Extraction of All Circuits Starting and Ending at Specified Vertices

Ajay Saini
Dept. of Information Technology,
Indian Institute of Information Technology, Allahabad

Ayonya Prabhakaran
Dept. of Information Technology,
Indian Institute of Information Technology, Allahabad

Ankit Mund
Dept. of Information Technology,
Indian Institute of Information Technology, Allahabad

***Abstract*—The paper details the algorithm to detect all circuits present in an undirected graph. Given a vertex and the adjacency list representation of the graph, the proposed algorithm will output all the circuits beginning from the given vertex.**

***Index Terms*—circuits, graphs, adjacency list, vertices, edges**

I. Introduction

A graph is an ordered pair G = (V, E) comprising a set V of vertices, with a set E of edges. The graph can be represented using an adjacency list. A closed walk consists of a sequence of vertices starting and ending at the same vertex, with each two consecutive vertices in the sequence adjacent to each other in the graph. A circuit is a closed walk allowing repetitions of vertices but not edges. However, it can also be a simple cycle i.e., a closed walk with no repetitions of vertices and edges allowed. This paper will be brief on the algorithm proposed to detect circuits present the the graph given the start vertices. Two variations of the algorithm are presented below.

II. Methods/ Algorithm

A. Input

The user inputs the total number of vertices and the adjacency list representation of the undirected graph. The start vertex to detect the circuits is also specified. The first version of the algorithms prints the repeated circuits while the improved version prints only the unique circuits.

B. Algorithm I

The first version of the algorithm prints the repeated circuits as in from Fig. 1 we observe the list of circuits beginning from vertex 0 are:

1) 0 1 3 2 0
2) 0 1 3 4 2 0
3) 0 2 4 3 1 0
4) 0 2 3 1 0

We observe from the list that items 1 & 3 and items 2 & 4 are the same circuits. In next section, the modified algorithm proposed, solves this drawback.

The depth first traversal method is applied to solve the problem of extracting circuits. Begin the traversal from the

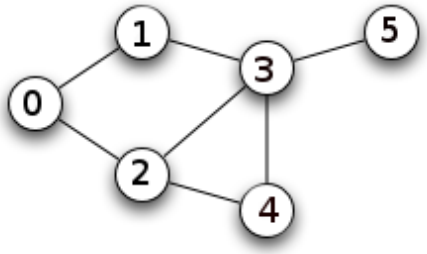

Fig. 1. An undirected graph containing 6 vertices and two circuits that begin from vertex 0.

Algorithm 1 Extract Circuits

```
procedure MAIN
    input start vertex: s
    for each i ∈ G.Adj[s] do
        ExtractAllPaths(i, s)

function EXTRACT-ALL-PATHS(s, d)
    visited[] ← false // initilize bool array of size|V|
    path[] // allocate bool array of size|V|
    path_index ← 0
    ExtractAllPathsUtil(s, d, visited, path, path_index)

function EXTRACT-ALL-PATHS-UTIL(s, d, visited, path,
path_index)
    visited[s] ← true
    path[path_index] ← s
    path_index + +
    if s == d and path_index > 2 then
        Print the output circuit
    else
        for each i ∈ G.Adj[s] do
            if !(visited[i]) then
                ExtractAllPathsUtil(i, d, visited, path, path_index)
    path_index − −
    visited[s] ← false
```

start vertex provided as input by the user. The pseudo code of the algorithm ***Extract Circuits*** is given below Fig. 1.

The time complexity of the above algorithm is $O(V*(V+E))$. The space complexity is $|(V+E)|$ for the *adjacency list* and $|V|$ each for the *path* and *visited* arrays hence giving a space complexity $O(V+E)$.

C. Algorithm II

The modified version of the above algorithm, ***Extract Unique Circuits*** prints only the unique circuits present in the graph. A boolean data structure *extracted* of size $|V|$ is used to keep track of the vertices whose circuits have already been extracted. Thus, when the same vertex is obtained while finding the circuit from another adjacent vertex, the recursive call breaks printing only unique circuits. The time complexity of this method is also $O(V*(V+E))$. The space complexity is $|(V+E)|$for the *adjacency list* and $|V|$ each for the *extracted, path* and *visited* arrays hence giving a space complexity $O(V+E)$.

Algorithm 2 Extract Unique Circuits

```
procedure MAIN
    input start vertex: s
    extracted[] ← false //initialize bool array of size|V|
    for each i ∈ G.Adj[s] do
        ExtractAllPaths(i, s)
        extracted[i] ← true

function EXTRACT-ALL-PATHS(s, d)
    visited[] ← false //initilize bool array of size|V|
    path[] //allocate bool array of size|V|
    path_index ← 0
    ExtractAllPathsUtil(s, d, visited, path, path_index)

function EXTRACT-ALL-PATHS-UTIL(s, d, visited, path,
path_index)
    visited[s] ← true
    path[path_index] ← s
    path_index++
    if s == d and path_index > 2 then
        Print the output circuit
    else
        for each i ∈ G.Adj[s] do
            if !(visited[i]) and !(extracted[i]) then
                ExtractAllPathsUtil(i, d, visited, path,
                                    path_index)
    path_index−−
    visited[s] ← false
```

D. Runtime Analysis

Figure 2 is a plot depicting the run time analysis when the number of vertices were varied, while keeping the number of edges fixed to values 45 and 50 in two cases as shown in the plot. The time in seconds, is the time taken to extract all the unique circuits present in the randomly generated graphs. The number of circuits extracted is printed above the line and varies depending on the nature of the graph.

