Graph Isomporphism and a Polynomial Test for Trivalent Graphs

Ekaterina Nepomnyashchaya

Abstract

This paper introduces the graph automorphism and isomorphism problems and covers the theory and algorithms needed to understand the trivalent case from *Isomorphism of Graphs of Bounded Valence Can Be Tested in Polynomial Time* by Eugene Luks.

Contents

1 Introduction

Two graphs X and Y are isomorphic if there exists an adjacency preserving bijective map between their two vertex sets $V(X)$ and $V(Y)$. An isomorphism is such a bijection. Formally, we seek a bijection $\sigma : V(X) \to V(Y)$ such that any two vertices $u, v \in V(X)$ are adjacent if and only if $\sigma(u), \sigma(v) \in V(Y)$ are adjacent.

A related question asks about bijections from G to itself. Such a bijection is called an automorphism.

The study of graph isomorphism and automorphism algorithms is both of practical and theoretical interest. For example, we may be interested in an algorithm that generates a list of graphs without repetition. This requires testing for equivalence and therefore an isomorphism test. On the other hand, automorphisms help us understand symmetry and structure. Automorphisms of molecules can be useful in predicting and explaining their chemical properties.

Of theoretical interest is the computational complexity of testing for isomorphism and finding automorphisms. While there are practical algorithms for graph isomorphism and automorphism that run in polynomial time for most inputs, the known algorithms for general graphs are exponential in the worst case. And while there exist polynomial algorithms for specific classes of graphs such as planar or bounded degree graphs studied in this paper, it is unknown if a polynomial algorithm on general graphs exists.

This paper will study the related problems of graph automorphism and isomorphism, specifically for bounded degree graphs. We base our discussion of isomorphism for bounded degree graphs on *Isomorphism of Graphs of Bounded Valence Can Be Tested in Polynomial Time* published by Eugene Luks in 1981.

First we cover basic notation, then the relationship between automorphism, isomorphism, and other related problems. Next we build up the required group theory of systems of imprimitivity, cover necessary group theory algorithms, before going into details of isomorphism of trivalent graphs.

1.1 Notation

We assume the reader is familiar with undergraduate level group theory and has worked with graphs before. For those new to group theory, see the appendix for a list necessary theorems and check out any modern algebra book such as Algebra by Michael Artin for definitions and proofs. For everyone else, a quick run through notation is below.

1.2 Graphs

We will work with simple graphs where for a graph X the vertex set is $V(X)$ and $E(X)$ is the set of edges. An undirected edge between the vertices v and w is denoted $\{v, w\}$. Two vertices are adjacent if there is an edge between them.

We will also consider vertex-labeled graphs. Formally, a vertex-labeled graph (X, λ) consists of a graph X and a mapping λ from the vertices $V(X)$ onto a set of labels $L = \{l_1, l_2, ..l_k\}$. There are also edge-labeled graphs, but the term labeled graph usually refers to a vertex-labeled graph.

The definitions of automorphism and isomorphism naturally extend to labeled graphs. Two labeled graphs (X, λ_x) and (Y, λ_y) are isomorphic if there is an isomorphism between X and Y that preserves the label of each vertex. That is, there is a bijection $\sigma : V(X) \to V(Y)$ such that for all $u \in V(X)$ we have $\lambda_x(u) = \lambda_y(\sigma(u))$, and as before $u, v \in V(X)$ are adjacent iff $\sigma(u), \sigma(v) \in V(Y)$ are also.

1.3 Groups

The set of automorphisms of a graph X forms a group we call $Aut(X)$. We can examine this group's actions on the vertices, edges, or sets of vertices of X. Therefore below we develop some notation for groups and group actions.

Let G be a group with identity element e. For $S \subseteq G$ we let $\langle S \rangle$ denote the subgroup generated by S. If H is a subgroup of G we write $H \leq G$. If H is a proper subgroup, that is $H \neq G$, we write $H < G$. A left coset is the subset $aH = \{ah | h \in H\}$, were $a \in G$. Analogously, a right coset is $Ha = \{ha | h \in H\}$. We have $[G : H] = |G|/|H|$ is the index of H in G.

We say a group G acts on the set A if a homomorphism $\phi : G \to Sym(A)$ is given. For $g \in G$ and $x \in A$ both $g(x)$ and gx are used to represent the action. If we also have $h \in G$, we have $gh(x) = g(h(x))$.[1] For $\sigma \in G$, and $B \subseteq A$ we write $\sigma B = \sigma(B) = \{\sigma b | b \in B\}$.

Given a group G acting on A with $x \in A$, the pointwise stabilizer of x in G is the subgroup $G_x = \{\sigma \in G | \sigma(x) = x\}$. For $B \subseteq A$, the pointwise stabilizer is $G_B = \{\sigma \in G | \forall b \in B, \sigma(b) = b\}$. A setwise stabilizer of B in G is the subgroup $G_{\{B\}} = \{\sigma \in G | \sigma B = B\}$.

2 Related Problems

While the isomorphism and automorphism problems ask different questions, they are actually computationally equivalent. That is, if there exists a polynomial algorithm to one, there is also a polynomial algorithm to the another.

In fact, all of the following problems are polynomially equivalent.

Problem 2.1 (Existence of Graph Isomorphism).
Given two graphs, decide whether they are isomorphic.

Problem 2.2 (Existence of Isomorphism of Labeled Graphs).
Given two labeled graphs (X, λ) and (Y, μ), decide if they are isomorphic.

[1]Unfortunately in many older sources such as *The Theory of Groups* by Marshall Hall, *Group-Theoretic Algorithms and Graph Isomorphism* by Chrisoph Hoffmann and Luks's work, this notation is different. In modern group theory multiplication is from left to right. In the older work multiplication is right to left, and gh acting on a was $h(g(a))$.

Problem 2.3 (Graph Isomorphism).
Given two graphs, decide whether they are isomorphic, and if so, construct an isomorphism.

Problem 2.4 (Graph Automorphism).
Given a graph X, determine a generating set for $Aut(X)$.

Problem 2.5 (Order of automorphism group).
Given a graph X, determine the size of $Aut(X)$.

Problem 2.6 (Number of Isomorphisms).
Given two graphs, determine the number of isomorphisms between them.

Below I sketch the proofs of polynomial equivalence for the above problems. Many of the ideas were originally published by Rudolf Mathon in 1979's *A note on the graph isomorphism counting problem.* Please note that ideas in proposition 2.9 and corollaries 2.10-2.12 are relevant beyond the discussion of polynomial equivalence. We refer back to these concepts when discussing efficient group algorithms.

Proposition 2.7. *If there is a polynomial time algorithm to determine if two graphs are isomorphic [Problem 2.1], then there is also one to determine if two labeled graphs are isomorphic [Problem 2.2].*

Proof. Given two labeled graphs (X, λ) and (Y, μ) we create two unlabeled graphs X' and Y' such that (X, λ) and (Y, μ) are isomorphic iff X' and Y' are. To ensure that vertices of X can only map to vertices of Y with the same label, we attach different subgraphs to vertices with different labels.

Suppose X and Y both have n vertices. If not, there is no isomorphism between X and Y and therefore between (X, λ) and (Y, μ). Let L be the set of labels in λ and μ. We have $L = \{l_1, l_2, ..., l_k\}$. Since λ has at most n distinct labels, and two isomorphic graphs must have the same label counts, if $|L| > n$, there is no isomorphism between (X, λ) and (Y, μ).

We obtain X' from (X, λ) by attaching to each vertex a complete graph of size dependent on that vertex's label. For a vertex v_i with label l_m, we create a complete graph on vertices $\{v_{i,1}, ..., v_{i,n+m}\}$ and attach it with an edge $\{v_i, v_{1,i}\}$. Do the same with all vertices of Y to create Y'.

Notice that two the vertices get attached to complete graphs of the same size iff they have the same label. Furthermore, since all the new attached complete graphs are of size at least $n + 1$, and original graphs X and Y were both on n vertices, the newly attached complete graph clusters representing the labels can only map to other clusters representing the same label. Thus, an isomorphism of X' and Y' exists iff there is an isomorphism between X and Y that preserves the labels. That is, (X, λ) and (Y, μ) are isomorphic iff X' and Y' are.

Finally, since $|L| \leq n$, the new graphs X' and Y' each have at most $2n^2$ more vertices and $2n^3 + n$ more edges than X and Y. That is, we can create X' and Y' in a polynomial time. Therefore given solution for Problem 2.1, we solve Problem 2.2 by creating X' and Y' and running the algorithm for Problem 2.1

on the two new graphs. Since X' and Y' are of size polynomial in the size of X and Y, if the algorithm for Problem 2.1 is polynomial, so is our solution to Problem 2.2. $\square$

Proposition 2.8. *If there is a polynomial time algorithm to determine if two labeled graphs are isomorphic [Problem 2.2], there is also one to construct an isomorphism between two regular graphs, if one exists [Problem 2.3].*

Proof. Suppose we are given two graphs X and Y. Notice that an algorithm for Problem 2.2 allows us to see if they are isomorphic simply by testing isomorphism of (X, λ_0) and (Y, μ_0) where all labels in λ_0 and μ_0 are set to zero. Suppose that an isomorphism exists, we proceed by finding a match for each vertex one at a time.

