Tabriz Journal of Electrical Engineering (TJEE), vol. 55, no. 1, 2025 Serial no. 111
DOI: 10.22034/tjee.2024.57285.4657

An Improved Heuristic Algorithm for the Graph Isomorphism Problem

Somayeh Check, Ali Nourollah *

Software Systems Research and Development Laboratory / Faculty of Computer Engineering, Shahid Rajaee Teacher Training University, Tehran, Iran.
somayeh_check@sru.ac.ir, nourollah@sru.ac.ir*
*Corresponding author

Received:01/07/2023, Revised:05/07/2024, Accepted:30/09/2024.

Abstract
The Graph Isomorphism Problem (GIP) is an open problem because of its high computational complexity. No polynomial-time deterministic algorithm has been proposed yet, and heuristic and meta-heuristic approaches have been the only ways to solve it. Even its belonging to the NP-complete problems has not yet been proven. This paper introduces a simple but efficient polynomial-time and -space algorithm, to determine the isomorphism of connected unlabeled graphs. The proposed algorithm introduces two functions that compute the features for all vertices and edges. The outputs of the functions provide a canonical labeling for the given graphs, and comparison of these labels specifies isomorphism of the graphs. The experimental results show that the proposed algorithm correctly detects the isomorphism of the graphs in more than 99% of the cases. The algorithm has $O(n^3)$ time, where n is the number of vertices of the given graphs.

Keywords
Graph isomorphism, polynomial-time algorithm, heuristic algorithm, canonical labeling.

1. Introduction

Graphs are powerful mathematical structures used to represent objects (represented as vertices) and their relationships (represented as edges). Graphs can be used to model many real-world problems. Many applications introduced by researchers in vertex labeling and multi-labeling as classification [1] and the role of graphs with sensitive edges in social networks [2].

Automorphism, homomorphism, that are belong to Graph Isomorphism (GI) scope are the fundamental scope in graph theory in which compare the structure of two finite graphs. They are correlated and introduce them "isomorphic" if the two graphs are equal in structural skeleton or shapes.

This problem has been studied by computer scientists, mathematics, chemistry, and medical sciences over the past few decades. This issue has been theoretically and practically raised in computer branches in various fields, including computer pattern recognition and vision [3-5], data mining [6], image processing [7], social networks [8], two-dimensional, three-dimensional, and four-dimensional scenes [9], chemical bonds [10], and cell biology [11].

For GI, it is necessary to have a one-to-one and spanning mapping (bijection) between the vertex sets of two graphs with edge-preserving bijection, which is ultimately equal to a structure-preserving bijection. At least $\Omega(n!)$ time is required to verify the existence of bijective mapping between two n-order graphs, because all the different ways of mapping from vertex to vertex must be explored, and within each of these comparisons there must be a correspondence between the edges as well, which in turn has a distinct time-complexity and this value is considerable ("A straightforward enumerative algorithm might require 40 years of running time on a very high-speed computer in order to compare two 15-node graphs" [12]. In other words, trivial isomorphic checking requires $\Omega(n!)$, it needs nearly 40 years to run for two graphs with $n = 15$) .

Therefore, the techniques that are different from reviewing all the cases to determine the graph isomorphism have been proposed, in this paper a heuristic algorithm to solve the problem of GI for unlabeled graphs by canonical labeling is proposed to reduce the time complexity and to present a polynomial-time algorithm to improve the results.

Babai and Luks have proposed an algorithm [13] that computes the canonical labeling of generic graphs in $O\left(e^{\sqrt{n+O(1)}}\right)$, which is the best time for GIP algorithms at present. In another paper [14], Babai et al. proposed a simple algorithm that can perform canonical labeling operations on most graph vertices in linear time. In fact, it has been shown that the probability that a graph with n vertices is labeled with that algorithm is typically of $1 - \sqrt[7]{1/n}$ (large enough for order n). Subsequently, Babai and Kučera [15] improved this result and proposed a conventional linear tagging algorithm with a probability of $exp(-cn \log n/\log\log n)$ failure whose two major drawbacks are the results in small graphs and regular graphs. The fastest algorithm for detecting the isomorphism of general graphs has the time complexity of $O\left(e^{\sqrt{n \log n}}\right)$ for graph with n vertices based on simple finite group classification method [16]. The most

powerful algorithm currently available is the Nauty&Traces package [17]. Despite its impressive performance in most cases, it is exponential-time for some graph families [18].

No polynomial-time algorithm has been presented yet to determine the isomorphism of general graphs, even it has not been proved NP-complete; hence, it is included in the NP problem class [19]. However, polynomial-time algorithms are known for specific classes of graphs, such as trees [20] and planner graphs [21]. In addition, some polynomial-time algorithms proposed to detect isomorphism are not suitable for implementation or are not practical because of their hidden complexity [22].

