Lecture Notes on

Algorithmic Algebra

Jayalal Sarma
Department of Computer Science and Engineering
IIT Madras, Chennai 600036

Draft—September 25, 2015 and forever

Preface

This lecture notes are produced as a part of the course *CS6842: Algorithmic Algebra* which was a course offered during August to November semester in 2013 and 2015 at the CSE Department of IIT Madras.

Acknowledgements

Thanks to Alexander Shrestov and Markus Blaser for creating nice templates for lecture notes which are being combined and in this document.

Scribe status

Each lecture has a field called **status**. It tells which stage of the edit pipeline is the document currently. Each scribe will be set to choose to either Monday-Tuesday (MT) or Wednesday-Friday (WF) pair of lectures. There are four versions.

1. Alpha (α) (deadline $X + 2$): Rough draft. The MT Scribe is expected to turn-in the rough notes by X+2 (Thursday night 10pm) and WF scribe is expected to turn this in by Sunday night 10pm.

2. Beta (β) (deadline $X+4$) : Final draft. The MT scribe is expected to turn this in by Saturday night 10pm and WF scribe is expected to turn this in by Tuesday night 10pm.

3. Gamma (γ) (deadline $X + 5$) : Corrected draft. This is done by the TAs and given to the instructor.

4. Delta (δ) (deadline $X + 6$) : Final notes. The instructor puts out this final version.

Even after these edits, it is possible that there are still errors in the draft, which may not get noticed. If you find errors still, please report at the git hub itself, or send emails to the instructor or the TAs.

Todo list

List of Scribes

Table of Contents

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: K Dinesh (*TA:* K Dinesh)
Date: Aug 3, 2015
Status: δ

Lecture
1

Introduction, Motivation and the Language

1.1 Overview of the course. Administrative, Academic policies

1.2 Introduction and Motivation

Main theme of this course is to use algebra to solve computational problems. Let us consider the following two problems :

Plagiarism check Given two C programs P_1 and P_2, check if they are the same under renaming of variables.

Molecule detection Given two chemical molecules check if they have the same structure.

Consider the following simpler variant of plagiarism checking where the program submitted has just its variables renamed and all other portions of the program are the same. In this case, given the two versions of the program, we need to check if there is a way to rename the variables of one program to get the other one.

In the second case one could view the given molecules as graphs. The problem is again similar problem, where we want to see if there is way to rename the vertex labels of the graph so that under the relabelling both the graphs are the same.

Our aim in both cases is to check whether the two structures are same under renaming. We are interested in solving this problem on graphs.

Definition 1.1 (Graph Isomorphism)**.** *Two graphs $X_1(V_1, E_1)$, $X_2(V_2, E_2)$ are said to be isomorphic if there is a bijective map $\sigma : V_1 \to V_2$ such that $\forall (u, v) \in V_1 \times V_1$,*

$$(u, v) \in E_1 \iff (\sigma(u), \sigma(v)) \in E_2$$

Problem 1.2. *The* graph isomorphism problem *is the decision problem of checking if given two graphs X_1, X_2 are isomorphic.*

We are also interested in the following special case of the above problem called graph automorphism problem.

Definition 1.3 (Graph Automorphism)**.** *For a graph $X(V, E)$, an automorphism of X is a renaming of the vertices of X given by a bijective map $\sigma : V \to V$ such that $\forall (u, v) \in V \times V$,*

$$(u, v) \in E \iff (\sigma(u), \sigma(v)) \in E$$

We are interested in the set of all bijections such that they are automorphisms of X. We denote this by $Aut(X)$.

Definition 1.4. *For a graph X on n vertices, $Aut(X) = \{\sigma \mid \sigma : [n] \to [n], \sigma$ is an automorphism of $X\}$*

Note that an identity map which takes a vertex to itself always belongs to $Aut(X)$ for all graphs X. Hence the question is : are there any bijections other than the identity map as automorphism of X.

Problem 1.5 (Graph Automorphism Problem)**.** *Given a graph X does $Aut(X)$ has any element other than the identity element.*

One way to see bijections is via permutations. This is because, every bijection define a permutation and vice versa.

Let X be an n vertex graph. Denote S_n to be the set of all permutations on n elements. Hence $Aut(X)$ can be defined as $\{\sigma \mid \sigma \in S_n \text{ and } \sigma \text{ is an automorphism of } G\}$.

We now show that the set $Aut(X)$ has some nice properties. Given $\sigma_1, \sigma_2 \in Aut(X)$, we can compose two permutations as $\sigma_1 \circ \sigma_2 = (\sigma_1(\sigma_2(1)), \sigma_1(\sigma_2(2)), \ldots, \sigma_1(\sigma_2(n)))$. This is same as applying σ_2 on identity permutation and then applying σ_1 to the result. We show that $Aut(X)$ along with the composition operation $\circ$ gives us many nice properties.

- If $\sigma_1, \sigma_2 \in Aut(X)$, then $\sigma_1 \circ \sigma_2$ is also an automorphism of X. The reason is that for any $(u, v) \in X \times X$, $(u, v) \in E \iff (\sigma_2(u), \sigma_2(v)) \in E$ Now applying σ_1 on the previous tuple, we get that $(\sigma_2(u), \sigma_2(v)) \iff (\sigma_1 \circ \sigma_2(u), \sigma_1 \circ \sigma_2(v)) \in E$. Hence $(u, v) \in E \iff (\sigma_1 \circ \sigma_2(u), \sigma_1 \circ \sigma_2(v)) \in E$. This tells that $Aut(X)$ is *closed* under $\circ$.
- The composition operation is also *associative*.
- *Identity* permutation belongs to $Aut(X)$ as observed before.
- Since we are considering bijections, it is natural to consider *inverse* for a permutation σ denoted σ^{-1} with the property that $\sigma \circ \sigma^{-1}$ is identity permutation.

We gave a definition of inverse for an arbitrary permutation. But then a natural question is : given $\sigma \in Aut(X)$, is it true that σ^{-1} also belongs to $Aut(X)$. It turns out that it is true.

Claim 1.6. *For the graph $X(V, E)$ $\sigma \in Aut(X) \iff \sigma^{-1} \in Aut(X)$*

Proof. Recall that $\sigma \in Aut(G)$ iff $\forall (u, v) \in V \times V$, $(u, v) \in E \iff (\sigma(u), \sigma(v)) \in E$. In particular, this must be true for $(\sigma^{-1}(u), \sigma^{-1}(v))$ also. This means,

$$(\sigma^{-1}(u), \sigma^{-1}(v)) \in E \iff (\sigma(\sigma^{-1}(u)), \sigma(\sigma^{-1}(v))) \in E$$

By definition of σ^{-1} we get that $(\sigma^{-1}(u), \sigma^{-1}(v)) \in E \iff (u, v) \in E$. This shows that $\sigma^{-1} \in Aut(X)$ □

Objects which satisfy these kind of properties are called groups.

1.3 Overview of the course

There are two major themes.

- Algorithms for permutation groups.
- Algorithms for polynomials.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: K Dinesh (*TA:* K Dinesh)
Date: Aug 4, 2015
Status: δ

Lecture 2

Algebraic Approach to Primality Testing

In this lecture, we will see an algebraic approach to solving a fundamental problem in Number Theory.

2.1 Application to Number Theory

Following is an algorithmic question that we are interested.

Problem 2.1. *Given a number n in its binary representation, check if it is a prime or not in time $O(poly(\log N))$.*

Note 2.2. *The trivial algorithms that we can think of will depend on n and hence takes time exponential in its input representation.*

Consider the following property about prime number proved by Fermat which is of interest in this context.

Theorem 2.3 (Fermat's Little Theorem)**.** *If N is a prime, then $\forall\ a, 1 \leq a \leq N-1$,*

$$a^N = a \mod N$$

Proof. Fix an $a \in \{1, 2, \ldots N-1\}$. Now consider the sequence $a, 2a, \ldots, (N-1)a$. The question we ask is : can any two of the numbers in this sequence be the same modulo N. We claim that this cannot happen. We give a proof by contradiction : suppose that there are two distinct r, s with $1 \leq r < s \leq N-1$ and $sa = ra \mod N$. Then clearly $N|(s-r)a$ which means $N|a$ or $N|(s-r)$. But both cannot happen as $a, s-r$ are strictly smaller than N.

This gives that all the N numbers in our list modulo N are distinct. Hence all numbers from $1, 2, \ldots, N-1$ appear in the list when we go modulo N. Taking product of the list and the list modulo N, we get

$$(N-1)!a^{N-1} = (N-1)! \mod N$$

By cancelling $(N-1)!$, we get that $a^{N-1} = 1 \mod N$. □

This tells that the above condition is necessary for a number to be prime. If this test is also sufficient (i.e, if the converse of the above theorem is true), we have a test for checking primality of

a number. But it turns out that this is not true due to the existence of Carmichael numbers which are not prime numbers but satisfy the above test.

So one want a necessary and sufficient condition which can be used for primality testing. For this, we need the notion of polynomials.

Definition 2.4. *A polynomial $p(x) = \sum_{i=0}^{d} a_i x^i$ with $a_d \neq 0$ denotes a polynomial in one variable x of degree d. Here a_is are called coefficients and each term excluding the coefficient is called a monomial.* ***A polynomial is said to be identically zero, if all the coefficients are zero.***

A polynomial time algorithm for this problem has been found in 2002 by Manindra Agarwal, Neeraj Kayal and Nitin Saxena (called as the AKS algorithm). In their result, they used the following polynomial characterization for a prime number.

Theorem 2.5 (Polynomial formulation (Agarwal-Biswas 1999))**.** *Let $N \geq 1$ be an integer. Define a polynomial $p_N(z) = (1+z)^N - 1 - z^N$. Then*

$$p_N(z) \equiv 0 (mod\ N) \iff N \text{ is prime}$$

Hence checking if N is prime or not boils down to checking if $p_N(z)$ is identically 0 or not except for the fact that the underlying operations are done modulo N.

Proof of the theorem is as follows.

Proof. Note that $p_N(z) = \sum_{i=1}^{N-1} \binom{N}{i} z^i$ and $\binom{N}{i} = \frac{N(N-1)\ldots(N-i+1)}{1 \cdot 2 \ldots i}$ with $1 \leq i \leq N-1$.

If N is prime, then $\binom{N}{i} = N \times k_i$ for some integer k_i as none of $1, 2, \ldots i$ divides N. Hence $\binom{N}{i} \mod N = 0$ for every i and $p_N(z) \equiv 0 \mod N$ since the polynomial has all coefficients as zero.

If N is composite, we need to show that $p_N(z) \not\equiv 0 (mod\ N)$ which means that there is at least one non-zero coefficient $p_N(z) \mod N$[1]. Since N is composite, there exists a prime p such that $p \mid N$. Let $p^k | N$ where $k \geq 1$ is the largest exponent of p in the prime factorization of n. Hence $p^{k+1} \nmid N$.

We first show that $p_N(z)$ is a non zero polynomial by showing that the coefficient of z^i for $i = p$, which is $\binom{N}{p}$, is non zero modulo p^k showing[2] $p_N(z)$ is not a zero polynomial modulo N. Let $N = g \times p^k$. In $\binom{N}{p}$, p of the denominator divides N. Note that p cannot divide g, for if it does, then $p^{k+1} | N$ which is not possible by choice of p and k.

$$\binom{N}{p} = \frac{g \times p^k (N-1)\ldots(N-p+1)}{1 \cdot 2 \ldots p} = \frac{g \times p^{k-1} (N-1)\ldots(N-p+1)}{1 \cdot 2 \ldots p-1} \tag{2.1}$$

Hence it must be that p^{k-1} divides $\binom{N}{p}$. This is because, there is no p left now in the denominator to divide N. Now $p^k \nmid \binom{N}{p}$ because, none of the terms $(N-1), \ldots, (N-p+1)$ can have p as a factor since all these terms are obtained by subtracting at most $p-1$ times from N. Hence $\binom{N}{p} \neq 0 \mod p^k$. This completes the proof. ∎

[1] In class we asked the following question of finding an $a \in \{1, 2, \ldots, N-1\}$ such that $p_N(a) \neq 0 \mod N$. This has a counter example. Consider the case where N is a Carmichael number. By definition, Carmichael numbers are composite numbers that satisfy Fermat's Little theorem. Hence if N is Carmichael, $\forall\ a \in \{1, 2, \ldots N-1\}$, we get $a^N = a \mod N$. This gives that $\forall\ a \in \{1, 2, \ldots, N-1\}$

$$p_N(a) = ((a+1)^N - a^N - 1) \mod N = ((a+1) - a - 1) \mod N$$

which is zero modulo N.

[2] Note that if $N | \binom{N}{i}$ then $p^k | \binom{N}{i}$. Taking contrapositive, we get that $\binom{N}{i} \neq 0 \mod p^k$ implies $\binom{N}{i} \neq 0 \mod N$

Note that Fermat's Little Theorem is a special case of Agarwal-Biswas formulation of primality testing.

Claim 2.6. *Fermat's Little Theorem is a special case of Agarwal-Biswas theorem.*

Proof. To prove Fermat's Little theorem, we need one direction of implication of Agarwal-Biswas theorem :

$$\text{"If } N \text{ is prime, then } (1+z)^N = (1+z^N) \mod N\text{"}.$$

Note that Fermat's Little theorem asks about $a^N \mod N$ for $a \in \{1, 2, \ldots, N-1\}$. Since the implication talks about $z^N \mod N$ and $(1+z)^N \mod N$, this suggests an induction strategy on a.

Let N be prime. We check the base case : for $a = 1$, the $1^N = 1 \mod N$. Hence the base case is true. By induction, suppose that $a^N = a \mod N$ for $a \in \{1, 2, \ldots, N-1\}$. Now,

$$\begin{aligned} (a+1)^N &= (a^N + 1) \mod N && \text{[By Agarwal-Biswas as } N \text{ is prime]} \\ &= (a+1) \mod N && \text{[By inductive hypothesis]} \end{aligned}$$

This completes the inductive case. □

This shows that checking if the polynomial $p_N(z)$ is a zero polynomial is an if and only if check for primality of N. So checking primality of a number now boils down to checking if a polynomial is identically zero or not. This is a fundamental problem of polynomial identity testing. We will be discussing about polynomial identity testing and AKS algorithm in our next theme.

Remark 2.7. *One could consider two possible definitions of a polynomial being identically zero and they are not equivalent. Indeed, if all coefficients of a polynomial are zeros then it evaluates to zero on all substitutions of the variables. However, the converse need not be true in all underlying algebraic structures. For example, consider the polynomials $x^p - x$ in a field $\mathbb{Z}_p$ (operations are $+$ and $\times$ modulo p). This is indeed a non-zero polynomial but all evaluations modulo p are zero.*

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: K Dinesh (*TA:* K Dinesh)
Date: Aug 5, 2015
Status: δ

Lecture 3

Algebraic Approach to finding Perfect Matchings in Graphs

3.1 Application to Graph algorithms

Consider the following problem of finding perfect matching.

Definition 3.1 (Finding Perfect Matching)**.** *Given a bipartite graph $G(V_1, V_2, E)$, we need to come up with an $E' \subseteq E$ such that $\forall u \in V_1 \cup V_2$, there is exactly one edge incident to it in E'.*

We shall give a polynomial formulation for the problem. Given $G(V_1, V_2, E)$ with vertex sets $|V_1| = |V_2| = n$. Define an $n \times n$ matrix A where $A(i, j) = 1$ if $(i, j) \in E$ and is 0 otherwise for all $(i, j) \in V_1 \times V_2$. Recall the determinant of A given by

$$det(A) = \sum_{\sigma \in S_n} sign(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)}$$

where

$$sign(\sigma) = \begin{cases} -1 & \text{if } inv(\sigma) \text{ is even} \\ 1 & \text{if } inv(\sigma) \text{ is odd} \end{cases}$$

and $inv(\sigma)$ is defined as $|\{(i, j) \mid i < j \text{ and } \sigma(i) > \sigma(j), 1 \leq i < j \leq n\}|$. We denote $f(x) \equiv 0$ to denote that polynomial $f(x)$ is the zero polynomial.

Lemma 3.2. *For the matrix A as defined as before, $det(A) \not\equiv 0 \Rightarrow G$ has a perfect matching*

Proof. Let $det(A) \not\equiv 0$. Hence there exists a $\sigma \in S_n$ such that $\prod_{i=1}^{n} A_{i,\sigma(i)} \neq 0$. Hence the edge set $E' = \{(i, \sigma(i)) \mid 1 \leq i \leq n\}$ exists in G and since $\sigma(i) = \sigma(j)$ iff $i = j$ for every i, j, E' form a perfect matching.

□

Note that converse of this statement is not true. For example, consider the bipartite graph whose A matrix is $A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. It can be verified that $det(A) = 0$ but the bipartite graph associated has a perfect matching.

Hence the next natural question, similar to our primality testing problem, is to ask for some kind of modification so that converse of previous lemma (lemma 3.2) is true. In the example considered, there were two perfect matchings in G having opposite sign due to which the determinant became 0. So the modification should ensure that perfect matchings of opposite signs does not cancel off in the determinant.

One way to achieve this is as follows. Define a matrix T as

$$T(i,j) = \begin{cases} x_{ij} & (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

where $(i,j) \in V_1 \times V_2$. This matrix is called as the Tutte matrix. Now, if we consider determinant of this matrix, we can see that the monomials corresponding a $\sigma \in S_n$ can be a product of at most n variables. Hence $det(T)$ is a polynomial in n^2 variables with degree at most n.

Also in the expansion of determinant, we can observe that each term picks exactly one entry from every row and column meaning each of the entries picked is from a distinct row and column. Hence each of the them is a bijection and hence is a permutation on n elements. Also corresponding to any permutation on n elements, we can get a term. This gives us the following observation.

Observation 3.3. *Set of monomials in $det(T)$ is in one-one correspondence with set of all permutations on n.*

We now give polynomial formulation for the problem of checking perfect matching in a bipartite graph.

Claim 3.4. *For the matrix T as defined before, $det(T) \not\equiv 0 \iff G$ has a perfect matching*

Proof. By $det(T) \equiv 0$, we mean that the polynomial $det(T)$ has all coefficients as zero. ($\Rightarrow$) Since $det(T) \not\equiv 0$, there exists a $\sigma \in S_n$ such that the monomial corresponding is non-zero and by definition of determinant it must be expressed as product of some n set of variables $\prod_i T(i, \sigma(i))$. The variable indices gives a matching and since there are n variables this is a perfect matching.

($\Leftarrow$) Suppose G has a perfect matching given by a M. Let τ denote the permutation corresponding to M. We now need to show that $det(T) \not\equiv 0$. To prove this, it suffices to show that there is a substitution to $det(T)$ which evaluates to a non-zero value.

Consider the following assignment, $\forall\ i, j$

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in M \\ 0 & \text{otherwise} \end{cases}$$

From the formula of determinant,

$$\begin{aligned} det(T) &= \sum_{\sigma \in S_n} sign(\sigma) \prod_{i=1}^{n} T_{i,\sigma(i)} \\ &= sign(\tau) \prod_{i=1}^{n} A_{i,\tau(i)} + \sum_{\sigma \in S_n \setminus \{\tau\}} sign(\sigma) \prod_{i=1}^{n} T_{i,\sigma(i)} \end{aligned}$$

Now substituting $x_{ij} = a_{ij}$, we get that the first term evaluates to $sign(\tau)$ since all the entries are 1. The second term evaluates to 0 since for all $\sigma \neq \tau$, there must be a j such that $\sigma(j) \neq \tau(j)$. Hence it must be that $a_{j,\sigma(j)} = 0$ and hence the product corresponding to σ goes to 0. ∎

Hence to check if the bipartite graph G has a perfect matching or not, it suffices to check if the polynomial $det(T)$ is identically zero or not. Checking if a polynomial is identically zero or not is one of the fundamental question in this area.

Note that this problem becomes easy if the polynomial is given as sum of monomial form. In most of the cases, the polynomial will not be given this way. For example if we consider our problem, we are just given the T matrix and $det(T)$ is the required polynomial. Trying to expand $det(T)$ and simplifying will involve dealing with $n!$ monomials which is not feasible.

Hence the computational question again boils down to checking if a polynomial is identically zero or not.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma M.N.
Scribe: Ramya C (*TA:* Ramya C)
Date: Aug 7, 2015
Status: δ

Lecture
4

Graphs, Groups and Generators

In this lecture we will pose three graph theoretic questions and find answers using approaches in algebra.

4.1 Graph Isomorphism, Automorphism and Rigidity

Definition 4.1. *(Graph Isomorphism.) Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be graphs. We say $G_1 \cong G_2$ (read as G_1 is* isomorphic to G_2*)* if there exists a bijection $\sigma : V_1 \to V_2$ such that $\forall (u, v) \in V_1 \times V_2$ we have

$$(u, v) \in E_1 \iff (\sigma(u), \sigma(v)) \in E_2$$

In other words, we say a graph G_1 is isomorphic to G_2 if there exists a relabeling of the vertices in G_1 such that the the adjacency and non-adjacency relationships in G_2 is preserved.

Observation 4.2. *If $|V_1| \neq |V_2|$,then G_1 is not isomorphic to G_2.*

The graph isomorphism problem is stated as follows.

Problem 4.3 (Graph Isomorphism Problem). *Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, test if $G_1 \cong G_2$ or not.*

A natural question to ask in this setting is that if there is an isomorphism from a graph G to itself.

Let $[n] = \{1, 2, \ldots, n\}$. Let S_n denote the set of all permutations from the set $[n]$ to $[n]$. Let $G = (V, E)$ be a graph. An automorphism of G is a bijection $\sigma : V \to V$ such that $\sigma(G) = G$. Let

$$Aut(G) = \{\sigma | \sigma \in S_n \text{ and } \sigma(G) = G\}$$

be the set of all automorphisms of G. Ideally, we would like to compute the set of automorphism of a graph to itself.

Problem 4.4 (Graph Automorphism Problem). *Given a graph G, list the elements of $Aut(G)$.*

The above problem can be expected to be solved in polynomial time only if the output expected is polynomial in length. This brings up the size of the $Aut(G)$ into question. Unfortunately, if G is the complete graph on n vertices. Then $|Aut(G)| = n!$. Hence the above question is not well-formulated.

Since identity permutation is trivially an automorphism for any graph, the $Aut(G)$ is always a non-empty subset of S_n where n is the number of vertices. Hence, one can ask a natural decision variant of the above problem, namely the graph rigidity problem.

Formally, the graph rigidity problem is stated as follows.

Problem 4.5 (Graph Rigidity Problem). *Given a graph G, test if $Aut(G)$ is trivial. That is, whether $Aut(G)$ contains only the identity permutation or not.*

More than the size, in the first lecture of this course, we have seen that $Aut(G)$ forms a *subgroup* of S_n. To utilize this structure, we first refresh the definition of an abstract group. From now on, we will use the letter X to denote a graph and G to denote a group.

