Extracting all the unique circuits from a vertex of a graph

Mudit Rathore[1], Ishani Mishra[2], and Gugloth Lavanya[3]

[1]Indian Institute of Information Technology Allahabad, ICM2015502
[2]Indian Institute of Information Technology Allahabad, IWM2015008
[3]Indian Institute of Information Technology Allahabad, ISM2015003

***Abstract*— A matrix is often an efficient way of representing a graph for analysis. There is a correspondence between many graph-theoretic properties and matrix properties that makes the problem easier to visualize and solve. Matrix representations of graphs are often used to analyze networks, such as those for communications, power distribution, transportation, and shipping. Often it is difficult to employ a computer to analyze a pictorial problem. Since the digital age has set in and all problems are being solved by computers, converting the pictorial problem into a matrix is the need of the hour. Matrix representations are fruitful as they allow easy hand calculations when the problem cannot be solved by inspection. One such matrix is the circuit matrix which is a useful representation of graphs for network analysis and similar problems.**

The step prior to making a circuit matrix, in order to aid our computers, is detecting the circuits present in a graph. This paper details two algorithms to detect all circuits present in a graph. Given a starting vertex and the matrix representation of the graph, the proposed algorithm will output all the circuits beginning from the given vertex.

I. INTRODUCTION

A Graph[1] ***G*** is defined as an ordered pair of a finite set ***V*** of vertices and a finite set ***E*** of edges, where a vertex is a node and an edge is a connection between any two vertices. A graph can be represented using Adjacency Matrix and Incidence Matrix.

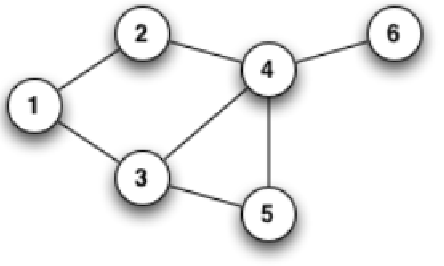

Fig. 1. A simple undirected graph

An important concept in graph theory is the *path*, which is any route along the edges of a graph. A path[2] may follow a single edge directly between two vertices, or it may follow multiple edges through multiple vertices. If there is a path linking any two vertices in a graph, that graph is said to be connected.

A path that begins and ends at the same vertex without traversing any edge more than once is called a *circuit*, or a closed path [1].

For example, circuits of Fig. 1. starting at vertex 1 are:

$$1 - e_{13} - 3 - e_{34} - 4 - e_{42} - 2 - e_{21} - 1$$

and

$$1 - e_{13} - 3 - e_{35} - 5 - e_{54} - 4 - e_{42} - 2 - e_{21} - 1$$

where e_{ij} represents an edge between i and j. Similarly other circuits from the other vertices can be written.

Circuit Matrix[3] Consider a circuit **C** in a connected directed graph **G** with **n** vertices and m edges. This circuit can be traversed one of two directions, clockwise or counter-clockwise. The direction we choose for traversing **C** is called the orientation of **C**. If an edge e=(v_i, v_j) directed from v_i to v_j is in **C** and if v_i appears before v_j as we traverse **C** in the direction specified by the orientation of **C**, then we say that the orientation agrees with the orientation of **e**.

The circuit matrix B_c = $[b_{ij}]$ of **G** has **m** columns, one for each edge, and has one row for each circuit in **G**. The element b_{ij} is defined as: **1** if the j^{th} edge is in the i^{th} circuit and its orientation agrees with the circuit orientation, **-1** if the j^{th} edge is in the i^{th} circuit and its orientation does not agree with circuit orientation and **0** if the j^{th} edge is not in the i^{th} circuit.

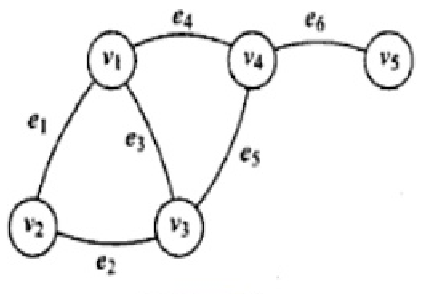

Fig. 2. Example of an undirected graph

Circuits can be of different types. Eulerian and Hamiltonian circuits are two main types.

