A Fast Algorithm For Kinematic Chain Isomorphism Identification Using Dividing And Matching Algorithm

Farhana Zaman Glory

Student ID : 7856215

Department of Computer Science

Course Title : Graph Theory COMP 7750

University of Manitoba

Winnipeg, MB R3T 2N2, Canada

gloryfz@myumanitoba.ca

Contents

1. Introduction

Isomorphism is one of the most mysterious properties of kinematic chains that has been fascinated many researchers. Identification of isomorphism is very much important in many graph related applications. Besides graph related applications isomorphism is also used in image processing, mechanisms, pattern matching, data mining, data security, bio-informatics and also in predicting protein structure. So detecting isomorphism at a low time cost is very much tempting now a days. In this paper a simple but effective method using matrix is used to identify isomorphism between two graphs.

2. Motivation

Detecting possible structural isomorphism between two given chains is one of the major problems encountered in intelligent computer aided design (CAD) for the design of kinematic chains [5]. Along with the development of computer industry and the trend of more sophisticated mechanism design, computer-aided mechanism design (CAMD) software is developed to be more integrated. The structural synthesis of kinematic chains involves enumerating and classifying all possible kinematic chains having a specified number of links and degrees of freedom [8]. Thus the redundant isomorphic kinematic chains should be eliminated. Otherwise, if isomorphic kinematic chains are classified into categories, CAMD software will mislead an engineering design task. According to Hwang et al. [6] the most time consuming process in structural synthesis and analysis is testing and eliminating isomorphisms. So fast processing time for isomorphism detection is very important factor in mechanical engineering. Thus lowest time complexity for isomorphism detecting is the motivation for this project's work.

3. Background Literature Survey

In mechanical engineering, a kinematic chain is an assembly of rigid bodies connected by joints to provide constrained (or desired) motion that is the mathematical model for a mechanical system. While manufacturing a machine body, structure similarity identification is the foremost task, other-

wise a lot of time and effort can go in vain eventually. Many researchers have researched on finding the way of isomorphism on graphs. Uicker and Raicu [10] used the characteristic polynomial of the link-link adjacency matrix of the graph of the chain as an index of isomorphism. However, later it was proved that invariance of the polynomial characteristic was not sufficient condition for two graphs to be isomorph [7]. In 2002, Z. Chang et al. [1] proposed an approach based on eigenvalues and eigenvectors of adjacency matrices. P.S. Rajesh and C.S. Linda [9] proved that this approach is able to identify all non-isomorphic chains, with upto 14 links and one, two, three degrees of freedom, and pointed out that the time complexity would be exponential in the worst case. H. Ding and Z. Huang proposed the Canonical Perimeter Topological Graph and the Characteristic Adjacency Matrix approach to test isomorphism for both simple joint chains [3] and multiple joint chains [4]. The time complexity of their algorithm is $O(2^L)$, where L is the number of basic loops of a graph. In general, finding basic loops of a graph costs time with complexity of $O(n)$ or $O(n * n)$, or higher exponential order, depending on concrete algorithm, where n is the number of vertices. However, usually there are not many basic loops in a graph representing a mechanism kinematic chain, which makes this approach run very fast in the automatic synthesis of kinematic chains [2]. In this project's work, kinematic chains are represented by graphs and a novel fast algorithm named the Dividing and Matching Algorithm (DMA) is used. Its main contribution is to find two connection properties among vertices, named the expanded square degree and the correlation degree respectively, which are used to divide vertices of a graph into sets. In addition, it is proved that for vertices from two isomorphic graphs, only ones having the same connection properties are possible to be bijective. Accordingly, the vertices having the same connection properties compose a set. This avoids exhaustive search for matching vertices because more sets of vertices, less match. In the case that each vertex is a set, no search is needed for a match, which is the goal of developing a timesaving algorithm. The DMA algorithm is a deterministic algorithm and holds the high reliability and fast speed simultaneously. The time complexity of DMA is $O(M * n * n)$, where n is the number of vertices of a graph and M has no reasonable bound. DMA is faster than H. Ding and Z. Huang's approach and heuristic algorithms for some kinematic chain isomorphism tests.

