Graph Isomorphism Problem : Status and Perspective

Rithin Manoj,Sri Charan Battu
IIT Kharagpur
rithin.mr@gmail.com,sricharanbattu@gmail.com

Abstract

An isomorphism between two graphs is a connectivity preserving bijective mapping between their sets of vertices. Finding isomorphisms between graphs, or between a graph and itself (automorphisms), is of great importance in applied sciences.Here we go through the status of one of most famous problems in computer science The graph isomorphism Problems,the recent advances in it and efficient algorithms for some special classes of graphs by delving deeper into the problem

1 Introduction

A graph can exist in different forms having the same number of vertices, edges, and also the same edge connectivity. Such graphs are called isomorphic graphs. Two Graphs G(V,E) and H(W,F) are said to be isomorphic if there is a bijective function $f : V \to$ W such that for all u, v in V : $(u, v)\epsilon E \Leftrightarrow (f(u), f(v))\epsilon F$

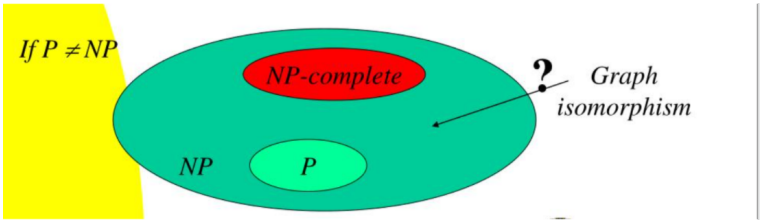
The graph isomorphism problem does not focus solely on checking if two graphs are isomorphic or not.Rather, the question is whether there is an algorithm that is faster than the ones that are known. Mathematicians have a way of ranking problems according to their difficulty (more precisely, their complexity). The graph isomorphism problem asks where in the hierarchy of complexity classes the graph question belongs

1.1 Notable Personalities

Eugene Michael Luks is an American mathematician and computer scientist, a professor emeritus of computer and information science at the University of Oregon.He Worked with Babai to create lowest theoretical bound algorithm(1983)
László "Laci" Babai is a Hungarian professor of computer science and mathematics at the University of Chicago .He Created Quasi Polynomial time algorithm(which is being reviewed), involving group theory and formulations of his own(2015).
Brendan Damien McKay (born 26 October 1951 in Melbourne, Australia) is an Emeritus Professor in the Research School of Computer Science at the Australian National University (ANU).He co authored a paper which gave a practical algorithm for graph Isomorphism Problem. He also created a software implementation of the same,which is still one of the best practical implementation of graph isomorphism.

1.2 Complexity Class

The graph isomorphism problem is not known to be solvable in polynomial time nor to be NP-complete, and therefore may be in the computational complexity class NP-intermediate. It is known that the graph isomorphism problem is in the low hierarchy of class NP, which implies that it is not NP-complete unless the polynomial time hierarchy collapses to its second level. At the same time, isomorphism for many special classes of graphs can be solved in polynomial time, and in practice graph isomorphism can often be solved efficiently



Isomorphism Complete

Researchers have sought to gain insight into the problem by defining a new class GI, the set of problems with a polynomial-time Turing reduction to the graph isomorphism problem. If in fact the graph isomorphism problem is solvable in polynomial time, GI would equal P. On the other hand, if the problem is NP-complete, GI would equal NP and all problems in NP would be solvable in quasi-polynomial time.

A problem is called GI-hard if there is a polynomial-time Turing reduction from any problem in GI to that problem, i.e., a polynomial-time solution to a GI-hard problem would yield a polynomial-time solution to the graph isomorphism problem (and so all problems in GI). A problem is called complete for GI, or GI-co,if it is both GI-hard and a polynomial-time solution to the GI problem would yield a polynomial-time solution to problem.

History of Complexity

The graph isomorphism problem is one of Garey and Johnson's 12 problems unknown to be NP-complete or in P.In 1980s the a new algorithm was introduced by Babai and Luke which had a run time of $2^{O(nlogn)}$.This was the benchmark algorithm for a long time,until Babai published a milestone paper in 2015,where he gave a quasi-polynomial time algorithm.This new algorithm had a runtime of exp(log $n^{O(1)}$)

On January 4, 2017, Babai retracted the quasi-polynomial claim and stated a sub-exponential time bound instead after Harald Helfgott discovered a flaw in the proof. On January 9, 2017, Babai announced a correction (published in full on January 19) and restored the quasi-polynomial claim, with Helfgott confirming the fix. Helfgott further claims that one can take c = 3, so the running time is exp((log n)^3).The new proof has not been fully peer-reviewed yet.