An Euler path[4], in a graph or multi-graph, is a walk through the graph which uses every edge exactly once. An Euler circuit[4] is an Euler path which starts and stops at the same vertex.

A Hamiltonian cycle[4], also called a Hamiltonian circuit,

$$C = \begin{array}{c|cccccc} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\ \hline 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 1 & 1 & 0 \\ 3 & 1 & 1 & 0 & 1 & 1 & 0 \end{array}$$

Circuit 1 : $\{e_1, e_2, e_3\}$
Circuit 2 : $\{e_3, e_4, e_5\}$
Circuit 3 : $\{e_1, e_2, e_5, e_4\}$

Fig. 3. Circuit Matrix of graph in Fig. 2

Hamilton cycle, or Hamilton circuit, is a graph cycle (i.e., closed loop) through a graph that visits each node exactly once.

II. METHODOLOGY

The paper deals with the details for detecting all the circuits present in a simple graph. We are given a vertex and adjacency matrix representation of the graph, the proposed algorithm will output the all the circuits starting from a given vertex [4].

A. Input

Input from the user is either an incidence matrix or an adjacency matrix representation of the graph followed by the user entering the number of vertices and edges (in case of incidence matrix as input). If incidence matrix is entered, it is converted into its adjacency matrix representation. Then the user specifies the starting vertex for finding the circuits. After which the program prints all the circuits starting from the specified start vertex.

B. Converting incidence matrix into adjacency matrix:

1. Traverse the incidence matrix column-wise.
2. For each column, identify the two row entries for which the matrix cell entries are 1 and 1 (in case of undirected graph) or 1 and -1 (in case of directed graph). Say they are u and v.
3. Update the adjacency matrix (initially filled with zeros) of the corresponding (u,v) to 1.
4. If the graph is undirected, update (u,v) and (v,u) of the adjacency matrix both as 1.
5. Repeat 2, 3, 4 till all columns of the incidence matrix are not seen.

The time complexity of conversion is $O(V * E)$ because all edges are checked for their end vertices in a double loop traversing till $|E|$ and $|V|$ respectively.

C. Algorithm 1:

This algorithm takes start vertex as input along with the adjacency matrix of the graph. Then it extracts all the paths from the chosen adjacent vertex to the start vertex specified. i.e, if our start vertex is A and then vertex adjacent to A are B, C, and D. Then we will identify the various paths from B, C, D to vertex A without considering the edge between vertex A and the chosen vertex adjacent to it. We have used an application of the depth first traversal to solve this problem of extracting the circuits.

The algorithm is a recursive approach to identify the various paths. We repeat the same above approach in order to reach from start vertex to the destination vertex. We change the start vertex in every function call while keeping the destination fixed in order to trace the path. We initialize to boolean array to store the visited state for all the vertices present in the graph and path array to store the vertices in the path from the start vertex to the end vertex initially. The path index keeps the record of the sequence of the encounter of the vertex in the path.

The output for the algorithm for the graph in Fig. 1 (A simple undirected graph) with start vertex A are:
1. 1 2 4 5 3 1
2. 1 2 4 3 1
3. 1 3 4 2 1
4. 1 3 5 4 2 1

Here we can see that path represented by 1, 4 and 2, 3 are same circuit. They just the differ in representation.

The complexity of the proposed algorithm is $O(V * (V + E))$. The space complexity is $O(|V * V|)$ for the adjacency Matrix and $O(|V|)$ for the path and visited boolean matrix. Hence giving the space matrix of $O(|V * V|)$.

