Algebra and Computation

Ramprasad Saptharishi[*]

TIFR, Mumbai

January 16th, 2017 – May 15th, 2017

Contents

[*]ramprasad@tifr.res.in

Last updated: Thu Apr 27 11:22:38 IST 2017

This course is going to be roughly split into three parts. The first part would deal with group theoretic algorithms and their application to the graph isomorphism problem. The second part would deal with polynomials and factorization algorithms for them. The third part would deal with very basic algebraic geometry from a computational perspective.

Part I: Group theoretic algorithms

Lecture 1: January 16th, 2017

Notation

There may be more to include... will add as they come end up being used

- We shall use S_n to denote the symmetric group on n elements. Sometimes we might also look at a symmetric group action on a set A and we shall denote that group by $\mathrm{Sym}(A)$
- Groups will often be specified by a generating set. We shall denote this by $G = \langle S \rangle$.
- We shall use $H \leq G$ to denote that H is a subgroup of G.
- We shall use $H \lhd G$ to denote that H is a normal subgroup of G, and use $H \trianglelefteq G$ to denote that H is a normal (not necessarily proper) subgroup of G.
- Most of the groups considered in this course would be *acting* on a set Ω. We shall use the standard exponent notation, such as α^g for $\alpha \in \Omega$ and $g \in G$ to denote the image of α under the action of g. This has the useful property that $(\alpha^g)^h = \alpha^{hg}$ etc. The change in order is annoying but I prefer this as I want it to mimic $h(g(\alpha))$.

  In the same notation, α^G will denote the *orbit* of α under the action of G.
- If G acts on Ω and $\alpha \in \Omega$, we shall denote the *stabilizer* of α by $\mathrm{stab}_G(\alpha)$, or just G_α.

1 Introduction

The first part of the course would focus on the *Graph Isomorphism* question.

Definition 1.1 (Graph Isomorphism)**.** *Two graphs on n vertices $G = (V, E)$ and $H = (V, E')$ are said to be* isomorphic *if there is a permutation $\sigma \in S_n$ such that*

$$(v_i, v_j) \in E \iff (v_{\sigma(i)}, v_{\sigma(j)}) \in E'.$$

That is, there is a relabelling of the vertices such that edges and non-edges of G are mapped to edges and non-edges of H respectively.

◊

The computational task GraphIso is the task of checking if the two given graphs G, H are isomorphic or not. What do we know about this problem?

Fact 1.2. GraphIso $\in$ NP $\cap$ co-AM.

This intuitively suggests that GraphIso is unlikely to be NP-complete. In fact, recently there was a huge breakthrough by Babai[1].

Theorem 1.3 (Babai)**.** *There is a deterministic algorithm for* GraphIso *running in time* $n^{\text{poly}\log n}$.

We would not be covering this result as a part of this course (alas!) but we shall however look at the technique of *group theoretic algorithms* developed by Eugene Luks and László Babai to attack this problem.

1.1 *Bringing in group theory*

Let us ask a slightly different question. Suppose G and H were actually the same graph. Now of course they are isomorphic because $\sigma = \text{id} \in S_n$ preserves the edge relations. But are there other non-trivial relabellings of vertices that preserve the edge relations? These are what are called *automorphisms*.

Definition 1.4 (Graph Automorphisms)**.** *For a graph $G = (V, E)$ on n vertices, the* automorphisms of G*, denoted by* $\text{Aut}(G)$*, is the following set of permutations:*

$$\text{Aut}(G) = \left\{\sigma \in S_n \;:\; (v_i, v_j) \in E \iff (v_{\sigma(i)}, v_{\sigma(j)}) \in E\right\}. \qquad \Diamond$$

Observation 1.5. $\text{Aut}(G)$ *is a subgroup of S_n.*

Understanding the group of automorphisms of a group turns out to be very important for the question of graph isomorphism. Here is a concrete reason for it.

Lemma 1.6. *Let $G = (V, E)$ and $H = (V', E')$ be two connected graphs on n vertices. Consider the graph $X = (V \cup V', E \cup E')$ which is just the union of the two graphs. Then, G is isomorphic to H* if and only if *there is a $\sigma \in \text{Aut}(X) \subseteq S_{2n}$ that maps some vertex of G (in X) to a vertex of H (in X).*

Proof. Go figure. □

Thus, if we could somehow get a handle on $\text{Aut}(X)$, we may be able to check if there are automorphisms that sort of swap G and H.

One issue here is that $\text{Aut}(X) \subseteq S_{2n}$ and hence can be a *huge* set (potentially all of S_{2n} which has size $(2n)!$). Can we even hope to answer reasonable questions for such exponential sized groups? In this part of the course, we shall basically be addressing questions of this sort and eventually applying that to construct algorithms for some special cases of GraphIso.

2 Crash course on Group Theory

There is a series of fantastic blog posts by Tim Gowers on group theory. Highly recommended that you go over them at some point.

I guess it is fair to assume that you know what a *group* is — a finite set with an associative binary operation on it, with an identity, and inverses etc.

Here are some basic definitions/properties that one should know about groups.

[1] This result is currently being peer-reviewed.

Theorem 2.1 (Lagrange's Theorem)**.** *If H is a subgroup of G (denoted by $H \leq G$), then $|H|$ divides $|G|$.*

One word proof: Cosets. □

2.1 *Homomorphisms, kernels, normal subgroups etc.*

Definition 2.2 (Homomorphisms)**.** *A homomorphism is a map $\phi : G \to H$ between two groups that behave well with the group multiplication in the sense that*

- $\phi(\mathrm{id}_G) = \mathrm{id}_H$,
- $\phi(g_1 g_2) = \phi(g_1)\phi(g_2)$, *and*
- $\phi(g^{-1}) = (\phi(g))^{-1}$. ◊

Once we have a homomorphism we can look at the *kernel* of the map ϕ defined by

$$\ker(\phi) = \{g \in G \,:\, \phi(g) = \mathrm{id}_H\}\,.$$

Proposition 2.3. *Let $\phi : G \to H$ be a homomorphism between groups. Then,* $\ker(\phi) = \{g \in G \,:\, \phi(g) = \mathrm{id}_H\}$ *is a subgroup of G. Furthermore, it satisfies the property that*

$$\forall g \in G, k \in \ker(\phi) \quad , \quad gkg^{-1} \in \ker(\phi).$$

This idea of gkg^{-1} staying inside the subgroup is very useful property that we shall refer to such groups as *normal subgroups*.

Definition 2.4 (Normal subgroups)**.** *A subgroup H is said to be a* normal subgroup *of G (denoted by $H \trianglelefteq G$) if for every $h \in H$ and $g \in G$ we have that*

$$ghg^{-1} \in H.$$ ◊

Hence, Proposition 2.3 states that $\ker(\phi) \trianglelefteq G$. Normal subgroups are extremely useful and you can do operations on them that you cannot do on other subgroups.

Proposition 2.5. *Let $H \trianglelefteq G$. Consider the set $GH = \{gh \,:\, g \in G, h \in H\}$. Then, GH is a group.*

One line proof: $gh = h'g$ for some $h' \in H$. □

Proposition 2.6. *Let $H \trianglelefteq G$. Then, the set of cosets of H in G forms a group — this is called the* quotient group *G/H. Furthermore, there is a natural homomorphism $\phi : G \to G/H$ for which* $\ker(\phi) = H$.

There are many more properties that can be stated about normal subgroups but we'll cross those bridges when we come to it.

2.2 *Group Actions*

But often, it is better to look at groups via its *action* on a set. Throughout this course, we'll only look at finite groups that act on a finite set.

Definition 2.7 (Group actions)**.** *A group G is said to* act *on a set Ω if every $g \in G$ induces a permutation of Ω (i.e. each g "moves" elements of Ω around) that satisfies the following properties.*

- id *corresponds to the identity permutation. That is, $\alpha^{\mathrm{id}} = \alpha$ for all $\alpha \in \Omega$,*
- *The actions compose in the natural way using the group multiplication — $(\alpha^g)^h = \alpha^{hg}$. That is, moving α by g and then by h is the same as moving α by hg.* ◊

I prefer actions on the left, hence the twist in order.

A succinct way of saying this is that a group action is a homomorphism from G to $\mathrm{Sym}(\Omega)$ but the definition above spells out what this means. However, the action could be redundant in the sense that there could be say two different elements of G that induce the same permutation on Ω by their action. We shall say that the action is *faithful* if the action of distinct elements of G is distinct. Or in other words, the homomorphism from G to $\mathrm{Sym}(\Omega)$ is injective.

Observation 2.8. *If G acts* faithfully *on a set Ω, then G is isomorphic to a subgroup of* $\mathrm{Sym}(\Omega)$.

Often in this course, G would actually be a subgroup of permutations on n elements so there is an immediate action of G on a set of n elements. But sometimes the actions may not be immediate from the definition of the group. One specific example that is very relevant in this context is the action of a group of n-permutations $G \leq S_n$ on $\Omega = \binom{[n]}{2}$ defined by

$$(i,j)^\sigma = (\sigma(i), \sigma(j))$$

which is just *lifting* the action on vertices to edges.

This part was not done in the lecture but might as well keep it here.

If G is a group of size n, then then there is a very natural standard action of G on itself just defined by left-multiplication — $g^h = hg$. It is easy to check that this is indeed a group action (called the *left regular group action*) by verifying all the properties. Thus, one way to look at the elements of G is how it permutes the elements of G upon left multiplication. This is formalized in the following observation.

Observation 2.9. *Any group G with $|G| = n$ is a subgroup of S_n, the symmetric group on n elements.*

A more interesting(?) action of G on itself is what is called the *conjugation action* defined by $g^h = hgh^{-1}$. We'll see more on these at a later point.

There is a lot that we can understand about a group from its action. We'll need a few definitions.

2.3 *Orbits and stabilizers*

Definition 2.10 (Orbits)**.** *If G acts on Ω, the* orbit of a point $\alpha \in \Omega$ *with respect to the action of G (denoted by α^G) is the set of all points that it can be moved to. Formally,*

$$\alpha^G = \{\beta \in \Omega \,:\, \beta = \alpha^g \text{ for some } g \in G\}\,. \qquad \text{◊}$$

We shall say that the action of G is transitive *if $\alpha^G = \Omega$. (It doesn't matter which α we choose.)*

Definition 2.11 (Stabilizers)**.** *Suppose G acts on Ω and $\alpha \in \Omega$. Then, the* stabilizer of α*, denoted by* $\mathrm{stab}_G(\alpha)$ *is defined as*

$$\mathrm{stab}_G(\alpha) = \{g \in G \,:\, \alpha^g = \alpha\}. \qquad \Diamond$$

It is pretty clear that $\mathrm{stab}_G(\alpha)$ is a subgroup of G but we can actually say more.

Theorem 2.12 (Orbit Stabilizer Theorem)**.** $|G| = \left|\alpha^G\right| \cdot |\mathrm{stab}_G(\alpha)|$*. Thus, in particular, both $\left|\alpha^G\right|$ and $|\mathrm{stab}_G(\alpha)|$ divide $|G|$.*

This is quite a powerful theorem and is used to prove some cool results about the structure of groups. Here is one application.

Theorem 2.13 (Cauchy's Theorem)**.** *Let G be a group whose size is divisible by a prime p. Then, G contains as a subgroup C_p which is a cyclic group of order p.*

Proof. Consider the following set of p-tuples of elements of G:

$$\Omega = \left\{(g_1, \ldots, g_p) \,:\, g_i \in G \text{ for all } i \text{ and } g_1 \cdots g_p = \mathrm{id}\right\}.$$

Observe that $|\Omega| = |G|^{p-1}$, which in particular is divisible by p.

One natural group that acts on Ω is the group of cyclic shifts. That is, if $C_p = \langle\sigma\rangle$ where σ is the p-cycle then define

$$(g_1, \ldots, g_p)^\sigma = (g_2, \ldots, g_p, g_1).$$

Now take any tuple $(g_1, \ldots, g_p) \in \Omega$. Since C_p acts on Ω, the size of the orbit of $(g_1, \ldots, g_p)$ must divide $\left|C_p\right|$ and hence must either be 1 or p. What are tuples whose orbit is just of size 1? Any tuple of the form $(g, g, \ldots, g)$ would be one such element but for this to be in Ω we must have $g^p = 1$. We do know one such element, which is $(\mathrm{id}, \ldots, \mathrm{id})$. Can this be the only such element?

Notice that the set Ω is naturally partitioned into orbits. If $(\mathrm{id}, \ldots, \mathrm{id})$ is the only element of orbit size 1, then the $|\Omega| = 1 \bmod p$, which contradicts our earlier observation that $|\Omega| = |G|^{p-1} = 0 \bmod p$. Hence, we must indeed have at least p elements in G such that $g^p = \mathrm{id}$. Any of these gs generate a cyclic group of order p. □

In fact, these sort of group actions can be used to prove other cool results, and one of the popular example are Sylow's Theorems. We won't need it now so we'll prove them at a later time. We shall just state the theorems for now.

Theorem 2.14 (Sylow Theorems)**.** *Let G be a finite group of size $p^r m$ where p is a prime, and $(p, m) = 1$. Then the following holds:*

1. *G contains a subgroup P of order p^r (these are called p-Sylow subgroups).*
2. *All p-Sylow subgroups of G are conjugates of each other. That is, for any other $Q \leq G$ of size p^r, there is some $g \in G$ such that $gPg^{-1} = Q$.*
3. *If s is the number of p-Sylow subgroups of G, then $s \mid m$ and $s \equiv 1 \bmod p$.*

3 Studying huge groups

Much of this part of the course would be dealing with groups acting on n elements, like symmetries of some graph on n vertices etc. A lot of times, these could be arbitrary subgroups of S_n which has $n!$ elements in it. Therefore, we would have to deal with subgroups $G \leq S_n$ that has exponentially many elements in it. How is G to be provided to us?

Fortunately, we can succinctly specify large groups by providing a generating set.

Definition 3.1 (Generating set)**.** *A set $S \subseteq G$ is said to be a* generating set *for G is every element of G can be expressed as a product of elements in S (possibly using elements of S multiple times). We shall denote this by $G = \langle S \rangle$.* ◊

Lemma 3.2. *Any finite group G has a generating set S of size $O(\log |G|)$.*

Proof. Let G be a finite group. If G is trivial, then $\varnothing \subset G$ generates G, and we are done. Suppose G is non-trivial, and pick a non-identity element $x_0 \in G$. The group $G_0 = \langle x_0 \rangle$ generated by x_0 satisfies $G_0 \leq G$ and $\langle x_0 \rangle \geq 2$. If $G_0 = G$, then the singleton set $S_0 := \{x_0\} \subset G$ does the job. Else, there is $x_1 \in G \setminus G_0$ (and automatically, $x_1 \in G$ is non-identity). Notice that the subgroup $G_1 = \langle S_1 \rangle$ where $S_1 := S_0 \cup \{x_0\}$, satisfies $G_0 < G_1 \leq G$ (strictly contains G_0 as a subgroup), and hence, by Lagrange's theorem (Theorem 2.1), satisfies $|G_1| \geq 2|G_0|$. We exploit this observation to find a generating subset for G, of order $O(\log |G|)$.

Set $S_{-1} = G_{-1} := \varnothing \subset G$. Having found a subset $S_{k-1} \subset G$ and the subgroup $G_{k-1} \leq G$ that it generates, we let $S_k := S_{k-1} \cup \{x\}$ for some $x \in G \setminus G_{k-1}$. The process terminates because G is finite, and the resulting subset is indeed a generating set because of the way it is constructed. On the other hand, if $G = G_k$, then

$$|G| = |G_k| \geq 2|G_{k-1}| \geq 2^2|G_{k-2}| \geq \cdots \geq 2^{k+1}|G_{-1}| = 2^{k+1}$$

which shows that $|S_k| = k + 1 \leq \log |G|$. This proves the *lemma* since $S_k \subset G$ is a generating set. □

With this, we can start asking a whole lot of questions given a succinct representation of a group. First, we shall connect these questions to the graph isomorphism and graph automorphism questions.

3.1 *Algorithmic tasks given a succinct group*

Lecture 2: January 20th, 2017

Membership: Given a group $G \leq S_n$ as $G = \langle S \rangle$ and a permutation $\sigma \in S_n$, check if $\sigma \in G$.

Size of the group: Given a group $G \leq S_n$ as $G = \langle S \rangle$, compute $|G|$.

Subgroup: Given two groups $G, H \leq S_n$, presented as $G = \langle S \rangle$ and $H = \langle T \rangle$, check if $H \leq G$.

Normal subgroup: Given two groups $G, H \leq S_n$, presented as $G = \langle S \rangle$ and $H = \langle T \rangle$, check if $H \trianglelefteq G$.

Group Intersection: Given two groups $G, H \leq S_n$, presented as $G = \langle S \rangle$ and $H = \langle T \rangle$, computing a generating set of $G \cap H$.

Orbit computation: Given a group $G \leq S_n$, presented as $G = \langle S \rangle$ acting on $\Omega = [n]$, compute the orbits of the group action.

Stabilizer computation: Given a group $G \leq S_n$, presented as $G = \langle S \rangle$, acting on $\Omega = [n]$ and a point $\alpha \in \Omega$, compute a generating set of $\mathrm{stab}_G(\alpha)$.

Normalizer computation: Given a group $G \leq S_n$, presented as $G = \langle S \rangle$, find the maximum sized subgroup $N = \langle T \rangle$ of S_n such that $G \trianglelefteq N$.

Computing automorphisms of a graph: Given a graph G, compute a generating set for $\mathrm{Aut}(G)$.

... and many more.

For each of the above problems, we shall either show that there are polynomial time algorithms or prove some sort of hardness result. Througout this section, unless otherwise stated, when we say "Given input $G = \langle S \rangle$", we shall mean that the input is the set $\{S\}$. Furthermore, we will always mean that $G \leq S_n$ naturally acts on $\Omega = \{1, \ldots, n\}$.

3.2 *Computing orbits and the orbit graph*

We are given an input $G = \langle S \rangle$ and an $\alpha \in \Omega$ and we want to compute its orbit

$$\alpha^G = \{\beta \,:\, \beta = \alpha^g \text{ for some } g \in G\}.$$

The following natural algorithm works.

Algorithm 1: Orbit

Input : $\langle S \rangle$ and $\alpha \in \Omega$
Output: The orbit of α

```
1 Δ = {α}
2 while Δ grows do
3     for β ∈ Δ and g ∈ S do
4         if β^g ∉ Δ then
5             Δ = Δ ∪ {β^g}
6 return Δ
```

Correctness of the algorithm is evident. The algorithm runs while Δ grows, and hence, there can be at most $1 + |\alpha^G| \leq 1 + n$ iterations of step 2, the additional one being to verify that Δ has stopped growing. In each iteration of step 2, the algorithm executes step 2 $\leq (|\alpha^G| \cdot |S|) \leq n \cdot |S|$ times, and for each of these executions, carries out membership-check $\beta^G \notin \Delta$ which takes $\leq |\alpha^G| \leq n$ times. Therefore, the running time of the algorithm is $O((n+1) \cdot n \cdot |S| \cdot n) = O(n^3|S|)$.

We can augment this algorithm to not just find the orbit, but find a representative $g_{\alpha,\beta} \in G$ such that $\alpha^{g_{\alpha,\beta}} = \beta$ for every α, β from the same orbit.

Algorithm 2: OrbitRepresentatives

Input : $G = \langle S \rangle$ and an $\alpha \in \Omega$
Output: A set of orbit representatives of α

1 $X = (\Omega, \varnothing)$, the empty graph with vertices Ω
2 **for** $\alpha \in \Omega$ *and* $g \in S$ **do**
3 **if** (α, α^g) *is not a directed edge in* X **then**
4 Add the directed edge (α, α^g) to X with label g.
5 Compute T, the connected component containing α.
6 Let $R = \{\mathrm{id}\}$
7 **for** *every* $\beta \in T$ **do**
8 Let $g_{\alpha,\beta}$ be the (left-)product of edge labels of a path from α to β in X so that $g_{\alpha,\beta}$ maps α to β.
9 Add $g_{\alpha,\beta}$ to R.
10 **return** R

The strongly connected components of the orbit-graph X, constructed in the above algorithm, are orbits induced by G. Furthermore, if α, β are in the same orbit the above algorithm finds an element $g \in G$ such that $\alpha^g = \beta$ by taking a path from α to β and multiply the edge labels along the way. We make a convention here, that our choice element $g \in G$ satisfying $\alpha^g = \alpha$ will always be the identity element of G.

Again this algorithm runs in $\mathrm{poly}(n, |S|)$ time. Indeed, step 2 involves at most $(|\alpha^G| \cdot |S|) \leq n|S|$ execution of step 3, and for each execution of step 3, the algorithm carries out at most $|\alpha^G|^2 \leq n^2$ edge-membership checks. Therefore, computing the orbit graph takes $O(n^3|S|)$ time. Once we have the orbit graph, computing connected components, shortest paths etc. are clearly polynomial time.

3.3 *Computing stabilizers*

Suppose that we are given as input a group $G = \langle S \rangle$ and a point $\alpha \in \Omega$. We wish to compute $G_\alpha = \mathrm{stab}_G(\alpha) = \{g \in G \,:\, \alpha^g = \alpha\}$.

Note that by Lagrange's theorem, $\mathrm{stab}_G(\alpha)$ tiles G exactly. Also, by Theorem 2.12, the number of tiles is exactly the orbit size $|\alpha^G|$. From this, what can we say about the *cosets* of $\mathrm{stab}_G(\alpha)$? These are sets of the form $g \cdot G_\alpha$. If $\alpha^g = \beta$, then every element of $g \cdot G_\alpha$ moves α to β. Hence, the cosets of G_α exactly corresponds to fixing a $\beta \in \alpha^G$ and collecting the set of all $g \in G$ that maps α to β.

This (mis)leads us to consider the following "natural algorithm" for computing a generating set for G_α.

Algorithm 3: Not-StabiliserGenSet

Input : $G = \langle S \rangle$ and $\alpha \in \Omega$
Output: A generating set for $G_\alpha = \mathrm{stab}_G(\alpha)$

1 Using Algorithm 2, compute α^G and a set of distinct representatives $\{g_\beta : \beta \in \alpha^G\}$ such that $\alpha^{g_\beta} = \beta$ for every $\beta \in \alpha^G$.
2 Let $T = \varnothing$
3 **for** *each* $g \in S$ **do**
4 **if** $\alpha^g = \alpha$ **then**
5 Add g to T
6 **else**
7 Let $\beta = \alpha^g$.
8 Add $g_\beta^{-1} \cdot g$ to T.
9 **return** T

Notice that the "generating set" $T \subset G$ returned by our "natural algorithm" satisfies $|S| = |T|$. Let us try proving correctness of this "algorithm". For this, we make the following observation: that the subset $S' := T \cup \{g_\beta : \beta \in \alpha^G\}$ generates G, as this generates everything in the original generating set S. Now, in order to prove "correctness", we just need to show the following

Not-a-Lemma. *The set T that Algorithm 3 outputs is indeed a generating set for $G_\alpha = \mathrm{stab}_G(\alpha)$.*

And here is how a possible proof might be sketched out.

(Incorrect) "Proof." Firstly note that all the elements we add to T in line 8 are indeed in T. So certainly $H = \langle T \rangle \leq G_\alpha$. To show that $H = G_\alpha$, observe that $T \cup \{g_\beta : \beta \in \alpha^G\}$ generate G as they clearly generate everything in S. But then, this can at most give $|\alpha^G|$ distinct cosets of $\langle T \rangle$. The only way this can cover all of G is if $|H| \cdot |\alpha^G| = |G|$ which forces $H = \mathrm{stab}_G(\alpha)$. ?QED?

This sentence isn't true...

Unfortunately, as the title indicates, Not-a-lemma is indeed not a lemma. We first consider an example that will show that $T \subset \mathrm{stab}_\alpha(G)$ is not a generating set of $\mathrm{stab}_\alpha(G)$.

We didn't do this counter-example in class but thanks to Somnath for this!

Example 3.3. *Consider the symmetric group S_{10} of permutations of the set $\{0, 1, \cdots, 9\}$, and let the cyclic permutations $\sigma_0, \sigma_3, \sigma_6 \in S_{10}$, be given as follows: $\sigma_0 = (0\ 1\ 2)$, $\sigma_3 = (3\ 4\ 5)$, $\sigma_6 = (6\ 7\ 8)$. The subset $T' := \{\sigma_0, \sigma_3, \sigma_6\} \subset S_{10}$ generates a subgroup of S_{10}. Let us write $H' = \langle T' \rangle$. Since $\sigma_i \sigma_j = \sigma_j \sigma_i$ for all $i, j \in \{0, 3, 6\}$, the subgroup $H' \leq S_{10}$ is commutative, and the non-identity elements of H' can have order 3 only; in particular, H' is not cyclic since there is no element of order $27 = |H'|$. Now consider any 2-elements subset $T_2 := \{h_1, h_2\} \subset H'$. The subgroup $\langle T_2 \rangle \leq H'$ has order ≤ 9. Therefore, H' can not have a generating set of less than three elements. Notice that the elements in H' are all even permutations, and therefore, $H' \leq A_{10}$, where $A_{10} \leq S_{10}$ is the alternating group.*

Recall that S_{10} has a generating set of size 2. Thus, if it was known that $H' \leq S_{10}$ is the stablizer of some element under some group-action, then we could have directly contradicted the statement of Not-a-lemma. And this is easy to come-up with. Let

$m := [S_{10} : H']$ be the number of cosets of $H' \leq S_{10}$, and let $\Omega = \{\mathrm{id}, x_1, \cdots, x_{m-1}\} \subset S_{10}$ be a complete set of coset-representatives. Explicitly, $m = (10)!/27$. Define action of S_{10} on Ω as follows: for $g \in G$ and $x \in \Omega$, set $x^g = x_i$ if and only if $gx \in x_i H$. Finally, note that $H = \mathrm{stab}_{S_{10}}(\mathrm{id})$.

