Graph Isomorphism

Vriddhi Pai

University of Florida
UFID: 16641323

Abstract

The graph isomorphism problem por GI problem aims to determine whether 2 apparently dissimilar input graphs structurally same [2]. The problem is widely applicable in context of analogizing combinatorial structures such as hypergraphs, multigraphs and finite automata. Despite many approaches attempting to do so, the Graph Isomorphism Problem remains algorithmically unsolvable in polynomial time. It is famously considered an open problem since it has still not proven to fall under P and NP-Complete classes of Problems [19]. It has only been proven to belong to class NP which is why it is rather considered under the coined title of "class NP-intermediate problems" [1]. To this day, there exist known solutions that solve the graph isomorphism only for a limited to few instances of graphs using polynomial-time algorithms. Current state-of-the-art is utilizing group theory and topology theory to generalize its solvability to general graphs [2]. This survey defines and summarizes the graph isomorphism problem by reviewing relevant literature to underscore existing approaches, algorithms, its utility and pertaining challenges in existing methods.

Keywords: *Graph isomorphism, Combinatorial, Hypergraphs, Multigraphs, Automata, NP, NP-Complete*

1. Introduction

1.1 Problem Definition

Let G= (V, E) be some input graph where V is a set of nodes/vertices and E is the set of edges. V and E are bounded by an $E \subseteq (V \times V)$ relationship. Any two input graphs possessing disjoint structural appearances but retaining the equal number of components such as edges, vertices and connectivity between the edges are considered isomorphic. The property of isomorphism makes one graph appear as a rearranged or "tweaked" version of the second graph.
Formally,

Definition: Given 2 instances of the form $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ and a bijective function, f, G_1 and G_2 are isomorphic iff V_1 and V_2 have a relation of one-to one correspondence via f such that:
for any two nodes/vertices x, $y \in V_1$ and $(x,y) \in E_1$, $(f(x),f(y)) \in E_2$ [4].
Hence, 2 vertices in G_1 are adjacent in G_1 iff f(x) and f(y) are adjacent in the other graph. This is also called the edge-preserving bijection property of isomorphism.

1.2 Background

Meticulous research in GI problem led to Reid and Corneil in 1977 coin it as ""The Graph Isomorphism Disease" [5]. The famously accepted version of the algorithm was produced by Babai and Luks in 1983; essentially an extension of existing work by Luks alone in 1982. Later in 1985, V.N. Zemlyachenko came up with a 'subfactorial' algorithm as an addendum to what Babai and Luks had done. This "final" algorithm had runtime $2^{O(\sqrt{n\log n})}$ at first, but was later improved to $2^{O(\sqrt{n}\ \log_2 n)}$. However, the root n factor in the exponent remains a persisting problem in general case of graphs. In more recent context, Babai in his 2015 work, produced a "quasi-polynomial time" algorithm extendible to graph theory with an even better runtime of $2^{O(\log^c)}$ with some positive constant c [6][7][8]. He further improved it in 2017 after incorporating Helfgott's proposal to fix c to 3 making the final complexity as $2^{O(\log^3)}$ [9][10].

1.3 Applications

The GI problem is proved useful in mathematics, computing and chemistry. It is also commonly used as a subprocess in the fields of CV and Pattern Recognition to find structural similarity between given graphs. In fields like Chemistry, graph isomorphism is utilized to recognize certain chemicals in chemical records through structural comparisons. In electronics, graph isomorphism proves to be a useful subroutine within the circuit design setup of Layout Versus Schematic (LVS) to find similarities in circuit representation [3].

2. State-of-the-Art

2.1 Existing General Approaches

Scott Fortin in [5] produced a technical report on graph Isomorphism defining complexity classes of the GI problem through untraditional means. Since this problem does not qualify as P/NP-Complete class problem, it was considered under the tailor-made title "isomorphism-complete"; a new class of problems [15]. The second unconventional approach was by generalizing NP class within other classes and check for the problem's fit under those classes and the ultimate goal was to identify classes of graphs where the problem satisfies membership Polynomial class.