Algorithm 1 Extract Circuits

```
procedure MAIN()
    $StartVertex \leftarrow s$
    for each i $\in$ G.adjacency[s] do
        ExtractAllPaths($i, s$)
    end for
end procedure

function EXTRACTALLPATHS($s, d$)
    visited[ ] $\leftarrow$ false
    path[] // array of size V
    path_index $\leftarrow$ 0
end function

function EXTRACTALLPATHSUTIL($s, d, visited, path, path\_index$)
    visited[s] $\leftarrow$ true
    path[path_index] $\leftarrow$ s
    path_index ++
    if s == d and path_index > 2 then
        Print Output Circuit
    else
        for each i $\in$ G.adjacency[s] do
            if !(visited[i]) then
                ExtractAllPathUtil(i,d,visited,path,path_index)
            end if
        end for
    end if
    path_index −−
    $visited[s] \leftarrow false$
end function
```

In the next subsection we propose a modified version of the algorithm in order to extract the unique circuits starting from a specified vertex.

D. *Algorithm 2:*

This is the modified version of the algorithm proposed above, it overcomes the limitation of redundant paths with different representations. This algorithm resolves the problem of finding unique circuits in the graph starting at a particular vertex. We have used a boolean array, extracted of size $|V|$ (number of vertices) to keep track of all those vertices whose circuits have already been taken in consideration for the presence of the circuit. Thus, when we encounter same vertex while finding the circuit to destination from another adjacent vertex, the recursive call breaks the approach with current set to proceed, printing only the unique circuits.

The time complexity of this proposed algorithm is also $O(V * (V + E))$. The space complexity is $O|(V * V)|$ for the adjacency matrix representation of the graph and $O(|V|)$ for each of the extracted array, path array and visited array, hence giving the overall space complexity of $O(|V * V|)$.

Algorithm 2 Extract Circuits

```
1: procedure MAIN()
2:     $StartVertex \leftarrow s$
3:     $extracted[] \leftarrow false$//array of size V
4:     for each i $\in$ G.adjacency[s] do
5:         ExtractAllPaths($i, s$)
6:         $extracted[i] \leftarrow true$
7:     end for
8: end procedure

9: function EXTRACTALLPATHS($s, d$)
10:     visited[ ] $\leftarrow$ false
11:     path[] // array of size V
12:     path_index $\leftarrow$ 0
13: end function

14: function EXTRACTALLPATHSUTIL($s, d, visited, path, path\_index$)
15:     visited[s] $\leftarrow$ true
16:     path[path_index] $\leftarrow$ s
17:     path_index ++
18:     if  s == d and path_index > 2  then
19:         Print Output Circuit
20:     else
21:         for each i $\in$ G.adjacency[s] do
22:             if !(visited[i]) and !(extracted[i]) then
23:                 ExtractAllPathUtil(i,d,visited,path,path_index)
24:             end if
25:         end for
26:     end if
27:     path_index −−
28:     visited[s] $\leftarrow$ false
29: end function
```

E. *Complexity Analysis*

1) Algorithm 1: The complexity of the proposed Algorithm 1 is $O(V * (V + E))$. The space complexity is $O(|V * V|)$ for the adjacency Matrix and $O(|V|)$ for the path and visited Boolean matrix. hence giving the space matrix of $O(|V * V|)$.

2) Algorithm 2: The time complexity of this proposed Algorithm 2 is also $O(V * (V + E))$. The space complexity is $O|(V * V)|$ for the adjacency matrix representation of the graph and $O(|V|)$ for each of the extracted array, path array and visited array, hence giving the overall space complexity of $O(|V * V|)$.