Let $V(X) = \{v_1, ..., v_n\}$, and $V(Y) = \{w_1, ..., w_n\}$. We start by finding a match for the first vertex v_1.

$$\text{Let } \lambda_1(v) = \begin{cases} 1 & \text{if } v = v_1 \\ 0 & \text{otherwise} \end{cases} \quad \text{and } \mu_1^j(w) = \begin{cases} 1 & \text{if } w = w_j \\ 0 & \text{otherwise} \end{cases}$$

There exists an isomorphism ϕ with $\phi(v_1) = w_j$ iff there exists and isomorphism between (X, λ_1) and (Y, μ_1^j). And since there is at least one isomorphism between X and Y, there is certainly some vertex w_{j_1} such that exists an isomorphism ϕ with $\phi(v_1) = w_{j_1}$. We find this w_{j_1} simply by testing every vertex in $V(Y)$.

We proceed by finding a match for each next vertex. Suppose we already matched $v_1, ..., v_i$ to $w_{j_1}, ..., w_{j_i}$ respectively, and there exists an isomorphism that maps these pairs together. Next we find a match for v_{i+1}.

$$\text{Let } \lambda_{i+1}(v) = \begin{cases} 1 & \text{if } v = v_1 \\ \vdots & \\ i & \text{if } v = v_i \\ i+1 & \text{if } v = v_{i+1} \\ 0 & \text{otherwise} \end{cases} \quad \text{and } \mu_{i+1}^j(w) = \begin{cases} 1 & \text{if } w = w_{j_1} \\ \vdots & \\ i & \text{if } w = w_{j_i} \\ i+1 & \text{if } w = w_j \\ 0 & \text{otherwise} \end{cases}$$

Since we know that there exists an isomorphism matching $v_1, ..., v_i$ to $w_{j_1}, ..., w_{j_i}$, there must be some $w_{j_{i+1}}$ such that with $w_j = w_{j_{i+1}}$ the labeled graphs (X, λ_{i+1}) and (Y, μ_{i+1}) are isomorphic. We find such a $w_{j_{i+1}}$ by testing all vertices not already matched. Once such a $w_{j_{i+1}}$ is found, we know there exists an isomorphism between X and Y that maps $v_1, ..., v_{i+1}$ to $w_{j_1}, ..., w_{j_{i+1}}$. We proceed until all vertices are matched.

To time this procedure we count the number of calls to the algorithm for Problem 2.2. We need to match the n vertices of $V(X)$. Each time we test out at most n vertices of $V(Y)$, so we have at most n^2 calls to the algorithm of Problem 2.2. Therefore a polynomial solution to Problem 2.2 leads to a polynomial solution to Problem 2.3. $\square$

To continue we must prove a group theory lemma. The construction and proposition below will come up again when discussing efficient representations of permutation groups in section 3.

Suppose G is a subgroup of $Sym(A)$, where $A = \{a_1, ..., a_n\}$. Let G_i denote the subgroup of G that pointwise fixes the points $\{a_1, ..., a_i\}$, with $G_0 = G$. We get a subgroup chain called stabilizer chain with

$$\{e\} = G_n = G_{n-1} \subseteq ... \subseteq G_1 \subseteq G_0 = G.$$

Let U_i be the set of left coset representatives of G_{i+1} in G_i. We have the following lemma.

Proposition 2.9. *Each $g \in G_k$, with $0 \leq k \leq n-2$, can be uniquely represented as a product $g = u_k u_{k+1} ... u_{n-2}$ with each $u_i \in U_i$.*

Proof. We use induction on $n - k$.

Let $n - k = 2$. Since $G_{n-1} = \{e\}$, we have $U_{n-2} = G_{n-2}$, and we're done.

Now suppose that for $n-k = r$, every $g \in G_{n-r}$ can be uniquely represented by the product $u_{n-r}u_{n-r+1}...u_{n-2}$. If $r < n$, let $h \in G_{n-r-1}$. There is a unique coset representative $u_{n-r-1} \in U_{n-r-1}$ such that $u_{n-r-1}^{-1}h \in G_{n-r}$. But by the inductive hypothesis we can uniquely express every element of G_{n-r} as a product $u_{n-r}u_{n-r+1}...u_{n-2}$. Therefore we can write $h = u_{n-r-1}u_{n-r}...u_{n-2}$. $\square$

Corollary 2.10. *The order of G_k is $\displaystyle\prod_{r=2}^{n-k} |U_{n-r}|$.*

Corollary 2.11. *The set $\displaystyle\bigcup_{r=2}^{n-k} U_{n-r}$ generates G_k.*

Corollary 2.12. *Let G be a subgroup of $Sym(A)$ with $|A| = n$. There exists a generating set for G of size $O(n^2)$.*

Proof. For $0 \leq i \leq n-3$, we have G_{i+1} is a pointwise stabilizer of point a_{i+1} in G_i. Therefore, by the Lagrange orbit stabilizer theorem, $|U_i| = |G_i|/|G_{i+1}|$ is equal to the size of the orbit of a_{i+1} in G_i, which cannot be no more than n.

Since there are $n-2$ sets U_i, we get $\displaystyle|\bigcup_{r=2}^{n} U_{n-r}| \leq \sum_{r=2}^{n} |U_{n-r}| \leq \sum_{r=2}^{n} n < n^2$. $\square$

Proposition 2.13. *If there is a polynomial time algorithm to construct an isomorphism between two graphs [Problem 2.3], then there is a polynomial time algorithm to determine the generating set for the automorphism group of a graph [Problem 2.4].*

Proof. We determine the generating set of $G = Aut(X)$ by computing the left coset representatives U_i for the stabilizer chain of G. By Corollary 2.11 and 2.12 the union $\bigcup_{r=2}^{n} U_{n-r}$ is a $O(n^2)$ size generating set.

To find coset representatives of G_i in G_{i-1}, we need to understand when two cosets are G_{i-1} are distinct. Remember that G_i fixes $\{v_1, \dots, v_i\}$ and G_{i-1} fixes $\{v_1, \dots, v_{i-1}\}$. Below we argue that $\sigma, \pi \in G_{i-1}$ are in the same coset iff $\sigma(v_i) = \pi(v_i)$.

Suppose that $\sigma, \pi \in G_{i-1}$ are in the same coset. That is, $\sigma = ug$ and $\pi = uh$ for some $g, h \in G_i$ and $u \in U_{i-1}$. But remember that G_i fixes v_i, so $g(v_i) = h(v_i) = v_i$. Which means that $\sigma(v_i) = u(v_i) = \pi(v_i)$.

Now suppose that $\sigma(v_i) = \pi(v_i)$. Let $g = \sigma^{-1}\pi$. Notice that $g(v_i) = v_i$. Furthermore, since G_{i-1} fixes all vertices $\{v_1, \dots, v_{i-1}\}$, we have that $\sigma(v_j) = \pi(v_j) = v_j$ for all $j \leq i-1$. So $g = \sigma^{-1}\pi$ also fixes $\{v_1, \dots, v_{i-1}\}$. We get that g fixes $\{v_1, \dots, v_i\}$ and therefore in $\sigma^{-1}\pi \in G_i$. That is, σ and π are in the same coset of G_{i-1}.

Therefore, to find the coset representatives of G_i in G_{i-1}, we need to find the automorphisms u that fix $\{v_1, \dots, v_{i-1}\}$ and map v_i to v_j for $j \geq i$. Since the identity element is a coset representative mapping v_i to v_i, we only concern ourselves with $j > i$.

Consider two labeled graphs (X, λ_i) and (X, μ_i^j) with

$$\lambda_i(v) = \begin{cases} 1 & \text{if } v = v_1 \\ \vdots & \\ i-1 & \text{if } v = v_{i-1} \\ i & \text{if } v = v_i \\ 0 & \text{otherwise} \end{cases} \quad \text{and } \mu_i^j(v) = \begin{cases} 1 & \text{if } v = v_1 \\ \vdots & \\ i-1 & \text{if } v = v_{i-1} \\ i & \text{if } v = v_j \\ 0 & \text{otherwise .} \end{cases}$$

An isomorphism from (X, λ_i) to (X, μ_i^j) exists iff there is an element of U_{i-1} which maps v_i to v_j. Therefore, to find the coset representatives in U_{i-1} we take the identity element and an isomorphism from (X, λ_i) to (X, μ_i^j) for all $j > i$ where one exists.

The only remaining issue is that we are given an algorithm to find an isomorphism, if one exists, of normal graphs. However, we need to get isomorphisms of labeled graphs. To deal, in polynomial time, we transform the labeled graphs (X, λ_i) and (X, μ_i^j) into equivalent unlabeled graphs Y_i and Z_i^j using the technique presented in the proof of Lemma 2.7. If there is an isomorphism from Y_i to Z_i^j, there is an isomorphism from (X, λ_i) to (X, μ_i^j) recoverable in polynomial time. If an isomorphism exists, it is in fact an automorphism and coset representative in U_{i-1}.