The rest of this report is organized as follows. Next section introduces the basic knowledge of graph method for kinematic chain isomorphism identification. The later section presents the DMA algorithm in detail. After that section, the experiments and comparisons with existing algorithms are reported. Next section discusses the time complexity and reliability as well as effectiveness for large scale problems. Finally, I will represent the conclusion.

4. Terminology and definitions

4.1. Graph of a Kinematic Chain

A graph G (V,E) consisting of a set of vertices and edges (unordered pairs of vertices). Generally graph is used to represent kinematic chain while vertices are supposed to be the link of chain and edges are the joint of the chain. A graph representing a kinematic chain is a connected simple graph probably with a self-loop or self-loops. Consequently isomorphism identification of kinematic chains is rendered to isomorphism identification of corresponding graphs. For example, the graphs representing kinematic chains shown in the below figure are shown respectively. The DMA algorithm used in this project is dedicated for the connected simple graphs probably with self-loops, i.e. the graphs of kinematic chains.

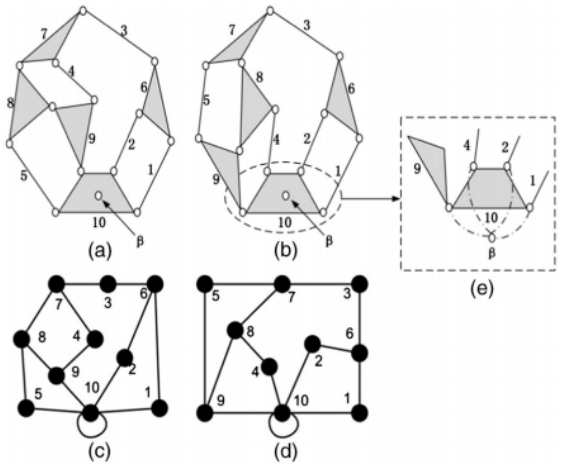


Figure 1. kinematic chains and corresponding graph representation.

4.2. Adjacency Matrix

- If there is an edge between the vertices of a graph, the vertices are called adjacent and their relationship is called adjacency relationship. 1 represents edge and 0 represents no edge.
- Used to represent graph in memory.

4.3. Graph Isomorphism

If two graphs are physically, but their vertices and edges aren't same in number then it is not isomorphic. If there belongs one to one mapping function between the vertices, the relationship is called as isomorphism. Two graphs are said to be isomorphic if and only if all properties of graph isomorphism are met otherwise there won't ne no isomorphism relationship. Properties of graph isomorphism are as follows :

- Number of vertices and edges, and parallel edges should be equal for both the graphs.
- In degree and out degree should be equal.
- Number of connected components should be same.
- Number of loops should be same.

These are the properties for identifying if two graphs are isomorphic or not.

Here are some isomorphic graphs in the figures below:

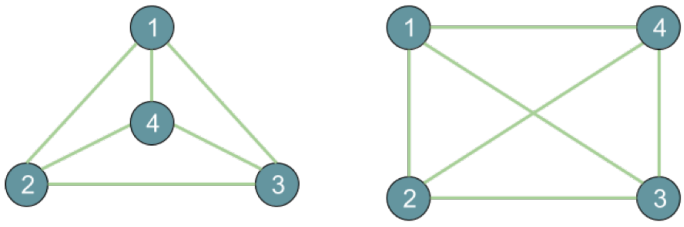


Figure 2. Isomorphic graphs.

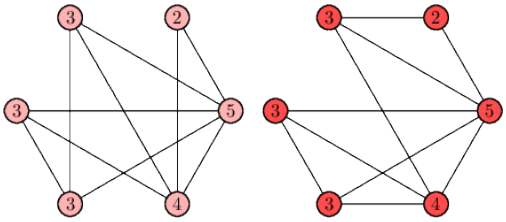


Figure 3. Non-Isomorphic graphs.