F. *Runtime Analysis*

Figure is a plot is showing the runtime analysis when the number of vertices were varied, while keeping the number of edges fixed to values 45 and 50.The time in seconds, is the time taken to extract all the unique circuits present in the graph. Figure is a plot is showing the runtime analysis when

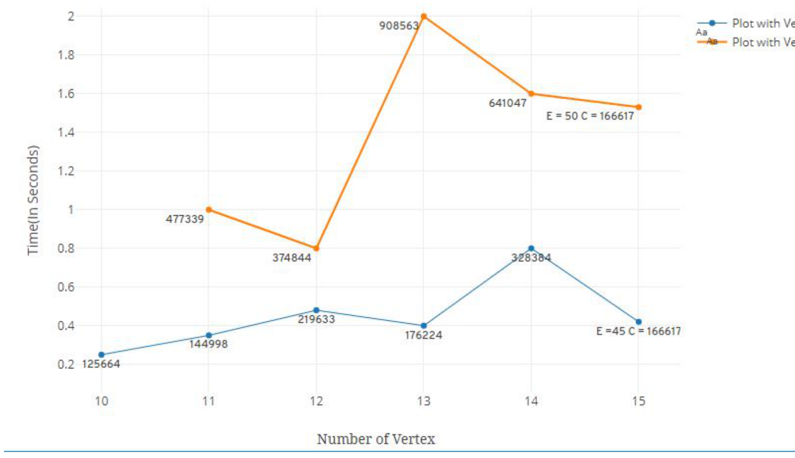

Fig. 4. Plot of Time V/s Number of Vertex(Number of Edge constant with 45 and 50)

the number of edges were varied, while keeping the number of vertex fixed to values 10 and 12.The time in seconds, is the time taken to extract all the unique circuits present in the graph.

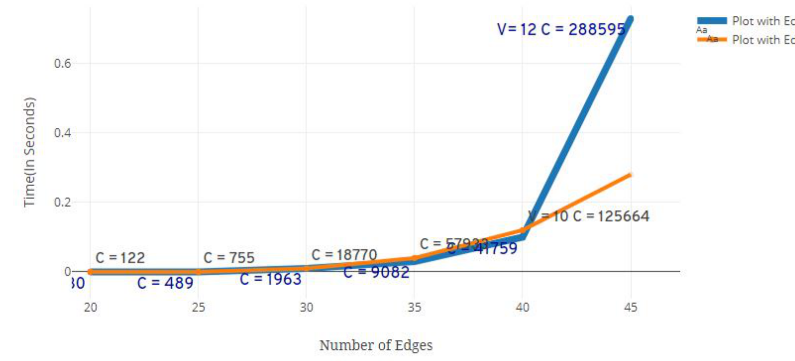

Fig. 5. Plot of Time V/s Number of Edge(Number of vertex constant with 10 and 12)

III. RESULTS

The proposed algorithm prints all circuits from a user specified starting vertex.

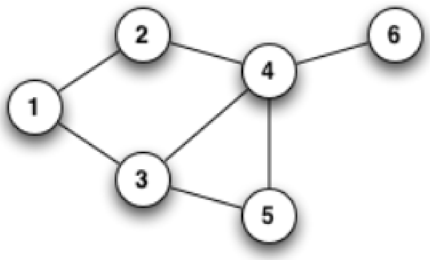

Fig. 6. A simple undirected graph

1) Input: Undirected graph adjacency matrix: Enter the type of matrix. Enter 1 for incidence, 2 for adjacency matrix: 2
Enter total number of vertices: 6
Enter the type of the graph. Enter 1 for directed or 2 for undirected: 2
Enter the (n x n) Adjacency Matrix:
0 1 1 0 0 0
1 0 0 1 0 0
1 0 0 1 1 0
0 1 1 0 1 1
0 0 0 1 0 0
Enter the starting vertex of the circuit: 1
The circuits from vertex 1 are:
1 2 4 5 3 1
1 2 4 3 1
1 3 4 2 1
1 3 5 4 2 1

2) Input: Undirected graph incidence matrix: Enter the type of matrix. Enter 1 for incidence, 2 for adjacency matrix: 1
Enter total number of vertices: 6
Enter the total number of edges in your graph: 7
Enter the type of the graph. Enter 1 for directed or 2 for undirected: 2
Enter the (n x e) Incidence Matrix:
1 1 0 0 0 0 0
1 0 1 0 0 0 0
0 1 0 1 1 0 0
0 0 1 1 0 1 1
0 0 0 0 1 1 0
0 0 0 0 0 0 1
Enter the starting vertex of the circuit: 1
The circuits from vertex 1 are:
1 2 4 5 3 1
1 2 4 3 1
1 3 4 2 1
1 3 5 4 2 1