2 Properties of Isomorphic Graphs

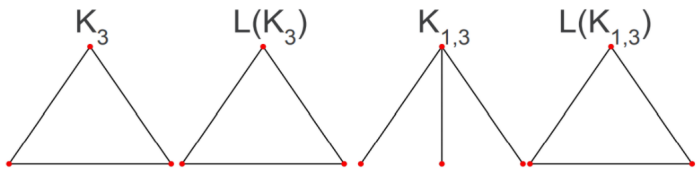
There are two types of properties for isomorphic graphs.Properties can be isomorphism invariant which means that,a function f which represents the property, such that $G \approx H \Rightarrow f(G) = f(H)$.
Properties can also be complete isomorphism invariant ,which means that $G \approx H \Leftrightarrow f(G) = f(H)$.
Isomorphic graphs have the following properties,they have:

- Same number of vertices
- Same number of edges
- Same degree sequence
- Same chromatic number, domination number etc.
- Same eigen values for their adjacency matrices

All of these properties are isomorphism invariant and not complete isomorphism invariant.

2.1 Whitney Theorem

The whitney theorem states that two graphs are isomorphic if and only if their line graphs are isomorphic, with the exceptions of K_3 and $K_{1,3}$



We can see from the figure that even though both the line graphs are same,the corresponding graphs are not isomorphic.

3 Canonization of graphs

Canonization of graphs is the problem of finding the canonical form of a given graph G(V,E). Canonization is an operation of placing the vertex labels in a way that does not depend on their previous labellings.The canonical form a graph has following properties.

- It is isomorphic to G
- It is a complete graph isomorphism invariant i.e two isomorphic graphs have same canonical forms and the graphs with same canonical forms are isomorphic

Computational Complexity

The Canonization of graph is atleast as hard as Graph isomorphism problem.Graph isomophism is even AC^0 reducible to graph canonization.However it is still an open question whether the two problems are polynomial time equivalent.

While the existence of (deterministic) polynomial algorithms for graph isomorphism is still an open problem in computational complexity theory, in 1977 Laszlo Babai reported that with probability at least $1 - e^{-O(n)}$, a simple vertex classification algorithm produces a canonical labeling of a graph chosen uniformly at random from the set of all n-vertex graphs after only two refinement steps. Small modifications and an added depth-first search step produce canonical labeling of such uniformly-chosen random graphs in linear expected time. This result sheds some light on the fact why many reported graph isomorphism algorithms behave well in practice. This was an important breakthrough in probabilistic complexity theory which became widely known in its manuscript form and which was still cited as an "unpublished manuscript" long after it was reported at a symposium.

A commonly known canonical form is the lexicographically smallest graph within the isomorphism class, which is the graph of the class with lexicographically smallest adjacency matrix considered as a linear

string. However, the computation of the lexicographically smallest graph is NP-hard.

For trees, a concise polynomial canonization algorithm requiring O(n) space is presented by Read (1972). Begin by labeling each vertex with the string 01. Iteratively for each non-leaf x remove the leading 0 and trailing 1 from x's label; then sort x's label along with the labels of all adjacent leaves in lexicographic order. Concatenate these sorted labels, add back a leading 0 and trailing 1, make this the new label of x, and delete the adjacent leaves. If there are two vertices remaining, concatenate their labels in lexicographic order.

4 Algorithms

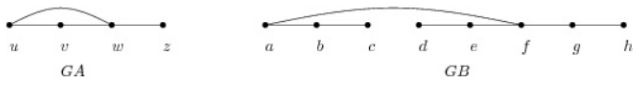
Now we will be discussing some of the famous algorithms related to graph isomorphism.These include algorithms for sub graph isomorphism,and also for some special classes of functions.

4.1 Ullman's Algorithms

This is an algorithm for subgraph isomorphism which has a run time of $O(n!n^3)$.The algorithms is as follows:

- Find Candidate vertices for each vertex
- Refine the candidate list
- Determine a candidate vertex for mapping
- Recursively try to map its neighbors using DFS
- If a neighbor cannot be mapped we backtrack

So let's discuss this with an example below. We want to know whether GB has a subgraph isomorphic to GA. That is, we will be trying to map vertices of GA to vertices of GB so that edges of GA are mapped on edges of GB.



