CS498A UGP-II
Report

Walking through Babai's Algorithm

Bachelor of Technology
in
Computer Science and Engineering

Submitted by

13020 Abhimanyu Yadav

Under the guidance of
Professor Nitin Saxena

Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

Fall Semester 2016

Contents

Chapter 1

Introduction

1.1 Problem Definition

Given two graphs $X_1 = (V_1, E_1)$ and $X_2 = (V_2, E_2)$, the graph isomorphism (GI) problem is the decision problem whether $X_1 \cong X_2$. Here, an isomorphism is a bijection from V_1 to V_2 that preserves adjacency i.e. :
$\forall\ i, j$ s.t. $v_i, v_j \in V_1 : (v_i, v_j) \in E_1 \Leftrightarrow (f(v_i), f(v_j)) \in E_2$.

1.2 Prologue

The following report covers the details of Babai's algorithm by actually trying to solve to an arbitrary isomorphism problem that is assumed to be adversarial and will thus touch every aspect of the algorithm.
It is safe to assume in the above problem that $\mid V_1 \mid = \mid V_2 \mid = n$, because otherwise we could trivially reject isomorphism. It is this number "n" that governs the runtime of the algorithms operating on this problem. The previous known algorithm to solve the problem was given by Eugene Luks in 1982 which had a run time $exp(\mathcal{O}\sqrt{(n \log n)})$, which is moderately exponential. This algortihm [2], however, solves the problem in quasipolynomial time i.e. $exp((\log n)^{\mathcal{O}(1)})$.

Chapter 2

Babai vs Luks

2.1 Target Recurrence

Babai's algorithm, just like Luks' algorithm is a recursive algorithm that breaks the initial problem into a "moderate" number of "significantly smaller" instances. The following is the target recurrence:

$$f(n) \leq q(n) f(\tfrac{9n}{10})$$

For all sufficiently large values of n, the recursion unfolds into the following:

$$f(n) \leq q(n)^{\mathcal{O}(\log n)}$$

We observe that for $f(x)$ to be quasipolynomially bounded, $q(x)$ must be quasipolynomially bounded. And thus our notion of "moderate" number of instances becomes "quasipolynomially bounded" number of instances and our notion of "significantly smaller" instances becomes smaller by a fixed fraction, which is $\frac{9}{10}$ in this case.

2.2 From Luks to Babai

We observe that the run time of Luks's algorithm is not quasipolynomial because in the process of recursion, for some graphs, we encounter a barrier case where we are not able to break the problem into quasipolynomially bounded number of instances and hence the algorithm overshoots the quasipolynomial bound. The scheme used in the Babai's algorithm is to use Luks's method to solve the problem until we encounter that barrier case. Once we encounter that barrier case, we use Babai's method to break it into the desired quasipolynomial number of instances.

Chapter 3

Luks's Algorithm

We first describe Luks's Algorithm and how it reduces the problem of Graph Isomorphism to an instance of Color Isomorphism problem. Here, we introduce the framework in which we will solve the problem.

3.1 Group Theoretic Framework

By definition, every bijection from a set to itself is also a permutation of the members of that set. The set of all bijections from any set A to itself forms a group under the composition operation. This group is called the symmetry group and falls under the general class of groups called permutation groups. This realisation opened the doors for solving GI using group theory.

3.2 Basics for the framework

3.2.1 Permutation Group

Consider a set Ω, the permutation domain. This will be the set over which we will be permuting. Then, Sym(Ω), the set of all permutations over Ω is called the symmetric group. Any subgroup G $\subseteq \Omega$ is called a permutation group.

3.2.2 Setting up GI in this framework

Embedding of the action on vertices to the action on edges

As mentioned earlier, it is safe to assume that the two graphs are on the same vertex set V. Let V be the permutation space and let Sym(V) be the group

acting on V. In order to define GI in the group theoretic framework, we need to understand how this action embeds itself over another permutation domain, namely, the domain of edges. If we consider the set E_1 of edges to be our permutation domain, the symmetric group $\mathrm{Sym}(E_1)$ is of considerably larger size than $\mathrm{Sym}(V)$. Now, consider $\sigma \in Sym(V)$. The action of σ is defined over the domain V. We extend the action of σ over the domain E_1 in the following manner:
$e(\sigma(v_i), \sigma(v_j)) = \tau e(v_i, v_j)$
Now, it is trivial to show that any such τ is indeed a permutation. Due to this one to one map, we would only need to consider a small portion of $\mathrm{Sym}(E_1)$, which is precisely the embedding of $\mathrm{Sym}(V)$ on $\mathrm{Sym}(E_1)$. For simplicity, we will denote embedding of the action of any $\sigma \in Sym(V)$ as $\sigma\ e(v_i, v_j))$

Redefining GI

Let $X_1 = (V, E_1)$ and $X_2 = (V, E_2)$ be two graphs over the same vertex set V. The GI problem, in the group theoretic framework, is to determine whether $\exists \sigma \in \mathrm{Sym}(V)$ s.t. $X_2 = (V, \sigma(E_1))$.

