A Graph Isomorphism Algorithm

Glen Woodworth

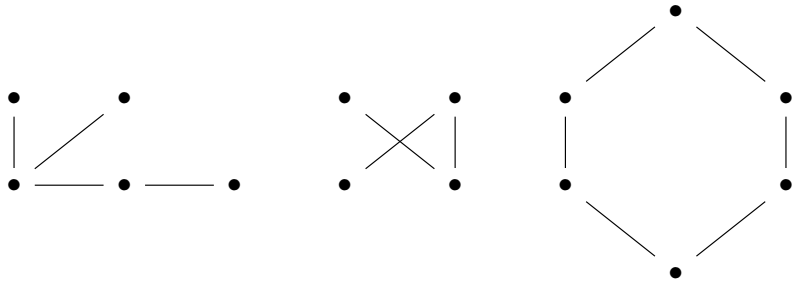
Department of Mathematics and Statistics
University of Alaska Fairbanks

March 2, 2023

Graphs

Definition

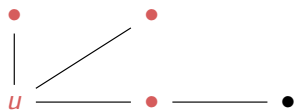
An undirected, simple graph $G = \{V, E\}$ is a set of vertices V, and a set of edges, E, where E is a set of unordered pairs of vertices from V such that for all $u \in V$, $\{u, u\} \notin E$.



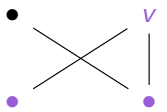
Neighbors

Definition

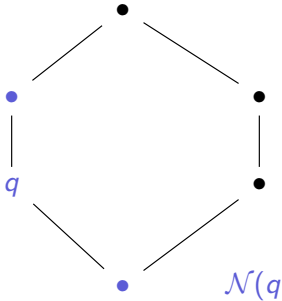
For a graph $G = \{V, E\}$, if $\{u, v\} \in E$, we say u and v are *adjacent* or *neighbors*. The *neighborhood* of $v \in G$, denoted $\mathcal{N}(v)$ is the set of all vertices $u \in G$ such that v and u are neighbors.



$\mathcal{N}(u)$



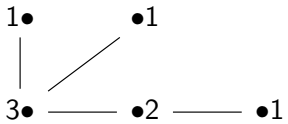
$\mathcal{N}(v)$

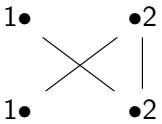


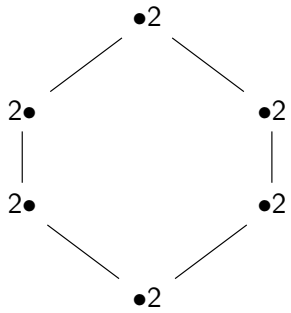
$\mathcal{N}(q)$

Labelling

We want to talk about properties of specific vertices, so we give a label to each vertex. The vertices in the graphs below are labelled using the cardinality of their neighborhoods, also called the *degree* of a vertex.

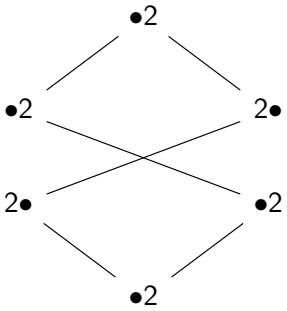






Similar graphs

Rearrnging the vertices does not change a graph. We want to know when two graphs are the same, even though they may not look like it. The graphs below are the same as the graphs we have been looking at.



Similar or the same?

Before formalizing what we mean by two graphs $G = \{V, E\}$ and $G' = \{V', E, \}$ being the same, here are some obvious properties G and G' must share if they are the same, such as

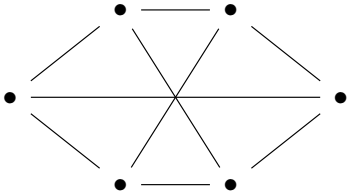
- the same number of vertices, $|V| = |V'|$
- the same number of edges, $|E| = |E'|$
- vertices with mathcing degrees, i.e., G has two vertices of degree 3 iff G' does too
- similar neighborhoods, i.e., $v \in V$ has a neighborhood containing 2 vertices of degree 2 iff there exists $v' \in V$ such that $\mathcal{N}(v')$ has the same property

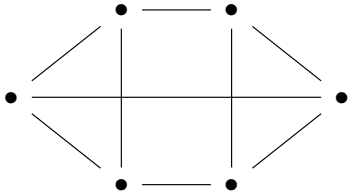
Remark

These are necessary conditions that follow if G and G' are the same. Are they sufficient?

