Exploring Algorithms to Solve The Isomorphic Graph Problem

Stu Synakowski
Clarkson University Department of Mathematics
synakosr@clarkson.edu

Abstract

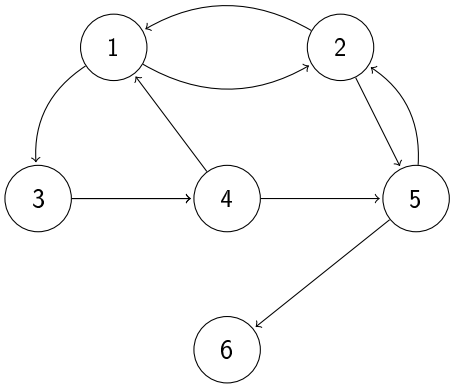
This report will be exploring various methods to compare and determine whether two graphs are isomorphic. We will first inspect an exhaustive search algorithm and verify its impracticality by showing its computation has an exponential running time. We will also explore more useful algorithms inspired by Google's web-page ranking algorithm. The ability to determine if two graphs are isomorphic has myriad applications including object and facial recognition.

I. Introduction

Quantifying and comparing the intrinsic properties of an object is often very difficult. Currently humans have an astounding ability to recognize an object, people, and their emotion with little to no effort. The current trend in mathematics and computer science is to emulate the strengths of the human brain when it comes to their methods of computation. Graph theory is an excellent tool to bridge our understanding of mathematics as and the innerworkings of our minds. The classic comparison problem in graphs is known as the isomorphic graph problem, in which two graphs are identical but arranged in a completely unrecognizable form. This report will begin to explain some fundamentals of graph theory and analyze the running times of various methods that attempt to solve the isomorphic graph problem.

II. Notation and Backround

To properly clarify what exactly the isomorphic graph is, let us give a brief introduction to graph theory with a simple example. Suppose we are examining a directed graph of node size n = 6, with the following edge configuration labeled with arrows pointing to other various nodes.



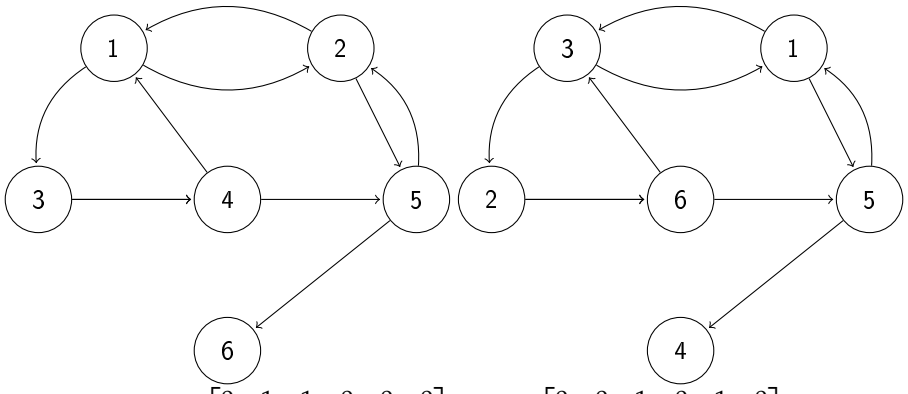
The nodes and edges in the graph shown above can be represented by the following matrix known as an adjacency matrix.

$$A_g = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Each row in the matrix represents a single node, while the columns in each particular row indicate edges being pointed to the neighboring nodes.

