Finding Equal Weighted Eulerian Circuits in a Weighted Graph

Kritika Sharma*, Saurabh Singh†, Anujraj Goel‡
Indian Institute of Information Technology, Allahabad
*icm2014502@iiita.ac.in
†iwm2014004@iiita.ac.in
‡iim2014002@iiita.ac.in

***Abstract*—The problem of finding the eulerian circuits has its place in many fields. Given a graph G = (V, E), the problem is to find all the equal weighted eulerian circuits possible in this graph. In this paper, the given graph G is first represented in the form of adjacency matrix. We introduced two algorithms to find all the eulerian circuits possible. In both of the algorithms, we consider each possible subset and find the possible eulerian circuits. The first algorithm results into worst case time complexity of O((N+M)^2*2^N) and the second algorithm takes the worst case time complexity of O((N+M)*2^N). Eulerian circuits has many applications like postman problem, door to door problem,etc.**

I. Introduction

In a graph G(V, E), where V represents a set of vertices and E represent set of edges, a path is defined as a finite or infinite sequence of edges which connect a sequence of vertices' which are all distinct from one another. This can be thought of as a trail formed by walking through v0, e1, v1, ..., vk with no repeated edge. The length of a trail is its number of edges. As an example, A path *a-f-c-d-e-b-h* is highlighted in Figure 1. The path length for the depicted path is 7. Now, A u,v-trail is a trail with first vertex u and last vertex v, where u and v are known as the endpoints. Hence , A trail is said to be closed if its endpoints are the same. Consequently, a circuit is a A closed trail is called a circuit when it is specified in cyclic order but no first vertex is explicitly identified. As an example, A cycle *3-4-6-3* is highlighted in Figure 2. Finally, a Eulerian trail (or Eulerian path) is a trail in a finite graph which visits every edge exactly once. Similarly, an Eulerian circuit or Eulerian cycle is an Eulerian trail which starts and ends on the same vertex. As an example, one such Eulerian circuit in Figure 3 can be depicted by : 1-2-0-4-3-0-1.

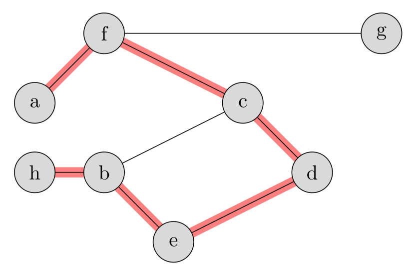

Fig. 1. A from vertex a to vertex h is highlighted in red.

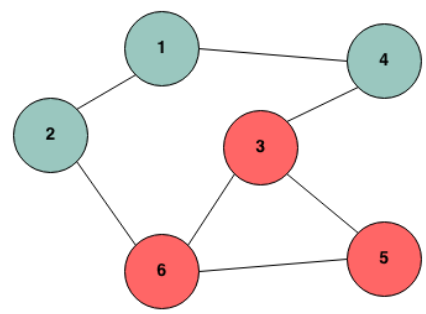

Fig. 2. A cycle in the above graph is depicted in red.

II. Motivation

This paper is motivated by the various real life examples where euler circuits are important. Euler circuits are useful when we consider postmen who want to have a route such that they should not revisit any of the edges or roads more than once but these should be visited atleast once. Euler circuits and paths also find their applications in painters, garbage collectors, airplane pilots and all world navigators with some daily travsersals. Chinese postman problem also describes the usefulness of the euler circuits. Other applications are : Finding Hurricane Victims in time of natural disasters, selling door to door starting from a warehouse and ending at the same warehouse such that all the roads are covered.