First observation is that a vertex of GA can only be mapped to a vertex of GB of at least as large degree. So initially, candidates(u) = candidates(v) = a,b,e,f,g, candidates(w) = f and candidates of z are all the vertices of GB.

Now is the first time we do the "refine M" procedure, which is the main ingredient of the Ullmann's algorithm. That is, we check that whenever vertex x of GB is among candidates for a vertex y of GA, then every neighbor of y has at least one candidate among neighbors of x. If this check fails, then we remove x from candidates of y. We check for these removals until no further removals are possible.

For example, h is among candidates for z. However, w is a neighbor of z, but none of the neighbors of h (that is, g) is among cadidates(w)=f. Therefore we will never be able to map z to h, because the edge w,z would be mapped to a non-edge. So we can safely remove h from candidates(z). The result of the refinement is: candidates(u) = candidates(v) = candidates(z) = a,e,g and candidates(w) = f.

Now we start backtracking.We first try mapping u to a. That is, we set candidates(u) = a and remove a from the other candidate sets. Refine finds out that neither e nor g is a neighbor of a and so we remove e and g from candidates(v). This leaves candidates(v) empty and so we return from this branch; undoing the changes made to candidates.

Now, we try mapping u to e. Again, candidates(v) will end up empty.Finally, we try mapping u to g with the same result.We conclude that GA is not a subgraph of GB. Without having to try all the 8*7*6*5 assignments.

4.2 Babai's Algorithm

Babai's proposed algorithm doesn't bring graph isomorphism all the way into P, but it comes close. It is quasi-polynomial, he asserts, which means that for a graph with n nodes, the algorithm's running time is comparable to n raised not to a constant power (as in a polynomial) but to a power that grows very slowly.

The previous best algorithm — which Babai was also involved in creating in 1983 with Eugene Luks, now a professor emeritus at the University of Oregon — ran in "subexponential" time, a running time whose distance from quasi-polynomial time is nearly as big as the gulf between exponential time and polynomial time.

Babai's new algorithm starts by taking a small set of nodes in the first graph and virtually "painting" each one a different color. Then it begins to look for an isomorphism by making an initial guess about which nodes in the second graph might correspond to these nodes, and it paints those nodes the same colors as their corresponding nodes in the first graph. The algorithm eventually cycles through all possible guesses.

Once the initial guess has been made, it constrains what other nodes may do: For example, a node that is connected to the blue node in the first graph must correspond to a node that is connected to the blue node in the second graph. To keep track of these constraints, the algorithm introduces new colors: It might paint nodes yellow if they are linked to a blue node and a red node, or green if they are connected to a red node and two yellow nodes, and so on. The algorithm keeps introducing more colors until there are no connectivity features left to capture.

Babai showed that highly symmetrical "Johnson graphs" were the only case his algorithm's painting scheme didn't cover.Once the graphs are colored, the algorithm can rule out all matchings that pair nodes of different colors. If the algorithm is lucky, the painting process will divide the graphs into many chunks of different colors, greatly reducing the number of possible isomorphisms the algorithm has to consider. If, instead, most of the nodes end up the same color, Babai has developed a different way to reduce the number of possible isomorphisms, which works except in one case: when the two graphs contain a structure related to a "Johnson graph." These are graphs that have so much symmetry that the painting process and Babai's further refinements just don't give enough information to guide the algorithm.
But Johnson graphs can be handled fairly easily by other methods, so by showing that these graphs are the only obstacle to his painting scheme,Babi was able to solve this issue too

5 Special Cases

Let's discuss graph isomorphism algorithm for some of the special classes of Graphs.These algorithms have a polynomial time run time.

5.1 Caterpillars

Caterpillars are trees where all non-leaf nodes lie on a single path. It is easy to verify if two caterpillars are isomorphic. There are linear time algorithms to verify if a graph is caterpillar or not. After establishing that both the given graphs are caterpillars, the following algorithm is proposed to verify if they are isomorphic.