2.1.1 Graph Isomorphism-Complete Class

GI Complete class of problems are shown as being equivalently hard to GI i.e. a graph isomorphism problem can be reduced in polynomial time to a GI Complete problem using "Turing reduction" [23]. [24] shows that problems that include enumerating isomorphism cases for 2 graph samples as isomorphism complete problems [16]. In turn, a GI-complete problem can be so iff it is also isomorphism-hard. Such problems are polynomial-time solvable if GI problem is polynomially-solvable. The classes of graphs that belong to this class can be Line Graphs, Bipartite Regular Graphs,

Bipartite Eulerian Graphs, Split Graphs, Bipartite Graphs with non-trivial strongly regular subgraphs, Chordal Graphs, Connected Graphs, Regular Self-Complementary Graphs and those with Diameter 2 and Radius 1 [3].

***Example* - Graph Isomorphism for a chordal bipartite graph [20]**

A chordal graph is one wherein every cycle has a minimum length of 4 and has a chord and chordal bipartite if the graph is bipartite and every cycle of is at least 6 in length and has a chord [20]. In [22], the bipartite graph G= (X,Y,E) where $|X \cup Y| = n$ and $|E| = m$ has been reduced to a directed path (DP) graph $G' = (X \cup Y \cup E, E')$ to show isomorphism existing in the bipartite graphs only if it exists in the reduced directed path graphs. The DP graph obtained is a split graph further reduced to a chordal bipartite graph $\check{G}$ having its vertex set defined as $(X \cup Y \cup E \cup E' \cup B \cup W)$ making every vertex e belonging to E correspond to 3 other vertices $e' \epsilon E'$, $e_b \epsilon B$, $e_w \epsilon W$. These reductions take place in polynomial time.

Lemma: For G_1 and G_2 which are bipartite, isomorphism holds iff their chordal bipartite reductions $\check{G}_1$ and $\check{G}_2$ are isomorphic.

Proof: For every edge e = {u,v} where u, $v \epsilon (B \cup W)$ a cycle called a "handle" exitsts having length 4 containing the edge e. As per the isomorphism property, two chordal bipartite graphs are mapped such that handles in $\check{G}_1$ are mapped to those in $\check{G}_2$. Also, for i equals to 1 and 2, we can obtain an isomorphic duplicate $\check{G}_i$ out of contracting the handles of $\check{G}_1$ and $\check{G}_2$ into a single vertex in $\bar{G}_i$. This implies that if $\check{G}_1$, $\check{G}_2$ show isomorphism, $\bar{G}_1$ and $\bar{G}_2$ are isomorphic. It's trivially true other way round.

2.1.2 Generalization of NP Class

Problems that can be checked via a proof certificate in polynomial time are NP Class Problems. The prover and verifier do this verification. While the prover checks the instance to provide a certificate, sends it to the proof verifier for verification in polynomial time and based on the decision, accepts/rejects the instance. The new class 'IP' enables multiple polynomial message exchanges instead of a single one amongst the prover-verifier pair. Algorithm verifying GI problem membership:

Algorithm

1) Let G_1 and G_2 be two input instances.
2) Let verifier selects {a,b} = {1,2}
3) Let I = G_a's isomorphism and J = isomorphism of G_b.
   (I, J are sent in parallel to the verifier)
4) Prover, in turn passes two values i and j such that G_i is isomorphic to I and G_j is isomorphic to J.
5) If a = i and b= j → accepted by Verifier.

Hence, if G_1 and G_2 are not each other's isomorphs, verifier will always accept the input. If, however, they are isomorphic, the prover will have no way to know where graph I was derived from and ends up returning some i ≠ a with a 0.5 probability. Similarly, it'll return some j ≠ b with ½ probability. In this way, the verifier accepts with ≤ 0.25 probability if G_1 and G_2 are isomorphic or not.

2.1.3 Graphs with P-time GI Computability

GI can be credited to having a polynomial-time solution for a subset of graphs including Trees, eigenvalue multiplicity, Circular Graphs, Planar Graphs, Circulant Graphs Interval Graphs, Bounded-parameter Graphs such as treewidth, genus, multiplicity, Permutation Graphs, color multiplicity in Colored Graphs.

