An Algorithm for Graph Isomorphism Based on Path-length Numbers

Lijun Tian[1, a]

[1]*Information Technology Department, Hunan University of Finance and Economics. No.139, Fenglin 2nd Road, Changsha, 410205, China*
[a]*tianlijun@hufe.edu.cn*

Keywords: graph isomorphism, algorithm, path-length number.

Abstract: This paper proposed an algorithm for graph isomorphism based on path-length number (AGIPN). AGIPN used Length-L path numbers as partition metrics which could divide not similar vertices into trivial cells easily. For those similar vertices, AGIPN took a dynamic mapping procedure. AGIPN was tested on graphs, such as random connected graphs, pseudo random k-regular graphs, regular rings, and regular 2D meshes. The results indicate that the time requirement of AGIPN do not increase exponentially with the graph size of all these types.

1. Introduction

The graph isomorphism (GI) is of interest in a variety of different pattern recognition contexts [1,2], and structure comparison and identification of complex networks[3].

The subgraph isomorphism is proved to be a NP-complete problem [4,5], while it is still an open question if also GI is a NP-complete problem[1,4]. Therefore, many algorithms of GI have been proposed to solve this problem, such as McKay's Nauty algorithm [6], Ullmann[7], VF[8], etc.

All existed algorithms for GI are efficiency for some types of graphs, but their time requirements increase exponentially with the size of the graphs in the worst case [2].

In this paper, we propose a new algorithm which partition the vertices by path numbers, and use a dynamic mapping method for those similar vertices.

2. Definitions

Here, we consider only undirected graphs without parallel edges and loops, i.e., undirected simple graph, as showed in Fig.1. Definitions are mainly referenced from graph theory [4] and practical graph isomorphism [6].

Definition 1: a graph G is an ordered pair (V (G),E(G)) consisting of a set V (G) of vertices and a set E(G), together with an incidence functionψ_G that associates with each edge of G an unordered pair of vertices of G.

Definition 2: two graphs G and H are isomorphism, if there is a bijection θ: V (G)→V (H) which preserves adjacency (that is, the vertices u and v are adjacent in G if and only if their imagesθ(u) andθ(v) are adjacent in H), written G≌H, else written G!≌H.

There is G≌H in Fig.1. In fact, G and H in Fig.1 all are a same famous graph – Peterson Graph, of course they are isomorphism.

It is not difficult to validate the map ($V_1V_2V_3V_4V_5V_6V_7V_8V_9V_{10}\rightarrow U_1U_2U_3U_7U_6U_5U_9U_{10}U_4U_8$) is isomorphism.

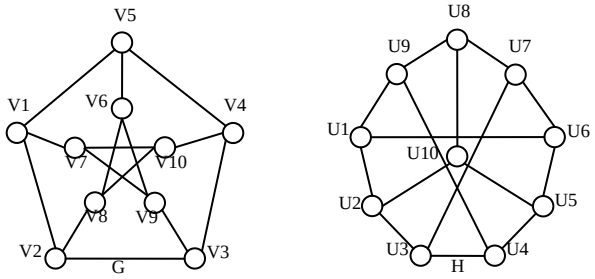

Fig.1 graphs of isomorphism

Definition 3: an automorphism of a graph is an isomorphism of the graph to itself.

For example, there is an automorphism of G in Fig.1 ($V_1V_2V_3V_4V_5V_6V_7V_8V_9V_{10}\rightarrow V_4V_3V_2V_8V_{10}V_7V_5V_9V_1V_6$).

Definition 4: a partition of graph G is a set of disjoint non-empty subsets of V(G).

Let$\pi=(\pi_1,\ldots\pi_r)$ is such a partition. The elements of a partition, $\pi_i(1\leq i\leq r)$, are usually called its cells. A trivial partition is a partition with only one cell, e.g., partition the Peterson Graph by vertex degree would get a trivial partition. A cell with only one vertex is called as trivial cell, and if all elements of a partition all are trivial cell, the partition is a discrete one.

