International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552, Volume-8, Issue-3, May 2020
https://doi.org/10.21276/ijircst.2020.8.3.15
www.ijircst.org

Reducing Complexity of Graph Isomorphism Problem

Shalini Bhaskar Bajaj

ABSTRACT- Graph isomorphism has been discussed in the literature as NP-hard problem. It has applications in various areas. Work done earlier in this area employs backtracking for identifying isomorphism between given two graphs as a result with the increase in size of the graph the time to solve the problem becomes exponential. The proposed work presents a polynomial time algorithm (PTGI) which tries to solve graph isomorphism between given two directed graphs. Some distinguishing features associated with each vertex are stored. These features are exploited in dividing the vertices of the graph into equivalence classes from which canonical representation of the graph is generated. Comparing these features of the vertices of the given two graphs solves isomorphism in polynomial time.

KEYWORDS- Graph isomorphism, canonical representation, equivalence class, polynomial

I. INTRODUCTION

Graph isomorphism is one of the hardest problems that need attention. It is defined as a bijection between the vertices(nodes) of given graphs G1 and H1. A graph G1 can be represented by *G1 = (V1, E1).* Depending on whether the edge has a direction or not; graphs can be classified as directed or undirected. Numerous algorithms were presented earlier for solving graph isomorphism. In the year 1976, Ullmann [10] proposed an algorithm that uses backtracking and reduces the size of the search space. In the same year, Schmidt and Druffel [9] proposed another backtracking algorithm that establishes initial partitions in the graph by referring to the information stored in the distance matrix of the graph. Around the same period, Weisfeiler and Lehman [11] worked on canonical representation of a graph in order to find stable partitions of vertices in the graph. Nauty algorithm was proposed by McKay [5] [6] in the year 1981 that constructs canonical representation for the graph. It is considered to be the fastest.

Manuscript received May 19, 2020

Shalini Bhaskar Bajaj, Professor, Department of Computer Science and Engineering, Amity University Haryana, Gurugram, India (e-mail: shalinivimal@gamil.com)

algorithm for finding isomorphism in graphs but Miyazaki [7] in the year 1997 showed that there are certain categories of graphs for which this algorithm takes exponential time. In the year 2001, Cordella, Foggia, Sansone and Vento [1] proposed VF2 which is a depth first search algorithm. VF2 uses a set of rules to efficiently prune the search space. Another graph isomorphism algorithm 'conauto' was discussed in the year 2004 by Presa and Farnandez [8] that exploits both canonical representation and backtracking to solve isomorphism. As the size of the graph increases, computational time also increases exponentially and restricts the algorithm applicability to smaller graphs. In the year 1974, Hopcraft and Wang [3] developed polynomial time algorithm for solving graph isomorphism that works only for planar graphs. Luks [4] in the year 1982 worked with bounded valance graphs for resolving isomorphism in polynomial time. Though the above mentioned algorithms can be completed in polynomial time but they impose restrictions on the graphs. In the year 2018, a parallel frequent subgraph mining algorithm was proposed wherein the frequent subgraphs were mined in a single large graph using Apache Spark framework [20]. In the same year, Kaleido [19] was proposed that is an out of core graph mining system on a single machine. Another algorithm was proposed in the year 2018, which used Facenet graph mining algorithm [16] to generate useful knowledge from the communication transaction data. ASAP algorithm [17] was also proposed in the same year which produces state of art results in graph approximation theory. The ASAP algorithm was also extended to the general graph patterns in distributed settings. In the year 2019, fractal [18] was proposed for supporting distributed graph pattern mining. Fractal is a high performance and high productivity system which employs a dynamic load balancing system based on locality aware stealing and hierarchical mechanism. The proposed approach aims at developing an algorithm for finding graph isomorphism in polynomial time. This is based on finding canonical representation for the given graphs and grouping the vertices into a set of equivalence classes so that the problem of isomorphism can be reduced to solving isomorphism between the vertices in the same equivalence class. For each vertex, the important characteristic features like direct / indirect path(s), shortest path length and count of the vertices traversed in all available shortest paths are stored in the form of adjacency list. Vertices belonging to same equivalence class can be differentiated from each other on the basis of their interaction with the vertices belonging to other equivalence classes. Dividing vertices of a given graph into equivalence classes solves isomorphism problem in polynomial time. Count of vertices in all the equivalence classes in the two graphs is same; if the given two graphs under test exhibit isomorphism.

The present paper is divided into different sections focusing on providing overview of the earlier work, giving details on the proposed work, explaining the proposed work with the help of an example and arriving at the computational complexity of the proposed work. Section 2 gives overview of the work done in the previous years in this area, section 3 discusses the proposed approach- PTGI, section 4 presents detailed discussion on the proposed algorithm-PTGI, section 5 gives computational complexity of PTGI and last section gives concluding remarks.