In the paper published in 2020 [23], an algorithm was presented that layered the vertices based on their distance from the center of the graph(Eccentricity), then transmits the parenthetical code according to the priority of the layers. The accuracy of the algorithm in detecting isomorphism on simple connected graphs was greater than 98%, according to the tests conducted in the article.

In this proposed algorithm, the canonical labeling of each graph is obtained by computing f_v, which is a value for each vertex and f_e which contains two values for each edge. f_e includes two values $f_e(\overrightarrow{u,v})$ and $f_e(\overrightarrow{v,u})$ for each edge $(u,v) \in E$, which assigns a value to each of the end-vertices of the edge. Calculating the values of f_v and f_e is similar to calculating the value of "Betweenness Centrality" in graph theory, which is a measure of centrality in a graph based on shortest paths [24].

The results of the proposed algorithm on a set of 10,000 samples of input graphs were 100% accurate, and nearly 100% in the complete set of graphs. One case of misidentification of the algorithm is discussed in this paper, and the weakness of the algorithm is analyzed using this example. In addition, a set of regular graphs is considered as an input because regular graphs are problematic graphs in canonical labeling-based GI algorithms that the proposed algorithm correctly identifies with an average accuracy above 99.9%.

After giving the preliminary definitions in Section 2, Section 3 details the proposed algorithm, the results of the implementation of the algorithm are given in Section 4. The performance of the preposed algorithm is compared with VF2 algorithm in Section 5, and Section 6 presents the conclusion of this paper.

2. Preliminary definitions

A graph is a set of vertices (V) and edges (E) represented by $G = (V, E)$. A simple graph is one with no loops and no parallel edges. An undirected graph is one whose edges have no direction. A connected graph is a graph with at least one path between any pair of vertices.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two given graphs, G_1 and G_2 are isomorphic if there exists a bijective function $\sigma: V_1 \to V_2$ such that for every $(u,v) \in E_1: \big(\sigma(u), \sigma(v)\big) \in E_2$.

Simply, recognizing isomorphism between two graphs indicates whether the two graphs have the same topological structure and are represented by the expression $G_1 \cong G_2$.

The proposed algorithm, which employs the canonical labeling method, first receives graph G_1 and executes the function $C(G_1)$ on the graph. Then for graph G_2 it also calculates the value of $C(G_2)$. Subsequently, the two graphs are considered isomorphic if $C(G_1) = C(G_2)$.

Let a simple connected undirected graph $G = (V, E)$ be given, for some vertices $u, v \in V$: $P(u,v)$ is the set of all paths between u and v. For some $p \in P(u,v)$, the length of p is defined as the number of edges in path p, and is denoted as $|p|$. The distance between u and v is defined as follows:

$$dist(u,v) = \min_{\forall p \in P(u,v)} \{|p|\}$$

For every edge $(u,v) \in E$, we introduce two functions: $f_v(u)$ and $f_e(\overrightarrow{u,v})$. $f_v(u)$ denotes the number of shortest paths to which vertex u belongs. $f_e(\overrightarrow{u,v})$ is the number of shortest paths to which edge (u,v) belongs.

Function $f_v(u)$ in which for each vertex of the graph equals the number of shortest paths between other vertices of the graph passing through vertex u is introduced. Furthermore a function $f_e(\overrightarrow{u,v})$ where all the edges of an undirected graph are assumed to be bi-directional edges is introduced.

Every edge is assigned an f_e value. The value of $f_e(\overrightarrow{u,v})$ for edge u to v is the number of shortest paths between every pair of vertices passing through the directed edge.

$f_v(u)$ is computed according to equation (1):

$$f_v(u) = \left| \left\{ (v,w) \middle| \begin{matrix} dist(v,w) \\ = dist(v,u) + dist(u,w)\,, \\ \forall\, u,v,w \in V\ that \\ u \neq v\ and\ u \neq w\ and\ v \neq w \end{matrix} \right\} \right| \quad (1)$$

The range of $f_v(u)$ values is integer numbers and intervals $0 \le f_v \le \sum_{i=1}^{|V|-2} i$ (The maximum value is obtained in the star graphs for the center vertex).

