Finding at least 3 isomorphic representation for the same circuit matrix

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*—In this paper, we suggested an algorithm to find atleast 3 isomorphic graphs gi(Vi,Ei) from the given circuit matrix. The time complexity for the algorithm suggested is exponential for N is number of circuits and M is number of edges.**

***Index Terms*—Isomorphism . Circuit matrix.**

I. Introduction

Given a graph G(V,E), the circuit matrix C[N][M] where N is the number of circuits and M is the number of edges present in graph. Each of the entries in circuit matrix tells whether a particular edge is present in this circuit or not i.e. C[i][j] = 1 if ith circuit has jth edge in it and C[i][j] = 0 if ith circuit doesnt have jth edge.

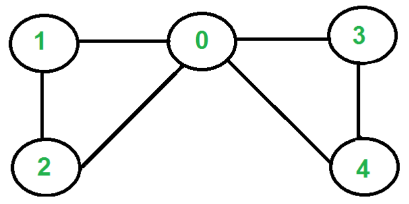
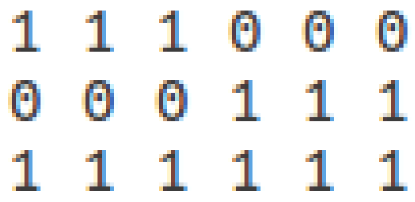


Fig. 1. A graph G(V,E).



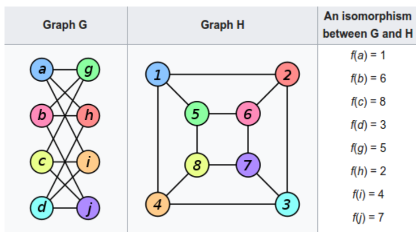
$$\begin{array}{cccccc} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{array}$$

Fig. 2. Circuit matrix for G(V,E) in fig.1

For graph given in fig. 1, the circuit matrix is provided in fig. 2.

Two graphs are isomorphic if these two have same number of graph vertices connected in the same manner. In graph theory, an isomorphism of graphs G and H is a bijection between the vertex sets of G and H, : V(G) -> V(H), such that any two vertices u and v of G are adjacent in G if and only if (u) and (v) are adjacent in H as shown in fig.3. This kind of bijection is commonly described as ”edge-preserving bijection”, in accordance with the general notion of isomorphism being a structure-preserving bijection.

If an isomorphism exists between two graphs, then the graphs are called isomorphic. Graph isomorphism is an equivalence relation on graphs and as such it partitions the class of all graphs into equivalence classes. A set of graphs isomorphic to each other is called an isomorphism class of graphs. Two graphs G and H with graph vertices V = 1, 2,...., n are said to be isomorphic if there is a permutation P of V such that u,v is in the set of graph edges E(G) iff P(u),P(v) is in the set of graph edges E(H).



An isomorphism between G and H
f(a) = 1
f(b) = 6
f(c) = 8
f(d) = 3
f(g) = 5
f(h) = 2
f(i) = 4
f(j) = 7

Fig. 3. Isomorphism with mapping f

II. Motivation

The formal notion of ”isomorphism”, e.g., of ”graph isomorphism”, captures the informal notion that some objects have ”the same structure” if one ignores individual distinctions of ”atomic” components of objects in question. Whenever individuality of ”atomic” components (vertices and edges, for graphs) is important for correct representation of whatever is modeled by graphs, the model is refined by imposing additional restrictions on the structure, and other mathematical objects are used: digraphs, labeled graphs, colored graphs, rooted trees and so on. The isomorphism relation may also be defined for all these generalizations of graphs: the isomorphism bijection must preserve the elements of structure which define the object type in question: arcs, labels, vertex/edge colors, the root of the rooted tree, etc.

The notion of ”graph isomorphism” allows us to distinguish graph properties inherent to the structures of graphs themselves from properties associated with graph representations:

graph drawings, data structures for graphs, graph labelings, etc.