II. OVERVIEW OF THE EXISTING WORK

This section focuses on providing overview of the major algorithms developed in the field of graph isomorphism. Graph mining problem has gained importance gradually and lot of new closeness measures were proposed for image and pattern recognition. [12] The graph isomorphism problem along-with the generalizations is essential in large application areas which are directly dealing with similarity problems such as pattern and image recognition [13].

Algorithms discussed in the literature use either canonical representation or backtracking or both. Nauty algorithm [5] [6] identifies automorph groups in a set of vertex colored graphs. This algorithm is based on group theory and provides information about the set of generators, size of the group and the orbits of the group. It also generates canonically labeled isomorph to assist in testing isomorphism. It uses backtracking which can be described in terms of search tree. Except in simple cases, only parts of the tree are generated; other parts are shown either equivalent to the already existing parts or are shown as uninteresting.

Figure 1 gives sample graph for generating automorphs using Nauty agorithm and Figure 2 shows part of the search tree generated using this algorithm. In Figure 2, DEF and ABC are equitable partitions. Partition DEF is considered for further partitioning and is represented as the target cell (underlined in Fig. 2). The labels on tree edges represents the target set element being fixed, for example, the first edge shows that element D is fixed and second edge shows that element E is fixed. Further partitioning is done in the same way. All leaves in the search tree generated are equivalent. Figure 2 show automorphs (AC) (DF) and (BC) (EF).

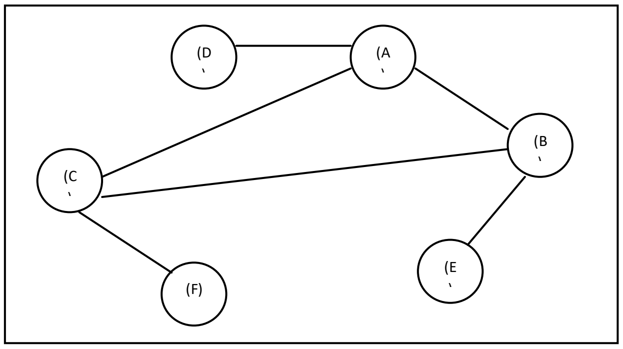

Fig 1: Sample graph for generating automorphs using Nauty

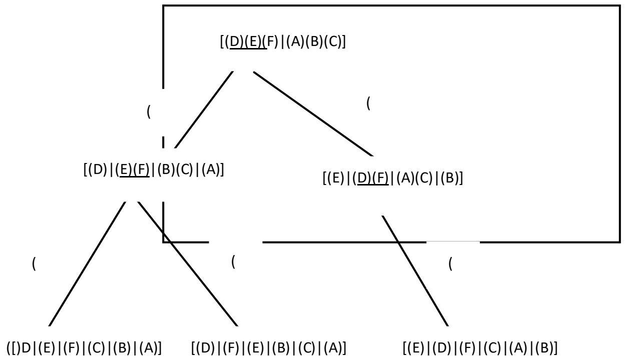
[(D)(E)(F)|(A)(B)(C)]
(
(
[(D)|(E)(F)|(B)(C)|(A)]
[(E)|(D)(F)|(A)(C)|(B)]
(
(
(
([)D|(E)|(F)|(C)|(B)|(A)]
[(D)|(F)|(E)|(B)|(C)|(A)]
[(E)|(D)|(F)|(C)|(A)|(B)]

Fig 2: Part of search tree generated using Nauty

VF2 [1], a depth first algorithm is a deterministic matching method that can be used for identifying both graph and subgraph isomorphism. The algorithm does not impose any constraints on the graph topology and hence can exploit the semantic information associated with nodes and edges. As this algorithm takes less memory, it can be used with large graphs. The algorithm prunes the search space by using certain set of rules. The memory requirement of this algorithm is linear in number of nodes and the number of edges.

Conuto algorithm [8] tries to resolve isomorphism by identifying automorphs and generating series of partitions. It works on the following steps: sequence of vertex partitions is built; autotrophism is identified without backtracking technique; Even if automorphism is not resolved till this stage, sequence of partitions are generated in the second graph (that matches first graph) using backtracking. Conauto used adjacency matrix representationand checks the count of vertices and edges in the given graphs. Graphs with same count for vertices and edges qualify for partitioning.