Note that the action we have just defined does not really "look like" the natural action of S_n on $\{1, \cdots, n\}$. However, that is only superficial. As the definition of group action suggested, this action induces a homomorphism $S_{10} \to S_m$ whose kernel is the subgroup $K := \cap_{g \in S_{10}} g H' g^{-1}$. Notice that $K \leq S_{10}$ is a normal subgroup of S_{10}, contained in $H' \leq A_{10}$, and has order ≤ 27; therefore, $K \trianglelefteq A_{10}$ is a proper normal subgroup. Since A_{10} is simple, this shows that $K \leq H'$ is trivial. Thus, the action we defined is just induced by the natural action of S_m on the set $\{1, \cdots, m\}$, up to relabelling of the elements of Ω. ◊

What went wrong with the proof then? Well, an element $g \in G$, as we have observed, can be written as $g = g_1 \cdots g_t$ with all $g_i \in S' = T \cup \left\{g_\beta \,:\, \beta \in \alpha^G\right\}$. Let us suppose that $g \notin \langle T \rangle$; then there is some $g_k \in \left\{g_\beta \,:\, \beta \in \alpha^G\right\}$ appearing in the above expression such that $g_{k+1}, \cdots, g_t \in T$. For our line of argument in the above "proof" to work, we need a g_β, $\beta \in \alpha^G$ such that $g_1 \cdots g_k \in g_\beta H$, or equivalently, $g_\beta^{-1}(g_1 \cdots g_{k-1}) g_k \in H$. However, we are now in hopeless situation, since the "algorithm" did not assure us of even having $g_{k-1} g_k \in g_\beta H$ for some $\beta \in \alpha^G$.

Lecture 3: January 23rd, 2017

Heaven is not lost! Although our algorithm could not return what it was expected to, we have a clue as to what could be a way around. Notice that we have derived the following in our preceding discussion: in order for $T' \subset \mathrm{stab}_G(\alpha)$ to be a generating set, we needed to have $g_\beta^{-1}(g_1 \cdots g_{k-1}) g_k \in H$ for some $\beta \in \alpha^G$, whenever $g_k \in \left\{g_\beta \,:\, \beta \in \alpha^G\right\}$ and $g_i \in T \cup \left\{g_\beta \,:\, \beta \in \alpha^G\right\}$. We formulate our revamped strategy along this lines.

Lemma 3.4. *Let $G = \langle S \rangle$ be acting on $\Omega = \{1, \cdots, n\}$ and $R = \{x_1, \cdots, x_m\} \subset G$ a complete set of orbit-representatives and assume $\mathrm{id} \in R$. For any $g \in G$ and $x_i \in R$, there exists a unique $x_j \in R$ such that $x_j^{-1} g x_i \in \mathrm{stab}_G(1)$.*

This restriction of $\mathrm{id} \in R$ can be removed. See Lemma 3.6 for a more general statement.

Proof. In fact, this lemma is almost a tautology. For any $g \in G$ and $x_i \in R$, if $1^{x_i g} = j$ then there is a unique $x \in R$ such that $1^{x_j} = j$; this is because $R = \{x_1, \cdots, x_m\} \subset G$ a complete set of orbit-representatives. But this implies $x_j^{-1} g x_i \in \mathrm{stab}_G(1)$. □

Let us consider $T := \{x_j^{-1} g x_i : g \in S,\ x_i, x_j \in R,\ x_j^{-1} g x_i \in \mathrm{stab}_G(1)\}$. We have the following

Lemma 3.5. *Every $g \in G$ can be written as $g = x_i t$ for a unique $x_i \in R$ and unique $t \in \langle T \rangle$.*

Proof. If $g = \mathrm{id}$, there is nothing to prove. Else, write $g = g_k g_{k-1} \cdots g_1$ with $g_i \in S$ for all $1 \leq i \leq k$. Recall that our convention was to have $\mathrm{id} \in R$. Assume that $k = 1$ in the expression $g = g_k g_{k-1} \cdots g_1$, so that $g \in S$. Since $\mathrm{id} \in R$, the above lemma tells us that there is a unique $x \in R$ such that $x^{-1} g \in \mathrm{stab}_G(1)$. By definition of T, we have $x^{-1} g \in T$; since $g = x(x^{-1} g)$, we are done.

We now proceed via induction on k. Assume the induction hypothesis; that is, for every string $g_k g_{k-1} \cdots g_1 \in G$, where $g_i \in S$ for all $1 \le i \le k$, there is a $x_k \in R$ and $t \in \langle T \rangle$ such that

$$\begin{aligned} & g_k g_{k-1} \cdots g_1 = x_k \, t \\ \implies \quad & g = g_{k+1} g_k g_{k-1} \cdots g_1 = g_{k+1} \, x_k \, t. \end{aligned}$$

By the Lemma 3.4, there is a unique $x_{k+1} \in R$ such that $x_{k+1}^{-1} g_{k+1} x_k \in \mathrm{stab}_G(1)$, and by definition of T, again $t_k' := x_{k+1}^{-1} g_{k+1} x_k \in T$. But this implies $g = x_{k+1}(x_{k+1}^{-1} g_{k+1} x_k) t_k = x_{k+1} t_k' t$

$$\begin{aligned} g &= g_{k+1} \, x_k t \\ &= x_{k+1} \, \left(x_{k+1}^{-1} \, g_{k+1} \, x_k \right) t \\ &= x_{k+1} \, t_k' t \\ &\in x_{k+1} \, \langle T \rangle . \end{aligned}$$

This proves the lemma modulo the uniqueness part.

Now to the uniqueness. Notice that $T \subset \mathrm{stab}_G(1)$, and hence, $\langle T \rangle \le \mathrm{stab}_G(1)$. Supose that $x_i t_i = x_j t_j$ for some $t_i, t_j \in \langle T \rangle$ and $x_i, x_j \in R$; then we have $x_i^{-1} x_j = t_i t_j^{-1} \in \langle T \rangle \subset \mathrm{stab}_G(1)$, which shows that $1^{x_i} = 1^{x_j}$. This implies $x_i = x_j$ since R contains a unique representative for each $l \in 1^G$. But then $t_i = t_j$. □

We now have an algorithm to find the generating set of $\mathrm{stab}_G(\alpha)$, for $\alpha \in \Omega$.

Algorithm 4: StabiliserGenSet

Input : $G = \langle S \rangle$ and $\alpha \in \Omega$
Output: A generating set for $G_\alpha = \mathrm{stab}_G(\alpha)$

1 Using Algorithm 2, compute the orbit $\alpha^G \subset \Omega$ and a complete set of distinct coset-representatives $R := \left\{ g_\beta \, : \, \beta \in \alpha^G \right\}$, such that $\alpha^{g_\beta} = \beta$ for every $\beta \in \alpha^G$, and $g_\alpha = \mathrm{id}$.
2 Let $T = \varnothing$
3 **for** *each* $g \in S$ *and* $g_\beta, g_\gamma \in R$, **do**
4 **if** $g_\beta^{-1} \, g \, g_\gamma$ *fixes* α **then**
5 Add $g_\beta^{-1} \, g \, g_\gamma$ to T
6 **return** T

Invoking Algorithm 2 costs $O(n^3|S|)$ time. Step 3 involves $\le n^2|S|$ runs, and each run involves checking if the orbit graph contains the edge (γ, β) which involves at most $\le n^2$ time. Therefore, the running time of this algorithm is polynomial in n and $|S|$. To show the correctness of the algorithm, notice that $T \subset \mathrm{stab}_G(\alpha)$, and hence, $\langle T \rangle \le \mathrm{stab}_G(\alpha)$. Now suppose $g \in G$ is any element; by Lemma 3.5, there is a unique $g_\beta \in R$ such that $g \in g_\beta t$ for some $t \in \langle T \rangle$. That is, there can be at most $\left|\alpha^G\right|$ distinct cosets of $\langle T \rangle$

The usefulness of Lemma 3.5 merits a reformulation in greater generality, originally due to Schreier. Here it goes.

Lemma 3.6 (Schreier's lemma)**.** *Let G be a group with a generating set $S \subset G$, and let $H \leq G$ a subgroup. Let $R = \{x_1, \cdots, x_k\} \subset G$ be a complete set of distinct coset-representatives of $H \leq G$, where $x_1 = \mathrm{id}$. Let $T := \{x_i^{-1} g x_j \in H : g \in S,\ x_i, x_j \in R\}$; then $H = \langle T \rangle$.*

We didn't do the proof in the lecture but I'll leave it here.

Proof. For any $g \in S$ and $x \in R$, observe that gx belongs to some coset of H and hence $gx \in x_i H$ for some $x_i \in R$ which implies that $x_i^{-1}\, gx \in H$, and hence in T by definition.

$$g\ x = x_i \cdot \left(x_i^{-1}\ g\ x\right)$$

Since this is true for every $g \in S$ and $x \in R$, this can be compactly written as

$$S \cdot R \subseteq R \cdot T$$

Now if we look at $g_1, g_2 \in S$ and $x \in R$, we once again get by a similar reasoning that $g_1 g_2 x = g_1 \cdot (x_i h) = x_j h$. That is,

$$\begin{aligned} S \cdot S \cdot R &\subseteq S \cdot R \cdot T \\ &\subseteq R \cdot T \cdot T \\ &\vdots \\ \implies \langle S \rangle R &\subseteq R \cdot \langle T \rangle . \end{aligned}$$

But the left hand side of the last equation is just G, and the right hand side is always contained in G. Hence it must be the case that $G = R \cdot H$. Hence, $\langle T \rangle = H$. □

As was the case with $\mathrm{stab}_G(\alpha) \leq G$, Lemma 3.6 prescribes polynomial time algorithm to find the generating set of any subgroup $H \leq G$ that is recognizable, in the sense that there is a polynomial time algorithm to determine if a given element $g \in G$ belongs to H. Notice that the Lemma 3.6 on its own does not assume finiteness of G; as long as $H \leq G$ has finitely many cosets, the recipe for the generating set works. However, in that case the recognizability of H is lost.

3.4 *Membership testing*

We now come to the central algorithm which is testing membership. Once again, we are given $G \leq S_n$ by a generating set $\{S\}$ and a $\sigma \in S_n$, and we wish to check if $\sigma \in G$. The cool thing is that this can be checked in polynomial time.

How do we go about this? Here is one sanity check we can make. Let $\Omega = \{1, \ldots, n\}$. Certainly if $\sigma = \mathrm{id}$, then we just accept. Otherwise, pick an arbitrary $\alpha \in \Omega$ that is moved by σ and let $\beta = \sigma(\alpha)$. Suppose it turns out that $\beta \notin \alpha^G$ (recall we can compute α^G using Algorithm 1), then we can immediately infer that $\sigma \notin G$. Say $\beta \in \alpha^G$, and suppose $g \in G$ is an element such that $\alpha^g = \beta$ (which we can compute using Algorithm 2). Let us look at $g^{-1}\sigma$. Clearly, $\sigma \in G$ if and only

if $g^{-1}\sigma \in G$. What have we gained? The key point is that $g^{-1}\sigma$ does not move α, i.e $g^{-1}\sigma(\alpha) = \alpha$. Hence, we have the following simple observation.

Observation 3.7. *If $\sigma(\alpha) = \beta$ and if $\alpha^g = \beta$ for some $g \in G$; then, $\sigma \in G$* if and only if *$g^{-1}\sigma \in \mathrm{stab}_G(\alpha)$.*

By Algorithm 4, we can construct a generating set for $\mathrm{stab}_G(\alpha)$ in polynomial time. We just recursively keep applying the above observation. This results in the following algorithm for membership testing.

Algorithm 5: MembershipTest(E)

Input : $G = \langle S \rangle$ and $\sigma \in S_n$
Output: Yes if $\sigma \in G$, and No otherwise.

1 **if** $\sigma = \mathrm{id}$ **then**
2 **return** Yes
3 Since $\sigma \neq \mathrm{id}$, let $\alpha \in \Omega$ such that $\sigma(\alpha) = \beta \neq \alpha$.
4 **if** $\beta \notin \alpha^G$ **then**
5 **return** No
6 Since $\beta \in \alpha^G$, using Algorithm 2 compute an element $g \in G$ such that $\alpha^g = \beta$.
7 Using Algorithm 4, compute a generating set T for $\mathrm{stab}_G(\alpha)$.
8 **return** MembershipTest(E)$(T, g^{-1}\sigma)$

The correctness of the algorithm is clear. However, there is a catch. The size of the successive generating sets grows as $|T_{\mathrm{stab}_G(\alpha)}| \leq |S| \cdot |\alpha^G|$, without an apparent reasonable bound. On the other hand, it is apparent that having some bound of the form $|T| \in O(n^k)$, for example, will turn Algorithm 5 into a polynomial time algorithm. This is accomplished as follows.

Suppose that a group $G \leq S_n$ is given as $G = \langle S \rangle$. We intend to find $T \subset G$ such that $|T| \leq n^2$, and $G = \langle T \rangle$. The following algorithm builds a strictly upper-triangular $n \times n$ tableau M in which, each cell holds at most one element of G, and the cells on/below the diagonal are empty.

Algorithm 6: ReduceGenSet

Input : $G = \langle S \rangle$, where S is given as an array $S = \{g_1, \cdots, g_k\}$
Output: A generating set for G of size at most n^2

Let $M[i][j] = \varnothing$ for all integers $i, j = 1, \cdots, n$
Let $t = 1$
while $t \leq k$, **do**
    if $g_t = \mathrm{id}$ **then**
        $t \longleftarrow t + 1$
    else
        Let i be the smallest index such that $i^{g_t} \neq i$ and Let $j := i^g$.
        if $M[i][j] = \varnothing$ **then**
            insert g in $M[i][j]$
            $t \longleftarrow t + 1$
        else
            Replace g_t in S by $g' := (M[i][j])^{-1} g$.
return M

Notice that this is a polynomial time algorithm because each execution of the while loop may involve at most n repetitions each of which is polynomial-step. Moreover, in each stage, the set $M \cup \{g_{t+1}, \cdots, g_k\}$ forms a generating set for G. Therefore, the algorithms ends with a generating set, namely, the set of elements in the tableau M; the size of this set is $\leq \binom{n}{2} \leq n^2$.

Following modification of the preceding membership testing algorithm (Algorithm 5) now yields a polynomial time algorithm.

Algorithm 7: MembershipTest(S, σ)

Input : $G = \langle S \rangle$ and $\sigma \in S_n$
Output: Yes if $\sigma \in G$, and No otherwise.

if $\sigma = \mathrm{id}$ **then**
    return Yes
Since $\sigma \neq \mathrm{id}$, let $\alpha \in \Omega$ such that $\sigma(\alpha) = \beta \neq \alpha$.
if $\beta \notin \alpha^G$ **then**
    return No
Since $\beta \in \alpha^G$, using Algorithm 2 compute an element $g \in G$ such that $\alpha^g = \beta$.
Using Algorithm 4, compute a generating set T for $\mathrm{stab}_G(\alpha)$.
return MembershipTest(ReduceGenSet$(T), g^{-1}\sigma)$

Now that we have membership testing, a lot of other natural problems follow almost immediately. For example, given two groups H and G, we can test if

$H \leq G$.

Algorithm 8: SubgroupTest

Input : $G = \langle S \rangle$ and $H = \langle T \rangle$
Output: Yes if $H \leq G$, and No otherwise.

```
for each h ∈ T do
    if MembershipTest(S, h) = No then
        return No
return Yes
```

We can also check if $H \trianglelefteq G$.

Algorithm 9: NormalSubgroupTest

Input : $G = \langle S \rangle$ and $H = \langle T \rangle$
Output: Yes if $H \trianglelefteq G$, and No otherwise.

```
if SubgroupTest(S, T) = No then
    return No
for each g ∈ S and h ∈ T do
    if MembershipTest(T, ghg^{-1}) = No then
        return No
return Yes
```

3.5 *Computing the size of a group*

Given a group $G = \langle S \rangle$, we wish to compute $|G|$. The following natural attempt that uses the Orbit-Stabilizer theorem (Theorem 2.12) works.

Algorithm 10: GroupSize

Input : $G = \langle S \rangle$
Output: $|G|$

```
if S = ∅ then
    return 1
Let α ∈ Ω with |α^G| > 1
Compute r = |α^G| using Algorithm 1.
Compute a generating set T of size at most n^2 for stab_G(α) using
  Algorithm 4 and Algorithm 6.
Recursively compute M = GroupSize(T) = |stab_G(α)|.
return r · M
```

But if we were to unravel the recursion, we get a chain of subgroups of G that are successive stabilizers. Let $\Omega = \{1, \ldots, n\}$. For any group $G \leq S_n$, we shall define the following groups:

$$G^{(i)} = \{g \in G \ : \ j^g = j \text{ for all } j \leq i\}.$$

In other words, $G^{(1)} = \text{stab}_G(1)$ and $G^{(2)} = \text{stab}_{G^{(1)}}(2)$ which are all elements of g that fix both 1 and 2, and so on. Hence we have a sequence of group containments

$$G_0 = G \geq G^{(1)} \geq G^{(2)} \geq \cdots \geq G^{(n)} = \{\text{id}\}.$$

We shall be returning often to such a *tower* of subgroups leading from G to $\{\text{id}\}$.

3.6 *Normal closures and subnormality*

Lecture 4: January 26th, 2017

We will now see a natural extension of the notion of a normal subgroup, called *sub-normality*.

Definition 3.8 (Sub-normality)**.** *A subgroup H of G, is said to sub-normal in G (denoted by $H \lhd\lhd G$) if and only if $\exists$ groups $G_0, G_1, \ldots, G_k$ such that $H = G_k \lhd G_{k-1} \lhd \cdots \lhd G_1 \lhd G_0 = G$.* ◊

A natural algorithmic task that follows is to check if $H \lhd\lhd G$ for any given $H = \langle T \rangle$ and $G = \langle S \rangle$. The strategy will be to find a candidate for G_1 and then recurse. This seems doable as we already have an efficient algorithm for testing whether $A \trianglelefteq B$. Since the length of any such chain can be at most $\log_2 |G|$, a recursive solution will also be efficient. All that remains to be done is to get an apt candidate for G_1.

Definition 3.9 (Normal Closure)**.** *Given $H \leq G$, the normal closure of H in G (denoted by $\text{NormClosure}_G(H)$) is defined to be the smallest normal subgroup of G that contains H.* ◊

It is worth pointing out that phrase "the smallest" in the above definition is legit; there is a unique smallest normal subgroup of G that contains H. Indeed, if $H \leq G_1, G_2 \trianglelefteq G$ that contain H, then $H \leq G_1 \cap G_2 \trianglelefteq G$ which would yield a smaller normal subgroup if $G_1 \neq G_2$.

Lemma 3.10. $H \lhd\lhd G \Leftrightarrow H \lhd\lhd \text{NormClosure}_G(H)$

Proof. Let N be $\text{NormClosure}_G(H)$ for convenience.

- ($\Leftarrow$) Any chain $H \lhd \cdots \lhd N$ can be immediately extended to G as $N \trianglelefteq G$.
- ($\Rightarrow$) Let $H = G_k \lhd G_{k-1} \lhd \cdots \lhd G_1 \lhd G_0 = G$. Let $G'_i = G_i \cap N$. Therefore, $G'_0 = N$.
  Say $h \in G_i \cap N$ and $g \in G_{i-1} \cap N$. Then, $ghg^{-1} \in G_i$ since $G_i \lhd G_{i-1}$ and $ghg^{-1} \in N$ since $g, h \in N$. Hence $G'_i \lhd G'_{i-1} \Rightarrow H = G'_k \lhd G'_{k-1} \lhd \cdots \lhd G'_1 \lhd G'_0 \lhd G$. □

Now all we need is an algorithm to find a generator for $\text{NormClosure}_G(H)$, given H as $\langle T \rangle$ and G as $\langle S \rangle$. A good first attempt is to start with everything in T, and compute sts^{-1} for all $t \in T$ and $s \in S$. Such elements that are not yet in our collection should be added. Even if we don't get a generating set for $\text{NormClosure}_G(H)$, we should be getting closer. So just repeat this till the collection saturates. Interestingly enough, this works. So let us formalize this.

Algorithm 11: NormalClosure

Input : $G = \langle S \rangle, H = \langle T \rangle$
Output: $B : \langle B \rangle = \text{NormClosure}_G(H)$

```
set B = T
repeat
    set B ← B'
    for each b ∈ B, s ∈ S do
        if MembershipTest(B', ghg^{-1}) = No then
            B' ← B' ∪ {gbg^{-1}}
until B' = B
return B
```

Correctness: Every element that is added to B' in the algorithm are clearly elements that ought to be in $\text{NormClosure}_G(H)$. Furthermore, whenever the algorithm terminates, the group generated by B' is normal in G. Hence the algorithm indeed outputs a generating set for the smallest normal subgroup of G that contains H.

Running time: Every increase in size of B' makes the group generated by B' double in size. Hence, we have at most $\log_2(n!)$ iterations, each of which takes polynomial time.

Putting together all these ideas, we get the following algorithm for subnormality testing.

Algorithm 12: SubnormalityTest

Input : $G = \langle S \rangle, H = \langle T \rangle$
Output: Yes if $H \lhd\lhd G$, No otherwise

```
i ← 0
S_0 ← S
repeat
    i ← i + 1
    S_i ← NormalClosure(S_{i-1}, T)
until ⟨S_i⟩ = ⟨S_{i+1}⟩
if ⟨S_i⟩ = ⟨T⟩ then                    // Can be checked via Algorithm 7
    return Yes
else
    return No
```

Remark. *The above algorithm has the added bonus that if $H \lhd\lhd G$, then the sets $\{S_i\}$ computed in fact give generating sets for the subnormal chain between H and G.* ◊

3.7 *Commutators and solvability*

A group G is said to be *abelian* if $\forall a, b \in G$, we have $ab = ba$. As a natural extension, we want to comment about how *close* G is to being abelian. This notion goes via

something called a *commutator subgroup* of G.

Definition 3.11 (Commutator Subgroup)**.** *Let G be a group. The commutator subgroup of G, denoted by $[G,G]$, is given by:*
$[G,G] := \left\langle \left\{ xyx^{-1}y^{-1} \ : \ x,y \in G \right\} \right\rangle$ ◊

Here is an important fact that we won't prove in class.

Fact 3.12. *$[G,G] \trianglelefteq G$ and $\frac{G}{[G,G]}$ is abelian. Furthermore, $[G,G]$ is the smallest normal subgroup $H \trianglelefteq G$ such that $\frac{G}{H}$ is abelian. That is, if $H \trianglelefteq G$ and $\frac{G}{H}$ is abelian, then $[G,G] \leq H$.*

Task: Given $G = \langle S \rangle$, find a generating set for $[G,G]$.

A guess would be again to try $B := \{ghg^{-1}h^{-1} : g,h \in S\}$. This however may not always work, since we are not guaranteed that the element, say, $gghg^{-1}g^{-1}h^{-1}$ will be in $\langle B \rangle$. But, we know that $H := \langle B \rangle \leq [G,G] \trianglelefteq G$. Therefore our next try should be the normal closure of this H. Let us now verify that this will actually work. For that, it is enough to show that $\frac{G}{H'}$ is abelian, where $H' := \text{NormClosure}_G(H)$. This follows from Fact 3.12.

Lemma. *Given a group $G = \langle S \rangle$ and $H := \left\{ xyx^{-1}y^{-1} \ : \ x,y \in S \right\}$. If $H' := \text{NormClosure}_G(H)$, then $\frac{G}{H'}$ is abelian.*

Proof. The intuition is that we know H' contains $\left\{ xyx^{-1}y^{-1} \ : \ x,y \in S \right\}$ which in essence means that in the quotient group $\frac{G}{H'}$, we have the "relations" $xy = yx$ for all $x,y \in S$. Given the fact that every element of G is generated by S should allow us to permutate elements of $\frac{G}{H'}$ as well. The following proof formalizes this.

Claim. For every $a,b \in G$, there is some $h \in H'$ such that $ab = ba \cdot h$.

Clearly, proving the above claim immediately implies that $\frac{G}{H'}$ is abelian.