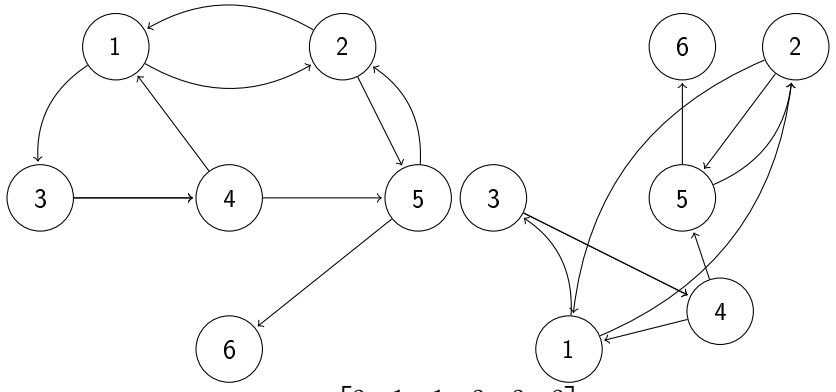
A graph is considered isomorphic if it contains the same number of nodes and edge configuration. This definition leads to 2 main issues. Assuming nodes are arbitrarily labeled, which they typically are, isomorphic graphs can look exactly the same visually but be notated in a completely different adjacency matrix if they are labeled differently. Additionally

isomorphic graphs can look completely different, yet have an identical adjacency matrix(see figures).



$$A_{g1} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad A_{g2} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Even though these graphs are identical, their adjacency matrices are completely different.



$$A_{g1} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure: Exact same adjacency matrix! However, we are unable to visually determine whether they are the same graph.

III. Exhaustive Search Algorithm

For this discussion we are primarily interested in examining two graphs that appear identical, but because of labeling result in completely different adjacency matrices. A preliminary step in determining whether two graphs are isomorphic is merely examining the number of nodes in each graph. If there is a discrepancy in the number of nodes, than no further computation is needed. When it comes to two visually identical graphs with different adjacency matrices, it appears the underlying issue is the labeling of each node. To combat

this issue we can exploit the properties of permutation matrices. A permutation matrix is merely an arbitrary row reversal of the identity matrix. Recall the following.

$$\boldsymbol{I} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Shown above is the identity matrix while shown below is a particular permutation matrix.

$$\boldsymbol{P} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

It should be noted that the number of possible configurations for an n-sized square matrix is n! Another interesting phenomenon behind permutation matrices is that they are orthogonal, which imply the following

$$\boldsymbol{P}^2 = \boldsymbol{P}^{-1} = \boldsymbol{P}^T \tag{1}$$

Some interesting properties of permutation matrices are the following. To reverse the rows of a given matrix we can just treat a particular permutation matrix as an operator. For example if we wanted to reverse row 2 and 3. we can just multiply the given matrix by the operator.

$$\boldsymbol{P}_{R23}\boldsymbol{M} = \boldsymbol{M}_{R23} \tag{2}$$

An example: $\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 \\ 6 & 6 & 6 & 6 & 6 & 6 \end{bmatrix} = \begin{bmatrix} 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 \\ 6 & 6 & 6 & 6 & 6 & 6 \end{bmatrix}$

The most important aspect of the permutation matrix in regards to our discussion is the following relation: If you wanted to switch the labeling of a given adjacency matrix between two nodes: you perform the following operation.

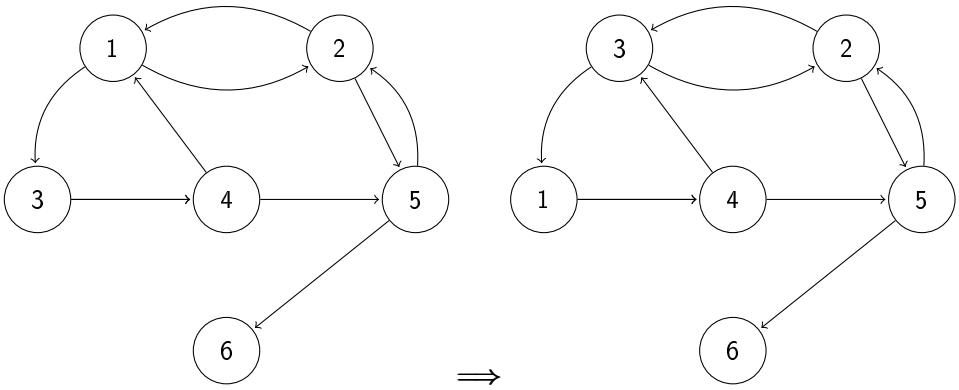
$$\boldsymbol{P}\boldsymbol{A}_{\boldsymbol{g}}\boldsymbol{P}^T = \boldsymbol{A}_{\boldsymbol{giso}} \tag{3}$$