Conauto algorithm behaves uniformly for all classes of graphs on the other hand Nauty and VF2 algorithms have specific classes of graphs for which they take large amount of time even if graphs are smaller in size. Worst case time complexity for the all these algorithms is exponential as backtracking is used. In 2016, Mortuza proposed polynomial time algorithm for graph isomorphism [15] for a specific category of the graphs that are not locally triangle-free. Another algorithm, GI-Ext is capable of dealing graphs with large number of edges (approx. thousand) but is more suited for partial isomorphism as compared to subgraph isomorphism problem. [14].

III. PROPOSED APPROACH.-PTGI

In PTGI, adjacency matrix definition is modified as discussed below. For graph G1, adjacency list can be represented by A_L (G1) for the given n vertices. Each node in the adjacency list represents the shortest distance between the given two vertices x and y. The characteristics features of the vertices x and y helps in identifying isomorphism. The format used for storing information in adjacency list is discussed as follows. For storing information about the shortest path from x to y the adjacency list for vertex x will have the value [y]||[shxy]||[shdxy]||[Cyx]||[Cxy] and for vertex y it will be [x]||[shyx]||[shdyx]||[Cxy]||[Cyx] where

[sh_{xy}] = dp if there is a direct path between x and y;

= ip if there is an indirect path between x and y;

[shd_{xy}]= shortest path between x and y;

[C_{yx}] = count of the vertices in the shortest path between y and x (including x and y);

[C_{xy}] = count of the vertices in the shortest path between x and y (including x and y).

Breadth first search is used to identify shortest paths between all vertex pairs and details are noted using the format defined above. Canonical representation for the graph is generated once the graph is recorded as adjacency list. For finding canonical representation, all distinct values in the adjacency list are identified and their occurrence is recorded for each vertex along-with the in-degree and out-degree for the vetex. This gives the summary regarding the characteristics of the vetex which is represented by CSV (Characteristic Summary of the Vertex). For different vertices CSV is recorded and is arranged in increasing or decreasing order and is assigned a class number. The vertices sharing same CSV are assigned to the same equivalence class.

Graphs under consideration are compared once their canonical representation is generated. For finding isomorphism, the given two graphs must have the same number of equivalence classes and each class must have same number of vertices belonging to it. Vertices in the same equivalence class from the two graphs are compared for their distinguishing features. If vertices of all the equivalence class shows mapping between their vertices (in the given graphs), isomorphism is detected else the two graphs are not isomorphic.

A. Algorithm:

1. Traverse all possible shortest paths between vertex *x* and *y* in the given graph *G1* using Breadth First Search algorithm. If there is only one shortest path between vertex *x* and *y* then find the cardinality of the vertices encountered in the identified shortest path else find the cardinality of the union of the vertices encountered in all the identified shortest paths.

2. Create Adjacency list representation, A_L(G1) for graph G1 (as explained above).

3. Compute canonical representation for the given graph G1 (as explained above).

4. Perform step 1 to step 3 for given graph H1.

5. Compare the vertex belonging to the same equivalence class from the adjacency lists of graph *G1* and *H1*. If all the equivalence classes in the two graphs have same number of vertices, then the vertices of the respective equivalence classes from the two graphs are matched for their distinguishing features. If all the vertexes of all the equivalence classes show same features, isomorphism is detected.

IV. ILLUSTRATIONS OF THE PROPOSED ALGORITHM – PTGI

PTGI is explained as follows using an example. Figure 3 and Figure 4 gives two graphs G1and H1 respectively. Graph G1 and graph H1 has a set of twenty vertices. Figure 5 shows the adjacency list of graph G1

