Notes on Graph Isomorphism

Eric Bach
University of Wisconsin
May 2003

The problem is to take two undirected graphs, and determine if one is a relabeling of the other.

A. Why Do We Care?

Surprisingly this was not a major topic in graph theory from the beginning. (None of my graph theory textbooks mention it beyond giving the definition.) But now we can find some reasons for its prominence.

1. Chemistry.

   For hydrocarbons (C and H only) the number of each type of atom is not enough to determine a molecule's structure. The smallest example is propane (C_4H_{10}) which can have two shapes. If we ignore the node labels, these shapes give us two different trees.

   Stripping away all the chemistry and associated constraints, we are left with the isomorphism problem.

2. Geometry

   We may ask, "how many essentially different shapes are there?" Depending on what you think is essential, there are different kinds of geometry: topology, differential geometry, algebraic geometry, etc.

   The grand question is to classify these different shapes, and to do this you want to have ways to tell if shapes are the same or not.

   If we insist on algorithms to tell shapes apart, this can only work in low dimensions. Markov proved that the equivalence of 4-manifolds (given as cell complexes?) is undecidable.

   Graphs form a nice special case to think about.

3. Computational Complexity

   Graph isomorphism is in NP (guess the relabeling). It is not known whether it is in P or NP-complete.

   Ladner proved that if P $\neq$ NP, there are problems in NP that are neither in P nor NP-complete. It is possible that GI is such a problem.

4. The "Canon Effect"

   It is an important problem because lots of other people have studied it.

B. How Might We Solve It?

Here are some general approaches that one might think of. They are all in some sense algebraic, reflecting the idea that a major goal of topology is to reduce geometric questions to algebra.

1. Topology

A graph G is a kind of 1-dimensional manifold with singularities. (1-d manifolds without singularities are not very interesting: there are essentially only 2 of them).

Here are some plausible invariants we could use to try to tell graphs apart.

1a. Embedding dimension

If edge crossings are not allowed, then not all graphs fit into R^2. Those that do are called planar. There is a linear-time algorithm to test planarity (and do the embedding?).

Every graph fits into R^3 (use the twisted cubic given parametrically by t, t^2, t^3).

1b. Homology/Cohomology.

These are vector spaces that can be defined for any (?) space, and provide topological information. Since we are only concerned with graphs we will not define them in full generality.

$H^0(G)$ is the space of locally constant functions on G. Its dimension is the number of connected components. The connected components are easy to find using depth-first search, but it is clear that there are non-isomorphic graphs with the same number of connected components.

$H_1(G)$ is the set of possible "current flows" on G. (Imagine that the edges of G are made of some kind of conducting material, and these are soldered together to make the graph. DC current can flow around any loops that are formed in this graph.) Its determination is a classical problem, analyzed by Kirchhoff in his study of electrical circuits.

The underlying geometric idea is that we can add cycles in a graph. (Example: two boxes joined by an edge. We can go round one box, the other, or outside. However, going outside is the same as the sum of the first two, i.e. we are allowed to cancel an edge if used in opposite direction.)

Suppose that G is connected. To make a basis for the currents, we first find a spanning tree ($n-1$ edges). Adding one more edge to our tree gives us a unique cycle on which we can place any current we want, and these cycles are obviously independent. So

$$h_1 = \dim H_1 \geq m - (n-1) = m - n + 1.$$

On the other hand any current is determined precisely by its values across the non-tree edges, so h_1 is exactly $m - n + 1$.

By considering trees alone, we see that h_1 is too coarse to give an isomorphism test.

All the higher dimensional spaces are trivial, since graphs are essentially 1-dimensional objects.

1c. Fundamental group.

This is defined as follows. Pick a vertex v of the graph. A loop is a path that starts at v and returns to there. If we think of these loops as given by a thread, there is an obvious equivalence relation: one loop can be slid, without cutting, into another. The fundamental group $\pi_1(G)$ is the set of equivalent loops. One can show it doesn't depend on v, if G is connected.

For a connected graph, π_1 is a free group with $m + n - 1$ generators (Fulton, Algebraic Topology, p. 202). Note that $m + n - 1 = h_1$, so this gives very little information concerning isomorphism.

Note: Free groups are defined as follows. We are given generators $a_1, \ldots, a_r$. We form words of length ≥ 0 using the symbols a_i, a_i^{-1}, where $i = 1, \ldots, r$. Multiplication is done by concatenation, and we are allowed to cancel inverses ($a_i a_i^{-1} = a_i^{-1} a_i = 1$).