To summarize, we calculate the generating set of $G = Aut(X)$ by calculating the coset representatives for the stabilizer chain of G. There are $n-1$ sets of coset representatives. For each set U_{i-1} we create the labled graph (X, λ_i) and $n-i$ labled graphs (X, μ_i^j) for $j > i$. We transform the labled graphs into unlabled graphs Y_i and Z_i^j in polynomial time. We then run a polynomial algorithm to compute an isomorphism, if one exists, between Y_i and Z_i^j for each $j > i$. That's $n-i$ runs of the algorithm for Problem 2.3 for each $1 \leq i \leq n-1$. And for each pair where an isomorphism exists, we take polynomial time to recover the coset representative. All in all, given a polynomial algorithm to compute an isomorphism between two graphs, there is a polynomial algorithm to compute the generators for the automorphism group of a graph. □

Proposition 2.14. *If there is a polynomial time algorithm that returns the generating set for $Aut(X)$ [Problem 2.4], there is one to determine $|Aut(X)|$ [Problem 2.5].*

See section 3 on efficient algorithms for permutation groups for a polynomial time algorithm that determines the size of a group given its generators.

Proposition 2.15. *If there is a polynomial time algorithm to determine the size of a graph's automorphism group [Problem 2.5], there is also a polynomial time algorithm to determine if two graphs are isomorphic [Problem 2.1].*

Proof. Suppose we are given two graphs X and Y. If they are of different size, they cannot be isomorphic. If they are both connected and of size n, let Z be the disjoint union of the graphs X and Y.

Notice that given permutations $\sigma_x \in Aut(X)$ and $\sigma_y \in Aut(Y)$, the permutation

$$\sigma_z(v) = \begin{cases} \sigma_x(v) & \text{if } v \in V(X) \\ \sigma_y(v) & \text{if } v \in V(Y) \end{cases}$$

is an automorphism of Z. Therefore the direct product $Aut(X) \times Aut(Y)$ is a subgroup of $Aut(Z)$.

If X and Y are not isomorphic, since they are both connected, no automorphism of Z can map a vertex of $V(X)$ to a vertex of $V(Y)$. That is, $Aut(Z) = Aut(X) \times Aut(Y)$.

Alternatively, suppose there is an isomorphism $\pi : V(X) \to V(Y)$. Then the permutation

$$\sigma_z(v) = \begin{cases} \pi(v) & \text{if } v \in V(X) \\ \pi^{-1}(v) & \text{if } v \in V(Y) \end{cases}$$

is an automorphism of Z not in $Aut(X) \times Aut(Y)$.

We get that connected X and Y are isomorphic iff $|Aut(Z)| > |Aut(X)||Aut(Y)|$.

Now suppose X and Y are not connected. Let X' be the graph X with an additional vertex adjacent to every other vertex. Similarly create Y'. Notice that X' and Y' are isomorphic iff X and Y are.

If we have a polynomial time algorithm to determine the size of a graph's automorphism group, we now have one to see if two graphs are isomorphic. We first check if the two graphs are connected. If they are, we simply compute $|Aut(X)|$, $|Aut(Y)|$, and $|Aut(Z)|$ where Z has $2n$ vertices. If they are not, we create X', Y' and $Z' = X' \cup Y'$, and compute $|Aut(X')|$, $|Aut(Y')|$, and $|Aut(Z')|$. □

Corollary 2.16. *Problems 2.1 through 2.5 are polynomial time equivalent.*

To show that Problem 2.6 is also polynomial time equivalent to problems 2.1 through 2.5, we need the following proposition.

Proposition 2.17. *Let X and Y be two isomorphic graphs of size n. Then the set of isomorphisms from X to Y is a left coset of the automorphism group $Aut(X)$ in S_n.*

Proof. Let $Iso(X, Y)$ be the non empty set of isomorphisms from X to Y containing the isomorphism π. Let α be an arbitrary automorphism in $Aut(X)$. An edge $\{u, v\} \in E_x$ iff the edge $\{\alpha(u), \alpha(v)\} \in E_x$ iff $\{\pi\alpha(u), \pi\alpha(v)\} \in E_y$. So $\pi\alpha$ is also an isomorphism from X to Y. We get that the coset $\pi Aut(X) \subseteq Iso(X, Y)$ and $|Aut(X)| \leq |Iso(X, Y)|$.

If $|Iso(X, Y)| = 1$, then $Aut(X)$ consists only of the identity element, and we are done. Otherwise suppose $|Iso(X, Y)| > 1$. Let $\sigma \in Iso(X, Y)$ be an arbitrary element distinct from π. Since π is an isomorphism from X to Y, an edge $\{u, v\} \in E_x$ iff the edge $\{\pi(u), \pi(v)\} \in E_y$. Furthermore, since σ^{-1} is an isomorphism from Y to X, the edge $\{\pi(u), \pi(v)\} \in E_y$ iff $\{\sigma^{-1}\pi(u), \sigma^{-1}\pi(v)\} \in E_x$. We get that an edge $\{u, v\} \in E_x$ iff the edge $\{\sigma^{-1}\pi(u), \sigma^{-1}\pi(v)\}$ is too. That is, $\sigma^{-1}\pi \in Aut(X)$. So σ and π must be in the same coset of $Aut(X)$ and $Iso(X, Y) \subseteq \pi Aut(X)$.

Since $\pi Aut(X) \subseteq Iso(X, Y)$ and $Iso(X, Y) \subseteq \pi Aut(X)$, we get $\pi Aut(X) = Iso(X, Y)$. That is, the set of isomorphisms from X to Y is a left coset of $Aut(X)$. □

Corollary 2.18. *If two graphs X and Y are isomorphic, then $|Iso(X, Y)| = |Aut(X)|$.*

Corollary 2.19. *Given a polynomial time algorithm to determine if two graphs are isomorphic [Problems 2.1] and a polynomial time algorithm to determine the size of a graph's automorphism group [Problem 2.5]. Then there is a polynomial algorithm that returns the number of isomorphisms between two graphs [Problem 2.6].*

Corollary 2.20. *Problems 2.1 through 2.6 are polynomial time equivalent.*

Proof. Clearly a polynomial time algorithm that returns the number of isomoprhisms between two graphs [Problem 2.6] can be used to output if two graphs are isomorphic [Problem 2.1].

Conversely, from Corollary 2.16 we know that a polynomial algorithm for any of the problems 2.1 through 2.5 will give us polynomial algorithms for problems 2.1 and 2.5. And by Corollary 2.19 that gives us a polynomial algorithm for Problem 2.6. □

We have proved the polynomial time equivalence of problems 2.1 through 2.6. While this is an important result, it is important to remember that these problems and our proofs assumed general connected graphs. Unfortunately, the only polynomial algorithms currently known are for special cases of graphs. In those cases our proofs may not hold. For example converting a planar labeled graph to an unlabeled graph using our construction results in a non-planar graph. Similarly the graph resulting from converting an bounded degree labeled graph does not necessarily have bounded degree.

Next we consider efficient algorithms for permutation groups. Many approaches to graph isomorphism, including the bounded degree case covered in this paper, are based in group theory. Therefore we need an understanding of which group theory constructs can be computed in polynomial time.

3 Efficient Permutation Group Algorithms

In many group algorithms we are dealing with a permutation group $G \leq Sym(A)$. For example, in the automorphism problem we are interested in $Aut(X) \leq Sym(V_x)$. Let $|A| = n$. Since $|Sym(A)| = n!$, just listing the elements of the group G may not be polynomial time in $|A|$. We need a more efficient way of storing, and looking up elements in a group.

One way to efficiently represent a group G is in terms of a small set of generators. We will present algorithms for finding a convenient set of generators that can be used to efficiently determine the order G and decide if a permutation is in G.

We begin with some results that build intuition for why representing groups using generators may be a good idea.

Theorem 3.1. *Let G be a finite group with $|G| = m$. There exists a subset K of G, where $|K| \leq |\log_2(m)|$, and K generates G.*

Proof. We give a very simple algorithm for finding such a subset K and argue that $m = |G| \geq 2^{|K|}$. K is built up by adding a single element at a time. We start with $K_0 = \pi_1$, where π_1 is an arbitrary element of G, and $G_0 = \langle K_0 \rangle$. Notice that $|G_0| = 2 \geq 2^1 = 2^{|K_0|}$.

Assume K_i generates the subgroup G_i, with $|G_i| \geq 2^{|K_i|}$. Either $G = G_i$ and we are done, or there is an element π_i of G not in G_i. Let $K_{i+1} = K_i \cap \{\pi_i\}$, and $G_{i=1} = \langle K_{i+1} \rangle$. We have that G_i is a proper subgroup of G_{i+1}. Therefore, $|G_{i+1}|/|G_i| > 1$. But by the Lagrange Theorem, $|G_{i+1}|/|G_i|$ is an integer. So, $|G_{i+1}|/|G_i| \geq 2$. We get that $|G_{i+1}| \geq 2|G_i| \geq 2^{|K_i|+1} = 2^{|K_{i+1}|}$.

To summarize, if $G = \langle K \rangle$ and for every $\pi \in K$, the smaller set $K' = K \setminus \pi$ does not generate G, we have $m = |G| \geq 2^{|K|}$. Equivalently, $|K| \leq |\log_2(m)|$. $\square$

Corollary 3.2. *Subgroups of S_n can be described in space polynomial in n. More specifically, using $O(nlogn)$ generators.*

Proof. The group S_n had size $n!$. Therefore, from Theorem 3.1, we have that there exists a generating set of size at most $log(n!)$. We have

$$\begin{aligned} log_2(n!) &= \sum_{j=1}^{n} \log_2(j) \\ &\leq \int_0^n \log_2(x)\,dx \\ &= \frac{1}{\ln(2)}[x \ln x - x]_0^n \\ &\leq 2n \ln n \end{aligned}$$

Therefore there exists a generating set K where $|K| = O(n \ln n) = O(n \log_2 n)$.
$\square$

We first give a polynomial algorithm to compute a group's orbits given only the generators. After, we consider a specific generating set, the union of left coset representatives of the stabilizer chain.