Pf. Firstly, if $a,b \in S$ then we already* have $a^{-1}b^{-1}ab \in H'$ hence $ab = ba \cdot a^{-1}b^{-1}ab$. If a,b are "strings" in S, we shall prove the above claim by induction on its length. Say $a = a_1 \cdots a_s$ and $b = b_1 \cdots b_t$, and say $t > 1$ (the other case is symmetric). By the inductive hypothesis

* *Wait, why? We only know that $aba^{-1}b^{-1} \in H'$... ah, because H' is normal.*

$$\begin{aligned} a_1 \cdots a_s b_1 \cdots b_t &= b_1 \cdots b_{t-1} a_1 \cdots a_s \cdot h \cdot b_t && (h \in H') \\ &= b_1 \cdots b_{t-1} a_1 \cdots a_s b_t (b_t^{-1} h b_t) \\ &= b_1 \cdots b_{t-1} a_1 \cdots a_s b_t \cdot h' && (h' \in H' \text{, normality}) \\ &= b_1 \cdots b_{t-1} b_t a_1 \cdots a_s \cdot h'' \cdot h && (\text{induction}) \end{aligned}$$

which is what we set out to prove. □

That finishes the proof of the claim and hence the lemma. □

With this we can write an algorithm to compute the commutator subgroup of G.

Algorithm 13: CommutatorSubgroup

Input : $G = \langle S \rangle$
Output: $T : [G, G] = \langle T \rangle$

```
1 set T ← {xyx^{-1}y^{-1} : x, y ∈ S}
2 set T' ← NormalClosure(S, T)
3 return T'
```

Now, just like the extension from normality to subnormality via towers of $\triangleleft$, we can study a tower that arises from repeated computation of commutator subgroups, (i.e.) $G \rhd [G, G] = G_1 \rhd [G_1, G_1] = G_2 \rhd \cdots \rhd G_k \rhd \cdots$. In the case when G is an abelian group, then $G_1 = [G, G] = \{\mathrm{id}_G\}$, and hence this "commutator chain" immediately terminates at $\{\mathrm{id}_G\}$. There may be other cases where it takes longer but eventually terminates at $\{\mathrm{id}_G\}$, and maybe other groups where it never terminates to identity.

Definition 3.13 (Solvable Groups)**.** *A group G is said to be* solvable *if there exists a chain of commutators terminating at $\{id_G\}$ in finitely many steps. That is,*

$$G = G_0 \rhd G_1 \rhd \cdots \rhd G_k = \{\mathrm{id}_G\}$$

for some k, where each $G_i = [G_{i-1}, G_{i-1}]$. ◊

An example of a non-solvable group is S_5 where $[S_5, S_5] = A_5$, the group of even permutations, and $[A_5, A_5] = A_5$, and the chain is stuck there. This fact turns out to be very important in results in complexity such as Barrington's theorem for width-5 branching programs, and also essential in Galois theory to show that degree 5 univariate polynomials are not "solvable by radicals"[2].

Since we have an algorithm to compute the commutator subgroups, we can also check if a given group is solvable or not.

Algorithm 14: SolvabilityTest

Input : $G = \langle S \rangle \leq S_n$
Output: Yes if G is solvable; No otherwise

```
1  i ← 0
2  S_0 ← S
3  repeat
4  |  i ← i + 1
5  |  S_i ← CommutatorSubgroup(S_{i-1})
6  until ⟨S_i⟩ = ⟨S_{i-1}⟩
7  if ⟨S_i⟩ = {id_G} then
8  |  return Yes
9  else
10 |  return No
```

[2] Degree 2 is "solvable by radicals" in the sense that we have the expression for roots as $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

4 Towers of subgroups

Lecture 5: January 30th, 2017

So far, we have been looking at algorithmic tasks given a generating set for the subgroup. But in often cases, we do not really have the generating set of the group we are interested in. Keep in mind the automorphisms of a graph; we do not really have a generating set for this group but what we instead know is to check if a given element is an automorphism or not. Often, existence of a *nice* tower of subgroups from a *known* group to our target group can be very useful. To understand this better, let us look at a few computational problems that would be sufficient to solve graph isomorphism.

4.1 *Set stabilizers, group intersection, and graph isomorphism*

We have already seen the notion of stabilizer of a point $\alpha \in \Omega$. The following is a generalization that applies to a subset $\Delta \subseteq \Omega$.

Definition 4.1 (Set Stabilizer)**.** *Given $G = \langle S \rangle \leq S_n$ and $\Delta \subseteq \{1, 2, \ldots, n\}$, the stabilizer of Δ in G denoted by* $\mathsf{SetStab}_G(\Delta) = \{g \in G : \Delta^g = \Delta, \bar{\Delta}^g = \bar{\Delta}\}$ ◊

Observation 4.2. $\mathsf{SetStab}_G(\Delta) \leq G$.

We now want to show that GraphAut (a problem to which GraphIso can be reduced, as mentioned in Lemma 1.6), where we want to find the generating set of $\mathrm{Aut}(X)$ given a graph $X = (V, E)$ is reducible to SetStab, where we want to find the generating set of $\mathsf{SetStab}_G(\Delta)$ given $G = \langle S \rangle \leq S_n$ and $\Delta \subseteq \{1, 2, \ldots n\}$. We formalize this as:

Lemma 4.3. $\mathsf{GraphAut} \leq_{\mathrm{TM}} \mathsf{SetStab}$.

The $\leq_{\mathrm{TM}}$ is a terminology used to say that one problem is Turing Machine reducible *to another. Or in simpler words, $A \leq_{\mathrm{TM}} B$ is saying that the problem B is at least as hard as problem A computationally.*

Proof. Suppose $X = (V, E)$ is a graph with $|V| = n$. Given an oracle which computes $\mathsf{SetStab}_G(\Delta)$ for every G and Δ, we want to compute $\mathsf{GraphAut}(X)$.

We take $G = S_n$ and let it act on $\Omega = \{(i, j) : i, j \leq n\} \equiv \{1, 2, \ldots n^2\}$ by lifting the natural action on $\{1, \ldots, n\}$ via $(i, j)^g = (i^g, j^g)$. Now clearly if $g \neq g'$, then the resulting action on the pairs is also different. This gives a way of *embedding* $G = S_n$ into S_{n^2} as every element $g \in G$ results in a permutation of the pairs in Ω. Just to spell it out exactly, if we were to think of an element $g \in G$ as an $n \times n$ permutation matrix M_g, the embedding of g in S_{n^2} is the $n^2 \times n^2$ permutation matrix $\tilde{M}_g$ defined by

$$\left(\tilde{M}_g\right)_{(i,j),(i',j')} = \begin{cases} 1 & i^g = i' \ , \ j^g = j' \\ 0 & \text{otherwise} \end{cases}$$

Hence, we can take a small generating set for $G = S_n$ (for example, the set of all $\binom{n}{2}$ transpositions), and embed each element in S_{n^2} as mentioned above and we have a small generating set for G as a subgroup of S_{n^2}.

Now, if we take Δ as the set of all edges in X then it is easy to see that $\mathsf{SetStab}_G(\Delta) = \mathsf{GraphAut}(X)$. Since this reduction can be done in polynomial time, if we had an oracle for SetStab, then we automatically get an algorithm to compute $\mathsf{GraphAut}(X)$. □

A problem very closely related to SetStab is the *group intersection* problem. The *group intersection* problem (which we'll denote by $\mathsf{GroupIntersect}(S, T)$) is where we are given $G = \langle S \rangle$ and $H = \langle T \rangle$ and we want ot find a generating set for $G \cap H$ efficiently. The following lemma shows that this problem is in fact equivalent to SetStab.

Lemma 4.4. $\mathsf{SetStab} \equiv_{\mathrm{TM}} \mathsf{GroupIntersect}$.

The notation $\equiv_{\mathrm{TM}}$ just means that both $\leq_{\mathrm{TM}}$ and $\geq_{\mathrm{TM}}$ holds. That is, if you can solve one, then you can solve the other as well.

Proof. First, we show that $\mathsf{SetStab} \leq_{TM} \mathsf{GroupIntersect}$. Suppose $G \leq S_n$ and $\Delta \subseteq \{1, 2, \dots n\}$. Given an oracle which computes $\mathsf{GroupIntersect}(H_1, H_2)$ for every $H_1, H_2 \leq S_n$, we want to compute $\mathsf{SetStab}_G(\Delta)$.

Take $H = \left\langle \{(i\ j) : i, j \in \Delta \text{ or } i, j \in \bar{\Delta}\} \right\rangle$. Note that the size of the generating set is small. Now clearly, H is the set of all permutations on $\{1, 2, \dots, n\}$ elements which permute elements of Δ within Δ and that of $\bar{\Delta}$ within $\bar{\Delta}$. Hence it is easy to see that $\mathsf{SetStab}_G(\Delta) = \mathsf{GroupIntersect}(G, H)$

Next, we show that $\mathsf{GroupIntersect} \leq_{TM} \mathsf{SetStab}$. Suppose $G, H \leq S_n$. Given an oracle which computes $\mathsf{SetStab}_{G'}(\Delta)$ for every G' and Δ, we want to compute $\mathsf{GroupIntersect}(G, H)$.

Consider $G' = G \times H$ which acts on $\Omega = \{(i, j) : i, j \leq n\}$ as $(i, j)^{(g,h)} = (i^g, j^h)$. We note that there is a small generating set for G' since it is generated by $\{(g, \mathrm{id}) : g \in S\} \cup \{(\mathrm{id}, h) : h \in T\}$ where $G = \langle S \rangle$ and $H = \langle T \rangle$. Then for $\Delta = \{(i, i) : i \leq n\}$, we see that $(g, h) \in \langle \mathsf{SetStab}_{G'}(\Delta) \rangle \Leftrightarrow \forall i, i^g = i^h \Leftrightarrow g = h$. Thus, $G \cap H = \langle \{g : (g, g) \in \mathsf{SetStab}_{G'}(\Delta)\} \rangle$.

Both the reductions described above are achievable in polynomial time and hence we have the lemma. □

4.2 *Descending towers*

Using Lemma 4.3 and Lemma 4.4 we have $\mathsf{GraphAut} \leq_{TM} \mathsf{GroupIntersect}$. But in many instances, knowing some additional properties about the groups would help us solve them efficiently. Much of this amounts to existence of some *nice* towers and we formalize that notion below.

Lemma 4.5 (Descending nice towers)**.** *Given $G = \langle S \rangle \leq S_n$, if there exist $G_1, \dots, G_k$ such that $G = G_0 \geq G_1 \geq G_2 \geq \cdots \geq G_k = H$ for some target subgroup H and if:*

- $[G_{i-1} : G_i] := \frac{|G_{i-1}|}{|G_i|} = \mathrm{poly}(n, |S|)$ *for all $i \in [k]$.*
- *For any $i \in [k]$, given a $g \in G_{i-1}$, there is an efficient algorithm to check if $g \in G_i$.*

then we can efficiently compute T such that $H = \langle T \rangle$.[3]

However, in order to prove Lemma 4.5, it is enough to solve it for $k = 1$, since we can then use it repeatedly. Hence, we show:

Lemma 4.6. *Let $G = \langle S \rangle$ and $H \leq G$ is a subgroup with $[G : H] = r = \mathrm{poly}(n)$ that is* recognizable *i.e there is an oracle which can test membership in H. Then, a generating set for H can be computed efficiently.*

[3] In more generality, if $[G_{i-1} : G_i] \leq r$, it can be computed in time polynomial in $n, |S|$ and r.

Proof. We note that it is enough to find distinct coset representatives for $H \leq G$, say $R = \{a_1, a_2, \ldots a_r\}$ (i.e. $G = \biguplus_{i=1}^{r} a_i H$), as then Schrier's Lemma (Lemma 3.6) says that $\{a_i^{-1} g a_j : a_i, a_j \in R, g \in S\} \cap H$ generates H. So, we look at an algorithm to find the distinct coset representatives for H.

Algorithm 15: FindCosetReps

Input : $G = \langle S \rangle \leq S_n$
Output: The coset representatives of H in G

```
1  CR ← {id}
2  repeat
3      for each g ∈ S, a_i ∈ CR do
4          add g a_i to CR
           // Check if g a_i was actually giving a new coset
5          for each a_j ∈ CR do
6              if a_j^{-1} g a_i ∈ H then          // via membership test in H
7                  remove g a_i from CR
8                  break out of for loop
9  until CR does not grow
10 return CR
```

First, we note that G acts on $\Omega = \{a_1 H, a_2 H, \ldots a_r H\}$ by left multiplication. Further, the action is transitive, that is, given any $a_i H$, for every $a_j H \in \Omega$, $\exists g = a_j a_i^{-1} \in G$ such that g acts on $a_i H$ to give $a_j H$.

Since $[G : H] = r$, the *orbit graph* of G action on Ω has r vertices and is connected as G acts transitively. Therefore, the above process *discovers* all the vertices of the orbit graph in $\text{poly}(|S|, n, r)$ time. □

4.3 *Revisiting the group intersection problem*

So now that we have proved Lemma 4.5, we revisit GroupIntersect problem in some special cases.

Definition 4.7. *Let $G, H \leq S_n$. G is said to* normalize *H if for every $g \in G$ and $h \in H$, $ghg^{-1} \in H$.* ◊

For example, if $H \trianglelefteq G$, then certainly G normalizes H just by the definition of normality. But the above definition includes cases when $H \trianglelefteq K$ and G is some subgroup of K (which may not contain H entirely).

Lemma 4.8. *Given $G = \langle S \rangle$, $H = \langle T \rangle$ such that G normalizes H, then problem* GroupIntersect(G, H) *can be solved efficiently.*

Proof. In order to use Lemma 4.5, we want to find a tower beginning in G and ending in $G \cap H$ which satisfies the conditions of the lemma. Consider the most natural tower beginning at G, namely the stabilizer chain:

$$G \geq G^{(1)} \geq G^{(2)} \geq \ldots \geq G^{(n)} = \{\text{id}\}$$

where $G^{(i)}$ consists of the permutations in G that do not move any of $1, \ldots, i$. From this, we get the tower

$$GH \geq G^{(1)}H \geq G^{(2)}H \geq \ldots \geq G^{(n)}H = H.$$

Note that these products $G^{(i)}H$ makes sense as $G^{(i)}H$ is indeed a group *because* G (and hence $G^{(i)}$) normalizes H. From this, we can also obtain the tower $G = G \cap GH \geq G \cap G^{(1)}H \geq G \cap G^{(2)}H \geq \ldots \geq G \cap G^{(n)}H = G \cap H$.

Clearly, thanks to Lemma 4.5, it is enough to show:

1. $[G^{(i)}H \cap G : G^{(i+1)}H \cap G]$ is small.
2. For every i and every $g \in G^{(i-1)}H \cap G$ we should be able to check $g \in G^{(i)}H \cap G$ efficiently.

We first note that the membership test can be done efficiently. Indeed, for every $g \in G^{(i-1)}H \cap G$, we already know that $g \in G$ and so it is enough to check $g \in G^{(i)}H$ efficiently. Now this can be done using Algorithm 7 since we have a small generating set for $G^{(i)}H$, namely $S^i \cup T$ where S^i a generating set of $G^{(i)}$ (which we know how to obtain via successive stabilizer applications using Algorithm 4)

To check that $[G^{(i)}H \cap G : G^{(i+1)}H \cap G]$ is small, since $[G^{(i)} : G^{(i+1)}]$ is small and $G^{(i)}$ normalizes H for every i, the following observations are enough:

1. $H, K \leq G$ are such that G normalizes $H \Rightarrow [GH : KH] \leq [G : K]$
2. $H, K \leq G \Rightarrow [G \cap H : K \cap H] \leq [G : K]$

The proofs of these two observations are left as an exercise*.

* *Would be a part of Problem Set 1 that would be coming up shortly.*

We already know that $[G^{(i)} : G^{(i+1)}] \leq n$ (why?) and hence the above two observations show that $[G^{(i)}H \cap G : G^{(i+1)}H \cap G] \leq n$ as well. Thus, we indeed have a *nice* tower and Lemma 4.5 lets us get a generating set for $G \cap H$. □

Next, we claim that if G, H are groups such that $H \lhd\lhd \langle G, H \rangle$, then we can compute $\mathsf{GroupIntersect}(G, H)$ efficiently. We formalize this as:

Lemma 4.9. *Given $G = \langle S \rangle$, $H = \langle T \rangle$ such that $H \lhd\lhd \langle S \cup T \rangle$, $\mathsf{GroupIntersect}(G, H)$ can be computed efficiently.*

Proof. We would have liked to use Lemma 4.8, but since we are not given any similar conditions and since $H \lhd\lhd \langle S \cup T \rangle$, we start with the most natural tower available,

Note that $\langle S \cup T \rangle = \langle G, H \rangle$

$$\langle G, H \rangle = G_0 \rhd G_1 \rhd \ldots \rhd G_k = H$$

from which, by intersecting with G throughout, we get the tower

$$G = \langle G, H \rangle \cap G = G_0 \cap G \rhd G_1 \cap G \rhd \ldots \rhd G_k \cap G = H \cap G.$$

Note that the elements in the second tower also have a normal subgroup relation and this follows directly from the normality relation among the corresponding elements in the first tower.

Now we note that $G_i \cap G$ normalizes G_{i+1}. Indeed, for $g \in G_i \cap G$, $h \in G_{i+1}$, we have $g^{-1}hg \in G_{i+1}$ due to the normal subgroup relations in the first tower. Thus, by Lemma 4.8, if we have a generating set for $G_i \cap G$, we have a generating set for $G_{i+1} \cap G$. Since we have a generating set for $G = G_0 \cap G$, we can descend this chain and find a generating set for $G_k \cap G = H \cap G$, thus proving the lemma. □

Lecture 6: February 3rd, 2017

In this lecture, equipped with Descending Nice Towers (see Lemma 4.5), we proceed to tackle automorphism problems:

4.4 *Solving an Automorphism Problem*

The Coloured Graph Automorphism problem is a natural generalization of the general automorphism problem. In this problem, we are given a graph X and a colouring of the vertices and we want to output a generating set for the group of automorphisms of X that respect the colours of the vertices. The idea of mapping vertices of a degree only to vertices of the same degree is nicely encapsulated here by colouring all vertices of a degree by a single colour. Although this colouring would be respected by any automorphism, abstracting such properties to colours will be useful as we will see later in the lecture. Let's first formally define coloured graph automorphism:

Definition 4.10 (Coloured Graph Automorphisms)**.** *For a graph $X = (V, E)$ on n vertices and a colouring of its vertices $\phi : V \to C$, where $C = \{1, 2, \ldots, n\}$, the* coloured automorphisms of X, *denoted by* $\mathrm{Aut}_\phi(X)$, *is the following set of permutations:*

$$\mathrm{Aut}_\phi(X) = \{\sigma \in \mathrm{Sym}(V) \ : \ (u,v) \in E \iff (\sigma(u), \sigma(v)) \in E \text{ and } \phi(v) = \phi(\sigma(v))\}. \diamondsuit$$

Task: Given X, ϕ, find a generating set for $\mathrm{Aut}_\phi(X)$.

If every vertex is coloured 1, this is just the automorphism problem. Also note that no matter what the colouring is, one can attach certain large gadgets (a unique gadget for each colour) to each vertex and remove the colours, thus reducing coloured graph automorphism to graph automorphism on a polynomially larger graph. Despite the equivalence of these two problems, we will now solve coloured graph automorphism for a certain class of colourings:

Bounded Colour Multiplicity Graph Automorphism: Given a graph $X = (V, E)$ on n vertices and a colouring of its vertices $\phi : V \to C$ such that $\forall i \in C \ \left|\phi^{-1}(i)\right| \leq b$, find a generating set for $\mathrm{Aut}_\phi(X)$.

Theorem 4.11. *Bounded Colour Multiplicity Graph Automorphism can be solved in poly$(n, 2^{b^2})$ time.*

Proof. We solve this problem using a quite intuitive descending nice tower. Notational warning: we call the elements of the tower $G^0, G^1, G^2 \ldots G^k$, but we will sometimes add subscripts to these groups to better illustrate how G^{t+1} is obtained from G^t. Before defining the tower, we introduce some more notation: Let $\Omega = V$ be the set that our groups act on. We split Ω into disjoint subsets

based on the colours of the vertices:

$$\Omega = \bigcup_{i \in C}^{\cdot} \Omega_i \text{ where } \Omega_i = \Phi^{-1}(i)$$

Similarly, we split E into $O(n^2)$ sets based on the colours of the endpoints of the edges:

$$E = \bigcup_{i \leq j \in C}^{\cdot} E_{i,j} \text{ where } E_{i,j} = E \cap \Omega_i \times \Omega_j$$

As the first element of our tower, we take $G^0 = \mathrm{Sym}(\Omega_1) \times \mathrm{Sym}(\Omega_2) \times \cdots \times \mathrm{Sym}(\Omega_n)$, the set of colour respecting permutations. Clearly $\mathrm{Aut}_\phi(X)$ is a subgroup of G^0 and we can construct a generating set for G_0.

We incrementally approach $\mathrm{Aut}_\phi(X)$ by ensuring that the permutations map (i,j) edges to edges (the permutations already respect colour, so they are mapped to (i,j) edges and then (i,j) nonedges would have to map to (i,j) nonedges) until we end up with only those colour respecting permutations that map edges to edges and nonedges to nonedges: $\mathrm{Aut}_\phi(X)$. With this in mind, we take elements from the set $C \times C$ in any order, and we build a tower of length $O(n^2)$, where G^{t+1}_{i-j} is defined as follows:

$$G^{t+1}_{i-j} = \left\{\sigma \in G^t \ : \ (u,v) \in E_{i,j} \iff (\sigma(u), \sigma(v)) \in E_{i,j}\right\}$$

It is easy to see that G^{t+1}_{i-j} is a subgroup of G^t and to see that the last element of the group is $\mathrm{Aut}_\phi(X)$. We claim that the tower so defined is a descending nice tower. For this we need to show the following:

- G^{t+1}_{i-j} is recognizable in G^t. This is easy to see, since we only need to check that (i,j) edges map to edges.
- $[G^t : G^{t+1}_{i-j}]$ is small. We will show that it is in fact $\leq 2^{b^2}$.

To bound the number of cosets, consider $g, g' \in G^t$. If $g^{-1}g' \in G^{t+1}$, then $g' = gh$ for some $h \in G^{t+1}$ and so g and g' lie in the same coset. Now we note that if $g(E_{i,j}) = g'(E_{i,j}) = E'$, then g' maps an element of $E_{i,j}$ to an element of E' and g^{-1} would map it back to $E_{i,j}$. Hence $g^{-1}g' \in G^{t+1}$ and so g and g' are in the same coset. By looking at the image of the action of g on $E_{i,j}$, we can now divide G^t into equivalence classes such that any two elements in the same equivalence class lie in the same coset. Since each equivalence class corresponds to an E' and $E' \subseteq \Omega_i \times \Omega_j$, there are atmost 2^{b^2} equivalence classes and hence atmost 2^{b^2} cosets.

Now that we have shown that our tower is a descending nice tower we can use Lemma 4.5 to solve Bounded Colour Multiplicity Graph Automorphism in $\mathrm{poly}(n, 2^{b^2})$ time. □

This result has some implications for graph automorphism. For example, if we had a graph in which there are atmost a constant number of vertices of each degree, we can find the automorphisms of that graph. However, a more

promising problem to be solving is bounded degree graph automorphism, which bounded colour multiplicity graph automorphism doesn't seem to be helping in.

5 Divide and Conquer techniques.

We will now introduce colour stabilizers and blocks, which we plan to use later on in a divide and conquer algorithm aimed at tackling bounded degree graph isomorphism.

Definition 5.1 (Colour Stabilizers)**.** *Suppose G acts on a coloured set Ω. Then, the* colour stabilizer of Ω, *denoted by* $\text{colourStab}_\Omega(G)$ *is defined as*

$$\text{colourStab}_\Omega(G) = \{g \in G \,:\, a^g \sim a \ \forall a \in \Omega\}$$

where $a \sim b$ means that a and b are coloured with the same colour. ◊

5.1 *Intransitive case*

Note that if you are given a graph $X = (V, E)$ and you define Ω as $V \times V$ coloured with two colours, red if $\alpha \in E$ and black otherwise, then it is clear that $\text{colourStab}_\Omega(\text{Sym}(V))$ is just the set-stabilizer of the edges, which we know is $\text{Aut}(X)$. While we do not know how to find colourStab, we will see that it might possible for this computation to be broken down into subproblems:

If G does not act transitively on Ω, let us split Ω into the connected components of its orbit graph: $\Omega = \biguplus_{i=1}^{r} \Omega_i$. We can then see that

$$\text{colourStab}_\Omega(G) = \text{colourStab}_{\Omega_1}(\text{colourStab}_{\Omega_2}(\cdots \text{colourStab}_{\Omega_r}(G)\cdots))$$

We can compute this by first computing the innermost $\text{colourStab}_{\Omega_r}(G)$ to get G', then trying to solve $\text{colourStab}_{\Omega_{r-1}}(G')$, which might again be split into subproblems. In the worst case, Ω is eventually split into singletons, so we will end up solving at most $|\Omega|$ instances of colourStab. Of course, we do not know how to solve colourStab yet, but this dreaming may prove useful.

5.2 *Blocks*

As it turns out, even if G acts transitively on Ω, it might still have some useful structure to exploit:

Definition 5.2 (Blocks)**.** $\Delta \subseteq \Omega$ *is a block if for all $g \in G$, either $\Delta^g = \Delta$ or $\Delta^g \cap \Delta = \varnothing$.* ◊

Trivial examples of blocks are singletons or the whole set Ω. A more informative example is the automorphism group of a complete binary tree, in which any subtree is a block. No automorphism can map a subtree so that its image is partially within the subtree and partially outside.