4.4. Sub-block of adjacency matrix

Sub-block of an adjacency matrix means a square sub-matrix along the principal diagonal. The biggest sub-block is the adjacency matrix itself and the smallest is a vertex. It is numbered from top-left to bottom-right.

v_{10}	v_9	v_7	v_8	v_6	v_1	v_3	v_4	v_5	v_2
1	1	0	0	0	1	0	0	1	1
1	0	0	1	0	0	0	1	0	0
0	0	0	1	0	0	1	1	0	0
0	1	1	0	0	0	0	0	1	0
0	0	0	0	0	1	1	0	0	1
1	0	0	0	1	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0

Figure 4. Sub-blocks of an adjacency matrix.

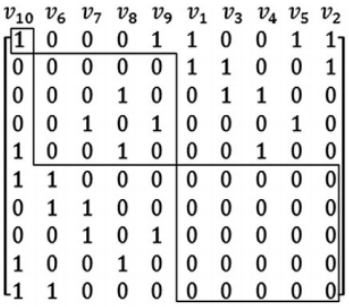
In the figure, the sub-blocks in an adjacency matrix where v10 is subblock1, and v9 and v7 compose subblock2, v8 is subblock3, v6 is subblock4 and the remaining nodes compose subblock5.

4.5. Degree Division

Degree Division is denoted by List(d) in an adjacency matrix. It means the descending sequence of the degrees of vertices of a graph via quick sorting. List(d) has an order of vertices.

4.6. Expanded Square Degree Division and List

Expanded Square Degree Division is the sum of squares of degrees of adjacent vertices for a vertex in a sub-block. This is denoted by S. Expanded Square Degree List means the descending sequence of S's of all vertices in a sub-block. This is denoted by List(S). For the given adjacency matrix, in the sub-block v6, v7, v8, v9 in the below figure v6 has no adjacent vertex; v7 has one



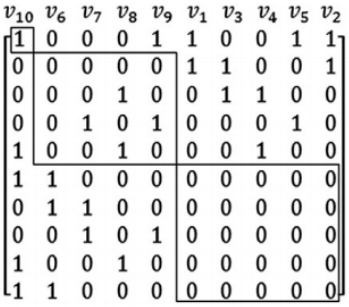
v_{10}	v_6	v_7	v_8	v_9	v_1	v_3	v_4	v_5	v_2
1	0	0	0	1	1	0	0	1	1
0	0	0	0	0	1	1	0	0	1
0	0	0	1	0	0	1	1	0	0
0	0	1	0	1	0	0	0	1	0
1	0	0	1	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0

Figure 5. Expanded Square Degree Division in a graph.

adjacent vertex v8; v8 has two adjacent vertices v7 and v9; v9 has one adjacent vertex v8. Thus, S of v6 is 0; S of v7 is the square of degree of v8 and equal to 4; S of v8 is 2; S of v9 is 4. In the sub-block v6, v7, v8, v9 in the below figure v6 has S of 0; S of v7 is equal to 4; S of v8 is 2; S of v9 is 4. For the sub-block the List(S) is 4, 4, 2, 0, and the order of vertices is v9, v7, v8, v6

4.7. Correlation Degree Division and List

Suppose, in a given graph, there are two sub-blocks, gn and gm in an adjacency matrix and a vertex vi in gn, the correlation degree, is the number of adjacent vertices of vi in gm. It is denoted by vi(gm). Correlation Degree Division List denotes the descending sequence of vi(gm)'s via quick sorting for all vertices in gn. This is denoted by List(gn(gm)). This maintains an order of vertices.



v_{10}	v_6	v_7	v_8	v_9	v_1	v_3	v_4	v_5	v_2
1	0	0	0	1	1	0	0	1	1
0	0	0	0	0	1	1	0	0	1
0	0	0	1	0	0	1	1	0	0
0	0	1	0	1	0	0	0	1	0
1	0	0	1	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0

Figure 6. Correlation Degree Division in a graph.

In this figure, in the adjacency matrix, v7 is in g2 and has two adjacent vertices v3 and v4 in g3, thus v7(g3) equals 2. In the adjacency matrix, in g2, v6 has 3 neighbors in g3, v7 has 2 neighbors in g3, v8 and v9 both have one neighbors in g3. So, List(g2(g3)) is 3, 2, 1, 1, and the order of vertices is v6, v7, v8, v9.

