On Pseudo trees and Graph Isomorphism

Dhananjay P. Mehendale
Sir Parashurambhau College, Tilak Road, Pune-411030, India
Email: dhananjay.p.mehendale@gmail.com

Abstract

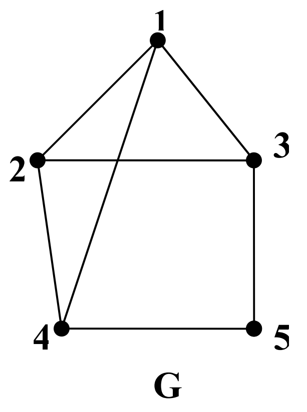
In this paper we obtain a new polynomial time algorithm for testing isomorphism of graphs. This algorithm is based on the idea of associating a rooted, unordered, pseudo tree with given graphs and thus reducing the isomorphism problem for graphs to isomorphism problems for associated rooted, unordered, pseudo trees. We show that isomorphism of the rooted, unordered, pseudo trees associated with graphs and so in effect isomorphism of given two graphs can be tested in polynomial (quadratic) time.

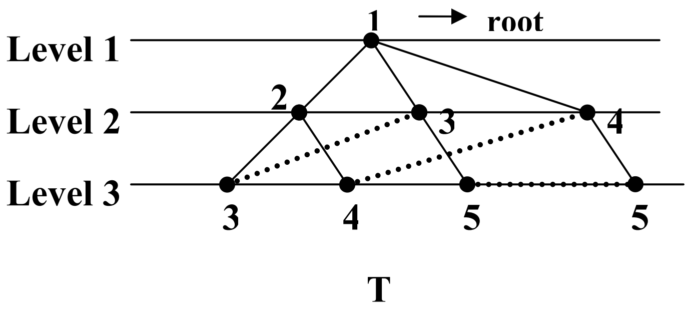
1. **Introduction:** The given graphs *G* and *H* are isomorphic when we have an adjacency preserving bijection between their vertex sets. To determine whether two given graphs are isomorphic is called the Graph Isomorphism Problem (GI). GI is of great interest to computer scientists and researchers in other fields such as chemistry, switching theory, information retrieval, and linguistics. In particular GI is of profound interest to complexity theorists because yet the graph isomorphism problem is neither proved P nor proved NP-complete. In this paper we obtain a polynomial time algorithm for testing isomorphism of two graphs in terms of isomorphism of the associated rooted, unordered, pseudo trees. It is well known that there exists a polynomial time algorithm for testing isomorphism of trees. We see that such algorithm can be extended to check isomorphism of rooted, unordered, pseudo trees maintaining its polynomial time complexity.

2. **Associating a Pseudo Tree with a Connected Graph:** In this section we introduce an idea of associating a rooted, unordered, pseudo tree with a connected graph. We show here the use of this association to obtain a new polynomial time algorithm for testing isomorphism of graphs. Thus

we show that isomorphism problem for graphs can be looked upon as isomorphism problem for its associated rooted, unordered, pseudo trees. We further show that the required algorithm for testing isomorphism of rooted, unordered, pseudo trees doesn't differ in complexity from the algorithm for testing isomorphism of trees (which can be done in quadratic time). We associate with (or, look at a) connected graph (as) a rooted, unordered, pseudo tree. To describe this idea let us begin with an

Example: Consider following connected graph, G say, and its associated rooted, unordered, pseudo tree, T say:





This tree T is called rooted because it has a root (vertex with label 1), unordered because it contains multiple vertices with same label, and pseudo because it contains pseudo edges joining vertices with same label shown by dotted line segments which are actually nonexistent.

Pseudo tree Construction Procedure: For given unlabeled (n, e) graph we carry out following steps to construct rooted, unordered, pseudo tree.