Claim 5.3. *Let G be a group with a transitive action on Ω. If $\Delta \subseteq \Omega$ is a block, then $|\Delta|$ divides $|\Omega|$.*

Proof. Let $\alpha \in \Delta$. Since G acts transitively on Ω, take elements $g_1, g_2, \ldots, g_{|\Omega|}$ such that $\Omega = \left\{\alpha^{g_1}, \alpha^{g_2}, \ldots, \alpha^{g_{|\Omega|}}\right\}$. Δ^{g_i} and Δ^{g_j} are both of size $|\Delta|$ since actions are permutations. Furthermore Δ^{g_i} and Δ^{g_j} are either the same or disjoint (otherwise $\Delta^{g_i g_j^{-1}}$ would not be Δ but would have a non-zero intersection with it). Since these cover Ω, it follows that they tile Ω and hence the claim follows. □

Definition 5.4 (Primitive Action)**.** *A transitive action of G on Ω is said to be primitive iff there are no nontrivial blocks.* ◊

Just like in the intransitive case, whenever the action contains non-trivial blocks, we want to *reduce* the problem somehow. But first, we'll look at a few computations tasks related to blocks.

Finding non-trivial blocks

Lecture 7: February 6th, 2017

Since we want to find such blocks when we're stuck with a transitive action, we'd want to solve the following computational task:

Task: Given $G = \langle S \rangle, \alpha \in \Omega$, find the smallest non-singleton block containing α.

We begin with our problem of finding the smallest non-singular block containing $\alpha \in \Omega$, given $G = \langle S \rangle$ and $\alpha \in \Omega$. We can solve this problem if we can find the smallest block containing both α and β for any $\beta \in \Omega$ (solving this problem for all $\beta \in \Omega$ and then choosing the one that gives smallest block).

We will assume that the group G acts on Ω transitively. We make the following observation.

Observation 5.5. *If $\alpha^g = \beta$ and $\beta^g = \gamma$, then any block containg α, β must contain γ.*

Proof. Let Δ be arbitrary block that contains α, β. Then we have $\beta \in \Delta^g \cap \Delta$ (as $\alpha^g = \beta$). By the definition of block, this forces $\Delta^g = \Delta$, which implies that $\beta^g = \gamma \in \Delta^g = \Delta$. □

The above observation gives us following algorithm to compute the smallest block containing both α and β. Construct a graph $X = (\Omega, E)$ with $E = \{(\gamma, \delta) : \exists g \in G \text{ such that } (\gamma, \delta) = (\alpha, \beta)^g\}$.

Claim 5.6. *The connected components of X form the block system generated by the minimum block Δ containing α and β.*

Proof. Although, this might look intuitively rather plain, let us be diligent and work out a few details to establish the veracity of our claim. Say Δ is the smallest block containing α, β. Let us say that $\Gamma = \{\Delta_1, \ldots, \Delta_m\}$ is the block system generated by Δ.

1. If γ is in the connected component of X containing α, then it must* belong to Δ. Hence the connected component of X containing α is contained in the smallest block containing α, β.

   * *If $\gamma \in \Delta \cap \Delta^g$ if $\alpha^g = \beta$.*

2. Now, we need to show that the connected compnent Y containing α indeed a block. Consider any $g \in G$ and observe that since Y is connected, so is Y^g.*

So, if $Y \cap Y^g \neq \varnothing$, then $Y = Y^g$ or else the fact that Y is a maximal connected sub-graph of X would be contradicted. **Why? Think about edges in Y.*

□

We would now have an algorithm if only we could somehow find out all the edges in E. All we are given is a generating set S of G. How do we find out all pairs $(\gamma, \delta) = (\alpha, \beta)^g$? This is precisely the orbit of (α, β) in the action of G on $\Omega \times \Omega$. Now we have a complete algorithm. *Convince yourself.*

Algorithm 16: MinimumBlockOfAPair

Input : $G = \langle S \rangle \leq S_n$ acting transitively, and $\alpha, \beta \in \Omega$
Output: The smallest block Δ containing α, β.

1 Consider the action of G on $\Omega \times \Omega$ via $(\gamma, \delta)^g = (\gamma^g, \delta^g)$. Using Algorithm 1 compute the orbit E of (α, β) in this action.
2 Construct the graph $X = (\Omega, E)$ where E is the set of all pairs in the above orbit computation.
3 **return** *connected component Δ of X that contains α, β.*

Algorithm 17: MinimumBlock

Input : $G = \langle S \rangle \leq S_n$ acting transitively, and $\alpha \in \Omega$
Output: The smallest non-singleton block Δ containing α

1 **for** $\beta \in \Omega \setminus \{\alpha\}$ **do**
2 $\quad \Delta_\beta :=$ MinimumBlockOfAPair(S, α, β)
3 Let Δ be the smallest of the Δ_βs.
4 **return** Δ.

5.3 *Block systems and structure forests*

Finding non-trivial blocks, informally, allow us to work with smaller group as we can think of each block as one element and group action between the blocks. We characterize this intution be defining Block System.

Definition 5.7 (Block System)**.** *Given a G that acts transitively on Ω, and a block $\Delta \subseteq \Omega$, the* block system generated by Δ *is a collection $\Gamma = \{\Delta_1, \ldots, \Delta_m\}$ that partition Ω, that is*

$$\Omega = \dot{\bigcup_{i=1}^{m}} \Delta_i,$$

where each $\Delta_i = \Delta^g$ for some $g \in G$. ◊

Say we found the smallest block Δ and suppose this Δ generates the block system $\Gamma = \{\Delta_1, \ldots, \Delta_m\}$. It is possible that a union of a few of these Δ_is form a large block (keep in mind the automorphisms of a complete binary tree). The question now can be can we find the smallest subset of Γ such that the union of those blocks form a large block? Indeed we can. All we need to do is think of G acting on Γ instead of Ω and use Algorithm 17 again. This allows us to build an

entire hierarchy of blocks starting with the singletons and going all the way to Ω. This is called the *structure forest* of G.

Algorithm 18: StructureForest

Input : $G = \langle S \rangle$ acting transitively on $\Omega = \{1, \ldots, n\}$
Output: Structure forest of G

$\ell = 0$ and $\Gamma_0 = \{\{1\}, \ldots, \{n\}\}$.
repeat
  Choose α arbitrarily from Γ_ℓ.
  Look at the action of G on Γ_ℓ and let $\Gamma_{\ell+1}$ be the block system generated by MinimumBlock$(\langle S \rangle, \Gamma_\ell, \alpha)$ (via Algorithm 17)
  $\ell = \ell + 1$
until $|\Gamma_\ell| = 1$
return $\{\Gamma_0, \Gamma_1, \ldots, \Gamma_\ell\}$

Block kernels

Say we have a block system Γ, intuitively we want to look at the action of G on the block system Γ instead of Ω, which is in some sense making progress as G now acts on a set of size m instead of n (and note $m = n/|\Delta|$, and hence smaller than n if Δ is a non-trivial block). Thiis what we did in Algorithm 18 but in other instances the issue would be that even though G acts faithfully on Ω, that is different elements of g induce different partitions on Ω, this may not be the case with G's action on Γ. There may be elements of g such that $\Delta_i^g = \Delta_i$ for all $\Delta_i \in \Gamma$ even though g shuffles elements inside each Δ_i. Hence, in order to make G's action on Γ faithful, we need to *quotient* these elements out. This leads us to naturally define the following notion of a *Block Kernel*.

Keep in mind the automorphisms of a complete binary tree, and elements that just permute sibling leaves.

Definition 5.8 (Block Kernel)**.** *Given G, Ω and a Block System $\Gamma = \{\Delta_1, \Delta_2, .., \Delta_m\}$, the* Block Kernel of Γ *denoted by* $\text{BlockKernel}_G(\Gamma)$ *is defined as the set of all elements in G that doesn't move any of the Δ_i's, i.e., $\{g \in G : {\Delta_i}^g = \Delta_i$ for all $i \in [m]\}$.* ◊

We now prove the following theorem.

Theorem 5.9. $\text{BlockKernel}_G(\Gamma) \trianglelefteq G$.

Proof. Let $h \in \text{BlockKernel}_G(\Gamma)$. If g^{-1} moves the block Δ_i to block Δ_j then g moves the block Δ_j to block Δ_i. Therefore, ghg^{-1} doesn't move any block. Hence, $ghg^{-1} \in \text{BlockKernel}_G(\Gamma)$. □

Great! Since $\text{BlockKernel}_G(\Gamma)$ is a normal subgroup of G, its quotient is a honest-to-God group and this quotient acts on Γ in a faithful way. We now have an obvious computational task.

Task: Given a group $G = \langle S \rangle$ and a block system Γ, compute a generating set for $\text{BlockKernel}_G(\Omega)$.

Lemma 5.10. *Given $G = \langle S \rangle$ and a block system Γ, a generating set for the block kernel can be computed in polynomial time.*

Proof. We will do this by showing the existence of a 'nice tower' $G = G_0 \geq G_1 \geq \ldots \geq G_m = H$ such that $[G_{i-1} : G_i]$ is small and given $g \in G_{i-1}$, we can efficiently check if $g \in G_i$.

Let $G_i = \left\{ g \in G \ : \ \Delta_j^g = \Delta_j \quad \forall j \leq i \right\}$. Note that G_i is $\mathsf{SetStab}_{G_{i-1}}(\Delta_i)$. We have seen that computing set stabilizer is hard . But here we know Δ_i's are block and we will show this leads to polynomial time algorithm. Since Δ_i's are block so only m distinct images of Δ_i is possible and thus $[G_{i-1} : G_i] \leq m \leq n$. Also given $g \in G_{i-1}$, it is easy to check if $g \in G_i$. This implies polynomial time algorithm for computing G_m i.e., $\text{BlockKernel}_G(\Gamma)$ (via Lemma 4.5). □

Blocks and subgroups

We now try and understand when groups actions would have non-trivial blocks. The following theorem gives an exact characterization of this.

Theorem 5.11. *Let G acts on Ω transitively. Then G's action is primitive if and only if $\text{stab}_G(\alpha)$ is a maximal strict subgroup of G.*

Proof. ($\Leftarrow$): Suppose G's action is not primitive. Therefore, there exists a non-trivial block containing α. Let Δ be the smallest non-trivial block containing α. Then we have $\text{stab}_G(\alpha) \leq \text{stab}_G(\Delta) \leq G$. Our goal is to show both of the inclusion is strict. Since Δ is a non-trivial block so it must contain some β ($\beta \neq \alpha$). Let $\alpha^g = \beta$. Since Δ is a block so $g \in \text{stab}_G(\Delta)$. But $g \notin \text{stab}_G(\alpha)$ (as it moves α). Thus, we have $\text{stab}_G(\alpha) < \text{stab}_G(\Delta)$.

For the second inclusion, we observe that G acts on Ω transitively but $\text{stab}_G(\Delta)$ doesn't (as it is a non-trivial block). This proves $\text{stab}_G(\alpha) < \text{stab}_G(\Delta) < G$.

($\Rightarrow$): Now assume there exists a H s.t. $\text{stab}_G(\alpha) < H < G$. We will show that $\Delta = \alpha^H$ (orbit of α on the action of H) is a non-trivial block. It suffices to show that Δ is indeed a block, and $1 < |\Delta| < |\Omega|$.

It immediately follows from $\text{stab}_G(\alpha) < H$ that $|\Delta| > 1$. From the Orbit Stabilizer theorem (Theorem 2.12), we have $|\Delta| \cdot |\text{stab}_G(\alpha)| = |H|$ and $|\Omega||\text{stab}_G(\alpha)| = |G|$. Also we have $|\text{stab}_G(\alpha)| = |\text{stab}_H(\alpha)|$ and $|H| < |G|$ (as $\text{stab}_G(\alpha) < H < G$). Thus, $1 < |\Delta| < |\Omega|$.

The fact that Δ is indeed a block is left as an easy exercise. □

With this lemma, we can hope to find out the structure of groups whose actions are primitive.

5.4 *Sylow Theorems and p-groups*

One instance of the above lemma that we'll use shortly for bounded degree graph isomorphism is the case when $|G|$ is a power of a prime p. Such groups are called p groups and there is a lot of theory about structure of p-groups. We'll take an aside for now and look at the Sylow Theorems. We won't be needing all of them in the course but they are good to know in any case.

Theorem 5.12 (Sylow's Theorem)**.** *Let $|G| = p^r m$, such that $\gcd(m, p) = 1$. Then*

1. *(Existence) There is a subgroup $P \leq G$ of size p^r (p-sylow subgroup).*

2. *(Relations) If P and Q are two p-sylow subgroups of G, then $\exists g \in G$, such that $gPg^{-1} = Q$.*
3. *(Count) If n_p is the number of p-sylow subgroups of G, then $n_p \equiv 1 \bmod p$ and $n_p | m$.*

Proof. (1) Let Ω be the set of all subsets of size p^r. Therefore, $|\Omega| = \binom{p^r m}{p^r} \not\equiv 0 \bmod p$. Let G act on Ω via left multiplication, i.e., if $S \in \Omega$ then $S^g = \{g \cdot \alpha | \alpha \in S\}$. Let $\Omega_1, \Omega_2, .., \Omega_t$ be the orbits of Ω under the action of G. Therefore, $\Omega = \Omega_1 \cup .. \cup \Omega_t$.

One of the sets $S \in \Omega$ is our p-Sylow subgroup. What would be the orbit of S? These must be precisely the cosets and hence we hope to find one Ω_i of size exactly m. Now how do we do that?

Since $|\Omega| \neq 0 \bmod p$, therefore there exists $i \in [t]$ such that $|\Omega_i| \not\equiv 0 \bmod p$. Also note that $|\Omega_j| \geq m$ for all j since

$$\bigcup_{S \in \Omega_j} S = G.$$

as for any $a, b \in G$, there is a $g \in G$ such that $ga = b$. This forces $\cup_{S \in \Omega_j} S$ to cover all of G and that is possible only if there are at least m elements in Ω as $|S| = p^r$.

Let S be any set in Ω_i and P be $\text{stab}_G(S)$. Claim is that $|P| = p^r$. To see this we apply the orbit stabilizer theorem and get $|P||\Omega_i| = p^r m$. Since $|\Omega_i| \geq m$ and $|\Omega_i| \not\equiv 0 \bmod p$, we get $|P| = p^r$.

(2) Let P be a p-sylow subgroup of G. Let Ω be the m left-cosets of P. Let $\Omega = \{a_1 P, a_2 P, .., a_m P\}$. Suppose Q is another p-sylow subgroup of G. Let Q act on Ω via left multiplication, as $(a_i P)^q = q \cdot a_i \cdot P$. Let $\Omega_1, \Omega_2, .., \Omega_t$ be the orbits of Ω under the action of Q. Therefore, we have $\sum_i |\Omega_i| = m \not\equiv 0 \bmod p$. By the orbit stabilizer theorem, each $|\Omega_i|$ must divide p^r. This implies that atleast one of Ω_i (say Ω_1) must have size equal to 1. This means $\Omega_1 = \{a_1 P\}$ and $qa_1 P = a_1 P \quad \forall q \in Q$. This implies $a_1^{-1} q a_1 \in P$ for all $q \in Q$. So we get $a_1^{-1} Q a_1 = P$.

(3) Let Q be a p-sylow subgroup of G. Consider the set of p-Sylow subgroups of G,

We didn't do this in class, but useful to know.

$$\Omega = \left\{P_1, \ldots, P_{n_p}\right\}.$$

Let us look at the action of G on Ω via $P_i^q = q^{-1} P_i q$. We know from (2) that the action is transitive. Hence in order to find $|\Omega|$. Hence we have that $n_p \mid |G|$. If we could show that $n_p \equiv 1 \bmod p$, then it would also follow that $n_p \bmod m$.

Instead of G acting on Ω, we shall consider the action of a p-Sylow subgroup P via conjucation. But we have a smaller group P acting on Ω, we could end up with a partition of Ω as orbits $\Omega = \Omega_1 \cup \cdots \cup \Omega_k$. By the Orbit-Stabilizer Theorem (Theorem 2.12), we know that $|\Omega_i|$ divides $|Q| = p^r$. We know that $Q \in \Omega$, and is clear that its orbit is just $\{Q\}$. The goal would be to show every other orbit has size at least p and this would show that $|\Omega| = n_p \equiv 1 \bmod p$.

As a bonus, we also know that $\text{stab}_G(P) = \text{Normalizer}_G(P) = N_P$ and hence $n_p = [G : N_P]$.

Take a $P \neq Q$. We wish to show that the orbit of P cannot be $\{P\}$. What is the $\text{stab}_Q(P)$? This precisely

$$\text{stab}_Q(P) = \left\{q \in Q \ : \ q^{-1} P q = P\right\} =: \text{Normalizer}_Q(P).$$

Again, by the Orbit-Stabilizer Theorem (Theorem 2.12), it suffices to show that $\text{Normalizer}_Q(P)$ is a proper subgroup of Q. Observe that $\text{Normalizer}_Q(P) = \text{Normalizer}_G(P) \cap Q$. So the only way $\text{Normalizer}_Q(P) = Q$ is if it happens that $Q \subseteq \text{Normalizer}_Q(P)$. But then Q and P are both p-Sylow subgroups sitting inside $\text{Normalizer}_G(P)$. Applying (2) to this means that $Q = g^{-1}Pg$ for some $g \in \text{Normalizer}_G(P)$, which is absurd as the very definition of the normalizer forces $g^{-1}Pg = P$. Hence $\text{stab}_Q(P) < Q$ and hence the orbit of P must have size bigger than 1. This forces every other orbit to have size that is a multiple of p. Therefore, $n_p \equiv 1 \bmod p$. □

Lecture 8: February 10th, 2017 and Lecture 9: February 13th, 2017

Remember that we were trying to study the structure of groups that act primitively on some set Ω. Suppose, $|G| = p^r$ and G acts faithfully, transitively and primitively, then what can we say about $|\Omega|$ and G? Using Theorem 5.11 we conclude* that $|\text{stab}_G(\alpha)| = p^{r-1}$, which forces hence $|\Omega| = p$. However, as $G \leq S_{|\Omega|}$ since G acts faithfully, we must have $r = 1$. Hence, $G = C_p$, the cyclic group of order p. We'll record this as an observation.

* *Why? Clearly, you have not gone through Problem Set 1.*

Observation 5.13. *If G is a p-group acting faithfully, transitively and primitively on a set Ω, then G must be a cyclic group the order p and $|\Omega| = p$.*

5.5 *Divide and conquer via blocks*

Let us return to the transitive action of G on Ω, where $\Gamma = \{\Delta_1, \ldots, \Delta_m\}$ is a block system and $H = \text{BlockKernel}_G(\Gamma)$. Let $\{a_1H, \ldots, a_rH\}$ be the set of cosets of H in G. Now, let us generalize the concept of $\text{colourStab}_\Omega(G)$, to replace G with any arbitrary set K as, $\text{colourStab}_\Omega(K) \triangleq \{g \in K \mid a^g \sim a \ \forall a \in \Omega\}$. The following is then immediate:

$$\text{colourStab}_\Omega(G) = \bigcup_{i=1}^{r} \text{colourStab}_\Omega(a_iH).$$

However, we must ask whether it is even sensible to efficiently "get hold" of $\text{colourStab}_\Omega(K)$, for an arbitrary K. After all, we have always been dealing with large sets that were groups and hence one could hope for getting hold of a small sized generating set. Fortunately, things don't go astray if K happens to be a coset.

Lemma 5.14. *If* $\text{colourStab}_\Omega(aH)$ *is non-empty, then it is a coset of* $\text{colourStab}_\Omega(H)$.

Proof. Consider τ and σ in $\text{colourStab}_\Omega(aH)$. Then, it is plain to see that $\tau^{-1}\sigma$ stabilizes colors, and as both τ and σ are members of aH, $\tau^{-1}\sigma \in H$. □

Hence, we can specify $\text{colourStab}_\Omega(aH)$ by giving a generator for $\text{colourStab}_\Omega(H)$ and an arbitrary coset representative.

Hence, if we are interested in solving the colour stabilizer for G, and if $H \leq G$, we can recursively call colour stabilizer on each of the cosets of H in G and somehow stitch the answers together. We'll see this entire machinary in action to see how we can solve the colour stabilizer on p-groups.

6 Colour Stabilizer for special groups

To begin with, we shall focus on the case when G is a p-group. At a later point, we would have to generalize to other structured groups but the following discussions would have all the ideas.

p-ColourStabilizer: We are given as input $G = \langle S \rangle \leq S_n$, some permutation $\sigma \in S_n$ and a set $B \subseteq \Omega$ that is a union of orbits . The goal is to compute

To begin with, B would just be Ω but we will need this general form in intermediate steps of the divide-and-conquer.

$$\text{colourStab}_B(\sigma G) := \{\pi \in \sigma G \ : \ b^\pi \sim b \ \forall \ b \in B\},$$

where the output is expected as an ordered pair (π, T) for some $\pi \in S_n$ and $T \subseteq S_n$ such that $\text{colourStab}(\sigma G) = \pi \cdot \langle T \rangle$ (thanks to Lemma 5.14).

6.1 *If G is not transitive*

If B is a union of orbits, say, $B = \Omega_1 \cup \cdots \cup \Omega_r$, then we use

$$\text{colourStab}_B(\sigma G) = \text{colourStab}_{\Omega_1}(\text{colourStab}_{\Omega_2}(\cdots \text{colourStab}_{\Omega_r}(\sigma G)\cdots))$$

6.2 *If G is not primitive*

If G is not primitive, we shall find a block system Γ and the associated block kernel $H \trianglelefteq G$ and if $G = a_1 H \cup \cdots \cup a_r H$ then

$$\text{colourStab}_B(\sigma G) = \bigcup_{i=1}^{r} \text{colourStab}_B(\sigma a_i H).$$

We can try to compute each of the r terms on the RHS and combine them together. There are a few issues with this that need to be addressed:

1. We do not know how large r is. If r is too large, then there would be too many recursive calls.
2. If each of the recursive calls on the RHS return some $\{\pi_i, T_i\}$ such that $\text{colourStab}_B(\sigma a_i H) = \pi_i \langle T_i \rangle$, how are we supposed to combine them together to a single $\{\pi, T\}$?

For the first issue, it is important that we make H as large as possible so that r is as small as possible. For this, it makes sense to take the block system at the highest level of the structure forest (Algorithm 18). So let's assume that Γ is that block system. What can we say about $|\Gamma|$ and r in that case? Observe that G/H acts transitively, faithfully and primitively on Γ! Hence, by Observation 5.13, $G/H = C_p$ and $|\Gamma| = p$. Hence, $r = [G : H] = p$. This solves the first issue.

As for the second issue, the following lemma tells us how to paste the different recursive calls together.

Lemma 6.1. *Let $\langle T \rangle = H \leq G$ and suppose $a_1 H \cup \cdots \cup a_r H$ is a coset of G. Then $S = \left\langle T \cup \left\{a_1^{-1} a_i \ : \ i = 2, \ldots, r\right\}\right\rangle$ and $a = a_1$* generate *the coset given by $a_1 H \cup \cdots \cup a_r H$.*

Proof. Exercise. □

This lemma essentially solves the second issue. Each recursive call would return some $\{\pi_i, T_i\}$ (if non-empty) for each $i = 1, \ldots, r$. Firstly, observe that $\langle T_i \rangle = \text{colourStab}_B(H)$ for all i so we might as well have a single T. Then, an application of the above lemma tells us how to paste the different cosets together.

We now write the entire algorithm for colour stabilizer for general groups (which may not be efficient for general groups as we may not have an analogue of Observation 5.13 for general groups) that is guaranteed to be efficient in the case of p-groups.

Algorithm 19: ColourStab

Input : $G = \langle S \rangle \leq S_n$, a permutation $\sigma \in S_n$ and a union of orbits $B \subseteq \Omega$
Output: $\{\pi, T\}$ such that $\text{colourStab}_B(\sigma G) = \pi \langle T \rangle$

1 **if** *G is "small"* **then**
2 Enumerate all the elements of G and solve $\text{colourStab}_\Omega(\sigma G)$ by brute-force.
3 **return** computed colour stabilizer.
4 **if** *G does not act transitively on B* **then**
5 Break B into orbits $\{\Omega_1, \Omega_2, \ldots \Omega_k\}$
6 **return** $\text{colourStab}_{\Omega_1}(\text{colourStab}_{\Omega_2}(\ldots \text{colourStab}_{\Omega_k}(\sigma G) \cdots))$
7 **else if** *G does not act primitively* **then**
8 Compute the structure forest of G using Algorithm 18. Let Γ be the block system at the highest level of the structure forest
9 Compute a generating set U for $H = \text{BlockKernel}_G(\Gamma)$.
10 Compute $\{a_1, a_2, \ldots a_r\}$, a set of representatives of H in G
11 **for** $i = 1, \ldots, r$ **do**
12 Compute the colour stabilizer of $\sigma a_i H$ recursively as $\{b_i, T\} = \text{ColourStab}(U, \sigma a_i, B)$
13 **return** $\left\{b_1, T \cup \left\{b_1^{-1} b_2, b_1^{-1} b_3, \ldots b_1^{-1} b_r\right\}\right\}$
14 **else**
  `// Want to set ''small'' in line 1 appropriately so that we never get here.`
15 **return** Fail

In the above algorithm, depending on the group we are working with, we need to appropriately assign the meaning to "small" in line 1. If the group G is from some structured class, we want this to be an upper bound of the size of any primitive, transitive group from this class. For the case of p-groups, this value would just be p (Observation 5.13).

