Check if a given undirected graph is Hamiltonian or not

Gaganjeet Reen ICM2015003
Vinay Surya Prakash IHM2015004
Swapnesh Narayan ISM2015002
Leetesh Meena ISM2014008
Group - D

***Abstract*— Consider a simple, undirected graph G with given Adjacency Matrices X(G).In this paper we propose an algorithm to find whether or not the graph is Hamiltonian or not . Further we evaluate the proposed algorithms time complexity by the help of a plot of Time vs Number of vertices.**

I. INTRODUCTION

An interconnection of points is known as Graphs , or more formally , A graph **G** is a set of **E** and **V** , where **E** denotes the set of edges and **V** denotes the set of Vertices . Here a vertex v represents a point or node and an edge e is a connection between two vertices. v_i and v_j are known as endpoints of that edge if an edge e_{ij} connects the vertices v_i and v_j.

A) Adjacency Matrix
The adjacency matrix, sometimes also called the connection matrix, of a simple labeled graph is a matrix with rows and columns labeled by graph vertices.If there exists an edge from v_i to v_j ,the corresponding entry in the adjacency matrix is 1,otherwise the entry is 0. For a simple graph with no self-loops, the adjacency matrix must have 0s on the diagonal. For an undirected graph, the adjacency matrix is symmetric.

B) Cycle in a graph
A cycle in a graph is a closed walk(alternating sequence of vertices and edges with no edge being repeated) in which each vertex except the terminal vertex appears once.

C) Hamiltonian Cycle
A Hamiltonian cycle, also called a Hamiltonian circuit, is a graph cycle through a graph that visits each node exactly once. In general, the problem of finding a Hamiltonian cycle is NP-complete so the only known way to determine whether a given general graph has a Hamiltonian cycle is to undertake an exhaustive search.

D) Hamiltonian Graph
A graph that contains a Hamiltonian cycle is said to be a Hamiltoninan Graph.

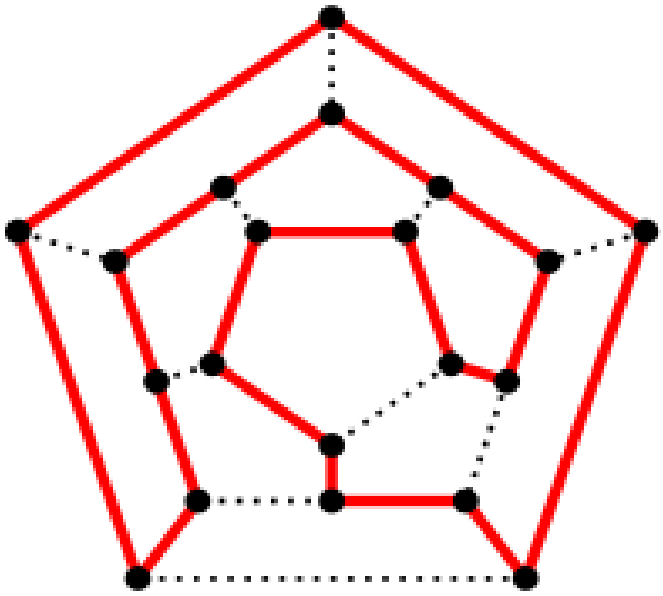

Fig. 1. Example of Hamiltonian Graph

II. MOTIVATION

It is not an easy task to determine whether a given graph is Hamiltonian or not.It is a NP-Complete problem and requires exhaustive search. Determining whether a graph is hamiltonian or not can have a lot of practical uses such as reducing communication costs,reducing delivery costs,efficient routing algorithms etc.

The basic properties of a Hamiltoninan Circuit are :-

A) If a graph has any vertex of degree one then the graph cannot have Hamilton Circuit.

B) If a vertex in the graph has degree two, then both edges that are incident with this vertex must be part of any Hamiltonian Circuit.

C) If there is an edge in the graph which is when removed, divide the graph into two disjoint sub-graphs then the original graph cannot have Hamiltonian Circuit.