Definition 5: vertices u and v of a graph G are similar, if there is an automorphismαwhich maps u to v, written u$\backsim$v.

All vertices of graph G and H in Fig.1 are similar.

3. AGIPN

3.1 Basic ideas

The goal of AGIPN is to find the isomorphism of two graphs G and H. If they are, give out a map, or they are not.

The basic process of AGIPN is as follows:

1) Label vertices of G and H in a random way, then we get two unsorted sequence {1,…n}, where n is the number of the vertices.
2) Partition the two sequences in a same method which is independent of labeling, such as sequence of path-length number, and sorted the partition.
3) If vertices in any cell of the partition are not similar, refine the partition.
4) Check the partition by the sorted order, if elements in a cell are similar, choose any one of them as the first element and refine all other unsorted cells. Continues according to this way until all cells are trivial.
5) Then we get two sorted sequence $<i_1,\ldots,i_n>$ and $<j_1,\ldots,j_n>$, and compare the two graphs by these two sequences. They are isomorphism if they map each other in this way, or else not.

3.2 Partition

As an example, we use graph G in Fig.1 to illustrate the process of AGIPN. AGIPN use adjacency matrixes of graphs to compute the path numbers, which are metrics of partition.

The adjacency matrixes of G and H are showed in (1) (Label as in Fig.1).

$$\begin{array}{cccccccccc}0&1&0&0&1&0&1&0&0&0\\1&0&1&0&0&0&0&1&0&0\\0&1&0&1&0&0&0&0&1&0\\0&0&1&0&1&0&0&0&0&1\\1&0&0&1&0&1&0&0&0&0\\0&0&0&0&1&0&0&1&1&0\\1&0&0&0&0&0&0&0&1&1\\0&1&0&0&0&1&0&0&0&1\\0&0&1&0&0&1&1&0&0&0\\0&0&0&1&0&0&1&1&0&0\end{array}\qquad\begin{array}{cccccccccc}0&1&0&0&0&1&0&0&1&0\\1&0&1&0&0&0&0&0&0&1\\0&1&0&1&0&0&1&0&0&0\\0&0&1&0&1&0&0&0&1&0\\0&0&0&1&0&1&0&0&0&1\\1&0&0&0&1&0&1&0&0&0\\0&0&1&0&0&1&0&1&0&0\\0&0&0&0&0&0&0&1&0&1\\1&0&0&1&0&0&0&0&1&0\\0&1&0&0&1&0&0&1&0&0\end{array}\qquad(1)$$

A(G) B(H)

Then, to get numbers of Length-2 path, the matrixes are multiplied by themselves. The products of $A(G)^{(2)}$ and $B(H)^{(2)}$ are showed in (2).

$$\begin{array}{cccccccccc}3&0&1&1&0&1&0&1&1&1\\0&3&0&1&1&1&1&0&1&1\\1&0&3&0&1&1&1&1&0&1\\1&1&0&3&0&1&1&1&1&0\\0&1&1&0&3&0&1&1&1&1\\1&1&1&1&0&3&1&0&0&1\\0&1&1&1&1&1&3&1&0&0\\1&0&1&1&1&0&1&3&1&0\\1&1&0&1&1&0&0&1&3&1\\1&1&1&0&1&1&0&0&1&3\end{array}\qquad\begin{array}{cccccccccc}3&0&1&1&1&0&1&1&0&1\\0&3&0&1&1&1&1&1&1&0\\1&0&3&0&1&1&0&1&1&1\\1&1&0&3&0&1&1&1&0&1\\1&1&1&0&3&0&1&1&1&0\\0&1&1&1&0&3&0&1&1&1\\1&1&0&1&1&0&3&0&1&1\\1&1&1&1&1&1&0&3&0&0\\0&1&1&0&1&1&1&0&3&1\\1&0&1&1&0&1&1&0&1&3\end{array}\qquad(2)$$

