New York .

West, D. B., 2017. Introduction to Graph Theory. Second Edition ed. USA: Pearson Education Heg .

www.ici.usi.edu, n.d. More NP-Complete and. [Online] Available at: http://www.ics.uci.edu/~goodrich/teach/cs162/notes/pnp3.pdf [Accessed 04 May 2018].