$f_e(\overrightarrow{u,v})$ is calculated according to equation (2):

$$f_e(\overrightarrow{u,v}) = \left| \left\{ (u,w) \middle| \begin{matrix} dist(u,w) = dist(v,w) + 1 \\ ,\forall\, u,v,w \in V\ that\ u \neq w \end{matrix} \right\} \right| \quad (2)$$

The range of $f_e(\overrightarrow{u,v})$ values is integer numbers in intervals $1 \le f_e \le \left\lceil \frac{|V|-2}{2} \right\rceil \left\lfloor \frac{|V|-2}{2} \right\rfloor$ (the maximum value is obtained for the cut edge or bridge)..

3. The Algorithm

In this section, we present an efficient algorithm that can generate effective canonical labeling based on the structure of graphs that generate different codes for non-isomorphic graphs. Despite the maximum positive results of the code generation algorithm, it is still not possible to make a deterministic decision regarding the GIP.

The pseudocode for the proposed algorithm is presented in Alg. 1 in the form of Algorithm ***Graph**Isomorphism**Check*( )**. As can be seen in the code, the number of vertices and edges for the two input graphs are first compared. If they are equal, the canonical labels for both graphs are generated by the ***Gen**Code()* function, and then their isomorphism or non-isomorphism is determined.

function *G*en*C*ode(G)
input: $G = (V, E)$: **graph** // a finite simple connected graph where
// for every vertex v computes the f_v
// $f_v(v)$: integer,
// for every edge (u, v) assumes two f_e
// $f_e(\overrightarrow{u,v})$: integer, $f_e(\overrightarrow{v,u})$: integer,
// for every vertex v: list of all f_e which start from v,
// gives the label (f_v, f_e) to each vertex
output: *G*raph*C*ode : **list** // concatenates the labels as a canonical code of *G*

---

begin
1:**for all** pairs of vertices $u, v \in V$ in where $u \neq v$ **do** // using *BFS* on all vertices
2: compute $dist(u, v)$;
3: **for all** vertices $v \in V$ **do**
4: $f_v(v) \leftarrow 0$
5: **for all** pairs of vertices $u, w \in V$ where ($v \neq u$ and $v \neq w$ and $u \neq w$) **do**
6: **if** $dist(u, w) = dist(u, v) + dist(v, w)$ **then**
7: $f_v(v) \leftarrow f_v(v) + 1$
8: **for all** edges $(u, v) \in E$ **do**
9: $f_e(\overrightarrow{u,v}) \leftarrow 0$
10: **for all** vertices $w \in V$ where $w \neq u$ **do**
11: **if** $dist(u, w) = dist(v, w) + 1$ **then**
12: $f_e(\overrightarrow{u,v}) \leftarrow f_e(\overrightarrow{u,v}) + 1$
13: $f_e(\overrightarrow{v,u}) \leftarrow 0$
14: **for all** vertices $w \in V$ where $w \neq v$ **do**
15: **if** $dist(v, w) = dist(u, w) + 1$ **then**
16: $f_e(\overrightarrow{v,u}) \leftarrow f_e(\overrightarrow{v,u}) + 1$
17: **for all** vertices $v \in V$ **do**
18: $L[v] \leftarrow \emptyset$; //$L[v]$ is empty list that stores f_v for every vertex v
19: **for all** vertices $u \in v.neighbors$ **do** //$v.neighbors$ is All the vertices that are adjacent to v
20: add $f_e(\overrightarrow{v,u})$ to $L[v]$;
21: sort($L[v]$); // sort list $L[u]$ as lexicographically
22: $L[v] \leftarrow$ concat($f_v(v), L[v]$)
23: sort(L); // sort all elements of list L as lexicographically
24: $G\text{raph}C\text{ode} \leftarrow$ concatenation of all elements of list L;
25: **return** $G\text{raph}C\text{ode}$;
end of function;

Algorithm *G*raph*I*somorphism*C*heck (G, G')
input: $G = (V, E), G' = (V', E')$ // two finite simple connected graphs
output: *true* // if G is isomorphic to G'
false // else

---

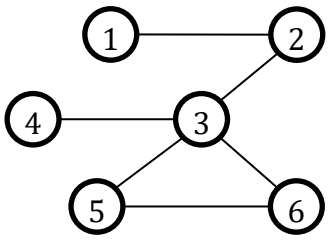
begin
1.**if** $|V| \neq |V'|$ **or** $|E| \neq |E'|$ **then**
2. **return** *false*;
3.**if** *G*en*C*ode (G)≠ *G*en*C*ode (G') **then**
4. **return** *false*;
5.**return** *true*;
end of Algorithm.

Alg. 1. Pseudo-code of the proposed algorithm.