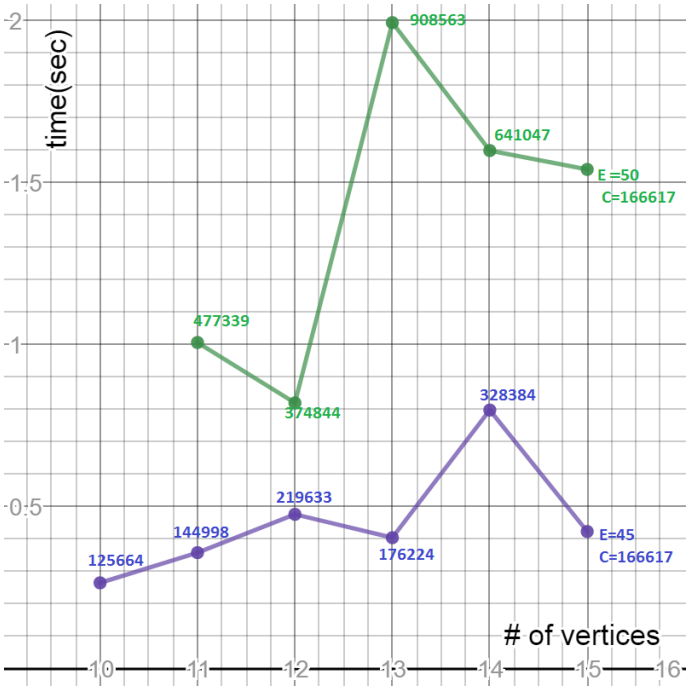

Fig. 2. Plot of time vs. number of vertices(with number of edges fixed).

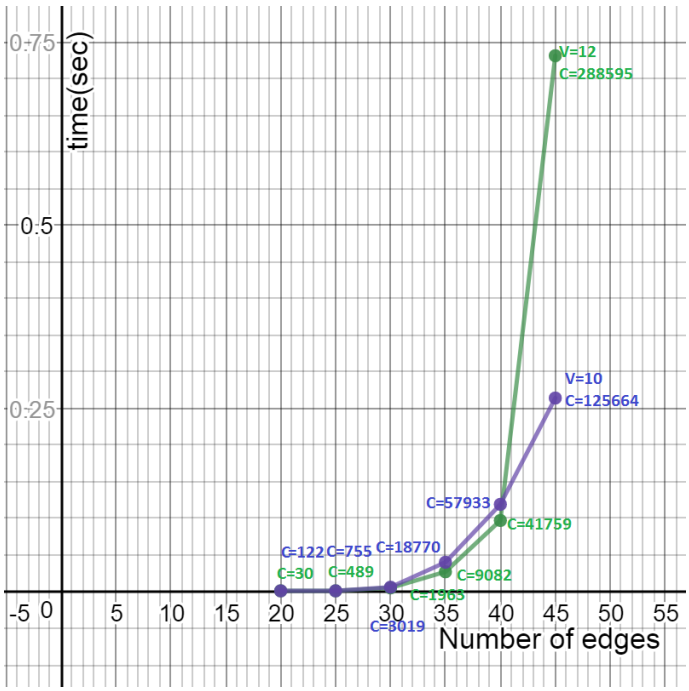

Fig. 3. Plot of time vs. number of edges(with number of vertices fixed).

Figure 3 is a plot depicting the run time analysis when the number of edges were varied, while keeping the number of

vertices fixed to values 10 and 12 in two cases as shown in the plot. The time in seconds, is the time taken to extract all the unique circuits present in the randomly generated graphs. The number of circuits extracted is printed above the line and varies depending on the nature of the graph.

III. Conclusion

The above paper details the implementation of the algorithms which solves the problem of extraction of all circuits in a graph given the starting vertex. The solution has been optimized to print only unique paths, using optimum space and time possible.

References

[1] Stack Overflow (https://stackoverflow.com)
[2] Math Stack Exchange(http://math.stackexchange.com)
[3] CS Stack Exchange(https://cs.stackexchange.com)
[4] Geeks For Geeks (http://www.geeksforgeeks.org/)
[5] Google Images (https://images.google.com/)
[6] C Plus Plus Reference(http://www.cplusplus.com)
[7] Wikipedia (https://en.wikipedia.org/)
[8] Wolfram Mathworld (http://mathworld.wolfram.com/)