http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 4 (1): 6-10, 2005
ISSN 1812-5638

Matrix Equality: An Application of Graph Isomorphism

Khadija Riaz, Malik Sikander Hayat Khiyal and Muhammad Arshad
Department of Computer Science, Faculty of Applied Sciences,
International Islamic University, Islamabad, Pakistan

Abstract: In this study we discuss about an application of graph Isomorphism. If we look at two unequal matrices of undirected graphs, we cannot say either the graphs of these matrices are isomorphic or not isomorphic. It is because the two matrices may look different even the graphs are isomorphic. In this study, by using our algorithm of Graph Isomorphism it can be told about the given unequal matrices that the matrices have isomorphic graphs or not. In the case the graphs are isomorphic; the given unequal matrices are made equal by moving the vertices of a matrix according to the corresponding vertices in other graph.

Key words: Discrete structures, graphs, algorithms, isomorphism and matrices

INTRODUCTION

Among all the problems defined on graphs, the exact isomorphism[1] and the sub-graph isomorphism detection play a key role and are used in a variety of real applications, such as chemistry, switching theory, information retrieval, networking and linguistics. Two graphs, G=(V,E) and H=(W,F), are *isomorphic* (normally written in the form G=H, where the = should have a third wavy line above the two parallel lines), if there are bijections f:V->W and g:E->F . Two isomorphic graphs must have exactly the same set of parameters as given in Fig. 1. For example, the cardinalities of the vertex sets must be equal, the cardinalities of the edge sets must be equal, the (ordered) degree sequences must be the same, any graph polynomials must agree on the two graphs, etc. Let G1 and G2 be graphs with vertex sets V(G1) and V(G2) and edges sets E(G1) and E(G2), respectively. G1 is isomorphic to G2 if and only if there exist, one-to-one correspondences of vertices and edges for example:

g: V(G1)→V(G2) and h:E(G1)→E(G2)

The two graphs are shown in Fig. 2.

It is often difficult to determine whether two simple graphs are isomorphic[2]. There are n! possible one-to-one correspondences between the vertex sets of two simple graphs with n vertices. Testing each such correspondence to see whether it preserves adjacency and non-adjacency is impractical if n is large. An algorithm guaranteeing a solution in running time proportional to a constant power of n is desirable, but no such algorithm has been discovered for determining if two arbitrary graphs are isomorphic[2,3]. However, we can often show that two graphs are not isomorphic by showing that

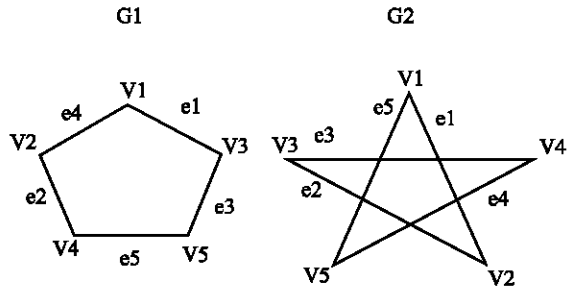

Fig. 1: A pair of isomorphicgraphs

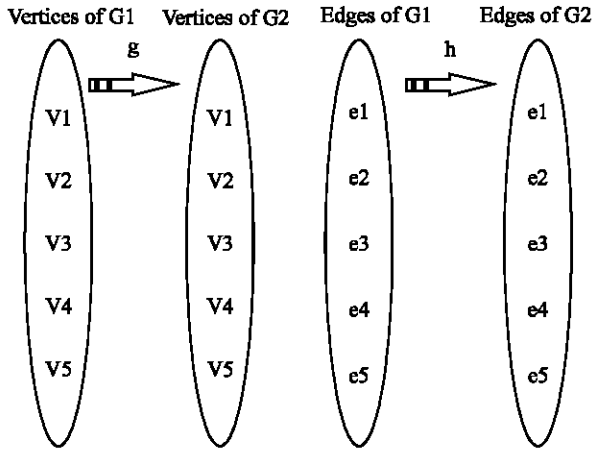

Fig. 2: One to one correspondence of Vertices and Edges for G1 and G2

they do not share a property (invariant) that isomorphic simple graphs must have. The number of vertices, the number of edges and degrees of the vertices are all

Corresponding Author: Dr. Malik Sikander Hayat Khiyal, Department of Computer Science, Faculty of Applied Sciences, International Islamic University, Islamabad, Pakistan