3.1 Stabilizer Chain Coset Representatives Generating Set

Representating a group with its stabilizer chain coset representatives was intrudced by Furst, Hopcroft, and Luks. We first introduced the stabilizer chain and its properties in section 2 pages 6-6.

Recall that we have G_i is a subgroup of G that fixes elements $a_1, \ldots, a_i$, and U_i is the set of left coset representatives of G_{i+1} in G_i. From Corollary 2.12, the union of coset representatives U_i is a generating set for G of size $O(n^2)$. Although this is a larger generating set than the size $O(n \log n)$ one guaranteed by Corollary 3.2, it is convenient because it allows for efficient computation with the group.

Below we show how to efficiently compute the stabilizer chain left coset representatives.

3.1.1 Get Stabilizer Chain Coset Representatives from Generators

Suppose we have a partial list of coset representatives $C_0, \ldots, C_{n-2}$ such that each $\{e\} \subseteq C_i \subseteq U_i$. The following polynomial subroutine Filter(α) checks if $\alpha \in G$ can be written as $c_0 c_1 \ldots c_{n-2}$ where each $c_i \in C_i$. If not, Filter(α) expands the sets C_i such that it can.

Notice that at the start of each loop we have $\alpha_i \in G_i$. We start with $a_0 \in G = G_0$. In each iteration, if a coset representative c_i is found, then at the start of the next iteration we have $\alpha_i = c_i \alpha_{i+1}$ and $\alpha_0 = c_0 c_1 \ldots c_i \alpha_{i+1}$ with $\alpha_{i+1} = c_i^{-1} \alpha_i \in G_{i+1}$. Otherwise, we add α_i to C_i and get $\alpha_0 = c_0 c_1 \ldots c_{i-1} \alpha_i = c_0 \ldots c_{i-1} c_i c_{i+1} \ldots c_{n-2}$ for $c_i = a_i \in C_i$ and $c_{i+1}, \ldots, c_{n-2} = e$. Either way, at the end of the algorithm we have $a_0 = c_1 \ldots c_{n-2}$ where all $c_i \in C_i$. Additionally, since the lists C_i are expanded by only one element and only if there is a missing coset representative, we maintain the property that $C_i \subseteq U_i$ for all i.

Algorithm 1 Filter(α_0)

```
for $i = 0$ to $n-2$ do
  if $\exists$ some $c_i \in C_i$ such that $c_i^{-1}\alpha_i \in G_{i+1}$ then
    $\alpha_{i+1} \leftarrow c_i^{-1}\alpha_i$
  else
    add $\alpha_i$ to $C_i$
    return $[\alpha_i, i]$
  end if
end for
return [null, -1]
```

Filter(α) runs in polynomial time. There at most $n-1$ iterations of the for loop. For each iteration we test $c_i^{-1}\alpha_i \in G_{i+1}$ for at most $|C_i| \le |U_i| \le n$ elements. In general checking a permutation for membership in a group is not obviously polynomial. However, in this case we have $c_i^{-1}\alpha_i \in G_i$ and therefore fixes the points $\{a_1, \ldots, a_i\}$. Therefore checking if $c_i^{-1}\alpha_i \in G_{i+1}$ only requires testing if $c_i^{-1}\alpha_i$ fixes the point a_{i+1} which can be done in polynomial time.

Next we show how to use Filter to compute the stabilizer chain coset representatives U_i in polynomial time from any (polynomial-sized) generating set.

Algorithm 2

```
Input: Generating set $S$ with $\langle S \rangle = G$
Output: Stabilizer chain soset representatives $C_i$
  $C_i \leftarrow \{e\}$ for all $0 \le i \le n-2$
  toFilter $\leftarrow$ empty set
  for all $s \in S$ do
    toFilter $\leftarrow$ toFilter $\cup \{s, s^{-1}\}$
  end for
  while toFilter.$notEmpty()$ do
    $\alpha \leftarrow$ toFilter.$removeNext()$
    $[c_i, i]$ = Filter($\alpha$)
    if $c_i$ not null then
      for all $j \le i$ do
        toFilter $\leftarrow$ toFilter $\cup$ $c_i C_j$
      end for
    end if
  end while
  return $C_i$ for all $i$
```

Theorem 3.3. *Algorithm 2 runs in polynomial time and returns the stabilizer chain coset representatives $U_1, \ldots, U_{n-2}$.*

Proof. First, notice that after the original generators and their inverses are added to the set toFilter, new elements are only added whenever Filter(α) expands one of the sets C_i. Since $|C_i| \leq |U_i| \leq n$ and $0 \leq i \leq n-2$, the sets are expanded at most n^2 times. Each time the sets are expanded, we add at most n^2 new elements to the set toFilter. Therefore, the while loop is run at most $2|S| + n^4$ times. In the while loop, all operations including Filter are polynomial.Therefore if $|S|$ is polynomial in n, so is the runtime of Algorithm 2.

Next we prove that the returned lists C_i are in fact the complete coset representatives U_i. Since the lists C_i are only modified through Filter, we know that $C_i \subseteq U_i$ for all i. To show that the lists C_i are complete, we show that we can write an arbitrary $g \in G$ as a product of coset representatives $c_0 \dots c_{n-2}$ with $c_i \in C_i$. This product is named the canonical representation of g.

Since the algorithm calls Filter on all the generators S and their inverses, we can write them as a product of coset representatives in cannonical form. Therefore g can also be written as a product of coset representatives from C_i, and we can prove that the product can be rewritten using coset representatives in the right order, in cannonical form. The key to that proof is that the algorithm also called filter on all elements $c_i c_j$ with $i \geq j$. Notice that since c_i fixes elements $\{a_1, \dots, a_i\}$ and c_j fixes $\{a_1, \dots, a_j\}$, the product $c_i c_j$ fixes $\{a_1, \dots, a_j\}$. Therefore $c_i c_j \in G_j$ and can therefore be written as $a_j \dots a_{n-2}$ (see Proposition 2.9 and its proof for a refresher is this is not clear).

□

4 Systems of Imprimitivity

The algorithm testing bounded degree graphs for isomorphism relies on the understanding of systems of impromitivity. We first introduce this concept and some related theorems. Next we give the necessary polynomial algorithms related to systems of improimitivity.

4.1 Theory

Let G be a group acting on the set A and let the disjoint sets $A_1, A_2, ..., A_r$ partition A. We have A **invariant** under G if for all $g \in G$ and all $1 \leq i \leq r$, we have $g(A_i) = A_j$ for some $1 \leq j \leq r$. If a partition is invariant and nontrivial, i.e. $r \neq 1$ and $r \neq |A|$, than the partition is called a **system of imprimitivity** and the sets A_i are called **sets of imprimitivity** or **blocks**.

If G acts intransitively on A, the orbits of the set A form a system of imprimitivity. For a transitive graph we can define a system of imprimitivity in terms of a single block $B \subset A$. If there exists a non trivial subset B, with $|B| > 1$ where for all $\sigma, \tau \in G$ either

1. $\sigma(B) = \tau(B)$ or
2. $\sigma(B) \cap \tau(B) = \emptyset$,

then the collection $\{\sigma(B) | \sigma \in G\}$ is a system of imprimitivity also called a **G-block system**, where B is a **G-block**.

We say that G acts **primitively** on A if there is no system of imprimitivity for G. The theorems below develop a characterization of primitivity.

Theorem 4.1. *Let the group G be acting transitively and imprimitively on A, where B is a block , and $x \in B$ is an element of that block.*

(a) The setwise stabilizer $G_{\{B\}}$ is properly contained between G and the point stabilizer G_x. That is $G_x < G_{\{B\}} < G$.

(b) The block B has $[G_{\{B\}} : G_x]$ elements, and the system of imprimitivity containing B as a block has $[G : G_{\{B\}}]$ blocks.

Proof. (a) Let e be the identity element, and $\sigma \in G_x$. Since B is a block we have either $e(B) = \sigma(B)$ or $e(B) \cap \sigma(B) = \emptyset$. But since $\sigma(x) = x$ and $e(B) = B$ with $x \in B$, we have $e(B) \cap \sigma(B) \neq \emptyset$. Therefore $\sigma(B) = B$, or equivalently $\sigma \in G_{\{B\}}$. Since for every $g \in G_x$ we have $g \in G_{\{B\}}$, we get G_x is a subgroup of $G_{\{B\}}$.

Let y be another element in B. Since G is transitive, there exists $\tau \in G$ such that $\tau(x) = y$. By similar logic as above, $\tau(B) = B$. Since $\tau \in G_{\{B\}}$ but $\tau \notin G_x$, we get G_x is a proper subgroup of $G_{\{B\}}$.