Theorem 6.2. *If $G = \langle S \rangle \leq S_n$ is a p-group, if "small" in line 1 is set to p then Algorithm 19 always outputs the colour stabilizer in time* $\text{poly}(n, p, |S|)$.

Proof. Let us get a sense of the number of recursive calls and the size of each of those instances. Say $|B| = m$ and let $T(m)$ be the time taken to compute

colourStab$_B(G)$. If G does not act transitively on B, then we decompose B into orbits. Say $B = \Omega_1 \cup \cdots \Omega_t$ and let $m_i := |\Omega_i|$. Once we have these orbits, we then compute the colour stabilizer in each of these orbits.

Each such recursive call would then construct the highest level block system from the structure forest of Ω_i. Let Γ_i be the highest level block system of Ω_i and let H_i be the block kernel. Since G/H_i acts transitively, primitively and faithfully* on Γ_i, we know that $|\Gamma_i| = p$ and $[G : H_i] = p$. Hence, if $G = a_{i1}H_i \cup a_{ip}H_i$, for each i, then if we can compute colourStab$_{\Omega_i}(\sigma a_{ij}H_i)$ for each $i = 1, \ldots, t$ and $j = 1, \ldots, p$, we can stitch together a generating set for colourStab$_B(\sigma G)$ with polynomial overhead. Hence, if $T'(m_i)$ is the time for computing colourStab$_{\Omega_i}(\sigma a_{ij}H_i)$, then

** Why faithful?*

$$T(m) \leq pT'(m_1) + \cdots + pT'(m_t) + \text{poly}(n, m, |S|).$$

Now comes the key observation: Each H_i stabilizes the blocks in Γ_i and hence the orbits of H_i are of size at most $|\Omega_i|/p$ as H_i never takes elements of one block to another. Therefore, if $\Gamma_i = \{\Omega_{i1}, \ldots, \Omega_{ip}\}$ (and recall $|\Omega_i| = m_i/p$) we have

$$\begin{aligned} &\text{colourStab}_{\Omega_i}(\sigma a_{ij}H_i) = \text{colourStab}_{\Omega_{i1}}(\cdots \text{colourStab}_{\Omega_{ip}}(\sigma a_{ij}H_i)\cdots) \\ &\qquad \implies T'(m_i) \leq pT(m_i/p) + \text{poly}(n, m, |S|). \end{aligned}$$

Putting it all together,

$$T(m) \leq p^2 \left(T(m_1/p) + \cdots + T(m_t/p)\right) + \text{poly}(n, m, |S|).$$

where $\sum m_i = m$. Solving this recurrence shows that the time taken by the algorithm is poly$(n, m, |S|)$. □

7 Graph isomorphism for bounded degree graphs

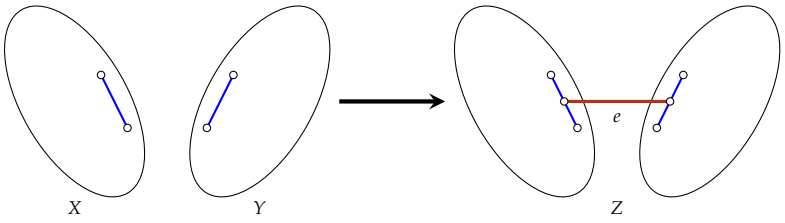
We have already seen that graph isomorphism and graph automorphism are equivalent (Lemma 1.6). In fact, it suffices to get a generating set for the set of automorphisms that fixes an edge.

$$\text{Aut}_e\left(X = (V, E)\right) = \{\sigma \in \text{Sym}(V) \ : \ \sigma(e) = e \text{ and } (u, v) \in E \Leftrightarrow (\sigma(u), \sigma(v)) \in E\}$$

We stress that it is possible that a $\sigma \in \text{Aut}_e(X)$ swaps the two end-points of e but all we insist is that the edge e is mapped to the same edge (and end-points could possibly be swapped).

Lemma 7.1. GraphIso *reduces to the problem of computing a generating set for* $\text{Aut}_e(X)$.

Proof. By picture:



□

It is also worth noting that the above reduction also maintains the maximal degree of the graph X (unless X has maximal degree at most 2 which is just a set of cycles and paths).

The general road-map would be the following.

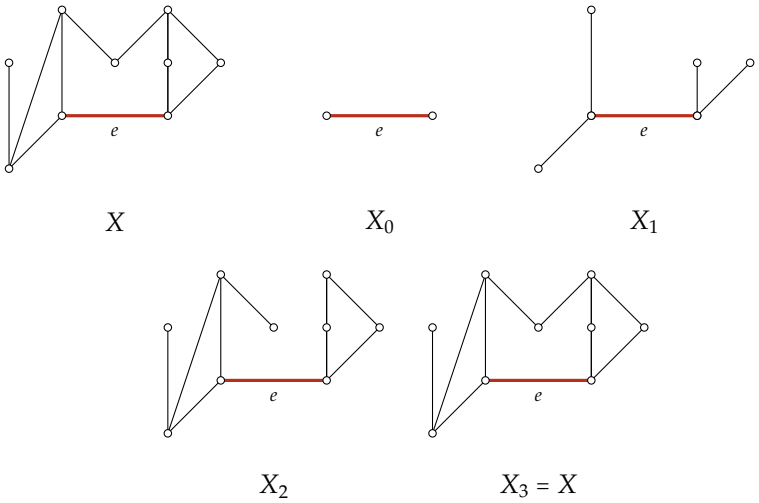
- **Step 1: Structural properties -** Prove some structural properties of the group $\mathrm{Aut}_e(X)$ when X is a bounded degree graph.
- **Step 2: Reduction to colour stabilizers -** Reduce the task of computing a generating set of $\mathrm{Aut}_e(X)$ to colourStab *on such structured groups*.
- **Step 3: Solving** colourStab **-** Show that colourStab *of such structured groups* can be solved in polynomial time.

We shall first look at graph isomorphism for the class of degree 3 graphs, also called *trivalent graphs*. These would have all of the crucial ideas in Luks' algorithm for bounded degree graph isomorphism.

7.1 *Trivalent Graphs*

Assume without loss of generality that X is a connected trivalent graph (if not, we have to work with each connected component, etc.). The goal is to get a generating set for $\mathrm{Aut}_e(X)$. We shall do that incrementally and in the process prove some structural properties of the automorphism group, as well as see how to get a generating set for it. The idea is to discover the remaining edges in order of their "distance" from e, and use induction on the chain of sub-graphs, denoted by X_i, that arise.

Formally, let X_i consist of vertices that are reachable from one of the two end points of e by a path of length at most i, and the edges in X_i would be all those edges whose distance from e is at most i, i.e. there is a path of length at most i containing that edge and e. The following picture is an example of these intermediate graphs.



We first proceed to **Step 1** of the above outline which is to prove a structural result about $\mathrm{Aut}_e(X)$ in the case when X has degree at most 3. We will prove the following amazing result by Tutte.

It is one of those results where the statement and the proof are both equally amazing.

Theorem 7.2 (Tutte)**.** *If X is a connected trivalent graph, then $\mathrm{Aut}_e(X)$ is a 2-group, i.e., $|\mathrm{Aut}(X)| = 2^k$.*

What happens if X is not connected?

Proof. We shall prove by induction on r that $\mathrm{Aut}_e(X_r)$ is a 2-group. In the case of X_0, which consists of a single edge, the claim is true as $\mathrm{Aut}_e(X_0)$ consists of just two elements — the identity and the swap of the two end points of e. Now assuming that $\mathrm{Aut}_e(X_r)$ is a 2-group, we wish to show that $\mathrm{Aut}_e(X_{r+1})$ is also a 2-group. Informally, we shall show that the new elements, vertices or edges so discovered, either reduce or increase the size of $\mathrm{Aut}_e(X_{r+1})$ by a factor of 2^j compared to $\mathrm{Aut}_e(X_r)$.

Let $G_r = \mathrm{Aut}_e(X_r)$, and let V_r be the set of vertices in X_r that are not in X_{r-1} (vertices that haven't been discovered yet). We already have the claim for $r = 0$. For $r > 0$, let us define a function $\phi_r : G_r \to G_{r-1}$. This is just the projection map, i.e., it ignores the action of G_r on $V_r \setminus V_{r-1}$ as any e-preserving automorphism of X_r induces an automorphism of X_{r-1}.* It is easy to see that ϕ_r is therefore a homomorphism and hence we have:

* *Any g in G_r must permute V_{r-1} within itself. Why?*

$$\frac{|G_r|}{|\ker(\phi_r)|} = |\mathrm{Im}(\phi_r)|.$$

So, if we demonstrate that $|\ker(\phi_r)|$ is a 2-group, we would have established the theorem because by the inductive hypothesis, $|G_{r-1}|$ is a 2-group, and hence $\mathrm{Im}(\phi_r) \leq G_r$ is a 2-group.

Let us now articulate on $\ker(\phi_r) = \{g \in \mathrm{Aut}_e(X_r) \mid \phi_r(g) = \mathrm{id} \text{ on } V_{r-1}\}$. We now wish to know what the elements of $\mathrm{Aut}_e(X_r)$ whose action on X_{r-1} is just identity. For this purpose, for a vertex $a \in V_r \setminus V_{r-1}$, define the *neighbourhood* $N(a) = \{b \in V_{r-1} \,:\, \{a, b\} \in E_r\}$.

Call distinct a and b twins* if $N(a) = N(b)$, and distinct a, b and c triplets if

* *What else?*

$N(a) = N(b) = N(c)$. $|N(a)| \leq 3$ and there are no triplets as the graph is trivalent. Hence, the vertices in $V_r \setminus V_{r-1}$ can be partitions into twins and single children. As, any $g \in \ker(\phi_r)$ is an automorphism that fixes V_{r-1}, its action on the set $V_r \setminus V_{r-1}$ may only be that of swapping twins. Furthermore, any permutation which fixes V_{r-1} and swaps twins is a member of $\ker(\phi_r)$.** Thus, $\ker(\phi_r)$ is generated by twin swaps, and hence, it is a 2-group. □

*** Why? Ahh! the edges between the vertices of $V_r \setminus V_{r-1}$ are not present in E_r; they only show up in E_{r+1}.*

An alternate proof by Uma Girish was that any $g \in \ker(\phi_r)$ satisfies $g^2 = \mathrm{id}$. This is easy to check and immediately implies $\ker(\phi_r)$ is a 2-group.

For every r and every vertex v, let X_r, G_r, $N(v)$ be as defined in the above proof. We note that the proof actually gives us an algorithm to find a generating set for $\ker(\phi_r)$.

Algorithm 20: Kernels between successive layers

Input : X_r, X_{r-1}
Output: T such that $\langle T \rangle = \ker(\phi_r)$

```
T = ∅
For every v ∈ V_r \ V_{r-1}, find N(v).
Partition V_r \ V_{r-1} into twins and single children.
for each pair of twins v, v' ∈ V_r \ V_{r-1} do
    Add the transposition (v, v') to T.
return T
```

Now the road map to compute the generating set for $\mathrm{Aut}_e(X)$ would also be an inductive procedure of starting from X_0 and slowly building a generating set for the e-preserving automorphisms of larger X_is. Suppose we have a generating set for $G_{r-1} = \mathrm{Aut}_e(X_{r-1})$. The following would be our strategy.

1. We shall use the map

   $$\phi_r : G_r \to G_{r-1}$$

   and first construct the generating set for $\ker(\phi_r) = \langle K \rangle$ via Algorithm 20.

2. Recursively find a generating set for $\mathrm{Im}(\phi_r)$. Note that $\mathrm{Im}(\phi_r)$ consists of precisely the automorphisms of X_{r-1} that can be *extended* to X_r. If we could compute a generating $\langle T \rangle = \mathrm{Im}(\phi_r)$, then for each $g \in T$ let $\tilde{g}$ be an extension of g to an automorphism of X_r. Let $\tilde{T} = \{\tilde{g} : g \in T\}$. Then, $K \cup \tilde{T}$ generate G_r as in essense this includes a generating set for a subgroup ($\ker(\phi_r) \leq G_r$) and also its coset representatives (lifts of $\mathrm{Im}(\phi_r)$).

So now, the only thing left to do is to find a generating set for $\mathrm{Im}(\phi_r)$ given G_{r-1}. To do that we shall show that $\mathrm{Im}(\phi_r)$ is a suitable colour stabilizer of G_{r-1}. Since we know from Theorem 7.2 that G_{r-1} is a 2-group, we can use Theorem 6.2 to solve the colour stabilizer in polynomial time.

If $\sigma \in G_r$, then note that $N(\sigma(v)) = \sigma(N(v))$. In words, this is just stating that the neighbours of v must be mapped to the neighbours of $\sigma(v)$. Since we are working with a trivalent graph, $N(v)$ is some subset of X_{r-1} of size at most 3. Define

$$\mathcal{A} = \{\text{subsets of } V_{r-1} \text{ of size at most } 3\}.$$

Suppose $a \in \mathcal{A}$ such that there are $v, v' \in V_r \setminus V_{r-1}$ such that $N(v) = N(v') = a$ but $b \in \mathcal{A}$ such that there is only one v'' such that $N(v'') = b$, then note that no $\sigma \in G_r$ can map a to b. Hence it makes sense to define the following subsets.

$$A_1 = \left\{a \in \mathcal{A} \,:\, \left|N^{-1}(a)\right| = 1\right\}$$
$$A_2 = \left\{a \in \mathcal{A} \,:\, \left|N^{-1}(a)\right| = 2\right\}$$
$$A = \{\{u, v\} \in \mathcal{A} \,:\, \{u, v\} \in X_r \setminus X_{r-1}\}$$

If we were to consider the action of G_r on $\mathcal{A}$ via the natural lift, then clearly any $\sigma \in G_r$ must stabilize A_1, and A_2 and A. Turns out, the converse is also true.

Lemma 7.3. *Let $\sigma \in G_{r-1}$ such that σ stabilizes A_1, A_2 and A. Then σ can be extended to an automorphism of X_r.*

Proof. The edges in $X_r \setminus X_{r-1}$ are of three kinds:

- edges present in X_{r-1},
- edges connecting a vertex in V_{r-1} to a vertex in $V_r \setminus V_{r-1}$,
- edges connecting two vertices in V_{r-1}, but was not present in X_{r-1}.

The first kind of edges are of course preserved as $\sigma \in G_{r-1}$. The third kind of edges are also preserved by σ as we are given that σ stabilizes A. We only need to ensure that second kind of edges are preserved by σ as well.

This is sort of immediate once we write things down properly. Consider listing all the vertices in $V_r \setminus V_{r-1}$ on the left and all elements of $\mathcal{A}$ on the right. Imagine connecting each v on the left with the $a = N(v)$ on the right. Just naturally extend σ to the vs by following these connections to the left. For instance, if $\sigma(a) = \sigma(a')$ and $N^{-1}(a) = \{v\}$ and $N^{-1}(a') = \{v'\}$, then extend σ by mappign v to v'. If $\sigma(a) = \sigma(a')$ and $N^{-1}(a) = \{v_1, v_2\}$ and $N^{-1}(a') = \{v_1', v_2'\}$, then extend σ by mapping v_1 to v_1' and v_2 to v_2' (or vice-versa; both would be valid extensions but we just need one). This therefore preserves all edges between V_{r-1} and $V_r \setminus V_{r-1}$. Hence σ can indeed be extended to an automophism of X_r and hence $\sigma \in \mathrm{Im}(\phi_r)$. □

Since we have solved $\mathrm{colourStab}_\Omega(G)$ for G which are p-groups, consider the following algorithm to find the generating set for $\mathrm{Im}(\phi_r)$ given G_{r-1}. Note that since we are working with trivalent graphs, G_{r-1} is a 2-groupfor every r by Theorem 7.2 and hence we can compute $\mathrm{colourStab}_\Omega(G_{r-1})$ for any Ω.

Algorithm 21: Image between successive layers

Input : $G_{r-1} = \langle S \rangle, X_r, X_{r-1}$
Output: T such that $\langle T \rangle = \mathrm{Im}(\phi_r)$

1 Consider the action of G on $\mathcal{A} = \{\text{subsets of } V_{r-1} \text{ size at most } 3\}$.
2 Define $A = \{(u, v) \in X_r \setminus X_{r-1} : u, v \in V_{r-1}\}$.
3 Define $A_1 = \{a \in \mathcal{A} : N(v) = a \text{ for exactly one } v \in V_r\}$.
4 Define $A_2 = \{a \in \mathcal{A} : N(v) = a \text{ for exactly two } v \text{ s } \in V_r\}$.
5 Colour elements of $(A_1 \cap A)$, $(A_1 \setminus A)$, $(A \setminus A_1)$, A_2 and $A \setminus (A_1 \cup A_2 \cup A)$ with different colours.
6 **return** $\mathrm{colourStab}_{\mathcal{A}}(G_{r-1})$

As mentioned earlier, once we obtain a generating set T for $\mathrm{Im}(\phi_r)$, we lift each of its elements to form $\tilde{T}$, add to it the generating set for $\ker(\phi_r)$ and we get a generating set for G_r.

Repeating this for $r = 1, \ldots,$ we eventually have $X_r = X$ and thus a generating set for $\mathrm{Aut}_e(X)$. Hence we have completed the following theorem.

Theorem 7.4 (Luks)**.** *Graph isomorphism for trivalent graphs can be solved in deterministic polynomial time.* □

7.2 *Generalizing to higher (but bounded) degree graphs*

Let us go through the motions again for graphs of degree bounded by d (think of $d = 100$ or something). Once again, let us consider the layered graph $X_0, X_1, \ldots,$ and let $G_r = \mathrm{Aut}_e(X_r)$ and $\phi : G_r \to G_{r-1}$ is the natural projection map. Previously, we first showed that the kernel of ϕ_r was a 2-group and that enabled us to inductively show that $\mathrm{Aut}_e(X_r)$ is a 2-group. However, this need not be the case here. Nevertheless, the kernel is still a pretty structured group.

Structure of $\ker(\phi_r)$

As earlier, we shall say that $\{v_1, \ldots, v_t\} \in V_r \setminus V_{r-1}$ are *identical siblings* if $N(v_1) = \cdots = N(v_t)$. Note that $t < d$, for the same reasons that in a trivalent graph we cannot have triplets. The following is a natural generalization to obtain a generating set for $\ker(\phi_r)$ and the proof is straightforward.

Lemma 7.5. *Let $V_r \setminus V_{r-1} = T_1 \cup \cdots \cup T_k$ be a partition of $V_r \setminus V_{r-1}$ into sets of identical siblings. Then,*

$$\ker(\phi_r) = \mathrm{Sym}(T_1) \times \cdots \times \mathrm{Sym}(T_k).$$

This isn't quite as nice as a 2-group but nevertheless what we do know is that $|T_i| \leq d - 1$ which for us is a constant. So in some sense, $\ker(\phi)$ has made as a combination of *small* groups. This is formalized by the notion of *composition factors* and we briefly describe this now.

Groups with bounded composition factors

Definition 7.6 (Bounded composition factors)**.** *A group G is said to have* composition factors bounded by d*, denoted by $G \in \mathcal{B}_d$ if there is a normal series*

$$G = G_0 \rhd G_1 \rhd \cdots \rhd G_r = \{\mathrm{id}\}$$

such that G_i/G_{i+1} is isomorphic to a subgroup of S_d. ◊

If $K = \mathrm{Sym}(T_1) \times \cdots \times \mathrm{Sym}(T_r)$ with $|T_i| < d$, then $K \in \mathcal{B}_d$ as

$$K = K_0 \rhd K_1 \rhd \cdots \rhd K_r$$

where $K_i = \mathrm{Sym}(T_{i+1}) \times \cdots \times \mathrm{Sym}(T_r)$ and successive quotients are just $\mathrm{Sym}(T_i) \leq S_d$. Hence, $\ker(\phi_r) \in \mathcal{B}_d$.

The class $\mathcal{B}_d$ have some very useful properties that we will state without proof.

Lemma 7.7. *If $G \in \mathcal{B}_d$ and $H \leq G$, then $H \in \mathcal{B}_d$.*

Lemma 7.8. *If $N \lhd G$ and if $N \in \mathcal{B}_d$ and $G/N \in \mathcal{B}_d$, then $G \in \mathcal{B}_d$.*

With these two lemmas, we have an analogue of Tutte's theorem in this setting.

Lemma 7.9. $G_r = \mathrm{Aut}_e(X_r) \in \mathcal{B}_d$.

Proof. As in Theorem 7.2, we shall prove this by induction. Consider the map $\phi_r : G_r \to G_{r-1}$. We know that $G_r/\ker(\phi_r) \cong \mathrm{Im}(\phi_r) \leq G_{r-1}$. By Lemma 7.7 and Lemma 7.8, it follows that $G \in \mathcal{B}_d$. □

Great! We know some structural result about the groups we are working with.

Computing a generating set for $\mathrm{Im}(\phi_r)$

As earlier, we can let $\mathcal{A}$ be the set of subsets of $V_r \setminus V_{r-1}$ of size at most d, and define sets $A_1, \ldots, A_{d-1}$ as

$$A_i = \left\{a \in \mathcal{A} \,:\, \left|N^{-1}(a)\right| = i\right\}$$

and

$$A = \left\{\{u, v\} \in \mathcal{A} \,:\, \{u, v\} \in X_r \setminus X_{r-1}\right\}.$$

As in Lemma 7.3, any $\sigma \in G_{r-1}$ that stabilizer the above sets (when G_{r-1} is acting on $\mathcal{A}$) can be extended to a $\tilde{\sigma} \in G_r$.

We still however need to solve the colour stabilizer problem. For that, it was important to know that any transitive, faithful and primitive p-group must be small. Fortunately, there is an analogue of that for $\mathcal{B}_d$ which again we shall state without proof.

Theorem 7.10 (Babai, Cameron, Palfy)**.** *There is an absolute constant $c > 0$ such that if $G \in \mathcal{B}_d$ act faithfully, transitively and primitively on a set Ω of size n then, $|G| \leq O(n^{cd})$.*

With this result, it is relatively straightforward to check that the colour stabilizer problem on groups in $\mathcal{B}_d$ can be solved efficiently.

Theorem 7.11. *Let $G = \langle S \rangle \in \mathcal{B}_d$ acting on Ω of size m. Given S, a $\sigma \in S_m$ and $B \subseteq \Omega$, a generating set for* $\mathrm{colourStab}_B(\sigma G)$ *can be computed in deterministic* $\mathrm{poly}(|S|, m^d)$ *time.*

With this theorem, it follows that we can compute the generating set for $\mathrm{Im}(\phi_r)$ in $\mathrm{poly}(n^{d^2})$ time (since G is now acting on $\mathcal{A}$ which has size $m = n^d$, and the colour stabilizer algorithm takes time $\mathrm{poly}(m^d) = n^{O(d^2)}$). Thus putting everything together, we get the following theorem.

Theorem 7.12 (Luks)**.** *Graph isomorphism for graphs of degree at most d can be solved in deterministic $n^{O(d^2)}$ time.*

8 General graph isomorphism

Lecture 10: February 20th, 2017 and Lecture 11: February 24th, 2017

In the previous lecture, we have seen algorithms to solve the Graph Isomorphism for restricted graphs like *Bounded degree* graphs and *Trivalent* graphs. In this lecture, we will see an algorithm due to Babai, Luks and Zemlyachenko which runs in time $n^{O(n^{2/3})}$. Note that the naïve algorithm runs in time $n! \approx n^n$.

8.1 *Colour Refinements*

The natural thing to do for solving Graph Isomorphism is to break up the vertices into groups which are distinct in some way there by reducing the search space. We can think of this grouping as a colouring of the graphs. One way of grouping vertices in a graph is according to their degree because in any Isomorphism between X and Y, degree d vertices in X can only be mapped to degree d vertices in Y. We can further extend this argument and say neighbours of degree d vertices in X can only be mapped to neighbours of degree d vertices in Y and so on. This idea is used most of the time in practice. The Weisfeiller Lehman refinmenet process, which we see below, formalizes this idea.

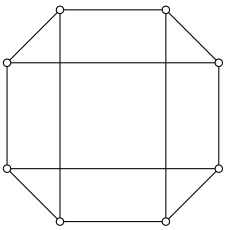
Weisfeller-Lehman colour refinement

Let's assume we have a colouring $\chi : V \to [k]$ to begin with. This could be just assigning the degree of each vertex as its colour, or even the trivial colouring. The following is a way to attempt to *refine* the colouring $\chi : V \to [k]$ to a new colouring χ' as follows:

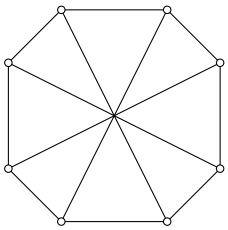
- On each vertex v, set its colour as $(\chi(v); d_1, \ldots, d_k)$ where where d_i is the number of neighbours of v that are given colour i by χ.
- Since the graph has at most n vertices, there can be at most n distinct vectors as colours. Re-index them using $[k']$ for a suitable k'.

This will indeed be a refinement of the previous colouring in the sense that if $\chi'(v) = \chi'(u) \implies \chi(v) = \chi(u)$. By repeating the above process, at some point there would be no further refinement in the colouring. We shall refer to this as the *WL process*, where we start with a given colouring χ and keep refining until the colouring stabilizes.