4.2 Groups and Generators

Definition 4.6. *(Groups.) A set G together with a binary operation $*$ is said to be a group if the following four conditions are met*

- ***Closure** : $a, b \in G$, the element $a * b \in G$.*
- ***Associative** : For any $a, b, c \in G$, we have $(a * b) * c = a * (b * c)$.*
- ***Existence of Identity** : For any $a \in G$ there exists a unique element $e \in G$ such that $a * e = e * a = a$.*
- ***Existence of Inverse** : For any $a \in G$ there exists a unique element $b \in G$(denoted by a^{-1}) such that $a * b = b * a = e$.*

Example 4.7.
- *S_n forms a group under composition.*
- *$(\mathbb{Z}_5, +)$ is a group.*

From now on, we will use the letter X to denote a graph and G to denote a group.

Remark 4.8. *Let $(G, *)$ be a group. Let $H \subseteq G$ such that $(H, *)$ also forms a group. We say H is a* subgroup *of G and denote by $H \leq G$.*

Exercise 4.9. *For any graph X, the set $Aut(X)$ forms a group under the composition operation. That is, $Aut(G) \leq M$ where $M = (S_n, \circ)$ is the permutation group.*

Let $(G, *)$ be a group. Let $H \subseteq G$ such that $(H, *)$ also forms a group. We say H is a *subgroup* of G and denote by $H \leq G$. We showed in the first lecture of the course that, for any graph X, the set $Aut(X)$ forms a group under the composition operation. That is, $Aut(G) \leq M$ where $M = (S_n, \circ)$ is the symmetric group.

Let $(G, +)$ be a finite group and $g \in G$ be an element. Let $g^2 = g * g, g^3 = g * g * g$. Similarly $g^k = \underbrace{g * g * \cdots * g}_{\text{k times}}$. Now consider the set $H = \{g, g^2, g^3, \ldots\}$. Since $(G, *)$ is a finite group there must exist a k such that $g^k = g$ in H.

Lemma 4.10. *Let $(G, +)$ be a finite group and $g \in G$ be an element. Let $H = \{g, g^2, g^3, \ldots\}$ be a set of elements. The unique identity e of G is in H.*

Proof. Since $(G, *)$ is a finite group there must exist a k such that $g^k = g$ in H. By definition, $g^k = g^{k-1} * g = g$. As $(G, *)$ is a group, g^{-1} exists in G.
Therefore,

$$g^{k-1} * g * g^{-1} = g * g^{-1} = e$$

□

Definition 4.11 (Generator). *Let $(G, +)$ be a finite group and $g \in G$ be an element. We say an element $g \in G$ is a generator of the set H if for every element $h \in H$ there exists a m such that $h = g^m$. (denoted by $H = \langle g \rangle$).*

Observation 4.12. *$H = \langle g \rangle$ is a subgroup of G. That is, $H \leq G$.*

A quick example is that 1 is a generator for $(\mathbb{Z}_5, +)$

Definition 4.13. *An group $(G, *)$ that can be generated by a single element is called a* cyclic group. *For instance, $(\mathbb{Z}_5, +)$.*

Not every group is cyclic. For instance, one can verify that S_3 (with 6 elements in it) is not cyclic.

Definition 4.14 (Generating Set). *Let $S \subseteq G$ be the set $\{u_1, \ldots, u_k\}$. S is said to be* generating *if $\langle S \rangle = G$.*

Having observed that $Aut(X)$ could have potentially be of exponential size, a reasonable way to formulate the graph automorphism problem is in terms of the generating set of the group. For this, we establish first that $Aut(X)$ have a generating set of size $\mathsf{poly}(n)$? In fact any group has !.

4.2.1 Lagrange's theorem

Let $(G, *)$ be a group. Let $H \leq G$. For any $g \in G$, define the right coset of H in G to be

$$Hg = \{hg \mid h \in H\}$$

Let $g_1, g_2 \in G$ and Hg_1, Hg_2 be the corresponding right cosets. Are there elements that belong to more than one coset of H in G? Is $Hg_1 \cap Hg_2 \neq \phi$? If yes, then the set of right cosets of H in G form a partition of the ground set of G. In that case, how many such cosets are required to cover the entire set G ? Let us answer these two questions.

Lemma 4.15. *Let $g_1, g_2 \in G$ and $Hg_1 = \{hg_1 \mid h \in H\}, Hg_2 = \{hg_2 \mid h \in H\}$. Then*

$$Hg_1 = Hg_2 \text{ or } Hg_1 \cap Hg_2 = \phi.$$

Proof. If $g_1 = g_2$, then by definition $Hg_1 = Hg_2$. Therefore let $g_1 \neq g_2$. We will prove : If $Hg_1 \cap Hg_2 \neq \phi$ then $Hg_1 = Hg_2$. Let $Hg_1 \cap Hg_2 \neq \phi, g \in Hg_1 \cap Hg_2$ we will show

(i) $Hg_1 \subseteq Hg_2$; and

(ii) $Hg_2 \subseteq Hg_1$.

Since $g \in Hg_1$ we know that there exists a $h_1 \in H$ such that $g = h_1 g_1$. Similarly $g \in Hg_2$ suggests that there exists a $h_2 \in H$ such that $g = h_2 g_2$.

$$h_1 g_1 = h_2 g_2 = g$$

As $(H, *)$ is a group by itself, h_1^{-1} and h_2^{-1} exists.

$$g_1 = h_1^{-1} h_2 g_2 \tag{4.2}$$

$$g_2 = h_2^{-1} h_1 g_1 \tag{4.3}$$

(i) $Hg_1 \subseteq Hg_2$
Let $g' \in Hg_1$. This implies there exists a $h' \in H$ such that $g' = h' g_1$. Therefore,

$$\begin{aligned} g' &= h' g_1 \\ g' &= h'(h_1^{-1} h_2 g_2) \qquad \text{[By equation (4.2)]} \end{aligned}$$

By closure property in $(H, *)$, we have $h'' = h' h_1^{-1} h_2 \in H$. Therefore $g' = h'' g_2$, $g' \in Hg_2$.

(ii) $Hg_2 \subseteq Hg_1$
Let $g' \in Hg_2$. This implies there exists a $h' \in H$ such that $g' = h' g_2$. Therefore,

$$\begin{aligned} g' &= h' g_2 \\ g' &= h'(h_2^{-1} h_1 g_1) \qquad \text{[By equation (4.3)]} \end{aligned}$$

By closure property in $(H, *)$, we have $h'' = h' h_2^{-1} h_1 \in H$. Therefore $g' = h'' g_1$, $g' \in Hg_1$.

□

Lemma 4.16. *For every $g \in G$, $|Hg| = |H|$.*

Proof. By construction, for every element in H there exists an element in Hg. So $|Hg| \leq |H|$. We first argue that $|H| \leq |Hg|$. Suppose not. Let $|Hg| < |H|$. Then there exists $h_1, h_2 \in H, h_1 \neq h_2$ such that $h_1 g = h_2 g$. Since $(G, *)$ is a group, g^{-1} exists. We have $h_1 g g^{-1} = h_2 g g^{-1}$ which implies $h_1 = h_2$, a contradiction. □

Theorem 4.17 (Lagrange's Theorem). *Let $(G, *)$ be a group and $H \leq G$. Then $|H|$ divides $|G|$.*

Proof. Direct consequence of Lemmas 4.15 and 4.16 □

Observation 4.18. *Let $H =< g >$ and $H' =< H, g' >$ where $g' \in G \backslash H$. Then $H \leq H' \leq G$. We have $g \in H' \backslash H$, therefore $|H'| > |H|$. $H' \leq H$. By Theorem 4.17, $|H'| \geq 2|H|$. This shows that every group has a generating set of size* $\log |G|$.

Remark 4.19. *We know that $Aut(X) \leq S_n$ for any graph $X = (V, E)$. By Observation 4.18 $Aut(X)$ has a generating set of size* $\log |S_n| = log(n!) \in \mathcal{O}(n \log n)$.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Ameya Panse (*TA:* Ramya C)
Date: August, 10 2015
Status: δ

Lecture 5

Orbit-Stabilizer Lemma

We first start with some notations and definitions. Order of a group G is number of elements in the group, that is $|G|$. Let $H \leq G$ and $g \in G$. The right coset of H in G is defined as $Hg = \{hg \mid h \in H\}$. If the multiplication is on the left, we call it the left coset. That is, $gH = \{gh \mid h \in H\}$. In general, it is not necessary that the left and right cosets are the same. If H is a group such that left and right cosets are the same element-wise, (that is, $\forall g \in G, Hg = gH$), H is a said to be a normal subgroup of G.

Let H be a normal subgroup of G. Choose one element from each of these as a representative of the set (say $[g]$ denote the coset representative of the coset $Hg = gH$). These elements have a group structure among them. This requires a proof, which we will come to in the next few lectures.

5.1 Group Action and Orbits

Let G be a Subgroup of S_n. Let $\alpha \in [n]$ and $g \in G$. We denote by α^g is the image of α under the permutation g. Orbit of an element α in G is the set of elements it gets mapped to under permutations in G. More formally,

Definition 5.1 (**Orbit of α in G**). *The orbit of α in G is defined as*

$$\alpha^G = \{\alpha^g \mid g \in G\}$$

This defines a natural relation among elements of $[n]$.

$$\alpha \backsim \beta \leftrightarrow \exists g \in G, \alpha^g = \beta$$

We check that this is an equivalence relation. $e \in G$, where e is the identity element. Hence, $\alpha \backsim \alpha$. Let $\alpha \backsim \beta$. Thus $\exists g \in G, \alpha^g = \beta$. Hence, $\alpha = \beta^{g^{-1}}$. Thus, $\beta \backsim \alpha$. The relation $\backsim$ is an equivalence relation. For transitivity, let $\alpha \backsim \beta, \beta \backsim \gamma$. By definition, $\exists g_1, g_2 \alpha^{g_1} = \beta, \beta^{g_2} = \gamma$. By composition of permutations, $(\alpha^{g_1})^{g_2} = \gamma$. Hence, $\alpha \backsim \gamma$.

The stabilizer of α in G is

$$G_\alpha = \{g \mid \alpha^g = \alpha\}$$

. This is the set of elements in G which sends α to α itself.

5.1.1 Orbit-Stabilizer Lemma

Is there any connection between the number of permutations in G that fixes α and the number of elements to which α can be taken to? The orbit stabilizer lemma, which is an easy consequences of Lagrange's theorem gives a neat answer.

Lemma 5.2 (**Orbit-Stabilizer Lemma**). *Let $G \leq S_n$. Then for any $\alpha \in [n]$,*

$$|\alpha^G| * |G_\alpha| = |G|$$

Proof. We quickly observe that G_α is a Subgroup of G. Indeed, identity belongs to G_α trivially. If g and g' both fixes α, then so does gg' and $g'g$. If g fixes α, then so does g^{-1}. Since G_α forms a Subgroup of G, by Lagrange's Theorem,

$$\frac{|G|}{|G_\alpha|} = \text{number of distinct right cosets of } G_\alpha \text{ in } G$$

To complete the proof, it suffices to argue that the number if distinct cosets of G_α in G is the size of the orbit of α under the action of G. We now show the bijection.

Let $\beta \in \alpha^G$ via $h \in G$. That is,$\alpha^h = \beta$. Consider the map,

$$\Gamma : \beta \to \{g \in G | \alpha^g = \beta\}$$

We first show that this is well-defined map between the elements in the orbit of α to the cosets. For that, we first show that $\{g \in G | \alpha^g = \beta\}$ is indeed a right coset of G_α in G. To argue this, it suffices to show that

$$\begin{aligned}\Gamma(\beta) = \{g \in G \mid \beta = \alpha^g\} = \{g \in G \mid \alpha^h = \alpha^g\} = \{g \in G \mid \alpha^{gh^{-1}} = \alpha\} \\ = \{g \in G \mid gh^{-1} \in G_\alpha\} = \{g \in G \mid g \in G_\alpha h = G_\alpha h\end{aligned}$$

Thus Γ is a function.

We now show that the function Γ is injective. Let β and γ be two different elements of the orbit of α via the group elements h_β and h_γ. That is,

$$\Gamma(\beta) = \{g \in G \mid \alpha^g = \beta\}$$

$$\Gamma(\gamma) = \{g \in G \mid \alpha^g = \gamma\}$$

Indeed, these two sets cannot have an intersection since $\beta \neq \gamma$. Thus, $\Gamma(\beta) \neq \Gamma(\gamma)$.

We now argue that the function Γ is surjective. Consider any coset $C = G_\alpha g$ of G_α in G, where $g \in G$. We show that there is a $\beta \in \alpha^G$ such that $\Gamma(\beta) = C$. Indeed, define β to be α^g. Consider any $h \in G$ such that $\alpha^h = \beta$.

1: JS says: Surjectivity proof to be completed

□

5.2 Graph Automorphism and Graph Isomorphism

We define the problems that we are interested in, technically.

Problem 5.3 (Graph Isomorphism Problem (GI)). *Given a graph $X_1 = (V_1, E_1)$ and $X_2 = (V_2, E_2)$, decide if $X_1 \cong X_2$ or not.*

Problem 5.4 (Graph Automorphism Problem (GA)). *Given a graph $X = (V, E)$, compute a Generating Set for $Aut(X)$.*

Problem 5.5 (Graph Rigidity Problem (GR)). *Given a graph $X = (V, E)$, decide if $Aut(X)$ is trivial or not.*

Problem 5.6 (Counting Isomorphisms (#GI)). *Given a graph $X_1 = (V_1, E_1), X_2 = (V_2, E_2)$, output the number of Isomorphisms from X_1 to X_2, in binary.*

Problem 5.7 (Counting Automorphisms (#GA)). *Given a graph $X = (V, E)$, compute the size of the automorphism group, $|Aut(X)|$, in binary.*

Problem 5.8 (Computing the Isomorphism (ISO)). *Given a graph $X_1 = (V_1, E_1), X_2 = (V_2, E_2)$, output an isomorphism map between V_1 and V_2 if it exists.*

We use the notion of polynomial time reducibility between two problems. Given two problems A, B, we say that $A \leq B$ (A reduces to B), if given a polynomial time (in terms of input size n) algorithm for B, we can give out a polynomial time algorithm for A. We quickly observe some easy relationship among these problems. Indeed, if we can solve GA, we can solve GR as well. Given a graph X, to check if it is rigid, it is only a matter of checking if there is a nontrivial element in the generating set for the group $Aut(X)$. Hence we conclude that $GR \leq GA$. The same is the case for GR and $\#GA$ as well. That is, $GR \leq \#GA$. Similarly, if we can compute the isomorphism, we can decide it as well. Trivially, $GI \leq ISO$. It is interesting to think about whether $\#GA$ can be done using GA. That is, given the generating set of a permutation group (that is, subgroup of S_n), can we compute the size of the generated group?

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Ameya Panse (*TA:* Ramya C)
Date: August, 11 2015
Status: δ

Lecture
6

Reduction Between Variants of GI

Our aim is to understand the relationship between various problems related to graph isomorphism that we listed in the last lecture. To quote the names, we talked about, graph isomorphism problem (GI), graph automorphism problem (GA), graph rigidity problem (GR), counting isomorphisms (#GI), counting automorphisms (GA), and computing the isomorphism (Iso). As a main tool towards understanding these, we now introduced a colored version of graph isomorphism.

6.1 Colored Graph Isomorphism Problem

The driving thought is the following. Consider the following problem. Given two graphs X_1 and X_2, and consider a vertex $u \in V_1$ and $v \in V_2$. Can we test if there is an isomorphism between X_1 and X_2 that maps u to v itself? To create a terminology, the scenario can also be described as : we will color vertex u and v with a color (say *blue*), the rest of the vertices in both X_1 and X_2 as red, and ask if there is a *color preserving isomorphism* between the graphs.

More formally, let $X = (V, E)$ be a graph. Consider a coloring function function $\Psi : V(X) \to [c]$, where $i \in [c]$ denotes a color. Thus, for a vertex v, $\Psi(v)$ denotes its color. We call the set of vertices $\Psi^{-1}(i)$ as the i^{th} color class, and we call the graph as c-colored.

Problem 6.1 (Colored Graph Isomorphism (CGI))**.** *Given two c-colored graphs (X_1, Ψ_1) and (X_2, Ψ_2) decide if there exists $\sigma : V(X_1) \to V(X_2)$ such that*

- *for all $(u, v) \in V(X_1) \times V(X_2)$, $(u, v) \in E(X_1)$ if and only if $(\sigma(u), \sigma(v)) \in E(X_2)$*
- *for every $u \in V(X_1), \Psi_1(u) = \Psi_2(\sigma(u))$.*

How hard can colored graph isomorphism be? In general can there be an efficient algorithm which solves CGI for any c? Indeed, an easy observation is that GI is a special case when $c = 1$. That is, give all vertices in the graphs the same color so that any isomorphism preserves the colors. Thus, CGI seems harder when number of vertices in a color class is allowed to be large. Indeed, later in the course, when the number of vertices in any color class is bounded by a constant, we will give a polynomial time algorithm for the problem. However, without any restrictions the problem is as hard as graph isomorphism. However, what about the cases when $c > 1$. at a first sight, they don't seem easier than GI. We start by showing that they are not harder for sure.

6.1.1 Gadget Trick : From Colored GI to Non-colored GI

We show that $CGI \leq GI$. That task is as follows, given (X_1, Ψ_1) and (X_2, Ψ_2) construct graphs X_1' and X_2' (uncolored graphs) such that

$$((X_1, \Psi_1), (X_2, \Psi_2)) \in CGI \iff (X_1', X_2') \in GI$$

The construction of X_1' and X_2' are as follows : For every vertex $u \in V(X_1)$ such that $u \in \Psi_1^{-1}(i)$:

1. Add ni extra vertices to X_1.
2. Add edges from each of the extra vertices to u.

Do the same for (X_2, Ψ_2) to get X_2'. This completes the reduction.

We denote by Y_u the extra vertices that we added for the vertex u. Notice that the degree of all extra vertices added is 1 each, and the degree of all original vertices in color class i ($i \geq 1$) is at least ni. The number of vertices in the new graph produced is at most $n + \sum_{i=1}^{c} ni \leq O(n^2)$. The reduction runs in polynomial time since we can construct the graphs X_1' and X_2' in polynomial time. Thus the only thing remaining is to prove the correctness of the reduction which we do by the following claim.

Claim 6.2. $((X_1, \Psi_1), (X_2, \Psi_2)) \in CGI \iff (X_1', X_2') \in GI$

Proof. ($\Longrightarrow$) Let $((X_1, \Psi_1), (X_2, \Psi_2)) \in CGI$. We need to show that $X_1' \cong X_2'$. From the assumption, we know that there exists $\sigma : V(X_1) \to V(X_2)$ such that :

1. $\forall (u, v) \in V(X_1) \times V(X_2), (u, v) \in E(X_1)$ if and only if $(\sigma(u), \sigma(v)) \in E(X_2)$.
2. $\forall u \in V(X_1), \Psi_1(u) = \Psi_2(\sigma(u))$.

To extend this to an isomorphism σ' between X_1' and X_2', we need to define the images of the extra vertices that we added in the above construction. However, since the σ is color preserving, we know that for any $u \in V(X_1$, $|Y_u| = |Y_{\sigma(u)}|$. Extend σ to σ' by choosing any bijection between the vertices Y_u and $Y_{\sigma(u)}$.

We now argue that σ' is an isomorphism. Observe that, for any $u \in V(X_1)$, the only edges incident on Y_u are of the form (a, u) where $a \in Y_u$. Consider the pair $(\sigma'(a), \sigma'(u)$ which is same as $(a', \sigma(u)$ where $a' \in Y_{\sigma(u)}$. This is an edge in X_2' by construction. The converse of the above implications also hold, and hence $X_1' \cong X_2'$.
($\Longleftarrow$) Suppose $X_1' \cong X_2'$. We need to show that $(X_1, \Psi_1) \cong (X_2, \Psi_2)$. By assumption, there is a bijection $\sigma : V(X_1') \to V(X_2')$ such that $\forall (u, v) \in E(X_1'), (\sigma(u), \sigma(v)) \in E(X_2')$.

To begin with, we argue that σ must map elements of $V(X_1)$ to $V(X_2)$ itself. This is simply because the degrees of the vertices outside $V(X_2)$ are all 1 and the degree of all vertices in $V(X_1)$ in the graph X_1' are all at least n. Since σ is originally an isomorphism, σ restricted to $V(X_1)$ immediately gives an isomorphism between X_1 and X_2.

We now need to argue that the map σ preserves color. Suppose not. Let $u \notin \Psi_2^{-1}(i)$. Hence $\sigma(u) \in \Psi_2^{-1}(j)$ and $j \neq i$. Note that $ni + n > deg(u) \geq ni$. This implies $n(i+1) > deg(u) \geq ni$. And, $\sigma(u) \in \Psi_2^{-1}(j) \Rightarrow n(j+1) > deg(u) \geq nj$. Since both of these can not be true simultaneously, we have a contradiction. Hence, $\sigma(u) \in \Psi_2^{-1}(i)$. Thus, $(X_1, \Psi_1) \cong (X_2, \Psi_2)$. ∎

6.1.2 Computing Isomorphism $\leq$ GraphIso

We show that ISO $\leq CGI$. As $CGI \leq GI$ we have ISO $\leq GI$.

Given X_1, X_2 and oracle access to a black box which checks for isomorphism between any two given graphs, output a permutation that maps X_1 to x_2.

The Reduction is as follows:

1. Check if $X_1 \cong X_2$. If NO, end.
2. For each vertex vertex $i_1 \in [n] = V_1$
   Get X_1' by coloring i_1 with color i_1.
   - Get X_2' by coloring $i_2 \in V_2$ with color i_1.
   - Query CGI to check if $X_1' \cong X_2'$.
   - Remove colors from X_2' and repeat the last two steps for $i_2 \in V_2$ until we get a "yes" answer. For the i_2 on which we get a "yes" answer, fix the color as i_1 and do not reuse color i_1 again in the outer loop.
3. Output the permutation.

To see the correctness : suppose the graphs are isomorphic. That is the algorithm reaches step (2), 3rd sub part is guaranteed to find a j for each vertex i. Observing that we will color a vertex on the left and the right with some color i_1 on if we are sure that there is a color preserving isomorphism. Hence the isomorphism map, if exists, will be found. To see the running time, observe that we make at most $O(n^2)$ queries to CGI which in turn makes a single query each to GI. Thus we can solve ISO using at most $O(n^2)$ queries to GI.

6.1.3 GraphIso $\leq$ GraphAuto

We need to construct a graph G such that the generating set S of $Aut(G)$ enables to decide if $X_1 \equiv X_2$.

Assume that the graphs are connected to begin with. The reduction is simply to take $X = X_1 \cup X_2$. Let S be the generating set of the reduction, check if there exists a $\sigma \in S$ such that σ maps at least one vertex in X_1 to a vertex in X_2. We directly show the correctness of the reduction.

