Solving Graph Isomorphism with Recursive Minimum Bit Strings

Josh Krcadinac 7624406  
Rafael Lastiri 7658033  
March 2015

Table of Contents

Graph Isomorphism 3

Sequential Representation of a Graph 3

Applications 4

Bit String Isomorphism Algorithm 4

Bibliography 5

# Graph Isomorphism

The definition of two graphs being isomorphic is being able to find a 1:1 mapping of the vertices in the first graph to the second graph such that adjacency is preserved [1]. In other words, can we take two graphs and move the vertices around so that it’s ordering of edges and vertices make it look like the other graph. In order to do this we require the two graphs that we check for isomorphism to have some similar properties and similar structure. If two graphs do not have the same properties then right away we can conclude that they are not isomorphic. After we determine if two graphs are candidates for being isomorphic there are a number of ways that we can check to see if they are indeed isomorphism’s of each other.

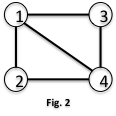
A few of the requirements that must first be met when checking for isomorphism are the number or vertices, the number of edges, vertex degrees, and structure between graphs. It is straight forward why we need the same number of nodes and edges in two graphs, without this consistency isomorphism is impossible. This leads to the next step of counting the nodes of similar degree. We must have the same number of vertices with similar degrees because this is the only way that we can create the 1:1 mapping correctly. After have determined that two graphs have these requirements then we can move on to checking the structure of the graphs and see if one can be mapped to the other.

# Sequential Representation of a Graph

Throughout this implementation adjacency matrices are used to represent the graphs. An adjacency matrix is a means of representing which vertices of a graph are adjacent to which other vertices. Specifically, the adjacency matrix of a finite graph G on *n* vertices is the *n x n* matrix where the non-diagonal entry *aij* is the number of edges from vertex *i* to vertex *j*. We assume that every vertex cannot have an edge to itself therefore the diagonal will always be filled with zeros. Furthermore, there exists a unique adjacency matrix for each isomorphism class of graphs, and it is not adjacency matrix of any other isomorphism class of graphs [5]. On this paper graphs are considered to be undirected which make the adjacency matrices to always be symmetric.



**Fig. 1**

 At the beginning of the implementation there will be two graphs generated with different orderings. Right away the adjacency matrix will get constructed for each graph. The adjacency matrix *Mij* has the following representation: *Mij = {1 if there is an edge between vi to vj, and 0 otherwise}*. An example of an adjacency matrix is shown on Fig. 1, which is a representation of the graph on Fig. 2. An adjacency matrix is chosen as the desired structure for this problem since they can be represented in a very compact way, occupying only *n\*(2/8)* bytes of contiguous space, where *n* is the number of vertices [6]

# Applications

While researching isomorphism we found many different applications that use graph isomorphism to solve real world problems. We provide in here some of the applications found, however this is not an exhaustive list. We discuss in brief applications in Chemistry, Civil Engineering, Cryptography, and social network pattern analysis.

In chemistry isomorphism can be used to check and see if molecular compounds are similar or not in structure [2]. This is helpful for identifying substances that are unknown to us using subgraph isomorphism. We can detect parts of the structures that we have previous knowledge of and figure out what compounds we are dealing with this way. In civil engineering we can use isomorphism to find geographic locations that have desired qualities. We would want to do this to find a prime location for where we want to construct buildings that have constraints on where they can or can’t be built. Isomorphism can also be applied to cryptography in an interesting way. The idea is to use an isomorphic S-Box instead of using the classical S-Box for AES encryption [3]. We are able to do replacement of the S-Box and still be able to keep the cryptographic properties of AES encryption as well as increase its complexity. Social networks are popular and it is possible to apply isomorphism to pattern analysis of these networks. It is possible to look for patterns in the network [4] that represent known bad behaviour to root out any suspicious activities taking place.