When we are given two graphs X and Y that we need to check if they are isomorphic, we can simultaneously run the WL process on both graphs starting with the colouring by degree. If we ever notice a discrepancy (that is, X has more 1 coloured vertices than Y), we immediately know that the graphs are not isomorphic. However, we might still have non-isomorphic graphs that end up with the same WL refinements. Here is an example:



X doesn't have 5 cycles



Y has a 5 cycle

So whenever the WL process gets stuck, we need to force progress in some way. This is done by what is called *individualisation* where you pick an arbitrary vertex $v \in X$ and give it a completely fresh colour. This way, the WL process would hopefully refine the colours further. The issue now of course is that we do not know what the correct image of v is in the graph Y – which vertex of Y must be coloured by this new colour? We know, if they are indeed isomorphic, then this must be one of the vertices of Y that had the same (previous) colour of v. We shall try out all possibly candidates.

Thus, every time we individualize a vertex, we make the colours more refined, but we pay a *multiplicative cost* of (potentially) n we we need to try out all possibilities of this individualized vertex in Y. But here is a heuristic that we can attempt.

Algorithm 22: GraphIso WL+Ind Heuristic

Input : Graphs X and Y on n-vertices

```
Start with χ(v) = deg(v) for every v ∈ X and Y.
Pairs = {(X, Y)}
repeat
    if Pairs is empty then
        return No
    for (X, Y) ∈ Pairs do
        Run the WL process in parallel on both X and Y. If there is is every
          a discrepency observed, remove (X, Y) from Pairs.
        Pick a vertex v ∈ X and individualize v, and call this new coloured
          graph X_v.
        For every u ∈ Y, let Y_u be the graph where u is given the same new
          colour.
        In Pairs, replace (X, Y) with {(X_v, X_u) : u ∈ Y}.
until all graphs in Pairs become "nice"
Run the Graph Isomorphism for "nice" graphs for every (X, Y) in Pairs
  and return Yes if any of them is an isomorphic pair.
```

The ideal scenario is when we only have to individualize a few vertices and we end up with a situation where we know how to solve graph isomorphism. One example could be the setting where all the colour classes are small. In that case, we can use the bounded colour multiplicity algorithm that we have seen earlier. Unfortunately, this approach doesn't work as there are graphs where we need to individualize far too many vertices before the WL process results in

bounded colour class sizes.

Turns out if we work with a new notion called generalized colour valence instead colour class size, then we can hope to reach this situation with few individualization operations.

Definition 8.1. *Generalized colour valence Let $X = (V, E)$ be a graph and $C_1, \ldots, C_m$ be a colouring of V. Define the generalized colour valence for a vertex v and a colour class C_i as*

$$\widetilde{\deg}_i(v) := \min \left\{ \begin{array}{c} \#\text{ neighbours} \\ \text{of } v \text{ in } C_i \end{array} , \begin{array}{c} \#\text{ non-neighbours} \\ \text{of } v \text{ in } C_i \end{array} \right\}.$$

In words, either v has at most $\widetilde{\deg}_i(v)$ neighbours or non-neighbours in C_i.

We shall denote by $\widetilde{\deg}(X)$ the maximum $\widetilde{\deg}_i(v)$ over all possible v, i. ◊

The key point about this definition is the following theorem that we shall defer to the end of this section.

Theorem 8.2. *Graph Iso for graphs with* gcd $\leq d$ *can be solved in $n^{O(d^2)}$ time.*

The following crucial theorem tells us that by isolating a few vertices in the WL algorithm we end up with a graph with small generalized colour valence.

Theorem 8.3 (Zemlyachenko)**.** *For any X on n vertices and $1 \leq d \leq n$, we can find at most $4n/d$ vertices to individualize such that the WL refinement stabilizes to a colouring where $\widetilde{\deg}(X) \leq d$.*

With the above two theorems, the main result of this lecture is immediate.

Theorem 8.4 (Babai,Luks)**.** *Graph Iso can be solved in $n^{O(n^{2/3})}$ time.*

Proof. From Theorem 8.3, obtain a colour refinement which isolates at most $4n/d$ vertices and leaves a colouring such that $\widetilde{\deg}(X) \leq d$. For every vertex that is individualized, try out all possible choices in the other graph. For each such pair, check if the two graphs are isomorphic using the algorithm guaranteed by Theorem 8.2. The over all running time of the algorithm is $\text{poly}(n^{4n/d} \cdot n^{d^2})$. By optimising over d and choosing $d = n^{1/3}$ we get the running time to be $n^{O(n^{2/3})}$. □

All that's left to do is prove Theorem 8.2 and Theorem 8.3.

8.2 *Reducing to the bounded generalized colour valence case*

This section shall be devoted to Zemlyachenko's trick (Theorem 8.3), and this follows from the following lemma.

Lemma 8.5. *Given a graph X and a colouring such that $\widetilde{\deg}(X) \leq D$. Then by doing the WL process + individualizing step at most $2n/D$ times we end up with a colouring such that $\widetilde{\deg}(X) \leq D/2$.*

Proof. We'll run the following process.

1. Run the WL process.

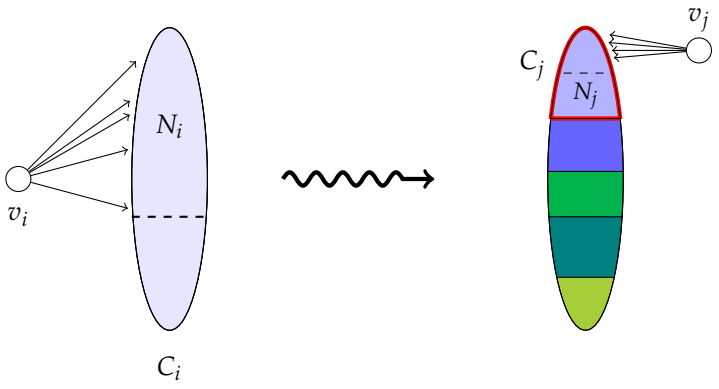
2. If $\widetilde{\deg}(X) \leq D/2$, then Stop; we are already done.
3. Else there exists some vertex v and a colour class C such that v has $> D/2$ neighbours and $> D/2$ non-neighbours in C.
4. Individualize the vertex v and go back to step 1.

Assume that the above process individualizes a series of vertices $v_1, v_2, \ldots, v_r$ and let the colour class (at that instant) that v_i violates be C_i. Since each subsequent colouring is a refinement of the original colouring, we are always guaranteed that $\widetilde{\deg}(X) \leq D$. Hence, we always know that either v_i has at most D neighbours in C_i, or at most D non-neighbours in C_i. If v_i had at most D neighbours in C_i, set N_i to be these neighbours. If instead v_i had at most D non-neighbours in C_i, choose N_i to be these non-neighbours. This is to ensure that $N_i \subseteq C_i$ and

$$\frac{D}{2} < |N_i| \leq D.$$

Claim 8.6. *If $v_1, \ldots, v_r$ are chosen to be isolated, then $N_i \cap N_j = \varnothing$ for $i \neq j$.*

Pf. Assume $i < j$ WLOG. Observe that v_i was adjacent to some set of vertices in C_i and not adjacent to some of them. The moment v_i is individualized, the set of vertices in C_i that were adjacent to C_i and those that are not become different colour classes (or union of colour classes) via the WL process. By the time we get to the point of isolating v_j, the set N_i could have potentially split into many more colour classes. The following should be a better picture of what is happening.



Therefore, by the time we get to individualizing v_j, if $N_i \cap N_j \neq \varnothing$, then the entire colour class C_j must lie inside N_i. But observe that the reason v_j and C_j was chosen was because v_j had more than $D/2$ neighbours in C_j and more than $D/2$ non-neighbours. This forces $|C_j| > D$, which is impossible as $C_j \subseteq N_i$ and $|N_i| \leq D$.

Therefore $N_i \cap N_j = \varnothing$. □

Since each of the N_is are pairwise disjoint, and each has size at least $D/2$, this shows that the number of vertices we would have to individualize at most $2n/D$ vertices before the graph has generalized colour valence bounded by $D/2$. And this completes the proof of the lemma. □

With Lemma 8.5, the proof of Theorem 8.3 is just repeated applications of this lemma to reach the target bound on generalized colour valence.

Proof of Theorem 8.3. To begin with, start with the trivial colouring on all vertices of X. The generalized colour valence is bounded by $D_0 = n$. Using Lemma 8.5, by individualizing at most $2n/D_0 = 2$ vertices and applying the WL process, we now have a graph of generalized colour valence bounded by $D_1 = n/2$. Applying Lemma 8.5 again, by individualizing $2n/D_1 = 4$ vertices, we can make the generalizing colour valence at most $D_2 = 8$. This is just a geometric sum and the number of vertices we need to isolate until we hit the target bound of d is at most $4n/d$. □

8.3 GraphIso *for bounded generalized colour valence graphs*

To finish the proof, we need to prove Theorem 8.2 that claims that graph isomorphism on graphs of generalized colour valence bounded by d can be solved in $n^{O(d^2)}$ time. Much if this would basically observing that the machinary developed for bounded degree graph isomorphism in section 7 almost directly applies here. We briefly sketch the proof but hopefully the sketch should be sufficient for the reader to check that indeed the techniques of section 7 apply here.

Preprocessing

We are given two coloured graphs X and Y as input whose generalized colour degree is bounded by d. First, we run the WL process on both of of them simultaneously. Notice that this can only shrink colour classes so the generalized colour valence cannot increase. Observe that when the WLrefinement stabilizes,

(i) the induced sub-graph $C_i \times C_j$ is bi-regular for all C_i, C_j i.e any two vertices from C_i will have the same number of neighbours in C_j.

(ii) every $v \in C_i$ has either at most d neighbours or at most d non-neighbours in C_j. Without loss of generality, we can assume that the number of neighbours is at most d because if the number of neighbours of each $v \in C_i$ in C_j is more than d then we can switch $C_i - C_j$ edges by non-edges in both X and Y. It easy to check that this preserves graph isomorphism.

Wait... can't this now force you to flip it again for C_j? Think about it.

Whenever the process terminates, the resulting coloured graph would be very regular in the sense that any red vertices will be adjacent to the same number of blue vertices etc. and all these numbers are bounded by d.

Let us further assume the the graph is connected; if it wasn't, we'll work with each connected component etc.

Structure of the automorphism group of X, Y

As in section 7, the problem reduces to finding $\mathrm{Aut}_e(X)$ for a graph X of colour valence bounded by d and e is a special edge. The following lemma basically is the key as to why the earlier machinary holds here too.

Theorem 8.7. *If X is a connected coloured graph whose colour valence is bounded by d, then* $\mathrm{Aut}_e(X) \in \mathcal{B}_d$.

Proof. Just as in Theorem 7.2, we will build the graph X in layers $X_0, \ldots$ and let $G_r = \mathrm{Aut}_e(X_r)$. As earlier, we have a projection map $\phi_r : G_r \to G_{r-1}$. To prove the claim by induction, it suffices to show that $\ker(\phi_r) \in \mathcal{B}_d$.

Suppose $v, u \in X_r \setminus X_{r-1}$. When would it be possible for element $\ker(\phi_r)$ swap these two? Firstly, u and v must have the same colour, say *red*. Furthermore, their neighbourhoods must be identical. We shall call such vertices as *identical siblings*. Hence, the elements of the kernel can only switch around identical siblings.

How many identical siblings can there be? Not more that d; this is because any set of identical siblings must be a set of vertices of the same colour that are adjacent to some vertex in X_{r-1}. We know there aren't more than d of them as X has bounded colour valence.

Therefore, $\ker(\phi_r) \in \mathcal{B}_d$. Thus by induction, so is $G_n = \mathrm{Aut}_e(X)$. □

We just need to follow the same outline for the bounded degree graph isomorphism case. The above theorem also gives a generating set for the kernel. The generating set for the image can be obtained by a colour stabilizer computation as earlier. To give a little more detail, if $X_{r-1} = (V_{r-1}, E_{r-1}$ and if $V_{r-1} = C_1 \cup \cdots \cup C_m$ is the partition into colour classes, then we need to think of G_{r-1} as acting on

$$\Omega' = \binom{C_1}{\leq d} \cup \cdots \cup \binom{C_m}{\leq d} \cup \binom{V_{r-1}}{2}$$

and stabilize the right tuples. We'll leave the rest of the details as an exercise* to the reader. :-)

□Theorem 8.2

Part II: Computations on polynomials

Lecture 12: March 10th, 2017

Notation

- We shall be using the standard blackboard font for fields – $\mathbb{F}$, $\mathbb{C}$, $\mathbb{Q}$ etc.
- We shall use $\mathbb{F}_q$ to denote the finite field of size q (of course, when q is a power of a prime). We shall insist on using $\mathbb{F}_p$ for the finite field of size p and *not* $\mathbb{Z}_p$ (which is often reserved for the field of *p-adic* integers; we won't encounter them in this course).

  We shall use $\mathbb{F}^\times$ shall denote the multiplicative group of $\mathbb{F}$.
- $\mathbb{Q}$ shall refer to the field of rational numbers, and $\mathbb{C}$ shall refer to the field of complex numbers.
- Polynomial rings would be denoted by $\mathbb{F}[x_1, \ldots, x_n]$, and the *field of rational functions* would be denoted by $\mathbb{F}(x_1, \ldots, x_n)$.
- $\deg(f)$ shall refer to the total degree of the polynomial, and $\deg_x(f)$ would be the degree of x in f.

9 Preliminaries

So far we have been looking at computations on groups and we now move on to polynomials. You should be generally pally with polynomials but it would be useful to establish some very basic properties about polynomial rings and fields that we would need in this course. We begin with some basics about rings, fields etc.

Definition 9.1 (Rings)**.** *A ring R consists of a set of elements and two binary operations $+$ and $\times$ that satisfy the following properties:*

- $(R, +)$ *forms a commutative group with identity element called* 0*.*
- *The operation $\times$ is associative, i.e. $(a \times b) \times c) = a \times (b \times c)$,*
- *The operation $\times$ distributes over $+$, i.e. $a \times (b + c) = (a \times b) + (a \times c)$.*

If the operation $\times$ is commutative, then the ring is said to be a commutative ring*.*

It is said to be a domain *(or* integral domain*) if the product of two non-zero elements of R is always non-zero.* ◊

Examples of rings are say $\mathbb{Q}$, $\mathbb{Z}$, $\mathbb{F}[x, y]$ (which are all commutative rings) and the set of $n \times n$ matrices over $\mathbb{F}$ is a non-commutative ring. The set of integers modulo 42 is a ring, but not a domain as $7 \cdot 6 = 0$. Throughout this part of the course, we'll mainly deal with just commutative rings and often polynomial rings such as $\mathbb{F}[x, y]$ etc.

As in the case of groups, once we have rings, we can talk about homomorphisms between two rings R and S. These are maps from $\phi : R \to S$ that gel well with the operations inside R and S. That is, $\phi(ab - c) = \phi(a)\phi(b) - \phi(c)$, etc. And as in the case of groups, we can ask for elements of R that are mapped to 0_S by ϕ — the kernel of ϕ. These are important objects called *ideals*.

Definition 9.2 (Ideal)**.** *A subset $I \subseteq R$ is called an* ideal *if*

- *I forms a sub-ring. That is, it is internally closed under addition, additive inverses and multiplication.*
- *If $p \in I$, then for every $q \in R$, we have $pq \in I$. That is, it is closed under multiplication by elements even outside I.*

We shall say an ideal I is generated *by $\{q_1, \ldots, q_r\}$, and denote this by $I = \langle q_1, \ldots, q_r \rangle$, if*

$$I = \{p_1 q_1 + \cdots + p_r q_r \ : \ p_i \in R \ \forall \ i\} .$$ ◊

For example, if $R = \mathbb{F}[x]$ then the set of all polynomials divisible by x^2 is an ideal $\langle x^2 \rangle$.

Sometimes, rings may not even have a multiplicative identity element* but we won't be looking at such weird objects. But if it did have a multiplicative identity element, then it makes sense to talk about inverses of elements. It may be the case that a lot of the elements in the ring do not have multiplicative inverses. The elements of R that do have a multiplicative inverse are called *units*.

** - Apparently, they are sometimes called rngs. :-/*

A field is a commutative ring, with identity, where every non-zero element is a unit.

Definition 9.3 (Field)**.** *A field* $\mathbb{F}$ *consists of a set of elements and two binary operations* $+$ *and* $\times$ *that satisfy the following properties:*

- $(\mathbb{F}, +)$ *forms a commutative group with identity element called* 0*.*
- $(\mathbb{F} \setminus \{0\}, \times)$ *forms a commutative multiplicative group,*
- *The operation* $\times$ *distributes over* $+$*, i.e.* $a \times (b + c) = (a \times b) + (a \times c)$*.* ◊

A field is a commutative ring where non-zero elements are also invertible. Examples are of course $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ etc. The more interesting examples are *finite fields*.

Kindergarten algebra tells us that if p is a prime, then $\{0, \ldots, p-1\}$ with the usual $\times$ and $+$ modulo p is a field. This shall be referred to as $\mathbb{F}_p$. We can also construct fields of size p^r for any prime p and integer r. For that, it helps to know a few things about polynomial rings.

9.1 *Polynomial rings*

Let us look at the ring $\mathbb{F}[x]$ of univariate polynomials with coefficients from a field $\mathbb{F}$. There is a natural notion of a degree that is associated with these polynomials; we'll denote that by $\deg(f)$. Univariate polynomials have the useful Euclidean property a.k.a division-plus-remainder property.

Lemma 9.4. *Let* $f, g \in \mathbb{F}[x]$*. Then there is unique* $q, r \in \mathbb{F}[x]$ *with* $\deg(r) \leq \deg(g)$ *such that* $f = gq + r$*.*

Thus, univariate polynomial rings $\mathbb{F}[x]$ form what is called a *Euclidean domain*.

Definition 9.5 (Irreducible elements)**.** *A element* $a \in R$ *is said to be* irreducible *if it cannot be written as a product of two or more non-units.* ◊

In the case of $R = \mathbb{F}[x]$, irreducible just means that the polynomial does not have any non-trivial factors.

Definition 9.6 (Unique factorization domains)**.** *A ring* R *is said to be a* unique factorization domain (UFD) *if every element of* R *can be expressed as a product of irreducible elements* uniquely *(up to reordering, and multiplication by units).* ◊

Fact 9.7. *If* R *is a Euclidean domain, then* R *is also a UFD.* □

Fact 9.8. *If* R *is a UFD, then* $R[x]$ *is also a UFD.* □

Corollary 9.9. *Polynomial rings over fields* $\mathbb{F}[x_1, \ldots, x_n]$ *are unique factorization domains.*

We won't really need the proofs of these but the interested reader can find this in any basic algebra text.

One of the most important fact about univariate polynomials over fields is this.

Fact 9.10. *A non-zero univariate polynomial in* $\mathbb{F}[x]$ *of degree at most* d *can have atmost* d *zeros.*

Proof. This basically follows from Lemma 9.4 and showing that if a is a root of $f(x)$ then $(x - a)$ divides f. Divide, degree drops, and repeat. □

9.2 *Fraction fields*

Suppose R is a commutative domain with identity, there is a standard method to create a field from it by introducing fractions.

Definition 9.11 (Fraction fields)**.** *Let R be a commutative ring, with identity, which is a domain. The fraction field of R, denoted by* $\mathrm{Frac}(R)$*, whose elements are* equivalence classes *of*

$$\{(a,b) \ : \ a,b \in R \ , \ b \neq 0\}$$

under the equivalence relation $(a,b) \equiv (c,d)$ if $ad = bc$. The addition and multiplication are defined via

$$[(a,b)] + [(c,d)] := [(ad+bc, bd)]$$
$$[(a,b)] \times [(c,d)] := [(ac, bd)] \qquad \Diamond$$

The field $\mathbb{Q}$ is the fraction field of the ring $\mathbb{Z}$. We shall use $\mathbb{F}(x_1, \ldots, x_n)$ to denote $\mathrm{Frac}(\mathbb{F}[x_1, \ldots, x_n])$, which consists of just rational polynomial expressions.

9.3 *Finite fields*

We have already seen fields $\mathbb{F}_p$ of prime order. To build larger fields, consider $\mathbb{F}_p[x]$, the univariate polynomial ring over $\mathbb{F}_p$. Let $f(x)$ be an irreducible polynomial of degree k.

Now observe by Lemma 9.4, every polynomial $g(x)$ has a unique remainder $r(x)$ of degree at most $(k-1)$ when divided by $f(x)$. The field $\mathbb{F}_{p^k}$ consists of $\left\{r(x) \in \mathbb{F}_p[x] \ : \ \deg(r) \leq k-1\right\}$, where addition and multiplication are defined modulo $f(x)$. That is,

$$r(x) + s(x) := r(x) + s(x) \bmod f(x)$$
$$r(x) \times s(x) := r(x)s(x) \bmod f(x)$$

Lemma 9.12. *The above definition indeed yields a field of size p^k.*

Proof. The only thing that needs to be checked is that multiplicative inverses exists. Notice that the ring constructed above is a finite integeral domain; if $r(x), s(x)$ are non-zero polynomials with $\max\{\deg r, \deg s\} < k$, then $r(x)s(x) \notin f(x)$ because otherwise, $f \mid rs$ would imply $f \mid r$ or $f \mid s$ (by unique factorization in $\mathbb{F}_p[x]$ and irreducibility of $\in \mathbb{F}_p[x]$) which is absurd because of degree consideration. Therefore, it suffices to show that finite integral domains are fields.

Let R be a finite integeral domain, and $\alpha \in R$ is a non-zero element. The function $m_\alpha : R \to R$, $r \mapsto \alpha r$ is injective and hence bijective. Therefore, there is $\beta \in R$ such that $\alpha\beta = 1$. □

Therefore, we now know that for every prime p and every positive integer k, there is a field of size p^k*.

It is sometimes useful to think of the field $\mathbb{F}_{p^r}$ as the vector space $(\mathbb{F}_p)^r$ endowed with a multiplication. The *characteristic** of these fields is p.

* - *assuming that there is an irreducible polynomial of that degree. Let's take that for granted.*

* - *defined as the smallest number k such that* $\underbrace{1 + \cdots + 1}_{k} = 0$

Fields such as $\mathbb{R}$ or $\mathbb{C}$ or $\mathbb{Q}$ are called *characteristic zero* fields. Why zero and not infinite? Given any commutative ring R with multiplicative identity $1_R \in R$, there is a unique ring morphism $\phi : \mathbb{Z} \to R$, induced by $\phi(1_R) = 1$. The kernel $\ker(\phi) \subset \mathbb{Z}$ is an ideal in the principal ideal domain $\mathbb{Z}$, and therefore, there is $n \in \mathbb{Z}_+$ such that $\ker(\phi) = n\mathbb{Z}$. Notice that $n \in \mathbb{Z}_+$ is the minimal non-negative integer satisfying $n \cdot 1_R = 0$; in other words, the ideal $\ker(\phi) = n\mathbb{Z}$ is "the characteristic" of the ring R. Notice that the induced morphism $\mathbb{Z}/n\mathbb{Z} \to R$ is injective. When R is a field, this shows that $\mathbb{Z}/n\mathbb{Z}$ must be an integeral domain, which implies that either $n \in \mathbb{Z}$ is a prime or $n = 0$.

Another useful fact to know about finite fields is that the multiplicative group $\mathbb{F}_q^\times$ is cyclic.

Fact 9.13. *Let $\mathbb{F}_q$ be a finite field. Then, the set of non-zero elements of $\mathbb{F}_q$ under multiplication is a* cyclic group *of order $q - 1$.* □

Thus, there are elements $a \in \mathbb{F}_q$ such that every other $0 \neq b \in \mathbb{F}_q$ can be expressed as a^e. Such elements are called *generators* of $\mathbb{F}_q$ or *primitive elements* of $\mathbb{F}_q$.

9.4 *Adjoining elements and splitting fields*

Let us look at $\mathbb{Q}$. The polynomial $x^3 - 2$ does not have a root in $\mathbb{Q}$. But we can think of adding $\sqrt[3]{2}$ to $\mathbb{Q}$ and *completing* it to a field. For instance, we should add $\sqrt[3]{4}$. This is what is meant by adjoining elements — just take the smallest field that contains the base field (which in our example was $\mathbb{Q}$) and the element that was being adjoined (which in our example was $\sqrt[3]{2}$).

Often, we would be adjoining elements α that are *algebraic*. What this means is that the element α we are adjoining is a root of some polynomial over the base field, such as $\sqrt[3]{2}$. The formal way of adjoining an element α to a field $\mathbb{F}$ is to consider its *minimum polynomial* $\mu(x)$ (which is the smallest polynomial with coefficients in $\mathbb{F}$ such that $\mu(\alpha) = 0$) and consider the following field $\frac{\mathbb{Q}[x]}{\mu(x)}$. The elements here are polynomials of degree strictly smaller than $\mu(x)$, and addition and multiplication are modulo $\mu(x)$. So for example, $\mathbb{Q}(\sqrt[3]{2}) \cong \mathbb{Q}[x]/(x^3 - 2)$. Thus in essense, x plays the role of $\sqrt[3]{2}$.

One point to note here was that the way we were looking at $\mathbb{Q}(\sqrt[3]{2})$ involved denominators that may involve $\sqrt[3]{2}$ but not in $\mathbb{Q}[x]/(x^3 - 2)$. What happened to the denominators?