$G_{\{B\}}$ is a proper subgroup of G, since if every element of G fixed B the system of imprimitivity $\{\sigma(B) | \sigma \in G\}$ would only contain the single block B with all the elements of A. But B was supposed to be a proper subset of A.

(b) Since G_x is a subgroup of $G_{\{B\}}$, we can $G_{\{B\}}$ in terms of its cosets as $G_{\{B\}} = G_x + \tau_1 G_x + \tau_2 G_x + ... + \tau_n G_x$ where $\tau_i \in G_{\{B\}}$. We demonstrate a bijection between the cosets of G_x in $G_{\{B\}}$ and elements of B therefore proving that $|B| = [G_{\{B\}} : G_x]$.

Notice that for every $\sigma \in \tau G_x$ we have $\sigma(x) = \tau(x) \in B$. Therefore we have a map from the cosets into the set B where τG_x gets mapped to $\tau(x)$.

This map is surjective. Because G is transitive on A and therefore B, we have $G_{\{B\}}$ is transitive on B. For every $b \in B$ there is a $\gamma \in G_{\{B\}}$ such that $\gamma(x) = b$, and therefore for every $b \in B$ there exists a coset γG_x that maps to b.

The map is also injective. Suppose there are two cosets $\tau_i G_x$ and $\tau_j G_x$ where $\tau_i(x) = \tau_j(x)$. Then for any $\sigma_i \in \tau_i G_x$ and $\sigma_j \in \tau_j G_x$ we have $\sigma_i(x) = \sigma_j(x)$ and therefore $\sigma_i^{-1}\sigma_j(x) = x$. We get $\sigma_i^{-1}\sigma_j \in G_x$, which means that σ_i and σ_j are actually in the same coset and the cosets $\tau_i G_x$ and $\tau_j G_x$ are equal.

Since there is a bijection between B and the cosets of G_x in $G_{\{B\}}$, we get $|B| = [G_{\{B\}} : G_x]$. Furthermore, by applying the orbit stabilizer theorem to G, we get $|A| = |Orb_G(x)| = [G : G_x]$. Therefore there are $|A|/|B| = [G : G_x]/[G_{\{B\}} : G_x] = [G : G_{\{B\}}]$ blocks in the system.

□

Corollary 4.2. *A group of prime degree is necessarily primitive.*

Theorem 4.3. *Let G be a group acting transitively on A, and G_x be the stabilizer for some point $x \in A$. If there is a proper subgroup H of G that properly contains G_x, then G has a system of imprimitivity consisting of the left cosets of H in G acting on x.*

Proof. We show that Hx is a block. First, notice that because $G_x < H$, we have $|Orb_H(x)| > 1$. Next, the orbit stabilizer theorem gives us that $|Orb_H(x)| = [H : G_x]$ and $|Orb_G(x)| = [G : G_x]$, and since $H < G$ we get $|Orb_H(x)| < |A|$. So Hx is a nontrivial subset of A of size greater than 1.

Now let $\sigma, \tau \in G$ be arbitrary permutations. Suppose $\tau(Hx) \cap \sigma(Hx) \neq \emptyset$. Then there exist $h_1, h_2 \in H$ such that $\tau h_1 x = \sigma h_2 x$. But then we get $h_2^{-1}\sigma^{-1}\tau h_1 x = x$, or equivalently $h_2^{-1}\sigma^{-1}\tau h_1 \in G_x$, and therefore $\sigma^{-1}\tau \in h_2 G_x h_1^{-1}$. Since $G_x \in H$ we have $\sigma^{-1}\tau = h$ for some $h \in H$. And therefore $\tau(Hx) = \sigma\sigma^{-1}\tau(Hx) = \sigma h(Hx) = \sigma(hHx) = \sigma(Hx)$ which is what we wanted.

Finally, for the block Hx, the system of imprimitivity $\{\sigma(Hx)|\sigma \in G\}$ is indeed the left cosets of H in G acting on x since $\sigma Hx = \tau Hx$ for $\sigma, \tau \in G$ iff σ and τ are in the same coset of H.

□

Corollary 4.4. *A group G that acts transitively on set A is primitive iff for all $x \in G$ the stabilizer G_x is a maximal subgroup of G.*

Corollary 4.5. *Let G be a transitive p-subgroup of $Sym(A)$ with $|A| > 1$. If $|A| \neq p$, then G is imprimitive. Equivalently if G primitive, then $|A| = p$.*

Proof. If G is intransitive, its orbits form a system of imprimitivity and G is therefore imprimitive. Otherwise, suppose G is transitive on A and $|G| = p^n$. For any $x \in A$ we have $|A| = |Orb(x)| = [G : G_x] = |G|/|G_x|$. Therefore if $|A| > 1$ and $|A| \neq p$, then $|A| = p^m$ for $1 < m \leq n$ and $|G_x| < |G|/p = p^{n-1}$. According to Sylow's first theorem, since G_x is a $p-$subgroup of G, there exists some subgroup $H < G$ of order p^{n-1} that contains G_x. And since $|G_x| < p^{n-1} = |H| < |G|$, we get that G_x is not maximal and therefore G is imprimitive. □

A system of imprimitivity is **minimal** if G acts primitively on the blocks. While in general, the number of blocks in a minimal system of imprimitivity is not uniquely determined, we have the following.

Theorem 4.6. *Let G be a transitive p-subgroup of $Sym(A)$ with $|A| > 1$. Then any minimal system of imprimitivity consists of exactly p blocks and the subgroup $H = \{\sigma \in G|\forall$ blocks $A_i, \sigma(A_i) = A_i\}$ that stabilizes the blocks has index p in G.*

Proof. Let $\bar{A}$ be some system of imprimitivity for G on A. Let H be the subgroup of G that stabilizes the blocks of $\bar{A}$. Notice that H is a proper subgroup of G since G is transitive and H is not. We will show that H is normal in G. For $g \in G$ and $h \in H$, g permutes the blocks and elements withing blocks,

h permutes within the blocks, and g^{-1} permutes the blocks back while also reordering within the blocks. Since blocks are stabilized under h, in $g^{-1}hg$ the g^{-1} action reverses the block swaps of g, and the blocks are stable under $g^{-1}hg$. That is, $g^{-1}hg \in H$, so H is normal in G.

Since H is normal in G, the cosets of H in G form a group acting on the blocks. Formally, G/H is isomorphic to a transitive $p-$subgroup of $Sym(\bar{A})$, where $|\bar{A}|$ is the number of blocks. When $\bar{A}$ is minimal, G/H acts primitively on the blocks, and therefore by Corollary 4.5 we have $|\bar{A}| = p$. Since G/H is a $p-$subgroup of $Sym(\bar{A})$, and $H < G$, we have $|G/H| = p$. □

4.2 Systems of Imprimitivity Algorithms

A partition p_1 is finer than a partition p_2 if every element of p_1 is a subset of some element of p_2. Given a generating set S for group G acting on A and two points $a_1, a_2 \in A$ there is a polynomial algorithm to find the finest invariant partition of A that has a_1 and a_2 in the subset.

In the following algorithm we use the union-find data structure. This data structure has two main functions. *Find* determines the subset of an element, usually by returning a representative element of the subset. Two elements a_1 and a_2 are in the same subset if $find(a_1) = find(a_2)$. *Merge* takes two elements and joins the two subsets which contain these elements into one. This data structure can be implemented in a number of ways including linked lists and arrays, but either way these operations are polynomial time.

Algorithm 3

```
Input: Generating set $S$ with $\langle S \rangle = G$ acting on $A$ and $a_1, a_2 \in A$.
Output: Finest invariant partition of $A$ which has $a_1$ and $a_2$ in same subset
  part ← partition of singletons on $A$
  part.$merge(a_1, a_2)$
  toCheck ← $\{\{a_1, a_2\}\}$
  while toCheck not empty do
    $\{a, b\}$ ← toCheck.$removeNext()$
    for all $s \in S$ do
      $p_a$ ← part.$find(s(a))$
      $p_b$ ← part.$find(s(b))$
      if $p_a \neq p_b$ then
        part.$merge(p_a, p_b)$
        toCheck.$add(\{p_a, p_b\})$
      end if
    end for
  end while
  return part
```

Proposition 4.7. *Given a polynomial sized generating set S, Algorithm 3 runs in polynomial time.*

Proof. New elements are added to toCheck only after a merge. Therefore, since there are at most $|A|$ merges before the partition becomes trivial and no more pairs are added to toCheck, the while loop runs at most $|A|$ times. All actions in the for loop are polynomial, so as long as $|S|$ is polynomial in $|A|$, so is the algorithm. □

All other algorithms for systems of imprimitivity we need directly follow.

Corollary 4.8. *Given a polynomial sized generating set, there is a polynomial time algorithm to find a system of imprimitivity, if one exists.*

Algorithm 3 finds a finest invariant partition, but that partition may be trivial. Therefore we run Algorithm 3 on all pairs $a_1, a_2 \in A$. We will either find a system of imprimitivity or, if for all pairs the partition returned is trivial, G acts primitively on A.

Corollary 4.9. *Given a polynomial sized generating set, there is a polynomial time algorithm to find a minimal system of imprimitivity.*

5 Isomorphism of Graphs of Bounded Valence

5.1 Trivalent Case

To show that isomorphism of trivalent graphs can be tested in polynomial time, we reduce the problem to the Color Automorphism Problem for 2-groups and give a polynomial algorithm for that problem.