5. Methodology

DMA Algorithm has 5 parts. Here it is describing shortly according to its parts:

- First part is Degree Division checking between two graphs. If the degrees are different then the algorithm gives decision that graphs are not isomorphic. If the degrees are same then adjacency matrices of the graphs are adjusted based on the orders of the degree order for both the graphs.

- Second part is Expanded square degrees checking between two graphs. If they are not equal, the algorithm gives decision that graphs are not isomorphic. If they are equal then again adjacency matrices are adjusted according to the orders of vertices in the expanded square list for both the graphs.

- Third part is checking Correlation degrees of sub blocks between two graphs. If they are not equal, the algorithm gives decision that graphs are not isomorphic. Otherwise, new sub blocks are created and again check correlation degrees till it is possible to recreate new sub-blocks.

- Fourth part is making all the vertices into sub-blocks in both the graphs. Then all the sub-blocks of Graph 1 is compared to all the sub-blocks of Graph 2. If every sub-block of Graph 1 matches with the corresponding sub-block of Graph 2, then the algorithm gives decision that graphs are isomorphic. Otherwise the graphs are not isomorphic.

- Last part of DMA Algorithm is to find out the ultimate decision that the graphs are isomorphic or not using some extra checking using backtracking process.

6. Algorithms to be implemented

Algorithm 1: Degree Division

Input: two connected simple graphs G_1 and G_2

1 find the $List(d)$'s of G_1 and G_2, denoted by $List_{G_1}(d)$ and $List_{G_2}(d)$, respectively
2 **if** $List_{G_1}(d) \neq List_{G_2}(d)$ **then**
3 G_1 and G_2 are non-isomorphic
4 stop the procedure
5 **else**
6 adjust the adjacency matrices of G_1 and G_2 based on the orders of vertices of $List_{G_1}(d)$ and $List_{G_2}(d)$, respectively
7 **end if**

Figure 7. Algorithm 1.

Algorithm 2: Expanded square degree division

Require: there are two connected simple graphs G_1 and G_2; they have been divided by **Algorithm 1**.
/* a sub-block from two adjacency matrices are denoted by $G_1(i)$ and $G_2(i)$ respectively; */
1 **for** each i **do**
2 compare $List_{G_1(i)}(S)$ and $List_{G_2(i)}(S)$
3 **if** $List_{G_1(i)}(S) \neq List_{G_2(i)}(S)$ **then**
4 G_1 and G_2 are non-isomorphic /* according to **Theorem 1** */
5 **end if**
6 **end for**
7 adjust the adjacency matrices of G_1 and G_2 according to the orders of vertices of $List_{G_1(i)}(S)$ s and $List_{G_2(i)}(S)$ s, respectively

Figure 8. Algorithm 2.

Algorithm 3: Correlation degree division

Require: there are two connected simple graphs G_1 and G_2; they have been divided by **Algorithm 1** and then by **Algorithm 2**; and then, there are p sub-blocks $g_1, g_2, \ldots, g_p$ in each adjacency matrix; and, a sub-block in each adjacency matrix is denoted by $g_i(G_1)$ and $g_i(G_2)$ respectively.

1 **for** $i \leftarrow 1$ **to** $p-1$ **do**
2 **for** $j \leftarrow i+1$ **to** p **do**
3 find $List_{G_1}(g_i(g_j))$, $List_{G_2}(g_i(g_j))$, $List_{G_1}(g_j(g_i))$ and $List_{G_2}(g_j(g_i))$
4 **if** $List_{G_1}(g_i(g_j)) \neq List_{G_2}(g_i(g_j))$ or $List_{G_1}(g_j(g_i)) \neq List_{G_2}(g_j(g_i))$ **then**
/* according to **corollary** (1) or **corollary** (2) of **Theorem 2** */
5 G_1 and G_2 are non-isomorphic
6 **else if** some new sub-blocks are generated **then**
7 update p
8 Move the first new sub-block to the top-left to be g_1, for G_1 and G_2 respectively
9 $i \leftarrow 1$ and $j \leftarrow 2$
10 **end if**
11 **end for**
12 **end for**

Figure 9. Algorithm 3.