3) Input: Directed graph adjacency matrix: Enter the type of matrix you want to traverse. Enter 1 for incidence, 2 for adjacency matrix: 2
Enter total number of vertices: 7
Enter the type of the graph. Enter 1 for directed or 2 for

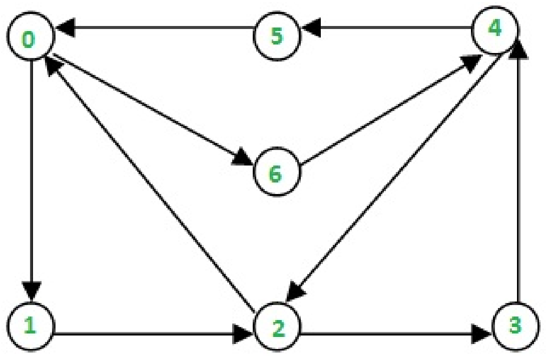

Fig. 7. Input directed graph

undirected: 1
Enter the (n x n) Adjacency Matrix:
0 1 0 0 0 0 1
0 0 1 0 0 0 0
1 0 0 1 0 0 0
0 0 0 0 1 0 0
0 0 1 0 0 1 0
1 0 0 0 0 0 0
0 0 0 0 1 0 0
Enter the starting vertex for traversal: 0
The circuits from vertex are:
0 6 4 5 0
0 6 4 2 0
0 1 2 0
0 1 2 3 4 5 0

4) Input: Directed graph incidence matrix: Enter the type of matrix you want to traverse. Enter 1 for incidence, 2 for adjacency matrix: 1
Enter total number of vertices: 7
Enter the total number of edges in your graph: 10
Enter the type of the graph. Enter 1 for directed or 2 for undirected: 1
Enter the (n x e) Incidence Matrix:
1 0 -1 0 1 0 -1 0 0 0
0 0 0 0 0 0 1 -1 0 0
0 0 0 0 -1 1 0 1 -1 0
0 0 0 0 0 0 0 0 1 -1
0 -1 0 1 0 -1 0 0 0 1
-1 1 0 0 0 0 0 0 0 0
0 0 1 -1 0 0 0 0 0 0
Enter the starting vertex for traversal: 0
The circuits from vertex are:
0 6 4 5 0
0 6 4 2 0
0 1 2 0
0 1 2 3 4 5 0

IV. DISCUSSION

The results obtained above are the outputs of the 2^{nd} algorithm. We chose the 2^{nd} algorithm over the 1^{st} because in the output of the 1^{st} algorithm, we observed that that path represented by 1, 4 and 2, 3 were same circuits and just differed in representation. 2^{nd} algorithm overcomes this limitation of redundant paths with different representations. This algorithm proposes a solution to the problem of finding unique circuits in the graph starting at a particular vertex.

REFERENCES

[1] J. Cook, M. Wootters, V. Williams, R. Verma, S. Hildick-Smith, and G. Valiant, *Graphs.* [On-line]. Available: https://web.stanford.edu/class/cs161/Lectures/Lecture9/CS161Lecture09.pdf. [2017]

[2] N. Deo. *Graph Theory with applications to engineering and computer science*. Prentice-Hall of India, 2007, pp. 21-30.

[3] K. Thulasiraman. *Circuit Theory.* [On-line]. Available: http://www.cs.ou.edu/ thulasi/Misc/circuittheory.pdf. [2002]

[4] G. Danielson. *On Finding the Simple Paths and Circuits in a Graph* IEEE Transactions on Circuit Theory, Volume: 15, Issue: 3,September 1968, pp. 264-295)