3.2.3 The giants

This terminology is specific to Babai's Algorithm and can be explained using what we have already discussed. For a given permutation domain Ω, the groups $\mathrm{Sym}(\Omega)$ and $\mathrm{Alt}(\Omega)$ are called giants. The alternating group $\mathrm{Alt}(\Omega)$ is set of all even permutations of the members of the domain i.e. $\sigma \in Alt(\Omega)$ can be obtained by using even number of transpositions. The distinction whether the group whose action is being considered at any point during the algorithm, is a giant or not will be crucial to the timing of the algorithm.

Chapter 4

Trivalent Case

To proceed further, we would need to understand an important framework upon which Babai's Algorithm is based. This framework is the one developed by Eugene Luks in this paper [1]. Valence, here, can be used interchangeably with degree. This framework essentially takes forward the group theoretic framework we discussed in the previous chapter.
To understand and appreciate Luks's framework, we begin by trying to solve the problem for graphs having degree 3. We would later extend these arguments to degree n.

4.1 Reducing GI to Graph Automorphism

4.1.1 Automorphisms

An automorphism of a graph $X = (V, E)$ is an isomorphism to itself. It is denoted by $Aut(X)$ In other words, it is a permutation of the vertex set V that preserves the structure of the graph by mapping edges to edges and non-edges to non-edges.
$Aut(X) \subseteq Sym(V)$

4.1.2 The Graph Automorphism problem

Given a graph $X = (V, E)$. The Graph Automorphism problem requires us to determine a set of generators for $Aut(X)$. $\forall \sigma \in Aut(X)$, $X = (V, \sigma(E))$ where the action of σ on the set of edges is as defined previously. The following is a polynomial time reduction of GI to graph automorphism problem.

4.1.3 The reduction

Consider two graphs $X_1 = (V_1, E_1)$ and $X_2 = (V_2, E_2)$. We need to determine whether they are isomorphic. We proceed in the following manner. We take the disjoint union of the two graphs and construct another graph $X = X_1 \cup X_2$. We have $X = (V_1 \cup V_2, E_1 \cup E_2)$. We solve the automorphism problem on this graph X. Let us try to characterize the members of $Aut(X)$. Any member to $Aut(X_1)$ can be extended to $Aut(X)$ by letting it permute the elements of V_1 and having identity map on the elements of V_2. Similarly, we can extend any member of $Aut(X_2)$. An important observation here is that if $X_1 \cong X_2$, then $\exists\ \sigma \in Aut(X)$ which maps V_1 to V_2 and still preserves the structure of the disjoint graph. The converse is also true. Upon solving for automorphism, we pick every $\sigma \in Aut(X)$'s generator set, and check whether it maps V_1 to V_2, upon success, we say "YES" to GI otherwise "NO". Reduction complete.

4.2 Reducing the Trivalent case

4.2.1 Problem Statement

Consider two $X_1 = (V_1, E_1)$ and $X_2 = (V_2, E_2)$, each of degree 3. We are required to determine whether $X_1 \cong X_2$. One way to proceed is to reduce using the above methodology. But, here we are dealing with a special case of degree 3 and hence there is room for some exploitation.

4.2.2 A clever reduction

We observe that, if $X_1 \cong X_2$, then any given $e_1 \in E_1$ must be mapped to some $e_2 \in E_2$. Using this observation, we fix an egde $e_1 \in E_1$ and pick an edge $e_2 \in E_2$. We take one point on each of these edges (v_1 on e_1 and v_2 on e_2) and join the two points, call this new edge e. Let X be the resultant graph. Note that X will also be trivalent. We define $Aut_e(X) \subseteq Aut(X)$ as the subgroup consisting of all those $\sigma \in Aut(X)$ that fix the edge e. This group, as will see prove later turns out to be a 2-group for the trivalent case. Now, if there is indeed an isomorphism from X_1 to X_2, that maps e_1 to e_2, then $\exists\ \sigma \in Aut_e(X)$ that maps v_1 to v_2 and vice versa. Therefore, we solve for $Aut_e(X)$. We pick every $\sigma \in Aut(X)$'s generator set, and check whether it maps v_1 to v_2. If yes, we say "YES" to GI and terminate. If no, we pick another $e_2 \in E_2$ and repeat the process. If we exhaust all edges in E_2, we say "NO" to GI. Reduction complete.