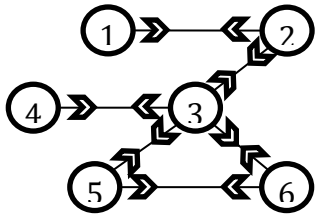
The algorithm generates the canonical labeling using f_v and f_e which is described in the previous section and proceeds as follows:

- The algorithm calculates the value of f_v associated with each vertex.
- Each edge of the graph is considered to be two edges in two opposite directions, and the value of f_e is calculated for each of them, that is, the algorithm allocates two values f_e to each edge.
- It then assigns a list of f_e values of the output edges to each vertex.
- f_e values within each list are sorted in non-descending order.
- The label of each vertex is formed by concatenating f_v with f_e.
- Finally, an array with the length of the graph order is composed of labels for all vertices, and the array is sorted in a non-descending order.

The output array is the canonical label of the input graph. Here, a sample graph and implementation of the proposed algorithm are presented. Considering the graph in Fig. 1, the algorithm considers the undirected graph (a) shown in Fig. 1 as the directed graph (á) shown in Fig. 1:



(a)



(á)

Fig. 1. Graph (a) is considered as graph (á) by the proposed algorithm.

The algorithm calculates the values of f_v for each vertex of the graph in Fig. 1, as follows:
$f_v(1)=0$, $f_v(2)=4$, $f_v(3)=8$, $f_v(4)=0$, $f_v(5)=0$, $f_v(6)=0$.
The value of f_e corresponding to the two directions of each edge is calculated as follows:

$edge(1,2)$:

$f_e(\overrightarrow{1,2}) = |\{(1,2),(1,3),(1,4),(1,5),(1,6)\}| = 5$
$f_e(\overrightarrow{2,1}) = |\{(2,1)\}| = 1$

$edge(2,3)$:

$f_e(\overrightarrow{2,3}) = |\{(2,3),(2,4),(2,5),(2,6)\}| = 4$
$f_e(\overrightarrow{3,2}) = |\{(3,2),(3,1)\}| = 2$

$edge(3,4)$:

$f_e(\overrightarrow{3,4}) = |\{(3,4)\}| = 1$
$f_e(\overrightarrow{4,3}) = |\{(4,1),(4,2),(4,3),(4,5),(4,6)\}| = 5$

$edge(3,5)$:

$f_e(\overrightarrow{3,5}) = |\{(3,5)\}| = 1$
$f_e(\overrightarrow{5,3}) = |\{(5,1),(5,2),(5,3),(5,4)\}| = 4$

$edge(3,6)$:

$f_e(\overrightarrow{3,6}) = |\{(3,6)\}| = 1$
$f_e(\overrightarrow{6,3}) = |\{(6,1),(6,2),(6,3),(6,4)\}| = 4$

$edge(5,6)$:

$f_e(\overrightarrow{5,6}) = |\{(5,6)\}| = 1$
$f_e(\overrightarrow{6,5}) = |\{(6,5)\}| = 1$

This algorithm uses f_v and f_e values to assign a list of them to each vertex:
$V(1) \leftarrow (0,[5])$,
$V(2) \leftarrow (4,[1,4])$,
$V(3) \leftarrow (8,[1,1,1,2])$,
$V(4) \leftarrow (0,[5])$,
$V(5) \leftarrow (0,[1,4])$,
$V(6) \leftarrow (0,[1,4])$,
and then sorts the list values in non-descending order.
Finally, this algorithm forms an array from sorted lists and performs lexical sorting on array cells:
$[(0,[1,4]),(0,[1,4]),(0,[5]),(0,[5]),(4,[1,4]),(8,[1,1,1,2])]$
The length of this array is $|V|$, that's actually the canonical labeling of the graph in Fig. 1.

4. Experiments

The set of input graphs for the algorithm is as follows:

1. All simple connected graphs with 3 to 9 vertex number [25]
2. 10,000 samples of simple connected graph with 10 to 20 vertex number [26]
3. 10 samples of relatively large random connected graph with 100 to 600 vertex number [27]
4. All regular graphs with 8 to 32 vertex number [28]

All the non-isomorphic graphs of simple connected graphs with small order known to date are included in Set 1. Because the total number of known graphs with 10 vertices is more than 11 million, and as the degree of the graph increases, this value becomes much larger, so Set 2 includes a part of these graphs. The production functions of random graphs are applied to simple connected graphs with at least 100 vertices in Set 3. In this set, the number of vertices and edges are optionally selected, and the graph is created using random graph generating functions. The last dataset consists of all regular graphs with maximum order of 32.
In each dataset, the graphs are categorized according to their order. The implementation of this algorithm is evaluated using a batch of them. At runtime, if the unit is detected as an isomorphism between any graph pairs that are non-isomorphic, one unit is added to the algorithm error rate.