1d. Embedding on surfaces.

The orientable compact smooth surfaces (2-d manifolds) are characterized topologically by the genus, a nonnegative integer. ($g = 0$ specifies a sphere, $g = 1$ a torus, and so on. Essentially, g counts the number of holes.) [There is a similar classification for non-orientable surfaces, like the projective plane, Klein bottle, etc.]

We say that a graph has genus g if it can be embedded without crossings into a genus g surface but not into a genus $g - 1$ surface. So planar = genus 1, toroidal = genus 2, etc. For example, the "three utilities" graph is not planar, but it is genus 1. [The essential reason it is not planar is that it has too many edges. From the Euler polyhedron formula $V - E + F = 2$ one can conclude that any planar graph has $m \leq 3n - 6$.]

Unless P = NP, there is no polynomial time algorithm to find g [Thomassen, J. Algorithms, 1989].

There is a polynomial-time algorithm to decide, for fixed genus, if two graphs are isomorphic [Filotti and Mayer, 1980 IEEE FOCS]. Unfortunately the time required depends severely upon g.

2. Group Theory

Related to isomorphism is the following question: given a graph G, how many automorphisms does it have?

Example: a circle graph with n vertices has as automorphisms D_n, the dihedral group.

Counting automorphisms is at least as hard as testing isomorphism.

Proof: Suppose we can evaluate $|\mathrm{Aut}(G)|$. To test if G is isomorphic to H, form $K = G \cup H$. (This is the disjoint union: a copy of G, a copy of H, and nothing else.) Then

$$|\mathrm{Aut}(Z)| \geq |\mathrm{Aut}(X)||\mathrm{Aut}(Y)|$$

with equality iff the two graphs are not isomorphic.

From this proof, one can see that there is a constant $c > 0$ with the property that it is hard to approximate the size of the automorphism group to relative error less than c. Solving the equation $2(1-c) = (1+c)^2$ I get $c = 0.23606....$

Of course, counting the elements of a group might be hard. As it turns out, we only need to determine a set of generators for $\mathrm{Aut}(Z)$. For, if G and H (connected) are isomorphic, at least one of these generators must interchange G with H, and this is easy to test. Of course, if G and H are not isomorphic, there is no such generator.

We can reduce computing the automorphism group (and thereby, determining isomorphism) to the calculation of the intersection of two permutation groups.

What do we mean by permutation groups? These are groups of permutations of $\{1, ..., N\}$, specified by giving generators in cycle notation.

Example: S_N, the set of all permutations, is generated by $(1, 2)$ and $(1, 2, ..., N)$. [Proof: Consider an array of distinct elements, i.e. $[X_1, \ldots, X_N]$. For each generator there is an operation we can do to our array, i.e.

$$[X_1, X_2, \ldots, X_N]^{(1,2)} = [X_2, X_1, \ldots, X_N]$$

and

$$[X_1, X_2, \ldots, X_N]^{(1,\ldots,N)} = [X_N, X_1, \ldots, X_{N-1}].$$

This is a group action in the sense that $(([...]^\sigma)^\tau = [...]^{\sigma\tau}$. Combining these two operations we can rearrange the array any way we want. Why? We can sort! In particular it suffices to interchange adjacent pairs, and to do that we can shift until our pair occupies the first two cells, apply (1,2) to swap it, and shift back. This gives $N!$ different rearrangements, so our operations must generate that many permutations.]

Reducing automorphism computation to group intersection.

Let G be a graph with n vertices. We will let P denote the set of possible edges $\{i, j\}$. Note that a graph is determined by the partition of P into E (the edges) and $\bar{E}$ (the non-edges).

Observe that P has cardinality $N = \binom{n}{2}$. We will number the elements of P so that

$$E = \{1, \ldots, m\} \qquad \bar{E} = \{m+1, \ldots, N\}.$$

Suppose π is a permutation of the vertices. We will let S_V denote the group of permutations on P that are induced by such a vertex permutation, i.e.

$$\{i, j\}^\pi = \{i^\pi, j^\pi\}.$$

We can get generators for S_V by taking generators for S_n and letting them act on P.

Now, if π is an automorphism of G, its action on P must preserve E (and therefore $\bar{E}$). Let us denote by by S_E the group of permutations of P that preserve E, and define $S_{\bar{E}}$ similarly. We have

$$(1,2) \qquad (1,\ldots,m)$$

and

$$(m+1,m+2) \qquad (m+1,\ldots,N)$$

as generators for S_E and $S_{\bar{E}}$, respectively.