Claim 6.3. *$X_1 \cong X_2$ if and only if there exists a $\sigma \in S$ and $v \in V(X_1)$ such that $\sigma(v) \in V(X_2)$.*

Proof. ($\Rightarrow$) Suppose $X_1 \cong X_2$. Then there exists a τ which is an isomorphism from X_1, to X_2. $\tau \in Aut(X)$. Hence, there is a σ that maps a vertex in X_1 to a vertex in X_2.

($\Leftarrow$) Let there exist a $\sigma \in S$ such that σ maps $u \in X_1$ to a vertex in X_2. Let $v \in X_1$ be such that $\sigma(u) \in X_1$. Since X_1 is connected u, v are connected. But $\sigma(u) \in X_2, \sigma(v) \in X_1$ are not connected which is a contradiction. Therefore for $\sigma \in S$, σ maps all the vertices in X_1 to X_2. Thus $X_1 \cong X_2$. □

What do we do if the graphs are not connected? We simply add an extra vertex each to both the graphs that is adjacent to all the vertices in the corresponding graphs. Since the new vertices have degree n, they can only be mapped to each other by any isomorphism (or an automorphism of X). Thus GI $\leq$ GA.

Exercise 6.4 (Problem Set 1, Question 1)**.** *We will reduce variants of graph isomorphism problem to the original problem.*

(a) We defined the graph rigidity problem GR *as - given a graph X, test if $Aut(X)$ is trivial. Show that* $\text{GR} \leq \text{GI}$.

(b) Notice that the GI *is defined for undirected graphs. Define the graph isomorphism problem for directed graphs* DirGI. *Show that* $\text{DirGI} \leq \text{GI}$.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Sahil Sharma (*TA:* K Dinesh)
Date: August, 12 2015
Status: δ

Lecture 7

Reduction of $\mathcal{GA}$ to $\mathcal{GI}$

7.1 Recap and Lecture overview

Let us denote by $\mathsf{Color}-\mathsf{GI}$ the problem of finding whether there is a coloring preserving isomorphism. In the previous few lectures we showed that $\mathsf{Color}-\mathsf{GI} \leq \mathcal{GI}$. We also showed that $\mathcal{GI} \leq \mathcal{GA}$. We also showed how to compute an isomorphism map between two graphs (COMPUTE-ISO) if we can check if two graphs are isomorphic. Recall that $Aut(X)$ is the group of all automorphisms of a given graph X and $\mathcal{GA}$ is the problem of obtaining a generator for $Aut(X)$. In this lecture, we show that $\mathcal{GA} \leq \mathcal{GI}$. That is, given a subroutine to solve $\mathcal{GI}$, we show how we can compute a generator set of $Aut(X)$ for an input graph X. Along with $\mathcal{GI} \leq \mathcal{GA}$ proved in the earlier class, this shows that the graph isomorphism and graph automorphism problems are equivalent in terms of hardness.

The main idea is that there is a unique way to express an element in G using Tower of subgroups of G.

7.2 Solving Graph Automorphism using Graph Isomorphism

7.2.1 Tower of Subgroups of Group

Definition 7.1 (Tower of Subgroups of G)**.** *Let $k \in \mathbb{N}$. For a group G, the k subgroups of G namely $G^{(1)}, G^{(2)}, \ldots, G^{(k)} = \{id\}$ is said to form a tower of subgroups of G if*

$$G = G^{(0)} \geq G^{(1)} \geq, \ldots, G^{(k)} = \{id\}$$

The above concept is defined for an arbitrary group, but we will use this specifically for understanding about Tower of subgroups of $Aut(X)$, for a given graph X.

Note that in the sequence, $G^{(i)}$ is a subgroup of $G^{(i-1)}$ for $1 \leq i \leq k$. Hence there is a coset structure that $G^{(i)}$ generates (or induces) in $G^{(i-1)}$ and we seek to exploit this structure to get a $O(n \log n)$ sized generating set of $Aut(X)$ efficiently, given a procedure for solving $\mathcal{GI}$.

Definition 7.2. *For a group G and a subgroup H of G denote $H : G$ to denote the set of cosets of H in G and the number of cosets in $H : G$, denoted, $[H : G]$ is called as the index of H in G.*

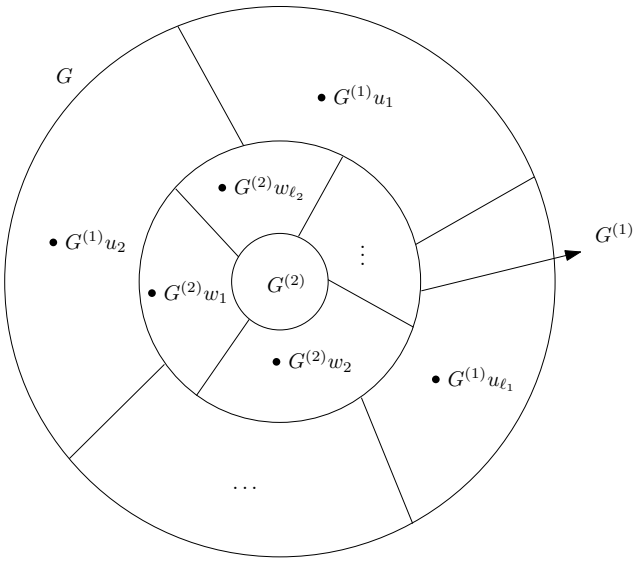

FIGURE 7.1: Tower of groups of G for two levels with cosets in the first level fixed as $u_1, u_2, \ldots, u_{\ell_1}$ and second level fixed as $w_1, w_2, \ldots, w_{\ell_2}$.

For example, for $1 \leq i \leq k$, $[G^{(i+1)} : G^{(i)}]$ denotes the number of cosets of induced in $G^{(i)}$ by $G^{(i+1)}$. Denote this number by ℓ_i. Note that by Lagrange's theorem $\ell_i = \frac{|G^{(i)}|}{|G^{(i+1)}|}$. Hence $\prod_i \ell_i = |G^{(0)}| = |G|$.

Recall the notion of a *coset representative* of a coset, with respect a given group and a subgroup of the given group. For example, in the figure 7.1, consider the subgroup $G^{(1)}$ of G. Here we choose u_1 as the coset representative of the coset $G^{(1)}u_1$.

Note that any arbitrarily chosen element of that coset $G^{(1)}u_1$ can be made its representative due to the following reason. For any two elements in the same coset, say g and g' we have $G^{(1)}g = G^{(1)}g'$, by the definition of the coset (since it generates the same coset)[3]. Hence, there is nothing special about u_1, g, g' and any elements of the coset can be chosen as its representative.

7.2.2 Unique representation of a group in terms of coset representatives

Given this setup, we are now ready to give a unique way of representing group elements once we fix a tower of subgroups for the group and a coset representatives at each level.

Consider an element $g \in G^{(i-1)}$ for some $i \geq 1$ and $G^{(i+1)}$ be the subgroup in the next level in the tower of subgroups. Since the cosets of $G^{(i+1)}$ partitions $G^{(i)}$ it must be that g must lie in exactly one coset.

Observation 7.3. *Consider the groups $G^{(i)}$ and $G^{(i-1)}$ for some $i \geq 1$. Let $g \in G^{(i-1)}$ lie in the coset whose representative is u_1. Then there exists a unique $h_i \in G^{(i)}$ such that*

$$g = h_i u_1$$

[3]One way to show this is as follows : let $g, g' \in G^{(1)}u_1$ where u_1 is a coset representative. Hence $g = h_1u_1$, $g' = h_2u_2$ for $h_1, h_2 \in G^{(1)}$. For any $w \in G^{(1)}g$, $w = hg$ for some $h \in G^{(1)}$. Using the fact that $g = h_1u_1$, we get $w = hh_1u_1$. Hence $w \in G^{(1)}u_1$. Hence $G^{(1)}g \subseteq G^{(1)}u_1$. A similar argument shows that $G^{(1)}u_1 \subseteq G^{(1)}g$. This also shows that $G^{(1)}u_1 = G^{(1)}g'$. Hence the cosets formed by g and g' are the same as the original coset.

Proof. By definition, $g \in G^{(i)}u_1 \Rightarrow \exists h_i \in G^{(i)}$ such that $g = h_i.u_1$. Such a h_i is unique since if there is another $h'_i \in G^{(i)}$ such that $h_iu_1 = h'_iu_1$, then $h_i = g.u_1^{-1} = h'_i$. □

This gives us a way to uniquely represent the elements of G.

Theorem 7.4. *Suppose we fix the tower of subgroups of G denoted $G = G^{(0)} \geq G^{(1)} \geq, \ldots, G^{(k)} = \{id\}$ as well as the coset representatives for $G^{(i+1)} : G^{(i)}$ for all $0 \leq i \leq k-1$ in the tower of subgroups. Then, each element of the group G can be represented as a unique product of coset representatives.*

Proof. Let $g \in G = G^{(0)}$. Let ℓ_1 be the number of cosets in $G^{(1)} : G^{(0)}$. Let $u_1, u_2, \ldots, u_{\ell_1}$ be the coset representatives. By the above observation 7.3 (setting $i = 1$), we know that there exists a unique $h_1 \in G^{(1)}$ such that $g = h_1u_1$. Now consider the cosets $G^{(2)} : G^{(1)}$. Since they partition $G^{(1)}$, we ask, in which coset does h_1 belong to ? By the application of the same claim to h_1 and the pair of groups (G_1, G_2) we get a unique h_2 such that $h_1 = h_2u_2$, where u_2 is the coset in which h_1 belongs. Hence g can be expressed as $h_2u_2u_1$.

Continuing in this way, as we go down the tower of subgroups while fixing the coset representations, in each stage we get a unique h_i and u_i whose product gives h_{i-1} and we end up with a unique representation for the element g. Note that this representation is unique since at each stage the h_i were found in a unique way, and the coset representatives are fixed. □

Note that this also gives us a way to solve $\#\mathcal{GA}$. If we can count the number of coset representatives at each level (which is ℓ_i by our notation), then $\prod_i \ell_i = |G|$.

7.2.3 Finding a generating set for $Aut(X)$

To find a generating set for $Aut(X)$ it suffices to find a tower of sub-groups of $Aut(X)$ and the coset representatives at each level.

Applying Theorem 7.4 for $Aut(X)$, we can represent each element of $Aut(X)$ uniquely as a product (which in our case is composition operation) of a sequence of coset representatives once we define a suitable tower of subgroups for $Aut(X)$ and compute the coset representatives efficiently. Hence it suffices to output these representatives to obtain a generating set.

We define the tower of subgroups in such a way that computing the coset representatives becomes efficient (using $\mathcal{GI}$). Denote $G^{(0)}$ as $Aut(X)$. The subgroups are defined as, for $0 \leq i \leq n-1$,

$$G^{(i+1)} := \{g \in G^{(i)} \mid i^g = i\}$$

That is, $G^{(i)}$ is the sub-group of automorphisms which maps all the nodes of the graph in the range $\{1, \ldots, i\}$ to themselves.

We can check that $G^{(i)}$ is indeed a group since the composition of two permutations which maps all the nodes of the graph in the range $\{1, \ldots, i\}$ to themselves also does the same. Moreover, the identity permutation is the identity for this sub-group too, and the inverse permutation is defined in the standard way and satisfy the property of inverse element of a group. Hence, this is indeed a subgroup. By the definition of $G^{(i)}$ it can be verified that the subgroups defined indeed forms a tower of subgroups of $Aut(X)$.

Now the task is reduced to finding the coset representatives of $G^{(i+1)}$ in $G^{(i)}$.

Claim 7.5. *Let the number of cosets generated by $G^{(i+1)}$ in $G^{(i)}$ be ℓ_i. Then*

$$\ell_i \leq n - i$$

Proof. Consider $g \in G^{(i)}$. Note that g maps the element $i+1$ to an element in the range $\{i+1, \ldots, n\}$. This is because the other elements are already fixed. Let $k = (i+1)^g$ be the element to which $i+1$ is mapped and r be the element such that $r^g = i+1$. Then consider the permutation g' which retains other mappings from g but changes the above two mappings to $(i+1)^{g'} = i+1$ and $r^{g'} = k$. Note that g' is also an automorphism since r being mapped to $i+1$ implies that if r is mapped to the image of i the automorphism would still be preserved (adjacency and non-adjacency is preserved). Also note that g' maps $\{1, \ldots, i+1\}$ and hence is in G^{i+1}. This means that whatever is the number of automorphisms in G^i cannot exceed the number of automorphisms in G^{i+1} multiplied by $n-i$ since the $i+1$ can potentially map to only $n-i$ elements. □

This claims shows that at each level the number of representatives that we need to output is at most $n-i$ and hence the size of generating set collected over all levels is at most $\sum_{i=1}^{n}(n-i) = O(n^2)$. The algorithm for finding the coset representatives is the following. We explain how to get the representatives for a level i.

Look for automorphisms in $G^{(i)}$ which map all of $\{1, \ldots, i\}$ to themselves and map $i+1$ to each element in the set $\{i+1, \ldots, n\}$, one by one and ask the question: is there an automorphism which preserves these mappings. This gives us the cosets since the only freedom in the coset representatives is in where $(i+1)$ maps to.

We answer this using $\mathsf{Color} - \mathsf{GI}$ problem. Since we already have a solution to $\mathcal{GI}$, as demonstrated in the previous lectures, we can use this to also solve $\mathsf{Color} - \mathsf{GI}$. Now, this can done by using $\mathsf{Color} - \mathsf{GI}$ in the following way.

1. Make two copies of the input graph X and call it X_1 and X_2.
2. Colour the vertex $v_k \in \{v_1, \ldots, v_i\}$ by with the color k for both the graphs X_1 and X_2.
3. Colour the vertex v_{i+1} in X_1 and X_2 with the same colour $i+1$ and use a new colour $i+2$ and colour the rest of the vertices in both the graphs with the colour $i+2$.

In this way, we can cast the problem as a $\mathsf{Color} - \mathsf{GI}$ and solve it using $\mathcal{GI}$. Each time we get an answer as yes, we then use the reduction $\mathsf{Color} - \mathsf{GI} \leq \mathcal{GI}$ to find the actual permutation. This permutation is one of the coset representatives and hence in this manner we get each coset representative. Note that for each search for coset representative we make only polynomially many calls to $\mathcal{GI}$ and the number of such representatives is upper bounded by n^2 and hence our reduction is polynomial time and computes a generating set of $Aut(X)$ in polynomial time, given that we can solve $\mathcal{GI}$ in polynomial time.

Note that this procedure can me modified not only to compute an automorphism, but can also be used to compute $|Aut(X)|$. The reason is that the ℓ_i which corresponds to number of coset representatives in $G^{(i+1)} : G^{(i)}$ can be exactly computed in our setting. Our final task is to obtain a permutation in $G^{(i)}$ that sends $i+1$ to $\{i+1, \ldots, n\}$. The set of permutations which satisfy this is nothing but the orbit of $i+1$ in $G^{(i)}$. Hence by Orbit stabilizer theorem, $|G^{(i)}| = \left|(i+1)^{G^{(i)}}\right| \ldots |G^{(i+1)}|$. This gives that $\left|(i+1)^{G^{(i)}}\right| = \frac{|G^{(i)}|}{|G^{(i+1)}|} = \ell_i$. This tells that to obtain ℓ_i it suffices to estimate the size of the orbit of $i+1$.

The generating set that we obtained by fixing a tower of subgroups and coset representatives have many nice properties. We will see more about them in subsequent lectures.

Definition 7.6 (Strong Generating Set)**.** *A generating set for $Aut(X)$ obtained in the manner above making use of coset representatives and tower of sub-groups is known as a strong generating set.*

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Sahil Sharma (*TA:* K Dinesh)
Date: August, 14 2015
Status: δ

Lecture 8

Some Group-theoretic problems in Permutation Groups

8.1 Recap

In the previous lecture we showed that $\mathcal{GA} \leq \mathcal{GI}$. We also defined certain generic group theoretic concepts like *tower of sub-groups*, *coset representatives* and the notion of *strong generating sets*. Also note that the reduction $\#\mathcal{GA} \leq \mathcal{GI}$ can be done using the set of reductions $\#\mathcal{GA} \leq \mathcal{GA} \leq \mathcal{GI}$ where the first reduction follows from the fact that the generating set of $Aut(X)$ was in fact a strong generating set and hence each element of G could be obtained uniquely by composing elements of the generating set. This would imply that the size of the automorphism group is $\prod_i \ell_i$ where ℓ_i is the number of cosets in the i^{th} level of the tower of subgroups of G.

Our aim is to cast the graph theoretic problems in group theoretic terms and solve the problem using the machinery of group theory. Towards this aim, we abstract the various problems and questions which were encountered while doing the previous reductions and give a set of four related group-theoretic problems.

8.2 Some Computational Questions in Group theory

Following are two natural question to ask.

Problem 8.1 (**Order Computation**). *Given a group G via its generating set, can we compute the order of the group, that is, the number of elements in the group.*

Problem 8.2 (**Membership Testing**). *Given an element $g \in G$ and an $H \leq G$ expressed via a generating set S (i,e. $\langle S \rangle = H$), test if $g \in H$ or not.*

Note that problem 2 can be solved using problem 1. This is because we could just add g to the generating set S of H and use problem 1 to obtain the sizes of the groups generated by S and $S \cup \{g\}$. If the sizes are unequal we conclude that g was not in H. This is because if g were in H, then the generating set S could have generated g as well and adding it to S could not have resulted in any new elements.

Here is a recursive strategy to solve Problem 1 (Order computation). The orbit stabilizer lemma that for any $\alpha \in G$,

$$|\alpha^G| \cdot |G_\alpha| = |G|$$

. Suppose we can estimate $|\alpha^G|$ which is the size of the orbit, then to compute $|G|$, we are left with the smaller recursive sub problem of estimating the size of $|G_\alpha|$.

To implement this strategy, we need to solve the two sub problems. (1) finding the size of the orbit and (2) finding the generating set of G_α. Moreover, in last lecture, we also wanted to count the number of permutations in $G^{(i)}$ which first i elements identically and maps the $i+1^{th}$ element to any of the $n-i$ elements. We saw that this is actually a question of obtaining the size of orbit. This motivates the following question.

Problem 8.3 (**Orbit Computation**). *Given a group $G \leq S_n$ (via its generating set S), compute the orbits of the action of G on $[n]$.*

Diversion: The following is a slight diversion to address a question raised in class. So far we were interested in groups which are subgroups of S_n. But how do we answer the same questions for general abstract groups? We show that an abstract group can nevertheless be viewed as a subgroup of a permutation group.

Claim 8.4. *A group G acts on itself.*

Proof. Consider an element $g \in G$. Consider any arbitrary ordering of the elements in G. Then the multiplication of G by g, $G.g$ sends the elements of G to a permutation of themselves[4]. Hence, with each element $g \in G$, we can associate a permutation of the elements of G in the following way : if $G = \{g_1, g_2, \ldots, g_k\}$ then for a $g_i \in G$, corresponding $\sigma_i \in S_k$ is defined as the one which satisfy $(g_i g_1, g_i g_2, \ldots, g_i g_n) = (g_{\sigma_i(1)}, g_{\sigma_i(2)}, \ldots, g_{\sigma_i(k)})$

Collect all the permutations associated with the group elements and call it H. Note that the resulting set of permutations is forms a group since multiplication in original group translates to composition in the new group[5] and the identity element in the original group is associated with the identity permutation and so on.

Hence if the order of the group is k, then the resulting group of permutations is a sub-group of S_k. This defines the notion of a group acting on itself. □

Note 8.5. *Hence from now on, we will assume that G acts on an arbitrary set Ω which we can think $[n]$.*

End of diversion.

8.3 Set Stabilizer Problem

We know that $Aut(X)$ acts on $[n]$. This can also be visualized as $Aut(X)$ acting on the set of potential edges in the graph X. What does it do to the actual edges in this action? Indeed, the

[4]The reason is that the resulting operation acts bijectively on the elements. That is if $g_1 \neq g_2$ then $gg_1 \neq gg_2$. Also for any $h = g_i x$, there is a unique solution $x = hg_i^{-1}$ in G

[5] The statement amounts to showing that for any $g_1, g_2 \in G$ with $\sigma_1, \sigma_2 \in H$ as the corresponding elements defined by the map, $g_1 g_2 \in G \iff \sigma_1 \circ \sigma_2 \in H$. This is because,

$$\begin{aligned} g_1 g_2 (g_1, g_2, \ldots, g_k) &= (g_1 g_{\sigma_2(1)}, g_1 g_{\sigma_2(2)}, \ldots, g_1 g_{\sigma_2(k)}) \\ &= (g_{\sigma_1(\sigma_2(1))}, g_{\sigma_1(\sigma_2(2))}, \ldots, g_{\sigma_1(\sigma_2(k))}) \end{aligned}$$

automorphisms map edges to edges and non-edges to non-edges. Hence, any edge of the graph gets mapped to another edge. To be more precise, we can also think of S_n as acting on the set of all potential edges given by $K = \{\{i, j\} | i, j \in [n] \text{ and } i \leq j\}$. In other words, the action of $Aut(X)$ on the set $E(X)$ does not changes $E(X)$ or it *fixes* $E(X)$. This leads us to the fourth problem called the *Set Stabilizer problem.*

Definition 8.6 (Set Stabilizer of Σ)**.** *Given a G and a subset $\Sigma \subseteq \Omega$,*

$$\mathsf{SetStab}(\Sigma) = \{g \in G \mid \Sigma^g = \Sigma\}$$

where $\Sigma^g = \{\alpha^g \mid \alpha \in \Sigma\}$

Given a group $G \leq S_n$ which acts on a set Ω, we define the Set Stabilizer of $\Sigma \subseteq \Omega$ as the set of all permutations in G which map elements of Σ to elements of Σ and all non elements of Σ to non-elements of Σ. Note that $\mathsf{SetStab}(\Sigma)$ is a sub-group of G. In the case of automorphism groups, the mapping is $G = S_n$, $\Omega = K$, $\Sigma = E(X)$ and $\mathsf{SetStab}(\Sigma)$=$Aut(X)$.

Having set up all the notation, let us state the computational question we seek to answer.

Problem 8.7 (**Set Stabilizer Problem**)**.** *Given a group G via its generating set S, a set Ω on which it acts and a subset $\Sigma \subseteq \Omega$, output a generating set for the group* $\mathsf{SetStab}(\Sigma)$*.*

By our previous discussion on $Aut(X)$ fixing the edges of X, we can conclude that $Aut(X)$ is the stabilizer of $E(X)$. Hence the problem $\mathcal{GA}$ reduces to the problem $\mathsf{SetStab}$.

8.4 Orbit Computation

For the sake of completeness, let us restate the question.
Orbit Computation : Given a group $G \leq S_n$ (via its generating set S), compute the orbits of the action of G on $[n]$.

We can see that the orbits of the action of G on $[n]$ partition the set into different subsets. Hence, we would want to compute the partitions. We cast this problem as a graph theoretic problem.