- Identify the spine of the caterpillars.The spine essentially contains all the non leaf nodes and no leaf nodes. The spine is a path.
- If the spine has different number of nodes, they are not isomorphic. End the algorithm.If they have equal number, let that number be n.
- Mark the ends of the spine. Let one end be called 'start' and the other be called 'end'.
- We can label the nodes of the caterpillars beginning from the start and traversing through the spinal path from 1 to n consecutively, for both the caterpillars.
- If all nodes with same labels have equal number of leaves,they are isomorphic. End the algorithm.
- Now relabel the nodes of the spine of the second caterpillar in decreasing order from n to 1 beginning from the start node.
- If all nodes with same labels have equal number of leaves, they are isomorphic. Otherwise, they are not isomorphic. End the algorithm

The algorithm is of linear time complexity once the graphs are established to be caterpillars. Hence the overall worst case time complexity of isomorphism of caterpillars is $O(n)$

5.2 Rooted Trees

The rooted trees are the trees where a node is fixed as a special node called 'root'. Even if the trees are isomorphic, the rooted trees need not be isomorphic as the root needs to be preserved for isomorphism of rooted trees.
The algorithm for isomorphism problem for rooted trees was proposed by Aho,Hopcroft and Ullman. The algorithm, which solves the rooted tree isomorphism problem in polynomial time complexity, is called "AHU" algorithm in literature.

The algorithm assigns a string to each of the node. Each string has the following properties:

- The characters in the string are either '(' or ')'
- Each leaf is assigned the following string "( )", a simple valid bracket pair
- The string of a parent node is formed by concatenating the strings of its children in their lexicographic order and enclosing the whole string in a pair of brackets.
  ($s_1 + s_2 + s_3 + ... + s_n$)
  where $s_1, s_2, s_3, .., s_n$ are the strings assigned to its children and these are in lexicographic order(It is assumed that '('comes before ')' lexicographically)

It can be seen that one can construct strings from leaf onwards, propagating up to the root uniquely. Thus, there is a unique way of assigning these strings for a rooted tree.Hence, the root will always get a unique string. Hence two isomorphic rooted trees will always assign same string at the root by assigning these tuples called as 'Knuth tuples'.It can also be observed that a string at a particular node contains information about the strings at all its children and the strings can be uniquely assigned.When the starting bracket of a string is matched, then it marks the end of that string, and this part of the string is assigned to one of its children. Thus given a knuth tuple at root, a rooted tree can be uniquely constructed. Thus this is a complete isomorphism invariant

The time complexity of AHU algorithm is $O(|V|)$,where $|V|$ is the number of vertices of the rooted tree.Although it appears that the time complexity should be $O(|V|^2)$, this can be optimised to decrease it to $O(|V|)$.

5.3 Trees

It is possible to solve the isomorphism of trees in $O(|V|_1 + |V|_2)$ tie complexity, where $|V|_1$ and $|V|_2$ are the number of vertices of given trees. If the root a tree can be fixed uniquely, then AHU can be used to solve the tree isomorphism problem.

The centre of a tree can be a single vertex or an edge. There are efficient algorithms that can identify the centre of a tree in linear time complexity.There are three possible cases:

- **Case 1: Both trees have single nodes as their centre** . In that case, one can assign these centres as roots and make the trees rooted. Once the rooted trees are formed, AHU algorithm can be used to solve the isomorphism problem
- **Case2 : One tree has a node as centre and the other has edge as centre** . In this case, the trees are clearly not isomorphic as the centre is an isomorphism invariant.
- **Case3 : Both trees have edges as centres**. In that case, we can assign one vertex at the end of the centre as root for one tree, forming a rooted tree T_0. Two rooted trees can be formed from the second tree by choosing two ends of its centre as roots. Let them be denoted by T_1, T_2. If either T_0 is isomorphic to T_1 or T_0 is isomorphic to T_2, then the given trees are isomorphic. Else, they are not isomorphic.

The Centre of tree can be found in $O(|V|)$ for each tree and AHU algorithm is of order $O(|V|_1+|V|_2)$. Hence the overall time complexity for tree isomorphism problem is $O(|V|_1 + |V|_2)$

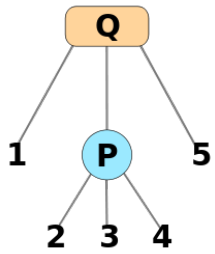
5.4 Interval Graphs

PQ Trees

PQ Trees are used to represent a subset of permuatations or orderings of a the elements of a given set S. PQ trees are characterised by the following:

- It is a rooted tree
- Each leaf of the tree represent a unique element of the set S. The number of leaves is same as the cardinality of this set
- The non leaf nodes are partitioned into two sets called a P-set and a Q-set. The nodes of the P-set are called P-nodes and the nodes of the Q-set are called Q-nodes.
- A P-node has at least 2 children and a Q-node has at least 3 children.
- A transformation of a P node is defined as reordering its children in any order from left to right. A transformation on Q node is defined as reversing the order of its children from left to right
- By reading the leaves from left to right(or rather reading the elements denoted by the leaves from left to right), an ordering of the elements can be given
- A permutation is consistent with tree PQ if it can be read from the tree by transforming any number of P-nodes or Q-nodes in any order.