4.3 Solving for $Aut_e(X)$

Consider a trivalent graph X, and an edge e on the graph. The subgroup $Aut_e(X)$ is determined iteratively using the following scheme. X_r is a subgraph of X consisting of all vertices and all edges of X which appear in paths of length $\leq r$ through e. Therefore, X_1 will simply be the edge e and X_{n-1} will be X itself.Notice that X_r will be embedded inside X_{r+1}, by the very definition. Also note that, we may not have to go until X_{n-1} to get the graph X. We now focus on $Aut_e(X_r)$'s. The groups are related via the following homomorphism:
$\pi_r : Aut_e(X_{r+1}) \to Aut_e(X_r)$

4.3.1 Understanding π_r

Consider $\sigma \in Aut_e(X_{r+1})$. The permutation domain here is the vertex set of $Aut_e(X_{r+1})$. Now, let us focus on the embedding of X_r inside X_{r+1}. Any $\sigma \in Aut_e(X_{r+1})$ would fix the edge and also preserve the structure around it as it is an automorphism. Therefore, all the vertices belonging to this embedding of X_r will have to be mapped among themselves by $\sigma \in Aut_e(X_{r+1})$. Also, the vertices outside this embedding would have to mapped among themselves to preserve the structure. So, if we restrict the domain of $\sigma \in Aut_e(X_{r+1})$ to $V(X_r)$, we know that resulting permutation would stay in that domain. This restriction is precisely what $\pi_r(\sigma)$ is.

4.3.2 Ker(π_r) and Img(π_r)

Let K be the set of generators for $\mathrm{Ker}(\pi_r)$. And I be set of pre-image for the set of generators of $\mathrm{Img}(\pi_r)$. Then, it is a trivial exercise to show that $K \cup I$ generates $Aut_e(X_{r+1}$. Note that, $Aut_e(X_1)$ is the trivial subgroup would be the starting point of out iteration. Thus, $\mathrm{Img}(\pi_1)$ would be a 2-group. This observation would come in handy as will realize that subsequent $\mathrm{Img}(\pi_r)$'s would be 2-groups.

4.3.3 A scheme for characterizing Ker(π_r)

From the above description of restriction, the action of any $\sigma \in Ker(\pi_r)$ on the embedded structure X_r would be the identity permutation. We focus our attention back to X_{r+1}. Consider $V' = \mathrm{V}(X_{r+1}) \setminus \mathrm{V}(X_r)$. Any vertex $v \in V'$ would be connected to either 1, 2 or 3 vertices in $\mathrm{V}(X_r)$. Not more than 3 because of trivalency. To codify this relationship, we define set A as the set of all subsets of $\mathrm{V}(X_r)$ of size 1, 2 or 3. We declare a function called

the father function f as follows:
$f : V' \to A$
The function takes as input a member of V' and returns who are its fathers from the set $\mathrm{V}(X_r)$. Note that, there can't be more than two vertices with the same set of fathers to preserve trivalency. Such vertices, if any, will be called twins. Take any $\sigma \in Aut(X_{r+1})$, and see it's action on $v \in V'$. As σ is an automorphism, $\sigma(v)$ should be a member of V' such that the structure is preserved. Thus, the following is implied:
$f\sigma(v) = \sigma(f(v))$
But, if $\sigma \in Ker(\pi_r)$, then $\sigma(f(v)) = f(v)$ as $f(v) \in V(X_r)$. Therefore, for $\sigma \in Ker(\pi_r)$:
$f(\sigma(v)) = f(v)$
This means that either, $\sigma(v) = v$ or v and $\sigma(v)$ are twins. Doing this, we have completely characterized the action of every $\sigma \in Ker(\pi_r)$ over the permutation domain $V(X_{r+1}$. For the embedded structure $V(X_r)$, the action is simply identity and for $V' = \mathrm{V}(X_{r+1}) \setminus \mathrm{V}(X_r)$, the action is to simply interchange the twins. Consider the group generated by the transpositions mapping a twin to its brother. This group is precisely the $\mathrm{Ker}(\pi_r)$, evident from the discussions mentioned above. It is easy to show that this group will be elementary abelian 2-group.

Chapter 5

Babai's Algorithm

5.1 Key aspects: Luks's Algo

The important takeaway from the previous discussion on Luks's Algorithm is that we reduce the GI problem to the Color Automorphism problem of a colored set A which is under the action of a group G. More specifically, we are required to find generators for the subgroup H of G that fixes the color classes. The algorithm solves this problem by breaking the set into orbits when the group action is transitive and into G-Blocks in the intransitive case. Notice that as we downsize our problem, the size of the group acting on the smaller set also reduces accordingly.