Let $\Omega = [n]$. We construct a directed graph X with $V(X) = \Omega$ and $E(X) = \{(\alpha, \beta) \mid \exists g \in G, \beta = \alpha^g\} \subseteq V(X) \times V(X)$. We can have the edge $(\alpha, \beta) \in E(X)$ labelled by $g \in G$ if $\alpha^g = \beta$. To check if two elements α, β lie in the same partition, it suffices to check if there is a directed path from α to β (or from β to α). Hence the connected components of the graph corresponds to the orbits. In the next lecture, we will give another algorithm for solving this problem.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Aditi Raghunathan (*TA:* Jayalal Sarma)
Date: August, 17 2015
Status: δ

Lecture 9

Set Stabilizers and Point-wise Stabilisers

9.1 Recap and Exercise

In the previous class, we were looking at ways to solve $\mathcal{AUT}(\mathcal{X})$. We looked at 4 different problems in this context. Consider *Problem 3* of finding the orbit. We are interested in computing the size of the orbit of $\alpha \in G$. Since some of the students expressed difficulty in coming up with an algorithm, we decide to state the algorithm and leave the correctness proofs as an exercise.

9.1.1 Orbit Computation

The following is an algorithm to calculate the orbit of α

Algorithm 1 Algorithm for Orbit Computation

```
1: procedure ORBIT COMPUTATION( Input : Generating set S )
2:     Δ = {α}
3:     repeat
4:         Δ^1 = Δ
5:         for g ∈ S and δ ∈ Δ do
6:             Δ = Δ^1 ∪ {δ^g}
7:     until Δ^1 ≠ Δ
```

The algorithm was developed as step by step in the lecture which also developed its correctness. However, the formal proofs are left as an exercise. To be precise, it is left as an exercise to the reader to do the following : (1) Prove that Algorithm 1 eventually terminates. (2) Prove that Algorithm 1 terminates with $\Delta = \text{Orbit}(\alpha)$ (3) Calculate the running time of Algorithm 1.

The problem can be viewed in terms of the graph. Consider the following definition of a directed graph X. $V(X)$, the vertex set of the graph G is equal to the set Ω. $E(X) = \{(\alpha, \beta) \mid \exists g \in S \text{ such that } \alpha^g = \beta\}$

The key observation is that, if there is a path in X from α to γ then γ is in the orbit of α. Finding the orbit of α is equivalent to computing the transitive closure of the graph X. It is left as an exercise to the reader to complete the details of the above algorithm including rigorous proof of correctness.

9.2 Set Stabilisers Problem

We recall the problem that we defined in the last lecture. Given group $G \leq S_n$ and a set $\Sigma \subseteq \Omega$, where G acts on Ω and $|\Omega| = n$, we are interested in computing the generating set of the following group :

$$\mathcal{SETSTAB}(\Sigma) = \{g \in G \mid \Sigma^g = \Sigma\} \text{ where} \tag{9.4}$$

$$\Sigma^g = \{\alpha^g \mid \alpha \in \Sigma\} \tag{9.5}$$

As we saw in the last lecture, solving the set stabilizer problem gives an algorithm for obtaining $\mathcal{AUT}(X)$.

We now define a variant of the problem called the *Point Stabilizer Problem*

Point-wise Stabilizer Problem

$$\mathcal{POINTSTAB}(\Sigma) = \{g \in G \mid \forall \alpha \in \Sigma, \alpha^g = \alpha\} \tag{9.6}$$

$\mathcal{POINTSTAB}$ is *easier* to solve than $\mathcal{SETSTAB}$ The Point Stabilizer Problem is to obtain a generating set for the group $\mathcal{POINTSTAB}(\Sigma)$. It is easy to check that $\mathcal{POINTSTAB}(\Sigma)$ is indeed a group.

In the reduction from $\mathcal{GA}$ to $\mathcal{GI}$, we defined a tower of sub-groups where

$$G^{(i)} = \{g \in G \mid \forall 1 \leq j \leq i j^g = j\} \tag{9.7}$$

Observe that each tower here is a point-wise stabilizer of $\{1, 2, \cdots i\}$ For each $G^{(i)}$, we have $\Omega = \{1, 2, \cdots n\}$ and $\Sigma = \{1, 2, \cdots i\}$

Here is a general strategy for solving point-wise stabilizer problem. Without loss of generality, as above, assume that Σ is the first i elements of Ω. The problem to solve is precisely the following, given the generators S of a group G, find the generators of $G^{(i)}$. A natural attack on the problem is step wise, given the generating set of G, find that of $G^{(1)}$, and then using that to find the generating set of $G^{(2)}$ and so on. In i levels of this recursion, we will be done.

The key problem that we identified for solving the order computation as well as point-stabilizer problem is the following. Given the generating set of $G^{(i-1)}$, find that of $G^{(i)}$. This is exactly what we will do in the next section.

9.2.1 Schreier's Lemma

Schrier gave a clever way of completing our task. If in addition to the generating set of G^{i-1}, we are also given the coset representatives of G^i as a subgroup in $G^{(i-1)}$.

In the following, we will call G to be the bigger group ($G^{(i-1)}$) and H to be the subgroup $G^{(i-1)}$. Let R be set of right coset representatives of the subgroup H in G. The following lemma gives a direct way of writing the generating set for H.

Lemma 9.1 (**Schreier's Lemma**). *Let S be the generating set of G and R be the set of coset representatives of H. $S' = \{r_1 g r_2^{-1} \mid r_1, r_2 \in R, g \in S\}$ $S' \cap H$ forms a generating set for H.*

Before we prove this, notice that R contains the identity of the groups. Hence the Set $S \subseteq S'$. By taking $S' \cap H$, we are extracting the elements in S' which are also in H. In order to do this, we do need a membership testing method for H. In our context, H is $G^{(i)}$ and has an easy membership test given an element $g \in G^{(i-1)}$.

Proof. Define,

$$RS = \{rs \mid r \in R, s \in S\}$$

Since each element in RS can be written as $(rsr_1^{-1})r_1$, we immediately get that,

$$RS \subseteq S'R \tag{9.8}$$

Let $\langle S' \rangle$ denotes the group generated by S'. By definition, $S'R \subseteq \langle S' \rangle R$. From 9.8, we have $RS \subseteq \langle S' \rangle R$ Therefore, $RSS \subseteq \langle S' \rangle RS$, and hence $RSS \subseteq \langle S' \rangle RS$. Repeating this process, we have $\forall t \geq 0, RS^t \subseteq \langle S' \rangle R$

Since R contains the identity, and S generates G, $G \subseteq \bigcup_{i=1}^{k} RS^t$ for some fixed finite k. In fact, we will see later that $k \leq |G|$, but we do not need this bound here.

We argue that $H \leq \langle S' \rangle$. Since $G \subseteq \langle S' \rangle R$, if $\langle S' \rangle$ does not contain H, $\langle S' \rangle R$ cannot cover G Hence proved by contradiction. Therefore, $S' \cap H$ generates H.

2: JS says: To be completed, why should it exactly generate H?

□

The following is a different view of the above proof. This was not done in the lecture, but we record this for completeness of this notes and for future use. We derive an equivalent statement to the above, by observing that for every r_1 and a, there is a unique r_2 such that $r_1 a r_2^{-1} \in H$. Call this r_2 as $\overline{r_1 a}$. Hence the following is an equivalent restatement.

Lemma 9.2 (**Schreier's Lemma restated**). *Let H be a subgroup of G, S be the generating set of G and R be the set of right coset representatives (containing identity), then,*

$$T = \left\{rs(\overline{rs})^{-1} \mid r \in R, s \in S\right\}$$

is a generating set for H, where $\overline{g}$ denotes the coset representative corresponding to the element g (that is, the unique element in $Hg \cap R$).

Proof. First observe that the elements of T are in H (that is the way we redefined it to begin with). Hence, it cannot generate a larger group. It is enough to show that $T \cup T^{-1}$ generates all the elements of H.

Consider an arbitrary element $h \in H$. Since $H \leq G$, it can be written as the product of elements from $S \cup S^{-1}$. Let $h = s_1 s_2 \dots s_k$. We will incrementally keep modifying this product representations from left to right to obtain a product representation of h using elements of $T \cup T^{-1}$. We call this modification sequence to be $h_1, h_2 \dots h_k$ where each $h_i = h$. In general,

$$h_i = t_1 t_2 t_i r_{i+1} s_{i+1} s_{i+2} \dots s_k$$

where $t_i \in T \cup T^{-1}$ and $r_{i+1} \in R$.

The initial element $h_0 = \prod_{i=1}^{k} s_i$ with $r_1 = 1$, which is exactly the original representation, and the final element $h_k = (\prod_{i=1}^{k} t_i) r_{k+1}$. Since $h_k = h \in H$, it must be that r_{k+1} is the identity. This gives the representation for h in terms of $T \cup T^{-1}$.

Now we complete the proof by defining h_{i+1} from h_i. We just need to define t_{i+1} and r_{i+2}. Take, $t_{i+1} = r_{i+1} s_{i+1} \overline{r_{i+1} s_{i+1}}^{-1}$ and $r_{i+2} = \overline{r_{i+1} s_{i+1}}$. Clearly, the product h_{i+1} has the required form, $t_{i+1} \in T \cup T^{-1}$ and $r_{i+2} \in R$. □

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Aditi Raghunathan (*TA:* Jayalal Sarma)
Date: August, 18 2015
Status: δ

Lecture
10

Using Schreier's Lemma - Reduce Algorithm

Let G be a group generated by a set S. In the first part of this lecture, we went after a question from the previous lecture, about the value of k such that $G \subseteq \cup_{t=0}^{k} S^t$. This is an important question to be answered for the following variant of the membership problem. Given a group $G \leq S_n$ via the generating set S, and a $g \in S_n$, we asked the problem of testing whether $g \in G$. In the previous versions, we expected a yes or no answer to this question. However, it is also useful to ask the computational version of this problem. That is, if $g \in G$, then give a representation of g as the product of elements in S. However, for this, the first question to ask is a bound on the length of any such representation.

10.1 Bounds on length of generating sequence

We start with the following easy bound.

Proposition 10.1. $k \leq |G|$

Proof. We prove the statement by contradiction.

Let the smallest length of the generating sequence of some element $g_1 \in G$ be $|G| + m$, $m > 0$.

$$g_1 = \prod_{i=1}^{|G|+m} a_i \tag{10.9}$$

Consider partial products of the sequence above :

$$S_l = \prod_{i=1}^{l} a_i \tag{10.10}$$

There are $|G| + m$ partial products. There are only G possible values for S_i By Pigeon Hole Principle, we have the following :

$$\exists l_1, l_2, \quad l_1 < l_2 \quad S_{l_1} = S_{l_2} \tag{10.11}$$

This means, we can remove the elements of the generating sequence between l_1 and l_2, which will generate the same element g_1 (From Equation 10.11). Hence we get a shorter generating sequence for g_1 which contradicts the original assumption. Thus, $k \leq |G|$ □

The above bound is too weak for our purpose because, in our context, $|G|$ could be very large. Can we improve this bound in general? Unfortunately we end up proving that the bound is tight.

Proposition 10.2. *There is an $n \in N$, group $G \leq S_n$ and a generating set S of G, and a $g \in G$ such that the shortest product representation of sequence is of length $|G|$.*

Proof. Let $p_1, p_2, \cdots p_m$ be the first m distinct prime numbers. Choose $n = p_1 p_2 \cdots p_m$. We will define a $G \leq S_n$. The G that we define will be a cyclic group generated by the following element a.

$$a = (1, 2, \cdots p_1)(p_1 + 1, p_1 + 2 \cdots p_2) \cdots (\cdots) \tag{10.12}$$

Let M be the order of the element a. That is, M is the smallest M such that $a^M = id$. First observation is that, $M = \prod_{i=1}^{m} p_i$, which is the LCM of the length of the above cycles. It is easy to see that the element a^{M-1} cannot have a shorter representation than as the product of a, $M - 1$ times. Notice that $M \geq 2^m$.

From the prime number theorem, $m \geq \Omega(\frac{p_m}{\log p_m})$. Also notice that $n = \sum_{i=1}^{m} p_i \leq 1+2+3 \cdots + p_m \leq p_m^2$. Thus, $p_m \geq \sqrt{n}$.

Hence, $m \geq \Omega(\frac{\sqrt{n}}{\log \sqrt{n}})$. Thus, $M \geq 2^{\frac{c\sqrt{n}}{\log n}} - 1$. ∎

That is negative news. However, the above example does not rule out the existence of a polynomial length expression or computation for the element. For example, a^{M-1} can be expressed in terms of a through repeated squaring.

But then, how do we formulate our membership problem? This is where the strong generating strikes again. If we are allowed to change the generating set S to and S' (precomputation before seeing the g) we can always compute a generating set such that we can output a short representation.

10.2 Need of a Reduction Step

We now return to the algorithm to compute the generators of a sub group. Using Schreier's lemma, we concluded that it's sufficient to compute the set $S' \cap H$.

Before getting into the algorithm for the same, let us first look at the size of the set that is generated.

By the definition of the set S', we can only say

$$|S'| \leq |S||R|^2 \tag{10.13}$$

If we have to implement this recursively, we see that each level we are incurring a $|R|^2$ term. Hence

$$|S'| \leq |S| l_1^2 \cdot l_2^2 \cdots \tag{10.14}$$

We need to carefully control the size of the generating set S' at each level.

10.3 REDUCE algorithm

How do we reduce the generating set to control its size? Consider $\pi, \psi \in S'$ such that $1^\pi = 1^\psi = k$. Do we need to keep both of them? Yes, indeed, it may be that $2^\pi = k'$ and $2^\psi = k''$. We cannot afford to throw any of them away.

However, here is an observation. Consider replacing *psi* with $\pi^{-1}\psi$.

We can observe the following :

1. We can still obtain all the elements ψ can be obtained $\pi(\pi^{-1}\psi)$. Observe that we also do not add any new elements in the process as well. Hence by this change, the subgroup that is generated remains the same.

2. Doing the above process of replacement for every pair of elements that map 1 to the same element, we end up with elements that all map 1 to different elements (except those that fix 1).

3. $\pi^{-1}\psi$ fixes 1. Hence all the newly added elements fix 1.

4. We can repeat the same procedure for $2, 3, \ldots n$.

The idea seems neat. At the end of the above procedure, for every pair (i, j), we have exactly one π such that $i^\pi = j$. This immediately gives an upper bound of $O(n^2)$ in the size of the sets.

However, there is a catch. When we repeat the same procedure for $2, 3, \ldots n$, what is the guarantee that the property for 1 will not be violated?

The ideas is that once we fix the elements which are mapping 1 to k where $k \neq 1$, all the extra elements which are mapping now 1 to 1 are pushed to the stage 2. In stage 2, the operations of the form $\pi^{-1}\psi$ are done on elements which are stabilizing 1. Hence they will continue to stabilize 1. In the end, we may get several trivial elements, which we can simply discard.

To clarify this, we write this entire algorithm as a procedure.

Algorithm 2 REDUCE Algorithm. Input : Generating set S, Output: Reduced Generating Set

```
1: B_0 = B
2: A[ ][ ], an empty n × n array
3: for i = 0 to n − 1 do
4:     for all ψ ∈ B_i do
5:         j = i^ψ
6:         if A[i][j] is empty then
7:             if j = i then
8:                 B_{i+1} = B_{i+1} ∪ {ψ}
9:             else
10:                A[i][j] = ψ
11:        else
12:            π = A[i][j]
13:            B_{i+1} = B_{i+1} ∪ {ψ^{-1}π}
14: discard all trivial elements from ∪B_i
15: return ∪B_i
```

The correctness proof for the algorithm is left as an exercise.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Samir Otiv (*TA:* K Dinesh)
Date: August 19, 2015
Status: γ

Lecture 11

Pointwise Stabilizer, Membership Testing and Group Intersection

11.1 Recap

We first attacked the Membership Problem. We saw that this reduces to the problem of Order Computation. (See section 8.2)

To solve this problem of Order Computation, we use the Orbit-Stabilizer Lemma. This involved finding out the size orbit of an element α (which we reduced to Graph Reachability), and then recursively finding the size of G_α which is the stabilizer group of α. Product of these two gives the order of the group.

Note that in the above approach, G is specified via a generating set. Hence before we recurse, we need to obtain a generating set for G_α. For this purpose, we used Schreier's lemma to find a generating set. We also observed that size of the generating set can grow very fast. Using the reduction algorithm that Aditi very kindly explained in the previous lecture, we could ensure that the size of the generating set always stayed within $O(n^2)$ (where $n = |\Omega|$).

11.2 Finding Coset Representatives in the tower of subgroups

Now that we have obtained a way of obtaining a generating set for the subgroup, we are left with the problem of finding the orbit size. By Orbit-Stabiliser Lemma (Lemma 5.2), due to one-one correspondence with cosets, we get that to estimate orbit size, it suffices to estimate the number of cosets of the subgroup. In our setting this can be achieved by finding the coset representatives in the tower of subgroups.

Denote R as the set of right coset representatives of $G^{(i+1)}$ in G^i for $i \geq 0$. Note that there are at most $n - i$ cosets, since $i + 1^{th}$ location can be taken to at most $n - i$ different places. Recall that the set of locations where $i + 1$ is taken to by $G^{(i)}$ is precisely $i + 1$.

$$(i+1)^{G^{(i)}} = \{k \in \{i+1, \ldots, n\} | \exists g \in G^{(i)}, (i+1)^g = k\}$$

Let $X_i \subseteq \{i + 1, \ldots, n\}$ be the orbit of $i + 1$ in $G^{(i)}$ (We have seen an algorithm to do it in Section 9.1.1). For each $k \in X_i$, let $g_{i+1,k}$ be an element in $G^{(i)}$ that maps 1 to k. Note that such an element must exists in $G^{(i)}$. To find this group element, it suffices to obtain a path from $i + 1$ to k in the graph of generating set (see Section 9.1.1 for definition) and take the product of edge labels on the path. The set R is now the collection of all $g_{i+1,k}$ for each $k \in X_i$.

11.3 Algorithm for Pointwise Stabiliser Problem

We now piece together the above ideas to show that for any $\Sigma \subseteq \Omega$, the Generator set of $\mathsf{PointStab}(\Sigma) \leq S_n$ can be computed in $poly(|\Omega|)$ time. Let $i \geq 1$. Recall that $\mathsf{PointStab}(\{1, 2, \ldots, i\}) = G^{(i)}$.

To compute a generating set for $G^{(i)}$

1. Use the technique specified above to find coset representative for $G^{(1)}$ in G. We consider $G^{(0)} = S_n$ and use the generator set for S_n as $\{(1)(2)\ldots(n), (1\ 2), \ldots, (n-1\ n)\}$ while constructing the graph for generating set.

2. Previous step gives us the set R which consists of the right coset representatives of $G^{(1)}$ in $G^{(0)}$. Now we can apply Schreier's Lemma to obtain the generating set for $G^{(1)}$.

3. Apply the REDUCE algorithm (Algorithm 2) to obtain a smaller generator set.

4. Recurse by following the same procedure, and finding the generating set of $G^{(2)}$ given $G^{(1)}$, until the desired $G^{(i)}$ is found.

It can be seen the each step terminates in $\mathsf{poly}(|\Omega|)$ time and the number of recursive calls is at most $i \leq |\Omega|$. Hence overall run time is polynomial.

11.4 Algorithm for Membership Testing

Note that we already saw how to check membership of an element in a group using an algorithm to compute order of the group. But this does not tell anything about how the element is expressed in terms of its generators. Hence we consider the following stronger version of Membership problem.

Problem 11.1 (Stronger Membership Test)**.** *Given a generating set A of G with, $G \leq S_n$ and a $g \in S_n$, give a representation in terms of any desired generating set, and the generating set itself.*

Algorithm 3 MEMBERTEST : Algorithm for Membership Testing

```
1:  procedure MemberTest(Input : Element g, Generating set A for G^(i), Index i)
2:      if g = id then
3:          return true
4:      X_i = (i+1)^{G^(i)} (Use Orbit Computation Algorithm 1)
5:      Compute R, the set of coset representatives of G^(i+1) in G^(i) (see Section 11.2)
6:      k = (i+1)^g (Image of i+1 on action of g)
7:      if k ∉ X_i then
8:          return false
9:      else
10:         B = Generating set of G^(i+1) (Use Schreier's Lemma)
11:         Apply Algorithm Reduce to B and denote A' for the reduced set obtained
12:         Pick g_ik from R that maps i to k
13:         g' = g ∘ g_ik^{-1}
14:         return MemberTest(g', A', i+1)
```

Note that if insisting on giving the representation of g in terms of A, then the problem may not be solvable in polynomial time since there are groups which require large size to represent in terms of its generators (Proposition 10.2). Now we call MemberTest on the input $(g, A, 0)$ for checking if $g \in \langle A \rangle$ or not. It is left as an exercise to show that the algorithm terminates in $poly(|\Omega|)$ time.

11.5 Group Intersection Problem

We consider the following two problem :

Problem 11.2 (Subgroup Problem)**.** *Given groups G and H via their generating sets A and B respectively, test if $H \leq G$*

Problem 11.3 (Group Intersection Problem)**.** *Given groups G and H via their generating sets A and B respectively, find the generating set of $G \cap H$*[6]*.*

Since we have already given an algorithm for membership checking, Subgroup problem has the following algorithm : for each $b \in B$ check if $b \in G$ using MemberTest algorithm.

So how about Group Intersection problem ? We show that this problem is as hard as the Set Stabilizer Problem. We show this by first reducing SetStab to GroupInter.

Claim 11.4. SetStab $\leq$ GroupInter

Proof. Given $\Sigma \subseteq \Omega$, generating set K of G we need to produce groups J, H via generators A, B such that $J \cap H =$ SetStab(Σ). Consider the group, $H = Sym(\Sigma) \times Sym(\Omega \setminus \Sigma)$ where $Sym(T)$ consists of all permutations defined over T of length $|T|$. Hence $Sym(T)$ fixes T. Note that $H \leq S_n$ and consists of all permutations that have Σ mapped to Σ. Since SetStab(Σ) talks about permutations in G that fix Σ, $G \cap H$ is the SetStab(Σ). Now it remains to give a generator for $Sym(\Sigma)$. It can be shown that if $\Sigma = \{1, 2, \ldots, n\}$, then the generator set for $Sym(\Sigma)$ is $(1, \ldots, n), (1\ 2), (2\ 3), \ldots, (n-1\ n)$. □

In the next lecture, we will show GroupInter reduces to SetStab thereby showing that they are equally hard.

[6]Verify that if G and H are groups then $G \cap H$ is also a group

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Samir Otiv (*TA:* K Dinesh)
Date: August 21, 2015
Status: γ

Lecture
12

Group Intersection to Set Stabiliser and Jerrum's Filter

In last lecture, we completed the discussion on algorithms for Pointwise Stabilizer problem and Membership testing. We saw two new problems – Subgroup problem and Group Intersection problem and showed that Subgroup problem can be reduced to Membership testing and hence poly time solvable while Set Stabilizer problem reduces to Group Intersection problem. We continue with our discussion on Group Intersection and Set Stabilizer problem.