5.1.1 Reduction to the Color Automorphism Problem

Let $Aut_e(X)$ be the group of all automorphisms of the graph X that fix edge e. That is, given $e = \{v, w\}$, for $\sigma \in Aut_e(X)$ either $\sigma(v) = v$ and $\sigma(w) = w$, or $\sigma(v) = w$ and $\sigma(w) = v$.

First we show that testing isomorphism for a trivalent graph X is polynomial time reducible to determining the generators for $Aut_e(X)$. Next we show that $Aut_e(X)$ is a 2-group. Finally we show that we can determine the generators for $Aut_e(X)$ in polynomial time if we have a polynomial algorithm for the Color Automorphism Problem on 2-Groups.

Proposition 5.1. *Testing two trivalent connected graphs X' and X'' for isomorphism is polynomial-time reducible to the problem of determining generators for $Aut_e(X)$, where X is a trivalent connected graph and e is a distinguished edge.*

Proof. Suppose we are given two trivalent connected graphs X' and X''. Pick any edge $e_1 \in E(X')$. If X' and X'' are isomorphic, then there exists an edge $e_2 \in E(X'')$ such that an isomorphism from X' to X'' maps e_1 to e_2. For each $e_2 \in E(X'')$ we test if such an isomorphism exists by constructing a new graph X.

Let X be the disjoint union $X' \cup X''$ with new vertices v_1 and v_2 inserted into e_1 and e_2 respectively and connected by a new edge $e = \{v_1, v_2\}$. Notice that X is connected and trivalent and there is an isomoprhism from X' to X'' mapping e_1 to e_2 iff there is an element in $Aut_e(X)$ that transposes v_1 and v_2. Since $Aut_e(X)$ only has an element that transposes v_1 and v_2 iff at least one if its generators does, all we have to do is compute the generators for $Aut_e(X)$ and check if any of them switch v_2 and v_2. □

To better understand $Aut_e(X)$ for a trivalent connected graph X, we analyze a sequence of groups $Aut_e(X_r)$ where X_r is the subgraph of X containing all the vertices and edges which appear in paths of length $\leq r$ through the edge e. That is, for a graph with n vertices, X_1 contains only the edge e, and $X_{n-1} = X$.

Note that since an automorphism must preserve shortest path lengths, vertices a distance d away from e can only be mapped to other vertices the same distance away from e. Therefore for any $s < r$, any automorphism in $Aut_e(X_r)$ must map vertices of X_s to other vertices of X_s.

If we have an automorphism $\sigma \in Aut_e(X_{r+1})$, we can easily get an automorphism in $Aut_e(X_r)$ simply by restricting σ to the vertices in X_r. Therefore, the groups $Aut_e(X_r)$ are related via the induced homomorphisms

$$\pi_r : Aut_e(X_{r+1}) \to Aut_e(X_r),$$

where for $\sigma \in Aut_e(X_{r+1})$, $\pi(\sigma)$ is the same permutation restricted to X_r.

These homomorphisms π_r allow us to iteratively build up $Aut_e(X)$. Given the generators for $Aut_e(X_r)$, we determine the generators of $Aut_e(X_{r+1})$ by solving the following problems:

Problem 5.2. *Find the set, $\mathscr{R}$, of generators for K_r, the kernel of π_r.*

Problem 5.3. *Find the set, $\mathscr{S}$, of generators for $\pi_r(Aut(X_{r+1}))$, image of π_r.*

The next proposition details how to compute the generators of $Aut_e(X_{r+1})$ using $\mathscr{R}$ and $\mathscr{S}$.

Proposition 5.4. *If $\mathscr{S}'$ is any pullback of $\mathscr{S}$ in $Aut_e(X_{r+1})$, i.e. $\pi_r(\mathscr{S}') = \mathscr{S}$, then $\mathscr{R} \cup \mathscr{S}'$ generates $Aut_e(X_{r+1})$.*

Proof. Fix an arbitrary $\sigma \in Aut_e(X_{r+1})$ where $\pi_r(\sigma) = \psi$. We know $\mathscr{S}'$ contains ψ' such that $\pi_r(\psi') = \psi$. Since $\pi_r(\sigma) = \pi_r(\psi')$, we know σ and ψ' act the same on vertices in $V(X_r)$. Therefore the permutation $\sigma{\psi'}^{-1}$ stabilizes all the vertices in $V(X_r)$ and is in K_r. We get that for some $\rho \in K_r$, $\sigma = \rho\psi'$. Since $\rho \in K_r$ it is generated by permutations in $\mathscr{R}$, and since $\psi' \in \mathscr{S}'$, we get $\sigma = \rho\psi'$ is generated by $\mathscr{R} \cup \mathscr{S}'$. □

To solve these two problems let us consider the new vertices $V(X_{r+1}) \backslash V(X_r)$. Since X is a trivalent graph, each one of these vertices is connected to one, two or three vertices in X_r. We summarize this parent-child relationship as follows: let A be the collection of all subsets of $V(X_r)$ of size one, two, or three, and define

$$f : V(X_{r+1})\backslash V(X_r) \to A$$

where $f(v) = \{w \in V(X_r) | \{v, w\} \in E(X)\}$. Note that f is defined for on all vertices except the two on the edge e since they don't have any parents.

We call vertices $v_1 \neq v_2$ twins if $f(v_1) = f(v_2)$. Notice that since the two vertices in X_1 are connected to each other and any vertex in X_r for $r > 1$ is connected to at least one vertex in X_{r-1}, a vertex in $V(X_r)$ is connected to at least one vertex in $V(X_r)$. We get that triplets cannot exist as that would imply a vertex in $V(X_r)$ is connected to at least one vertex in $V(X_r)$ and three more vertices in $V(X_{r+1})\backslash V(X_r)$ and therefore has degree of more than three.

The relationship between an automorphism in $Aut_e(X_{r+1})$ and parent-child relationship is as follows.

Proposition 5.5. *For $\sigma \in Aut_e(X_{r+1})$ we have $f(\sigma(v)) = \sigma(f(v))$.*

Proof. We have $w \in f(v)$ iff there is an edge $\{v, w\} \in E(X)$ and $w \in V(X_r)$. Since σ is an automorphism, we have $\{\sigma(v), \sigma(w)\} \in E(X)$ iff $\{v, w\} \in E(X)$. As we noted before, $\sigma(w) \in V(X_r)$ iff $w \in V(X_r)$. Therefore we get $w \in f(v)$ iff $\sigma(w) \in f(\sigma(v))$ and so $f(\sigma(v)) = \sigma(f(v))$. □

We now have enough to solve problem 5.2 of finding the generators for K_r.

Proposition 5.6. *K_r is a 2-group generated by the transpositions in each pair of twins.*

Proof. The previous proposition tells us that for $\sigma \in K_r$ we have $f(\sigma(v)) = f(v)$. That is, either $v = \sigma(v)$ or v and $\sigma(v)$ are twins. It follows that σ can be written as a product of permutations in some of the twin pairs. Since any permutation in a twin pair is a valid permutation in K_r, we get that K_r is generated by the transpositions in each pair of twins.

We have that K_r is a 2-group since each element of K_r is of order two. This follows from Cauchy's Theorem (see appendix) since if K_r was not a 2-group it would have to contain an element of prime order other than two. □

We use this proposition and an inductive argument to prove that $Aut_e(X_r)$ must also be a 2-group.

Proposition 5.7. *For any r, $Aut_e(X_r)$ is a 2-group.*

Proof. The group $Aut_e(X_1)$ only contains two elements: the identity and the permutation that switches the two end vertices of the edge e. Now let $r \geq 1$ and suppose $Aut_e(X_r)$ is a 2-group. Combining the first group isomorphism theorem and Lagrange's theorem we get that

$$|Aut_e(X_{r+1})| = |\text{Image}(\pi_r)| \cdot |K_r|.$$

Since Image(π_r) is a subgroup of $Aut_e(X_r)$, it is a 2-group. We know K_r is a 2-group. We get that $Aut_e(X_{r+1})$ is also a 2-group. □

Now that we have solved problem 5.2 of finding the generators for K_r, we solve problem 5.3 by figuring out which permutations in $Aut_e(X_r)$ can be extended to a permutation in $Aut_e(X_{r+1})$.

First, we show that any permutation $\sigma \in Aut_e(X_{r+1})$ must map pairs of twins to other pairs of twins. Let $v_1, v_2 \in V(X_{r+1}) \backslash V(X_r)$ be twins. That is, $f(v_1) = f(v_2)$. Proposition 5.5 gives us

$$f(\sigma(v_1)) = \sigma(f(v_1)) = \sigma(f(v_2)) = f(\sigma(v_2)),$$

and therefore $\sigma(v_1)$ and $\sigma(v_2)$ are twins. We get that σ maps pairs of twins to pairs of twins, and therefore the single children (those not part of a twin pair) must be mapped to other single children.

Now let's consider how σ acts on the vertices of $V(X_r)$. Since σ maps twins to twins and single children to other single children, proposition 5.5 implies that parents of twins are mapped to parents of twins and parents of a single child are mapped to parents of a single child. Therefore in order for a permutation of X_r to be extended to a valid permutation of X_{r+1} it must setwise stabilize the following two sets:

$$A_1 = \{a \in A | a = f(v) \text{ for some unique } v \in V(X_{r+1}) \backslash V(X_r)\},$$

the sets of parents with a single child, and

$$A_2 = \{a \in A | a = f(v_1) = f(v_2) \text{ for some } v_1 \neq v_2\},$$

the sets of parents with twins.