1) Choose any vertex of given unlabeled graph as root and assign label "1" to it.
2) Assign labels "2", "3", …, "n" to other vertices of graph.
3) Create 1st level of desired tree consisting of vertex with label "1".
4) Take all vertices adjacent to vertex with label "1" and paint the edges emerging from vertex with label "1" and entering in all vertices s1, s2, … etc adjacent to it with **red** color in the graph as a distinguishing mark. This marking is done to understand that these edges are now used and are not to be used again.
5) Create 2nd level by putting all vertices "s1", "s2", … etc., adjacent to the only vertex with label "1" in 1st level, in the 2nd level and draw the corresponding edges joining vertex labeled "1" in the first level to each vertex, among the vertices "s1", "s2", … etc, in the second level.
6) Take vertex "s1" in the graph. Treat all vertices connected to it by a red edge as nonadjacent (e.g. in this case the vertex with label "1"). Determine all vertices adjacent to vertex "s1", say "t11", "t12", …, etc. and paint the edges emerging from vertex with label "s1" and entering in all vertices "t11", "t12", … etc adjacent to it with red color in the graph as a distinguishing mark. . This marking is done to understand that these edges are now used and are not to be used again.
7) Create 3rd level **partially** by putting vertices "t11", "t12", … etc., adjacent to the vertex with label "s1" in 2nd level, in the 3rd level and draw the corresponding edges joining vertex labeled "s1" in the second level to each vertex, among the vertices "t11", "t12", … etc, in the third level.
8) Repeat steps 6), 7) for **all** other vertices "s2", "s3", … etc. in the 2nd level, by taking them one by one in a sequence, and thus complete the creation of 3rd level, i.e. create the third level **completely**.
9) Continue this procedure of pseudo tree construction till all edges in the graph become red and all possible further levels have been created completely.
10) In the rooted, unordered, pseudo tree since we avoid cycle formation by following special procedure for its construction described in above steps (which forcefully forbids presence of edges joining vertices in the same level), many vertices show their more than once appearance. So, finally, join all pairs of vertices with same label by a pseudo edge (dotted line segment).

Important care to be taken while we construct rooted, unordered, pseudo tree:

"***All vertices adjacent to root (1st level) must appear in the 2nd level. All vertices adjacent to vertices in the 2nd level connecting to so far unused and so unmarked edges must appear in the 3rd level and so on and every circuit formation is avoided and finally all vertices with same label are joined to each other by dotted line segments representing pseudo edges which are actually nonexistent***"

Remark 2.1: Note that in order to avoid formation of any cycle (which should not exist in a tree) we take such vertices as additional vertices in the next level. Now, when vertices with labels, say m and n, and present in the same level and if these vertices with labels m and n are adjacent we need to take either n in the next level joined by edge to vertex labeled m in the current level or we need to take m in the next level joined by edge to vertex labeled n in the current level. Thus, while proceeding with construction of pseudo tree taking into account the above mentioned two possibilities of construction at each multiple occurrences of vertices in the same level we may get more than one different (nonisomorphic) pseudo trees associated with same graph.

Remark 2.2: Note that in order to avoid formation of any cycle (which should not exist in a tree) we take such vertices as additional vertices in the next level so clearly the rooted tree contains more vertices than the one contained in the original graph.

Remark 2.3: It is straightforward to check that if the connected graph G under consideration is an (n, e) graph, i.e. it contains n vertices and e edges, then its associated rooted, unordered, pseudo tree contains (e+1) vertices.

Remark 2.4: It is easy to see further that in all there are (e+1-n) repetitions of vertices, i.e. there are in all (e+1-n) vertices which show appearance more than once in the associated rooted, unordered, pseudo tree.

Remark 2.5: There are well known linear time algorithms to test isomorphism of rooted trees [1]. For problem of isomorphism in general case, i.e. unrooted trees, where we do not have a useful start as root, the algorithm for rooted trees can be easily adopted to make it work on unrooted trees: we only have to pick a vertex in first tree and declare it as root, and try all the vertices of second tree one after the other. The two unrooted trees will be isomorphic if and only if there is at least one vertex in second tree which, when declared as root prompts the linear time algorithm to test isomorphism of rooted trees to answer positively. This makes the algorithm for testing isomorphism of unrooted trees quadratic in time.

3. **Algorithm for Graph Isomorphism:** In this section we state the result that decides the isomorphism of two given graphs in $\sim O(n^2)$.

Definition 3.1: Two rooted, unordered, pseudo trees are isomorphic if they are isomorphic as rooted trees under some adjacency preserving bijection, σ and further σ also preserves pseudo-adjacency, i.e. if w is a vertex that occurs more than one time, say k times, in first pseudo tree so that there are pseudo edges joining each pair formed from these multiple copies of w, then the image of w, σ(w), also will appear k times in second rooted, unordered, pseudo (image) tree and there are also pseudo edges present, joining each pair of these corresponding multiple copies of σ(w), under the same bijection σ.

Theorem 3.1: Let G and H be the two given (n, e) graphs. Let V(G) and V(H) be the vertex sets for G and H respectively. Let vertex u belongs to V(G) and vertex v belongs to V(H) and let Tu(G) and Tv(H) be the associated rooted, unordered, pseudo trees rooted at vertex u and v for graphs G and H respectively. If Tu(G) is isomorphic to Tv(H) then the graph G is isomorphic to graph H under the same mapping depicting isomorphism of associated pseudo trees.