12.1 GroupInter $\leq$ SetStab

We show that Group Intersection problem reduces to Set Stabilizer problem. Before we start, let us look at ways of combining groups.

Definition 12.1 (Direct Product)**.** *For groups G, H, define direct product of G and H denoted $G \times H$ as the group*

$$G \times H = \{(g, h) \mid g \in G, h \in H\}$$

with group operation for $(g_1, h_1), (g_2, h_2)$ defined as,

$$(g_1, h_1)(g_2, h_2) = (g_1 g_1, h_1 h_2)$$

If G, H are subgroups of S_n acting on the set Ω, then the group $G \times H$ acts on $\Omega \times \Omega$ with the action defined as $(\alpha, \beta)^{(g,h)} = (\alpha^g, \beta^h)$ where $\alpha, \beta \in \Omega$ and $(g, h) \in G \times H$.

The reduction is as follows : given groups G and H, consider the group $G \times H$ and $\Sigma' = \{(i, i) \mid i \in \Omega\}$. To establish correctness, we show that if we consider the action of $G \times H$ on $\Omega \times \Omega$, then $\mathsf{SetStab}(\Sigma')$ is $G \cap H$. More precisely,

Claim 12.2. *$(G \cap H)_{duplicate} = \mathsf{SetStab}(\Sigma')$ where $(G \cap H)_{duplicate} = \{(g, g) \mid g \in G \cap H\}$ and Set Stabilizer computation is with respect to the action of $G \times H$ on $\Omega \times \Omega$.*

Proof. We show containment in both ways.

$[(G \cap H)_{duplicate} \subseteq \mathsf{SetStab}(\Sigma')]$ Consider $g \in G \cap H$. Clearly $(g, g) \in G \times H$. For $(i, i) \in \Sigma'$, $(i, i)^{(g,g)} = (i^g, i^g) \in \Sigma'$ by definition of Σ'. Hence $(g, g) \in \mathsf{SetStab}(\Sigma')$.

$[SetStab(\Sigma') \subseteq (G \cap H)_{duplicate}]$ Let $(g, h) \in SetStab(\Sigma')$. By definition, $\forall (i, i) \in \Sigma'$ we have $(i, i)^{(g,h)} = (i^g, i^g) = (j, j)$ for some $j \in \Omega$. In other words, $i^g = i^h$ for all i. Hence it must be that g and h are the same giving $g = h \in G \cap H$. Hence $(g, h) \in (G \cap H)_{duplicate}$. ∎

Now that we know that GroupInter and SetStab problems are equivalent to each other and are both seemingly harder than the graph automorphism problem which we want to solve. Our next aim would be to solve special cases of these problems.

Before we do the special cases, we would like to do one interesting optimization for the REDUCE algorithm.

12.2 Jerrum's Filter: Obtaining generating set of size $n-1$

3: JS says: All the arguments are in place after the edit. A little more cleanup is to be done, will do that at the end of the semester.

We know from Schreier's lemma how to obtain a strong generating set for $G \leq S_n$ of size $O(n^2)$. We also know that for any subgroup of S_n, there exists a generating set of size $O(n \log n)$. A natural question is : can we improve this bound further ?

Mark Jerrum showed that one can give a generator of size $n-1$ instead. The process is algorithmic and is called as Jerrum's filter[7].

12.2.1 Main ideas

Given a generating set S of G and G acts on a set Ω, our aim is to obtain another generating set A for G of size $\leq n-1$.

Given a set S of group elements, we define a undirected graph $X_S(V, E)$ with $V = \Omega$ and $E \subseteq V \times V$ is defined as for each $g \in S$, let i_g denote the least index moved by g as i_g i,e. $\forall i \leq i_g$ $i^g = i$. Now we add the edge (i_g, i_g^g) to E. We also label this edge by g.

From the definition of X_S, the following can be observed.

Observation 12.3. *For a $g \in S$ g^{-1} maps i_g^g to i_g. Also both g, g^{-1} has the same least index.*

The main idea is the following : we maintain a set A of generators in such a way that X_A is acyclic. When we get a new $g \in S$, we add g to A and compute $X_{A\cup\{a\}}$. Now we make some operation on this graph $X_{A\cup\{g\}}$ in such a way that the resultant graph has no cycles and the updated set A still generates G. This process is *online*, since we update the graph as and when g arrives and do not need the entire S a priori. Continuing in this manner, we get the X_A to be a forest and hence has $n-1$ edges.

Before explaining the details, we need the following measure.

Definition 12.4 (Weight of T)**.** *Let $T \subseteq S_n$. Then*

$$wt(X_T) = \sum_{g \in T} i_g$$

Observe that for any T, $wt(T) \leq |T|n$.

[7] It turns out that this can be improved to $\lfloor n/2 \rfloor$ (Neumann) and is tight. We will not be discussing it in this lecture.

12.2.2 The Algorithm

The main idea of the algorithm is to maintain a set A such that X_A is acyclic. In the construction, we ensure that we add edges corresponding to element only in A (as explained while we defined X_A) and remove those edges not in A.

At the end of the process, we get an acyclic graph which is a forest with number of edges at most $n-1$. Hence $|A|$ is upper bounded by number of edges in the graph which is at most $n-1$.

Algorithm 4 JerrumFilter : Computing a generating set of size at most $n-1$

1: **procedure** JerrumFilter(Input : Generating set S for G given in online fashion)
2: Let A be generating set maintained. Initially $A = \emptyset$
3: **for** a given $g \in S$ **do**
4: If $g \notin A$, then add g to A and add $e_g = (i_g, i_g^g)$ to the graph X_A
5: **if** a cycle is created in X_A **then**
6: Let C be the unique cycle created of length k
7: Let i_0 be the least index in the cycle C, g_0 be an element corresponding to i_0
8: Let $\{g_t\}_{t\in[k]}$ be the edge labels on the cycle C with g_0 as label of edge incident to i_0
9: Denote $g_0 g_1^{\epsilon_1}, \dots g_k^{\epsilon_k}$ as h where $\epsilon_i \in \{1, -1\}$ defines the direction
10: Remove g_0 and add h to the set A and update X_A to reflect the changes
11: If h is identity, ignore by not adding it to A

The algorithm is self explanatory till step 5. Note that there is exactly one cycle formed by adding of e_g. This is because, so far we have maintained the invariant that X_A is acyclic. Let i be the least index in the cycle C. We claim that one of the edges incident to i_0 labelled by g in C must have $i = i_0^g$ as its least index. If not then the neighbours of i_0 would not be the least moved element by the corresponding group element.

Since i_0 is the least index in the cycle, it is fixed by all of $g_0, g_1, \dots, g_k$ and therefore the least element moved by h is strictly more than i. Therefore, the weight of the graph increases by at least on 1 on performing this operation. Since the weight of S is upper bounded by $\mathsf{poly}(|S|, n)$, the process terminates in in polynomially many steps.

On repeating this with all elements, we have a generating set A with an acyclic graph X_A, and therefore $|A| \leq n-1$.

Exercise 12.5 (Problem Set 0, Question 1)**.** *We stated in class that for any $n > 3$, every subgroup of S_n can be generated by at most $\lfloor \frac{n}{2} \rfloor$ elements. Show that this bound is tight by giving an example of an Ω and a group $G \leq S_{|\Omega|}$ acting on it such that G requires $\frac{|\Omega|}{2}$ elements to generate it. (Hint : Consider $\Omega = \{a_1, a_2, \dots, a_m, b_1, b_2 \dots b_m\}$).*

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Monosij Maitra (*TA:* Ramya C)
Date: August, 24 2015
Status: α

Lecture
13

Group Intersection Problem under Special Cases-I

13.1 Recap

In the previous lecture, we used *Jerrum's filter* to reduce the size of the generating set $\langle S \rangle$ of a subgroup G of S_n to at most $n-1$. Let us revisit the *Group Intersection Problem* :

Problem 13.1 (Group Intersection). *Given two groups G and H via their generating sets $\langle A \rangle$ and $\langle B \rangle$ respectively, with $G \leq S_n$ and $H \leq S_n$, compute the generating set for the group $G \cap H$?*

We also showed that the *Group Intersection Problem* can be reduced to the *Set Stabilizer Problem* and vice-versa. As $GA \leq SETSTAB \equiv$ Group Intersection Problem both are harder than *Graph Automorphism Problem*. In this lecture we will be interested in solving Problem 13.1 in a special setting of *Normal Subgroups*. But before that let's review the basics of *Normal Subgroups*.

13.2 Normal Subgroups

We already know that if H and G are two groups and $H \leq G$, then the right cosets of H in G are given by $Hg = \{hg \mid h \in H \text{ and } g \in G\}$. Similarly the left cosets are given by $gH = \{gh \mid h \in H \text{ and } g \in G\}$. By *Lagrange's Theorem* we also know that any two right cosets of a subgroup H are either *disjoint* or *equal* i.e. for any $g_1, g_2 \in G$ with $g_1 \neq g_2$, either $Hg_1 \cap Hg_2 = \emptyset$ or $Hg_1 = Hg_2$. The same conditions hold true for the left cosets of H too.
For any $H \leq G$ it is not necessary that $Hg = gH$. Normal subgroups achieve the above condition.

Definition 13.2. *Let $H \leq G$ such that $\forall g \in G,\ Hg = gH$. Then H is called a* Normal Subgroup *of G and is denoted by $H \triangleleft G$.*

One special property of such a subgroup H follows directly from the definition: The set of all left cosets and right cosets remain the same. Consider the operation $*$ defined as $Hg_1 * Hg_2 = Hg_1g_2\ \forall g_1, g_2 \in G$.

Observation 13.3. *The set of all cosets form a group with respect to the operation $*$ defined above.*

- **Closure** *Since G has closure property and $Hg_1 * Hg_2 = Hg_1g_2$, hence this itself implies closure of the set of cosets under $*$.*

- **Existence of Identity** *The subgroup H serves as the identity element in the set as $\forall g \in G,\ Hg * Hg^{-1} = Hgg^{-1} = H$.*
- **Existence of Inverse** *Since $\forall g \in G,\ Hg * Hg^{-1} = Hgg^{-1} = H$, therefore inverse exists.*
- **Associativity** *Follows from the definition.*

The set of all cosets of a normal subgroup H in G has a distinct name in the literature of Group theory.

Definition 13.4. *Let $H \triangleleft G$. The set of all cosets $\{Hg \mid g \in G, h \in H\}$ along with the binary operation $*$ is called the* Quotient Group *of G by H (denoted by G/H).*

If $g' \in Hg$, then $Hg' = Hg$ i.e. Hg and Hg' refers to the same coset.

Claim 13.5. *Let $H \triangleleft G$ and Hg_1, Hg_2 be right cosets of H in G. Let $g_1' \in Hg_1,\ g_2' \in Hg_2$. Then*

$$\text{If } Hg_1 = Hg_1' \text{ and } Hg_2 = Hg_2' \text{ then } Hg_1g_2 = Hg_1'g_2'$$

Proof.

$$\begin{aligned} Hg_1g_2 &= Hg_1'g_2 \quad [\text{Since } Hg_1' = Hg_1] \\ &= g_1'Hg_2 \quad [\text{Since } H \triangleleft G] \\ &= g_1'Hg_2' \quad [\text{Since } Hg_2' = Hg_2] \\ &= Hg_1'g_2' \quad [\text{Since } H \triangleleft G] \end{aligned}$$

□

We want to emphasize two things here: firstly, the above claim shows that $*$ is well-defined whenever $H \triangleleft G$. Secondly, note that this claim holds only when $H \triangleleft G$. Otherwise, the operation $*$ on the set of cosets is in general not well-defined.

It is natural to ask if the above claim characterizes normal subgroups. That is is the converse of the above claim true ?

Claim 13.6. *Let $H \leq G$. Let $g_1, g_1', g_2, g_2' \in G$ be such that if $Hg_1 = Hg_1'$ and $Hg_2 = Hg_2'$ implies $Hg_1g_2 = Hg_1'g_2'$. Then $H \triangleleft G$.*

Proof. Let us first argue that $\forall g \in G, gHg^{-1} \subseteq H$. That is $\forall g \in G, \forall h \in H, ghg^{-1} \in H$ For a suitable setting of g_1, g_2, g_1', g_2' we have $ghg^{-1} \in H$. As $gHg^{-1} \subseteq H$ we have $Hg \subseteq gH$ and $gH \subseteq Hg$. Therefore $H \triangleleft G$. □

Having geared up with the basic notions of Normal Subgroups, we can now ask the following computational question.

Problem 13.7 (Group Normality). *Given two groups G and H via their generating sets A and B ($\langle A \rangle = G$ and $\langle B \rangle = h$), check if $H \triangleleft G$.*

13.2.1 Normalizer and Normal Closure of a Group

Suppose for some groups H and G where $H \leq G$, $H \not\trianglelefteq G$. This happens because there is at least one $g \in G$ such that $Hg \neq gH$. Let us collect all such $g \in G$ such that $Hg = gH$. This leads us to the following definition

Definition 13.8. *The set $N_G(H) = \{g \in G \mid Hg = gH\}$ is called the* **Normalizer** *of H in G.*

Observe that for any $H \leq G$, $N_G(H) \neq \emptyset$ as $H \in N_G(H)$. AS every group is a normal subgroup of itself we have $|H| \leq |N_G(H)| \leq |G|$.

Observation 13.9. *$(N_G(H), *)$ forms a group.*

where * is defined in Definition 13.4.

One more computational question that arises in this context.

Problem 13.10 (**Normalizer Computation**)**.** *Given two groups G and H via their generating sets $\langle A \rangle$ and $\langle B \rangle$ respectively, output the generating set of $N_G(H)$.*

Clearly, the normalizer $N_G(H)$ is the largest subgroup of G in which H is a normal subgroup. Then a natural question that arises is the smallest normal subgroup of G that contains H.

Definition 13.11. *The smallest normal subgroup of G that contains H is called the* **Normal Closure** *of H in G.*

Keeping the above definition in our mind another question pops up naturally regarding the computability of normal closure of a group as defined formally below.

Exercise 13.12. *Given G, H via their generating sets and let $H \leq G$, compute the generating set of the normal closure of a subgroup H of G?*

13.3 Group Intersection Problem in the Context of Normalizers

Let $G, H \leq S_n$ and $N_{S_n}(H)$ be the normalizer of H in S_n.

We say G *normalizes* $H$when $\forall g \in G, Hg = gH$. This means $G \leq N_{S_n}(H)$. We solve the Group Intersection problem for this special case.

For general groups G and H, we are not aware of an efficient algorithm for computing the generating set of $G \cap H$. Before getting on to the actual algorithm for computing the generating set of G and H in the special setting let us have some more observations.

Consider the set $GH = \{gh \mid g \in G, h \in H\}$.

Observation 13.13. *Let $G, H \leq S_n$. If G normalizes H, then $GH \leq S_n$.*

- **Closure** *Let $g_1h_1, g_2h_2 \in G$, then*

$$\begin{aligned} g_1h_1 * g_2h_2 &= g_1g_2.h_1'h_2 \quad \textit{[Since } G \textit{ normalizes } H\textit{]} \\ &= gh \in GH. \end{aligned}$$

- **Existence of Identity** *Setting $g = h = e$ it is trivial to observe that $e \in GH$.*

- **Existence of Inverse** $\forall gh \in GH,\ h^{-1}g^{-1} \in GH$. *Thus,*

$$\begin{aligned} gh.(h^{-1}g^{-1}) &= g(hh^{-1})g^{-1} \\ &= g(e)g^{-1} \\ &= gg^{-1} \\ &= e \in GH. \end{aligned}$$

- **Associativity** *Follows naturally from the definition.*

We are now just a few steps away from describing the algorithm for Problem 13.1. Although the algorithm will be given explicitly in the next lecture, the first idea is to construct a tower of subgroups such that we can get the generating set of $G \cap H$ by applying Schreier's lemma. Observe that $G^{(i)}$ is the set of all permutations of S_n which fix the element i. As $G^{(i)}$ normalizes H, by Observation 13.13, we infer that $G^{(i)}H \leq S_n$.

Now we are ready to describe the construction of the *Tower of Subgroups* as follows:

$$G \cap G^{(0)}H \geq G \cap G^{(1)}H \geq G \cap G^{(2)}H \geq \ldots \geq G \cap G^{(n-1)}H$$

Two quick observations are:

Observation 1. On the right end of the above chain $G^{(n-1)}$ is the subgroup which *fixes* all the elements $1, 2, \ldots, n-1$. Hence it gives the identity permutation in S_n exactly. Thus $G \cap G^{(n-1)}H = G \cap H$.

Observation 2. On the left end of the above chain $G \cap G^{(0)}H = G \cap GH$. As $G \subseteq GH$, we have $G \cap G^{(0)}H = G \cap GH = G$.

Thus the above tower of subgroups really look like that as shown below.

$$G \geq G \cap G^{(1)}H \geq G \cap G^{(2)}H \geq \ldots \geq G \cap H$$

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Monosij Maitra (*TA:* Ramya C)
Date: August, 25 2015
Status: α

Lecture
14

Group Intersection Problem under Special Cases-II

14.1 Recap

In the previous lecture, we were devising the tools that we need for solving Problem 13.1 under the special setting of a group G acting as a **normalizer** for group H. We discussed **Normal Subgroups** along with the concepts of **Normalizers** and **Normal Closure**. Then we looked at the Tower of Subgroups that we are going to use to give the algorithm for Problem 13.1. The Tower of Subgroups look like the following:

$$G \geq G \cap G^{(1)}H \geq G \cap G^{(2)}H \geq \ldots \geq G \cap H$$

14.2 Getting to the Algorithm

In the above chain of subgroups we are already given the generating set of G. We can apply **Schreier's Lemma** to this chain of subgroups to get the generating set of $G \cap H$. But before doing that we must ensure of three things:

1. Computation of the coset representatives of the subgroups at each level in the chain.
2. Testing membership in the subgroups.
3. Ensuring that the number of cosets at all level is small.

These three properties must be satisfied in order to get a *strong generating set*.

- We can apply our known algorithm to find the set of coset representatives of the subgroup $G^{(i)}H$ in $G^{(i-1)}H$.
- For the second property, let S_i be the generating set of $G^{(i)}$. Observe that any element of $G^{(i)}H$ can be generated by the set $\langle S_i \cup B \rangle$. Once we get these, we can easily apply our *Membership Testing algorithm* which was done in a previous lecture.
- For an assurance of the third property we need to prove the following lemma.

Claim 14.1. *The index of $G \cap G^{(i)}H$ in $G \cap G^{(i-1)}H$ is at most $(n-i)$.*

The crux of the proof of the above claim lies in the following arguments about how a coset of $G^{(i)}H$ actually looks like which is inside $G^{(i-1)}H$. In other words, $G^{(i)}Hg$ is a coset of $G^{(i)}H$ inside $G^{(i-1)}H$ if and only if $g \in G^{(i-1)}H$. This implies that $g = g'h'$, where $g' \in G^{(i-1)}$ and $h' \in H$. Thus we see that

$$\begin{aligned} G^{(i)}Hg &= G^{(i)}Hg'h' \\ &= G^{(i)}g'Hh' \quad [\text{Since } G^{(i-1)} \text{ normalizes } H], Hg' = g'H \\ &= G^{(i)}g'H \quad [\text{As } h' \in H, Hh' = H] \\ &= (G^{(i)}g')H. \end{aligned}$$

The rest of the proof is left as an exercise. What remains to be argued is that after the intersection with G *number of cosets* still bounded above by $(n-i)$ in the *i-th* level of the tower of subgroups. As an intermediate observation, we claim the following:

Claim 14.2. $H \triangleleft GH$.

Proof. Let $g' \in GH$. Then $g' = gh$ for some $g \in G, h \in H$. For deducing $H \triangleleft GH$, we need to show that $\forall g' \in GH, Hg' = g'H$. So we proceed as follows:

$$\begin{aligned} Hg' &= Hgh \\ &= gHh \quad [\text{Since } G \text{ normalizes } H] \\ &= ghH \quad [\text{Since } Hh = hH] \\ &= g'H. \end{aligned}$$

□

14.2.1 The Algorithm

Now we are in a perfect set up to describe the algorithm for computing the generating set of $G \cap H$. The algorithm just applies **Schreier's Lemma** *recursively* at all the $(n-1)$ levels in the tower of subgroups to compute the generating set of $G \cap G^{(i)}H,\ \forall i = 1, 2, 3, \ldots, (n-1)$.

We know that the number of cosets in the *i-th* level is bounded above by a small quantity $(n-i)$ as Claim 14.1 shows. From the REDUCE algorithm that we saw in a previous lecture we know that the size of the generating set at each of the $(n-1)$ levels of the tower of subgroups computed by **Schreier's Lemma** is $O(n^2)$. Therefore the asymptotic running time of this algorithm is $O(n^3)$.

14.3 From Setwise Stabilizers to Point-wise Stabilizers : Bounded Color Class Graph Isomorphism Problem

We have already mentioned of the fact that $GA \leq SetStab \equiv$ Group Intersection Problem. Graph Isomorphism problem is hard in general. In this context we have seen that $\mathcal{COLOR} - \mathcal{GI} \leq \mathcal{GA}$. When the number of colours, $k = 1$, colored graph isomorphism is the same problem of Graph Isomorphism. Thus instead of getting our attention to the *number of colours*, we focus on the size of each colour class, i.e. $|\psi^{-1}(i)|, i \in \{1, 2, \ldots, k\}$. We assume that $\forall i \in \{1, 2, \ldots, k\}, |\psi^{-1}(i)| \leq b, b \in \mathbb{N}$ i.e. the size of each colour class in bounded by a ***constant*** b. Now the question we ask is the following:

Problem 14.3. *Given two graphs $X_1(V_1, E_2)$ and $X_2(V_2, E_2)$ with sizes of the color classes bounded by a constant, can we check if they are isomorphic?*

We reduce colored graph isomorphism problem to the colored graph automorphism problem.

Since we know that checking Isomorphism between X_1 and X_2 can be reduced to checking Automorphism group of $X = X_1 \cup X_2$. This additional constraint of colour imposes an association between an edge in the graph with another edge whose vertices are of same colour so that it becomes a colour preserving map. In other words,

$$Aut(X) \leq Sym(\Psi_1^{-1}) \times Sym(\Psi_2^{-1}) \times Sym(\Psi_3^{-1}) \times \ldots \times Sym(\Psi_k^{-1})$$

where k is number of colours. We must note that all elements in $Sym(\Psi_1^{-1}) \times Sym(\Psi_2^{-1}) \times Sym(\Psi_3^{-1}) \times \ldots \times Sym(\Psi_k^{-1})$ may not be elements in $Aut(X)$.