Algorithm 4: Dividing Vertex Algorithm (DVA)

Require: there are two connected simple graphs G_1 and G_2 ; they have been divided by **Algorithm 1**.

perform **Algorithm 2** for G_1 and G_2
perform **Algorithm 3** for G_1 and G_2
if every vertex is a sub-block **then**
    compare the adjacency matrices of G_1 and G_2
        /* according to **Theorem 3** */
    if they are equal **then**
        G_1 and G_2 are isomorphic
    else
        G_1 and G_2 are non-isomorphic
    end if
end if

Figure 10. Algorithm 4.

Algorithm 5: Dividing and Matching Algorithm (DMA)

Input: two connected simple graphs G_1 and G_2

Initialization:

c denotes the search degree of the backtracking process, $c \leftarrow 0$;
the adjacency matrices of G_1 and G_2 with a search degree are denoted by $AM1_c$ and $AM2_c$ respectively;
$AM1_c(g_i)$ and $AM2_c(g_i)$ denote the sub-blocks of $AM1_c$ and $AM2_c$ respectively;
j_c denotes the sequence number of a vertex in a sub-block with a search degree;
$AM1_c(g_i(v_{j_c}))$ and $AM2_c(g_i(v_{j_c}))$ denote the vertices in $AM1_c(g_i)$ and $AM2_c(g_i)$, respectively.

perform **Algorithm 1** for G_1 and G_2
function DMA ($AM1_c$ and $AM2_c$ **)**
    perform DVA (**Algorithm 4**) for G_1 and G_2
    if the result is that G_1 and G_2 are isomorphic **then**
        /* according to **Theorem 3** and **Theorem 4(2)**, G_1 and G_2 are isomorphic */
        stop the procedure
    end if
    if the result is that G_1 and G_2 are non-isomorphic **then**
        if $j_c <$ total of vertices in $AM1,2_c(g_i)$ **then**
            j_c ++
            exchange $AM2_c(g_i(v_{j_c}))$ and $AM2_c(g_i(v_1))$
            make $AM1_c(g_i(v_1))$ and $AM2_c(g_i(v_1))$ sub-blocks
            function DMA ($AM1_c$ and $AM2_c$ **)**
        else
            if $c = 0$ **then**
                /* according to **corollary** of **Theorem 4,** G_1 and G_2 are non-isomorphic */
                stop the procedure
            else
                c -- /* to back to the former state of $AM1$ and $AM2$ in backtracking process */
                j_c ++
                exchange $AM2_c(g_i(v_{j_c}))$ and $AM2_c(g_i(v_1))$
                make $AM1_c(g_i(v_1))$ and $AM2_c(g_i(v_1))$ sub-blocks
                function DMA ($AM1_c$ and $AM2_c$ **)**
            end if
        end if
    end if
    select any $AM1_c(g_i)$ and $AM2_c(g_i)$ which include more than 1 vertices
        and have the same i
    make $G_1(g_i(v_1))$ and $G_2(g_i(v_1))$ sub-blocks
    c ++ /* extend the search degree in backtracking process */
    function DMA ($AM1_c$ and $AM2_c$ **)**
end function

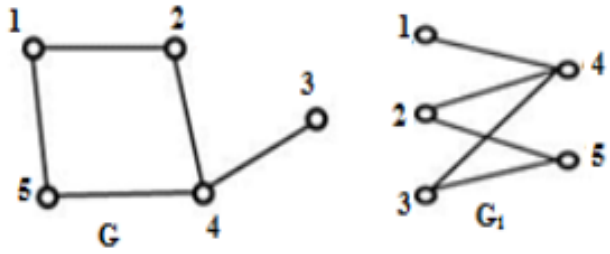
Figure 11. Algorithm 5.

7. My Work and future plan

I have implemented this project's algorithms in Python and have used Spyder (Python 3.6 version) integrated development environment (IDE) which is pre installed in Anaconda Navigator. As parallel with kinematic chains I also have checked my code for some other critical graphs to check the workability. Not all the tests are not possible to include in this report, I am including some of them of testing. In future I hope to work with this more on how to improve the time complexity of this algorithm. Some of the inputs are received from Zeng [11] et al.'s paper.