Proof: As we have Tu(G) is isomorphic to Tv(H) we have got adjacency and pseudo-adjacency preserving bijection σ. Now, if all the pseudo edges are contracted then we see that the number of vertices change from (e+1) to n and the same bijection, σ, will now act as isomorphism for graphs G and H from which these (isomorphic) pseudo trees were constructed.

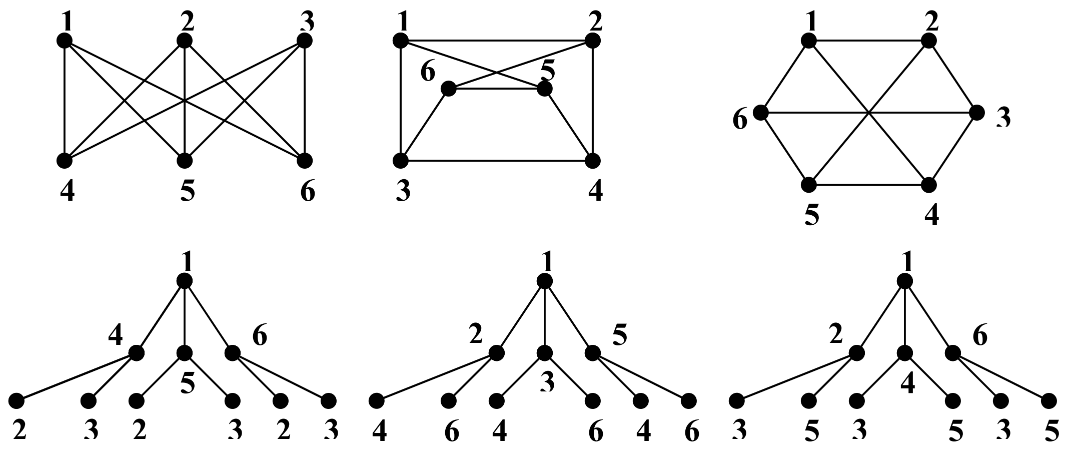
□

We now proceed to discuss the algorithm for checking isomorphism of given two graphs which has polynomial order because the polynomial order is enough for checking isomorphism of rooted, unordered, pseudo trees associated with these graphs.

Algorithm for checking Graph Isomorphism:

1) Let G and H be two (n, e) graphs given for isomorphism check. Let V(G) and V(H) be their vertex sets respectively. Let these sets are V(G) = {u1, u2, u3, ….} and V(H) = {v1, v2, v3, ….}.
2) Construct rooted, unordered, pseudo tree, say Tu1(G), rooted at vertex u1.
3) Construct rooted, unordered, pseudo tree, say Tv1(H), rooted at vertex v1. At this time construct all possible nonisomorphic pseudo trees (as per the remark 2.1) with vertex v1 as root. And thus construct set of nonisomorphic trees {Tv1(H)}.
4) Check whether Tu1(G) is isomorphic to some pseudo tree in set {Tv1(H)}.
5) If yes, declare that G is isomorphic to H and stop.
6) Else, take next vertex v2 in V(H) as root and construct set of nonisomorphic rooted, unordered, pseudo tree set, {Tv2(H)}, rooted at vertex v2.
7) Check whether Tu1(G) is isomorphic to some tree in {Tv2(H)}.
8) If yes, declare that G is isomorphic to H and stop.
9) Else, continue the above steps for next vertex v3, v4, …, till we reach either at isomorphism decision for G and H or at the last vertex of V(H).
10) If we don't find isomorphism of Tu1(G) with any of the rooted, unordered, pseudo trees, in the sets {Tvj(H), j = 1, 2, ….} that we construct using vertices of V(H) by taking them one by one in succession as roots, then declare that G and H are not isomorphic.

Example: We consider below three graphs and find their associated pseudo trees. The isomorphism of these three graphs becomes evident from the easy isomorphism of their associated pseudo trees (pseudo edges and levels are not shown to avoid clumsiness).



Remark 3.1: Since isomorphism of rooted, unordered, pseudo trees can be checked in linear time using any known algorithm and we may require to repeat this checking n number of times (n = |V(H)|, cardinality of vertex set) for desired decision, we get the overall complexity for algorithm for isomorphism of graphs as $\sim O(n^2)$.

Acknowledgements

I am thankful to Prof. M. R. Modak for useful discussions.

References

1. Marthe Bonamy, A Small Report on Graph and Tree Isomorphism, 2010. http://perso.ens-lyon.fr/eric.thierry/Graphes2010/marthe-bonamy.pdf