Now the idea is to use the known $SetStab$ reduction so that we can compute the neccessary stabilizer set efficiently. From that context we denote $E_i = E \cap \binom{C_i}{2}$ to be the set of all ***intracolour edges*** for each colour i and $E_{ij} = E \cap (C_i \times C_j)$ to be the set of all ***intercolour edges*** between any two *distinct* colour classes i and j. Note that an edge inside E_i is mapped within E_i again and an edge inside E_{ij} is mapped within itself too, thereby obtaining a *colour preserving map*. In other words, any permutation $g \in Sym(C_i)$ is an automorphism if and only if $E_i^g = E_i$ and $E_{ij}^g = E_{ij}$ i.e. we have got the desired $SetStab$ form.

Let the group be $Sym(\Psi_1^{-1}) \times Sym(\Psi_2^{-1}) \times Sym(\Psi_3^{-1}) \times \ldots \times Sym(\Psi_k^{-1})$. All we need to do is to find a subgroup of the above grove that pointwise stabilizes the sets E_i and E_{ij}.

Thus we view S_n as acting on the following Ω given by

$$\Omega = \left(\bigcup_i C_i\right) \cup \left(\bigcup_i 2^{\binom{C_i}{2}}\right) \cup \left(\bigcup_i 2^{C_i \times C_j}\right)$$

where the term $2^{\binom{C_i}{2}}$ gives the potential subsets of all *intracolour edges* while the term $2^{(C_i \times C_j)}$ gives the potential subsets of all *intercolour edges*. Now we bound the size of Ω as follows:

$$|\Omega| \leq n + k.2^{\binom{b}{2}} + \binom{b}{2}.2^{b^2}$$

Since b is a constant, $|\Omega|$ is polynomial in n.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Jayalal Sarma (*TA:* Jayalal Sarma)
Date: Aug 26, 2015
Status: α

Lecture
15

Group Intersection Problem (Contd)

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Mitali Bafna (*TA:* K Dinesh)
Date: Aug 31, 2015
Status: γ

Lecture
16

On Transitive Group Actions and their properties

In the last lecture, we saw algorithms for solving coloured $\mathcal{GI}$ when the colour class are bounded size. In this lecture, we will see some new concepts for computing $Aut(X)$ for graph X which has more a divide and conquer flavour.

16.1 Transitive Group action and Blocks

We start with the following example. Consider a complete binary tree T of depth k. Let us ask : how does $Aut(X)$ act on the leaves of T. If we consider the action of $Aut(X)$ on set the leaves of T, then we can observe that there is an automorphism that maps any leaf to any other leaf. For example, the permutation which takes 1 to 3 and 2 to 4 and fixes the rest takes leaf 1 to leaf 3.

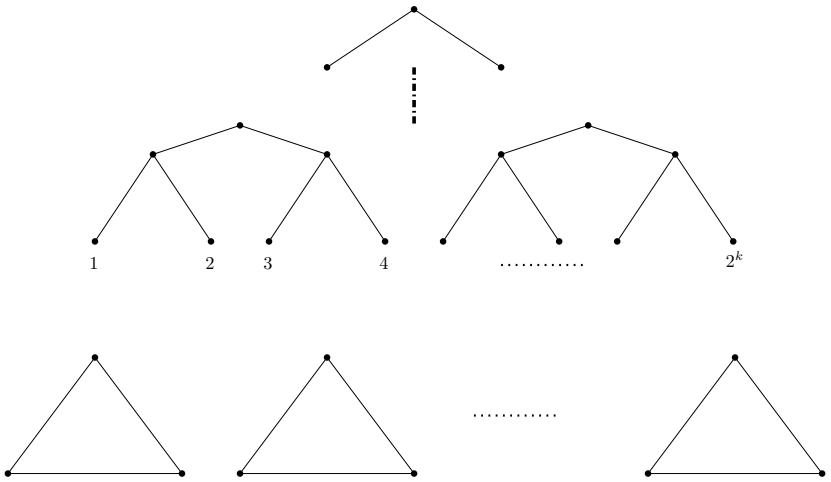

FIGURE 16.2: Transitive group action

This property of the group action is called Transitive.

Definition 16.1 (Transitive Group Action)**.** *For a group $G \leq S_n$ acting on a set Ω, the action is said to be transitive if, for all $\alpha, \beta \in \Omega$, there exists a $g \in G$ such that $\alpha^g = \beta$.*

In the above example, we can see $Aut(X)$ acts transitively on the set of leaves of T.

Observe that any nontrivial automorphism can either map 1 and 2 among each other (i,e. 1 to 2 and 2 to 1) or map both 1 and 2 to say 3 and 4. But it can never map say 1 to 2 and 2 to 3 as it violate adjacency. Hence the leaves with common ancestors always gets moved in a pair.

Consider another graph X which is k vertex disjoint triangles. Consider the action of $Aut(X)$ on $V(X)$. We again observe a similar phenomena as before. There is an automorphism that takes any vertex to any vertex. Also any automorphism maps vertices in a triangle to itself or to a totally different triangle. Hence either a triangle is mapped to itself or is moved whole together as a block.

These two examples motivate the notion of blocks.

Definition 16.2 (Blocks)**.** *For a group G acting on Ω, a set $\Delta \subseteq \Omega$ is a block if $\forall g \in G, \Delta^g = \Delta$ or $\Delta^g \cap \Delta = \phi$.*

We observe that Ω and the singletons sets $\{\alpha\}$ where $\alpha \in \Delta$ form blocks trivially.

Apart from the fact that such block structure naturally arise in the action in many graphs, they also can be potentially used to compute $Aut(X)$. In the case of trees, we can start with trivial blocks whose automorphisms are also trivial, combine the blocks and obtain the automorphism for larger blocks and proceed in a bottom up manner building larger blocks and thereby get the automorphism of the entire tree.

16.2 Primitive actions

We saw that for $\Omega = [n]$ and $G = S_n$, the sets Ω, $\{1\}, \{2\}, \ldots, \{n\}$ does form blocks. Are there any other blocks ? The answer is no since for any $T \subsetneq \Omega$ of size at least 2, there is a permutation in G that fixes one and moves the other. Hence these are the only blocks and we call such blocks trivial blocks and the action as *primitive*.

Definition 16.3 (Primitive action)**.** *A group action of G on Ω is primitive if there are no non-trivial blocks. An action which is not primitive is called imprimitive.*

Let us consider another action where $G \subseteq S_n$ acts on itself via right multiplication (see Lecture 8). That is for $g' \in G$, the action takes g to gg' with $\Omega = G$. This action is clearly transitive as given any $g, h \in G$, $g' = g^{-1}h$ takes g to h. Also, if G has a non-trivial [8] subgroup H then the action is not primitive. This is because, the cosets of H in G form the non-trivial blocks. That is, for any two shift of H say Hg_1, Hg_2, either $Hg_1 = Hg_2$ or $Hg_1 \cap Hg_2 = \emptyset$ (Lemma 4.15) which we observed in the proof of Lagrange's theorem.

This shows that if G has a non-trivial subgroup, then the action is not primitive. Is the converse observation that G has only trivial subgroup imply the action is primitive also true ? We will see in the next lecture that the converse holds too. This gives a very interesting characterization or a "connection" between blocks which are combinatorial in nature and subgroups which are algebraic in nature.

We first show that action of G on Ω induces a partition to form a block system.

Definition 16.4 (Block System)**.** *A partition of Ω into sets such that each part is a block under the action of G.*

To see this, we will explore some properties of blocks.

[8] That is H is not G or $\{id\}$

Claim 16.5. *Let $G \leq S_n$ be a group acting on Ω and $\Delta \subseteq \Omega$ be a block. Then for any $g \in G$, the following holds.*

1. *Δ^g is also a block*
2. *$|\Delta^g| = |\Delta|$*

Proof. **Proof of 1** We give a proof by contradiction. Suppose there exists a $g \in G$ for which Δ^g is not a block. By definition, since Δ^g is not a block, there exists an $h \in G$ such that

$$\Delta^{gh} \neq \Delta^g \text{ and } \Delta^{gh} \cap \Delta^g \neq \emptyset$$

Hence there exists $\beta \in \Delta^g$ but $\beta^h \notin \Delta^g$ and there is a $\gamma \in \Delta^{gh} \cap \Delta^g$. Since $\beta \in \Delta^g$, there exists an $\alpha \in \Delta$ such that $\alpha^{gh} \notin \Delta^g$. Hence $\alpha^{ghg^{-1}} \notin \Delta$ while $\alpha^{ghg^{-1}} \in \Delta^{ghg^{-1}}$. Hence $\Delta \neq \Delta^{ghg^{-1}}$.

Since $\gamma \in \Delta^g$ and $\gamma \in \Delta^{gh}$, there exists an $\omega, \delta \in \Delta$ such that $\gamma = \omega^g = \delta^{gh}$. Hence $\omega = \gamma^{ghg^{-1}} \in \Delta^{ghg^{-1}}$. Along with the fact that $\omega \in \Delta$, we get that $\Delta \cap \Delta^{ghg^{-1}} \neq \emptyset$. But $\Delta \neq \Delta^{ghg^{-1}}$ and $\Delta \cap \Delta^{ghg^{-1}} \neq \emptyset$ contradicts the fact that Δ is a block. Hence it must be that Δ^g is also a block.

Proof of 2 This follows from the fact that $g \in G$ is a permutation and it is a bijection map. □

If the group action is transitive, then Ω is partitioned by the blocks.

Claim 16.6. *For a group $G \leq S_n$ that acts transitively on Ω with $\Delta \subseteq \Omega$ as a block, the following holds.*

1. *$\bigcup_{g \in G} \Delta^g = \Omega$*
2. *For any $g_1, g_2 \in G$, $\Delta^{g_1} \cap \Delta^{g_2} \neq \phi \Rightarrow \Delta^{g_1} = \Delta^{g_2}$.*
3. *$|\Delta|$ must divide $|\Omega|$.*

Proof. **Proof of 1** Let $k \in \Delta$. For any $k' \in \Omega$, since the action is transitive $\exists g$ such that $k^g = k'$ giving $k' \in \Delta^g$. Hence $\Omega \subseteq \bigcup_{g \in G} \Delta^g$. The other containment is direct and hence they are equal.

Proof of 2 This is true since, with Δ^{g_1} being a block (Claim 16.5(1)), either $\Delta^{g_1} = \Delta^{g_1(g_1^{-1}g_2)}$ or $\Delta^{g_1} \cap \Delta^{g_1(g_1^{-1}g_2)} = \emptyset$.

Proof of 3 The above two claims prove that Ω is partitioned into subsets with equal cardinality and all of them have the same size as $|\Delta|$ (Claim 16.5(2)). So $|\Delta|$ must divide $|\Omega|$. □

We get an immediate sufficient condition on Ω for the action of any G to be primitive.

Corollary 16.7. *If $|\Omega|$ is prime, then the action of any $G \leq S_n$ on Ω is bound to be primitive.*

Before ending, let us see why transitive actions are important. For a general group G whose action is not necessarily transitive, if we look at the action on elements of an orbit, the action becomes transitive. Hence we can use our divide and conquer strategy to build blocks and at least obtain automorphism group for the orbits.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Mitali Bafna (*TA:* K Dinesh)
Date: Sep 1 2015
Status: γ

Lecture 17

Characterisation of Primitivity

In the last lecture, we saw transitive and primitive group action. In this lecture, we give an algebraic characterization of primitive group actions.

We need the notion of maximal subgroup of a group.

Definition 17.1 (Maximal Subgroup)**.** *$H \leq G$ is a maximal subgroup if there does not exist an H' such that $H \lneq H' \lneq G$ where $\lneq$ denotes strict subgroup containment.*

The main theorem is the following.

Theorem 17.2. *Suppose $G \leq S_n$ acts transitively on Ω. Then the action is primitive if and only if $\exists \alpha \in \Omega$ such that G_α is the maximal subgroup of G where G_α is the stabilizer of α*

Recall the action (in the last lecture) for $G \leq S_n$ where G acts on itself. We saw that if G has any non-trivial subgroup H', the action is not primitive. If H' has a super group H then, there is a nice relation between the cosets induced by H and H' on G described as an exercise below.

Exercise 17.3 (Problem Set 0, Question 2)**.** *Let $H' \lneq H \lneq G$. The cosets induced by H and H' partitions G. Show that the partition induced by H' is a refinement of the partition induced by G. That is, show that every cosets of H' in G must be contained in some coset of H in G.*

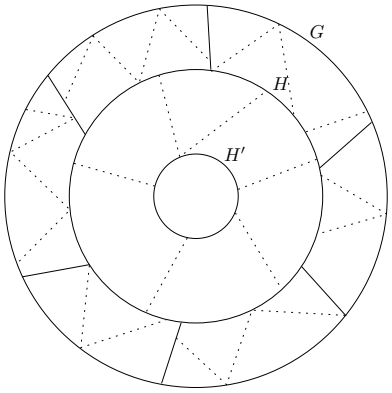

FIGURE 17.3: Coset structure of subgroups H and H'. Dotted ones are cosets of H'.

With this structure in place, we can also consider the cosets of H' as the elements and consider the action of G on this set. Since H has a smaller subgroup H', there are non-trivial blocks for this action. The action moves all those cosets of H' that fall in a coset of H. This must happen as this is precisely the action of G on the cosets of H. Hence the collection of cosets of H' within a cost of H forms a block. The characterization theorem for primitive action tells that if H' does not have a larger subgroup H, then the action is bound to be primitive. We state this as a corollary.

Corollary 17.4. *For $H' \lneq G$ and H' does not have a super group H in G, the action of G on the cosets of H' in G must be primitive.*

Before going to the proof, we first show that the following variant of Theorem 17.2 where $\exists\alpha$ is replaced by $\forall\alpha$ is also true. For this, it suffices to prove the following claim.

Claim 17.5. *For a group G acting transitively on Ω, if there exists an $\alpha \in \Omega$ such that G_α is maximal subgroup of G, then for all $\beta \in \Omega$, G_β is also maximal subgroup of G.*

Proof. Suppose there exists α such that G_α is maximal subgroup of G. Since G is transitive, for any $\beta \in \Omega$, there exist a g such that $\alpha^g = \beta$. Note that G_α and G_β can be related as $g^{-1}G_\alpha g = G_\beta$. This is because,

$$\begin{aligned}
G_\beta =& \{g' \mid \beta^{g'} = \beta\} \\
=& \{g' \mid \alpha^{gg'} = \alpha^g\} \\
=& \{g' \mid \alpha^{gg'g^{-1}} = \alpha\} \\
=& \{g^{-1}hg \mid \alpha^h = \alpha\} && \text{[Renaming } gg'g^{-1} \text{ as } h\text{]} \\
=& g^{-1}G_\alpha g
\end{aligned}$$

Now if G_β has a larger subgroup containing it in G, then we can give a larger subgroup for G_α also which contradicts maximality[9]. ∎

The pair of groups G_α, G_β in the previous claim are sometimes called as conjugate pairs.

Definition 17.6 (Conjugates)**.** *Subgroups H, H' of a group G are said to be conjugates, if there exists a $g \in G$ such that $gHg^{-1} = H'$.*

17.1 Proof of characterization

We restate the characterization in the light of Claim 17.5.

Theorem 17.7. *Suppose $G \leq S_n$ acts transitively on Ω. Then the action is primitive if and only if $\forall\alpha \in \Omega$ such that G_α is the maximal subgroup of G where G_α is the stabilizer of α*

Note that both these theorems are equivalent. We now give a proof.

Proof of Theorem 17.7. The backward direction ($\Longleftarrow$): if $\forall\alpha \in G$, G_α is maximal the maximal subgroup of G, then the action is primitive.

[9] If there is an H' such that $G_\beta \lneq H' \lneq G$ then consider $g^{-1}H'g$ as it satisfy $G_\alpha \lneq g^{-1}H'g \lneq G$.

We will prove the contrapositive that is, suppose that the action is not primitive then $\exists \alpha \in G$ such that , G_α is not a maximal subgroup.

Suppose G does not act primitively on Ω. Then there must be non-trivial block Δ and $\alpha \in \Omega$ such that $\{\alpha\} \subsetneq \Delta \subsetneq \Omega$. We need to show that there exists an H such that $G_\alpha \lneq H \lneq G$. We show this in two steps.

Show that $G_\alpha \leq H \leq G$: Consider $H = \mathsf{SetStab}(\Delta) = \{g \in G \mid \Delta^g = \Delta\}$. We now show that G_α is a subgroup of H. To show this, let $g \in G_\alpha$. Hence $\alpha^g = \alpha$ and α is a common element in Δ and Δ^g. Along with the fact that Δ is a block, we get that $\Delta = \Delta^g$. Hence g fixes Δ which gives that $g \in H$. Hence $G_\alpha \leq H$. By definition of H, we have $H \leq G$.

Both containments are strict : Let $\beta \neq \alpha$ belongs to Δ. Such a β must exist as $\{\alpha\} \subsetneq \Delta$. Since G is transitive, there exists a $g \in G$ such that $\alpha^g = \beta$. Clearly $g \notin G_\alpha$. Now since $\beta \in \Delta \cap \Delta^g$ and Δ is a block, we get $\Delta^g = \Delta$ and $g \in H$.

Hence $G_\alpha \lneq H \lneq G$ and G_α is not maximal.

The forward direction ($\Longrightarrow$): We again prove the contrapositive : if $\exists \alpha \in \Omega$ such that G_α is not a maximal subgroup, then the action is not primitive (i,e. come up with a non-trivial block).

Suppose $\alpha \in \Omega$ be such that $G_\alpha \lneq H \lneq G$. We will produce a Δ such that Δ is a non-trivial block. We define $\Delta = \alpha^H$. Now we are left to show that the Δ chosen is indeed a non-trivial block. To prove this, we need to show the following:

$\{\alpha\} \subsetneq \Delta$: From the Orbit-Stabiliser lemma (Lemma 5.2), we have that each coset of G_α in G corresponds to a different element in the orbit of α. That is all the elements in a coset take α to the same element and if two cosets are different then their action on α is too. Mathematically, for $g_1, g_2 \in G$, $g_1 G_\alpha = g_2 G_\alpha \Leftrightarrow \alpha^{g_1} = \alpha^{g_2}$.

Since $G_\alpha \lneq H, \exists$ a coset $C \neq G_\alpha \in H$. Let $C = G_\alpha h$ where $h \in H$ is the coset representative. Now, $G_\alpha \neq C \Rightarrow \alpha \neq \alpha^h = \beta$ for some $\beta \in \Omega$. But $\beta = \alpha^h \in \alpha^H = \Delta$. So we have that $\{\alpha\} \subsetneq \Delta$.

$\Delta \subsetneq \Omega$: Since $H \lneq G$ and $G_\alpha \lneq H, \exists$ a coset C of G_α in G which does not belong to H. Let $g \in G$ be its coset representative. Since different cosets take α to different elements $\alpha^g \notin \alpha^H$. So we have that $\Delta = \alpha^H \subsetneq \alpha^G = \Omega$.

Δ is a block : We show that for any $g \in G$, either $\Delta^g = \Delta$ or $\Delta^g \cap \Delta = \emptyset$. Suppose $\Delta^g \cap \Delta = \emptyset$, we are done. Suppose $\Delta^g \cap \Delta \neq \emptyset$. We then show that $\Delta^g = \Delta$.

Recall that $\Delta = \alpha^H$. Since $\Delta^g \cap \Delta \neq \emptyset$,

$$
\begin{aligned}
&\exists h', h \in H, \alpha^{h'g} = \alpha^h \\
\Rightarrow &h' g h^{-1} \in G_\alpha \\
\Rightarrow &g \in H && \text{[Using } G_\alpha \text{ is a subgroup of } H\text{]} \\
\Rightarrow &\Delta^g = \alpha^{Hg} = \alpha^H = \Delta
\end{aligned}
$$

This completes the proof. ∎

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Arinjita Paul (*TA:* Ramya C)
Date: September 2, 2015
Status: γ

Lecture
18

Characterising Primitive Actions(Continuation), Minimal Block Problem

18.1 Recap

In the last lecture, we discussed about the algebraic characterization of primitive actions. We add to the previous lecture by checking for primitivity of the transitive action of G on Ω, and further continue with an algorithm that finds out the minimum sized block containing $\{\alpha, \beta\}$.

Problem 18.1. *Let $G = \langle A \rangle$ act transitively on Ω. How do we check if the action is primitive?*

To check if the group G acts transitively on Ω we can check for every pair of elemets $\alpha, \beta \in G$ if $\beta \in Orbit(\alpha)$.

Now if $G = \langle A \rangle$ act transitively on Ω, then the action is not primitive (imprimitive) if there are non-trivial blocks. As any non-trivial block cannot be a singleton, any non-trivial block must have atleast 2 elements. In other words,

Claim 18.2. *If the action of G on Ω is not primitive, then for every $\alpha \in \Omega$, there exists $\beta \neq \alpha$ such that $\{\alpha, \beta\}$ is contained in a non-trivial block.*

Proof of the above claim is left as an exercise.

Hence, for every $\alpha \in \Omega$ there must be a β such that α and β are contained in a non-trivial block. We can run over all pairs $\alpha, \beta \in \Omega$ and check if they are present in any non-trivial block. We solve a problem called min-block problems that helps us answer the question.

18.2 Min Block Problem

Problem 18.3. *Given an $\{\alpha, \beta\} \in \Omega$, compute the smallest block containing $\{\alpha, \beta\}$.*

We solve the Min-block problem in order to compute the smallest block that contains α as follows.

Let $\alpha, \beta \in \Omega$. Consider the undirected graph $X_{\alpha,\beta} = (V, E)$, where $V = \Omega$ and the edge set E is defined as follows

$$E = \{(\alpha^g, \beta^g) \mid g \in G\}$$

Now let us discuss the algorithm to solve the minblock problem

Step 1 : Construct the graph $X_{\alpha,\beta}$.

Step 2 : Output the connected component C in $X_{\alpha,\beta}$ containing α.

As $e \in G$ by definition of $X_{\alpha,\beta}$ any connected component containing α also contains β.

We will now argue correctness of the above algorithm.

Lemma 18.4. *Let $\alpha, \beta \in \Omega$ and $X_{\alpha,\beta}$ be the graph defined above. Then $G \leq Aut(X)$.*

Proof. We know that G and $Aut(X_{\alpha,\beta})$ are both groups. Therefore it suffices to show that $G \subseteq Aut(X_{\alpha,\beta})$. Let $g' \in G$. To show that $g' \in Aut(X_{\alpha,\beta})$. That is we need to show the following :

(i) For any edge $e \in X_{\alpha,\beta}$, $e^{g'} \in X_{\alpha,\beta}$ for any $g' \in G$.
Let $e = (\alpha^g, \beta^g) \in E$. Then for any $g' \in G$ we have $e^{g'} = (\alpha^{gg'}, \beta^{g'g}) = (\alpha^{g''}, \beta^{g''})$ for some $gg' = g'' \in G$. But observe that the edge e' would be there in $X_{\alpha,\beta}$ by definition of $E(X_{\alpha,\beta})$.