III. Methods and Descriptions

In this paper, we introduced two algorithms for finding the the all possible equal weighted eulerian circuits in the subgraphs. Our approach is to consider all possible subsets

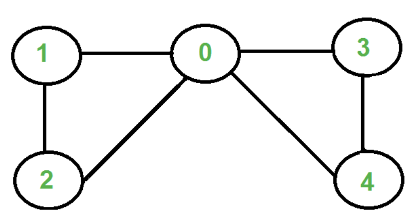

Fig. 3. A Eulerian circuit in the above graph.

of vertices and then to check if eulerian circuit is possible considering only the vertices in this subset. If eulerian circuit is possible we further move forward to find the eulerian circuit. For finding the eulerian circuits, two approached have been adopted. In the first algorithm, we find the eulerian circuits using the standard fluery algorithm. In this algorithm, we start at any one of the vertices in the subset considered. We then follow edges one at a time. While moving, if we have a choice between a bridge and a non-bridge, we always choose the non-bridge. After convering all the edges, we stop.

Implementation of algorithm 1 is as given below :

Algorithm 1

```
procedure isEulerian(currset,G)
    odd ← 0
    if Graphisnotconnected then return 0
    odd ← 0
    for (j = 1 to N) do
        if j in currset and degree(j) is odd then
            odd ← odd + 1
    if odd > 2 then return 0
    if odd > 0 then return 1
    elsereturn 2
procedure isValid(u,v)
    count ← 0
    for (each j adjacent to u) do
        if j in currset then
            count ← count + 1
    if count==1 then return 1
    count1 ← No. of vertices reachable from u
    G ← G − edge(u, v)
    count2 ← No. of vertices reachable from u
    G ← G + edge(u, v)
    if count1 > count2 then return 0
    elsereturn 1
procedure getEulerCircuitUtil(j,currset,G,crc)
    sum ← 0
    for (each v adjacent to j) do
        if v in currset and isValid(j,v) then
            sum ← sum + weight(j, v)
            crc ← crc + j
            G ← G − edge(j, v)
            sum ← sum + getEulerCircuitUtil(v, currset, G, crc)
    return sum
procedure getEulerCircuit(currset,G,N)
    for (j = 1 to N) do
        if j in currset then
            crc ← empty circuit
            wt ← getEulerCircuitUtil(j, currset, G, crc)
            crc ← crc + first entry of crc return (wt, crc)
procedure EquiWeightedEulerians(G,N)
    sets ← All possible subsets in graph
    res ← empty set
    for (each currset in sets) do
        res ← isEulerian(currset, G)
        if res==0 orres == 1 then
            "subset has no eulerian circuit"
        else
            res ← res ∪ getEulerCircuit(currset, G, N)
```

Here, isEulerian() is the function to check if there is eulerian circuit present in the subgraph or not. Basically, it checks if graph is connected or not and then also checks the vertices with odd degree. If number of vertices with odd degree are greater than 0 then the subgraph cant't have a eulerian circuit. The function isValid(u,v) checks if edge(u,v) can be chosen as next edge to be traversed or not. It can be traversed if this is the only possible edge left from u or if this is not a bridge edge.

The function getEulerCircuitUtil() recursively traverses all the egdes for getting the euler circuit in the considered subgraph. The function getEulerCircuit() starts from a vertex in the subgraph and call getEulerCircuitUtil() to traverse the complete euler circuit.

The function equiWeightedEulerians() simply call getEulerCircuit() for each possible subgraphs in the given graph G.

In the algorithm 2, we again considered all the possible subsets of vertices. Then for each subset, we choose starting vertex u and then follow edges until returning to starting vertex u. We will not get stuck at any other vertex than u. This is because indegree and outdegree of every vertex must be same. The circuit formed after this is a closed, but may not cover all the vertices and edges of the initial graph. As long as there is a vertex u that belongs to the current circuit but that has edges connected to it not part of the tour, start another traversal from u, following unused edges until returning to u. After this, attach the circuit resulted after this to the previous tour. This second algorithm is basically motivated from Hierholzers Algorithm.