$A(G)^{(2)}$ $B(H)^{(2)}$

Clearly, it is not enough to partition vertices by a length-2 path-number. To reduce the number of path while length is longer, every time we take the diagonal elements of the length-L matrix out to another matrix as showed in (8). The remaining of the matrixes is showed in (3), which named as C(G) and D(H).

$$\begin{array}{cccccccccc}0&0&1&1&0&1&0&1&1&1\\0&0&0&1&1&1&1&0&1&1\\1&0&0&0&1&1&1&1&0&1\\1&1&0&0&0&1&1&1&1&0\\0&1&1&0&0&0&1&1&1&1\\1&1&1&1&0&0&1&0&0&1\\0&1&1&1&1&1&0&1&0&0\\1&0&1&1&1&0&1&0&1&0\\1&1&0&1&1&0&0&1&0&1\\1&1&1&0&1&1&0&0&1&0\end{array}\qquad\begin{array}{cccccccccc}0&0&1&1&1&0&1&1&0&1\\0&0&0&1&1&1&1&1&1&0\\1&0&0&0&1&1&0&1&1&1\\1&1&0&0&0&1&1&1&0&1\\1&1&1&0&0&0&1&1&1&0\\0&1&1&1&0&0&0&1&1&1\\1&1&0&1&1&0&0&0&1&1\\1&1&1&1&1&1&1&0&0&0\\0&1&1&0&1&1&1&0&0&1\\1&0&1&1&0&1&1&0&1&0\end{array}\qquad(3)$$

$C(G)^{(2)}$ $D(H)^{(2)}$

The next step is that matrixes C and D multiply A and B respectively, and the diagonal elements of the products are taken out to matrix in (8), the remaining are showed in (4).

$$\begin{array}{cccccccccc}0&2&2&2&2&2&2&2&2&2\\2&0&2&2&2&2&2&2&2&2\\2&2&0&2&2&2&2&2&2&2\\2&2&2&0&2&2&2&2&2&2\\2&2&2&2&0&2&2&2&2&2\\2&2&2&2&2&0&2&2&2&2\\2&2&2&2&2&2&0&2&2&2\\2&2&2&2&2&2&2&0&2&2\\2&2&2&2&2&2&2&2&0&2\\2&2&2&2&2&2&2&2&2&0\end{array}\qquad\begin{array}{cccccccccc}0&2&2&2&2&2&2&2&2&2\\2&0&2&2&2&2&2&2&2&2\\2&2&0&2&2&2&2&2&2&2\\2&2&2&0&2&2&2&2&2&2\\2&2&2&2&0&2&2&2&2&2\\2&2&2&2&2&0&2&2&2&2\\2&2&2&2&2&2&0&2&2&2\\2&2&2&2&2&2&2&0&2&2\\2&2&2&2&2&2&2&2&0&2\\2&2&2&2&2&2&2&2&2&0\end{array}\qquad(4)$$

$C(G)^{(3)}$ $D(H)^{(3)}$

Continuing this procedure until the length is 9, the matrixes are showed in (5) and (6).

$$\begin{array}{cccccccccc}0&656&896&896&656&896&656&896&896&896\\656&0&656&896&896&896&896&656&896&896\\896&656&0&656&896&896&896&896&656&896\\896&896&656&0&656&896&896&896&896&656\\656&896&896&656&0&656&896&896&896&896\\896&896&896&896&656&0&896&656&656&896\\656&896&896&896&896&896&0&896&656&656\\896&656&896&896&896&656&896&0&896&656\\896&896&656&896&896&656&656&896&0&896\\896&896&896&656&896&896&656&656&896&0\end{array}\qquad(5)$$

$C(G)^{(9)}$

$$\begin{array}{cccccccccc}0&656&896&896&896&656&896&896&656&896\\656&0&656&896&896&896&896&896&896&656\\896&656&0&656&896&896&656&896&896&896\\896&896&656&0&656&896&896&896&656&896\\896&896&896&656&0&656&896&896&896&656\\656&896&896&896&656&0&656&896&896&896\\896&896&656&896&896&656&0&656&896&896\\896&896&896&896&896&896&656&0&656&656\\656&896&896&656&896&896&896&656&0&896\\896&656&896&896&656&896&896&656&896&0\end{array}\qquad(6)$$

$D(G)^{(9)}$

Clearly, the origin matrix of graph is dependent on the labeling method, so the products would dependent on it also. We sort the product to get a matrix which is independent of labeling order. Firstly, every column is sorted respectively, then all columns are sorted. The sorted length-9 path-number matrixes of G and H are showed in (7). There is only one matrix in (7), because they are the same one.