Then,

$$\mathrm{Aut}(G) = S_V \cap (S_E \times S_{\bar{E}})\,.$$

(Note that it is easy to get generators for the direct product once we have generators for its factors.)

Mathon (cited in Hoffmann) gives a bunch of other reductions of this type. See also K. Booth, SIAM J. Computing, 1978.

3. Linear Algebra

Let A_G denote the adjacency matrix of the graph G. Then G and H are isomorphic iff there is a permutation matrix P such that

$$P^{-1}A_GP = A_H.$$

Since P must be invertible this implies that the two adjacency matrices are similar. So another quick and dirty isomorphism test is to see if the matrices have the same eigenvalues, minimal polynomial, etc. Unfortunately this is not definitive: there are examples of non-isomorphic graphs with the same spectrum.

The "generic" polynomial has distinct zeroes, and there is a polynomial time algorithm to determine if two graphs with distinct eigenvalues are isomorphic. [Babai et al., Proc. 1982 ACM STOC.] The difficult case for this algorithm is when the eigenvalues are degenerate, with high multiplicity.

The Laplacian matrix is, by definition $A_G - D$, where D is a diagonal matrix whose entries form the degree sequence of G. (The degree of a vertex is the number of neighbors it has.)

Examples are known of non-isomorphic graph pairs whose Laplacian spectra are identical.

I don't know if there is a polynomial time algorithm for graphs with bounded Laplacian-eigenvalue multiplicity. [Jin-Yi Cai thinks there is one.]

4. Heuristics

There are a bunch of ad hoc tricks based on the idea of taking each vertex and looking around, and combining this local information to get what we hope is a unique "name" for a type of graph. Some of these methods are described by Fortin. I can't say much about them, except to point out that if any of them could be proved to work, GI would be in P.

C. What is its Complexity?

1. Positive results.

The trivial algorithm involves looking at $n!$ permutations, which is $\exp(n \log n + O(n))$ by Stirling's formula.

Two graphs can be tested for isomorphism in $\exp(n^{1/2} + o(1))$ steps, where n is the number of vertices. [Claimed at the end of E. Luks, JCSS, v. 25, pp. 42-65, 1982. I assume someone has checked this by now.]

Several results show that graph isomorphism is easy in an average sense. Here's an example: Use the G_{np} model to make two random graphs with n vertices. (That is, we include each possible edge with probability p, independently.) A necessary condition for these graphs to be the same is for the edge counts to be the same, and these numbers are is binomially distributed. From the central limit theorem, we believe this probability to be about

$$\Theta\left(\frac{1}{npq}\right)$$

This means that an overwhelming fraction of "random" graph pairs (in this model) will be easy to distinguish, provided that p and q are not too close to 1.

There are, I assume, more sophisticated results of this type. See Köbler and Hoffmann for references.

2. Negative results.

The informed speculation on this matter suggests that GI is unlikely to be NP-complete. (Note that to settle this question either way would be entirely consistent with current knowledge.)

One of the reasons for this is that $\overline{\text{GI}}$ – that is the collection of pairs of graphs that are not isomorphic – belongs to a complexity class that is a "randomized" version of NP. If one accepts that plausible idea that there are efficient high-quality pseudo-random number generators, then this class should collapse to NP. We would then get a co-NP complete problem (namely, $\overline{\text{GI}}$) in NP, making the entire polynomial hierarchy collapse.

Independently of any hypotheses, it has been proved that if GI is NP-complete, the polynomial hierarchy collapses to level 2.

Here are two interesting constructions.

1. An Isomorphism Guessing Game (Goldreich).

   We have two players, Arthur and Merlin. Arthur can do polynomially complicated computations and flip coins. Merlin can do anything. Arthur wants to determine whether G and H are isomorphic, and enlists Merlin's help.

   Arthur chooses K to be one of G, H, and selects a randomly permuted version of it, and sends it to Merlin with a challenge to tell him which graph it came from. Merlin can answer the challenge with certainty if and only if G and H are not isomorphic. Otherwise, he has a 50-50 chance of being right, assuming that Arthur used a perfect coin to pick K.

   It's important that Merlin not be allowed access to Arthur's choice process, or the game becomes trivial. Also, it is not obvious what this has to do with NP.

