Automorphism Groups of Graphs

Graph Theory - Term Paper

14th January 2020

Vivek Muskan	Harsh Kumar	Aayush Somani
2017MT10755	2017MT10729	2017MT10722

Course Coordinator: Prof. B.S. Panda

Department of Mathematics, IIT Delhi

Index

1. Introduction and Problem Statement

1.1 Abstract

An automorphism of a graph $G = (V, E)$ is a permutation σ of the set of vertices V, such that there is an edge between the pair of vertices (u,v) if and only if the pair $(\sigma(u),\sigma(v))$ also form an edge in the graph. It is a graph isomorphism from G to G (itself). Automorphisms is defined in this way both for directed graphs and undirected graphs. The composition of any two automorphisms is also an automorphism, and the set of automorphisms of a given graph, under the composition operation, forms a group which is known as the automorphism group of the graph. Conversly, by using Frucht's theorem, all groups can be represented as the automorphism group of a connected graph, more specifically, of a cubic graph.

1.2 Automorphism Groups of Graphs

The symmetries of a graph G are described by its automorphism group Aut(G). Every automorphism is a permutation of vertices which preserves adjacencies and non-adjacencies of graphs. It was proved by Frucht that every finite group is isomorphic to the automorphism group of some graph. General mathematical structures may be encoded by graphs while preserving automorphism groups.

1.3 Graph Automorphism Problem

It is the problem of testing whether a graph has a nontrivial automorphism. It belongs to the NP class of computational complexity. Just like the graph isomorphism problem, it is unknown whether it has a polynomial-time algorithm or NP-complete. There is a polynomial-time algorithm for solving the graph automorphism problem but for the graphs where vertex degrees are bounded by a constant. The graph automorphism problem can be polynomially timed, in many-one ways be reduced to the graph isomorphism problem, but the converse reduction is still unknown. The computational hardness of this problem is known when the automorphisms are constrained in a certain way; for example, determining the existence of a fixed-point-free automorphism (an automorphism that fixes none of the vertices) is NP-complete, and the problem of counting no. of such automorphisms is #P-Complete.

2. Application of Graph Automorphism

Practical applications of Graph Automorphism include

a. *Graph drawing*

A drawing of a graph is a pictorial representation of the vertices and edges of a graph. A graph drawing should not be confused with the graph itself because there can be different layouts that correspond to the same graph.

b. *Boolean Satisfiability*

Graph automorphism can help in solving structured instances of Boolean Satisfiability arising in the context of Formal verification and Logistic.

c. *Molecular symmetry*

In chemistry, Molecular symmetry describes the symmetry present in different molecules and the classification of these molecules according to their symmetry. Automorphic groups help in identifying molecules with similar structure and property.

d. *Graph Coloring*
e. *Constraint Programming*
f. *Model Checking*
g. *Markov Models*
h. *Integer Programming*

3. Literature Review

3.1 <u>Related Graph classes</u>

- **Caterpillar graphs** (CATERPILLAR) are trees with every leaf attached to a central path. They form the intersection of tree graphs and interval graphs. Chordal graphs are intersection graphs of subtrees of simple tree graphs. They don't contain induced cycles of length four or more and naturally generalise interval graphs. Chordal graphs have universal automorphism groups.
- **Pseudoforests** (PSEUDOFOREST) are graphs for which every connected component is a pseudo tree, where the pseudo tree is a connected graph with at most one cycle. Each pseudoforest is a circle graph. The automorphism groups of pseudoforests can be constructed from the automorphism groups of tree graphs by semidirect products with

cyclic and dihedral groups, which makes the automorphisms rotating/reflecting unique cycles.

- **Function graphs** are intersection graphs of continuous functions f : [0, 1] → R. Equivalently, function graphs are co-comparability graphs, meaning their complements can be transitively oriented. Every interval graph is a co-comparability graph since disjoint pairs of intervals can be oriented from left to right. Permutation graphs are function graphs that can be represented by linear functions.
- **Claw-free graphs** are graphs with no induced K1,3 (bipartite graphs). Roberts proved that CLAW-FREE ∩ INT is equal to the class of proper interval graphs which are interval graphs with representations in which no interval properly contains another. The complements of bipartite graphs (co-BIP) are universal. They are claw-free and contained in function graphs since each of the bipartite graphs is transitively orientable.
- **Interval filament graphs** are intersection graphs of the following mentioned sets: For every Ru, we choose a closed interval [a, b] where Ru is a continuous function defined from [a, b] → R such that it is zero on the boundaries, i.e. Ru(a) = Ru(b) = 0 and Ru(x) > 0 for x ∈ (a, b). They generalize circle, chordal, and function graphs.