From the results presented in Tables I and Table II, it can be concluded that the size and order of the input graphs have no effect on the correct detection of the proposed algorithm.

Table I. Algorithm performance on simple graphs of set 1 and 2.

Graph Order	The number of graph pairs	Algorithm output correctness percentage
3	1	100%
4	15	100%
5	210	100%
6	6,216	100%
7	363,378	100%
8	61,788,286	100%
9	34,081,252,660	99.999%
10	49,995,000	99.9999%
11	49,995,000	99.9999%
12	49,995,000	99.9999%
13	49,995,000	100%
14	49,995,000	100%
15	49,995,000	100%
16	49,995,000	100%
17	49,995,000	100%
18	49,995,000	100%
19	49,995,000	100%
20	49,995,000	100%

Table II. Algorithm performance on relatively large simple graphs.

Graph Order ($\|\boldsymbol{V}\|$)	Graph Size ($\|\boldsymbol{E}\|$)	The number of graph pairs	Algorithm output correctness percentage
100	1000	45	100%
200	2000	45	100%
300	3000	45	100%
400	4000	45	100%
500	5000	45	100%
600	6000	45	100%

It is important to note that isomorphism detection algorithms that are based on canonical labeling have poor performance on regular graphs. Therefore, in this paper, the proposed algorithm is tested by using regular graphs as input. The results of the proposed algorithm on these types of graphs are shown in Table III.
According to the table III, the lower bound for regular graphs with $8 \leq |V| \leq 32$ is 99.42%.

Table III. Algorithm performance on regular graphs**.**

Regular Graph Type	The number of graph pairs	Algorithm output correctness percentage
8-3-3	10	100%
8-4-3	15	100%
8-5-3	3	100%
9-4-3	120	100%
9-6-3	6	100%
10-3-3	171	99.42%
10-4-3	1,711	99.77%
10-5-3	1,770	100%
10-6-3	210	99.52%
11-4-3	34,980	99.89%
11-6-3	35,245	99.97%
12-3-3	3,570	99.97%
12-4-3	1,191,196	99.97%
12-5-3	30,791,628	99.98%
13-4-3	58,077,253	99.997%
14-3-3	129,286	99.998%
14-4-3	53,359,615	99.9999%

16-3-3	8,239,770	99.9999%
18-3-3	79,694,785	100%
20-3-4	16,632,028	99.99999%
20-3-5	16,718,653	99.9998%
23-4-5	7,669,486	99.9998%
24-3-6	28,678,951	99.9999%
32-3-7	461,092,528	100%

4.1. Conflicts in the algorithm

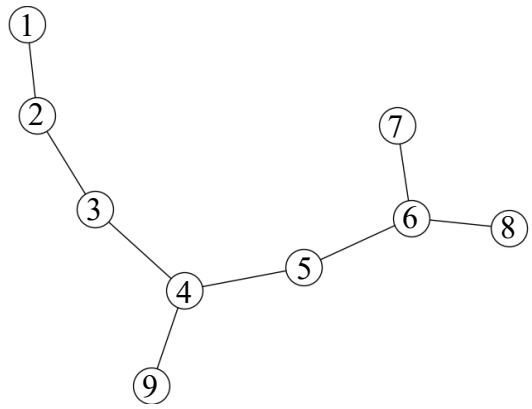
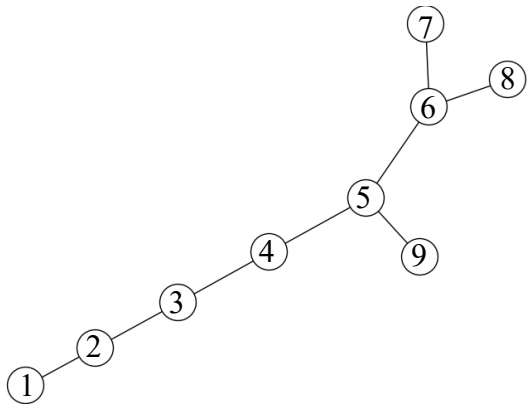
The two graphs G_1 and G_2 shown in Table IV are simple connected 9-vertex non-isomorphic graphs, and because the canonical labels obtained for both graphs in the algorithm experiment were identical, they were identified as one of the algorithm errors.

According to the drawing shown in Table IV of the graphs G_1 and G_2, vertices $V(1), V(2), V(3), V(6), V(7)$ and $V(8)$ in both graphs have identical status in their topology, and the generated codes are identical.

The generated codes for vertices V(4) and V(5) are shifted; it means that $V(4)$ in G_1 is equivalent to $V(5)$ in G_2 and $V(5)$ in G_1 is equivalent to $V(4)$ in G_2 because in both graphs, vertices are connected at equal distances from each other. Since the algorithm is designed based on the shortest path between vertices in the graph, in both graphs the values of f_v and f_e are the same and eventually the same canonical label is obtained, although it is not correct.