896	896	896	896	896	896	896	896	896	896
896	896	896	896	896	896	896	896	896	896
896	896	896	896	896	896	896	896	896	896
896	896	896	896	896	896	896	896	896	896
896	896	896	896	896	896	896	896	896	896
896	896	896	896	896	896	896	896	896	896
656	656	656	656	656	656	656	656	656	656
656	656	656	656	656	656	656	656	656	656
656	656	656	656	656	656	656	656	656	656
0	0	0	0	0	0	0	0	0	0

(7)

$SC(G)^{(9)}$ $(SD(G)^{(9)})$

Then we can partition the vertices by the matrix in (7), if the columns are not equal, the vertices are divided into different cells, or in the same. But sometimes there are not similar vertices into a cell, so we use matrix of the diagonal elements, DE as in (8), to refine the partition.

	1	2	3	4	5	6	7	8	9	10
2	3	3	3	3	3	3	3	3	3	3
3	0	0	0	0	0	0	0	0	0	0
4	6	6	6	6	6	6	6	6	6	6
5	12	12	12	12	12	12	12	12	12	12
6	36	36	36	36	36	36	36	36	36	36
7	96	96	96	96	96	96	96	96	96	96
8	264	264	264	264	264	264	264	264	264	264
9	720	720	720	720	720	720	720	720	720	720

(8)

DE(diagonal elements)

Here, DE of G and H are a same one. A column denotes a vertex, and the i-row means it is from matrix of length-i.

Finally, after this process, we can get a partition for each graph, if the partition is discrete, we can compare the graphs for isomorphism testing, and else a dynamic mapping procedure is needed.

3.3 Dynamic mapping procedure

As an example, the mapping procedure of G in Fig.1 is illustrated. We got a partition of G by the method described in subsection B, which is showed in (9).

1	1	1	1	1	1	1	1	1	1
10	10	10	10	10	10	10	10	10	10
1	2	3	4	5	6	7	8	9	10

(9)

The first row of the matrix in (9) means the serial number of the cells, the second is number of vertices in each cell, and the last row is vertex label.

In (9), there is a trivial partition which has only one cell with all vertices. It means that all vertices in G are similar.

Then we take a dynamic mapping procedure to partition all similar vertices. Firstly, we choose any of vertices in a cell, then use the matrixes in (7) and (8) to refine all other vertices. The process continues in this way, until all cells are trivial.

Usually, we choose the first vertex in the cell to refine, e.g. vertex 1 is selected in (9). Then the first and the second row of the matrix are reset, as showed in (10).

1	2	2	2	2	2	2	2	2	2
1	9	9	9	9	9	9	9	9	9
1	2	3	4	5	6	7	8	9	10

(10)

To reduce the complexity, a matrix of shortest path is used. Then, according this matrix and those in (7) and (8), refine all other vertices (2~10), and get a matrix as in (11).

1	2	2	2	3	3	3	3	3	3
1	3	3	3	6	6	6	6	6	6
1	2	5	7	3	4	6	8	9	10

(11)

Every time the matrix gotten by the refine process must be sorted by the first row, i.e., the value of the cell.

There are 3 cells in (11), the first has only one vertex, the second has 3 and the last has 6. Then we selected the first vertex (2) in cell 2, and refine all other vertices (5,7,3,4,6,8,9,10). If there is any cell is not trivial, the procedure would be continuing. The remaining procedure of G is showed in (12)-(17).