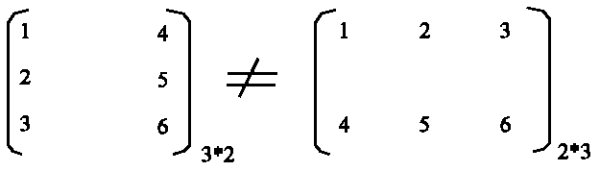

Fig. 3: Matrices having same number of elements but not equal

G1	v1	v2	v3	v4	v5
v1	0	1	1	0	0
v2	1	0	0	1	0
v3	1	0	0	0	1
v4	0	1	0	0	1
v5	0	0	1	1	0

G2	v1	v2	v3	v4	v5
v1	0	1	0	0	1
v2	1	0	1	0	0
v3	0	1	0	1	0
v4	0	0	1	0	1
v5	1	0	0	1	0

Fig. 4: Unequal Matrices for Isomorphic graphs G1 and G2

invariants under isomorphism[4-6]. If any of these quantities differ in two simple graphs, these graphs cannot be isomorphic. However, when these invariants are the same, it does not necessarily means that the two graphs are isomorphic. Matrix equality is a different concept, for two matrices to be equal, they must have:

- The same dimensions.
- Corresponding elements must be equal.

In other words, say that $A_{n \times m} = [a_{ij}]$ and that $B_{p \times q} = [b_{ij}]$. Then A = B if and only if n=p, m=q and $a_{ij}=b_{ij}$ for all I and j in range. Here are two matrices which are not equal even though they have the same elements. As shown in Fig. 3.

The pseudo code for isomorphism algorithm has presented[1], where we discussed about the design and development of the algorithm. This study present the performance and implementation of algorithm by using an application of matrix equality.

PROBLEM

The graphs make square matrices. Two isomorphic graphs may have unequal matrices. Matrix equality depends upon:

- How graphs are labeled.
- Graphs are isomorphic or not

If two isomorphic graphs are labeled in the same way, matrices of such graphs would be equal otherwise matrices are unequal. In this study we discussed on the application of graph isomorphism. Let us consider the isomorphic graphs given in Fig. 1. the matrices of these graphs are presented in Fig. 4.

By observing the two unequal matrices we cannot say that these are matrices of two isomorphic graphs. The algorithm[1] works for this problem. It can be easily finds that the input unequal matrices make isomorphic graphs or otherwise. If these are matrices of isomorphic graphs then the matrices are made equal. A data structure is required which can find the one-to-one correspondence between vertices of two isomorphic graph matrices and change the matrix elements of one matrix in a way that it becomes equal to the other matrix. In this study we use the algorithm of graph isomorphism to make matrices equal. We have implemented this application in C and found successful for every kind of undirected graph.

USE OF ISOMORPHISM ALGORITHM TO SOLVE THE PROBLEM

In order to solve the problem, by using algorithm of graph isomorphism[1], we proceed as follows:

Procedure:

Step 1. Input: The input of the problem is two unequal matrices. Initially the total number of vertices is asked in each matrix. If both matrices have same number of vertices then the further input is taken and is stored in memory. After the required information has been taken the remaining invariants are checked to ensure that both of the matrices have same invariants. The matrices having same invariants may or may not be isomorphic but if they have difference in invariants then the matrices would not be isomorphic and therefore they cannot be made equal. So the further processing is stopped.

Invariants: The invariants used in our algorithm are as follows:

- Number of vertices should be same in both matrices.
- Number of edges in both matrices should be equal.
- In both of the matrices the vertices having same degree are grouped to form classes. The number of classes in both of the matrices should be equal.
- Total degree of matrices should be equal.

After taking input graphs, the above invariants are checked. If any of these quantities differ in two graphs, we prompt a message that the graphs are not isomorphic. However when all these invariants have been satisfied further processing starts. Consider the following two input matrices in Fig. 5.

The data structure used to store the matrices is Ap-vector of ADT class. The data type of the vector is user defined linked list. The information about vertices of both matrices are stored in a vector, called LIST, while the information about the adjacent vertices of each vertex for both matrices are stored in a linked list called NODES.

	M (h1)					M (h2)				
	v1	v2	v3	v4	v5	v1	v2	v3	v4	v5
v1	0	2	0	0	1	0	0	1	1	0
v2	2	0	0	0	1	0	0	0	1	2
v3	0	0	0	1	1	1	0	0	1	0
v4	0	0	1	0	1	1	1	1	0	1
v5	1	1	1	1	0	0	2	0	1	0

Fig. 5: Two input matrices

After filling all information in the vector, the matrices M(h1) and M(h2) in the memory are shown in figure 6, where the fields for pointers are not presented. The matrices are stored in the way that to the right side of the list is matrix M(h1) and to the left side is matrix M(h2). No is the part of each element in which vertex number is stored and is separation between two matrices. d1 stores degree of the vertex for h1 and d2 contains degree for h2 vertices. The vertices of same degree are grouped into classes.