D) If there is a vertex in the graph which is when removed (connected/incident edges will also be removed) divide the graph into two disjoint sub-graphs, then the original graph cannot have Hamiltonian Circuit. (This property is not necessary for degree-3 graphs but useful for higher degree graphs therefore we are not including checks for this property in the following algorithm).

E) If edges of degree two vertices create a cycle and that cycle does not contain all vertices of graph, the graph cannot have any Hamiltonian Circuit.

F) When a Hamiltonian Circuit is being constructed this circuit passes through a vertex, then all remaining edges incident with this vertex, other than the two used in the circuit, can be removed

A few useful theorems which can be considered when talking about Hamiltonian Circuits are :-

A) Diracs Theorem:If G is a simple graph with n vertices with n greater than or equal to 3 such that the degree of every vertex in G is at least n/2 then G has a Hamilton Circuit.

B) Ores Theorem:If G is a simple graph with n vertices with n greater than or equal to 3 such that deg (u) + deg (v) is greater than or equal to n for every pair of non-adjacent vertices u and v in G, then G has Hamilton Circuit.

These theorems however, do not give enough information to find out a Hamiltonian Circuit and can not be used as the basis of an algorithm.

III. METHODS AND DESCRIPTION

In this paper,we have proposed two methods to determine whether a graph is Hamiltonian or not. In **Approach 1**, we first generate all the permutations which are possible for given set of vertices.Then we checked whether the particular permutations form Hamiltonian circuit or not. If the particular permutation satisfies the properties required for it to be a Hamiltonian Circuit,we conclude that the graph is a Hamiltonian graph.This is essentially a brute force method which checks all the possible permutations of graph vertices.
In **Approach 2**, we use a dfs and backtracking algorithm to find whether or not a graph contains a Hamiltonian Circuit. We create an empty path array and add vertex 0 to it. Add other vertices, starting from the vertex 1. Before adding a vertex, check for whether it is adjacent to the previously added vertex and not already added. If we find such a vertex, we add the vertex as part of the solution. If we do not find a vertex then we return false.

IV. PROPOSED ALGORITHMS

The two algorithms used in the two suggested approaches are given below.

Algorithm 1 Check for Hamiltonian Circuit using Permutations

```
1: while there are untried configurations do
2:     generate next configuration of Graph G
3:
4:     if there are edges between two consecutive
       vertices of this configuration and there is an
       edge from the last vertex to the first then
5:         print this configuration
6:         break
7:     end if
8: end while
```

Algorithm 2 Check for Hamiltonian Circuit using Backtracking

if all vertices are included in Hamiltonian Circuit **then**

    if there is an edge from the last included vertex to the first vertex **then**
        return true
    else
        return false
    end if
else
    for all vertex v in graph G **do**

        if vertex v can be added to Hamiltonian Circuit **then**
            recur to construct rest of the path
            if adding vertex v doesn't lead to a solution **then**
                Remove the vertex
            else
                return true
            end if
        end if
    end for
    if no vertex can be added to Hamiltonian Circuit constructed so far **then**
        return False
    else
        return true
    end if
end if

V. IMPLEMENTATION

We have used C++ to accept a graph from the user which is then converted into the corresponding adjacency matrix **X(G)**. In Approach 1 we first generate all the possible vertex permutations of the graph and then for each permutation check whether or not a Hamiltonian Cycle can be formed for that permutation.As soon as we get a permutation we return true indicating that the graph is a hamiltonian graph.

In Approach 2,we save time by using a backtracking approach which adds connected adjacent vertices to the result set.In case the particular vertex does not lead to an answer,we simply backtrack and remove it.

VI. RESULTS

In case of the ordinary code the time complexity of checking whether a graph is hamiltonian or not is $O(N! * N)$ however when we use the backtracking code,the time complexity reduces to $O(N!)$ The comparision between the run time of the two codes is shown in Fig.8.