3.2 Important Theorems and Lemma

Theorem 1.
*(i) Aut(**INT**) = Aut(**TREE**),*
*(ii) Aut(connected **PROPER INT**) = Aut(**CATERPILLAR**),*
*(iii) Aut(**CIRCLE**) = Aut(**PSEUDOFOREST**).*

The first equality is not well known. The structural analysis is based on PQ-trees, which combinatorially describes all different interval representations of an interval graph. It explains the first equality. Without PQ-trees, this equality is surprising since these classes are very different. Caterpillar graphs that form their intersection have minimal groups.
The second result follows directly from the known properties of proper interval graphs and our structural understanding of *Aut(INT)*.
Using PQ-trees, Colbourn and Booth give a linear-time algorithm to compute permutation generators of an interval graph's automorphism group. In comparison, our description allows us

to construct an algorithm that outputs the automorphism group in group products that reveal its structure.

Concerning (iii), we aren't aware of any results related to automorphism groups of circle graphs. One inclusion is trivial since PSEUDOFOREST. The other result is based on split trees which explain all representations of circle graphs. The semidirect product with a given cyclic group or a dihedral group corresponds to the rotations or reflections of a split-tree's central vertex. Geometrically, it corresponds to the rotations/reflections of the entire symmetric representation. This approach is very similar to the circle graph isomorphism algorithm.

Lemma 1 *(Fulkerson and Gross): A graph X is an interval graph if and only if there exists an ordering of the maximal cliques such that for every x* $\in$ *V (X) the maximal cliques containing x appear consecutively.*

Theorem 2 *(Jordan). If* $X_1, \ldots, X_n$ *are pairwise non-isomorphic connected graphs and X is the disjoint union of* k_i *copies of* X_i*, then* $Aut(X)$ ~= $Aut(X_1)(S_{k1} \times \cdots \times Aut(X_n))S_{kn}$

Lemma 2 *A group G* $\in$ *Aut(**INT**) if and only if G* $\in$ *I, where the class I is defined inductively as follows:*
(a) {1} $\in$ *I.*
(b) If G1, G2 $\in$ *I, then G1 × G2* $\in$ *I.*
(c) If G $\in$ *I and n* ≥ 2*, then G o Sn* $\in$ *I.*
(d) If G1, G2, G3 $\in$ *I and G1 ~= G3, then (G1 × G2 × G3) o*ϕ *Z2* $\in$ *I, where* ϕ*: Z2 → Aut(G1 ×G2 ×G3) is the homomorphism defined as* $\phi(0) = id$ *and* $\phi(1) = (g_1, g_2, g_3) \rightarrow (g_3, g_2, g_1)$

Proof: We first prove that I ⊆ Aut(**INT**). Clearly {1} $\in$ Aut(**INT**). It remains to show that the class Aut(**INT**) is closed under (b), (c), and (d). For (b), we can show this by attaching two-interval graphs X1 and X2 on an asymmetric interval graph. The resulting graph represents the direct product of Aut(X1) and Aut(X2). For (c), let G $\in$ *Aut(INT)* & n $\geq$ 2. There exists an interval graph Y such that Aut(Y) ~= G. We now construct X as the disjoint union of n copies of Y. By Theorem 3, it follows that Aut(X) ~= G*Sn. For (d), we construct an interval graph X by attaching X1, X2, and X3 to a path as in Fig. 4a, where Aut(Xi) = Gi and X1 ~= X3. Then Aut(X) ~= (G1×G2×G3)oϕZ2.

For the converse, we show that Aut(M) ∈ I. We have three cases for the root of M. For a P-node, Aut(M) is determined by the automorphism groups of its subtrees using Theorem 3, so the operations (b) and (c) are sufficient. For an asymmetric Q-node, Aut(M) is the direct product of its subtrees' automorphism groups. For asymmetric Q-nodes, we apply the operation (d) where G1 corresponds to the automorphisms of the left part of the Q-node, G2 to the middle part, and G3 to the right part. The semidirect product with Z2 corresponds to reversing Q-node.

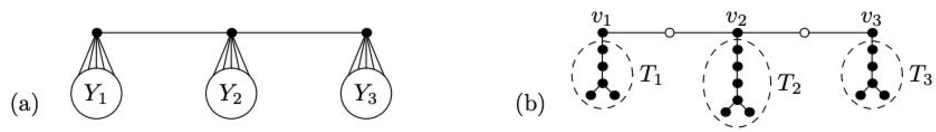

Figure 4 (a) Construction of the operation (d) from Lemma 4. (b) Trees attached to a path by their roots. Since the automorphism group is not isomorphic to $\big(\mathrm{Aut}(T_1)\times\mathrm{Aut}(T_2)\times\mathrm{Aut}(T_3)\big)\rtimes_\varphi\mathbb{Z}_2$, we fix it by subdividing v_1v_2 and v_2v_3.

Lemma 3. *Let X is the caterpillar graph and let P is the central path.*
*(i) If no automorphism swaps the path P, then the group Aut(**X**) is isomorphic to a direct product of symmetric groups.*
(ii) If there exists an automorphism of X that swaps the path P, then Aut(X) ~= (G1 × G2 × G3) oφ Z2, where G2 is isomorphic to Sk, G1 ~= G3 are isomorphic to a direct product of symmetric groups, and φ is the homomorphism defined as φ(0) = id and φ(1) = (g1, g2, g3) → (g3, g2, g1).