To check it, by removing $V(4)$ with its connections in G_1, we have a path graph $P_3(V(1), V(2), V(3))$ and a star graph $S_4(V(5), V(6), V(7), V(8))$ and a single vertex $V(9)$ while by removing $V(5)$ with its connections in G_2, it decomposes to two path graphs $P_4(V(1), V(2), V(3), V(4))$ and P_3 $(V(6), V(7), V(8))$ and a single vertex $V(9)$.

Table IV. Two graphs that are detected isomorphic in proposed algorithm.

G_2	G_1
	
$V(1) \leftarrow [0, [8]]$ $V(2) \leftarrow [7, (1,7)]$ $V(3) \leftarrow [12, [2, 6]]$ $V(4) \leftarrow [19, [1, 3, 4]]]$ $V(5) \leftarrow [15, [3, 5]]$ $V(6) \leftarrow [13, [1, 1, 6]]$ $V(7) \leftarrow [0, [8]]$ $V(8) \leftarrow [0, [8]]$ $V(9) \leftarrow [0, [8]]$	$V(1) \leftarrow [0, [8]]$ $V(2) \leftarrow [7, (1,7)]$ $V(3) \leftarrow [12, [2, 6]]$ $V(4) \leftarrow [15, [3, 5]]$ $V(5) \leftarrow [19, [1, 3, 4]]]$ $V(6) \leftarrow [13, [1, 1, 6]]$ $V(7) \leftarrow [0, [8]]$ $V(8) \leftarrow [0, [8]]$ $V(9) \leftarrow [0, [8]]$
Canonical Labeling for two graphs: $[[0, [8]], [0, [8]], [0, [8]], [0, [8]], [7, [1, 7]], [12, [2, 6]], [13, [1, 1, 6]], [15, [3, 5]], [19, [1, 3, 4]]]$	

4.2. Complexity analysis of the proposed algorithm

To calculate the complexity of the proposed algorithm, we assumed a graph with n vertices and e edges. Distance computing for all pairs of vertices requires $O(n^3)$. Lines 3-7 of the algorithm require $O(n.e)$ and lines 8-11 of the algorithm require $O(n.e)$. Consequently, the overall time complexity of the algorithm is $O(n^3)$ for dense and sparse graphs.

The space complexity of the algorithm is equal to the number of f_es stored for each vertex of the graph. This number is the degree of the vertex, and in a graph equals the sum of the degrees of the all vertices in that graph. In

the other words, the algorithm's storage space equals $\sum_{v_i \in V} deg(v_i) = 2e$ or $O(e)$.

5. Performance Comparison

In this section, we compare the performance of the proposed algorithm with the latest release of a well-known GI algorithm called VF2 [29] which is an improved matching algorithm that can be used for both graph and subgraph isomorphism.

The input graphs for this experimental comparison are the benchmark graph families in [22], containing 25 random graphs with Edge Probability $1/sqrt(n)$, where n is the number of nodes. The order of these graphs ranges from 100 to 500; in each order, there are five different graphs of the same size (Table V).

The accuracy of VF2 and the proposed algorithm in the case of the following test graphs is 100%.

The experiments were carried out on a laptop ASUS, CPU: AMD RADEON R4, 5 COMPUTE CORES 2C+3G, 2.50 GHz, Memory: 8GB, OS: 64-bit Windows 10 Enterprise, which is implemented in the Python environment.

This comparison was performed in terms of run time and space; the comparison of two algorithms in terms of runtime is shown in Fig. 5 and their comparison in terms storage space is shown in Fig. 6.

Table V. Benchmark graphs with their properties.

Properties / Benchmark	order of graphs	size of graphs
1-5	100	500
6-10	200	1414
11-15	300	2598
16-20	400	4000
21-25	500	5590