Similar but not the same

Those conditions are *not* sufficient for us to call two graphs the same. Here is an example:





- same number of vertices and edges (6 vertices, 9 edges each)✓
- every vertex has degree 3 in each graph ✓
- every neighborhood looks the same ✓
- *not* the same ×

Why not?

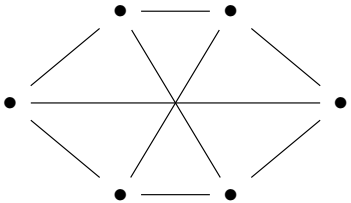
In order for two graphs to be the same, virtually *every* graph property we are interested in must be shared between the two graphs. To distunguish these graphs, we can use the following definition.

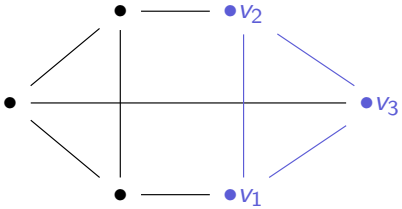
Definition

A *cycle* in a graph $G = \{V, E\}$ is an ordered list of vertices $c = (v_1, v_2, \ldots, v_n)$ where $v_i \in V$ are all unique, $\{v_1, v_n\} \in E$, and $\{v_i, v_{i+1}\} \in E$ for $1 \leq i \leq n$. Since c contains n vertices, we call c a n cycle.

Different cycles

Another graph property then is the number and type of cylces in the graph. The graph on the right has a 3 cycle (highlighted) but the graph on the left doesn't :(





Graph Isomorphism

Two graphs being the same is a high bar. The following definition formalizes exactly 'two graphs are the same' means.

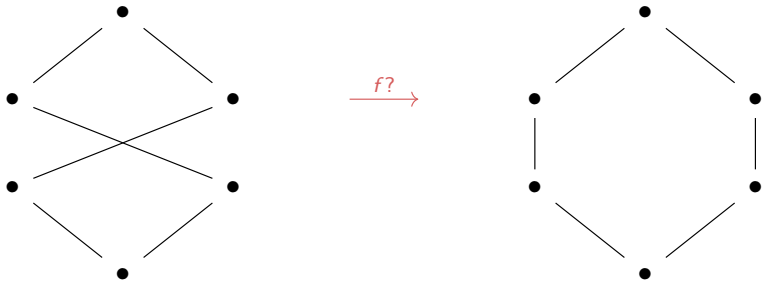
Definition

Two graphs, $G = \{V, E\}$ and $G' = \{V', E'\}$, are *isomorphic*, denoted $G \cong G'$, if there exists a bijection $f : V \to V'$ such that for every $u, v \in V$, $\{u, v\} \in E$ if and only if $\{f(u), f(v)\} \in E'$. If such a function f exists, we call f a *graph isomorphism* from G to G'.

A hard problem (maybe?)

Problem

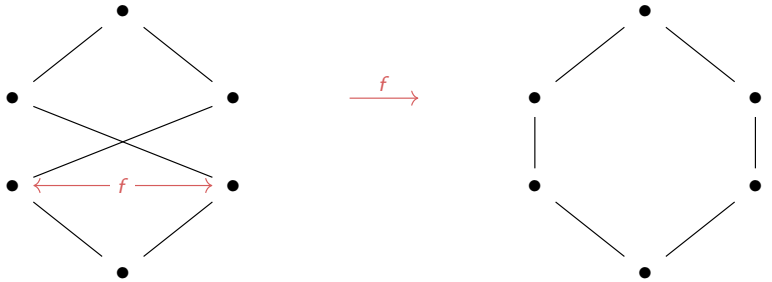
Given two graphs $G = \{V, E\}$ and $G' = \{V', E'\}$, the *graph isomorphism problem* (GI) is to determine if there exists a graph isomorphism from G to G'.



The solution

Solution

Given two graphs G and G' solving GI requires proving the existence of a graph isomorphism from G to G' (perhaps by producing such a funciton) or showing no such function exists.



Complexity of GI

The complexity class of solving GI for any two finite simple graphs is an open problem (also known as the graph isomorphism disease).

- since it is straightforward to check if some function f is a graph isomorphism between two graphs or not, GI is in NP (the class of problems for which a possible solution can be checked nondeterministically in polynomial time)
- lowest established upper bound is

$$\mathcal{O}(2^{(\log n)^c})$$

where n is the number of vertices and $c \in \mathbb{N}$ (fixed), called *quasi-polynomial* time (Babai 2018)

A bad method

Most efficient software for solving GI does *not* use this approach becasue it is not suitable for collections of graphs or finding graphs in a database (Mckay 2013)

Solution method

input: $G = \{V, E\}$, $G' = \{V', E'\}$
iterate: check functions $f : V \to V'$
output: true or false "G is isomorphic to G'"

Canonical form

Rather than check for isomorphism dirctly, most efficent GI solvers use an approach called "canonical labelling," meaning the vertex set is written in a canonical form.

