Checking Isomorphism Between Two Graphs

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

Group - D

***Abstract*— Consider two graphs G and H with given Adjacency Matrices A(G) and A(H).In this paper we propose an algorithm to find whether the given graphs are isomorphic or not . Further we evaluate the proposed algorithms time complexity by the help of a plot of Time vs Number of nodes.We also provide an improved algorithm which makes use of parallelization to achieve the same result faster.**

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 **V** represents a point or node and an edge e **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) Permutation Matrix

In mathematics, particularly in matrix theory, a permutation matrix is a square binary matrix that has exactly one entry of 1 in each row and each column and 0s elsewhere. Each such matrix, say P, represents a permutation of m elements and, when used to multiply another matrix, say A, results in permuting the rows (when pre-multiplying, i.e., PA) or columns (when post-multiplying, AP) of the matrix A.

C) Graph Isomorphism

Two graphs which have a similar structure visually on rearrangement of their vertices are said to be isomorphic. Formally, two graphs G and H with graph vertices $V_{n= 1,2,...,n}$ are said to be isomorphic if there is a permutation p of V_n such that u,v is in the set of graph edges **E(G)** if and only if p(u),p(v) is in the set of graph edges **E(H)**.

In a more formal way , in graph theory, an isomorphism of graphs **G** and **H** is a bijection between the vertex sets of G and H f : V(G) V(H) such that any two vertices u and v of G are adjacent in G if and only if f(u) and f(v) are adjacent in H. This kind of bijection is commonly described as edge-preserving bijection, in accordance with the general notion of isomorphism being a structure-preserving bijection.

The isomorphism problem consists of finding a mapping from the vertices of graph G to graph H such that they are identical. Such a mapping is called an isomorphism. If an isomorphism exists between two graphs, then the graphs are called isomorphic.
Fig. 3 shows two isomorphic graphs and the corresponding isomorphic function that exists between them .

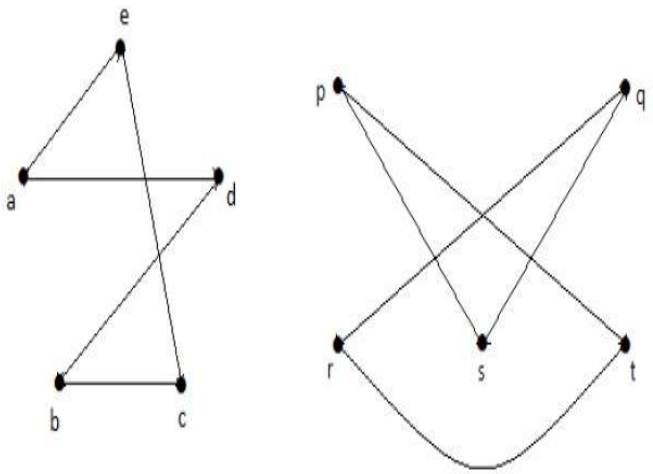

Fig. 1. Example of Isomorphic Graphs

Bijection: In mathematics, a bijection, bijective function or one-to-one correspondence is a function between the elements of two sets, where each element of one set has exactly one corresponding element in the other set, and each element of the other set is paired has exactly one corresponding element in the first set. There are no elements without a corresponding element in the other set. In mathematical terms, a bijective function f: X Y is a one-to-one (injective) and onto (surjective) mapping of a set X to a set Y.

II. MOTIVATION

It is not an easy task to determine whether or not 2 graphs are isomorphic.Graph isomorphism is still a largely

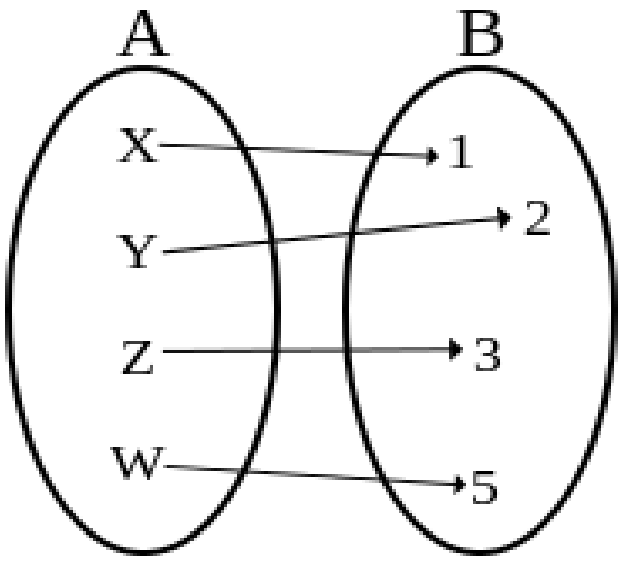

Fig. 2. Example of Bijective Function

researched problem in the field of graph theory and otherwise as well.Till date,a polynomial time algorithm to check whether or not two graphs are isomorphic has not been determined. For two graphs to be isomorphic,