Github Repository Link : `https://github.com/piano-man/graph_theory.git`

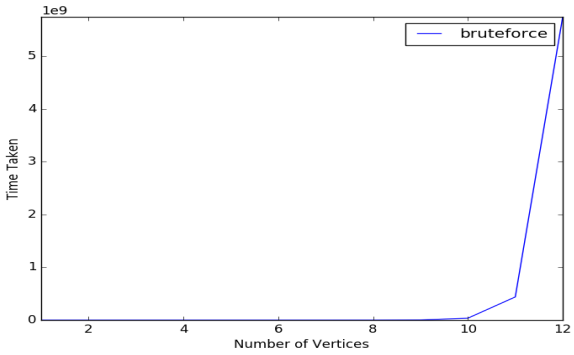

Fig. 2. Time vs vertices graph for brute force algorithm)

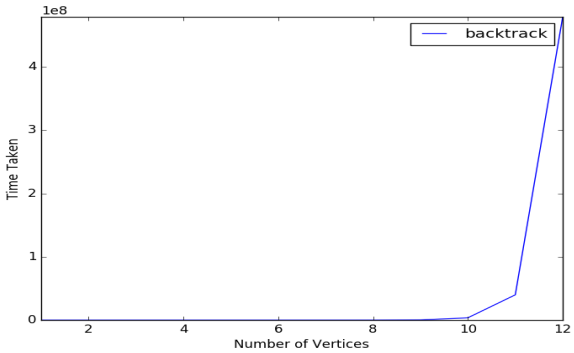

Fig. 3. Time vs vertices graph for backtracking algorithm

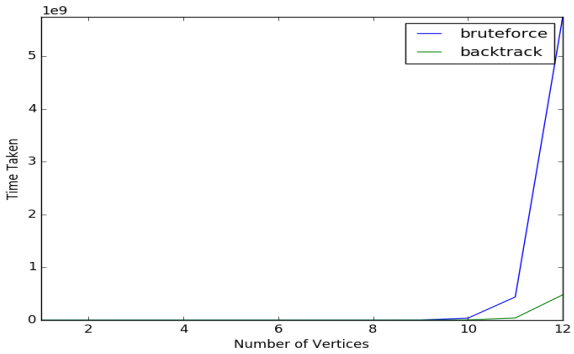

Fig. 4. Time vs vertices comparison graph for the two algorithm

VII. CODE SNIPPETS

```
def createRandomMatrix(n):
    a = permutations(range(n))
    fact = 1
    for i in range(1,n+1):
        fact *= i
        print(fact)
    l = fact/ 2
    count = 0
    per = []
    for x in a:
        count += 1
        if(count == l):
            break
        per = x
    print(l)
    mat = np.zeros((n,n))
    for i in range(n-1):
        mat[per[i]][per[i+1]] = 1
        mat[per[i]][per[i+1]] = 1
    return mat
```

Fig. 5. Creating permutations in the brute force method

```
def checkIfValid(mat,pattern):
    for i in range(len(pattern)-1):
        x = pattern[i]
        y = pattern[i+1]
        if(mat[x][y] == 0 and mat[y][x] == 0):
            return False
    return True
def checkIfHamiltonian(n,a):
    #a = permutations(range(n))
    mat = createRandomMatrix(n)
    isHamiltonian = False
    #print(mat)
    #mat = makeHamiltonian(mat)

    t = time()
    count =0
    for pattern in a:
        if(checkIfValid(mat,pattern)):
            isHamiltonian = True
            print("This is a Hamiltonian Path")
            count += 1
            print(count)
            break
        else:
            continue
    if(not isHamiltonian):
        print("Not a Hamiltonian")
```