Step 2. Call isomorphism ( ) function: This function checks either the matrices are isomorphic or not. The detailed algorithm is given by Riaz *et al.*[1]. In the case the matrices are isomorphic; the function finds all corresponding vertices. The matrix to the right side is treated as the standard matrix and the its corresponding vertices are found in other matrix which is at the left side of the List. Each res field in M(h1) points to the corresponding vertex in M(h2). In the case, the matrices are not isomorphic; a message is prompted that matrices cannot be made equal. Consider the input matrices in Fig. 5. The matrices are unequal, but after call to the function isomorphism () it is found that the two matrices are isomorphic. The graphical representation of matrices in Fig. 5 is given in Fig. 7.

Step 3. Repositioning of vertices: After the corresponding vertices of the standard matrix have been found, the vertices in other matrix are repositioned according to the correspondence. By changing position of vertices the matrix becomes equal to the standard matrix, as shown in Fig. 8. It means if we re label the graphs in Fig. 7 according to the corresponding vertices given in Fig. 9. Both matrices will have all same values and are equal now,

Re labeling in graph h2 will be as follows:

v1 → v2
v2 → v5
v3 → v1
v4 → v3
v5 → v4

PERFORMANCE

In order to understand the performance of the application, it is important to discuss the basic functionality of Isomorphism algorithm for the convenience of the readers.

Little about isomorphism algorithm: We discuss some functionality of the algorithm developed by Riaz *et al.*[1]. Here we use the term graph rather then matrix, it is because graph can be retrieved if matrix is available and vice versa. The data structure used to store graphs and matrices is same. To understand the functionality of algorithm one should keep in mind the memory figure presented in Fig. 6.

- The invariants of both graphs are compared.
- An Ap-vector of Linked List is used as book keeping device for graphs.
- The vertices of same degree are grouped into classes C1, C2 and so on.
- First element of the first class from standard graph is picked and is compared with the elements of same class in second graph. The picked vertex is called main vertex. Each and every field of a main vertex and the adjacent vertices are compared.
- A field named as *attach* performs an important role in comparison of vertices Once a corresponding vertex from graph h2 has been found for a vertex in graph h1. All the adjacent elements of both the vertices are also corresponding to each other. So there is no need to compare them again. And if any of the adjacent vertex is not found in h2. It means that the graphs cannot be isomorphic.

Memory: Figure 6 shows that the matrices/graphs are stored in a way that they occupy minimum memory in order to solve the problem. If we use arrays as book keeping device for graphs/matrices then the memory used is maximum. For n vertices there will be (n*n) elements used and if we require some other information according to any algorithm then a field will be added in each and every element of the two dimensional matrix. In the case of linked list (n+p) elements are reserved for one matrix/graph to store the basic information where n is total number of vertices and p is sum of adjacent vertices to each vertex. If we require some extra information about vertices then the adjacent vertices will add the required field. We make use of linked list and ap-vector (ADT class). In this algorithm one important aspect regarding memory is that, we do not reserve two different vectors for two graphs, rather we have made one vector that is pointing both the graphs. It is also helpful in fast

