Solving Graph Isomorphism with Recursive Minimum Bit Strings

Josh Krcadinac 7624406
Rafael Lastiri 7658033
March 2015

Table of Contents

Introduction 3

Graph Isomorphism 3

Sequential Representation of a Graph 3

Applications........ 4

Minimum Bit Strings 5

Recursive Minimum Bit Strings 5

Results........ 6

Conclusion........ 7

Bibliography 8

Introduction

We decided to do our project on Graph Isomorphism, specifically on how to solve the problem using the minimum bit string method. Our idea was to use the minimum bit string method and try to make a recursive version of the solution by breaking a graph into smaller pieces. The idea was presented by William Kocay who was teaching us Graph Theory over the Winter 2015 term [1]. This idea stimulated us and was the main source of motivation for our project. In this report we will discuss the Graph Isomorphism problem, representing graphs using adjacency matrices, applications of isomorphism, the minimum bit string method, relevant work, project findings and future research.

Graph Isomorphism

The definition of isomorphism is being able to find a 1:1 mapping of the vertices between graphs such that adjacency is preserved [2]. In other words, can we move around the vertices of one graph so that the ordering of edges and vertices make it look the same as the second graph. In order to do this we require that two graphs have some similar properties and similar structure. If two graphs do not have the same properties then we can immediately conclude that they are not isomorphic. If we determine two graphs are candidates for being isomorphic, there are a number of ways that we can check to see if they are indeed isomorphic to each other.

A few of the requirements that must be met when checking for isomorphism are the number of vertices, the number of edges, vertex degrees, and structure between graphs. We need to check these properties because if they are not consistent between two graphs there is no chance for isomorphism. This leads to the next step of counting the degrees of every vertex. We must have the same number of vertices with similar degree because this is the only way that we can create the 1:1 mapping successfully. After having determined that two graphs meet all of these requirements, we can then move on to check the structure of the graphs and see if one can be mapped to the other.

This problem is known to be in NP but it is unknown whether it belongs in NP-complete. As that implies, no polynomial time algorithm is known for the Graph Isomorphism problem. 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 $e^{O(\sqrt{nlogn})}$ [3].

Sequential Representation of a Graph

Throughout our project, adjacency matrices are used to represent graphs. An adjacency matrix is a means of representing which vertices of a graph are adjacent to each other. Specifically, the adjacency matrix of a graph G with n vertices is the n x n matrix where an entry on row i, column j is 1 if the vertices i and j are adjacent to each and 0 if they are not. 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 the adjacency matrix of any other isomorphism class of graphs [4]. Our project only deals with undirected graphs which always makes the adjacency matrices symmetric across the diagonal. 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 [5]. An example of an adjacency matrix is shown in Figure 1 which is a representation of the graph in Figure 2.

$$M_{ij} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Fig. 1

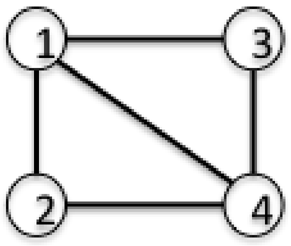


Fig. 2

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 [6]. 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. We can also use isomorphism to find if two structures of molecules can have the same formulaic makeup. In civil engineering we can use isomorphism to find geographic locations that have desired qualities. We can find prime locations for where we want to construct buildings that have constraints on where they are able to be built. Isomorphism can also be applied to cryptography by means of encryption with AES. The idea is to use an isomorphic S-Box instead of using the classical S-Box for encryption [7]. We are able to replace 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 [8] that represent known bad behaviour to root out any suspicious activities taking place. It is also possible to pick up user habits and be able to target certain profiles of user with certain information.

Minimum Bit Strings

The minimum bit string method 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) [9]. 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 value ordering inside the matrix. The upper triangle contains $\binom{n}{2}$ bits which can be written as a single binary number. We only care about the upper triangle as show in Figure 3 because the lower triangle contains the exact same information and we will not use it. We take the bits in the upper triangle either row by row or column by column. We are then able to append these rows or columns to each other in different orderings, producing a string of bits. The minimum value out of every possible ordering gives us the minimum bit string for that adjacency matrix, cert(G). The specifics of figuring out how to get the minimum bit string are not the most important part. We just need a consistent way to get a low value for every graph every time. In the implementation for our project we define the minimum bit string by the following process:

$$M_{ij} = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Fig. 3

1. Take the rows in the upper right triangle as our pieces (1101, 010, 11, 0)
2. Sort them lexicographically (0, 010, 11, 1101)
3. Append pieces together from right to left until a piece of all 1’s is encountered (0010)
4. Append to current bit string the pieces in our list from left to right (0010110111)
5. Resulting bit string is the minimum for the current vertex ordering

This process gives a way to define the minimum bit string in O(nlogn) time using sorting instead of O(n!) by trying every single combination. If we were to reorder the vertices in the adjacency matrix then we could then define a different minimum bit string for the reordering. Each ordering of the vertices defines a minimum bit string in this way. The disadvantage of this method is that there are n! different orderings of the set of vertices [9].

Recursive Minimum Bit Strings

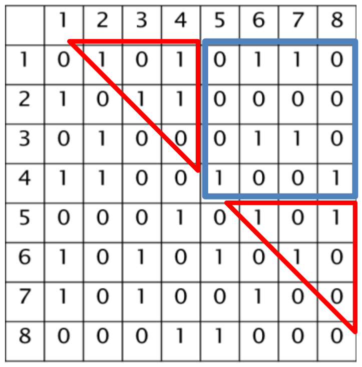
Our project will deal with breaking a graph into smaller pieces then using the Minimum Bit String method to solve the individual pieces. There is some research pertaining to using certificates of two graphs to see if they are isomorphic. The idea is that if two graphs can make the same certificate in a unique way then they are isomorphic [9]. The method described in the paper by Kocay touches on using minimum bit strings but is quite advanced for the idea behind our project. We want a simple way to construct the bit strings and it turns out this way is straight forward and inefficient. The process we determined would work is described in the above section.