1	2	3	3	4	4	4	4	4	4	1	2	3	3	4	4	4	4	4	4
1	1	2	2	6	6	6	6	6	6	1	1	2	2	6	6	6	6	6	6
1	2	5	7	3	4	6	8	9	10	1	2	3	8	5	7	4	6	9	10
1	2	3	4	5	5	6	6	6	6	1	2	3	4	5	5	6	6	6	6
1	1	1	1	2	2	4	4	4	4	1	1	1	1	2	2	4	4	4	4
1	2	3	8	4	9	5	7	6	10	1	2	3	8	4	9	5	7	6	10
1	2	3	4	5	6	7	7	8	8	1	2	3	4	5	6	7	7	8	8
1	1	1	1	1	1	2	2	2	2	1	1	1	1	1	1	2	2	2	2
1	2	3	8	6	10	9	5	4	7	1	2	3	8	6	10	9	5	4	7

(12)

1	2	3	4	5	5	5	5	5	5	1	2	3	4	5	5	6	6	6	6
1	1	1	1	6	6	6	6	6	6	1	1	1	1	2	2	4	4	4	4
1	2	3	8	5	7	4	6	9	10	1	2	3	8	4	9	5	7	6	10
1	2	3	4	5	5	6	6	6	6	1	2	3	4	5	6	7	7	7	7
1	1	1	1	2	2	4	4	4	4	1	1	1	1	1	1	4	4	4	4
1	2	3	8	6	10	4	9	5	7	1	2	3	8	6	10	4	9	5	7
1	2	3	4	5	6	7	7	8	8	1	2	3	4	5	6	7	8	9	9
1	1	1	1	1	1	2	2	2	2	1	1	1	1	1	1	1	1	2	2
1	2	3	8	6	10	4	7	9	5	1	2	3	8	6	10	4	7	9	5

(13)

1	2	3	4	5	5	5	5	5	5	1	2	3	4	5	5	6	6	6	6
1	1	1	1	6	6	6	6	6	6	1	1	1	1	2	2	4	4	4	4
1	2	3	8	5	7	4	6	9	10	1	2	3	8	4	9	5	7	6	10
1	2	3	4	5	5	6	6	6	6	1	2	3	4	5	6	7	7	7	7
1	1	1	1	2	2	4	4	4	4	1	1	1	1	1	1	4	4	4	4
1	2	3	8	6	10	4	9	5	7	1	2	3	8	6	10	4	9	5	7
1	2	3	4	5	6	7	7	8	8	1	2	3	4	5	6	7	8	9	9
1	1	1	1	1	1	2	2	2	2	1	1	1	1	1	1	1	1	2	2
1	2	3	8	6	10	4	7	9	5	1	2	3	8	6	10	4	7	9	5

(14)

1	2	3	3	4	4	4	4	4	4	1	2	3	3	4	4	4	4	4	4
1	1	2	2	6	6	6	6	6	6	1	1	2	2	6	6	6	6	6	6
1	2	5	7	3	4	6	8	9	10	1	2	3	8	5	7	4	6	9	10
1	2	3	4	5	5	6	6	6	6	1	2	3	4	5	5	6	6	6	6
1	1	1	1	2	2	4	4	4	4	1	1	1	1	2	2	4	4	4	4
1	2	3	8	4	9	5	7	6	10	1	2	3	8	4	9	5	7	6	10
1	2	3	4	5	6	7	7	8	8	1	2	3	4	5	6	7	7	8	8
1	1	1	1	1	1	2	2	2	2	1	1	1	1	1	1	2	2	2	2
1	2	3	8	6	10	9	5	4	7	1	2	3	8	6	10	9	5	4	7

(16)