Canonical form

Given a graph $G = \{V, E\}$, a *canonical form* of G is a partial ordering of V such that for all $u, v \in V$, if u and v are incomparable, then there exists a graph automorphism moving u to v. (Weisfeler, Leman 1968)

Canonical form example

Weisfeler-Leman (WL) canonical form (1968)

input: $G = \{V, E\}$
intialize: label vertices by their degree (the initial classes)
iterate: produce v_i, a characteristic vector for each vertex, then reclassify based on these vectors
output: a canonical form of G (i.e., a nice partition of V)

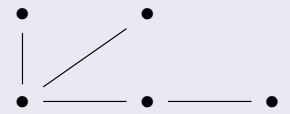
- Each characteristic vector v_i has the following form: $v_i = (l, v_{i1}, v_{i2}, \dots)$ where l is the class of the vertex and v_{ij} is the number of neighbors vertex i has in class j.
- WL canonical form terminates when the classes stop dividing (i.e., classes stabilize)

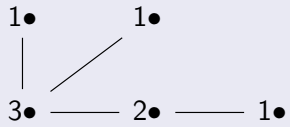
Canonical form example

Given the graph G below, we follow WL canonical form until the classes stabilize.

initialize

G

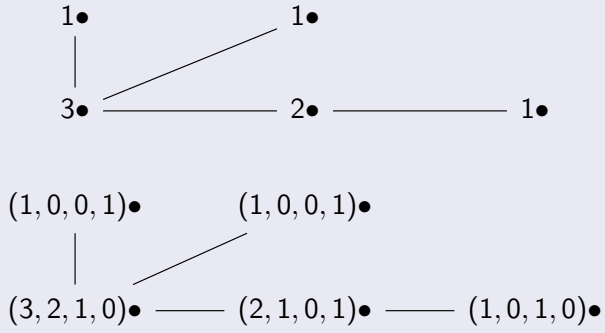




Canonical form example

We now generate the characteristic vector for each vertex.

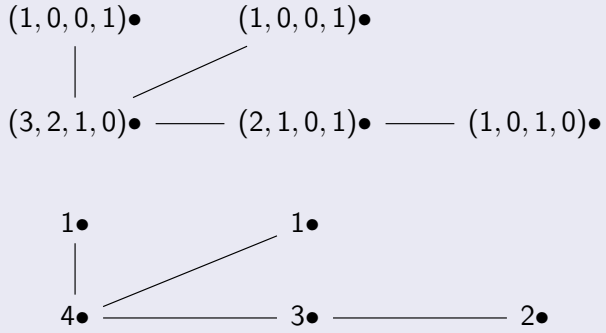
iterate (1)



Canonical form example

We now reclassify the vertices by lexicographically ordering the vetors.

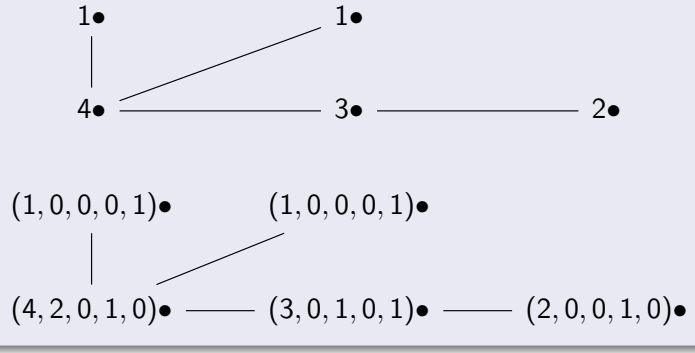
iterate (1)



Canonical form example

Since the classes divided, we must iterate again.

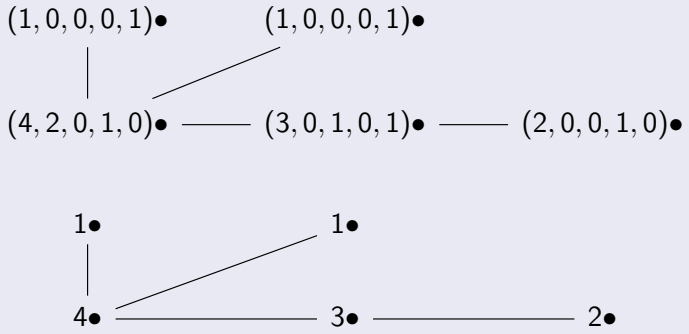
iterate(2)



Canonical form example

Reclassify.