- The number of vertices should be the same.
- The number of edges should be the same.
- The number of vertices n_i. with degree i should be the same in both, for i = 1,2, . . . n.
- The characteristic polynomials representing their adjacency matrices **X(G)** and **X(H** should be the same.

Although these conditions are necessary for two graphs to be isomorphic,they do not guarantee isomorphism between two graphs when satisfied.

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.

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. For example, if a graph has exactly one cycle, then all graphs in its isomorphism class also have exactly one cycle. On the other hand, in the common case when the vertices of a graph are (represented by) the integers 1, 2,... N, then the expression
$O(\sum_{v \in V(G)} v * degree(v))$ may be different for two isomorphic graphs.

III. METHODS AND DESCRIPTION

It is always possible to determine whether or not two graphs G1 and G2 are isomorphic by keeping G1, fixed and reordering vertices of G2 to check if their adjacency matrices become identical. This process may require all n reordering and comparisons, being the number of vertices. Such an inefficient procedure, in which the running time grows factorially with ,is of limited use for practical problems.

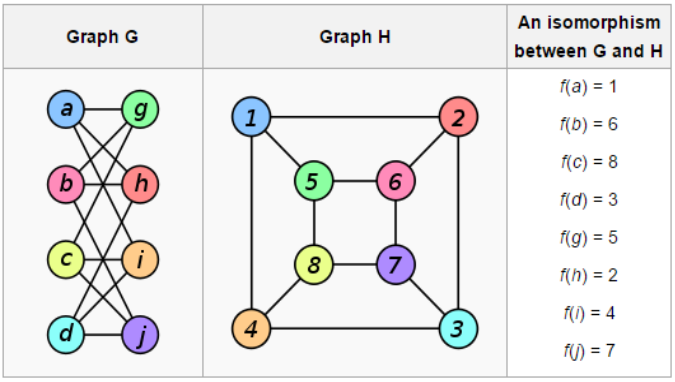

Fig. 3. Bijective Function

IV. PROPOSED ALGORITHMS

Algorithm 1 Check Isomorphism

```
if Number of vertices are different then
    Not Isomorphic
else if Number of edges are different then
    Not Isomorphic
else
    for all vertex in graph G do
        Calculate degree in dG
    end for
    for for all vertex in graph H do
        Calculate degree in dH
    end for
    sort dG and dH
    if dG and dH are different then
        Not Isomorphic
    else
        initialize ar with 0 to n-1
        for all permutations of ar do
            generate per such that all entries are 0 except
            per[i][ j]=1 where ar[i]= j
            generate transpose of per in perT
            if per H perT = G then
                Isomorphic
                return
            end if
        end for
    end if
end if
```

V. IMPLEMENTATION

We have used C++ to accept two graphs **G** and **H** from the user which are then converted into the corresponding adjacency matrices **X(G)** and **X(H)**.Following this we create a permutation matrix **per** which is essentially a mapping between the vertices of graphs G and graph H.We test all

such permutations.
Having generated the permutaton matrix,we multiply **per** with **X(G)** and transpose of **per**,**perT**.If this equals **X(H)** for any of the permutation matrices,the graphs are considered to be isomorphic.
Multiplying **per** with **X(G)** permutes the rows and multiplying with **perT** permutes the columns. In order to fasten the process of matrix multiplication we have written the same code using tensor flow as well.This code primarily converts each permutation matrix into tensors which in turn can used to carry out the matrix multiplication in parallel with the help of a GPU.The python code was run using Google Colab.

