Graph Isomorphism

Franklin van Nes

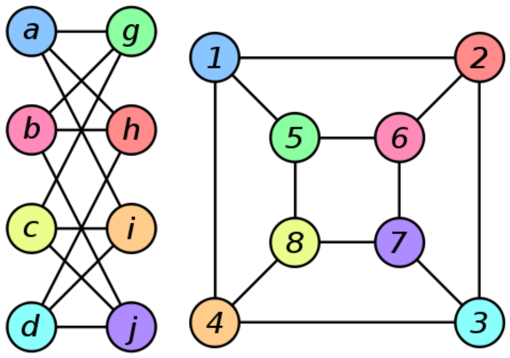
Tuesday 12th November, 2017

Graph Isomorphism is most intuitively understood through its greek etymology: *Iso*, meaning same, and *morphism*, meaning shape; *Graph Isomorhpism* $\rightarrow$ graphs that share the same shape. In more detail, two finite graphs are isomorphic if they share the same number of vertices connected in the same way.

Formally, by an *isomorphism* we mean from two graphs G_1 to G_2 we have a one-to-one mapping, $f : G_1.V \rightarrow G_2.V$, from $G_1.V$ onto $G_2.V$ so that vertices u_1 and v_1 are adjacent in G_1 if and only if the vertices $f(u_1)$ and $f(v_1)$ are adjacent in G_2. If an isomorphism exists between G_1 and G_2, we say they are *isomorphic* [3].

Graph isomorphism is also an equivalence relation which allows us to say that G_1 and G_2 are isomorhpic if they are equal, and they are equivalent if they are isomorphic. Figure 1 shows how two isomorphic graphs can look very different while still maintaining the same vertex adjacency.

Figure 1: Two isomorphic graphs. The colors indicate matching vertices, even if their labels do not match [6]



There also exists the *Subgraph Isomorphism* problems: given G_1 and G_2, determine if G_2 is a subgraph of G_1. However, Subgraph Isomorphism, which is definitively NP-complete [4], is a generalization of Graph Isomorphism, so we won't consider this in the same problem set as Graph Isomorphism.

It should be noted that determining Graph Isomorphism is complex in its own right. In fact, determining graph isomorphism falls into its own category of problem complexity, called *Graph Isomorphism Complete*. According to Lubiw, it has yet to fall into a typical classification, and is neither P nor NP-complete [5]. There exists no known P algorithm, yet graph isomorphism has not been shown to be NP-complete. This means that either a P algorithm must exist for graph isomorphism but is undiscovered, graph isomorhpism is a problem outside of P and NP, or, as Schning argues, Graph isomorphism problems are in the low hierarchy of NP, which "does not equal NP unless the polynomial hierarchy collapses to the second level" [7]. Deeper explanation of this exceeds the reaches of this paper.

While there is really just one representative problem of this class - determine if two graphs are isomorphic - there are a large variety of consequential applications to solving this problem which I will mention instead.

For instance, database searches that use graphs instead of keywords. A great example of this is the biochemical search engine described by Bonnici et al.; Given unknown "biological networks at the molecular, protein, or species level", in a graph representation, how can we determine if it subject already exists in current records? We can perform a graph isomorphism against the database of graphs of existing subjects to determine if our unknown subject has previously been discovered and other relevant information. Because this is such an expensive process, it would make sense to filter results on other criteria before determining graph isomorphism, or to use graph isomorphism algorithms to eliminate matches as quickly as possible to reduce the search space [2]. Other applications exist in automata (comparing languages), social networks, chemical structure analysis, and image processing.

The best currently accepted theoretical algorithm belongs to Lazlo Babai [1]. Babai's algorithm currently achieves Quasipolynomial time, represented as $2^{\lg n^{O(1)}}$. The $O(1)$ prevents the algorithm reducing to polynomial time. The algorithm relies on canonical labelling of the graphs. A basic algorithm that uses canonical labelling for graph isomorphism would be as follows.

Given two finite graphs with unknown isomorphism G_1 and G_2 where $|G_1.V| = |G_2.V|$, assign a count label $v.c$ that is the outdegree for every vertex in G_1 and G_2. First, check that the frequency of all outdegrees are the same for both graphs (if it isn't, they are not isomorphic). Then, starting with the outdegree of lowest frequency select vertices $v_1 \in G_1.V$ and $v_2 \in G_2.V$ with the same outdegrees. Perform a simultaneous BFS on G_1 and G_2 starting from v_1 and v_2, comparing the outdegrees of all of the neighoring vertices. If ever the neghboring outdegrees of v_1 and v_2 don't match, try a different pair of start vertices v_1 and v_2 until you have exhausted all vertices of that outdegree (at which point we can say that G_1 and G_2 are not isomorphic). If all outdegrees match for a certain combination v_1, v_2, continue this process for the remaining unvisitied vertices until both graphs are fully visited or G_1 and G_2 are found to not be isomorphic. If you can visit all vertices in G_1 and G_2 and match all $v.c$ in G_1 and G_2, we can consider them isomorphic.

Often, graphs that have more meaningful labels on their vertices can be more easily determined as isomorphic or not because more unique information can be used to more quickly filter out incorrect adjacent vertices.

References

[1] L. Babai. Graph isomorphism in quasipolynomial time. *CoRR*, abs/1512.03547, 2015.

[2] V. Bonnici, R. Giugno, A. Pulvirenti, D. Shasha, and A. Ferro. A subgraph isomorphism algorithm and its application to biochemical data. *BMC Bioinformatics*, 14(7):S13, Apr 2013.

[3] G. Chartrand. *Isomorphic Graphs*, pages 32–40. Dover, 1985.

[4] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[5] A. Lubiw. Some np-complete problems similar to graph isomorphism. In *SIAM Journal on Computing*, pages 11–22. Society for Industrial and Applied Mathematics, February 1981.

[6] C. Martin. Graph isomorphism - wikipedia.

[7] U. Schning. A low and a high hierarchy within np. *Journal of Computer and System Sciences*, 27(1):14 – 28, 1983.