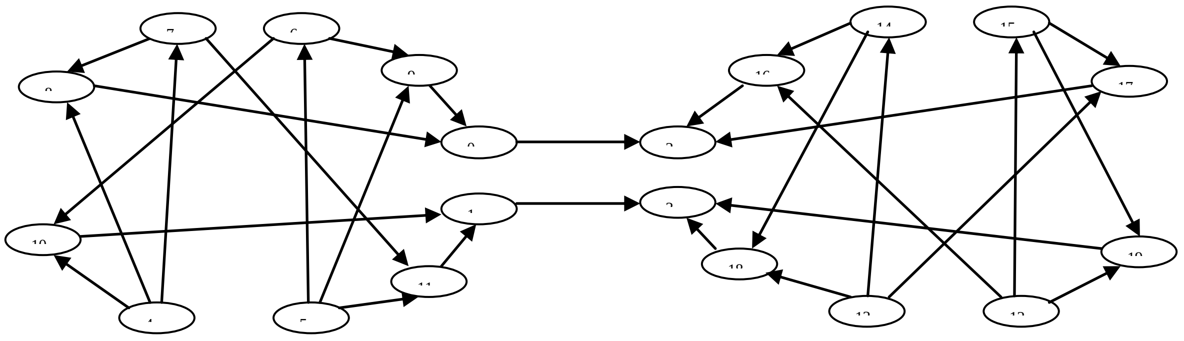

Fig. 3: Graph *G1* for testing isomorphism

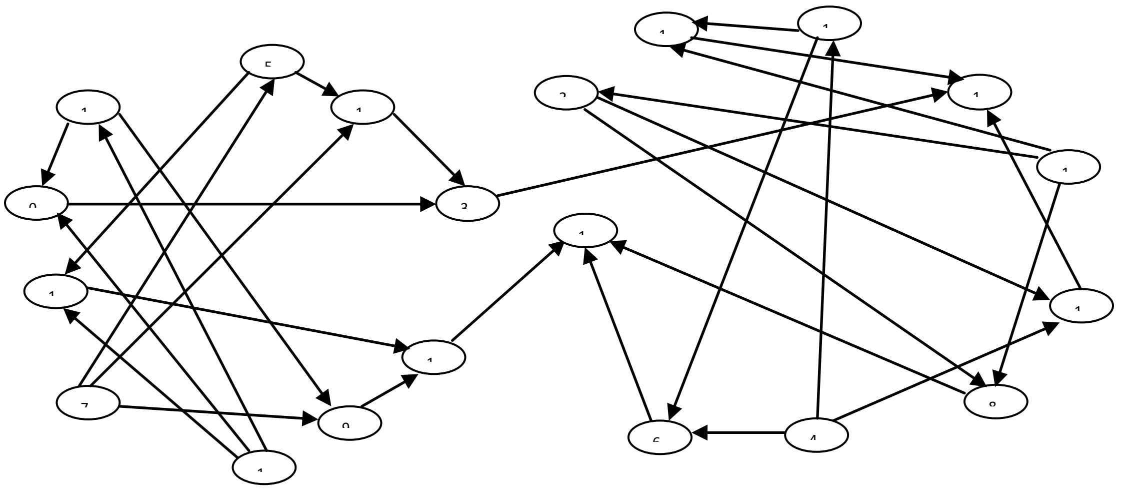

Fig. 4: Graph H1 for testing isomorphism

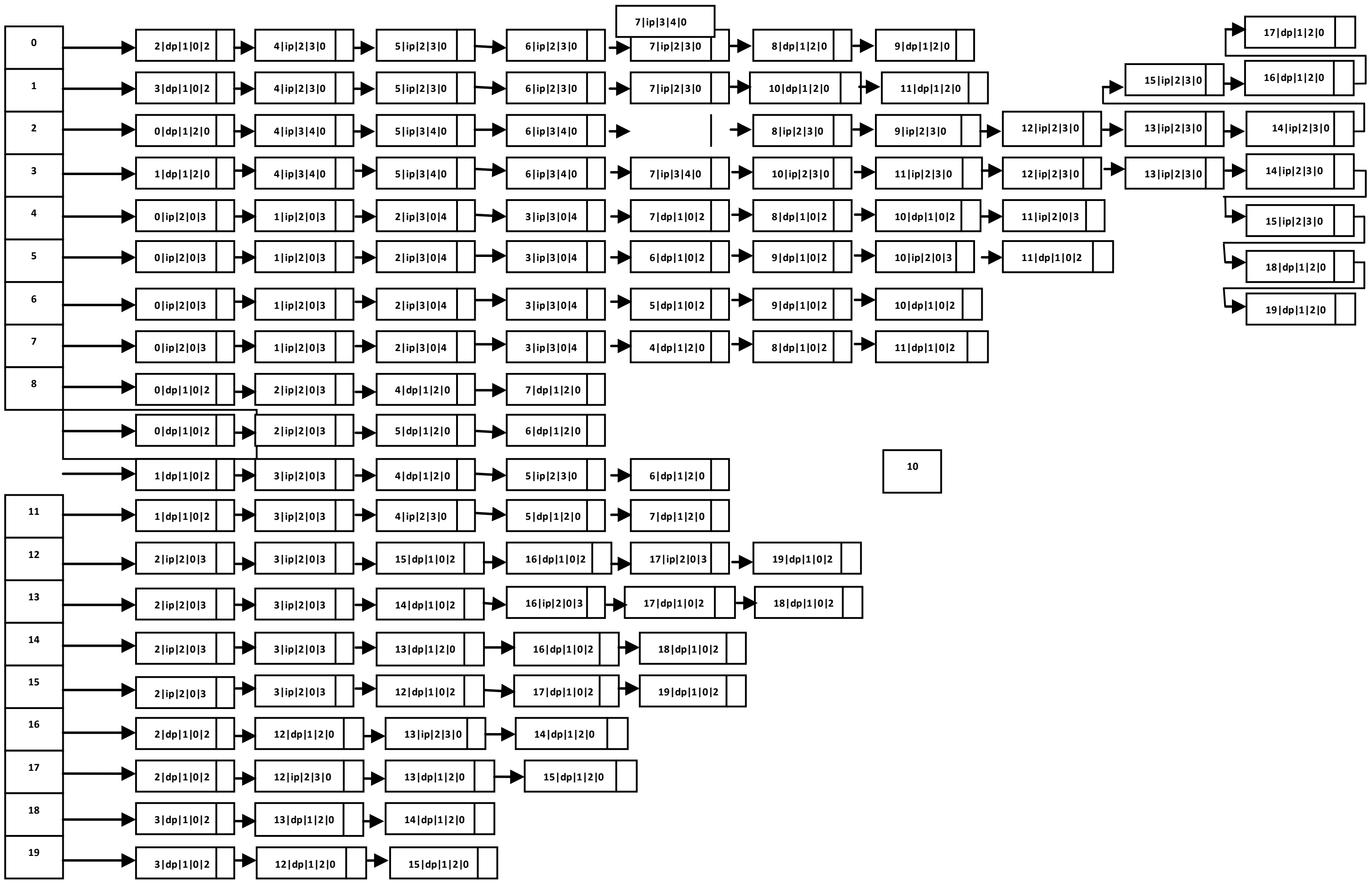

Fig. 5: Adjacency list representation for storing shortest paths from different nodes in graph G1

Canonical representation also known as Characteristics Summary of the Vertex (CSV) of graph G1 generated using the adjacency list is given in Table 1. CSV for all the vertices is read vertically and is arranged in increasing or decreasing order. Vertices with same CSV falls under the same equivalence class. In the sample example discussed here, for graph G1, the CSV's of the vertices are arranged in increasing order in order to give equivalence class labels. For example, CSV of vertices 18 and 19 is read as 00002121 (read the column of vertex 18 and 19 from top to bottom) and the CSV of the vertices 8 and 9 are read as 00012121. Since 00002121 is the smallest number in the CSV thus vertex 18 and 19 are given equivalence class label 1. CSV and equivalence class number of the different vertices of graph G1 are given in Table 1(a). Equivalence class labels in table 1 are given in the last row. In the similar fashion adjacency list and CSV of graph H1 is generated. CSV and equivalence class number of the different vertices of graph H1 are given in Table 2(a).

Table1: Characteristic Summary of vertices of graph *G1*

Vertex Label / Vertex Characteristic Summary	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