***Example*- Ullman's Algorithm for Subgraph Isomorphism Detection [21]**

Definition 1: For 2 graphs G_1 and G_2 having respective adjacency matrices M_1 and M_2, an isomorphism exists if a permutation matrix P exists such that:

$$\mathbf{M' = PM_1P^T}$$

Where P^T is also an adjacency matrix of G and the transpose of P.

Definition 2: For 2 graphs G_1 and G_2, a subgraph isomorphism exists if there exists a subgraph $S \subset G_2$ such that there's an isomorphism between G_1 and S.

Input: Model (a priori known) graph G = (V, E, υ, ν, L_v, L_e) and Input (Unknown) graph G_1 = (V_1, E_1, υ1, ν1, L_v, L_e); M= adjacency matrix of G and M_1= adjacency matrix of G_1.

Problem: To find all subgraph isomorphisms by gradually setting the permutation matrix P row by row.

Approach: Finding all permutation matrices P such that matrices $M_1 = S_{m,m}(PM_1P^T)$. Also, let $S_{i,n}(P)$ be an i x n permutation matrix representing a partial match from the first I matrices of G onto some vertices of G_1. Here graph isomorphism condition corresponds to:

$$\mathbf{S_{i,i}(M_1) = S_{i,n}(P)M(S_{i,n}(P))^T}$$

Hence, here $S_{i,n}(P)$ symbolizes the graph isomorphism from a subgraph of G_1 to a subgraph of G where G_1containing first i vertices of G's subgraph.

Algorithm

1) Let P = (p_{ij}) = (n x m) permutation matrix; n= |V|, m= $|V_1|$; M, M_1 = respective adjacency matrices
2) Procedure Backtrack(M, M_1,P, 1) called.
3) Backtrack(M, M_1, P, counter k):
    a. if k is more than m
        i. then P = subgraph isomorphism from G_1 to G. P is output and algorithm exits.
    b. for i from 1 to n
        i. set p_{ki}=1 and for all j ≠ i, set p_{kj}= 0
        ii. if $S_{k,k}(M_1) = S_{k,n}(P)M(S_{k,n}(P))^T$
            1. then call Backtrack(M, M_1, P, k +1)

This process continues until m rows of the P matrix are set and isomorphism is finally found.

2.1.4 Practical Isomorphism Testing Methodolgies

2.1.4.1 Non-traditional Approaches

For instance, the GI Problem can be thought of as a problem applying constraint algorithms for verifying isomorphism. This approach is not very ideal because constraint satisfaction problem is NP Hard while the GI problem tends to exist under the P complexity of problems which effectively makes the performance of this new approach as good as existing backtracking approaches [13][18]. Algorithms verifying isomorphism in planar graphs using image processing constraints cannot be generalized to general graphs hence have limited applicability as a common solution.

2.1.4.2 Vertex Invariants

This is an algorithm-independent approach wherein if a value on a graph's vertex maps to a corresponding value of another graph, both graphs being isomorphic, then the two graphs are confirmed to have the same degree.

2.1.4.3 Direct Solution

This approach works by looking for a direct mapping of vertices of the two graphs using a subgraph created out of selected vertices of one of the graphs and building isomorphism to a subgraph of the other graph [17]. The underlying algorithm then extends this to unincluded vertices by consistently adding a new vertex step by step until the graphs are fully explored and proved isomorphic. This method allows isomorphism to be detected earlier compared to a usually more exhaustive process. However, this approach may not work for some cases including eliminating isomorphs from a collection of graphs.

2.1.4.4 Canonical Labeling

Canonical Labeling assumes a vertex set of from 1 to n for both graphs to be checked for isomorphism. It further assumes the two graphs to be "automorphs" i.e. isomorphisms of themselves [9]. The program utilized in the procedure, called "nauty" uses a "canonical isomorph function" to identify the smallest automorphism of the two input instances and checks isomorphism between the automorphs. Every time the second graph is found to be isomorphic to the first one, a "canonical label" or "canonical isomorph" is delegated to the first instance by the canonical isomorph function. This method can further be utilized in conjunction with processes like hashing and sorting to solve specific problems like eliminating specific graphs from a collection of graphs.