(ii) For any $\gamma, \delta \in V$, if $(\gamma, \delta) \notin E$, then $(\gamma^g, \delta^g) \notin E$ for any $g' \in G$.
Let $\gamma, \delta \in V$ such that $(\gamma, \delta) \notin E$ and let there exists $g' \in G$ such that $(\gamma^{g'}, \delta^{g'}) \in E$. As G is a group $g'^{-1} \in G$. Therefore by definition of $X_{\alpha,\beta}$ we have $(\gamma^{g'g'^{-1}}, \delta^{g'g'^{-1}}) = (\gamma, \delta) \in E$ which is a contradiction.

□

As a result, we have $G \leq Aut(X)$.

Claim 18.5. *Let C be a connected component in $X_{\alpha,\beta}$ containing α output by the algorithm. Then C is the minimum block containing $\{\alpha, \beta\}$.*

Proof. **(i) C is a block**
By Claim 18.4 $G \leq Aut(X)$. Therefore $\forall g \in G$ we have $g \in Aut(X)$. Let C be a connected component in $X_{\alpha,\beta}$. Suppose C is not a block then $C^g \cap C \neq \emptyset$ and $C^g \neq C$. Hence there exists a vertex γ such that $\gamma \in C^g$ and $\gamma \in C$. As $C^g \neq C$ there exists a δ such that $\delta \in C^g$ such that $\delta \notin C$. Observe that by definition of C^g we have that (γ, δ) is an edge in $X_{\alpha,\beta}$. Since $\gamma \in C$ and (γ, δ) is an edge in $X_{\alpha,\beta}$ we have $\delta \in C$ which is a contradiction.

(ii) C is the minimum block
Suppose not. Let $C_1 \subsetneq C$ be a smaller block in $X_{\alpha,\beta}$ containing α. Therefore, there exists an edge $(\gamma, \delta) \in E$ such that $\gamma \in C$ and $\delta \in C \setminus C_1$. By definition of $X_{\alpha,\beta}$ there must exist a $g \in G$ such that $\alpha^g = \gamma$ and $\beta^g = \delta$. Now observe that $\gamma \in C_1 \cap C_1^g$ and $\delta \in C_1^g \setminus C_1$ which is a contradiction to the fact that C_1 is a block. □

One question left to answer is that if the graph $X_{\alpha,\beta}$ be efficiently constructed ? We are only given as input the generating set A of G and not the entire group. So we now need to modify the definition of the graph $X_{\alpha,\beta}$ so that it can be efficiently constructed.

We can compute minblock if E is defined based on G. Replacing G with A,suppose we define E as:

$$E = \{(\alpha^g, \beta^g) \mid g \in A\}$$

Suppose we only put the edges corresponding to A. The above claim still holds as there are a fewer edges present in the modification. The bigger set of edges itself are preserved by any element in G. Hence, G would still be an automorphism subgroup of the automorphism group of the modified graph.

It still holds that connected components has to be blocks as every element in the group is a subset of the automorphism group.

Given α and β,there is going to be a block that contains α and β. There always exists an edge between α and β and are contained in the same component always. With our algorthm,we are only interested in finding the smallest connected component containing α and β.

We consider a graph $H : \Omega \times \Omega$ defined as (V', E') where,

$$E' = \{(\alpha, \beta), (\gamma, \delta) \mid \exists g \in A \quad such \quad that \quad \alpha^g = \gamma, \beta^g = \delta\}$$

In the graph H, we consider (α, β) and (γ, δ) as vertices . We put an edge if there is a $g \in A$ that demonstrates that we can go from α to γ and β to δ simultaneously.

Claim 18.6. *Given a graph H, the edgeset E can be constructed.*

Proof. Considering some α, β,α' and β', if there is a path from α and β to α' and β' , it means that there is a group element that takes α to α' and β to β' simultaneously.We compute the transitive closure of the graph, and define the edges in E.

We define an edge α' β' as an edge in E if there is a path between (α, β) and (α', β') in H. This implies that there is a group element that takes α to α' and β to β'. In other words, $\alpha^g = \alpha'$ and $\beta^g = \beta'$. Hence, given the generating set, the graph can be computed. □

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Arinjita Paul (*TA:* Ramya C)
Date: September 04, 2015
Status: γ

Lecture
19

Special Case of Set Stabiliser Problem for Trivalent Graph

19.1 Recap

In the previous lecture, we talked about an algorithm to find the minimal blocks given a graph $G(V, E)$. Given α and β, there shall always be a block containing α and β. We were particularly interested in finding the smallest connected component containing α and β.

19.2 Special Case of Graph Automorphism Problem

We solve a special case of the Graph Automorphism Problem that reduces to the special case of Set Stabiliser Problem.

$$GA \leqslant SetStabProblem = GroupIntersection$$

1. Case when $d \leqslant 1$(trivial): In this case the graphs are only sets of vertex-disjoint edges(matchings) and isolated vertices. Given two graphs X_1 and X_2, we count the number of edges on both. If the count is same, then there is an isomorphism.

2. Case when $d \leqslant 2$: In such a case, there shall exist only paths and cycles.

   Given two graphs X_1 and X_2, we classify the cycles in the graphs of different sizes. Any cycle of length k in X_1 can be mapped to any cycle of length k in X_2.

3. Case when $d \leqslant 3$: A graph is said to be *Trivalent* if the degree of every vertex in the graph is atmost 3.

   We may try to identify the connected components, say C_{11}, $C_{12} \cdots$ in graph X_1 and C_{21}, $C_{22} \cdots$ in graph X_2 and try mapping the connected components by bruteforce. This leads to an obvious problem of a large number of connected components.

Idea :

Consider two graphs X_1, X_2. Let (p_1, q_1) be an edge in X_1 and (p_2, q_2) be an edge in X_2. We split both the edges into two and connect the mid-vertices by a special edge e. Let this be the gadget to construct the new graph X. If X_1 and X_2 are both trivalent then X should be trivalent. Now observe that any isomorphism that maps $(p_1, q_1) \in X_1$ to $(p_2, q_2) \in X_2$ must fix the edge e.

Following is an simpler question that we are interested in. Find the isomorphisms that preserve edge e. Infact we will be interested in the automorphisms of X hat preserves e.

Our approach is:

- We range (p_1, q_1) over all possible edges in X_2.
- Find out the generating set of the automorphism group individually for each.
- Take union.

We consider and name the special edges. Say, graph X_2 has l edges, then by ranging (p_2, q_2) over all possible edges in X_2 for X_1 one by one, we shall have the special edges as:

$$e_1, e_2, e_3, \cdots, e_l$$

Observation 19.1. *The only intersection possible for the automorphism groups that we get is identity.*

$Aut_{e_1}(X), Aut_{e_2}(X), \cdots, Aut_{e_l}(X)$ are all disjoint except for identity.

Problem 19.2. *Given a graph X is trivalent and a distinguished edge e, output the generating set of $Aut_e(X)$ that preserves the edge e in the undirected graph X.*

$Aut_e(X)$ is a special, highly structured group.

Theorem 19.3 (Tutte's Theorem)**.** *If X is a trivalent and connected graph, $Aut_e(X)$ is a 2-group. That is size of $Aut_e(X)$ is 2^p for some p.*

It outputs the set of all automorphisms of X which preserves the edge. We try to build chains of subgroup bottom-up, and we define these chains of subgroups based on the graph structure. Given an X that is trivalent and connected and given an edge e, we want some inductive structure based on e and in turn, tower of subgroups.

For every r, let X_r be the subgraph of vertices and edges in X which appear in a path passing through e of length $\leqslant r$.

The idea is to find $Aut_e(X)$ incrementally. We thus have a sequence of subgroups as:

$$Aut_e(X), \cdots, Aut_e(X_r), Aut_e(X_{r-1}), Aut_e(X_{r-2}), \cdots, Aut_e(X_2), Aut_e(X_1)$$

X_1 contains only 2 vertices and an edge. Therefore $Aut_e(X_1)$ is thus easy to compute. Now we need a method to ascend upwards in the chain.

Observation 19.4.

$$Aut_e(X_i) \leqslant Aut_e(X_{i+1})$$

We use group homomorphisms defined in the next secion to do this.

19.3 Introduction to Group Homomorphism

Definition 19.5. *Let $G, H \leqslant S_n$. A function $\Phi : G \to H$ such that,*

$$\forall g_1, g_2 \in G, \Phi(g_1 \cdot g_2) = \Phi(g_1) \cdot \Phi(g_2)$$

is called a group homomorphism.

Observation 19.6. *1. If $G, H \leqslant S_n$, it implies that $\Phi(e_G) = \Phi(e_H)$ That is, $\Phi(e) = e$ as $G, H \leqslant S_n$ We can say, $\Phi(g_1) \cdot \Phi(g_1^{-1}) = \Phi(g_1 g_1^{-1})$, which is equal to $\Phi(e)$.*

2. $\Phi(g_1^{-1}) = (\Phi(g_1))^{-1}$

3. $\Phi(g)\Phi(e) = \Phi(ge) = \Phi(g)$

We define a set as $\{g \in G \mid \Phi(g) = e\}$, which is called as Kernel of Φ. The set of elements that get mapped to identity is Kernel. We can say that, identity from G maps to H. However, other elements can also be mapped, and the mapping need not be onto.

Claim 19.7. *$Ker(\Phi)$ is a normal subgroup of G.*

Claim 19.8. *$Im(\Phi)$ is a subgroup of H (may not be a normal subgroup).*

Claim 19.9. *Every element in $\Phi(G)$ corresponds to a coset of $Ker(\Phi)$ in G.*
That is,there exists a one to one mapping between $Im(\Phi)$ and $G/Ker(\Phi)$.

Claim 19.10. *The size of the kernel times the size of the image equals size of G.*
$|Ker(\Phi)|/|Im(\Phi)| = |G|$ $(using \quad Lagrange's \quad Theorem)$.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Jayalal Sarma (*TA:* Jayalal Sarma)
Date: Sep 5, 2015
Status: α

Lecture
20

Reduction from Trivalent Graph Isomorphism to Restricted Set Stabilizer Problem

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Sanjay Ganapathy (*TA:* K Dinesh)
Date: September, 08 2015
Status: β

Lecture
21

Structure of groups for d-degree graphs

In the previous lecture, we were interested in finding the generating set for the automorphism group of X which is given to be a graph bounded by maximum degree 3. We in turn reduced this problem to solving the set stabilizer problem for the special case where the group is a 2-group that is $|G| = 2^k$. Given G, a 2-group acting on Ω and $\Delta \subseteq \Omega$, find the generating set of setstab (Δ). In order to solve the set stabilizer problem for this group, first we shall characterize the structure of groups for graphs with maximum degree d.

Definition 21.1 (**Composition Series**). *If a given group G has a sequence of subgroups G_1, G_2, ... G_k, such that*
$G = G_0 \triangleright G_1 \triangleright G_2 \ldots \triangleright G_k = \{identity\}$
and $\forall$ i, G_{i+1} is a maximal strict normal subgroup of G_i, then such a series is referred to as a composition series of G.

G_{i+1}/G_i is called composition factor.

Claim 21.2. *If a graph X has maximum degree $\leq$ d, then $Aut_e(X)$ has a composition series with quotient groups isomorphic to S_{d-1}.*

However our interest is in the case when d = 3 and $Aut_e(X)$ is a 2-group. We will make use of Sylow's theorem to prove the existence of a composition series for such a group where the size of each subgroup G_i will be 2^{k-i} where k is such that $|G| = 2^k$.

21.1 Sylow's Theorem

Theorem 21.3 (**Sylow's Theorem**). *If G is a group of size $p^m r$ where p is a prime and r is such that gcd (p,r) = 1, then $\exists$ H, such that $H \leqslant G$ and $|H| = p^m$*

However, this version of Sylow's theorem is not directly applicable to our problem, since in our problem, $|G| = 2^k$. Instead, we prove this restated version of Sylow's theorem.

Theorem 21.4. *If $p^l \mid |G|$ then there exists a subgroup of size exactly p^l for G.*

Proof. Consider $|G| = p^m r$ where p^k divides r but p^{k+1} does not divide r.

Consider the set $\Omega = \{$ set of all subsets of G of size p^m $\}$. Note this is not the Ω referred to at the starting of the lecture.

Action of g ∈ G on any subset of size p^m (assume action by left multiplication) will take the subset to some other subset of size p^m, that is it will be another element belonging to Ω.

Consider A ∈ Ω, that is A is such that A G and $|A| = p^m$.

$$A^g = \{\ ga \mid a\ \in\ A\ \}$$

We know that

$$|\Omega| = \binom{p^m r}{p^m}$$

Theorem 21.5 (**Lucas's Theorem**). *If there is a prime p and positive integer r, such that gcd (p,r) = 1, then* $p \nmid \binom{p^m r}{p^m}$

This can be extended for the case where gcd (p,r) ≠ 1, as follows.

Theorem 21.6. *If there is a prime p and positive integer r such that* $p^k \mid r$ *and* $p^{k+1} \nmid r$, *then* $p^{k+1} \nmid \binom{p^m r}{p^m}$.

Proof. The proof is quite simple. We just have to expand the term $\binom{p^m r}{p^m}$

$$\binom{p^m r}{p^m} = \frac{p^m r \times (p^m r - 1) \times (p^m r - 2) ... \times (p^m r - (p^m - 1))}{p^m \times (p^m - 1) \times (p^m - 2) ... \times (p^m - (p - 1))}$$

We directly observe a natural numerator-denominator pair structure emerge. We know p^k divides r. So, we just have to prove p does not divide the remaining number. Consider any numerator-denominator pair:

$$\frac{p^m r - i}{p^m - i}$$

If p^α divides the numerator, we know α is definitely less than m because $i < p^m$, and p^m cannot divide the numerator. So, if p^α divides the numerator, we observe that it will also divide the denominator. This follows from the observation that p^m is divisible by p^α and so i must also be divisible by p^α. Therefore $p^m - i$ is also divisible by p^α. Thus p does not divide the remaining number and so p^{k+1} does not divide $\binom{p^m r}{p^m}$. □

Consider θ_1, θ_2, ..., θ_K to be the orbits of Ω.

$$\Rightarrow |\Omega| = |\theta_1| + |\theta_2| + ... + |\theta_K|$$

Since p^{k+1} does not divide LHS, there exists at least one i such that p^{k+1} does not divide $|\theta_i|$. Since $A \in \theta_i$, orbit(A) = θ_i.

Consider the pointwise stabilizer of A, G_A. We know from orbit-stabilizer lemma that

$$|orbit(A)| \times |G_A| = |G|$$

p^{m+k} divides the RHS. We know p^{k+1} does not divide orbit(A). Therefore $|G_A|$ must be divisible by p^m. We'll now proceed to show that $|G_A|$ is in fact equal to p^m.

Claim 21.7.

$$|A| \geq |G_A|$$

Proof. Suppose $|G_A| > |A|$. We know any element $g \in G_A$ stabilizes A, that is every element A is mapped to some other element in A, by action of g by left multiplication. Let's take an element g_i belonging to A. There are at most $|A|$ different elements that g_i can go to. Since $|G_A|$ ¿ $|A|$, by pigeon hole principle we have that there exists at least 2 elements, say g_j and g_k belonging to G_A such that they map g_i to the same element say g'.

$$g_j * g_i = g_k * g_i$$

By right multiplication with g_i^{-1} on both sides, we will get that g_j and g_k are in fact equal. Thus, we arrive at a contradiction. □

Since $|A| = p^m$, this implies

$$p^m \mid |G_A| \text{ and } p^m \geq |G_A|$$

Therefore $|G_A| = p^m$. Thus for any p^l such that p^l divides $|G|$, we can prove the existence of a subgroup of size p^l.

Coming back to our original problem of proving the existence of a composition series for $Aut_e(X)$, since this is a 2-group of size 2^k, by Sylow's theorem we know that a normal subgroup of size 2^{k-1} exists for $Aut_e(X)$. We can inductively extend this proof to each subgroup, G_i and thus prove the existence of a composition series.

In general, for any group G, G $\in B_d$ if there exists a chain of normal subgroups such that each composition factor is isomorphic to S_{d-1}. We call such a group to be structured. For the trivalent graph case, d = 3 and hence the composition factor is isomorphic to S_2, which is what we have shown. Here S_{d-1} refers to a symmetric group on $d-1$ elements. □

21.2 Structure Forests

We will solve the problem of finding a generating set for setstab (Δ) using a recursive algorithm on a structure tree.

21.2.1 Motivating Structure Trees

Consider the same complete binary tree of depth k we had used while motivating groups and transitive actions, where we consider the existence of a group G such that $G \leqslant S_n$ and G acts transitively on Ω, where Ω is the set of leaves. Each internal node of the tree can be thought of as corresponding to a subset of Ω, which is a block. In general, when we have two vertices u and v such that u is the child of v, the following condition holds:

u is a child of v if Δ_u is a maximal block in Δ_v. It means that there does not exist a Δ such that

$$\Delta_u \subset \Delta \subset \Delta_v$$

Observation 21.8. *When G is general, you get a structure forest, with each tree corresponding to an orbit. This follows from the fact that G acts transitively on an orbit by definition.*

In the next lecture, we'll formally define structure forests and use them to formulate a recursive solution for the restricted set stabilizer problem and prove that the running time of the algorithm is bounded by a polynomial for d degree graphs.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Sanjay Ganapathy (*TA:* K Dinesh)
Date: September, 09 2015
Status: β

Lecture
22

Solving Restricted Set Stabilizer Problem

We will formalize the notion of structure forests that we motivated in the last lecture.

22.1 Structure Forests

1. Each tree in the forest corresponds to one orbit.
2. Leaves are the elements of Ω, internal nodes correspond to blocks and roots correspond to orbits.
3. A vertex u is a child of vertex v if Δ_u is a maximal block in Δ_v.

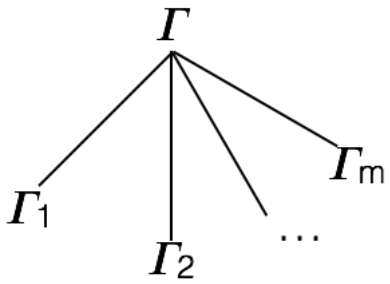

FIGURE 22.4: Each Γ_i is a maximal block in Γ.

Claim 22.1. *Action of G on the set $\Gamma_1, \Gamma_2, ..., \Gamma_m$ is primitive.*

Proof. Consider action of G on the set {1, 2, 3, ... m}. Suppose there was a non-trivial block δ such that

$$\delta \subset \{1, 2, ... \text{ m}\}$$
$$|\delta| \neq 1$$
$$\delta^G \cap \delta = \phi \text{ or } \delta^G = \delta$$
$$B = \bigcup_{i \in \delta} \Gamma_i$$

Thus, B is a block such that $\Gamma_i \subset$ B $\subset \Gamma$. But this contradicts the existence of the edge Γ - Γ_i. □

Consider the partition of Ω into it's orbits.

$$\Omega = \Omega_1 \cup \Omega_2 ... \cup \Omega_k$$

Observation 22.2. *It suffices to find the setstab of $\Delta \cap \Omega_i$ separately and take the union.*

This is because, there does not exist a group element g which can take any element belonging to Ω_i to Ω_j where $i \neq j$, by virtue of the two elements being in different orbits. Thus when Δ is stabilized, elements belonging to an orbit are also constrained to remain in the same orbit.

Thus we have reduced our original problem to finding the generating set of $\Delta \cap \Omega_i$. Thus hereon, we will focus on one orbit and refer to it as Ω and $\Delta \cap \Omega$ as Δ.

22.2 Algorithm for computing setstab(Δ)

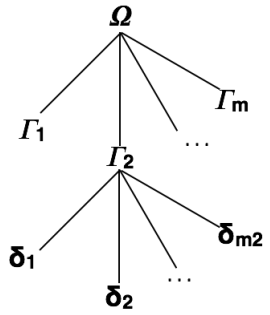

FIGURE 22.5: Structure tree for one orbit of Ω under the action of G.

Consider $H = \{\ g \in G \mid \forall\ i\ \Gamma_i^g = \Gamma_i\ \}$. H is the setwise stabilizer of Γ_i. This means we can write G in terms of the cosets of H.

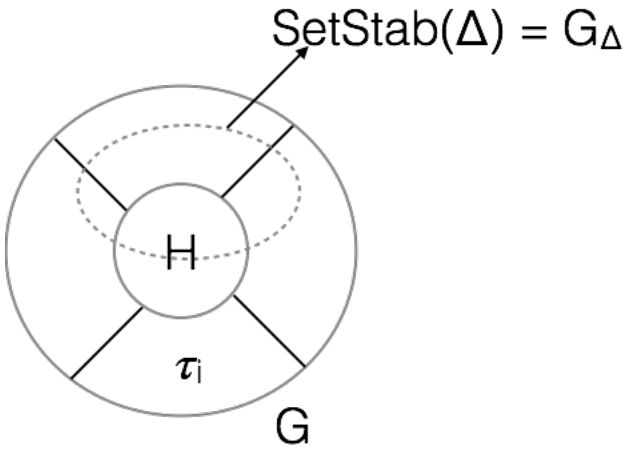

FIGURE 22.6: Set Stabilizer group of Δ in G and the cosets of H, which is the setwise stabilizer of Γ_i.

$$G = \bigcup_{i=1}^{k} H\tau_i$$

$$G_\Delta = \bigcup_{i=1}^{k} (H\tau_i)_\Delta$$

We need to find the elements in each coset of H, which stabilizes Δ. This leads to a natural recursive formulation which we formalize below.

22.2.1 Generalized Set Stabilizer

Given Δ, H and σ, such that H stabilizes Ω', where $\Omega' \subseteq \Omega$, find $(H\sigma)_\Delta$.

If we visualize Δ to be a 2 colouring on the elements of Ω, then we can write

SetStab (H, σ, Ω') = $\{h \in H\sigma \mid \forall\ \omega \in \Omega'\ \omega$ and ω^h has the same colour$\}$ where Ω' is one of Γ_i.

Therefore, we need to recursively solve SetStab($\Gamma_i \cap \Delta, \sigma$), for each level of the tree. To show that this algorithm runs in polynomial time, we make use of the following bound.

Theorem 22.3 (**Babai Luke Palfy Bound**). *If $G \leqslant S_n$ is a structured group, then $|G|$ is bounded by n^c for some constant c.*

We have already proved that $Aut_e(X)$ is a structured group, for a d-degree bounded graph. Thus, we can use this theorem to show that $|G| \leq n^{cd}$. Thus the number of nodes in the tree is polynomial and number of coset representatives is also polynomial since the size of the group itself is bounded by a polynomial. Thus, since the recursion only visits each node of the tree once, and the computation per node is bounded by a polynomial, the entire algorithm is of polynomial order.

More precisely, we can show that we can solve the graph automorphism problem in $O(n^{d^2})$ time.