An example can be shown below

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

or ... more explicitly

:



Now supposing we are to construct an algorithm that uses an exhaustive search we would be needing to performing this operation on average $\frac{n!}{2}$ times. We are primarily concerned with running time. Multiplying two matrices together using the Coppersmith Winograd algorithm (wikipedia) can be done with an upper bound running time of $O(n^{2.376})$ we have to do this two times. If we are clever with our indexing, should not spend much time computing the transpose: We can apply Sterlings Approximation to simplify the factorial term into a recognizable exponential.

$$n! \sim \sqrt{(2\pi n)}(\frac{n}{e})^n \tag{4}$$

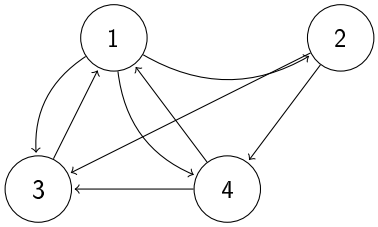
thus expected running time is the following

$$ERT = \frac{1}{2}\sqrt{(2\pi n)}(\frac{n}{e})^n(\frac{1}{2}O(n^{2.376}) + O(1)) = \tag{5}$$

Since our previous running time is approximately exponentially proportional to the number of nodes we are comparing, we desire a more appropriate method to compare more graphs. For example determining isomorphic graphs each of size 1000 nodes will almost certainly not finish computing by end of our lifetime, even if we used all of the computers in the world.

IV. Google's Page Rank Algorithm

Before we can develop a better algorithm to solve the isomorphic graph problem, let us consider one of the most apparent applications of linear algebra. Consider Google's page rank algorithm. The purpose of the algorithm is to determine the most likely desired selection of links related to the search the user provides. The algorithm exploits the use of related links to each page. Consider the following graph taken from the SIAM article titled The $25,000,000,000 Eigenvector.[1]



Related links to each page can be easily quantified and modeled with an adjacency matrix. The key difference between a typical adjacency matrix of a graph and Google's page rank matrix is the ability to weight each of the pages by their apparent reputability. In our case this "reputability" is quantified by the number of outgoing links from the adjacent page.

$$x_j = \sum_{i=1}^{n} \frac{x_i}{n_i} \tag{6}$$

for the previous SIAM graph we would have the following adjacency matrix.

$$A_{SIAMPageRank} = \begin{bmatrix} 0 & 0 & 1 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 1/2 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

If we were to model the graph in our introduction we would get the following

$A_{FirstGraphPageRank} =$

$$\begin{bmatrix} 0 & 1/2 & 0 & 1/2 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 1/2 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 \end{bmatrix}$$

This rule works fine for most graphs. However when we come across nodes with no outward edges we have an issue. We will see that these "sink nodes" we will become problematic in the future. We desire these matrices to be column-stochastic. That is where all of the columns sum to one. This allows matrices to have an eigenvalue of 1. Notice the column of all zeros in the previous matrix. For sink nodes we will merely be placing a 1 in the 6th row. For now we will also be making a slight modification to our current matrix by performing the following operation.

$$M = (1 - m)A + mS \tag{7}$$

where S is merely an n by n where the entries are all $1/n$ and m is a damping factor. In the Google's algorithm they use the value 0.15. solving for the eigenvectors, i.e.

$$x = Mx \tag{8}$$

then

$$x = (1 - m)Ax + ms \tag{9}$$

Where x are the computed "importance scores" of each node. This creates a signature which inspired our page rank algorithm.