Finally, consider the new edges in $E(X_{r+1}) \backslash E(X_r)$ that connect two vertices in $V(X_r)$. These correspond to the following subset of A,

$$A' = \{\{w_1, w_2\} \in A | \{w_1, w_2\} \in E(X_{r+1})\}.$$

We will show that any permutation $\sigma \in Aut_e(X_{r+1})$ must setwise stabilize the set A'. Fix an arbitrary edge $\{w_1, w_2\} \in A'$. Since $w_1, w_2 \in V(X_r)$, we have $\sigma(w_1), \sigma(w_2) \in V(X_r)$. Furthermore, because automorphisms map edges to edges, we know $\{\sigma(w_1), \sigma(w_2)\} \in E(X_{r+1})$. That is $\{\sigma(w_1), \sigma(w_2)\} \in A'$.

We have shown that an element of $Aut_e(X_{r+1})$ must stabilize the sets A_1, A_2, and A'. In fact, these three conditions are sufficient for an element $\pi \in Aut_e(X_r)$ to be extended to a valid element of $Aut_e(X_{r+1})$.

Proposition 5.8. *Image*(π_r) *is precisely the set of those* $\pi \in Aut_e(X_r)$ *which setwise stabilize the collections* A_1, A_2*, and* A'*. That is, any element* $\pi \in Aut_e(X_r)$ *that setwise stabilizes* A_1, A_2, A' *can be extended to an element in* $Aut_e(X_{r+1})$*.*

Proof. We have already shown that any $\sigma \in Aut_e(X_{r+1})$ must stabilize A_1, A_2, A'. Therefore any $\pi \in \text{Image}(\pi_r)$ does too.

It remains to show that this condition is sufficient. We do so by showing how to extend an arbitrary permutation $\pi \in Aut_e(X_r)$ that stabilizes A_1, A_2, A'. We define the extension as follows. For only children $v \in V(X_{r+1}) \backslash V(X_r)$, where $f(v) \in A_1$, there is only one option. Map v to the child of $\pi(f(v))$. Notice that since π stabilizes A_1, we have $\pi(f(v)) \in A_1$, and so every vertex in $\pi(f(v))$ has the same child. For twin pairs v_1 and v_2, we have $\pi(v_1) = \pi(v_2) \in A_2$. Map v_1 and v_2 to the twin children of $\pi(v_1)$ in any order.

Since all vertices in $V(X_{r+1}) \backslash V(X_r)$ are either in a pair of twins or a single child, we have successfully extended $\pi \in Aut_e(x_r)$ to a permutation of $V(X_{r+1})$. All that remains to be checked is that our extension π' is in fact an automorphism in $Aut_e(X_{r+1})$.

A permutation is an automorphism iff it maps edges to edges. We have three types of edges in $E(X_{r+1})$. The first are edges in $E(X_r)$. Since, π was an automorphism of X_r and π' acts identically on edges in $E(X_r)$, we can conclude those edges are mapped to edges in $E(X_r)$. Next are edges in A'. Since π stabilizes A', so does π', and therefore π' maps edges in A' to edges in A'. Finally, we have edges $\{v, w\}$ for $v \in V(X_{r+1}) \backslash V(X_r)$ and $w \in V(X_r)$. By construction $\pi'(w) \in f(\pi'(v))$ and therefore $\{\pi'(v), \pi'(w)\}$ is an edge. □

Notice that this proposition not only showed the path to solving problem 5.3 of computing the set $\mathscr{S}$, but also demonstrated how to compute the pullback $\mathscr{S}'$ given $\mathscr{S}$.

In fact, we have now almost completed the reduction from testing isomorphism of two trivalent graphs to the Color Automorphism Problem for 2-groups. First we give a formal definition of the Color Automorphism Problem. Next we summarize the work we have accomplished so far. Finally, we give the last details of the reduction.

The formal definition of the Color Automorphism Problem is as follows.

Problem 5.9. *Given a set of generators for a subgroup, G, of $Sym(A)$ where A is a colored set, find a set of generators for the subgroup $\{\sigma \in G | \sigma$ is color preserving$\}$. A permutation $\sigma \in G$ is color preserving if for all $a \in A$ the color of $\sigma(a)$ is the same as the color of a.*

The Color Automorphism Problem for 2-groups is the same problem except G is a 2-group.

Let us summarize what we have so far. First we reduced the problem of testing two connected trivalent graphs X' and X'' to computing, a polynomial number of times, the generators for $Aut_e(X)$ where X is trivalent, connected, and of size polynomial in the sizes of X' and X''.

We compute $Aut_e(X)$ iteratively by computing the generators of every group in the series $Aut_e(X_r)$. The generator for $Aut_e(X_1)$ is simply the permutation which switches the end vertices of the edge e.

Once we have the generators for $Aut_e(X_r)$, we compute the sets $\mathscr{R}$, the generators for K_r, and $\mathscr{S}'$, the pullback of the generators $\mathscr{S}$ for Image(π_r). The set $\mathscr{R} \cup \mathscr{S}'$ generates $Aut_e(X_{r+1})$.

In polynomial time we can find the twin pairs in $V(X_{r+1}) \backslash V(X_r)$. $\mathscr{R}$, consists of the transpositions in each pair of twins. $\mathscr{S}$ is computed by finding the generators for the subgroup of $Aut_e(X_r)$ that setwise stabilizes A_1, A_2, and A'. Finally, proposition 5.8 shows us how to extend $\mathscr{S}$ to $\mathscr{S}'$.

All that remains is to formulate the problem of finding the generators for the subgroup of $Aut_e(X_r)$ that stabilizes A_1, A_2, A' as a color automorphism problem. That is, we need to color code the elements of A such that a permutation is color preserving if and only if it stabilizes the sets A_1, A_2, and A'.

Let $A_0 = A \backslash (A_1 \cup A_2)$. Notice that A_0, A_1, A_2 are disjoint and partition A. However, having just three colors for these sets isn't enough since we also need to stabilize A'. Stabilizing A_i and A', which may intersect, is equivalent to stabilizing $A_i \cup A'$ and $A_i \backslash A'$. Therefore we partition A into the following six disjoint sets and assign each a different color:

$$A_0 \cap A', \quad A_1 \cap A', \quad A_2 \cap A', \quad A_0 \backslash A', \quad A_1 \backslash A', \quad A_2 \backslash A'.$$

The diagram bellow illustrates this partition.

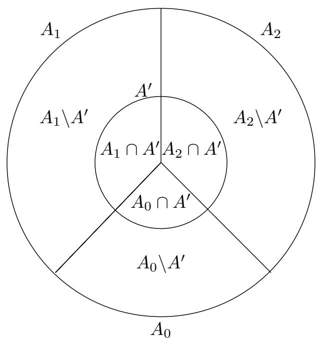

Figure 1: The six disjoint regions of A to be stabilized

Although immaterial to algorithm structure or runtime, it may be interesting to note that $A_2 \cap A' = \emptyset$ for all $r \geq 2$. Here is a short proof. Suppose there exists a set $a \in A_2 \cap A'$. Recall that a is a subset of $V(X_r)$. For $r \geq 2$, a vertex $v \in a$ has at least one neighbor in $V(X_{r-1})$. Since $a \in A'$, we know that v has a neighbor in $V(X_r) \backslash V(X_{r-1})$. However, since $a \in A_2$, we know that v also has two children in $V(X_{r+1}) \backslash V(X_r)$. We get that v is of degree at least 4, which is a contradiction.

Finally, notice that the Color Automorphism Problem asks for a group G, a subgroup of $Sym(A)$, where is A is the colored set. In our case, $Aut_e(X_r)$ is a subgroup of $Sym(V(X_r))$, and the colored set A is a set of subsets of $V(X_r)$. If A was the set of all subsets of $V(X_r)$, we would have a problem, since there is an exponential number of subsets. However, since our subsets are of bounded size, the size of A is polynomial in the number of vertices. We can think of $Aut_e(X_r)$ as implicitly acting on A. Computing the action of a permutation on $a \in A$ simply involves computing the result for each of the vertices in a. Therefore we in fact have $Aut_e(X_r)$, a subgroup of $Sym(A)$, and a colored set A, where $|A|$ is polynomial in the number of vertices. We have successfully matched all the requirements for the Color Automorphism Problem.

We have given a polynomial time reduction from the Trivalent Graph Isomorphism Problem to the Color Automorphism Problem. However, since we have shown that $Aut_e(X_r)$ is a 2-group for all r, we reduced the Trivalent Graph Isomorphism Problem to the Color Automorphism Problem for 2-groups. In the next section we give the Color Automorphism Algorithm for 2-groups.