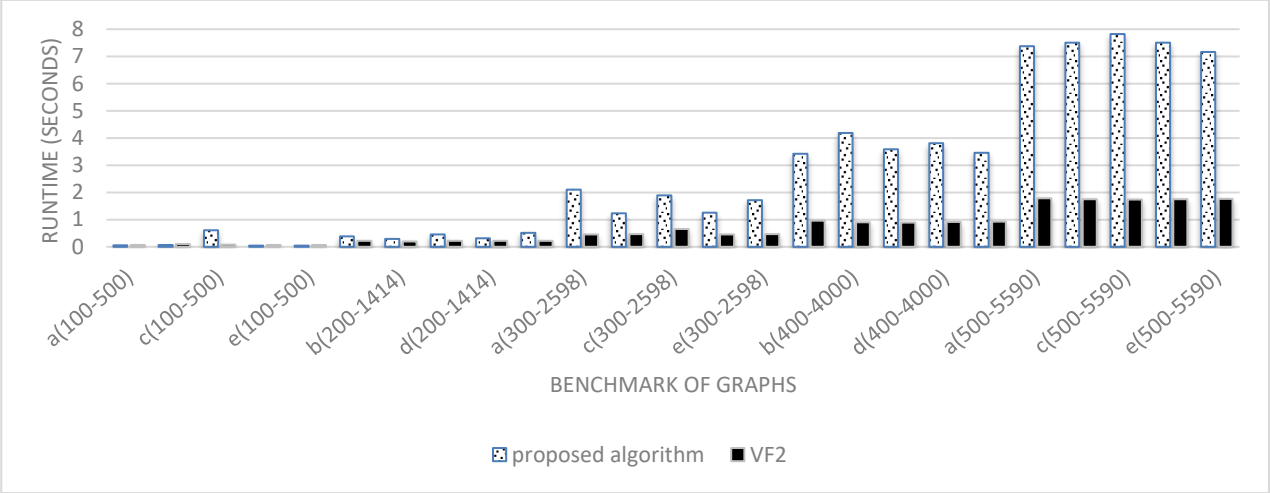


Fig. 5. Runtime of the proposed algorithm vs. VF2 (in Seconds) on selected benchmark of graphs.

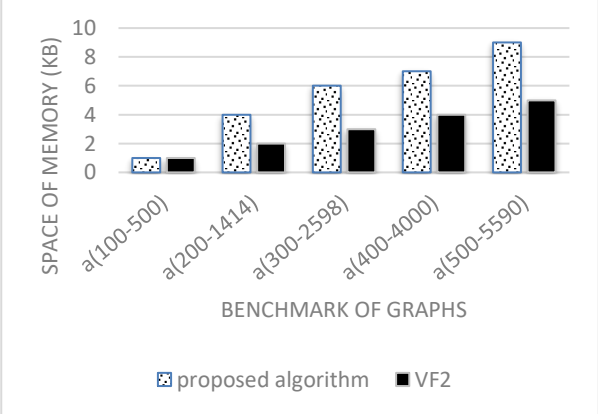


Fig. 6. Storage space of the proposed algorithm vs. VF2 (KB) on selected benchmark of graphs.

The difference in time and memory is due to the difference in the methods of the two algorithms; thus, the proposed algorithm converts each graph separately into a canonical label and then compares the two labels, but the VF2 algorithm stores both graphs simultaneously in the system's main memory. It loads and performs peer-to-peer traversal and displays the results as the corresponding vertices. The advantage of isomorphic detection algorithms with canonical labeling is that they create an identifier for each graph, which can be used to display the basic structure of the graph by reverse engineering on that label, while other algorithms do not have this capability.

Because the VF2 algorithm has been changing and optimizing since its introduction (2001), it can be said that its good runtime results are not far from expected. According to the above results, the proposed algorithm also has good efficiency in the set of relatively high order graphs.

6. Conclusion

According to the results obtained from the accuracy of isomorphism detection in the experimental results, the proposed algorithm is not input sensitive, and can be used for large graphs.

Regarding how the algorithm is tested, it can be said that the results are obtained from a complete and accurate review because, in each case, all the graphs registered so far are checked, and the input graphs at each stage have the initial condition of isomorphism (the number of vertices and edges are equal). In other words, all results are obtained by applying strict conditions to the algorithm (The input non-isomorphic graphs have the same isomorphism properties, and the only non-isomorphic response is expected due to the generated code). This is while many proposed algorithms use random input datasets, which means that they lack deterministic complexity.

Although the results obtained from the algorithm for general graphs are not completely deterministic, but it can be said that the proposed heuristic algorithm has a good ability to cover the properties of a general graph.

7. References

[1] A. Rafiee, P. Moradi, A. Ghaderzadeh, "A swarm intelligence based multi-label feature selection method hybridized with a local search strategy", *Tabriz Journal of Electrical Engineering*, vol. 51, no. 4, pp. 443-454, 2021.

[2] S. Mokhtarizadeh, B. Zamani Dehkordi, M. Mosleh, Ali Barati, "Influence Maximization using Time Delay based Harmonic Centrality in Social Networks", *Tabriz Journal of Electrical Engineering*, vol. 51, no. 3, pp. 359-370, 2021.

[3] E.K. Wong, "Model matching in robot vision by subgraph isomorphism", *Pattern Recognition*, vol. 25, no. 3, pp. 287-304, 1992.