V. Polynomial Running Time

The importance score derivation shown above inspired the following operations on the following adjacency matrix. [2]

$$x(t + 1) = \frac{1 - d}{N} \cdot \mathbb{1} + d \cdot W \cdot x(t) \tag{10}$$

where d is the damping factor between 0 and 1. N is the number of nodes. $\mathbb{1}$ is a column of ones.

$$W = (\Theta A)' \tag{11}$$

Where A is the adjacency matrix, and Θ is the diagonal matrix where each (p,p) element is $1/O_p$ or the number of outgoing edges leaving the particular node. Again, for sink nodes we merely have $1/O_p$ equal to 1. $x(t+1)$ can then be computed iteratively and then converge to the following.

$$x^* = \frac{1 - d}{N} \sum_{k=0}^{\infty} d^k W^k \cdot \mathbb{1} \tag{12}$$

Instead of approximating this sum the steady state can actually be compute what the series converges to. Recall

$$\sum_{k=0}^{\infty} ar^k = \frac{a}{1 - r} \tag{13}$$

thus the following operation can be performed.

$$x^* = \frac{1 - d}{N}(I - dW)^{-1}\mathbb{1} \tag{14}$$

Now, when the ordering of x^* is ranked from least to greatest, its signature allows us to quantify the relationship between nodes regardless of labeling! Now we can determine if two graphs with adjacency matrices A_1 and A_2 are equivalent by doing the following.
Step 1) Compute the x_1^* and x_2^* for a particular damping factor d.
Step 2) Construct Matrices V_1 and V_2 where $V_i = [x_i^*, I]$ containing the identity matrix.
Step 3) Sort the rows of V_1 and V_2 and obtain the transformed Identity matrix. It should be clear that these are Permutation matrices. Call them P_1 and P_2 respectively.
Step 4) Compute the supposed permutation matrix by multiplying the following.

$$P_{supposed} = P_2^{-1} P_1 \tag{15}$$

Step 5) compute and test whether the following matrices are equal.

$$A_2 = P_{supposed} A_1 P'_{supposed} \tag{16}$$

If they are equal, then the graphs are isomorphic. If they are not equal, increment the damping factor until the number of trials is greater than the number of nodes in the graph. The graphs will be presumed to be not isomorphic. Disclaimer!!!!! I am leaving a lot of material out of this step by step process. However the heart of the algorithm is here. Please see citation for full details.

VI. Results

To verify this algorithm we can randomly generate an adjacency matrix operate on the given matrix with a permutation matrix as shown in equation and then verify our algorithm with the two isomorphic graphs. The plot shows the running time for the function for N nodes.

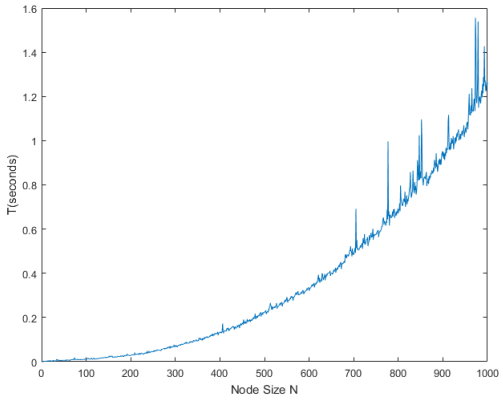


Figure 1: *Here is the polynomial running time for the page rank inspired algorithm. In an attempt to compare the brute force running time, it is realized the algorithm would still take too long to compute or even be shown on the plot. For example 20 nodes has a running time approximately proportional to 20 factorial which is still too large for any reasonable waiting time.*

VII. Conclusion

The Page-Rank inspired algorithm that was just implemented can have huge applications in future computer science research. Machine learning and artificial intelligence are booming fields of study that strive to understand the most intriguing computers ever created; our minds. An important step in learning something is the ability to emulate it. Graph theory will play a crucial role in algorithms that emulate mammalian brains. When we recognize a person's faces we have an ability to instantaneously recognize who they are and their current state of emotion with little effort. Our ability to recognize graphs in a near instantaneous time interval will bring us closer to solving the task of uncovering how our minds work.

References

[1] Tania Leise Kurt Bryan. The 250 billion dollar eigenvector linear algebra behind google. *SAIM*, 48:569–581, 2006.

[2] IEEE Marco Maggini Member IEEE Computer Society Marco Gori, Fellow and Lorenzo Sarti. Exact and approximate graph matching using random walks. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE,*, 27:1100, 2005.