Lemma 9.14. *If α is algebraic over a field $\mathbb{F}$, then $\mathbb{F}(\alpha) = \mathbb{F}[\alpha]$. That is, any rational expression involving α can be equivalently expressed as a polynomial involving α.*

Proof. Easy exercise. □

If we forget about multiplication, then the field extension $\mathbb{K} = \mathbb{F}[\alpha]$ is a *vector space* over $\mathbb{F}$ of dimension $\deg(\mu)$ (as every element is essentially a polynomial of degree smaller than $\deg(\mu)$. This is called the *degree* of the extension and denoted by $[\mathbb{K} : \mathbb{F}]$. The following property about field extensions is easy to check.

Fact 9.15. *If $\mathbb{K}$ is a finite dimensional extension of $\mathbb{F}$, and if $\mathbb{L}$ is a finite dimensional extension of $\mathbb{K}$, then*

$$[\mathbb{L} : \mathbb{K}] \cdot [\mathbb{K} : \mathbb{F}] = [\mathbb{L} : \mathbb{F}]. \qquad \square$$

Let us get back to $\mathbb{K} = \mathbb{Q}[\sqrt[3]{2}]$. We built this to get a root of the polynomial $x^3 - 2$. However, not all the roots of the polynomial $x^3 - 2$ are in $\mathbb{K}$; the other roots are $\omega\sqrt[3]{2}$ and $\omega^2\sqrt[3]{2}$ where ω is a cube-root of unity. Hence, the polynomial $x^3 - 2$ does factorize in $\mathbb{K}$ as $(x - \sqrt[3]{2})(x^2 + \sqrt[3]{2}x + \sqrt[3]{4})$, but not as linear factors. The smallest field containing $\mathbb{Q}$ where the polynomial $x^3 - 2$ splits into linear factors is called the *splitting field* of $x^3 - 2$. In our example, the splitting field of $x^3 - 2$ over $\mathbb{Q}$ is $\mathbb{Q}[\omega, \sqrt[3]{2}]$.

Recall how we built finite fields $\mathbb{F}_q$ of size $q = p^r$. Every element in the field $\mathbb{F}_q$ satisfies $x^q - x$. Hence, the polynomial $x^q - x$ *splits* in $\mathbb{F}_q$ as $x^q - x = \prod_{\alpha \in \mathbb{F}_q}(x - \alpha)$ and this must be the splitting field as any field where $x^q - x$ splits must have q elements (as the roots are distinct; check the derivative). Thus, one gets the following important fact.

Fact 9.16. *If p is a prime and $r \in \mathbb{N}$, there is a unique finite field (up to isomorphism) of size exactly $q = p^r$ — the splitting field $\mathbb{F}_q$ of $x^q - x$ over $\mathbb{F}_p$.* $\square$

A glimpse of part 2

In this part we shall be dealing with algorithmic tasks pertaining to polynomials over fields. Much of this part would be leading towards the question of polynomial factorization — given a polynomial f, can you find its factors efficiently.

We shall be asking this question for univariate and multivariate polynomials. As in the previous part, we would need to establish some preliminaries.

10 Basic operations on polynomials

Lecture 13: March 14th, 2017

In this section, we shall look at a few very basic operations on polynomials that can be performed in nearly linear time. These are not directly related to the course as you would not be judged for taking $O(n^2)$ time for multiplying polynomials using the high-school method, but it would be a crime to not do these wonderful algorithms in the course.

The setting is that we are working with univariate polynomials over a ring R. We shall assume that R is a commutative ring with identity. We shall assume that the ring operations (multiplications, additions etc.) can be performed in constant time.

Polynomial addition

Given two polynomials f and g in $R[x]$ of degree n each, how many ring operations does it take to add them? This can of course be done term by term — takes $O(n)$ time.

10.1 *Polynomial multiplication*

Given two polynomials f and g in $R[x]$ of degree at most n each, how many ring operations does it take to multiply them? The traditional high-school method of course takes $O(n^2)$ time, but there is a beautiful algorithm that allows one to perform this in nearly linear time.

The running time of this algorithm shall vary by a $\log\log n$ multiplicative factor depending on whether or not the ring R is "nice".

Definition 10.1. *A ring R is said to be n-nice if it contains the n-th principal root of unity.* ◊

We will define the n-th principal root of unity shortly. Let us first state our two main theorems for this section.

Theorem 10.2. *If R is $2n$-nice, $f \cdot g$ can be computed in $O(n \log n)$ time.*
If R is not $2n$-nice, $f \cdot g$ can be computed in $O(n \log n \log\log n)$ time.

The idea is as follows. There are two different ways of representing a degree n polynomial.

1. (Coefficient Representation) The first representation is the natural way where the polynomials are provided as a list of its coefficients.

$$f(x) = a_0 + a_1 x + \cdots + a_n x^n$$
$$g(x) = b_0 + b_1 x + \cdots + b_n x^n$$

2. (Value Representation) f and g are specified by their evaluations at $2n+1$ different points.

You only need $n+1$ evaluations, but the factor 2 would make sense shortly.

$$\begin{aligned} f(\alpha_0) &= \beta_0 & g(\alpha_0) &= \gamma_0 \\ f(\alpha_1) &= \beta_1 & g(\alpha_1) &= \gamma_1 \\ &\vdots & &\vdots \\ f(\alpha_{2n}) &= \beta_{2n} & g(\alpha_{2n}) &= \gamma_{2n} \end{aligned}$$

Note that for any $i \in \{0, 1, \ldots, 2n\}$, if $h = f \cdot g$ then $h(\alpha_i) = \beta_i \times \gamma_i$. The question now therefore is, can we go from the given coefficient representation to the value representation efficiently (and back)? This is what the *Fast Fourier Transform* (FFT) does.

The point of $2n+1$ points is because $h = f \cdot g$ is now a polynomial of degree at most $2n$ and hence needs $2n+1$ points to recover the coefficient representation.

10.2 *Fast Fourier Transform*

The Fast Fourier Transform (FFT) is an efficient algorithm to compute the value representation of a given polynomial where α_is are *principal roots of unity*.

Definition 10.3 (n-th principal root of unity)**.** *An element $\omega \in R$ is an n-th* principal root of unity *if*

1. $\omega^n = 1$.

In class, I had incorrectly claimed that this is equivalent to $\omega^i - 1$ not being a zero divisor. But this is false. Look at $\mathbb{Z}/6\mathbb{Z}$ and the element 5 here. It is a principal 2-th root of unity but $5 - 1$ is zero divisor. (hat-tip: Siddharth Bhandari)

2. *For all* $i \in \{0, \ldots, n-1\}$,

$$\sum_{j=0}^{n-1} \omega^{ij} = 0. \qquad \Diamond$$

For now, let us assume that we do have an $\omega \in R$ which principal n-th root of unity. We now show that f can be efficiently evaluated at the powers of ω

Lemma 10.4. *Let* $f \in R[x]$ *such that* $f(x) = a_0 + a_1 x + \cdots + a_n x^n$. *If* ω *is a* n*-th principal root of unity where* n *is a power of* 2, *then* $f(1), f(\omega), f(\omega^2), \ldots, f(\omega^{n-1})$ *can be together computed in* $O(n \log n)$ *time.*

The constraint of n being a power of 2 is not too limiting; this is just to ensure that we can keep performing a divide-and-conquer strategy.

Proof. If $n = 1$, then duh; so let us assume that n is a power of 2 larger than 1.

We split up $f(x)$ into two parts, the first part containing terms with even powers of x, and the second part containing terms with odd powers of x:

$$f(x) = \underbrace{f_0(x^2)}_{\text{even}} + \underbrace{x f_1(x^2)}_{\text{odd}}.$$

Thus, $f_0(y) = a_0 + a_2 y + a_4 y^2 + \cdots + a_n y^{n/2}$ and $f_1(y) = a_1 + a_3 y + a_5 y^2 + \cdots + a_{n-1} y^{n/2-1}$, where $y = x^2$. We shall first compute ω^i for each $i = 0, \ldots, n-1$ and store them. Suppose we can compute $\left\{f_0(1), f_0(\omega^2), \cdots, f_0(\omega^{2\cdot(n/2)})\right\}$ and also compute $\left\{f_1(1), f_1(\omega^2), \cdots, f_1(\omega^{2\cdot(n/2)})\right\}$. Notice that for any $k \in \{0, \ldots, n-1\}$, we have $\omega^{2k} = \omega^{2i_k}$ for some $i_k \in \{0, \ldots, n/2\}$. Therefore,

$$\begin{aligned} f(\omega^k) &= f_0(\omega^{2k}) + \omega^k f_1(\omega^{2k}) \\ &= f_0(\omega^{2i_k}) + \omega^k f_1(\omega^{2i_k}) \end{aligned}$$

Therefore each $f(\omega^k)$ can be computed from the previously computed values with just a few more ring operations. This is the algorithm.

Algorithm 23: FFT

Input : a polynomial f of degree at most n, and a n-th principal root of unity ω where n is a power of 2.

Output: The evaluations $\{f(1), f(\omega), \ldots, f(\omega^{n-1})\}$.

```
// Base case
```
if $n = 1$ **then**
  return $f(1)$

Collect the even degree terms and odd degree terms of f to write it as $f = f_0(x^2) + xf_1(x^2)$.

```
// Observe that ω^2 is a principal (n/2)-th root of of unity.
```
Recursively compute FFT(f_0, ω^2) and FFT(f_1, ω^2).

for $i = 0, \cdots, n-1$ **do**

  $k = 2i \bmod n/2$

  ```
  // This is to ensure that 0 ≤ k < n/2, and also note that k
     will be even.
  ```

  Let $\beta = f_0(\omega^k)$ and $\gamma = f_1(\omega^k)$, which were computed earlier.

  Store the value $f(\omega^i) = \beta + \gamma\omega^i$.

return $\{f(1), \ldots, f(\omega^{n-1})\}$.

We are now ready to state our recurrence. If $T(n)$ is the time complexity for FFT when ω is an n-th root of unity, then

$$T(n) = 2T(n/2) + O(n).$$

This is a well known recurrence that evaluates to $T(n) = O(n \log n)$, completing the proof. □

Let us go back to the question of polynomial multiplication. We are given the coefficient representations of two degree less than n polynomials f and g. Let ω be a $2n$-th root of unity and say n is a power of two. Algorithm 23 tells us that we can compute their value representation very efficiently i.e. in $O(n \log n)$ time and then compute the value representation of $h = f \cdot g$ with $O(n)$ multiplications of the corresponding values.

$$\{f(\omega^i) \mid i = 0, 1, 2, \ldots 2n-1\}$$
$$\{g(\omega^i) \mid i = 0, 1, 2, \ldots 2n-1\}$$

$h = f \cdot g$ is given by

$$\{h(\omega^i) \mid h(\omega^i) = f(\omega^i)g(\omega^i), i = 0, 1, 2, \ldots 2n-1\}$$

What remains to be done now is to convert h back to its coefficient representation. This can be done by performing another FFT! Basically if we invoke the FFT algorithm we just described using ω^{-1} instead of ω, this almost inverts the process! To understand this, let us express the connection between the coefficient

representation and the value representation of the polynomial f using matrices.

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{bmatrix}}_{M_\omega} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} f(1) \\ f(\omega) \\ f(\omega^2) \\ \vdots \\ f(\omega^{n-1}) \end{bmatrix}}_{F}$$

$$M_\omega \cdot A = F$$

Here, the matrix M_ω has a special name — it is known as the *Vandermonde* matrix. And since we know that $\sum_j \omega^{ij} = 0$ for all $i \in \{1, \cdots, n-1\}$, it is easy to check that

$$M_\omega \cdot M_{\omega^{-1}} = n \cdot I.$$

Hence, to go back to the coefficient representation, we only need to do an FFT with respect to ω^{-1}, and divide each coefficient by n. There is one slight technicality here which is that n is a power of 2 and the ring could be something like $\mathbb{F}_{2^r}$ where we cannot divide by 2. But basically the idea is that there wasn't anything special about 2 — we just needed n to be a power of something small so that we can keep recursing. We leave it to the readers to figure out how to make the process work with n is a power of 3 instead.

This completes the proof of the reverse-FFT and hence the algorithm for polynomial multiplication (first part of Theorem 10.2).

What if R does not have a $2n$-th root of unity?

In case R is not a nice ring, we work with a new ring R'_ℓ ($\ell < n$) where we manufacture a root of unity. Suppose we are given a polynomial f in its coefficient representation:

$$f(x) = a_0 + a_1 x + \cdots + a_n x^n.$$

We will instead represent it via $\widetilde{f}(x, y)$, a bivariate polynomial such that $f(x) = \widetilde{f}(x, x^\ell)$ (we shall fix ℓ soon).

$$\begin{aligned} f(x) = &\left(a_0 + a_1 x + \cdots + a_{l-1} x^{l-1}\right) \\ &+ x^\ell \left(a_\ell + a_{\ell+1} x^1 + \cdots + a_{2\ell-1} x^{\ell-1}\right) \\ &\cdots + x^{k\ell} \left(a_{k\ell} + a_{k\ell+1} x + \cdots + a_{(k+1)\ell-1} x^{\ell-1}\right) \\ &\text{where } (k+1)\ell = n. \\ \widetilde{f}(x, y) = &\left(a_0 + a_1 x + \cdots + a_{l-1} x^{l-1}\right) \\ &+ y \left(a_\ell + a_{\ell+1} x^1 + \cdots + a_{2\ell-1} x^{\ell-1}\right) \\ &\cdots + y^k \left(a_{k\ell} + a_{k\ell+1} x + \cdots + a_{(k+1)\ell-1} x^{\ell-1}\right) \end{aligned}$$

Note that $\deg_x \widetilde{f} < \ell$ and $\deg_y \widetilde{f} < k+1 = n/\ell$.

Our goal, as earlier, is to multiply $f(x)$ and $g(x)$ in the ring $R[x]$. We shall instead multiply $\widetilde{f}(x,y)$ and $\widetilde{g}(x,y)$ and then eventually substitute $y = x^\ell$.

Now comes the key point. Notice that if $\widetilde{h}(x,y) = \widetilde{f}(x,y) \cdot \widetilde{g}(x,y)$, then $\deg_x \widetilde{h} < 2\ell - 1$ and $\deg_y \widetilde{h} < 2k - 1$. These bivariates are polynomials in the ring $R[x,y]$, which can be thought of as polynomials in x with coefficients coming from $R[y]$. But because of these degree bounds, even if the coefficients are thought of as elements in the ring $R_\ell := \frac{R[y]}{\langle y^{2k}+1 \rangle}[x]$, multiplying $\widetilde{f}$ and $\widetilde{g}$ as elements in $R_\ell[x]$ yields the same $\widetilde{h}(x,y)$!

What have we gained by going to a more complicated ring? Note that in R_ℓ, we have $y^{4k} = (y^{2k})^2 = 1$ and hence y is a $4k$-th root of unity. Furthermore, the following claim can also be checked easily:

Claim 10.5. *Let R' be an arbitrary ring with characteristic not equal to 2. If N is a power of 2 and $\omega \in R'$ such that $\omega^N = -1$, then ω is a principal $2N$-th root of unity.*

Hence in the ring R_ℓ, we have that y is a $4k$-th principal root of unity. Thus, if we set up parameters such that $\deg_x \widetilde{f}, \deg_x \widetilde{g} \le 2k$, then we can use FFT (Theorem 10.2) to multiply these two polynomials. We'll choose $\ell = \sqrt{n}/10$ so that $k = 10\sqrt{n}$ and the above condition is satisfied.

Here is the rough sketch:

1. Compute from f and g the coefficient representations of $\widetilde{f}, \widetilde{g} \in R_\ell[x]$. Takes $O(n)$ time.
2. Compute the Fourier transform of $\widetilde{f}$ and $\widetilde{g}$ using y as the principal $4k$-th root of unity. This takes $O(k \log k)$ additions of elements in R_ℓ and $O(k \log k)$ multiplications of elements in R_ℓ by a power of y. Hence, this takes $O(n \log k)$ operations in R as $k, \ell = \Theta(\sqrt{n})$.
3. Once the value representations of $\widetilde{f}$ and $\widetilde{g}$ are obtained, we multiply the values pointwise to get the value representation of $\widetilde{h} = \widetilde{f}\widetilde{g}$. This requires $4k$ multiplications of elements in R_ℓ.
4. Perform the inverse FFT to obtain the coefficient representation of $\widetilde{h}(x,y)$. This again takes $O(n \log k)$ time.
5. Return $h(x) = \widetilde{h}(x, x^\ell)$.

Therefore, we obtain the following recurrence for the time complexity of multiplying two degree n univariate polynomials in $R[x]$:

$$T(n) = 4k \cdot T(\ell) + O(n \log k)$$

where $k = 10\sqrt{n}$ and $\ell = \sqrt{n}/10$. Solving this yields $T(n) = O(n \log n \log\log n)$, completing the proof.

Lecture 14: March 17th, 2017

So far, we have observed that, using "FFT" we could multiply two polynomials while performing only $O(n \log n)$ many ring operations, provided that R contains a primitive n-th root of unity. Moreover, for a general ring (*i.e.*, one

which may not contain such a root, or we do not have access to such a root) we have adjoined an n-th root of unity and performed the operation of multiplication, ending up with $O(n \log n \log\log n)$ many ring operations. Write $\mathrm{MPoly}(n)$ for the number of ring operations to be performed in determining $h = fg$. We have shown that $\mathrm{MPoly}(n) = O(n \log n \log\log n)$. For all the other operations that we will see in this section, we shall express its running time in terms of $\mathrm{MPoly}(n)$ and forget about whether or not the ring was nice.

10.3 *Polynomial division*

Next in line is the following task: *given a polynomial $f \in R[x]$ and a monic polynomial $g \in R[x]$ that satisfy $\max\{\deg f, \deg g\} \leq n$, find polynomials $q, r \in R[x]$ such that $f = qg + r$ and either $r = 0$, or $\deg r < \deg g$.*

We aim to do this by performing $\tilde{O}(\mathrm{MPoly}(n))$ many ring operations. To this end, we make the following observation. Given polynomial $f \in R[x]$, where $f(x) = a_0 + a_1 x + \cdots + a_n x^n$ and $a_n \neq 0$, we write $\mathrm{rev}(f) \in R[x]$ for the polynomial

$$\mathrm{rev}(f)(x) = a_n + a_{n-1}x + \cdots + a_0 x^n$$

Formally, $\mathrm{rev}(f)(x) = x^{\deg f} f(\frac{1}{x})$. We'll drop the (x) and just write $\mathrm{rev}(f)$ for brevity. Suppose that $q, r \in R[x]$ are such that $f = qg + r$, where either $r = 0$, or $\deg r < \deg g$; then the definition of $\mathrm{rev}(f)$ implies

$$\mathrm{rev}(f) = \mathrm{rev}(q)\mathrm{rev}(g) + x^{\deg f - \deg r}\mathrm{rev}(r).$$

Observe that if we work modulo $x^{\deg f - \deg g + 1}$ (in other words, in the quotient ring $R[x]/(x^{\deg f - \deg g + 1})$), we have $\mathrm{rev}(f) = \mathrm{rev}(q)\mathrm{rev}(g) \bmod x^{\deg f - \deg g + 1}$ as $\deg f - \deg r \geq \deg f - \deg g + 1$.

Let $\ell = \deg f - \deg g + 1$. Now for some reason, if we could find an *inverse* of $\mathrm{rev}(g)$, i.e. a polynomial $h \in R[x]$ such that $h \cdot \mathrm{rev} g \equiv 1 \bmod x^\ell$. Then, we can write $\mathrm{rev}(q) = h \cdot \mathrm{rev}(f) \bmod x^\ell$. Since we know that $\deg(q) < \ell$, this allows us to obtain q from $q \bmod x^\ell$ completely in $O(\mathrm{MPoly}(n))$ time.

Computing the inverse of rev(g)

Our task now is to therefore find an inverse of $\mathrm{rev}(g)$. Why should such an inverse even exist? And why did we do all this reverses in the first place? The key point is that because g was monic (that is, its leading coefficient was 1), we now have $\mathrm{rev}(g)$'s constant term to be 1. Fortunately, such polynomials can be inverted* modulo x^ℓ. Keep in mind the example $(1-x)^{-1} = 1 + x + x^2 + \cdots + x^{99} \bmod x^{100}$.

It follows that an efficient algorithm to find $\mathrm{rev}(g)^{-1} \in R[x]/(x^D)$, where $g \in R[x]$ is monic and $D > \deg g$ is an arbitrary positive integer, would lead to an efficient "euclidean algorithm" in $R[x]$. To this end, we find a sequence of polynomials $h_1, \cdots, h_l, \cdots$ in $R[x]/(x^D)$ such that $\mathrm{rev}(g)h_1 \equiv 1 \mod x$, and

$$\mathrm{rev}(g)h_i \equiv 1 \mod x^l \;\Rightarrow\; \mathrm{rev}(g)h_{i+1} \equiv 1 \mod x^{2l} *$$

Notice that $h_1 = 1$ works, since $\mathrm{rev}(g) = 1 + xh$ for some $h \in R[x]$. It now seems

Indeed, it is straightforward to check that, in any commutative ring R (with identity), an element $1 + a \in R$ is unit if $a \in R$ is nilpotent (i.e $a^k = 0$ for some positive integer k).

*-This is a special case of a general principle of Hensel, which specifies the recipie to obtain a factorization of an object, given a factorization of certain reduction of it.

reasonable to try using induction. Having determined $h_i \in R[x]$ such that $gh_i = 0 \bmod x^l$, let us try $h_{i+1} = h_i + ax^l$ for some $a \in R[x]$; writing $\mathrm{rev}(g) \equiv g_0 + g_1 x^l \bmod x^{2l}$ (with $\deg g_0 < l$), one obtains $g_0 h_i \equiv \mathrm{rev}(g)h_i - g_1 h_i x^l \equiv 1 \mod x^l$, and hence

$$\begin{aligned} \mathrm{rev}(g)h_{i+1} &\equiv (g_0 + g_1 x^l)(h_i + ax^l) \mod x^{2l} \\ &\equiv g_0 h_i + x^l(g_1 h_i + a g_0) \mod x^{2l} \\ &\equiv 1 + x^l(c + a g_0 + g_1 h_i) \mod x^{2l} \qquad (*) \end{aligned}$$

for some $c \in R[x]$. Finally, from $(*)$, it follows that if $a \in R[x]$ is such that we have $c + ag_0 + g_1 h_l \equiv 0 \mod x^l$, then $h_{i+1} := h_i + ax^l$ would satisfy the required property. Letting $a := -h_i(g_1 h_i + c)$, we get

$$c + ag_0 + g_1 h_i \equiv c - g_0 h_i (g_1 h_i + c) + g_1 h_i \equiv 0 \mod x^l$$

which shows that $h_{i+1} := h_i(1 - g_1 h_i x^l - cx^l)$ has the required property. Notice that, for each integer $i > 1$, finding $h_i \in R[x]/(x^D)$ requires $\mathrm{MPoly}(D)$-order ring operations to be performed. And $h_i = h_{i+1} = \cdots$ if $i > \log D$.

Going back to our "Euclidean algorithm" in $R[x]$, notice that Hensel lifting techniques allow us to determine $q \in R[x]$ in $\mathrm{MPoly}(n)$, as needed. Once we have that, we can determine r as well.

10.4 *Multi-point evaluations*

Our next task is the following: Given polynomial $f \in R[x]$ with $\deg f \le n$, and points $\alpha_1, \cdots, \alpha_n$, compute $f(\alpha_1), \cdots, f(\alpha_n)$.

Here is the strategy. Suppose we can find polynomials f_1 and f_2 of degree at most $n/2$ such that

$$f(\alpha_i) = \begin{cases} f_1(\alpha_i) & \text{if } i \le n/2 \\ f_2(\alpha_i) & \text{if } i > n/2 \end{cases},$$

then we can recursively call the multipoint evaluations on these two polynomials. A natural choice for f_1 and f_2 are just

$f_1 \equiv f \mod \prod_{i \le n/2}(x - \alpha_i)$ and $f_2 \equiv f \mod \prod_{i > n/2}(x - \alpha_i)$ and clearly we have $\deg f_1, \deg f_2 \le n/2$, we can apply recursion to solve the problem if we can compute f_1 and f_2 efficiently. For that, we need to first compute $\prod_{i \le n/2}(x - \alpha_i)$ and $\prod_{i > n/2}(x - \alpha_i)$ (and these polynomials then yield the polynomials f_1, f_2 as respective remainders of f). It is easy to see that grouping the factors into pairs and successively multiplying them in a binary-tree-like fashion takes time $O(\mathrm{MPoly}(n) \log n)$. Therefore, the total time for the recursive algorithm is

$$T(n) = 2T(n/2) + O(\mathrm{MPoly}(n) \log n) = O(\mathrm{MPoly}(n) \log^2 n).$$

11 Factorizing univariate polynomials over finite fields

From now on, we shall be looking at the task of factorizing univariate polynomials over finite fields.