Proof:
PQ-trees. Booth and Lueker invented a data structure called PQ-tree to solve the long-standing open problem of recognising interval graphs in linear time.
MPQ-trees. A modified PQ-tree is made from a PQ-tree by including information about the vertices. They were explained by Korte & Möhring to simplify linear-time recognition of the interval graphs.
The root of an MPQ-tree graph M representing a caterpillar graph X is a Q node. All twin classes are trivial cases since X is a tree graph. Every child of the root is either a P-node or a leaf node. All children of every P-node are leaves. If there exists an automorphism that swaps the central path P, then the root is symmetric. Otherwise, the root is asymmetric.

Lemma 4. *The vertices $x, y \in V(X)$ are adjacent if and only if there exists an alternating path $x, m_1, m_2, \ldots, m_k, y$ in the split-tree S such that each m_i is a marker vertex, each $m_{2i-1}m_{2i}$ is a tree-edge, and the remaining edges belong to $E(X)$.*

Lemma 5. *Let S be a split-tree representing X. Then $Aut(S) \cong Aut(X)$.*

Proof. First, we prove that each $\sigma \in Aut(S)$ induces a unique automorphism α of X. We define $\alpha = \sigma V(X)$. By Lemma 4, two vertices $x, y \in V(X)$ are adjacent if and only if there exists an alternating path in S connecting them. Since σ is an automorphism on X, the existence of this alternating path is preserved b/w both x and y & b/w σ(x) and σ(y). Therefore $xy \in E(X) \Longleftrightarrow \alpha(x)\alpha(y) \in E(X)$.
For the converse part, we prove that $\alpha \in Aut(X)$ induces a unique automorphism $\sigma \in Aut(S)$. On the non-marker vertex, σ is determined. On the marker vertices, we define σ recursively. Now, let *(A, B, A', B')* be the split in *X*. This split is mapped by α to another split *(C, D, C', D'), i.e.,* $\alpha(A) = C$, $\alpha(A') = C'$, $\alpha(B) = D$, and $\alpha(B') = D'$. By applying the split decomposition to the first split, we get the graphs X_A and X_B with the marker vertices $m_A \in V(X_A)$ and $m_B \in V(X_B)$. Similarly, for the second split, we get X_C, X_D with $m_C \in V(X_C)$ and $m_D \in V(X_D)$. Since α is an automorphism, we have $X_A \cong X_C$ & $X_B \cong X_D$. Clearly, it follows that the unique split-trees of X_A and X_C are isomorphic, and similarly for X_B and X_D. Therefore, we define $\sigma(m_A) = m_C$ & $\sigma(m_B) = m_D$, and we finish the rest proof recursively.

4. Open Problems

*Problem 1: What is Aut(**PERM**)?*

Circular-arc graphs (**CIRCULAR-ARC**) are intersection graphs of circular arcs and they naturally generalise interval graphs. Surprisingly, this class is very complex and more different from interval graphs than it seems. The paper of Hsu relates circular-arc graphs to circle graphs. It easily follows that *Aut(**CIRCULAR-ARC**) ⊇ Aut(**PSEUDOTREE**)*.

*Problem 2: What is Aut(**CIRCULAR-ARC**)? Is it equal to Aut(**PSEUDOTREE**)?*

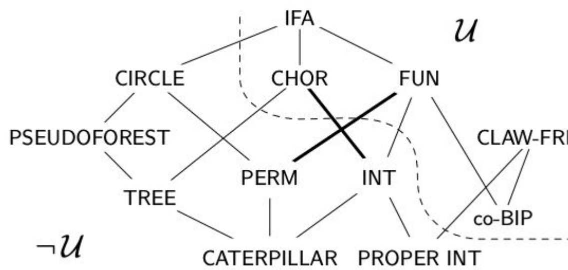

Figure 2 The inclusions between considered graph classes. We denote universal classes by $\mathcal{U}$, and non-universal by $\neg\mathcal{U}$. The bold edges are two infinite hierarchies, discussed in Section 6.

Figure 2 depicts two infinite hierarchies of graph classes, one between **INT** and **CHOR**, and the other one between **PERM** and **FUN**. In both cases, the bottom graph class has non-universal automorphism groups, and the top one has universal automorphism groups.

5. References

1. Pavel Klavík and Peter Zeman, "Automorphism Groups of Geometrically Represented Graphs"
2. Johannes Siemon, Arch. Math., Vol. 41,379--384 (1983), "Automorphism groups of graphs"
3. Luke Rodriguez (2014), "Automorphism Groups of Simple Graphs"
4. Weisstein, Eric W. "Graph Automorphism." From MathWorld
5. Web source :
   https://en.wikipedia.org/wiki/Graph_automorphism
   http://web.eecs.utk.edu/~cphill25/cs594_spring2014/Graph%20Automorphisms.pdf