5.2 Consequences of hitting the barrier

As the problem is recursive, let us place ourselves into one the many instances that the initial problem broke into. We are working now with some subgroup G' of G that is acting on some subset A' of A. Now, when we attempt to break the problem further down, we realize that by the method we had been using, we can't get a quasipolynomially bounded number of instances as we desired. And thus, we have hit the barrier case!

5.3 Giant Representation

It turns out that when the barrier case is encountered, it is because the group G' is so large that we can find a homomorphism φ from G' to $\mathrm{Sym}(\Gamma)$, such that the image of G' is a giant where Γ is any arbitrary set that satisfies the following inequality:

$$polylog(\mid A' \mid) < \mid \Gamma \mid \leq \mid A' \mid$$

In the case when there is giant homomorphism as described above, we can establish the following using Lagrange's theorem:

$$\mid G' \mid \geq \mid Sym(\Gamma) \mid$$

This means that the size of G' has a lower bound as follows:

$$\mid G' \mid \geq 2^{|\Gamma|}$$

Now, as $\mid \Gamma \mid > polylog(\mid A' \mid)$, the above inequality implies that the size of G' doesn't have a quasipolynomial bound. Now we are required to work on this new set Γ and reduce it so that any giant representation of G' under a homomorphism has a quasipolynomially bounded size which would in turn ensure that the size of G' is quasipolynomially bounded.

5.4 Target Recurrence: Inside a barrier

Recall that we want a quasipolynomial run time of the algorithm. Also, recall that we are trying to find a subgroup of G' that fixes the color classes. In the barrier case, Babai's algorithms uses the inputs to trim G' in such a way that we bring it down to a subgroup of itself that is sure to contain the color stabilising subgroup. The following is the target recurrence:

$$f(n, m) \leq q(n) f(n, \tfrac{9m}{10})$$
$$f(n, polylog(n)) \leq q(n)^{\log n} f(\tfrac{9n}{10})$$
$$f(n) \leq q(n)^{(\log n)^2}$$

The idea behind the recursion is that first we use the properties of the input to trim G' such that its size attains a quasipolynomial bound. And once we achieve that, we simply use Luks's recursion to solve further.

5.5 Breaking the barrier

This section describes the series of steps that the algorithm uses in order to break the barrier.

5.5.1 Local Certificates

Recall that we are interested in computing the subgroup that stabilises the color classes of a colored set X. Since the only structure in the set X is the color classes, we can equivalently state the problem as follows:
Given a colored set X, and group G acting on the set, we are required to find $Aut_G(X)$.
Using this notation, we proceed to break the set Γ by constructing certificates of fullness and non-fullness.

Fullness and Non-Fullness Certificates

From the set Γ, we pick small a small subset T s. t. $(\mid T \mid = t) < \log \mid X \mid$. Let G_T be the setwise stabiliser of T in G. Consider the following equations:

$$\varphi : G \longrightarrow Giant(\Gamma)$$

$$\varphi_T : G_T \longrightarrow Giant(T) \ ??$$

We know that there is a giant representation of G, but the *question* whether there is a giant representation of G_T under restriction φ_T is what encodes fullness and non-fullness. If the answer is yes, we call the set T full and if the answer is no, we call the set non-full. We use the local certificates algorithm from the paper to construct these certificates.

5.5.2 Design Lemma

Once we know whether given subset T of size t is full or non-full, we construct t-ary relations inside the set Γ using the information whether a particular subset T is full of non-full. We then use the design lemma, which can be stated as follows:

Given a k-ary relation with symmetry defect $\geq \frac{1}{10}$, one can find, at $m^{\mathcal{O}(k)}$, either:

- A good canonical coloring (Breaking the set up into orbits)
- A good canonical equipartition (Breaking the set up in to G-Blocks)
- A canonically embedded regular graph

By good, we mean that it is consistent with our target recurrence. So, the cases 1 and 2 can be solved by Luks recursion. For solving the case 3 without distubing the recursion, we use the split or Johnson routine.

5.5.3 Split or Johnson

Given a non trivial regular graph $X = (V, E)$, we can find, at qusipolynomial multiplicative cost, either:

- A good canonical coloring of V
- A good canonical equipartition of V
- A canonically embedded Johnson graph

Note that cases 1 and 2 can be solve by Luks' recurrence but if we find a Johnson graph, it makes the reduction even stronger as we know that any automorphism must also preserve the structure of the johnson graph.

References

[1] Isomorphism of graphs of bounded valence can be tested in polynomial time http://www.sciencedirect.com/science/article/pii/0022000082900095

[2] Graph Isomorphism in Quasipolynomial Time https://arxiv.org/abs/1512.03547