In the next lecture, we will try to refine the trivial brute force algorithm of finding a graph isomorphism between 2 graphs X_1 and X_2 and develop an algorithm which runs in $O(n^{n^{\frac{2}{3}}})$.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Vidhya Ramaswamy (*TA:* Ramya C)
Date: September, 11 2015
Status: β

Lecture
23

General Graph Isomorphism

23.1 Introduction

In these set of lectures, we consider the general graph isomorphism problem, and improve the solution. We know that we have a trivial solution in $O(n^n)$ time, by checking all possible mappings from graph X_1 to graph X_2. In the previous lecture, we have shown that when the graphs have a degree d bound, we have an $O(n^{d^2})$ time solution.
We now give a $O(n^{n^{\frac{2}{3}}})$ bound for solving the general graph isomorphism problem.

23.2 Colourings and Refinements of Colourings

A colouring of a graph is a function mapping vertices of a graph to integers.

$$f : V \longrightarrow \{1, 2, ...|V|\}$$

To make understanding easier, we assume that $Range(f)$ is some initial segment of $\{1, 2, ...|V|\}$. That is, if we assign k colours to all the vertices, we assume that the k colours used are $\{1, 2, ...k\}$.

We now introduce the idea of a Refinement of a Colouring. We can understand the idea of a refinement empirically as follows. Consider a colouring f of a graph X. This partitions the graph into some colour classes. Now, we say that g refines f if g further partitions each of these colour classes.
Formally, we define refinement as follows:

Definition 23.1. *We say that a colouring f_1 refines a colouring f_2 if $\forall x, y \in V$,*

$$f_2(x) \leq f_2(y) \Rightarrow f_1(x) \leq f_1(y)$$

We write this as $f_1 \leq f_2$

23.3 Algorithm

We notice that if we colour the graphs such that any isomorphism between the graphs would preserve the colouring, then, our job of finding an isomorphism becomes potentially easier, especially if each of the colour classes have very few vertices in them.

We do this by choosing a colouring which preserves the degree information.

1. Initially, all vertices are coloured with the same colour.
2. At each step, we refine this colouring, by encoding the number of vertices of a particular colour which are its neighbours.
3. We continue this refinement process until no changes are made in the colouring. We call this a **stable colouring** of the graph.

Formally, we describe the refinement process as follows:
Let f be a colouring, mapping a vertex x to colour $f(x)$, using k colour classes. Now, we describe g (the refinement of f) as follows.

$$g(x) = (f(x), \chi_1(x), \chi_2(x), ... \quad \chi_k(x))$$

where $\chi_i(x)$ = Number of neighbours of x in colour class i.
We can now order the tuples in some order (maybe lexicographic) and then map them to $\{1, 2, ...|V|\}$.
As we encode $f(x)$ in the beginning of g, g is obviously a refinement of f.

Definition 23.2. *We say that 2 vertices are **not distinguished** if they are in the same colour class for a stable colouring.*

When we get a stable colouring, we observe the following:

Observation 23.3. *For 2 vertices to be not distinguished, they have the same degree within each colour class.*

Observation 23.4. *When the colouring is stable,*

1. *The induced graph of any colour class is regular.*
2. *The bipartite graph between any 2 colour classes is semi-regular. That is, each vertex in the left partition has the same degree, each vertex in the right partition has the same degree, but these degrees need not be the same.*

Note: If 2 vertices are not distinguished, this does **not** imply that an automorphism mapping one vertex to the other is possible. Consider figure 23.7 below. As every vertex in this graph has a degree 3, the refinement process stops after one pass, and all vertices are coloured with the same colour. However, there is no automorphism mapping vertex 1 to vertex 3.
We can also see an example where all vertices are in the same stable colour class, but there are no non-trivial isomorphisms (See 23.8) .

We claim that this process of refinement does not lose any isomorphisms. That is,

Claim 23.5. *If $f_1 \leq f_2$,*

$$(X_1, f_1) \cong (X_2, f_1) \Leftrightarrow (X_1, f_2) \cong (X_2, f_2)$$

We leave the proof of this claim as an exercise to the reader.

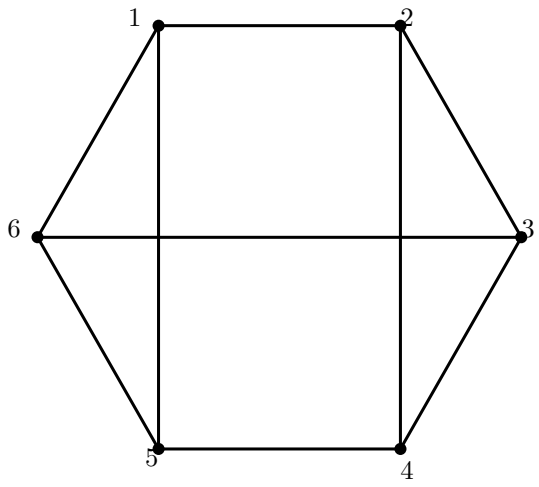

FIGURE 23.7: No automorphism mapping vertex 1 to 3

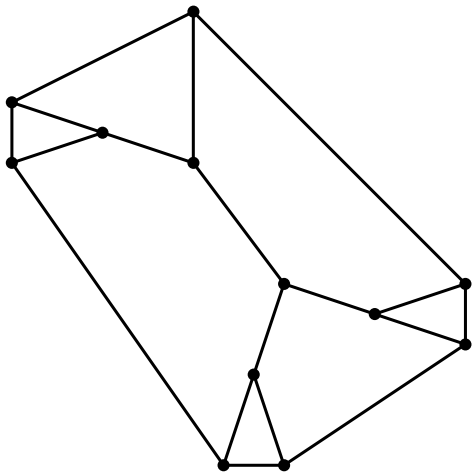

FIGURE 23.8: No non-trivial automorphism

If, after obtaining a stable colouring, we have only one vertex in each colour class, we are done, as this forces each vertex to be mapped to a specific vertex in the other graph. If not, we force singleton colour classes by the idea of **Individualisation**.

23.3.1 Individualisation

For individualisation, we do the following.

1. Choose any vertex x such that it is not in a singleton colour class, and colour it with a new colour.
2. Refine the colouring, until we get a stable colouring again.
3. Repeat steps 1 and 2 until we get singleton colour classes.

We notice that as soon as we start individualising vertices, we lose the property that the colouring preserves all isomorphisms, and hence, need to check it with all possible vertices in the corresponding colour class. In the next lecture, we look at how to pick these vertices to make our search smaller, and complete the proof of the general graph isomorphism.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Vidhya Ramaswamy (*TA:* K Dinesh)
Date: September, 14 2015
Status: β

Lecture
24

General Graph Isomorphism

In order to decide which vertices of graph X_1 to individualise, we introduce the notion of **Colour Valence**.

Definition 24.1. *Colour valence of a graph X with a colouring f is d, if, for every colour class c, and for every vertex $v \in V(X)$, there are atmost d neighbours of v in colour class c or there are at most d non-neighbours of v in the colour class c*

Like having a bounded degree, having a bounded colour valence gives us an interesting bound on checking if 2 graphs are isomorphic.

Claim 24.2. *If (X_1, f_1) and (X_2, f_2) are given coloured graphs of colour valence at most d, then we can check if $(X_1, f_1) \cong (X_2, f_2)$ in time $O(n^{d^2})$.*

Hence, if we can get a colouring of both graphs which preserves isomorphisms, and is of a constant colour valence, we can use the above claim (Claim 24.2) to check if the graphs are isomorphic in polynomial time. To do this, we need to ensure that when we individualise vertices, we reduce the colour degree. The following lemma allows us to do so.

Lemma 24.3. *Given a coloured graph (X, f) with a colour valence less than or equal to d, we can find vertices $x_1, x_2, ...x_k$ where $k \leq \frac{2n}{d}$, such that (X, f') has a colour valence $\leq \frac{d}{2}$ where f' is the stable colouring obtained by individualising (X, f) on vertices $x_1, x_2, ...x_k$.*

Assuming Claim 24.2 and Lemma 24.3, we get algorithm 5.

Hence, we have an algorithm for graph isomorphism, modulo the proofs of Claim 24.2 and Lemma 24.3.

Proof of Lemma 24.3

Proof. Let S be a a set of vertices chosen upto stage i, that is, $S = \{x_1, x_2, ...x_{i-1}\}$. Let the colouring after these individualisations be f_S. Since we are not yet done, colour valence of $(X, f_s) > \frac{d}{2}$.

Therefore, $\exists$ a colour m, and a vertex $x \in V(X)$ such that

$$d_m(x) > \frac{d}{2}$$

Algorithm 5 Algorithm for general GI

procedure GENERAL GI(Input : Graphs X_1, X_2)
    Given X_1, X_2
    Start with a trivial colouring of X_1 and X_2. Refine until stable.
    Choose $(x_1, x_2, ...x_k)_{\log n}$ according to Lemma 24.3 for X_1. This gives us a colouring with colour valence as some constant.
    for $(x_1, x_2, ...x_k)_{\log n}$ in graph X_2 **do**
        if Colour valence of X_2 is not a constant **then**
            break
        else
            Use claim 24.2 to check if X_1 and X_2 is isomorphic.
            If so, output that $X_1 \cong X_2$.
    Output that $X_1 \ncong X_2$

and

$$cod_m(x) > \frac{d}{2}$$

where $d_m(x)$ is the number of neighbours of x in colour class $f_S^{-1}(m)$, and $cod_m(x)$ is the number of non-neighbours of x in colour class $f_S^{-1}(m)$. We make the following claim:

Claim 24.4. *We can choose an x_i such that x_i violates the colour valence property, and, $\forall j < i$*

$$N(x_i) \cap N(x_j) = \phi$$

where $N(x)$ is the neighbour set of a vertex x in X.

Choosing x_i's as per claim 24.4 suffices. We can sum up the neighbours of all these x_i's, and get the following inequalities:

$$\frac{kd}{2} \leq \sum_{j=1}^{k} |N(x_j)| \leq n$$

Hence, we get

$$k \leq \frac{2n}{d}$$

□

We still need to prove claims 24.2 and 24.4.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Punit Khanna (*TA:* Ramya C)
Date: September, 16 2015
Status: β

Lecture
25

Introduction to Polynomials

25.1 Introduction

So far, we have studied various problems through the framework of group theory. Now we'll move on to the new theme of polynomials, while borrowing structural ideas from what we've studied so far.
We give broad definitions of the problems on polynomials that we're interested in:

Problem 25.1. *Given a system of k n-variate polynomial equtions*

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_k(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

find assignments to the variables $x_1, x_2, \ldots, x_n$ such that they satisfy the system of k equations.

Problem 25.2 (Polynomial Factorization)**.** *Given a polynomial, enumerate its irreducible factors.*

A polynomial f is said to be *identically zero* if all its cofficients are 0.

Problem 25.3 (Polynomial Identity Testing)**.** *Given a polynomial f, test if it is identically 0.*

25.2 Solving multivariate polynomial equations

We begin by giving a formal definition of polynomials:

Definition 25.4. *(Polynomial) A polynomial (in n variables) is an expression which is the sum of terms of the form $\alpha \prod_{i=1}^{n} x_i^{\beta_i}$. Each such term is called a* ***monomial****. α is the* ***coefficient*** *while $\prod_{i=1}^{n} x_i^{\beta_i}$ is the* ***power product****.*

Let us consider linear polynomials. That is, polynomials whose degree is 1. An intuitive way to represent the coefficients of k linear polynomials in n variables is to consider a matrix $A_{k\times n}$. Let X be the column vector $(x_1, x_2, \ldots, x_n)$. Then we can view any system of k linear equations as the matrix $A_{k\times n}$ acting on the left of the column vector X. A system of linear equations can therefore be written as the following:.

$$AX = B$$

Here the column vector B will be 0.
To solve this system of equations, we carry out the familiar process called **Gaussian Elimination**. We use a sequence of elementary row operations on $A_{k\times n}$ to transform it into a upper triangular matrix. Also observe that some of the rows of the matrix thus obtained can be completely zero. However solving the equations is now easy. When we apply this matrix on the column vector, the first non-zero row from the bottom is in only one variable (x_n); the equation resulting from the row just above has two variables (one of which x_n, has already been fixed), and so on.

Inorder to solve a system of linear equations we transfor the matrix representing the system of linear equation by applying elementary row and column operations. Why is this process correct ? Why does it not change the linear equations in the system ? To see this we will need a few basic definitions. We can view the rows of the matrix $A_{k\times n}$ as vectors of the form $(a_1, \ldots, a_n)$.

Definition 25.5. *(Row Space) The set of all the vectors generated by elementary operations carried on the row vectors is called row space.*

Definition 25.6. *(Column Space) The set of all the vectors generated by elementary operations carried on the column vectors is called column space.*

Example 25.7. *If R_1 and R_2 are row vectors that belong to some row space, then $R_1 + \alpha R_2$ also belongs to the same row space (similarly for columns as well), α being a constant.*

Definition 25.8. *(Span) Let R be a set of vectors. Then the set of all linear combinations of the vectors in R is called* Span$[R]$.

Definition 25.9. *(Kernel of a linear transformation.) Let V be a set of vectors. Let ϕ be a linear transformation from V to V. The set of all vectors in V that are mapped to 0 under ϕ is called Kernel of the linear transformation ϕ. (Observe that this analogous to the definition of kernel that we already studied as part of Group Homomorphisms).*

We can see that the elementary row operations we did as part of Gaussian Elimination do not change the row space. In addition, the Kernel of the vector set under some operation, remains unchanged. This therefore establishes the correctness of the transformation done as part of Gaussian elimination.
At this point we ask ourselves a question. How is the kernel related to the row space (or column space, for that matter)?

We can see that for B to be a zero vector, each row vector in A must a dot product with X equal to 0. Only then will the kernel contain X.

So far we were interested in linear combinations of polynomials. NOw we will look at combinations of polynomials by using polynomials as coefficients.

We now define a combination of the polynomials $f_1, f_2, ..., f_k$ as follows:

$$I = \left\{ \sum_{i=1}^{k} g_i f_i \mid \ g_i's \text{ are polynomials} \right\}$$

We can see that the following statement holds true:

Lemma 25.10. *Let $f_1(x_1, x_2, \ldots, x_n), f_2(x_1, x_2, \ldots, x_n), \ldots, f_k(x_1, x_2, \ldots, x_n)$ be polynomials.*

$$\forall i \in [k], (a_1, a_2, a_3....., a_n) \text{ is a solution to } f_i = 0 \Leftrightarrow \forall h \in I,\ h(a_1, a_2, a_3, ..., a_n) = 0 \qquad (25.15)$$

Proof. Since $h \in I$ we have $h = \sum_{i=1}^{k} g_i f_i$. If every f_i is zero on some value, h will be 0 as well. The reverse direction is true because I actually contains every f_i; if every h is zero on some value, every f_i is too. □

Therefore we can see that to calculate a common zero for all f_is, we need a common zero for all polynomials in I. Our aim is to therefore obtain polynomials $l_1, l_2, l_3....., l_{k'}$, such that $I = \{\sum_{i=1}^{k'} g_i l_i |\ g_i s$ are polynomials$\}$ is easy to solve. Here the l_is form a **basis** for I. In particular, we call these a Grobner basis (we will study more about this in subsequent lectures).

25.3 An Introduction to Algebraic Structures

We have already studies what groups are in detail. Now we define the following:

Definition 25.11. *(Ring) A ring is a set of elements that forms an Abelian group under addition, and respects the properties of associativity and closure under multiplication.*

Definition 25.12. *(Division Ring) A ring that contains a multiplicative inverse is called a division ring.*

Definition 25.13. *(Commutative Ring) A ring that is commutative under multiplication is called a commutative ring.*

Example 25.14. *$\mathbb{Z}$ is a commutative ring.*

Definition 25.15. *(Field) A field (denoted by $\mathbb{F}$) is a ring that has a multiplicative inverse, and is commutative under multiplication. (A field is therefore an Abelian group under both addition and multiplication)*

Example 25.16. *A polynomial in n variables with integer coefficients (denoted by $\mathbb{Z}[x_1, x_2, ..., x_n]$) is a commutative ring. Indeed, any polynomial is a commutative ring provided its coefficients are drawn from a commutative ring.*

Example 25.17. *$\mathbb{Z}_7$ is a finite ring.*

Definition 25.18. *(Zero Divisor) A ring R is said to have a zero divisor if $\exists a, b \in R \neq 0$ in the ring such that $ab = 0$.*

Definition 25.19. *(Integral Domain) An integral domain is a ring with no zero divisor.*

Example 25.20. *$\mathbb{Z}$ is an integral domain while $\mathbb{Z}_4$ is not. Any $\mathbb{Z}_p$ where p is prime is a field.*

Theorem 25.21. *Any finite integral domain is a field.*

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Punit Khanna (*TA:* Ramya C)
Date: September, 18 2015
Status: β

Lecture
26

Geometrical Representation of Polynomials

In the previous lecture, we looked at polynomials and defined some basic algebraic structures. Now, we will attempt to formalize some of our ideas as well as introduce the notions of **ideal** and **variety**.

Notation 26.1. *A set of polynomials in n variables whose coefficients are in $\mathbb{F}$ is denoted by $\mathbb{F}[x_1, x_2, x_3, \ldots, x_n]$.*

Observation 26.2. *$\mathbb{F}[x_1, x_2, \ldots, x_n]$ forms a ring. Let us denote $R := \mathbb{F}[x_1, x_2, \ldots, x_n]$.*

The polynomial $5x_1^2x_2 + 7x_1x_2^2 + 8x_1x_2$ is part of $\mathbb{Z}[x_1, x_2]$, which can also be written as $(\mathbb{Z}[x_1])[x_2]$. (Here $\mathbb{Z}[x_1]$ is the ring from which coefficients are drawn).

We can the definition of I from the previous lecture as follows:

$$I = \left\{ \sum_{i=1}^{k} g_i f_i \mid \forall i, g_i \in R \right\} \tag{26.16}$$

We can make the following observations about I:

- $I \subseteq R$
- I follows the closure property
- I has inverse
- I contains the additive identity (0)
- I is a subgroup of R w.r.t. addition
- I is a subring of R w.r.t. both addition and multiplication

Observation 26.3. $RI \subseteq I$

Proof. Let $h \in RI$
$\Rightarrow h = rh'$ such that $r \in R, h' \in I$
$\Rightarrow h = r\sum_{i=1}^{k} g_i f_i$
$\Rightarrow h = \sum_{i=1}^{k} (rg_i) f_i$
$\Rightarrow h \in R$ ∎

Observe that the above property is a stronger closure property than the usual closure property.
Having obtained this characterisation of I as defined above, we give the following definition:

Definition 26.4. *(Ideal) A set $I \subseteq R$ is said to be an ideal of the ring R if :*

- *I is a subgroup of R w.r.t. addition*
- *$RI \subseteq I$*

26.1 Geometry of polynomials

Let $\mathbb{F}$ be a field. We define $\mathbb{F}^n$ as

$$\mathbb{F}^n = \{(a_1, a_2, ..., a_n) \mid a_i \in \mathbb{F}\}$$

We define Variety ($\mathbb{V}$) as the following.
Let $f \in \mathbb{F}[x_1, \ldots, x_n]$. The variety of f denoted by $\mathbb{V}(f)$ is defined as

$$\mathbb{V}(f) = \{(a_1, a_2, ..., a_n) | f(a_1, a_2, ..., a_n) = 0\} \tag{26.17}$$

We also take variety to apply to a set of polynomials.
Let $S \subseteq R$. Then the variety of S denoted by $\mathbb{V}(S)$ is defined as

$$\mathbb{V}(S) = \{(a_1, a_2, ..., a_n) | \forall f \in S, f(a_1, a_2, ..., a_n) = 0\} \tag{26.18}$$

We can see that the Variety is analogous to the root of a function, or rather the set of points at which multiple functions evaluate to 0.

Given a set of polynomials S we defined the set of points that are common zeros of the polynomials S. Now given a set of points let us define the set of polynomials that vanish on those points.
Let $V \in \mathbb{F}^n$:

$$\mathbb{I}(V) = \{h \in \mathbb{F}[x_1, x_2, ..., x_n] | \forall (a_1, a_2, ..., a_n) \in V, h(a_1, a_2, ..., a_n) = 0\} \tag{26.19}$$

Having defined the above, we now pose the following questions:

- What is the relation between V and $\mathbb{V}(\mathbb{I}(V))$?
- What is the relation between I and $\mathbb{I}(\mathbb{V}(I))$?

Problem Set #1

(1.1) (See Exercise 6.4) We will reduce variants of graph isomorphism problem to the original problem.

(a) We defined the graph rigidity problem GR as - given a graph X, test if $Aut(X)$ is trivial. Show that GR $\leq$ GI.

(b) Notice that the GI is defined for undirected graphs. Define the graph isomorphism problem for directed graphs DirGI. Show that DirGI $\leq$ GI.

(1.2) Design a linear-time algorithm that takes two rooted trees T_1 and T_2 as input and tests if they are isomorphic. Note that an isomorphism between T_1 and T_2 must preserve edges as well as parent-child relations.

(1.3) Let G and H be subgroups of S_n. Give a polynomial (in n) time algorithm which given the generators of a group G, and H, write down algorithms for:

(a) Checks if H is a normal subgroup of G.

(b) Normalizer of H in G. That is the largest subgroup of G in which H is a normal subgroup.

(c) Normal closure of H in G, that is the smallest normal subgroups of G that contains H.

(1.4) Let G act transitively on Ω. In class, we showed a characterization of primitive actions. Extend the argument to show the following. Let $\alpha \in \Omega$. Let Γ be the set of all blocks B which contain α and set $\mathcal{H}$ be a set of all subgroups of $H \leq G$ that contain G_α. Establish a bijection between Γ and $\mathcal{H}$.

(1.5) For any group G, the commutator subgroup G' of the groups, sometimes denoted by $[G, G]$, is defined as,

$$[G, G] = \{g_1 g_2 g_1^{-1} g_2^{-1} \mid g_1, g_2 \in G\}$$

(a) Argue that G/G' is an Abelian group. (That is $\forall a, b \in G/G'$, $ab = ba$)

(b) Show that G' is the smallest group such that (a) holds.

(c) Give a polynomial time algorithm for the following problem : Given a generating set for G, compute that of G'. (Hint : Use one of the previous algorithmic problems in this set).

Exercises

(0.1) (See Exercise 12.5) We stated in class that for any $n > 3$, every subgroup of S_n can be generated by at most $\lfloor \frac{n}{2} \rfloor$ elements. Show that this bound is tight by giving an example of an Ω and a group $G \leq S_{|\Omega|}$ acting on it such that G requires $\frac{|\Omega|}{2}$ elements to generate it. (Hint : Consider $\Omega = \{a_1, a_2, \ldots, a_m, b_1, b_2 \ldots b_m\}$).

(0.2) (See Exercise 17.3) Let $H' \lneq H \lneq G$. The cosets induced by H and H' partitions G. Show that the partition induced by H' is a refinement of the partition induced by G. That is, show that every cosets of H' in G must be contained in some coset of H in G.