8. Experimental Results

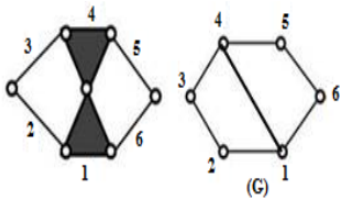
Testing Input1 and Output1:



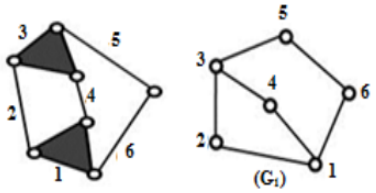

```
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.8.2] on linux
The input graphs are given below as adjacency matrices:

Graph A is:

[[0 1 0 0 1]
 [1 0 0 1 0]
 [0 0 0 1 0]
 [0 1 1 0 1]
 [1 0 0 1 0]]
Graph B is:

[[0 0 0 1 0]
 [0 0 0 1 1]
 [0 0 0 1 1]
 [1 1 1 0 0]
 [0 1 1 0 0]]

The Graphs are Isomorphic
```

Figure 12. Input and Output of testing1.

Testing Input2 and Output2:



Watt's chain and its graph



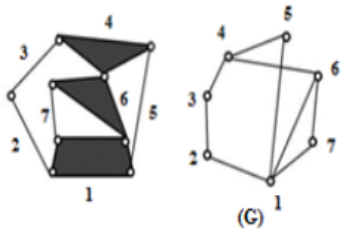
Stephenson's chain and its graph

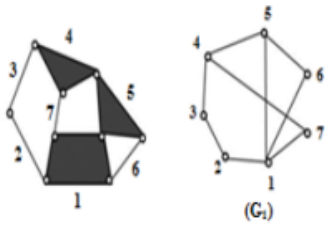
Fig. 3: Six links kinematic chains and their graphs

```
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.8.2] on linux
The input graphs are given below as adjacency matrices:

Graph A is:

[[0 1 0 1 0 1]
 [1 0 1 0 0 0]
 [0 1 0 1 0 0]
 [1 0 1 0 1 0]
 [0 0 0 1 0 1]
 [1 0 0 0 1 0]]
Graph B is:

[[0 1 0 1 0 1]
 [1 0 1 0 0 0]
 [0 1 0 1 1 0]
 [1 0 1 0 0 0]
 [0 0 1 0 0 1]
 [1 0 0 0 1 0]]

The Graphs are not Isomorphic
```

Figure 13. Input and Output of testing2.

Testing Input3 and Output3:

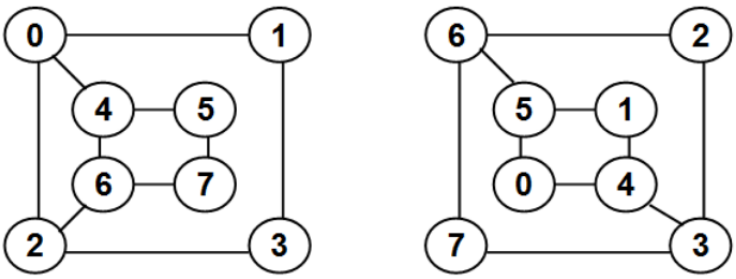





```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
The input graphs are given below as adjacency matrices:

Graph A is:

[[0 1 0 0 1 1 1]
 [1 0 1 0 0 0 0]
 [0 1 0 1 0 0 0]
 [0 0 1 0 1 1 0]
 [1 0 0 1 0 0 0]
 [1 0 0 1 0 0 1]
 [1 0 0 0 0 1 0]]
Graph B is:

[[0 1 0 0 1 1 1]
 [1 0 1 0 0 0 0]
 [0 1 0 1 0 0 0]
 [0 0 1 0 1 0 1]
 [1 0 0 1 0 1 0]
 [1 0 0 0 1 0 0]
 [1 0 0 1 0 0 0]]
The Graphs are Isomorphic
>
```

Figure 14. Input and Output of testing3.