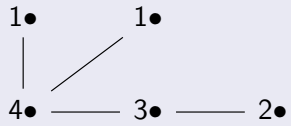
iterate(2)



Canonical form example

The classes did not divide this time, so the algorithm terminates.

output



There are two incomparable vertices (the two that are in class 1). There is an obvious automorphism that swaps these two. Hence, this is a canonical form.

Another bad method

Notice that WL canonical form could be used to determine that two graphs are *not* isomorphic.

Solution method

input: $G = \{V, E\}$, $G' = \{V', E'\}$
step: produce canonical forms $\tilde{V}$ and $\tilde{V}'$ using WL canonical form
output: if $\tilde{V} \neq \tilde{V}'$, then output G is not isomorphic to G'.
else, output: nothing

Regardless of implementation, this will be prohibitvely slow. No one does this.

WL Algorithm

Here is a sophisticated way to take advantage of the canonical labelling of WL, called the 1 dimensional Weisfeler Leman Graph Isomorphism test (1-D WL).

Algorithm 1 One iteration of the 1-dim. Weisfeiler-Lehman test of graph isomorphism

1: Multiset-label determination
- For $i = 0$, set $M_i(v) := l_0(v) = \ell(v)$. [2]
- For $i > 0$, assign a multiset-label $M_i(v)$ to each node v in G and G' which consists of the multiset $\{l_{i-1}(u) | u \in \mathcal{N}(v)\}$.

2: Sorting each multiset
- Sort elements in $M_i(v)$ in ascending order and concatenate them into a string $s_i(v)$.
- Add $l_{i-1}(v)$ as a prefix to $s_i(v)$ and call the resulting string $s_i(v)$.

3: Label compression
- Sort all of the strings $s_i(v)$ for all v from G and G' in ascending order.
- Map each string $s_i(v)$ to a new compressed label, using a function $f : \Sigma^* \to \Sigma$ such that $f(s_i(v)) = f(s_i(w))$ if and only if $s_i(v) = s_i(w)$.

4: Relabeling
- Set $l_i(v) := f(s_i(v))$ for all nodes in G and G'.

Algorithm 1 runtime

Let h be the number of iterations and m be number of different degree vertices in the two input graphs(??).

- step 1: labelling is $\mathcal{O}(m)$
- step 2: sorting is $\mathcal{O}(m)$
- Step 3: label compression is $\mathcal{O}(m)$
- step 4: relabelling is $\mathcal{O}(m)$
- total runtime: $\mathcal{O}(hm)$.

Another WL GI test

The following algorithm makes use of a hash function and the WL subtree kernel, which counts original and compressed labels in the input graphs.

Algorithm 2 One iteration of the Weisfeiler-Lehman subtree kernel computation on N graphs

1: Multiset-label determination
- Assign a multiset-label $M_i(v)$ to each node v in G which consists of the multiset $\{l_{i-1}(u) | u \in \mathcal{N}(v)\}$.

2: Sorting each multiset
- Sort elements in $M_i(v)$ in ascending order and concatenate them into a string $s_i(v)$.
- Add $l_{i-1}(v)$ as a prefix to $s_i(v)$.

3: Label compression
- Map each string $s_i(v)$ to a compressed label using a hash function $f : \Sigma^* \to \Sigma$ such that $f(s_i(v)) = f(s_i(w))$ if and only if $s_i(v) = s_i(w)$.

4: Relabeling
- Set $l_i(v) := f(s_i(v))$ for all nodes in G.

Algorithm 2 runtime

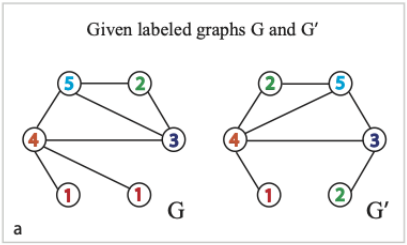
Algorithm 2 can be run on N graphs, pairwise comparing each graph.

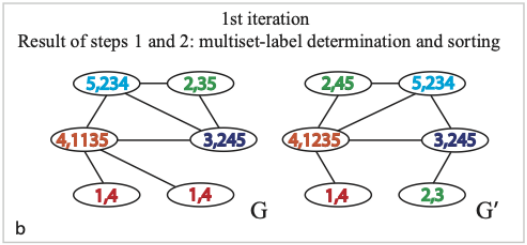
Theorem

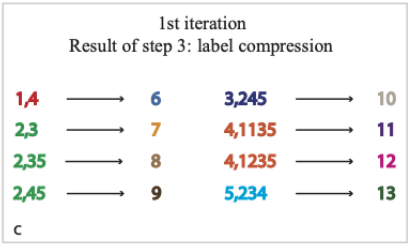
For N graphs, Algorithm 2 with h iterations on all pairs of these graphs can be computed in $\mathcal{O}(Nhm + N^2hn)$.