PQ trees are extremely useful data structures for solving problems involving permutations that have certain constraints. These are also used in interval graph isomorphism.



For instance, The orderings consistent with the shown PQ tree are : 12345, 12435, 13245, 13425, 14235, 14325, 52341, 52431, 53241, 53421, 54231, 54321.

Whether a graph is interval graph or not can be seen from Fulkerson and Gross and an algorithm based on it.

Fulkerson and Gross Theorem: A Graph G(V,E) is an interval graph if and only if there exists a linear ordering of its cliques such that for every vertex v,the elements of C(v) appear consecutively, where C(v) is the set of all cliques in which v is a member.

It should be noted that the interval graphs are chordal graphs. Efficient algorithms exist to identify if a graph is chordal or not, and to find the cliques of the graph. These clique finding might appear as a hard problem for general graphs, but for chordal graphs polynomial time algorithms exist. The following algorithm has been proposed to identify whether a graph G(V,E) is interval or not.

- Find if the graph is chordal. If it is not, the graph is not an interval graph. Skip the subsequent steps of the algorithm. If it is chordal follow the subsequent steps
- Let P be the set of all the permutations of the cliques of graph G.
- for each vertex v:
  - Remove the permutations form P, where the elements of C(v) don't appear consecutively

  - if P becomes empty, then return False(not interval graph) and break
- if P is non-empty, return True(interval graph),else return False

The above algorithm involves permuting all the cliques of G, which is computationally hard. This can be avoided if PQ trees are used. The problem has the constraints where certain elements must be consecutive. A PQ tree can be constructed with these constraints. A PQ tree with the constraint given by Fulkerson and Gross theorem, can be constructed if and only if the graph is an interval graph. Hence if a PQ tree can be constructed, then the graph is interval. Otherwise, not.

Graph Isomorphism for Interval graphs

Graph isomorphism for interval graphs can be efficiently solved in linear time with an algorithm which uses PQ trees for implementing it.The algorithm runs in O(n+e),where n is the size of vertex set and e is the number of edges.
Algorithm for graph G and G':

- First construct the PQ trees T(G) and T(G"),these can be done in O(n+e) time.
- There are some algorithms that are given to obtain corresponding proper canonical labeled PQ-trees T and T'
- The we try to determine whether these are L-identical.
- Two labeled PQ-trees T and T' are L-identical,if they are isomorphic as ordered trees, and corresponding nodes have equal labels
- Checking for L-identical can be done in linear time by a preorder scan of trees

5.5 Planar Graphs

Here we are discussing a practical implementation of a planar graph isomorphism algorithm of complexity O(n^2)

In order to compare two planar graphs for isomorphism, we construct a unique code for every graph. If those codes are the same, the graphs are isomorphic. Constructing the code starts from decomposition of a graph into biconnected components. This decomposition creates a tree of biconnected components.
First, the unique codes are computed for the leaves of this tree. The algorithm progresses in iterations towards the center of the biconnected tree. The code for the center vertex is the unique code for the planar graph. Computing the code for biconnected components requires further decomposition into triconnected components. These components are kept in the structure called the SPQR-trees . Code construction for the SPQR-trees starts from the center of a tree and progresses recursively towards the leaves.

If the unique codes are same for both the graphs,then they are isomorphic to each other

5.6 Isomorphism using spectral properties

Graph Isomorphism has close relation with the spectral properties of the ajacency matrices of the given graph.It is known that Isomorphic graphs have same set of eigen values, while the converse need no be true. The graphs whose adjacency matrices have same set of eigen values are called "Isospectral graphs".

Permutation matrices

A permutation matrix P is a matrix obtained by permuting the rows of an identity matrix I_n. For a given Identity matrix of order n, the number of permutation matrices possible is $n!$.It is known that the set of all Permutation matrices of order n form a group under matrix multiplication operation.

If a Permutation matrix P gets multipled by a matrix A($P.A$), the rows of A get permuted in the same order as P. If a matrix gets multiplied by the transpose of a Permutation matrix P($A.P^T$), then the columns of A get permuted in the same order as in P.