VI. RESULTS

In case for the ordinary code the time complexity of checking isomorphism between two graphs is $O(N! * N^3)$ however when we use the multi-threaded code,the matrix multiplication always takes $O(1)$ time irrespective of the size of the matrix. Thus the overall time complexity is reduced 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`

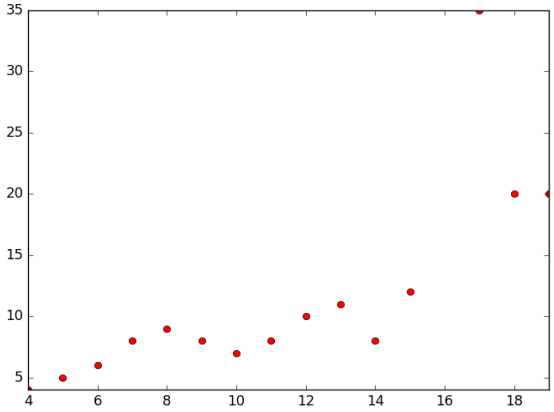

Fig. 4. Execution time in case edges between two graphs are different(same number of vertices)

The above graph is a time vs vertices graph which shows that as the number of vertices increases,the time taken to determine the fact that two graphs have different number of edges also increases.

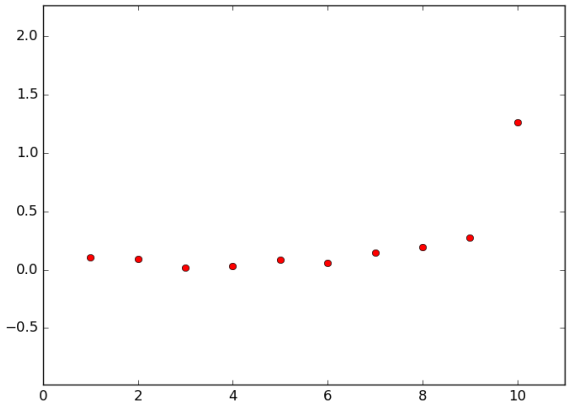

Fig. 5. Time vs vertex graph for parallel multi-threaded code

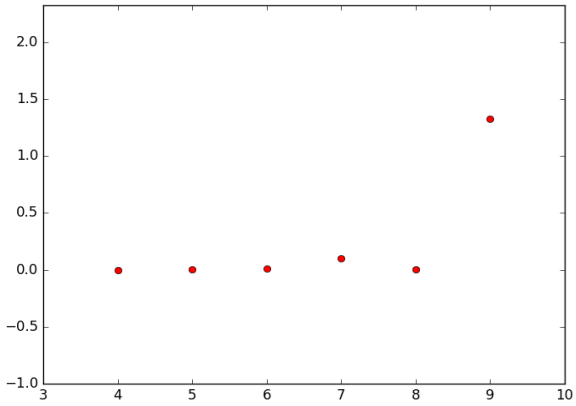

Fig. 6. Time vs vertex graph for normal code

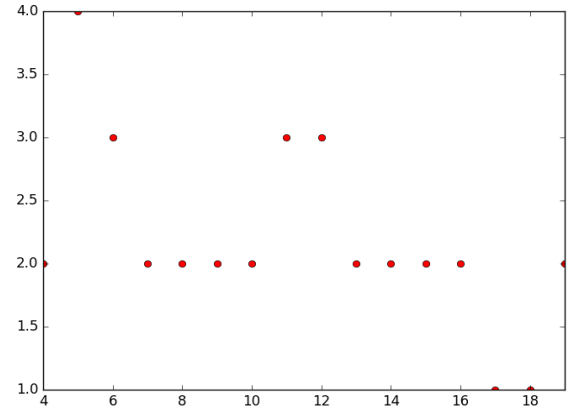

Fig. 7. Execution time in case the vertices between two graphs are different

The above graph is a time vs vertex graph which shows that time taken to determine the fact that two graphs have different number of vertices fluctuates around a constant value.

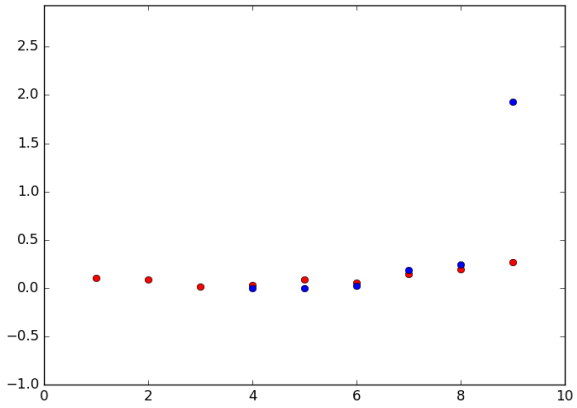

Fig. 8. Comparison of normal code with parallel code.