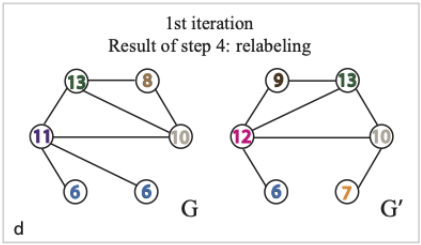
Note: Nhm dominates runtime.

Algorithm 2 example









1-D WL subtree test example

End of the 1st iteration

Feature vector representations of G and G′

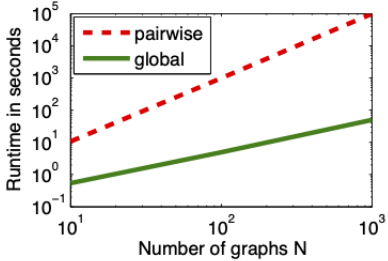
$$\varphi^{(1)}_{WLsubtree}(G) = (2, 1, 1, 1, 1, 2, 0, 1, 0, 1, 1, 0, 1)$$

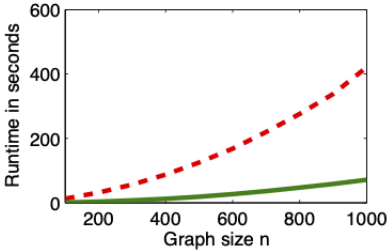
$$\varphi^{(1)}_{WLsubtree}(G') = (\underbrace{1, 2, 1, 1, 1}_{\text{Counts of original node labels}}, \underbrace{1, 1, 0, 1, 1, 0, 1, 1}_{\text{Counts of compressed node labels}})$$

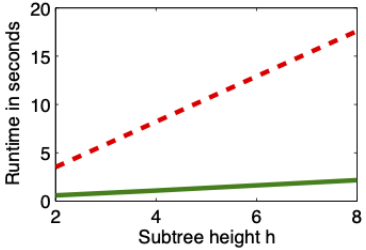
$$k^{(1)}_{WLsubtree}(G,G') = \langle \varphi^{(1)}_{WLsubtree}(G), \varphi^{(1)}_{WLsubtree}(G') \rangle = 11.$$

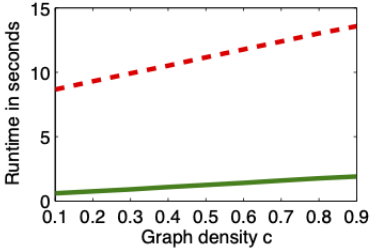
e

Algorithm 2 experiments









How good is 1-D WL in general?

It has been shown to be an efficient graph isomorphism test for almost all graphs (Babai, Kucera 1979). There are higher dimensional variants (which I do not understand) that can classify a larger set of graphs.

-
- can be implemented to run in
$$\mathcal{O}(n^{k+1} \log n)$$
where k is the dimension (k=1 for us) and n is the number of vertices of the input graphs. (Immmerman, Lander 1990)
- determining which graphs are classified by any dimension is usually highly nontrivial (see Kiefer (2020))
- 1-D WL cannot distinguish regular graphs
- other algorithms are better (broader classification abilities and faster) but ideas from WL show up in most of these

What's important for GI?

The powerful solvers are able to say whether or not two graphs are isomorphic or not using canonical labellings and producing the automorphism group of a graph (the set of all isomorphisms from a graph to itself).

- these are complicated
- Babai's isomorphism test is also complicated (he invented new algebra to produce it)
- but they both make use of canonical labelling in some way

The practical approach

Rather than check for isomorphism dirctly, most efficent GI solvers use some form of canonical labelling along with some other fast algorithm.(Mckay 2013). This is called *individualization refinement*.

individualization refinement

input: $G = \{V, E\}$, $G' = \{V', E'\}$
iterate: relabel vertices to produce canonical labelling for G
compile: produce automorphism groups of G
output: true or false "G is isomorphic to G'" via $G \cong G'$ if and only if $G' \in Aut(G)$

Applied GI solvers

Althought the complexity of GI is an open problem, in reality, powerful (open source) software exists for classifying graphs efficently (i.e., they can efficiently proudce the automorphism group containing the set of all graph isomorphisms from a graph to itself).

- Weisfeiler-Leman (color refinement)
- nauty
- saucy
- bliss
- Traces