# Bit String Isomorphism Algorithm

As previously discussed a common problem in Graph Theory is to find if two graphs are isomorphic to each other. There are many algorithms out there to find if the graphs are isomorphic however this problem belongs to NP but unknown whether it belongs to NP-complete. As that implies, no polynomial time algorithm is known (despite many published claims), but neither is Graph Isomorphism known to be NP-complete. NP-completeness is considered unlikely since it would imply collapse of the polynomial-time hierarchy. The fastest proven running time for Graph Isomorphism has stood for three decades at [9, 10]. In this paper we discuss an idea given to us by a professor. For the first iteration on the implementation of our solution to the problem we want to have a working algorithm not worrying too much about efficiency. And if the algorithm is valid then on future work making the algorithm more efficient. The idea for our algorithm is described in more detail later, but it is complementary to an already known graph isomorphism algorithm that uses bit strings. In this section we describe such algorithm.

This algorithm is based on comparing the certificates of graphs. Two graphs G and H are isomorphic if and only if they have equal certificates, cert(G) = cert(H) [7, 8]. As previously discussed the adjacency matrices will be constructed with the supplied graphs, which have a specific ordering. Changing the ordering of the rows and columns will change the matrix. The upper triangle contains bits which can be written as a single binary number, row after row, or column after column. Each ordering of the vertices set defines a bit string in this way. The bit string is defined by taking the upper triangle of the adjacency matrix (column by column). As an example the bit string for the adjacency matrix on Fig. 3 would be 1101010110. These bit strings can be ordered lexicographically, and the smallest (or largest) can be taken as the certificate of graph G represented as cert(G). We say that cert(G) corresponds to the *smallest adjacency matrix* for G. When defined in this way, cert(G) is obviously independent of the original ordering of the vertices. The disadvantage is that there are *n!* different orderings of the set of vertices [8]. The minimum bit string algorithm works the following way:



**Fig. 3**

*Input: Two adjacency matrices representing graphs G and H that are being tested for isomorphism.  
Output: Boolean true – graphs are isomorphic to each other  
 false – graphs are not isomorphic to each other*

**Boolean MinBitString(Gij , Hij )**

*If the graphs are not candidates to be isomorphic   
 return false*

*Construct the bitstrings for each matrix and ordered them lexicographically to get the certificates.*

*If the certificates are the equivalent  
 return true*

*For each mapping of graph H construct a certificate*

*If cert(G) == cert(H)  
 return true*

*return false*

As you may notice from the algorithm if the graphs are not isomorphic it would take on the worst case *O(n!)* since it would have to compare every mappings of G, with the certificate of H.

# Bibliography

[1] The Graph Isomorphism Problem, Scott Fortin, 1996

[2] An Efficient Algorithm for Similarity Analysis of Molecules, Johnnie Baker, Chun-che Tsai, Rohit Pasari, Paul Durand, 1999

[3] A New S-Box Structure Based on Graph Isomorphism, Bao Tran, Thuc Nguyen, Thu Tran, 2009

[4] Social Network Analysis: Methods and Applications, Stanley Wasserman, Katherine Faust, 1999

[5] [www.en.wikipedia.org/wiki/Adjacency\_matrix](http://www.en.wikipedia.org/wiki/Adjacency_matrix)

[6] Harmanjit Singh, Richa Sharma, Role of Adjacency Matrix & Adjacency List in Graph Theory, International Journal of Computers & Technology, 2012, Vol.3(1), p.179

[7] R.C. Read and D.G. Corneil, The graph isomorphism disease, J. *Graph Theory 1* (1977), 339-363.

[8] W. Kocay, “On Writing Isomorphism Programs”, in Computational and Constructive Design Theory, edited by W.D. Wallis, Kluwer Academic Publishers, 1996.

[9] Babai, L., Kantor, W.M. and Luks, E.M. 1983. Computational complexity and the classification of finite simple groups. In: Proceedings of the 24th Annual Symposium on the Foundations of Computer Science, 162-171.

[10] McKay, Brendan D.; Piperno, Adolfo, “Practical graph isomorphism II”, eprint arXiv:1301.1493, 2013