ip|3|4|0	0	0	4	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ip|3|0|4	0	0	0	0	2	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0
ip|2|3|0	4	4	6	6	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0
ip|2|0|3	0	0	0	0	3	3	2	2	1	1	1	1	3	3	2	2	0	0	0	0
dp|1|2|0	2	2	3	3	0	0	1	1	2	2	2	2	0	0	1	1	2	2	2	2
dp|1|0|2	1	1	0	0	3	3	2	2	1	1	1	1	3	3	2	2	1	1	1	1
InD_V	2	2	3	3	0	0	1	1	2	2	2	2	0	0	1	1	2	2	2	2
OutD_V	1	1	0	0	3	3	2	2	1	1	1	1	3	3	2	2	1	1	1	1
Equivalence Class Label	7	7	10	10	9	9	8	8	2	2	6	6	4	4	3	3	5	5	1	1

Table 1(a): CSV of different vertices of Graph G1arranged in increasing order

Vertex of graph G1	**CSV**	**Equivalence Class Label**
18 and 19	00002121	1
8 and 9	00012121	2
14 and 15	00021212	3
12 and 13	00030303	4
16 and 17	00102121	5
10 and 11	00112121	6
0 and 1	00402121	7
6 and 7	02021212	8
4 and 5	02030303	9
2 and 3	40603030	10

Table 2: Characteristic Summary of vertices of graph *H1*

Vertex Characteristic Summary \ Vertex Label	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
ip|3|4|0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4
ip|3|0|4	0	0	0	0	0	2	0	2	0	0	0	0	0	0	2	0	0	2	0	0
ip|2|3|0	0	1	0	4	0	0	0	0	0	1	4	0	0	0	0	1	1	0	6	6
ip|2|0|3	1	1	2	0	3	2	0	3	0	1	0	2	3	1	3	0	0	2	0	0
dp|1|2|0	2	2	1	2	0	1	2	0	2	2	2	1	0	2	0	2	2	1	3	3
dp|1|0|2	1	1	2	1	3	2	1	3	1	1	1	2	3	1	3	1	1	2	0	0
InD_V	2	2	1	2	0	1	2	0	2	2	2	1	0	2	0	2	2	1	3	3
OutD_V	1	1	2	1	3	2	1	3	1	1	1	2	3	1	3	1	1	2	0	0
Equivalence Class Label	2	6	3	7	4	8	1	9	1	6	7	3	4	2	9	5	5	8	10	10