Implementation of algorithm 2 is as given below :

Algorithm 2

```
procedure getEulerCircuit(currset,G,N)
    edgecount ← Count of edges with each vertex considering currset
    vis ← Initialised with all edges unmarked
    currpath ← empty stack
    circuit ← empty vector
    ind ← −1
    for (j = 1 to N) do
        if j in currset then
            ind ← j
            break
    currpath.push(ind)
    currv ← ind
    while currpath not empty do
        if (edgecount[currv]) then
            currpath.push(currv)
            nextv ← adjacent vertex v to currv such that edge(currv,v) is unvisited
            edgecount[currv] ← edgecount[currv] − 1
            edgecount[nextv] ← edgecount[nextv] − 1
            vis[edge(currv, nextv)] ← 1
            currv ← nextv
        else
            curcuit.push(currv)
            currv ←currpath.top()
            currpath.pop()
    tot ← 0
    for (j = circuit.size()-1 to 1) do
        tot ← tot + weight(j, j + 1)
    return (tot, circuit)
```

In the implementation algorithm 2, we maintain a stack to keep vertices, a vector to store final circuit. Now, we start from any vertex in the subset and push it to the stack. Also this vertex is assigned to currv. We process while the stack is not empty. At each loop, we check is there is any edges attached to currv. If edge is there, we push this currv to the stack. Now, we select an edge (currv,nextv) to be traversed where nextv is some vertex adjacent to currv and edge(currv,nextv) is unvisited. After this, we remove this edge from the graph. Now, we assign nextv to currv. If there is no edge adjacent to currv, then we backtrack to find remaining circuit. In this case, we push currv to circuit and then pop out the top entry of stack and assign this to currv. In this way, we keep on doing.

IV. Implementation and Results

In this paper, we implemented two algorithms. In algorithm 1, we take each possible subset of vertices. For each subset, we find the eulerian circuit if present. We start from a vertex and following this visit all the edges once and only once. We dont burn edges i.e. we avoid bridges. The time complexity of the this implementation is O($(N+M)^2$). The function getEulerCircuitUtil() acts like the DFS algorithm for traversing the edges and it calls isValid() to check if we should visit the particular edge or not. This function is similar DFS two times. As we know that the time complexity of the DFS algorithm for the adjacency list representation is O(N+M). Therefore the complete time complexity is O($(N+M)^2*2^N$). This complexity results into O(M^2)$*2^N$ for connected graph

The time complexity of this algorithm is O(N+M). This is because we are visiting all the vertices and edges once. This works similar to DFS algorithm, visiting the complete graph without repetition and we know that dfs algorithm takes O(N+M) time. Hence this algorithm takes O($(N+M)*2^N$) time.

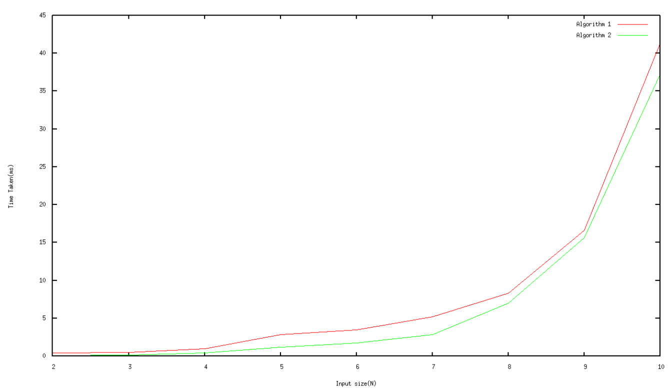

Fig. 4. **Input size vs Time graph 1**

In algorithm 2, we take each possible subset and for each subset we call this Hierholzers algorithm. In this algorithm, the idea is that we keep on traversing the unused edges and keep them removing until we get stuck. As soon as we get stuck, we go back to the last nearest vertex in our current path that has unused edges attached to it, and we repeat this process till all the edges have not been used.

In the fig.4, the graph for input size i.e. Number of vertices vs time (in milliseconds) is given. As we can see, the time taken by algorithm 1 is greater than that of algorithm 2. This graph is for N = 2 to 10.