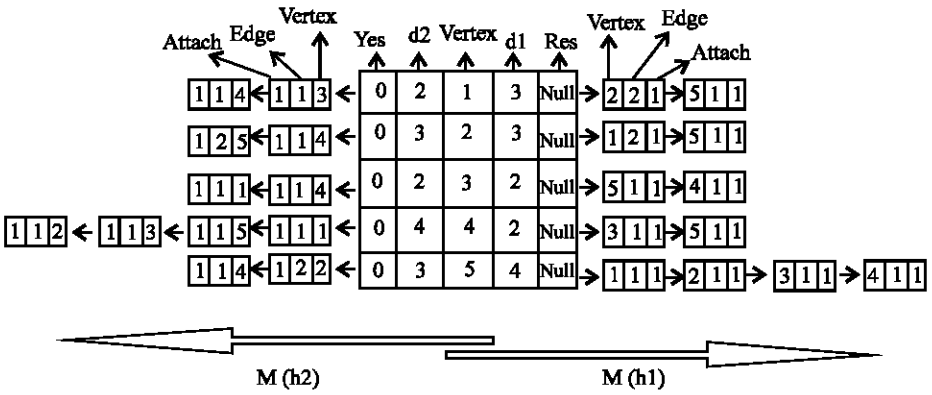

Fig. 6: Input matrices are stored in Ap-vector of linked list

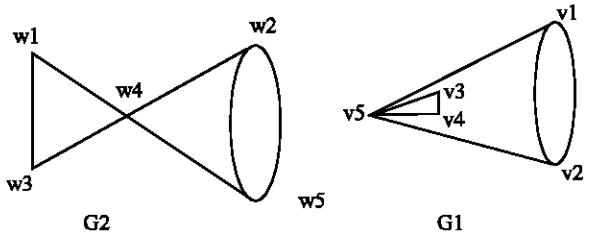

Fig. 7: Input matrices have isomorphic graphs

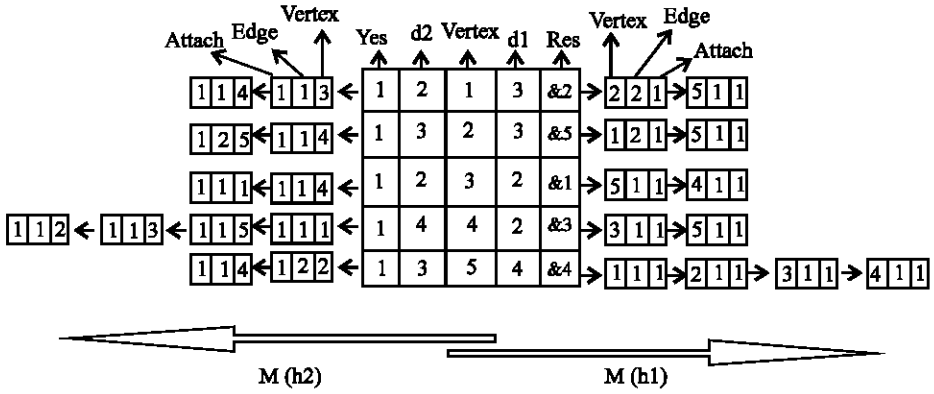

Fig. 8: Function isomorphism () return the data structure after finding the corresponding vertices

M (h2)

	v2	v5	v1	v3	v4
v2	0	2	0	0	1
v5	2	0	0	0	1
v1	0	0	0	1	1
v3	0	0	1	0	1
v4	1	1	1	1	0

M (h1)

	v1	v2	v3	v4	v5
v1	0	2	0	0	1
v2	2	0	0	0	1
v3	0	0	0	1	1
v4	0	0	1	0	1
v5	1	1	1	1	0

Fig. 9: The matrix in M (h2) is now equal to the standard matrix M (h1) after repositioning of vertices

processing as well. So the memory used in this algorithm is as minimum as it can be for solving this problem.

Timing efficiency: As we have solved the problem of matrix equality by using isomorphism algorithm. The processing of application is as efficient as our algorithm is. Followings are the main points regarding timing aspect:

- If input matrices cannot be made equal because their graphs are not isomorphic, this is found in fast polynomial time.

- When a vertex (of standard matrix) in any class has been compared with its corresponding vertex in M2 and found successful results. The adjacent vertices will directly be compared. If all comparisons are successful, all the adjacent vertices are also corresponding to each other. The corresponding vertices are therefore found in polynomial time.
- After we get a list of corresponding vertices there is no need to move them actually in data structure. Rather the *Res* field of each vertex in standard matrix M1 points towards its corresponding vertex in M2 matrix.

CONCLUSION

Our algorithm provides a successful and reliable mechanism to make matrices of undirected isomorphic graphs equal. The matrices for isomorphic graphs, labeled in different ways, are unequal. To make these matrices equal we have to get the corresponding vertices. By keeping one matrix standard we change the positions of vertices in other matrix according to their correspondences. In this way we can solve the problem of making unequal matrices equal.

ACKNOWLEDGMENTS

The authors wish to thank almighty Allah who enabled them to do such type of research work. We are grateful to Mr. Nadeem, for his technical help in writing the program in C++. We are also Grateful to Dr. Khalid Rashid for his moral support and encouragement.

REFERENCES

1. Riaz, K., M.S.H. Khiyal and M. Arshad, 2003. An improved algorithm to discover graph isomorphism. Proceedings INMIC 2003, 7th International Multi Topic Conference 8-9 December, pp: 396.
2. Rose, K.H., 2000. Discrete Mathematics and Its Applications, McGraw-Hill International Edition.
3. Deo, N., 1993. Graph Theory with Applications to Engineering and Computer Science. Pentice-Hall Englewood Cliffs, N.J., USA., pp: 480.
4. EPP, S.S., 1996. Discrete Mathematics with Applications. 2nd Edn., PWS Publishing Company, pp: 828.
5. Standish, T.A., 1994. Data Structures, Algorithms and Software Principles. 1st Edn. Addison Wesley Longman, pp: 748.
6. Rouvray, D.H. and A.T. Balaban, 1979. Chemichal Applications of Graph Theory, Applications of Graph Theory, Academic Press, New York, pp: 177-221.