5.1.2 Color Automorphism Algorithm for 2-Groups

First, some notation and observations. For $a, b \in A$, let $a \sim b$ denote that a has the same color as b. For $B \subseteq A$ and $K \subseteq Sym(A)$, let

$$\mathscr{C}_B(K) = \{\sigma \in K | \forall b \in B, \sigma(b) \sim b\}.$$

Notice that

1. $\mathscr{C}_B(K \cup K') = \mathscr{C}_B(K) \cup \mathscr{C}_B(K')$
2. $\mathscr{C}_{B \cup B'}(K) = \mathscr{C}_{B'}(\mathscr{C}_B(K)) = \mathscr{C}_{B'}\mathscr{C}_B(K)$

These two properties suggest a divide and conquer algorithm. In the transitive case we will consider orbits, otherwise blocks of imprimitivity. The number and distribution of colors is unimportant. However, a generalization of the Color Automorphism Problem is required in order to admit a recursive procedure. The generalization we need is

Problem 5.10. *Given a set of generators for a subgroup G of $Sym(A)$, a G-stable subset $B \subseteq A$, and $\sigma \in Sym(A)$, find $\mathscr{C}_B(\sigma G)$.*

Notice that Problem 5.9 is the special case with $B = A$ and $\sigma = 1$.

To understand how to represent the output of Problem 5.10 we have the following proposition.

Proposition 5.11. *If $\mathscr{C}_B(\sigma G)$ is not empty, then it is a left coset of the subgroup $\mathscr{C}_B(G)$.*

Proof. First, notice that since B is G-stable, $\mathscr{C}_B(G)$ is closed under products and inverses and therefore a subgroup of G. Next, we will show that for any $\sigma_0 \in \mathscr{C}_B(\sigma G)$, we have $\mathscr{C}_B(\sigma_0 G) = \sigma_0 \mathscr{C}_B(G)$.

Fix an arbitrary $\tau \in G$ and $b \in B$. We have $\tau(b) \in B$ since B is G-stable. That gives us $\sigma_0\tau(b) \sim \tau(b)$ since σ_0 is color preserving on B. Suppose that $\sigma_0\tau$ is color preserving on B and therefore in $\mathscr{C}_B(\sigma_0 G)$. Then $b \sim \sigma_0\tau(b)$, and since $\sigma_0\tau(b) \sim \tau(b)$ we get $b \sim \tau(b)$. That is, if $\sigma_0\tau \in \mathscr{C}_B(\sigma_0 G)$, we get $\tau \in \mathscr{C}_b(G)$ and therefore $\sigma_0\tau \in \sigma_0\mathscr{C}_B(G)$. So $\mathscr{C}_B(\sigma_0 G) \subseteq \sigma_0\mathscr{C}_B(G)$. Alternatively, if $\tau \in \mathscr{C}_B(G)$, we get $b \sim \tau(b) \sim \sigma_0\tau(b)$. So $\sigma_0\tau \in \mathscr{C}_B(\sigma_0 G)$ and therefore $\sigma_0\mathscr{C}_B(G) \subseteq \mathscr{C}_B(\sigma_0 G)$.

We have shown that $\mathscr{C}_B(\sigma_0 G) \subseteq \sigma_0\mathscr{C}_B(G)$ and $\sigma_0\mathscr{C}_B(G) \subseteq \mathscr{C}_B(\sigma_0 G)$. Therefore $\mathscr{C}_B(\sigma_0 G) = \sigma_0\mathscr{C}_B(G)$. Finally, since $\sigma_0 \in \mathscr{C}_B(\sigma G)$, we have $\sigma_0 \in \sigma G$ and therefore σG and $\sigma_0 G$ are the same coset. So we get $\mathscr{C}_B(\sigma G) = \mathscr{C}_B(\sigma_0 G) = \sigma_0\mathscr{C}_B(G)$. □

Therefore our algorithm for Problem 5.10 will return either $\emptyset$ or a coset representative element and a set of generators for $\mathscr{C}_B(G)$. The algorithm proceeds as follows:

First we compute the orbits of B under G to check if G acts transitively on B. If not, let $B_1, \ldots, B_i$ be the orbits B. We have

$$\mathscr{C}_B(\sigma G) = \mathscr{C}_{B_i} \ldots \mathscr{C}_{B_1}(\sigma G),$$

where, since the orbits are G-stable, the right hand side is computed with i calls to the algorithm for the transitive case for Problem 5.10 described next.

If G acts transitively on B, we break B into blocks of imprimitivity. Recall that since G is a 2-group, Theorem 4.6 guarantees that any minimal system of imprimitivity contains exactly 2 blocks of equal size[2], and the group H that stabilizes the blocks has index 2 in G. Notice that since B' and B'' are not G-stable, we cannot compute $\mathscr{C}_{B'}\mathscr{C}_{B''}(\sigma G)$ as in the intransitive case. Instead let $G = H \cup \tau H$. We have

$$\begin{aligned}\mathscr{C}_B(\sigma G) &= \mathscr{C}_B(\sigma H) \cup \mathscr{C}_B(\sigma\tau H) \\ &= \mathscr{C}_{B''}\mathscr{C}_{B'}(\sigma H) \cup \mathscr{C}_{B''}\mathscr{C}_{B'}(\sigma\tau H).\end{aligned}$$

If at least one of $\mathscr{C}_B(\sigma H)$ or $\mathscr{C}_B(\sigma\tau H)$ is empty, we are done. However, if not, we need a way to express the union of the two answers as a single coset. Proposition 5.11 guarantees that this can be done, and the next proposition shows how.

Proposition 5.12. *Suppose that $\mathscr{C}(\sigma H) = \rho_1\mathscr{C}_B(H)$ and $\mathscr{C}(\sigma\tau H) = \rho_2\mathscr{C}_B(H)$. Then $\mathscr{C}_B(\sigma G) = \mathscr{C}_B(\sigma H) \cup \mathscr{C}_B(\sigma\tau H) = \rho_1\langle\mathscr{C}_B(H) \cup \rho_1^{-1}\rho_2\rangle$.*

Proof. First, notice that $\rho_1\mathscr{C}_B(H), \rho_2\mathscr{C}_B(H) \subseteq \rho_1\langle\mathscr{C}_B(H) \cup \rho_1^{-1}\rho_2\rangle$. Therefore $\mathscr{C}_B(\sigma G) \subseteq \rho_1\langle\mathscr{C}_B(H) \cup \rho_1^{-1}\rho_2\rangle$.

Next we show that $\langle\mathscr{C}_B(H) \cup \rho_1^{-1}\rho_2\rangle \subseteq \mathscr{C}_B(G)$. Because $\rho_1 \in \mathscr{C}_B(\sigma H)$ and $\rho_2 \in \mathscr{C}_B(\sigma\tau H)$, we have $\rho_1, \rho_2 \in G$ are color preserving on B. So $\rho_1, \rho_2 \in \mathscr{C}_B(G)$ and $\rho_1^{-1}\rho_2 \in \mathscr{C}_B(G)$. Since $\mathscr{C}_B(H) \subseteq \mathscr{C}_B(G)$, we get $\langle\mathscr{C}_B(H) \cup \rho_1^{-1}\rho_2\rangle \subseteq \mathscr{C}_B(G)$.

Finally, proposition 5.11 states that $\mathscr{C}_B(\sigma G) = \rho\mathscr{C}_B(G)$ for some $\rho \in G$. Since $\langle\mathscr{C}_B(H) \cup \rho_1^{-1}\rho_2\rangle \subseteq \mathscr{C}_B(G)$ and $\mathscr{C}_B(\sigma G) \subseteq \rho_1\langle\mathscr{C}_B(H) \cup \rho_1^{-1}\rho_2\rangle$, we have $\mathscr{C}_B(\sigma G) = \rho_1\langle\mathscr{C}_B(H) \cup \rho_1^{-1}\rho_2\rangle$. $\square$

So to compute $\mathscr{C}_B(\sigma G)$ where G is intransitive on B, we can in polynomial time compute a minimal system of imprimitivity with two blocks B' and B'' and determine the generators for the group H which stabilizes the two blocks. We also find an element $\tau \in G$ such $\tau \notin H$ and therefore $G = H + \tau H$. We then make four recursive calls, solving the same problem with B' and B'' of half the size.

Finally, since this is a recursive algorithm, we need a base case. In both the general and the transitive algorithm that is when $|B| = 1$. In that case, since

[2]The equal size is not from Theorem 4.6, just a property of blocks

$B = \{b\}$ is G-stable, we have $Gb = b$. Therefore

$$\mathscr{C}_B(\sigma G) = \begin{cases} \sigma G & \text{if } \sigma(b) \sim b \\ \emptyset & \text{if } \sigma(b) \nsim b. \end{cases}$$

References

[1] Butler, G. Fundamental Algorithms for Permutation Groups. Lecture Notes in Computer Science, Vol. 559. Springer- Verlag. Heidelberg (1991).

[2] Furst M., Hopcroft J. and Luks E., Polynomial-time algorithms for permutation groups. 21st Annual Symposium on Foundations of Computer Science, 36-41. Syracuse (1980)

[3] Hoffmaun, C. M. Group-Theoretic Algorithms and Graph Isomorphism. Lecture Notes in Computer Science, Vol. 136. Springer- Verlag. Heidelberg (1982).

[4] Luks E.M. Isomorphism of graphs of bounded valence can be tested in polynomial time. Journal of Computer System Sciences 25, 42-65 (1982)

[5] Mathon R. A note on the graph isomorphism counting problem. Information Processing Letters Volume 8, Issue 3, 131-136 (1979)