2. BNP (Schöning?)

   BNP is a class of decision problems. We will think of these as sets comprising the "yes" instances to the decision problem. Formally, we make the following definition. A decision problem L belongs to BNP if there is another decision problem M in NP, such that:

$$x \in L \implies \text{ for most } y\ [(x,y) \in M]$$
$$x \notin L \implies \text{for few } y[(x,y) \notin M]$$

   Implicitly we assume that the predicate for M ignores all but the first $p(|x|)$ bits of y, where p is some fixed polynomial. We can interpret "most" as "at least $1/2$" and "few" as "at most $1/4$" but the precise constants don't matter as long as they provide a separation.

   Each BNP definition naturally provides a protocol. To determine if $x \in L$ or not, Arthur flips $p(|x|)$ coins to determine y, and then asks Merlin for a proof that (x,y) belongs to M. Repeating this enough times, Arthur gains confidence about whether x has the property L or not. In this game, however, Arthur's moves can be made public. Some people use the name AM (for Arthur-Merlin) instead of BNP.

   Our goal is to show that the complement of GI belongs to BNP.

   For each pair of graphs G_1, G_2, let X to be the pairs (G, π) such that G (labelled graph) is isomorphic to one (possibly both) of the G_i, and $\pi \in \mathrm{Aut}(G)$. If G_1 and G_2 both have n vertices then

$$|X| = \begin{cases} n!, & \text{if the } G_i \text{ are isomorphic;} \\ 2n!, & \text{if not.} \end{cases}$$

   Note that there is a 2:1 gap between the sizes in the two cases.. Also, X belongs to NP.

   Let $Y = X \times X$ be the cartesian product of X with itself. This increases the size ratio to 4:1.

We now think of the elements of Y as balls that can be thrown into bins, where the number of bins is as large as Y can possibly be. Arthur will send Merlin a specification of how the balls are thrown, and then challenge Merlin to find a ball that goes into a particular bin. If Y is large, Merlin can do this most of the time, and if Y is small, Merlin cannot. With repetition Arthur can be pretty sure which case applies. The non-obvious thing is that Arthur can specify his throws with very little information.

We now need to think about this quantitatively. Imagine that we have N balls, of which W are white and $N - W$ are red. (Concretely the white balls represent bona fide elements of Y.) We throw them into N bins. Intuitively, the probability that the first bin has a white ball is small when W is small, and large when W is large.

In the usual model (i.i.d. uniform placement of balls), this probability is exactly $1 - \left(\frac{N-1}{N}\right)^W$. We will use a different model: each ball is distributed uniformly, with placement of pairs independent. (No other assumptions.) By inclusion-exclusion we get

$$\frac{W}{N} - \frac{\binom{W}{2}}{N^2} \leq \Pr[\text{ a white ball hits the first bin}] \leq \frac{W}{N}$$

If $W = o(N)$, the upper bound is $o(1)$. Also, if $W = \alpha N$, the lower bound is at least $\alpha - \alpha^2/2$.

For the sake of exposition, we will make two simplifications. First, we pretend that N is prime. Also, we assume that we have an efficient way to code elements of Y as residues mod N. [This needs to be looked into more carefully. I think it is sufficient to choose N to be a prime near $4(n!)^2$. I have not thought about the details of the coding.]

If N is prime, we can compactly specify a mechanism for pairwise independent throws, as follows. Note that the integers mod N will form a finite field. Choose two elements a, b from this finite field. Then the element x is thrown to location $ax + b \bmod N$. It is easy to show that if $x \neq x'$ then the locations for x and x' are independent.

After N is agreed upon by both parties, Arthur does the following. He chooses (at random) a and b and then challenges Merlin to provide a $y \in Y$ that gets put in some particular bin (say, the first one). If G and H are not isomorphic, Merlin can succeed most of the time, and if they are isomorphic, Merlin cannot do this.

D. References

S. Fortin, The Graph Isomorphism Problem, Tech. Report TR 96-20, Dept. Computing Science, University of Alberta, 1996.

P. J. Giblin, Graphs, Surfaces, and Homology: An Introduction to Algebraic Topology, Chapman and Hall 1977.

C. Hoffmann, Group-Theoretic Algorithms and Graph Isomorphism, Springer-Verlag Lecture Notes in CS 136.

J. Köbler et al., Graph Isomorphism Problem: Its Structural Complexity, Birkhauser [Wendt QA267.7 K63 1993]

Chapter 18 of U. Schöning and R. Prium, Gems of Theoretical Computer Science, Springer-Verlag 1998.