***Example* [21]**

Aim: To distinguish all graph vertices using degrees of their neighbors.

Approach: Define a list of Degree neighborhood' for a vertex with degrees of its neighbor vertices in a sorted order. The degrees and thus the degree vertices are invariants under isomorphism [21]. This list created for every individual vertex will be its canonical label.

Input: Two graphs G and H

Algorithm

For both graphs individually-

1) Determine the vertex degrees.
2) For each vertex in the graph, determine degree-neighborhoods.
3) According to the degree neighborhoods in lexicographic order, sort the vertices.
4) Exit with a 'FAIL' message when degree neighborhoods are non-distinct.
5) According to the sorted order, number the vertices.

Theorem: The algorithm outputs a canonical labeling of G if it does not fail.

Proof:

In the algorithm, Steps 1 and 2 are invariant. If the failure condition is not reached i.e. the degree neighborhoods are distinct, Steps 3 and 5 also become invariant. Thus, the labeling produced is the canonical labeling of G.

Theorem: Runtime for the algorithm is linear in graph's size i.e. O(V+E).

Proof: Degrees of vertices is done in linear time. Step 2 involving sorting all vertex, neighbor degree pairs in lexicographic order is also done in linear time using radix sort. Step 3 involves sorting over the alphabet {0,1,2,3,….,n-1} is again accomplished in linear time.
Let: the algorithm succeeds for graph G but fails for the other graph H. This makes G and H non isomorphic. If it also succeeds for H, the edges of both G and H are sorted lexicographically using labels of their end-points followed by comparing their lists. These processes take linear time making the overall time linear.

3. Conclusion

Checking for isomorphic property between two input instances of graphs is conjectured a non NP-complete/P class problem making it "open". With consistent research, it kept changing titles from being one "graph-intermediate" type of problem to an "isomorphism-complete" problem. There is a certain class of graphs composed of popular structures like directed acyclic graphs that falls under graph isomorphism-complete. For general applicability, researchers till date are in a dilemma of striving to find a polynomial-time solvable approach or settling with known efficient approaches having imperfect runtimes. The most notably appeciated solution applicable to general graph theory is proposed by Babai. The state-of-the-art boasts of numerous polynomial-time algorithms with ideal runtimes for special cases of graphs, however, achieving a universally acceptable and applicable solution runnable in polynomial time to check isomorphism for two given specimens of graphs, continues to be an open challenge for the GI Problem.

4. References

[1] Kosovskaya, T. M., and N. N. Kosovskii. "Polynomial Equivalence of the Problems 'Predicate Formulas Isomorphism and Graph Isomorphism.'" *SpringerLink*, Pleiades Publishing, 4 Sept. 2019, https://link.springer.com/article/10.1134/S1063454119030105.
[2] Friedland, Shmuel. "On the Graph Isomorphism Problem," January 9, 2008. https://arxiv.org/pdf/0801.0398.pdf.
[3] "Graph Isomorphism Problem." Wikipedia. Wikimedia Foundation, November 7, 2019. https://en.wikipedia.org/wiki/Graph_isomorphism_problem#GI-complete_classes_of_graphs.
[4] Kobler, J., U. Schoning, and J. Toran. "The Graph Isomorphism Problem." Google Books. Google. Accessed November 26, 2019. https://books.google.com/books?hl=en&lr=&id=vhvnBwAAQBAJ&oi=fnd&pg=PA1&dq=Graph+isomor phism+problem&ots=ODtUqW_WCu&sig=oVpdBufrht0vsZcROYspuQI1w_E#v=onepage &q=Graph isomorphism problem&f=false.
[5] Fortin, Scott. "The Graph Isomorphism Problem." The University of Alberta, July 1994. https://pdfs.semanticscholar.org/3da3/ef796c6af4bafe2c0d023fc49f9646f4d949.pdf. http://drops.dagstuhl.de/opus/volltexte/2016/5802/pdf/dagrep_v005_i012_p001_s15511.pdf.
[6] Babai, László; Grigoryev, D. Yu.; Mount, David M. (1982), "Isomorphism of graphs with bounded eigenvalue multiplicity", Proceedings of the 14th Annual ACM Symposium on Theory of Computing, pp. 310–324, doi:10.1145/800070.802206, ISBN 0-89791-070-2.
[7] Babai, László; Kantor, William; Luks, Eugene (1983), "Computational complexity and the classification of finite simple groups", Proceedings of the 24th Annual Symposium on Foundations of Computer Science (FOCS), pp. 162–171, doi:10.1109/SFCS.1983.10.