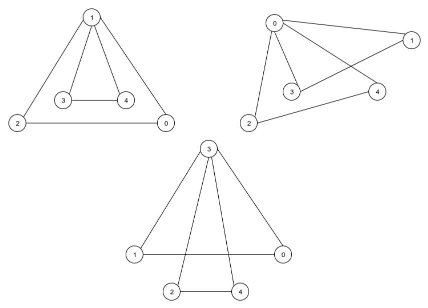


Fig. 4. Isomorphs from the circuit matrix in fig.2

```
cir_mat = Given circuit matrix
edge_map = initialised empty mapping of edges with end points

find_all_circuits(){
    cycles = find_all_cycles()
    circuits = initialised as an empty set
    for(each subset s in cycles){
        (c1,c2....,ck) = subset s
        if(all ci are connected to each other){
            circuits = circuits U (c1 U c2 U.....U ck)
        }
    }
    return circuits
}

check(){
    cir = find_all_circuits();
    for(each c in cir){
        if(c is not present in cir_mat)
            return 0
    }
    return 1
}
```

Fig. 5. Algorithm 1

```
construct(node,edge){
    if(all edges considered)
        return check();
    for(int i=1;i<node;i++){
        for(int j=i+1;j<=node;j++){
            edge_map[(i,j)] = edge
            next = node
            if(j==node)
                next = node+1
            if(check(next,edge+1))
                return 1;
            edge_map = edge_map - (i,j)
        }
    }
    return 0
}
```

Fig. 6. Algorithm 2

III. Methods and Descriptions

In this paper, we introduced an approach for the construction of different isomorphic representations from the same circuit matrix. In both the approaches, we will first construct a simple graph from the given circuit matrix. Then we will permute the node values to construct other isomorphic forms of the same graph constructed. The first part will run in time complexity of exponential and second part will take constant time because we will just find few isomorphs. These isomorphs can be shown in different visual representations. Using this, the results for the graph in fig.1 is shown in fig. 4.

```
isomorphic_representations(){
    isomorphs = empty set
    for(each permutation of nodeids in edge_map){
        temp = graph with new permutation
        isomorphs = isomorphs U temp
    }
    return isomorphs
}

main(){
    e = number of edges
    c = number of circuits
    edge_map.clear()
    edge_map[(1,2)] = 1
    construct(3,2)
    iso = isomorphic_representations(edge_map)
}
```

Fig. 7. Algorithm 3

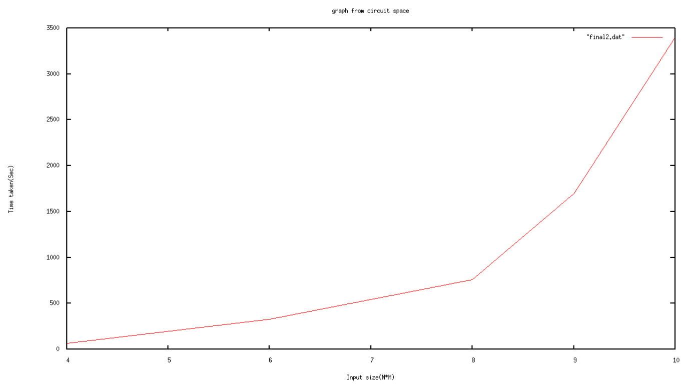


Fig. 8. Input size vs time complexity

IV. Implementation and Results

The algorithm explained previously has been implemented as shown in algorithm 1,2,3.

The graph for size (N*M) vs time complexity has been given as in fig.5. This clearly shows how time of computation increases with the increase in the value of N*M.

V. Conclusion

In this paper, we presented an algorithm to construct at least 3 isomorphic representations using the same circuit matrix. This was done in two steps. First a simple graph was constructed using the given circuit matrix. After that, the nodes were assigned different ids for different representations and then these were visually shown in different forms. The overall time complexity was exponential.

VI. References

[1] https://en.wikipedia.org/wiki/Graphisomorphism [Last accessed on October 27, 2017]

[2] http://www.cs.ou.edu/ thulasi/Misc/circuittheory.pdf [Last accessed on October 27, 2017]

[3] http://www.edmath.org/MATtours/discrete/concepts/ciso.html [Last accessed on October 28, 2017]