Table 2(a): CSV of different vertices of Graph H1arranged in increasing order

Vertex of graph G1	**CSV**	**Equivalence Class Label**
6 and 8	00002121	1
0 and 13	00012121	2
2 and 11	00021212	3
4 and 12	00030303	4
15and 16	00102121	5
1 and 9	00112121	6
3 and 10	00402121	7
5 and 17	02021212	8
7 and 14	02030303	9
18 and 19	40603030	10

Table 3: Isomorphism between the vertices of graph G1 and graph H1

Equivalenc e Class Label	graph G1	graph H1	Equivalence Class Label	graph G1	graph H1
1	{18, 19}	{6, 8}	6	{10, 11}	{1, 9}
2	{8, 9}	{0, 13}	7	{0, 1}	{3, 10}
3	{14, 15}	{2, 11}	8	{6, 7}	{5, 17}
4	{12, 13}	{4, 12}	9	{4, 5}	{7, 14}
5	{16, 17}	{15, 16}	10	{2, 3}	{18, 19}

By matching the equivalence class labels in Table 1 and Table 2, it can be concluded that isomorphism exists between the following vertex set of graph *G1* and *H1* (refer Table 3).
In order to resolve isomorphism between the vertices belonging to the same equivalence class of the given two graphs, distinguishing features noted for vertices of the two graphs are compared. For example, equivalence class 1 of graph G1 has vertices 18 and 19. From Figure 5, it is observed that vertex 18 has shortest paths with vertices 3, 13, 14 and vertex 19 has shortest paths with vertices 3, 12, 15. Further, it is observed from Table 3 that vertex 3 belongs to equivalence class 10, vertices 12 and 13 belong to equivalence class 4 and vertices 14 and 15 belong to equivalence class 3. Similarly, vertices 6 and 8 of equivalence class 1 in graph H (refer Table 3), interact with vertices of equivalence classes 3, 4 and 10. Hence, all the vertices of equivalence classes 3, 4 and 10 are considered in

resolving isomorphism between the vertices of equivalence class 1.

Table 4: Vertices of equivalence class 1 of graph *G1*

Vertex of EC^+ 3, 4, 10 / Vertex of EC^+ 1	EC^+ 3		EC^+ 4		EC^+ 10	
	14	15	12	13	2	3
18	d|1|2|0	i|0|0|0	i|0|0|0	d|1|2|0	i|0|0|0	d|1|0|2
19	i|0|0|0	d|1|2|0	d|1|2|0	i|0|0|0	i|0|0|0	d|1|0|2

Table 5: Vertices of equivalence class 1 of graph *H1*

Vertex of EC^+ 3, 4, 10 / Vertex of EC^+ 1	EC^+ 3		EC^+ 4		EC^+ 10	
	2	11	4	12	18	19
6	i|0|0|0	d|1|2|0	d|1|2|0	i|0|0|0	i|0|0|0	d|1|0|2
8	d|1|2|0	i|0|0|0	i|0|0|0	d|1|2|0	i|0|0|0	d|1|0|2

Rows of Tables 4 and 5 show vertices of equivalence class 1 of graph G1 and H1 respectively for which isomorphism is to be resolved, the columns of the given Tables show the vertices of equivalence classes 3, 4 and 10 with which vertices of equivalence class 1 has shortest path and elements of the given Tables show the distinguishing features on the basis of the interaction of the vertices of equivalence class 1 with the vertices of equivalence classes 3, 4 and 10.

From Tables 4 and 5, it is observed that isomorphism can be resolved between the vertices of equivalence class 1 of graph G1 and H1. It is seen from Tables 4 and 5 that the distinguishing features match between the vertex pairs (18, 14) and (6, 11) of Graph G1 and H1 and likewise all the matching pairs are found which is shown in Table 6.

Table 6: Vertex pairs of graph G1 and graph H1 matched for respective vertex pair values

graph G1	graph H1
(18, 14)	(6, 11)
(19, 14)	(8, 11)
(18, 13)	(6, 4)
(19, 13)	(8, 4)
(18, 15)	(6, 2)
(19, 15)	(8, 2)
(18, 2)	(6, 18)
(19, 2)	(8, 18)
(18, 12)	(6, 12)
(19, 12)	(8, 12)
(18, 3)	(6, 19)
(19, 3)	(8, 19)

Hence, it can be inferred that in equivalence class 1, following vertices of graph *G1* and *H1* are isomorphic (refer Table 7).