Adjacency Matrices of Isomorphic graphs

An adjacency matrix of graph G(V,E) is defined only if the graph is labelled. n^{th} row corresponds to the vertex labelled n, where the elements of that row will be 1 in j^{th} column if n is connected to the vertex with label j , otherwise 0. Hence Isomorphic graph H(V,E) of G, has its adjacency matrix B with rows and columns of A permuted(as the labels of H is a permutation of labels of G) in same order.Thus,

$B = P.A.P^T$ for some permutation matrix P

Diagonalization of adjacency matrices

Since A and B are symmetric matrices, they are diagonalizable and their eigen vectors are orthonormal. It is known that if a matrix T is diagonalizable and its eigenvalues are unique, then the normalized eigenvectors are unique within a sign difference(there are only two normalized eigenvectors +v and -v) for each eigenvalue.
Thus if matrix with unique eigenvalues is known to be diagonalized in two different ways, with the diagonal matrix sorted in increasing order and eigenvectors normalized i.e,
B = $Q_1.D_1.Q_1^T = Q_2.D_2.Q_2^T$, then $D_1 = D_2$ and $Q_1.S = Q_2$, where S is a diagonal matrix with +1 or -1 entries. It can be seen that the eigenvectors will differ within a sign.

Consider two isomorphic graphs G and H with their adjacency matrices A and B respectively, then
$B = P.A.P^T$ for some permutation matrix P.
Let's Diagonalize both of them.
Let B=$Q_2.D_2.Q_2^T$ and A=$Q_1.D_1.Q_1^T$ where Q_2 and Q_1 are normalized and D_1 and D_2 have their diagonal values(which are unique) arranged in increasing order , then we have
$Q_2.D_2.Q_2^T = (PQ_1).D_1.(P.Q_2)^T$
which means $Q_2.S = PQ_1$ and $D_1 = D_2$
where S is a Diagonal matrix with +1 and -1 entries.

This proves that the eigenvalues of isomorphic graph adjacency matrices are same.

Thus eigenvector components remain same but they change their position among themselves in the same way the labels of the original graphs are rearranged.

If we further constraint that the one norm of each eigenvector is distinct, then we can assign a cannonical labelling of the graphs for both G and H. The column with least 1-norm can be assigned the least label and so on. Thus based on this cannonical labelling,isomorphism can be verified.

Special algorithms with polynomial time complexity for diagonalizing a matrix are known. Hence under the constraints that the eigenvalues and 1-norm of normalized eigenvectors are unique, isomorphism can be verified in polynomial time complexity.

6 Practical Graph Isomorphism Algorithm

Nauty and Tracy are two of the most prominent and benchmark programs for graph isomorphism.These are written in highly portable subset of the C language.
Nauty is a set of procedures for determining the automorphism group of a vertex-coloured graph, and for testing graphs for isomorphism.Trace does the same thing but their implementation is slightly different as trace uses Search trees
Isomorphism checking is done by canonical labelling by these programs.To handle this,vertices in nauty and Traces can be coloured.nauty and Traces consider the colours to come in some order; i.e., there is a 1st colour, a 2nd colour, etc..The new vertex labels are in order of colour. The vertices of the first colour are labelled first,of the second colour next, and so on. This rule means that the canonical labelling can be used to determine if two coloured graphs are isomorphic.
Traces is currently one of the leader on the majority of difficult graph classes, while nauty is still preferred for mass testing of small graphs

References

[1] J. R. Ullmann. *An Algorithm for Subgraph Isomorphism.* Journal of the ACM, 1976.

[2] László Babai Graph Isomorphism in Quasipolynomial Time Dec 2015.

[3] Lueker, George S.Booth, Kellogg S. A linear time algorithm for deciding interval graph isomorphism 1977

[4] John Edward,J K Wong *Linear time algorithm for isomorphism of planar graphs.* STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing,April 1974

[5] Jacek P.Kukluk,Lawrence B.Holder,Diane J.Cook Algorithm and Experiments in Testing Planar Graphs for Isomorphism Journal of Graph Algorithms and Applicationss, 2004.

[6] Brendan D.McKaya,Adolfo Piperno Practical graph isomorphism, II Journal of Symbolic Computation, January 2014.

[7] A. Aho, J. Hopcroft, and J. Ullman The Design and Analysis of Computer Algorithms Addison-Wesley Publishing Co., Reading, MA, 1974, pp. 84-85.

[8] Usman.A.Zahidi Spectral Solution for Detecting Isomorphic Graphs with Nondegenerate Eigenvalues 2007 IEEE International Multitopic Conference