**ABSTRACT**

In this paper, I explain the Graph Isomorphism problem, including some of the research on the problem and related problems. Then I list some of the ideas and algorithms thought of to approach the problem to reduce the problem to polynomial complexity as well as one algorithm tested by writing a program.

**GRAPH ISOMORPHISM**

Graph Isomorphism is a one-to-one and on-to mapping between two graphs so that the properties of each vertex on one graph correspond to the properties of a vertex on the other graph. Formally, Isomorphism of graphs $G$ and $H$ is a bijection between the vertex sets of $G$ and $H$, $f: V(G) \rightarrow V(H)$ such that any two vertices $u$ and $v$ of $G$ are adjacent in $G$ if and only if $f(u)$ and $f(v)$ are adjacent in $H$. (Graph Isomorphism)

Certain pre-conditions can be easily tested to show that two graphs cannot be isomorphic, but those tests cannot be used inversely to demonstrate that two graphs are isomorphic. For example, two graphs cannot be isomorphic if the order and size of the graphs (number of vertices and edges) disagree, but having the same order and size will not prove that they are isomorphic. In a similar way, if a sorted list of degrees of one graph does not match the list of the other graph, they cannot be isomorphic.

One important aspect of graph isomorphism is finding an invariant for graphs that is independent of the way it is drawn. Eigenvectors of adjacency matrices was tested for graph invariants (Czerwinski) but it still may not be completely reliable. (Lysenko, Further thoughts on Graph Isomorphism and some lecture notes)

**IMPORTANCE OF GRAPH ISOMORPHISM**

The problem of determining whether two configurations of graphs are the same come up in fields from chemistry and structural equivalence to computer graphics and electric circuit design as a start. Being able to determine that two graphs of different labeling are in fact the same graph, efficiently, is misleading and far from simple as a problem.
COMPLEXITY OF GRAPH ISOMORPHISM

Determining isomorphism between two graphs is a problem in NP but is not yet known to be in P or NP-Complete (S. Skiena). A naïve solution can run in $O(n!)$ by checking labeling permutations of one of the graphs but because the complexity is unclear, some mathematicians have defined a GI complexity class for problems that similarly reduce to graph isomorphism but have unknown complexity between P and NP.

It is suspected, however, that the problem of Graph Isomorphism is not NP-Complete because the generalized Subgraph Isomorphism Problem is known to be NP-Complete, based on reducing the clique problem to this decision problem, and the polynomial hierarchy is believed to be strict following the arithmetic hierarchy. If Graph Isomorphism is also NP-Complete the hierarchy would collapse. (Subgraph isomorphism problem)

GOING A BIT FURTHER IN-DEPTH

SPECIAL CASES

It has been shown that there are efficient solutions in polynomial time for special cases of the graph isomorphism problem, which hint that this problem could potentially be in P due to the number of constraints limiting it. (Graph Isomorphism)

Special cases include:

- Graphs of bounded genus/degree/Eigen-value or color multiplicity
- Planar graphs (subset of bounded genus [genus 0])
- Trees
- Permutation graphs

MODERATELY EXPONENTIAL BOUND FOR GRAPH ISOMORPHISM

V.N. Zemlyachenko recently showed that there is a solution of $\exp(v^{1-c})$ bound where c is a positive constant, or more specifically in $T(v) < \exp(v^{2/3} + o(1))$ steps. There are various algorithms to solve graph isomorphism that try different methods to try to improve its solution, but a polynomial time algorithm has not yet been proven. (Babai)

PROJECT WORK

RELATED TOPICS

Several related problems were investigated in order to try to better grasp the Graph Isomorphism problem:
GRAPH NON-ISOMORPHISM

One related problem is the complement of Graph Isomorphism, Graph Non-isomorphism. It is not yet known if Non-isomorphism is in NP but rather it is said to be in IP/PSpace (probabilistic polynomial time space), meaning there is a Verifier and an arbitrary Prover for the language/problem that can convince the Verifier of its knowledge without revealing hints to the solution. (Sipser)

GRAPH AUTOMORPHISM

Automorphism is an isomorphic mapping of a graph to itself and a random graph will probably have only a few mappings or just itself (identical graphs to itself), but complete graphs $K_n$ have $n!$ automorphisms. If there is only the single automorphism of itself, it can be said that there are no other isomorphic labelings. The name of Brendan McKay’s graph isomorphism project, Nauty, was derived from automorphisms: “No-AUTomorphisms, Yes?” (S. Skiena)

NAUTY

The fastest isomorphism testing program known today is Brendan McKay’s Nauty program written in C that computes automorphism groups of graphs and digraphs. The basic algorithm is described in his 1981 paper, Practical Graph Isomorphism. (B. McKay)

STRONGLY REGULAR GRAPHS

Spielman proved that strongly regular graphs can be tested quicker in time $n^{\frac{1}{n^3\log n}}$. (Spielman)

PROJECT APPROACH IDEAS

The following were some ideas brainstormed for Graph Isomorphism. For the most part, these ideas failed in one way or another but provides some additional insight into the problem.

BREAKDOWN OF GRAPH INTO TREES

Because trees are a special case of Graph Isomorphism that has been determined to be within polynomial time, we could possibly utilize that fact. If a graph could be consistently divided up into a set of trees, we can verify if two graphs contain the same set of tree subgraphs. The difficulty though is that cycles could not be removed easily in polynomial time and systematically determining the edge to remove is also difficult.
TAKING 3-SAT TO GRAPH ISOMORPHISM

Although Graph Isomorphism is suspected not to be NP-Complete, looking at how the SAT/3-SAT problem could possibly reduce to Graph Isomorphism would be interesting. Finding if there is a set of Boolean verifiers to check two graphs similar to the mapping for the 3-coloring problem was considered but as expected, no good mapping could be found.