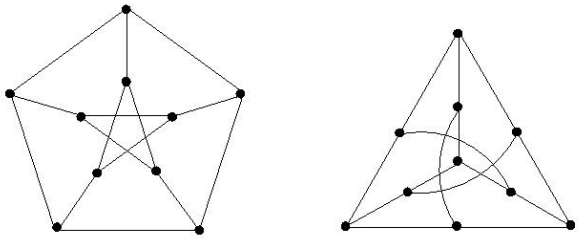
Testing Input4 and Output4:



```
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.8.2] on linux
The input graphs are given below as adjacency matrices:

Graph A is:

[[0 1 1 0 1 0 0 0]
 [1 0 0 1 0 0 0 0]
 [1 0 0 1 0 0 1 0]
 [0 1 1 0 0 0 0 0]
 [1 0 0 0 0 1 1 0]
 [0 0 0 0 1 0 0 1]
 [0 0 1 0 1 0 0 1]
 [0 0 0 0 0 1 1 0]]
Graph B is:

[[0 0 0 0 1 1 0 0]
 [0 0 0 0 1 1 0 0]
 [0 0 0 1 0 0 1 0]
 [0 0 1 0 1 0 0 1]
 [1 1 0 1 0 0 0 0]
 [1 1 0 0 0 0 1 0]
 [0 0 1 0 0 1 0 1]
 [0 0 0 1 0 0 1 0]]

The Graphs are not Isomorphic
```

Figure 15. Input and Output of testing4.

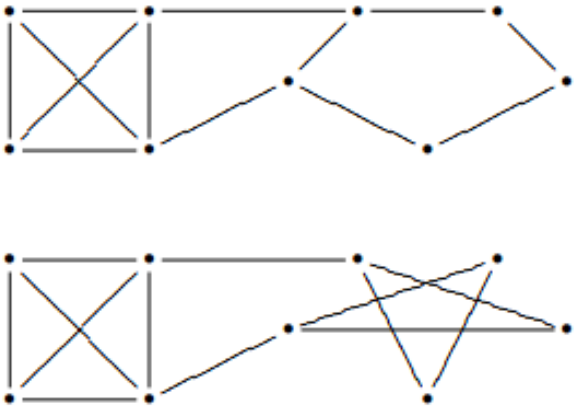
Testing Input5 and Output5:



```
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.8.2] on linux
The input graphs are given below as adjacency matrices:

Graph A is:

[[0 1 0 0 1 1 0 0 0 0]
 [1 0 1 0 0 0 1 0 0 0]
 [0 1 0 1 0 0 0 1 0 0]
 [0 0 1 0 1 0 0 0 1 0]
 [1 0 0 1 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 1 1 0]
 [0 1 0 0 0 0 0 0 1 1]
 [0 0 1 0 0 1 0 0 0 1]
 [0 0 0 1 0 1 1 0 0 0]
 [0 0 0 0 1 0 1 1 0 0]]
Graph B is:

[[0 1 0 0 0 1 0 1 0 0]
 [1 0 1 0 0 0 0 0 0 1]
 [0 1 0 1 0 0 0 0 1 0]
 [0 0 1 0 1 0 0 1 0 0]
 [0 0 0 1 0 1 0 0 0 1]
 [1 0 0 0 1 0 0 0 1 0]
 [0 0 0 0 0 0 0 1 1 1]
 [1 0 0 1 0 0 1 0 0 0]
 [0 0 1 0 0 1 1 0 0 0]
 [0 1 0 0 1 0 1 0 0 0]]

The Graphs are Isomorphic
```

Figure 16. Input and Output of testing5.

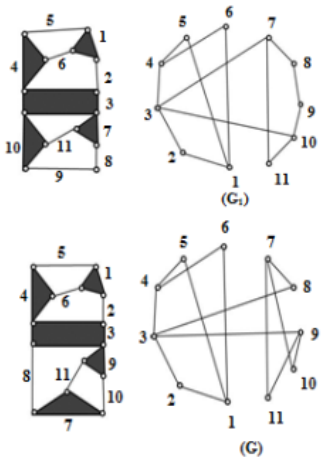
Testing Input6 and Output6:



```
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.8.2] on linux
The input graphs are given below as adjacency matrices:

Graph A is:

[[0 1 0 0 0 0 0 1 1]
 [1 0 1 0 0 0 0 1 1]
 [0 1 0 1 0 0 1 0 0]
 [0 0 1 0 1 0 0 0 0]
 [0 0 0 1 0 1 0 0 0]
 [0 0 0 0 1 0 1 0 0]
 [0 0 1 0 0 1 0 1 0]
 [1 1 0 0 0 0 1 0 1]
 [1 1 0 0 0 0 0 1 0]]
Graph B is:

[[0 1 0 0 1 1 0 0 0]
 [1 0 1 0 1 1 0 0 0]
 [0 1 0 0 0 0 1 1 0]
 [0 0 0 0 1 0 0 1 1]
 [1 1 0 1 0 1 0 0 0]
 [1 1 0 0 1 0 0 0 0]
 [0 0 1 0 0 0 0 0 1]
 [0 0 1 1 0 0 0 0 0]
 [0 0 0 1 0 0 1 0 0]]

The Graphs are not Isomorphic
```