In the fig.5 also, the graph for input size vs time (in milliseconds) is given. As we can see, the time taken by algorithm 1 is greater than that of algorithm 2. This graph is for N = 10 to 14.

Similarly, the graph for N = 15 to 20 is given in fig. 6.

Considering all the above, we can conclude that the algorithm 2 is an improvisation over algorithm 1. Also, as the number of vertices (N) increases, the overhead of time taken increases in algorithm 1 as compared to the algorithm 2. The difference between the time taken by algorithm 1 and algorithm 1 increases as the number of vertices increases by huge amount as can be shown in fig. 6.

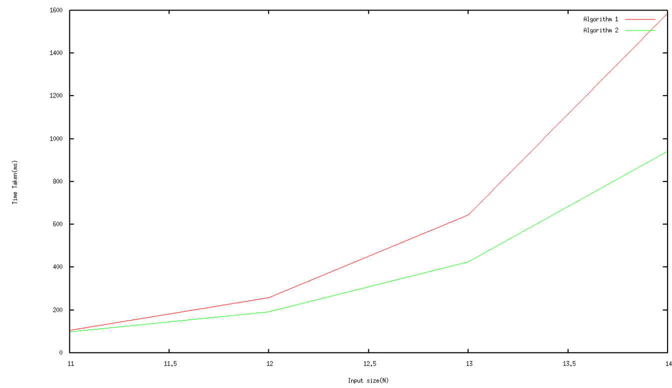

Fig. 5. **Input size vs Time graph 2**

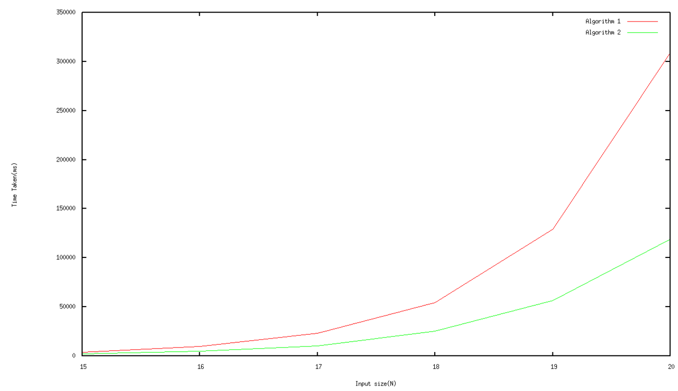

Fig. 6. **Input size vs Time graph 3**

V. Conclusion

In this paper, we introduced two algorithms for finding all the equal weighted eulerian circuits in the subgraphs. In both the algorithms, we created all possible subset for the graph given. Then, in the first algorithm, we take each possible subset of vertices. For each subset, we find the eulerian circuit if present. We start from a vertex and following this visit all the edges once and only once. We dont burn edges i.e. we avoid bridges. The time complexity of the this implementation is O($(N+M)^2*2^N$).In the second algorithm, the idea is that we keep on traversing the unused edges and keep them removing until we get stuck. As soon as we get stuck, we go back to the last nearest vertex in our current path that has unused edges attached to it, and we repeat this process till all the edges have not been used. The time complexity of this is O($(N+M)*2^N$). Eulerian circuits find its application in postman problem,etc.

VI. References

[1] http://www.geom.uiuc.edu/ doty/applications.html [Last accessed on September 1, 2017]

[2] https://prezi.com/9rhagn1jz-n4/euler-circuit-real-life-examples/ [Last accessed on Auguest 30, 2017]

[3] https://www.math.ku.edu/ jmartin/courses/math105-F11/Lectures/chapter5-part2.pdf [Last accessed on August 28, 2017]

[4] http://www.geeksforgeeks.org/eulerian-path-and-circuit/ [Last accessed on August 27, 2017]

[5] http://www.geeksforgeeks.org/fleurys-algorithm-for-printing-eulerian-path/ [Last accessed on September 2, 2017]

[6] http://www.geeksforgeeks.org/hierholzers-algorithm-directed-graph/ [Last accessed on September 2, 2017]