TABLE-FILLING ALGORITHM

Creating a table-filling algorithm that can compare to graph tables in P-time in the way that some of the other graph problems reduce. An algorithm that could uniquely take a setup of a graph and map it to a table accordingly could be used to determine graph isomorphism. It would be similar to a hashing function that puts to the same bucket, all isomorphic graphs of different labeling, but a systematic table-filling algorithms did not seem to work well because there is no clear starting point or node.

| AUTOMORPHISM | != 1 |

Determining that the number of automorphisms of a graph is greater than 1 can eliminate many arbitrary graphs as there is only one unique mapping, and aid in a non-isomorphic approach to determine that two graphs are either in the non-isomorphic set or its complementing isomorphic set. The difficulty seemed to be in finding the number of automorphisms, and it is similar to McKay’s Nauty approach.

MORE PROMISING IDEAS

COMPLETE GRAPH REDUCTION

One interesting part about the comparison of two graphs is that highly irregular graphs provide a limit to the starting points from which isomorphism is tested, but at the same time a complete graph is the most regular graph and yet it is easy to test if two graphs are both k-Complete. Additionally, one edge missing from two complete graphs are also easy to test as the complement of the graphs will be graphs of 1 edge. In this sense the case of complete graphs, complete graphs minus one edge, one-edge graphs, and no-edge graphs are four cases that are very regular but as easily tested.

Unfortunately, an inductive method does not seem to work although getting graphs by determining the difference from the graph to a complete graph appears as though it should. On the other hand, complements of graphs can be utilized to select G or G-Complement depending on the number of edges. Because any graph with more than half of the edges of a complete graph has a complement that is less than half, the processing can be essentially cut in half, but it was not found to be as useful in finding an isomorphism.
Finding the complement of a graph can be done in polynomial time in at most $O(n^2)$ by taking an adjacency matrix and flipping each element (removing an existing edge or adding an edge). Using this, one algorithm devised was to take two graphs $G_1$ and $G_2$ and test its isomorphism through removing edges. The following must be true if and only if $G_1$ and $G_2$ are isomorphic:

$$G_1 - m_1 \text{ iso } G_2 - m_2 \leftrightarrow m_1 = m_2 \leftrightarrow \overline{G_1 - m_1} \text{ iso } \overline{G_2 - m_2}$$

Algorithm:

Let $H_1 = G_1$ or $G_1$-Complement, whichever has less edges

Let $H_2 = G_2$ or $G_2$-Complement, whichever has less edges

Remove an edge from $H_1$ and $H_2$ until $H_1$ and $H_2$ are empty, in which case the two are isomorphic, or is determined to not be isomorphic by the difference in degree-sequence or some other property, then trace back and try again.

The problem is that the algorithm can end up processing all edges the number of times that the number of edges exists, so it is not within polynomial time. Some ideas here might be useful however.

A “most-unique” vertex can be determined in the sense that a starting point for isomorphic testing is found through selecting one of the nodes with a degree rarest to both graphs. Selecting a ‘unique’ node can be done simply by looking through all nodes sorted by degree but other properties could be used as a metric.

Algorithm:

Choose a ‘most-unique’ vertex $v$ from $G_1$ and $u$ from $G_2$ and remove those vertices from the respective graphs. Compare neighbors of $v$ to neighbors of $u$, if it is consistent then select a most-unique vertex from the neighbors of each, else re-select $u$.

This is an interesting method because the less ‘unique’ vertices there are, the more regular the graph is and the closer a graph is to being complete, so the probability of selecting a vertex $u$ in $G_2$ that has a valid isomorphic labeling increases. Because of this, I was not able to fully evaluate the complexity of this algorithm but it could show better results.

In representing graphs as adjacency matrices, it is possible to present it as a set of binary operations in terms of group theory in the following way:
Define binary operation * to be an XOR of two adjacency matrices.

- Associativity: For all adjacency matrices a, b, and c, (a * b) * c = a * (b * c)
- Identity: Defined to be the inverse of an n x n identity matrix (i.e. all 1's except the diagonal 0's)
- Inverse: The complement of a graph in an adjacency matrix is essentially an inverse in which two matrices A and !A become the Identity through A * !A and !A * A.

We can describe the graph isomorphism problem to be one in which the adjacency matrix A of G₁ must operate with some arrangement of the matrix B of G₂-Complement such that A * B is the Identity.

Looking closely at the ways in which matrix B can be arranged or permuted, it was found that for each pair of columns swapped in labeling, the corresponding rows must also be swapped to get back the same graph. However, this only means that there are n! permutations and so it goes back to the original naïve solution problem.

**SORTED ADJACENCY MATRICES**

Following the previous approach, one possibility was to sort the adjacency matrix in some way that would take polynomial time and using this method, two sorted adjacency matrices for two graphs could be determined for isomorphism. It was determined that sorting by columns or sorting by rows in which each column or row was taken as a binary number, would in fact converge, but a potential problem arose in using the above method of swapping corresponding rows for each pair of columns.

A program was written in C# to test random permutations of matrices and check for isomorphism. However, only about 50% of the matrices seem to result in the same configuration for two isomorphic graphs. It was not determined what subset of isomorphic graphs worked in this sorting algorithm, but it may show interesting results.

**CONCLUSION**

While this project was not able to find a polynomial time algorithm for Graph Isomorphism, it studied and found some interesting results in other parts of the problem. These results might prove useful in future analyses of Graph Isomorphism.

**REFERENCES**