NOTE:-In Fig.8 red dots represent the multi threaded run time whereas blue dots represent the normal code run time

VII. CODE SNIPPETS

```
def P_A_Pt(a,arr):

  a = tf.convert_to_tensor(a,tf.float32)

  at = tf.transpose(a)

  ans = tf.multiply(arr,a)
  ans1 = tf.multiply(at,ans)

  return ans1
```

Fig. 9. Multiplying matrices in parallel using tensor flow

The function shown in Fig.9 performs matrix multiplication in parallel by converting each matrix into tensors.It calculates the product **per** * **X(G)** * **perT**.

```
def checkIsomorphism(g,h):

  perms = list(permutations(range(n)))
  for a in perms:
    a = createPerm(a)
    #z = [[0,0],[0,0]]
    #a = np.asarray(z)
    x = P_A_Pt(a,g)
    equal = tf.equal(x,h)
    equal = tf.reduce_all(equal)
    with tf.Session() as sess:
      equal = sess.run(equal)
      if(equal):
        print("equal")
        return
  print("not equal")
```

Fig. 10. Checking equality of matrices in parallel

The function shown in Fig.10 checks for equality of the matrix returned by the function in Fig.9 and **X(H)** by comparing each element of the two matrices in parallel.

```
void multiply(vvi &a, vvi &b, vvi &ans, int n){
      int i, j, k, x;
      ans.clear();
      for(i=0;i<n;i++){
            vi tmp;
            for(j=0;j<n;j++){
                  x=0;
                  for(k=0;k<n;k++)
                        x+=a[i][k]*b[k][j];
                  tmp.pb(x);
            }
            ans.pb(tmp);
      }
}
```

Fig. 11. Multiplying matrices serially

The function shown in Fig.11 performs the matrix multiplication serially and has a time complexity of $O(N^3)$

Analysis and comparison of algorithms :-
The *best case* time complexity for the normal code is $O(N^3)$.This occurs when the first permutation being tested is the correct permutation and the graphs are proved to be isomorphic.

The *worst case* tine complexity for the normal code is $O(N! * N^3)$.This occurs when the correct permutation matrix is encountered as the last permutation being tested.
The *best case* time complexity for the parallel code is $O(1)$.This occurs when the first permutation being tested is the correct permutation and the graphs are proved to be isomorphic.
The *worst case* time complexity for the parallel code is :- $O(N!)$.This occurs when the correct permutation matrix is encountered as the last permutation being tested.

VIII. APPLICATIONS

The concept of graph isomorphism is widely used in many fields in the real life.A few examples are :-

- **Medicine :-**Isomorphism is widely used in the field of medicine to find compounds having similar structure.This concept is used to recommend alternate medications and find suitable replacement of compounds.
- **Social Networking :-**Isomorphism can be used to find people having similar interests.
- **Image Processing :-**Isomorphism can be used here to check whether or not two images have the same inherent structure.
- **Object Differentiation :-**Isomorphism can be used to distinguish objects based on their structure.This can be used to automate a variety of tasks.

IX. DISCUSSION

In this paper we used the given algorithm to determine whether or not two graphs are isomorphic.The algorithm generates permutations of one adjacency matrix to determine whether or not it equals the other adjacency matrix. We have also used tensorflow to fasten the process of matrix multiplication which in turn yields a significant improvement in run time of the code.

X. CONCLUSIONS

We can determine whether two graphs are isomorphic or not using the methods suggested in the paper.However,a polynomial time algorithm for the same is not yet possible.Algorithms running parallel multiplication of matrices yield a significant improvement in run time.

REFERENCES

[1] Narsingh Deo, Graph Theory with Applications to Engineering and Computer Science.
[2] www.uow.edu.au/ bmaloney/wuct121/GraphsWeek10Lecture2.pdf
[3] Jonathan L. Gross, Jay Yellen, Handbook of Graph Theory.
[4] www.wikipedia.org/wiki/Bijection
[5] http://mathworld.wolfram.com/IsomorphicGraphs.html
[6] https://www.tensorflow.org/tutorials/
[7] https://www.youtube.com/watch?v=UCle3Smvh1s
[8] https://pdfs.semanticscholar.org/12e3/d2640d2a5255666a91d3fb1be20ccebdec69.pdf
[9] https://www8.cs.umu.se/kurser/TDBAfl/VT06/algorithms/BOOK/BOOK4/NODE180.HTM