Table 7: Vertices belonging to equivalence class 1 of graph *G1* and *H1* showing isomorphism

graph G1	graph H1
18	6
19	8

For all other equivalence classes same procedure is repeated to resolve isomorphism between the vertices of the given graphs. Table 8 show vertices of the two graphs *G1* and *H1* that shows isomorphism.

Table 8: Vertices of graph *G1* and *H1* showing isomorphism

graph G1	graph H1	graph G1	graph H1
18	6	15	2
19	8	12	12
8	0	13	4
9	13	16	16
14	11	17	15
10	1	7	17
11	9	4	14
0	3	5	7
1	10	2	18
6	5	3	19

Hence, it is seen that isomorphism problem can be resolved between the given two graphs by comparing vertices belonging to the same equivalence class which can be computed in polynomial time.

V. COMPUTATIONAL COMPLEXITY OF PTGI

Computational complexity of the proposed algorithm PTGI is discussed in this section. Following definition, lemma and theorem will prove that the proposed algorithm terminates in polynomial time.

Definition 1Given vertices n1 and edges m1 of the graph G1, let shd be the total number of shortest paths such that n1 X m1 < shd < n1 X n1 then the complexity in terms of space of the given graph G1 is less than n1 X n1.

Lemma 1 Given vertices n1 and edges m1 in graph G1, breadth first search algorithm can be used for finding all possible shortest paths between vertex x and y in graph G1 in O(n1 + m1) time [2].

Theorem 1 Given vertices n1 and edges m1 in graph G1, the adjacency list representation for storing shortest path details between all vertex pair for the given graph G1 can be computed in O $(n1^3 + m1n1^2)$ time.

Proof: Let the total number of shortest paths identified between given vertex x and all other vertices be shd_x, then for n1 such vertices, $\sum_{x=1}^{n1} s_x = n1^2$. As it takes atmost (n1+m1) iterations (refer Lemma 1) to find shortest distance and to group all vertices encountered if there are more than one shortest path between the given pair of vertex (x, y). Therefore, order of complexity is given by O $(n1^3 + m1n1^2)$.

Theorem 2 Equivalence class labels can be computed in polynomial time

Proof: Let *n* be the number of vertices. In order to find the equivalence class of the vertex, its CSV needs to be compared with the CSV of all the vertices. Thus, in order to compare the CSV of a vertex with all the other vertices it takes *n* iterations. Therefore, the total number of iterations needed to compare the CSV of all the vertices with every other vertex is *n*n*. Hence, the complexity of finding equivalence class labels is O (n^2).

Theorem 3 Graph isomorphism problem can be solved in polynomial time

Proof: Let *i* be the number of equivalence classes; *j* be the number of vertices in each class for which isomorphism is to be identified by matching the values of *k* vertices in given graphs *G1* and *H1*. As there are *j* vertices to be matched for *k* values, it takes at most $j * k$ iterations to find the mismatch and at most $j * k$ iterations to resolve the mismatch. This may be repeated at most $j * k$ times, till isomorphism has been resolved in a given equivalence class. The above procedure is followed for all equivalence classes. Hence, time complexity for the given theorem is $i * j * k * (j * k + j * k)$. In the worst case, there is only one equivalence class (i=1), hence, all vertices lie in the same equivalence class (j = n1) and are matched for all the vertex values (k = n1). Hence, computational complexity becomes $1*n1*n1 * (n1 * n1 + n1 * n1) = n1^2 * (n1^2 + n1^2)$ and the order of complexity is given by $O(n1^4)$.

Order of complexity for the given algorithm can be computed by combining the complexity for theorem 1 and 2 and is given by $O(m1n1^2 + n1^3) + O(n1^4)$. Thus the complexity of the algorithm is given by $O(n1^4)$.

VI. CONCLUSIONS AND FUTURE SCOPE

In this paper, a polynomial time algorithm has been proposed for finding graph isomorphism. A new approach of using the concept of equivalence classes in conjunction with canonical representation of graphs has been proposed. Every vertex has some features that uniquely identifies it and distinguishes it from the other vertices. Exploiting these features by placing them together is the key to generate the canonical representation for the graph. The vertices that belong to the same equivalence class shows same features but can be distinguished from each other by their interaction with the other vertices. The future scope of the proposed algorithm can be seen in the field of networks. Network can be of any type such as network of modules, communication or flow analysis. The proposed algorithm finds its application in finding functional modules, program control flow analysis, mining communication networks, intrusion network analysis etc.