Late in our research we came upon a paper that dealt with Code Equivalence [10]. This essentially describes the use of certificates from adjacency matrices but shows a different way to compute a certificate. The method uses Hamming weights and matrix multiplication to generate certificates. Since the timing was poor for when we discovered this idea we were not able to look into it further. Besides the aforementioned ideas we were not very successful in discovering methods dealing with minimum bit strings and graph isomorphism. We reason that bit strings are a somewhat inefficient/difficult method to use and there are other efficient/easy algorithms that already exist for use.

Results

The results for our project showed a better understanding of how to determine and compare graphs using minimum bit strings. When we attempted to solve the problem using recursion we encountered great difficulty due to dealing with certain sections of the adjacency matrix when coming out of recursion. We were unable produce a reclusive solution, however we program a working solution using the Minimum Bit String method.

We decided to start by programming a working solution to graph isomorphism to help better out understanding of using minimum bit strings. This solution worked well although it was inefficient. The running time was O(n!) time where n is the number of vertices in the graph. The time to complete a comparison of two graphs of size 10 took more than 20 seconds to process in some cases. This time increased at an amazing rate for every node added to the two graphs.



	1	2	3	4	5	6	7	8
1	0	1	0	1	0	1	1	0
2	1	0	1	1	0	0	0	0
3	0	1	0	0	0	1	1	0
4	1	1	0	0	1	0	0	1
5	0	0	0	1	0	1	0	1
6	1	0	1	0	1	0	1	0
7	1	0	1	0	0	1	0	0
8	0	0	0	1	1	0	0	0

Figure 4

Our idea to break a graph into smaller pieces and find the minimum bit string of those pieces did not work. At the bottom of the recursion we were able to find the minimum bit string of the individual pieces of the adjacency matrix, but when coming out of the recursion we had trouble dealing with the part of the matrix that was untouched. This section is outlined in blue in Figure 4. We thought that maybe if we arrange the sections touched by the minimum bit string method (red section in Figure 4) that the other areas would happen to sort themselves out but they did not. We were expecting this problem. We then tried checking to see if we could sort coming out of every step in the recusrion, or at least check to see if the ordering we had was close to a minimum. We found that and processing coming out of the recusrion generated too much overhead and that our running time was taking as long as our previously tested algorithm, sometimes longer. We tried different techniques such as breaking pieces into different sizes, sorting by columns instead and even other types of sorting pieces into minimum bit strings. None of these ways worked and our resulting program did not successfully compare two isomorphic graphs using a recusrive minimum bit string method.

We reasoned that it was possible our approach of setting out to program a solution was not the most optimal solution. Our code needed such a big overhaul and reorganizing to support recusrion and minimum bit strings. We could have started with more theorectical research and possibly looked into some Machine Learning or Computer Engineering reasearch for answers.

Conclusion

In conclusion we were not successful with the idea we had originally set out to complete. We encountered problems with coding and theory that blocked us. Despite our setbacks we learned a lot more about the topic of Graph Isomorphism. We wish that we had more time to delve deeper into other ideas and possibly try more options for finding our solution.

We would have liked to look more into the Code Equivalence topic and see how it could apply to isomorphism. It would be nice to see if we would be able to apply the idea directly into our code and solve the problem we set out to research. This may have even included using the whole adjacency matrix instead of just the upper right triangle. In the future if we were to come back to the project we would like to improve our current solution and make it run faster. This could possibly lead to code shortcuts that may or may not show us a way that we could make our solution work.

Bibliography

[1] W. Kocay, *Conversation,* 2015.

[2] S. Fortin, "The Graph Isomorhpsim Problem," Department of Computer Science, University of Alberta, Edmonton, 1996.

[3] A. P. Brendan D. McKay, "Practical Graph Isomorphism II," *Journal of Symbolic Computing,* vol. 60, pp. 94-112, 2014.

[4] "Wikipedia," 2015. [Online]. Available: http://en.wikipedia.org/wiki/Adjacency_matrix.

[5] R. S. Harmanjit Singh, "Role of Adjacency Matrix & Adjacency List in Graph Theory," *International Journal of Co,* vol. 3, no. 1, pp. 179-183, 2012.

[6] R. P. J. W. B. C.-c. T. Paul J. Durand, "An Efficient Algorithm for Similarity Analysis of Molecules," Department of Mathematics/Computer Science, Kent State University, Kent.

[7] T. D. N. T. D. T. Bao Ngoc Tran, "A New S-Box Structure Based on Graph Isomorphism," in *International Conference on Computational Intelligence and Security*, 2009.

[8] K. F. Stanley Wasserman, Social Network Analysis: Methods and Applications, Cambridge: Cambridge Univeristy Press, 1999.

[9] W. Kocay, "On Writing Isomorphism Algorithms," in *Computational and Constructive DEsign Theory*, Kluwer Academic Publishers, 1996, pp. 135-175.

[10] E. P. a. R. M. Roth, "Is Code Equivalence Easy to Decide?," *IEEE TRANSACTIONS ON INFORMATION THEORY,* vol. 43, no. 5, pp. 1602-1605, 1997.