Figure 17. Input and Output of testing6.

Testing Input7 and Output7:



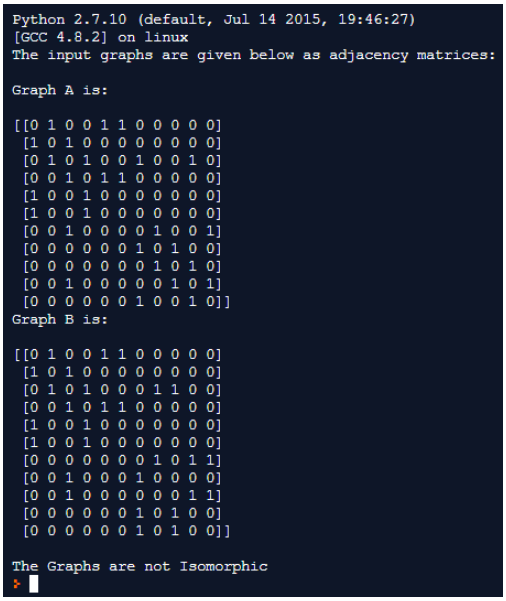


Figure 18. Input and Output of testing7.

References

[1] Z. Chang, C. Zhang, Y. Yang, and Y. Wang. A new method to mechanism kinematic chain isomorphism identification. *Mechanism and Machine Theory*, 37(4):411–417, 2002.

[2] H. Ding, F. Hou, A. Kecskeméthy, and Z. Huang. Synthesis of a complete set of contracted graphs for planar non-fractionated simple-jointed kinematic chains with all possible dofs. *Mechanism and Machine Theory*, 46(11):1588–1600, 2011.

[3] H. Ding and Z. Huang. A unique representation of the kinematic chain and the atlas database. *Mechanism and machine theory*, 42(6):637–651, 2007.

[4] H. Ding, W. Yang, P. Huang, and A. KecskemÃŠthy. Automatic structural synthesis of planar multiple joint kinematic chains. *Journal of Mechanical Design*, 135(9):091007, 2013.

[5] G. Galán-Marín, D. López-Rodríguez, and E. Mérida-Casermeiro. A new multivalued neural network for isomorphism identification of kinematic chains. *Journal of Computing and Information Science in Engineering*, 10(1):011009, 2010.

[6] W.-M. Hwang and Y.-W. Hwang. Computer-aided structural synthesis of planar kinematic chains with simple joints. *Mechanism and Machine Theory*, 27(2):189–199, 1992.

[7] T. Mruthyunjaya. Kinematic structure of mechanisms revisited. *Mechanism and machine theory*, 38(4):279–320, 2003.

[8] R. P. Sunkari. *Structural synthesis and analysis of planar and spatial mechanisms satisfying Gruebler's degrees of freedom equation*. PhD thesis, 2006.

[9] R. P. Sunkari and L. C. Schmidt. Reliability and efficiency of the existing spectral methods for isomorphism detection. *Journal of Mechanical Design*, 128(6):1246–1252, 2006.

[10] J. Uicker Jr and A. Raicu. A method for the identification and recognition of equivalence of kinematic chains. *Mechanism and Machine Theory*, 10(5):375–383, 1975.

[11] K. Zeng, X. Fan, M. Dong, and P. Yang. A fast algorithm for kinematic chain isomorphism identification based on dividing and matching vertices. *Mechanism and Machine Theory*, 72:25–38, 2014.