1	2	3	4	5	5	5	5	5	5	1	2	3	4	5	5	6	6	6	6
1	1	1	1	6	6	6	6	6	6	1	1	1	1	2	2	4	4	4	4
1	2	3	8	5	7	4	6	9	10	1	2	3	8	4	9	5	7	6	10
1	2	3	4	5	5	6	6	6	6	1	2	3	4	5	6	7	7	7	7
1	1	1	1	2	2	4	4	4	4	1	1	1	1	1	1	4	4	4	4
1	2	3	8	6	10	4	9	5	7	1	2	3	8	6	10	4	9	5	7
1	2	3	4	5	6	7	7	8	8	1	2	3	4	5	6	7	8	9	9
1	1	1	1	1	1	2	2	2	2	1	1	1	1	1	1	1	1	2	2
1	2	3	8	6	10	4	7	9	5	1	2	3	8	6	10	4	7	9	5

(17)

Finally, we got a discrete partition as in (18), which has 10 trivial cells.

1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1
1	2	3	8	6	10	4	7	5	9

(18)

Then we got a sorted sequence of vertices for graph G: (1,2,3,8,6,10,4,7,5,9). In the same way we can sort the vertices of H and get a sorted sequence: (1,2,3,10,5,8,7,9,6,4).

Compare G and H by these two sequences, we find there is a bijection between them, so G≌H. The map ($V_1V_2V_3V_8V_6V_{10}V_4V_7V_5V_9 \rightarrow U_1U_2U_3U_{10}U_5U_8U_7U_9U_6U_4$) is isomorphism.

3.4 Algorithm

More formally, AGIPN is described as follows:

Algorithm1: AGIPN

INPUT: G,H
OUTPUT: the map if G≌H, or G!≌H
1: set isomorphism=True
2: label vertices get two sequences L_G,L_H, set $S_G=S_H=0$
3: S_G=Sort(G,L_G), S_H=Sort(H,L_H)
4: if(G(SG) !=H(SH))
isomorphism=False
5: return isomorphism, S_G, S_H

Algorithm2: Sort

INPUT: G, L_G
OUTPUT: S_G
1: set k=1,l=2, sorted=false
2: G1=G; MSP=SHORTESTPATH(G);
3: ***while*** *(1<l<n)* ***do***
*4: G1=G1*G; DE(l)=DIAG(G1);*
5: Set the diagonal elements of G1 to be zero;
6: l=l+1
7: ***end while***
8: S_G *= partition(G, G1);* S_G *=refine(*S_G, DE);
9: ***while*** *(sorted==false)* ***do***
10: let P_{Gi} be the smallest unsorted cell, set $v_1 \in P_{Gi}$, $S_G(k)=l(v_1)$, $P_{Gi}=\{P_{Gi}-v_1\}$, k=k+1, if k=|V(G)|, sorted=true, ***exit***
11: S_G *=refine2(*S_G, MSP,G1,DE);
12: ***end while***
13: Return S_G

4. Results and Discussion

4.1 Implementation and results

The graphs we used to test AGIPN include: random connected graphs, k-regular graphs, regular rings and regular 2D meshes.

Random connected graphs are denoted by average degree, i.e., a k-random graph means there are k*n/2 edges in it, where n is the number of vertices.

A regular ring is a graph of n vertices, each connected to its *2K* nearest neighbours.

We tested the correctness of AGIPN by a way that labeled the graph twice randomly, then got two difference adjacency matrixes, and tested them, each graph 100 times. The way shows that AGIPN is correct for all those graphs with no more than 6 degree and 1000 vertices.

Then, we tested the time requirement of AGIPN on such graphs (each type with the same vertices 100 graphs): 3-random, 3-regular, 6-random, 6-regular and 6-regular ring, 2D meshes. The first two graphs are tested up to 1000 vertices, showed in Fig.2, and the others only up to 100 vertices, showed in Fig.3.

4.2 Analysis and discussion

AGIPN sort graphs by the path number, which are computed by matrix multiplication. The complexity of matrix multiplication is $O(n^3)$, and other process all are no more than $O(n^3)$. There need no more than (n-1) times to compute path-number of maximum length (n-1), so the complexity is no more than $O(n^4)$.

But AGIPN is restricted by the path number, which increases rapidly with the degree and the length, though we used a method to reduce the value of path-number.