[8] Babai, László; Luks, Eugene M. (1983), "Canonical labeling of graphs", Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing (STOC '83), pp. 171–183, doi:10.1145/800061.808746, ISBN 0-89791-099-0.
[9] McKay, Brendan D. “Practical Graph Isomorphism.” Vanderbilt University. Accessed November 2019. http://users.cecs.anu.edu.au/~bdm/nauty/pgi.pdf.
[10] Schmidt, Douglas C.; Druffel, Larry E. A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices. *J. Assoc. Comput. Mach.* 23 (1976), no. 3, 433–445.
[11] D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, S. Saurabh, “Fixed-Parameter Tractable Canonization
and Isomorphism Test for Graphs of Bounded Treewidth”, in Proc. of the 2014 IEEE 55th Annual
Symp. on Foundations of Computer Science (FOCS’14), pp. 186–195, IEEE Computer Society,
2014; to appear in SIAM Journal on Computing.
[12] Ponomarenko, Ilia. “On the Isomorphism Problem for Central Cayley Graphs.” Steklov Institute – St. Petersburg, RU. Accessed November 26, 2019. http://drops.dagstuhl.de/opus/volltexte/2016/5802/pdf/dagrep_v005_i012_p001_s15511.pdf.
[13] Toran, Jacob. “Complexity Classes and the Graph Isomorphism Problem.” Universität Ulm, DE. Accessed November 26, 2019. http://drops.dagstuhl.de/opus/volltexte/2016/5802/pdf/dagrep_v005_i012_p001_s15511.pdf.
[14] V. Arvind, J. Köbler, G. Rattan, O. Verbitsky, “On Tinhofer’s Linear Programming Approach to
Isomorphism Testing”, in Proc. of the 40th Int’l Symp. on Mathematical Foundations of Computer
Science (MFCS’15), LNCS, Vol. 9235, pp. 26–37, Springer, 2015.
[15] Boucher, C., and D. Loker. “Graph Isomorphism Completeness for Perfect Graphs and Subclasses of Perfect Graphs,” May 2006. https://cs.uwaterloo.ca/research/tr/2006/CS-2006-32.pdf.
[16] R. Mathon. A note on the graph isomorphism counting problem. Information Processing Letters, 8:131{132, 1979.
[17] J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphism. Information Sciences, 19:229{250, 1979}
[18] McKay, Brendan, and Adolfo Piperno. “Practical Graph Isomorphism.” Nauty Traces – Introduction, October 15, 2019. http://pallini.di.uniroma1.it/Introduction.html.
[19] Weisstein, Eric W. “Isomorphic Graphs.” MathWorld. Accessed November 26, 2019. http://mathworld.wolfram.com/IsomorphicGraphs.html.
[20] Uehara, Ryuhei, et al. “Graph Isomorphism Completeness for Chordal Bipartite Graphs and Strongly Chordal Graphs.” *Discrete Applied Mathematics*, North-Holland, 3 Aug. 2004, www.sciencedirect.com/science/article/pii/S0166218X0400191X#bib2.
[21] Czajka, Tomek, and Gopal Pandurangan. “Improved Random Graph Isomorphism.” Journal of Discrete Algorithms. Elsevier, January 27, 2007. https://www.sciencedirect.com/science/article/pii/S1570866707000147.
[22] L. Babel, I.N. Ponomarenko, G. TinhoferThe isomorphism problem for directed path graphs and for rooted directed path graphs J. Algorithms, 21 (1996), pp. 542-564