REFERENCES

[1] L. P.Cordella, P. Foggia, C. Sansone and M. Vento, An improved algorithm for matching large graphs, In Proc. of the 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Ischia (Italy), (2001).

[2] T. Coreman, C. Leiserson and R. Rivest, Introduction to Algorithms, Cambridge, MA: MIT Press, 1990.

[3] J. Hopcroft and J. Wong, Linear Time Algorithm for Isomorphism of Planar Graphs, Proc. 6th Annual ACM Symp. Theory of Computing, (1974) 172-184.

[4] E.M. Luks, Isomorphism of Graphs of bounded valance can be tested in polynomial time, Journal of Computer System Science, (1982) 42-65.

[5] B. D. McKay, The nauty page, March 2004, http://cs.anu.edu.au/~bdm/nauty/

[6] B.D. McKay, Practical Graph Isomorphism, Congressus Numerantium, 30 (1981) 45-87.

[7] T. Miyazaki, The complexity of McKay's canonical labeling algorithm, in Groups and Computation, II L. Finkelstein and W. M. Kantor, eds., Amer. Math. Soc., Providence, RI, (1997) 239-256.

[8] Jose Luis Lopez-Presa and Antino Fernandez, Graph Isomorphism Testing Without Full Authmorphism Group Computation, vol. 4 (2004), No. 3 (TR-GSYC-2004-3).

[9] D.C. Schmidt and L.E. Druffel, A Fast Backtracking Algorithm to Test Directed Graphs for Isomorphism Using Distance Matrices, 23 (1976) 433-445.

[10] J.R. Ullman, An Algorithm for Subgraph Isomorphism, Journal of the Association for Computing Machinery, 23 (1976) 31-42.

[11] B. Weisfeiler and Lehman, On Construction and Identification of Graphs, Lecture Notes in Math, Springer, Berlin, 558 (1976).
[12] Jose Luis Lopez-Presa and Antonio Fernandez Anta. Fast Algorithm for Graph Isomorphism testing, In SEA, volume 5526 of LNCS, pages 221-232, 2009
[13]S. Auwatananmongkol. Inexact graph matching using a genetic algorithm for image recognition. Pattern Recognition Letters, 28912), pages 1428-1437, 2007
[14]D. Conte, P. Foggia and M. Vento. Challenging complexity of maximum common subgraph detection algorithms: A performance analysis of three algorithms on a wide database of graphs. Journal of Graph Algorithms and Applications, 11(1), pages 99-143, 2007
[15] Fahad Bin Mortuza. A Polynomial Time Graph Isomorphism Algorithm for Graphs that are not locally Triangle-free, Cornell University Library, 2016
[16] Imelda Atastina, Benhard Sitohang, G.A.Putri Saptawati and Veronica S. Moertini. An implementation of graph mining to find evolution of communication data records. In the Proceedings of the 2018 Intl. Conf. on Data Science and InformationTechnology (DSIT '18), Singapore, July 20-22, 2018, pp. 79-84, 2018
[17] Anand Padmanabha Iyer, Zaoxing Liu and Xin Jin, Shivaram Venkataraman, Vladimir Braverman and Ion Stoica, ASAP: Fast, Approximate Graph Pattern Mining at Scale, Open Access to the Proc. of the 13th USENIX Symp. On Operating System Design and Implementation, October 8-10, 2018, Carlsbad, CA, USA
[18] Vinicius Dias, Carlos H.C. Teixeira, Dorgival Guedes, Wanger Meira and Srinivasan Parthasarthy, Fractal: A General-Purpose Graph Pattern Mining System, In Proc. Of the 2019 Intl. Conf. on Management of Data (SIGMOD '10), Amserdam, Netherlands, Jun 30- Jul 5, 2019, pp. 1357-1374
[19] Cheng Zhao, Zhibin Zhang, Peng Xu, Tianqi Zheng and Xueqi Cheng, Kaleido: An efficient out of core graph mining system on a single machine, Proc. Of the VLDB Endowment, vol. 11, no. 13, 2018, DOI: https://doi.org/TBD
[20] Fengcai Qiao, Xin Zhang, Pei Li, Zhao Y un Ding, Shanshan Jia and Hui Wang, A parallel approach for frequent subgraph mining in a single large graph using spark, Appl. Science, 2018, vol. 8, pp. 230, DOI: 10.3390/app8020230

ABOUT THE AUTHOR

1. **Shalini Bhaskar Bajaj** is working as Professor and HoD (Computer Science and Engineering). She completed her Ph.D. from IIT Delhi and is working in the field of Data Mining and Analytics. She is having 20 years of experience as an academician. She has more than 50 publications in reputed journals/conferences.