Fig. 6. Checking for Hamiltonian in brute force method

```
class Graph():
    def __init__(self, vertices):
        self.graph = [[0 for column in range(vertices)]\
                      for row in range(vertices)]
        self.V = vertices

    ''' Check if this vertex is an adjacent vertex
        of the previously added vertex and is not
        included in the path earlier '''
    def isSafe(self, v, pos, path):
        # Check if current vertex and last vertex
        # in path are adjacent
        if self.graph[ path[pos-1] ][v] == 0:
            return False

        # Check if current vertex not already in path
        for vertex in path:
            if vertex == v:
                return False

        return True
```

Fig. 7. Graph class functionality in Backtracking Algorithm

```
def hamCycle(self):
    path = [-1] * self.V

    ''' Let us put vertex 0 as the first vertex
        in the path. If there is a Hamiltonian Cycle,
        then the path can be started from any point
        of the cycle as the graph is undirected '''
    path[0] = 0

    if self.hamCycleUtil(path,1) == False:
        print ("Solution does not exist\n")
        return False

    self.printSolution(path)
    return True
```

Fig. 8. Backtracking Algorithm

```
def hamCycleUtil(self, path, pos):

    # base case: if all vertices are
    # included in the path
    if pos == self.V:
        # Last vertex must be adjacent to the
        # first vertex in path to make a cyle
            return True

    # Try different vertices as a next candidate
    # in Hamiltonian Cycle. We don't try for 0 as
    # we included 0 as starting point in in hamCycle()
    for v in range(1,self.V):

        if self.isSafe(v, pos, path) == True:

            path[pos] = v

            if self.hamCycleUtil(path, pos+1) == True:
                return True

            # Remove current vertex if it doesn't
            # lead to a solution
            path[pos] = -1

    return False
```

Fig. 9. Backtracking Algorithm

Analysis and comparison of algorithms :-

The *best case* time complexity for the brute force code is $O(N)$.This occurs when the first permutation being tested is the correct permutation which shows that the graph is Hamiltonian.

The *worst case* tine complexity for the brute force code is $O(N! * N)$.This occurs when the correct

permutation is encountered as the last permutation being tested.
The *best case* time complexity for the backtracking code is $O(N)$.This occurs when no wrong vertex is encountered and all the vertices included for the first time lie in the Hamiltonian graph.
The *worst case* time complexity for the parallel code is :- $O(N!)$.This occurs when several faulty vertices are encountered and backtracking is triggered several times.

VIII. APPLICATIONS

The problem of finding the Hamiltonian Cycle in a graph is of immense use in real life.Some of these applications are :-

- **Networking :-**Finding Hamiltonian Circuits could prove to be really advantageous in the field of networking to ensure efficient delivery and unique delivery of packets .
- **Mailing Services :-**Finding leas cost Hamiltonian Circuits could be really advantageous for efficient delivery of mail.
- **Door to Door Delivery Systems :-**Door to Door delivery can be made a lot more efficient by using Hamiltonian(rather least cost Hamiltonian)circuits.

IX. DISCUSSION

In this paper we used the given algorithms to determine whether or not the given graph is Hamiltonian.While the first approach which generates all the permutations works, the backtracking approach gives us a more effecient solution.In the backtracking algorithm the starting point is taken as vertex 0.However, this should not matter as the starting point could be anything in a cycle.

X. CONCLUSIONS

We can check whether a graph is Hamiltonian or not using the methods suggested in the paper.However,a polynomial time algorithm for the same is not yet possible.Algorithm using dfs and backtracking gives relatively better results though.

REFERENCES

[1] Narsingh Deo, Graph Theory with Applications to Engineering and Computer Science.
[2] https://www.geeksforgeeks.org/hamiltonian-cycle-backtracking-6/
[3] https://pdfs.semanticscholar.org/28bf/5fc14343000ee6b4e2422ab4910f8a867
[4] https://en.wikipedia.org/wiki/Cycle_(graph_theory)
[5] http://www.worldscientificnews.com/wp-content/uploads/2017/08/WSN-89-2017-71-81.pdf
[6] https://www.quora.com/How-can-we-tell-that-a-graph-is-Hamiltonian
[7] http://mathworld.wolfram.com/HamiltonianCycle.html