[4] A. Kandel, H. Bunke, M. Last, "Applied Graph Theory in Computer Vision and Pattern Recognition", Berlin, Springer, 2007.

[5] M.A. Abdulrahim, M. Misra, "A graph isomorphism algorithm for object recognition", *Pattern Analysis and Applications*, vol. 1, no. 3, pp. 189-201, 1998.

[6] T. Washio, H. Motoda, "State of the art of graph-based data mining", *ACM SIGKDD Explorations Newsletter*, vol. 5, no. 1, pp. 59-68, 2003.

[7] D. Conte, P. Foggia, C. Sansone, M. Vento, "Graph matching applications in pattern recognition and image processing", In IEEE International Conference on Image Processing, September 2003, Barcelona, Spain, pp. 21-24.

[8] S. Wasserman, K. Faust, "Social Network Analysis: Methods and Applications", Cambridge University Press, 1994.

[9] A. Sanfeliua, R. Alquézarb, J. Andradea, J. Climentc, F. Serratosad, J. Vergésa, "Graph-based representations and techniques for image processing and image analysis", *Pattern Recognition*, vol. 35, no. 3, pp. 639-650, 2002.

[10] D.H. Rouvray, A.T. Balaban, "Chemical applications of graph theory", Academic Press London, 1979.

[11] T. Aittokallio, B. Schwikowski, "Graph–based methods for analysing networks in cell biology", *Briefings in Bioinformatics*, vol. 7, no. 3, pp. 243-255, 2006.

[12] G. Valiente, "Algorithms on Trees and Graphs", Springer-Verlag, 2002.

[13] L. Babai, E.M. Luks, "Canonical labeling of graphs", In 15th Annual ACM Symposium on Theory of Computing, December 1983, New York, United States, pp. 171–183.

[14] L. Babai, P. Erdos, M. Selkow, "Random graph isomorphism", *SIAM Journal on Computing*, vol. 9, no. 3, pp. 628-635, 1980.

[15] L. Babai, L. Kucera, "Canonical labeling of graphs in linear average time", In 20th IEEE Symposium on Foundations of Computer Science, October 1979, San Juan, USA, pp. 39-46.

[16] L. Babai, "Graph isomorphism in quasi-polynomial time", In 48th Annual ACM Symposium on Theory of Computing, June 2016, New York, United States, pp. 684-697.

[17] B.D. McKay, A. Piperno, "Practical graph isomorphism II", *Journal of Symbolic Computation*, vol. 60, pp. 94-112, 2014.

[18] T. Miyazaki, "The complexity of McKay's canonical labeling algorithm", *Groups and Computation*, vol. 28, no. 2, pp. 239-256, 1996.

[19] R.M. Karp, "Reducibility Among Combinatorial Problems", In Symposium on the Complexity of Computer Computations, March 1972, Yorktown Heights, New York, pp. 85-103.

[20] A.V. Aho, J.E. Hopcroft, J.D. Ullman, "The Design and Analysis of Computer Algorithms", Addison-Wesley, 1974.

[21] J.E. Hopcroft, J.K. Wong, "Linear time algorithm for isomorphism of planar graphs", In 6th Annual ACM Symposium on Theory of Computing, April 1974, Washington, USA, pp. 172-184.

[22] B.D. McKay, A. Piperno, "Nauty Traces", Available: http://pallini.di.uniroma1.it.

[23] A. Nourollah, S. Check, "A Heuristic Algorithm for Graph Isomorphism Problem Based on Eccentricity", *Computing Sciences and Information Technologies*, vol. 17, no. 2, pp. 45-51, 2019 (in Persian).

[24] L.C. Freeman, "A set of measures of centrality based on betweenness", *Sociometry*, vol. 40, no. 1, pp. 35-41, 1977.

[25] ANU College of Engineering & Computer Science. "collections of non-isomorphic graphs," Available: http://users.cecs.anu.edu.au/~bdm/data/graphs.html.

[26] Database of interesting graphs. "The House of Graphs," Available: https://hog.grinvin.org.

[27] Generate graphs. "Generate all unlabeled simple graphs on a given number of vertices," Available: http://combos.org/nauty.

[28] Regular Graphs Page. "Connected regular graphs," Available: http://www.mathe2.uni-bayreuth.de/markus/reggraphs.html.

[29] L. P. Cordella, P. Foggia, C. Sansone, M. Vento, "An improved algorithm for matching large graphs", In Proceedings of the 3rd IAPR-TC-15 International Workshop on Graph-based Representations, May 2001, Ischia, Italy, pp. 149-159.