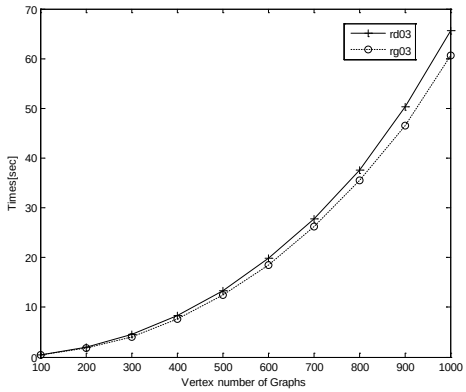

Fig.2 times of 3-random and 3-regular graphs

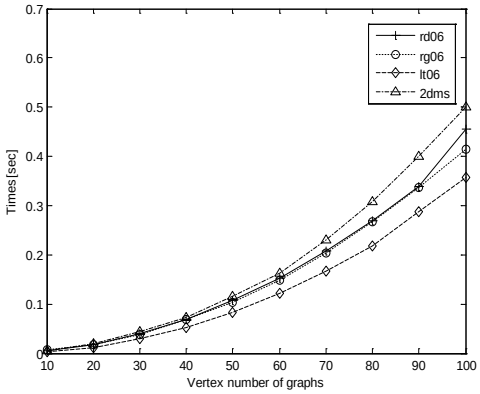

Fig.3 times of 6-random, 6-regular, 6-lattice ,2D meshes

5. Conclusion

In this paper, we proposed an algorithm for GI (AGIPN). Firstly AGIPN labeled graphs randomly, then partitioned each graph by path number and refined by a dynamic mapping procedure, then get two sorted sequences of label. Two graphs were compared by the sorted sequences, if they are equal, the graphs are isomorphism and the two sequences are isomorphism map, or else they are not isomorphism.

Then, we tested the algorithm on graphs such as: random connected graphs, k-regular graphs, regular rings and regular 2D meshes. The results show that AGIPN has polynomial time complexity on those types of graphs with no more than 6 degree and 1000 vertices.

But AGIPN is restricted by the large path numbers. Testing on other types of graphs and dealing with the large path numbers will be our works in the future.

References

[1] Kandel, Abraham, Bunke, eds. Applied Graph Theory in Computer Vision and Pattern Recognition, Springer, Berlin Heidelberg, New York, 2007. ISBN: 978-3-540-68019-2.

[2] Foggia P, Sansone C, Vento M. "A performance comparison of five algorithms for graph isomorphism". In: Jolion JM, Kropatsch W, Vento M, eds. Proc.of the 3rd IAPR-TC15 Int'l Workshop on Graph-Based Representation in Pattern Recognition. Ischia, 2001. pp. 188-199. http://amalfi.dis.unina.it/graph/db/ papers/benchmark.pdf.

[3] J. P. Bagrow1, E. M. Bollt2,1, J. D. Skufca2 and D. ben-Avraham1 . "Portraits of complex networks", EPL 81 68004, 2008. doi: 10.1209/0295-5075/81/68004.

[4] J.A. Bondy and U.S.R. Murty. Graph theory, Springer, 2008. ISBN: 978-1-84628-969-9, doi: 10.1007/978-1-84628-970-5.

[5] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP Completeness, Freeman & co., New York, 1990. ISBN:0-7167-1044-7.

[6] McKay BD. "Practical graph isomorphism". Congressus Numerantium, 1981, 30(1), pp.45-87.

[7] Ullmann JR. An algorithm for subgraph isomorphism.Journal of the Association for Computing Machinery, 1976,23(1), pp.31-42.

[8] Cordella LP,Foggia P,Sansone C,Vento M.An improved algorithm for matching large graphs.In:Jolion JM, Kropatsch W, Vento M,eds.Proc.of the 3rd IAPR-TC15 Int'l Workshop on Graph-Based Representation in Pattern Recognition. Ischia, 2001. pp.149-159. http://amalfi.dis.unina.it/graph/db/papers/ vfalgorithm.pdf