Remark 11.1. *The input would be a polynomial $f \in \mathbb{F}_q[x]$ where say $q = p^r$. How is the field given as input? We are assuming that the field $\mathbb{F}_q$ is provided via $\mathbb{F}_q = \frac{\mathbb{F}_p[y]}{\langle g(y) \rangle}$ where g is an irreducible polynomial in $\mathbb{F}_p[y]$ of degree r. So, we shall always be assuming that the irreducible polynomial $g(y) \in \mathbb{F}_p[y]$ is provided to us.*

Once this is provided, it is easy to check that all operations in $\mathbb{F}_q$ can be performed in $\mathrm{poly}(\log q)$ *time.* ◊

We will need a few preliminaries operations first and we address them first.

11.1 *Computing the GCD*

Next we look at the GCD computation. Suppose that $\mathbb{F}$ is a finite field, and $f, g \in \mathbb{F}[x]$ are polynomials of degree bound n. We start with the following elementary lemma.

Lemma 11.2. $\gcd(f,g) = d$ *if and only if there are $p_1, p_2 \in \mathbb{F}[x]$ such that $d = p_1 f + p_2 g$. Furthermore, given $f, g \in \mathbb{F}[x]$ with* $\deg f, \deg g \leq n$, *we can find* $\gcd(f,g)$ *in* $\mathrm{poly}(n)$ *many $\mathbb{F}$-operations.*

Proof. Recall that $\gcd(f,g) = \gcd(g,r)$, where $f = qg + r_1$ with $r_1 = 0$ or $\deg r_1 < \deg g$. In fact, if $d \mid f, g$, then $d \mid r_1$ since $r_1 = f - qg$; this implies that $\gcd(f,g) \mid g, r_1$ and therefore, $\gcd(f,g) \leq \gcd(g, r_1)$; on the other hand, if $d \mid r_1, g$, then $d \mid f$ too, and hence, $\gcd(g, r_1) \leq \gcd(f,g)$.

Algorithm 24: GCD

Input : polynomial $f, g \in \mathbb{F}[x]$ of degree at most n
Output: $\gcd(f,g)$

```
1 if g = 0 then
      // Base case
2     return f
3 else
4     Compute r = f mod g.
5     Return GCD(g, r).
```

Since the degrees of the recursive calls are dropping, the algorithm terminates in $\mathrm{poly}(n)$ steps. □

In fact, the algorithm even gives the following. We'll leave it to the reader to figure out why this is the case.

Lemma 11.3. *For any $f, g \in \mathbb{F}_q[x]$, there exists polynomials $a, b \in \mathbb{F}_q[x]$ such that*

$$af + bg = \gcd(f,g).$$

Furthermore, these polynomials a and b can be found in $\mathrm{poly}(n, \log q)$ *time where* $\deg f, \deg g \leq n$.

11.2 *Handling repeated factors*

Say $f = g_1^{e_1} \cdots g_r^{e_r}$ is the factorization of g into irreducibles, where each $e_i \geq 1$, the *square-free* part of f, denoted by SquareFree f, is given by SquareFree $f = g_1 \cdots g_r$. Given an n-variate polynomial f, how do we compute the square-free part of f? The idea is to use derivatives.

Define the linear map $D : \mathbb{F}_q[x] \to \mathbb{F}_q[x]$ via $D(x^i) = ix^{i-1}$ for all monomials, and extending linearly. The following properties are easy to verify.

1. D is a linear operator. That is, $D(af + bg) = aD(f) + bD(g)$ if $a, b \in \mathbb{F}_q$.
2. $D(fg) = f \cdot D(g) + g \cdot D(f)$.
3. $D(f^e) = ef^{e-1}D(f)$.

Lemma 11.4. *Let $f \in \mathbb{F}_q[x]$ and suppose g is an irreducible polynomial such that $g^e \mid f$. Then, $g^{e-1} \mid D(f)$.*

Proof. Say $f = g^e h$, we have $D(f) = eg^{e-1}hD(g) + g^e D(h)$. □

Therefore, a natural way to remove repeated factors is to repeatedly compute $f/\gcd(f, D(f))$. The only issue though is that it may be the case that $D(f) = 0$. It is a simple exercise to see why this might happen over a finite field, and what we can do about it.

11.3 *Some field properties, and Distinct Degree Factorization (DDF)*

The first step in most factorization procedures is to collect all degree 1 factors, degree 2 factors etc. and then factorize each such block separately. We would need a properties about fields before we can get there.

Lemma 11.5. *$x^{p^r} - x \mid x^{p^s} - x$ if and only if $r \mid s$.*

Proof. Suppose $r \mid s$; then $s = rt$ for some integer $t > 0$; this implies $p^s - 1 = p^{rt} - 1 = (p^r - 1)(p^{rt-r} + \cdots + 1) = b(p^r - 1)$, say. Hence,

$$x^{p^s} - x = x(x^{p^s-1} - 1) = x(x^{b(p^r-1)} - 1) = x(x^{p^r-1} - 1)a(x)$$

for some polynomial $a(x)$. Conversely, suppose that $r \nmid s$. By Euclidean property of the integers, there exist $a, b \in \mathbb{N}$ such that $s = ar + b$ with $0 < b < r$. Now

$$p^s - 1 = p^{ar+b} - 1 = p^b(p^{ar} - 1) + (p^b - 1) \equiv p^b - 1 \mod p^r - 1$$

implies that $p^r - 1 \nmid p^s - 1$. Let $z = p^r - 1$; again, there are integers $0 < q_z, r_z$ such that $p^s - 1 = q_z z + r_z$ and $r_z < z$. We now have

$$\begin{aligned} x^{p^s-1} - 1 &= x^{q_z z + r_z} - 1 \\ &= x^{r_z}(x^{q_z z} - 1) + (x^{r_z} - 1) \equiv x^{r_z} - 1 \mod x^z - 1 \end{aligned}$$

and since $r_z < z$, this shows that $x^{p^s-1} - 1 \not\equiv 0 \mod x^{p^r-1} - 1$. □

Corollary 11.6. *$\mathbb{F}_{p^r} \subseteq \mathbb{F}_{p^s}$ if and only if $r \mid s$.*

Lemma 11.7.

$$x^{p^s} - x = \prod_{\substack{\mu \text{ irred. over } \mathbb{F}_p \\ \deg(\mu) | s}} \mu(x)$$

Proof. Observe that $x^{p^s} - x = \prod_{\alpha \in \mathbb{F}_{p^s}} (x - \alpha)$. Every such $\alpha \in \mathbb{F}_{p^s}$ satisfies some monic irreducible polynomial of degree $r \mid s$ (as $\mathbb{F}_p(\alpha)$ is a subfield of $\mathbb{F}_{p^s}$). Therefore, each $(x - \alpha)$ must divide some $\mu_r(x)$ on the right and hence LHS divides the RHS.

For the reverse divisibility, it suffices to show that every monic irreducible polynomial $\mu(x) \in \mathbb{F}_p[x]$, with $\deg \mu \mid s$, divides $x^{p^s} - 1$. Let $\mu(x) \in \mathbb{F}_p[x]$ be a monic irreducible polynomial with $\deg \mu \mid s$. Then the field $\mathbb{F}_p[x]/(\mu) \cong \mathbb{F}_{p^{\deg \mu}}$ is contained in $\mathbb{F}_{p^s}$. This implies that there is some $\alpha \in \mathbb{F}_{p^s}$ that is a root of $\mu(x)$ and hence $(x - \alpha) \mid \gcd(\mu(x), x^{p^s} - x)$. In particular, the polynomial $d(x) := \gcd(\mu(x), x^{p^s} - x) \in \mathbb{F}_p[x]$ has positive degree. Since $\mu(x) \in \mathbb{F}_p[x]$ is irreducible this forces $d(x) = \mu(x)$. Hence $\mu(x) \mid x^{p^s} - x$ in $\mathbb{F}_p[x]$. □

Distinct Degree Factorization

Given $f \in \mathbb{F}_q[x]$, can we find $f_1, \cdots, f_n$ such that $f = f_1 \cdots f_n$ and f_i is just the product of all irreducible factors of f of degree exactly i? We will show that this can be done in $\text{poly}(n)$ field operations, and hence in $\text{poly}(n, \log q)$ time.

Let us assume that $f \in \mathbb{F}_q[x]$ has no repeated factor. It is immediate from the last lemma that $f_1 = \gcd(x^q - x, f)$. In general,

$$\gcd(x^{q^i} - x, f) = \prod_{j | i} f_j.$$

Therefore, our task will be accomplished if we can efficiently compute $\gcd(x^{q^i} - x, g_i(x))$. To this end, we observe that

$$x^{q^i} \bmod f \equiv (x^{q^{i-1}} \bmod f)^q \bmod f.$$

Therefore, if we could efficiently compute $x^{q^{i-1}} \bmod f$, then exponentiation by squaring would produce $x^{q^i} \bmod f$ efficiently.

Algorithm 25: Distinct Degree Factorization

Input : polynomial $f \in \mathbb{F}[x]$ of degree at most n with no repeated factors.
Output: $f = f_1 \cdots f_n$ where f_i is the product of irreducible factors of f of degree i.

```
g_0 = x = x^{q^0} mod f
for i = 1, ..., n do
    // To obtain g_i = x^{q^i} mod f
    Compute g_{i-1}^{2^i} mod f for every i = 0, ..., log q using repeated squaring,
      and thus compute g_i = g_{i-1}^{q} mod f.
    Set f_i = gcd(f, g_i - x).
    Set f = f / f_i.
return {f_1, ..., f_n}.
```

The factorization algorithms

Lecture 15: March 20th, 2017

We now move on to the actual task of factorizing univariate polynomials. In this section, we shall be looking at two algorithms for factorizing univariate polynomials over finite fields:

- Cantor-Zassenhaus
  A randomized algorithm that factors an n degree univariate polynomial over $\mathbb{F}_q$ ($q = p^r$) in time polynomial in n and $\log q$.
- Berlekamp
  A deterministic algorithm that factors an n degree univariate polynomial over $\mathbb{F}_q$ ($q = p^r$) in time polynomial in n, p and $\log q$.

Both of these algorithms use the Chinese Remainder Theorem, so called because the earliest known statement of such a theorem is by Sunzi in his 3rd century book Sunzi Suanjing (The Mathematical Classic of Master Sun):

> There are certain things whose number is unknown. If we count them by threes, we have two left over; by fives, we have three left over; and by sevens, two are left over. How many things are there?

11.4 *The Chinese Remainder Theorem*

The classic Chinese Remainder Theorem for integers says that if a set of numbers $\{n_1, n_2, \ldots, n_r\}$ are pairwise co-prime, then for any setting of the moduli $\{a_1, a_2, \ldots, a_r\}$, there is a unique solution in $\{0, \ldots, \prod n_i - 1]\}$ to the set of constraints $x_i = a_i \bmod n_i$. In fact the map $x \mapsto (x \bmod n_1, \ldots, x \bmod n_r)$ is an isomorphism between $\frac{\mathbb{Z}}{\langle \prod n_i \rangle}$ and $\frac{\mathbb{Z}}{\langle n_1 \rangle} \times \cdots \times \frac{\mathbb{Z}}{\langle n_r \rangle}$.

For rings, we have the following setup: Let R be a commutative ring with an identity element. We say ideals I, J are *coprime* if $\exists a \in I, b \in J$ st $a + b = 1$. We have the following Chinese Remainder Theorem:

Theorem 11.8. *If a set of ideals $\{I_1, I_2, \ldots, I_r\}$ that are pairwise coprime, then*

$$\frac{R}{\cap I_i} \cong \frac{R}{I_1} \times \cdots \times \frac{R}{I_r}$$

with the map $\Phi : a \mapsto (a \mod I_1, \ldots, a \mod I_r)$.

We proved a weaker statement in class though.

Proof. It is easy to see that $\Phi' : R \to \frac{R}{I_1} \times \cdots \times \frac{R}{I_r}$ via $a \mapsto (a \bmod I_1, \cdots, a \bmod I_r)$ is a homomorphism, so we will not explicitly show it here. It suffices to show that Φ' is surjective as it is clear that the kernel is precisely $\bigcap I_j$.

We begin with the case when $r = 2$. Since I_1 and I_2 are coprime, $\exists a \in I_1, b \in I_2$ such that $a + b = 1$. Note that $a \mod I_1 = b \mod I_2 = 0$ and hence $\Phi'(a) = (0,1), \Phi'(b) = (1,0)$. Therefore, $\Phi'(r_1 a + r_2 b) = (r_1, r_2)$ and so Φ' is surjective.

For the general case, it suffices to show that there is some element in $a \in R$ that $a \bmod I_1 = 1$ but $a \bmod I_j = 0$ for all $j \neq 1$. For each $j \neq 1$, since I_1 and I_j are coprime, we have elements $a_j \in I_1$ and $b_j \in I_j$ such that $a_j + b_j = 1$. Then,

$$\begin{aligned} 1 &= \prod_{j=2}^{r} (a_j + b_j) \\ &= b_2 \cdots b_r + (\text{sum of terms involving at least one } a_j) \\ &= b_2 \cdots b_r \bmod I_1 \end{aligned}$$

Hence if $a = \prod_{j=2}^{r} b_j$, then this satisfies our requirements. □

Relevance to Factoring

Let $R = \mathbb{F}_q[x]$. Say we have a polynomial $f \in R$ that we want to factorize. The factorization of f into irreducible polynomials is, say, $f = g_1 g_2 \cdots g_r$. *If f does not have repeated roots*, then using the Chinese Remainder Theorem, we can write any element in $\frac{R}{\langle f \rangle}$ as an element in $\frac{R}{g_1} \times \cdots \times \frac{R}{g_r}$. Let us call this representation the Chinese representation. We make the following important observation:

Let $h \in \frac{R}{\langle f \rangle}$. Note that h has a 0 in coordinate i of its Chinese representation iff h is a multiple of g_i. So $\gcd(f, h)$ is non-trivial iff $h (\neq 0)$ has a 0 in its Chinese representation. We call such an h a non-trivial zero divisor since such elements and only such elements (apart from 0) can be multiplied with a non-zero element to get zero.

11.5 *The Cantor-Zassenhaus Algorithm*

The Cantor-Zassenhaus algorithm is a randomized algorithm that factorizes a univariate polynomial $f \in \mathbb{F}_q[x]$ ($q = p^r$) in time $\text{poly}(n, \log q)$.

The algorithm starts with the following setup: We use the tricks we've seen to reduce the polynomial that we want to factor into a polynomial with no repeated roots, and then do distinct degree factorization. Now we're left with polynomials, each of which is such that all its irreducible factors are of the same degree. We now proceed to factor one of these polynomials, let's call it f. Let

$f = g_1 g_2 \cdots g_r$, with the degree of each g_i being t. By Theorem 11.8,

$$\frac{R}{\langle f \rangle} \cong \frac{R}{\langle g_1 \rangle} \times \cdots \times \frac{R}{\langle g_r \rangle} \cong \mathbb{F}_{q^t} \times \cdots \times \mathbb{F}_{q^t},$$

since each g_i is irreducible and of degree t.

We first deal with the case where q is odd: Let $h \in \frac{R}{\langle f \rangle}$. Let $(h_1, \ldots, h_r)$ be the Chinese representation of h. It's clear from the Chinese representation that $h^{q^t} = h$. If h is not a zero divisor, $h^{q^t-1} = 1$, which in Chinese is $(1, 1, \ldots, 1)$. Therefore $h^{\frac{q^t-1}{2}}$ has each component either 1 or -1. If it is not all 1 or all -1, then $h^{\frac{q^t-1}{2}} - 1$ will be a non-trivial zero divisor. To find such an h, we have the following claim:

Claim 11.9. *Let h be an element chosen uniformly at random from the elements that are not zero divisors. Let $h' = h^{\frac{q^t-1}{2}}$. Then $\forall i \ \Pr[h'_i = 1] = \frac{1}{2}$.*

Proof. Recall that $\mathbb{F}_{q^t}^{\times}$ is a cyclic group. Let η be its generator. If h_i is an even power of a, then h'_i is a power of $\eta^{q^t-1} = 1$. If h_i is not an even power of η, then h'_i is not a power of 1 and hence must be -1. □

We complete the odd q case by compiling the above into Algorithm 26. Since zero divisors are exactly those which have a non-constant gcd with f, our claim above states that the repeat loop runs at most twice in expectation. Hence the total number of times that the repeat loop runs across all recursive calls is less than $2n/t$ in expectation. Within the loop, computing $\gcd(f, h')$ is computed by computing $\gcd(f, h' \mod f)$, where $h' \mod f$ is computed by repeated squaring and taking moduli. h' itself is never computed. Hence the whole algorithm runs in polynomial time in n and $\log q$.

Algorithm 26: UnidegFactors when the field is of odd size

Input : a square-free polynomial $f \in \mathbb{F}_q[x]$ (of degree n), t
Output: The factorization of f, assuming all its irreducible factors have degree t

if $\deg(f) = t$ **then**
  return $\{f\}$
repeat
  Sample a non-zero polynomial h of degree $\leq n-1$ uniformly at random.
  if $\gcd(f,h)$ *is non-constant* **then**
    return $\textsc{UnidegFactors}(\gcd(f,h),t) \cup \textsc{UnidegFactors}(f/\gcd(f,h),t)$
  Let $h' = h^{\frac{q^t-1}{2}} - 1$
  if $\gcd(f,h')$ *is non-constant* **then**
    return $\textsc{UnidegFactors}(\gcd(f,h'),t) \cup \textsc{UnidegFactors}(f/\gcd(f,h'),t)$
until *Forever*

In the case where q is even, we can no longer talk of $h^{\frac{q^t-1}{2}}$. So let $q = 2^r$. We look at an operation called trace: $\mathsf{Tr} : \mathbb{F}_{q^t} \to \mathbb{F}_2$, is defined as $\mathsf{Tr}(x) = x + x^2 + x^{2^2} + \cdots + x^{2^{rt-1}}$. Just from this definition, it isn't clear as to why $\mathsf{Tr}(x) \in \mathbb{F}_2$ but turns out it is. Trace is in fact a linear map from a bigger field such as $\mathbb{F}_{q^t}$ to a subfield such as $\mathbb{F}_2$ and can be easily seen to satisfy$\mathsf{Tr}(x)(\mathsf{Tr}(x)+1) = x^{2^{rt}} + x$. Since $x^{q^t} + x$ has every element of $\mathbb{F}_{q^t}$ as roots and $\mathsf{Tr}(x)$ and $\mathsf{Tr}(x)+1$ are both $q^t/2$ degree polynomials, both $\mathsf{Tr}(x)$ and $\mathsf{Tr}(x)+1$ must each have half of $\mathbb{F}_{q^t}$ as their roots. This preemptively proves the following claim:

Claim 11.10. *Let h be an element chosen uniformly at random from $\leq \deg(f) - 1$ degree polynomials. Let $h' = h + h^2 + h^{2^2} + \cdots + h^{2^{rt-1}}$. Then $\forall i \ \Pr[h'_i = 0] = \frac{1}{2}$.*

Again it follows that the probability that h' is a non-trivial zero divisor is at least $\frac{1}{2}$. Note that $h' \bmod f$ and hence $\gcd(h', f)$ can be computed in $\mathrm{poly}(n, \log q)$ time using repeated squaring tricks. This algorithm for factoring over an even sized field is summarized in Algorithm 27.

Algorithm 27: UnidegFactors when the field is of even size

Input : a square-free polynomial $f \in \mathbb{F}_q[x]$ (of degree n), t
Output: The factorization of f, assuming all its irreducible factors have degree t

if $\deg(f) = t$ **then**
  return $\{f\}$
repeat
  Sample a non-zero polynomial h of degree $\leq n-1$ uniformly at random.
  Let h' be $h + h^2 + h^{2^2} + \cdots + h^{2^{rt-1}}$
  if $\gcd(f, h')$ *is non-constant* **then**
    return UnidegFactors$(\gcd(f,h'),t) \cup$ UnidegFactors$(f/\gcd(f,h'),t)$
until *Forever*

For completeness, the overall algorithm is given in Algorithm 28.

Algorithm 28: RandUnivariateFactoring, Cantor-Zassenhaus

Input : $f \in \mathbb{F}_q[x]$ (of degree n)
Output: The factorization of f

`// Remove repeated roots`
Compute $f' = \text{SquareFree}\, f$.
`// Factorize` f' `based on the degree of its irreducibles`
$f_1, f_2, \cdots, f_n =$ DistinctDegreeFactorize(f')
Factors $\leftarrow$ RandUnivariateFactoring(f'')
for $i \in [n]$ **do**
  Append UnidegFactors(f_i, i) to Factors
`// For each factor, one can also find their multiplicity by repeatedly dividing` f `with them`
return Factors

11.6 *Berlekamp's Algorithm*

Berlekamp's algorithm deterministically factorizes a univariate polynomial $f \in \mathbb{F}_q[x]$ (where $q = p^r$) in time $\text{poly}(n, p, \log q)$.

The crux of the algorithm is the following observation:

Claim 11.11. *Given $f, h \in \mathbb{F}_q[x]$ with $1 \leq \deg(h) < \deg(f)$ such that $f \mid h^p - h$, we can find a non-trivial factor of f in poly$(n, p, \log q)$ time.*

Proof. As we've seen before, $h^p - h$ factors as $\prod_{\alpha \in \mathbb{F}_p}(h - \alpha)$. All the factors of f are factors of $h^p - h$. Hence all the factors of f are contained in the factors of the $(h - \alpha)$s. Since $\deg(h - \alpha) < \deg(f)$, by going through all $\gcd(f, h - \alpha)$, we get a decomposition of f into non-trivial factors. □

To see why a non-constant h ought to exist when f is reducible *and does not have repeated roots* let us ask ourselves what is the Chinese representation of any h that satisfies $h^p - h \equiv 0 \bmod f$? If the Chinese representation of h was $(h_1, \ldots, h_r)$, then the Chinese representation of $h^p - h$ is just $(h_1^p - h_1, \ldots, h_r^p - h_r)$ and this would be zero if and only if each $h_i^p - h_i = 0$. In other words, each $h_i \in \mathbb{F}_p$. Thus, any element h whose chinese representation is in $\mathbb{F}_p^r$ is a polynomial such that $h^p - h = 0$. This would be non-constant if not all the coordinates are the same. Such an h is of course present as long as the chinese representation has at least two coordinates which just means that f is reducible.

Now comes the task of finding such a h. Consider the following map $T : \frac{\mathbb{F}_q[x]}{\langle f \rangle} \to \frac{\mathbb{F}_q[x]}{\langle f \rangle}$ be defined as $T(h) = h^p - h$. We are a looking for a non-constant polynomial $h \in \ker(T)$.

An observation here is that T satisfies $T(h+g) = T(h) + T(g)$. Hence it is a linear operator, ... or is it? Unfortunately, if $\alpha \in F_q$, then $T(\alpha h) = \alpha^p h^p - \alpha h$ which doesn't seem to be equal to $\alpha T(h)$. However, if $\alpha \in \mathbb{F}_p$, then of course $T(\alpha h) = \alpha T(h)$. Hence, T is a linear operator over $\mathbb{F}_p$. So let us write $\frac{\mathbb{F}_q}{\langle f \rangle}$ as a vector space over $\mathbb{F}_p$ and then look at the map T.

Our field $\mathbb{F}_q = \frac{\mathbb{F}_p[y]}{\langle g \rangle}$ where g is some irreducible polynomial of degree r (Remark 11.1). Thus, elements of $a \in \mathbb{F}_q$ are of the form $a^{(0)} + a^{(1)} y + \cdots + a^{(r-1)} y^{r-1}$ where each $a^{(i)} \in \mathbb{F}_p$. We shall look at $\frac{\mathbb{F}_q[x]}{\langle f \rangle}$ as a $\mathbb{F}_p$-vector space with basis

$$\left\{ x^i y^j \ : \ 0 \le i \le n-1 \ , \ 0 \le j \le r-1 \right\}.$$

by writing any $h \in \frac{\mathbb{F}_q[x]}{\langle f \rangle}$ as

$$h = \sum_{i=0}^{n-1} \sum_{j=0}^{r-1} h_i^{(j)} x^i y^j \quad , \quad \text{where each } h_i^{(j)} \in \mathbb{F}_p.$$

Thus, the operator T can be expressed as a matrix in $\mathbb{F}_p^{nr \times nr}$, and all this takes $\text{poly}(n, \log q)$ time. What we want is the kernel of this matrix and computing a basis for its kernel just takes a Gaussian elimination[4], so in $\text{poly}(n, \log q)$ time. If f is square-free and reducible, the basis must include a non-constant polynomial; we take one of them as our h.

The complete algorithm is given in Algorithm 29.

[4] How this is done is nicely explained over here.

Algorithm 29: DetUnivariateFactoring, Berlekamp

Input : $f \in \mathbb{F}_q[x]$ (of degree n)
Output: The factorization of f

```
// Remove repeated roots
```
$f' = f/\gcd(f, D(f))$
Let $T : x \mapsto x^p - x \mod f'$
Compute $T(x^i y^j)$ to get the matrix representation for T as a linear operator over $\mathbb{F}_p$
Compute a basis for the kernel of T via Gaussian Elimination
if *there is a non-constant polynomial h in the basis* **then**
  for $\alpha \in \mathbb{F}_p$ **do**
    if $\gcd(f', h - \alpha)$ *is non-constant* **then**
      return DetUnivariateFactoring$(\gcd(f', h - \alpha)) \cup$ DetUnivariateFactoring$(f/\gcd(f', h - \alpha))$

```
// If we ever get here, then f' must be irreducible.
// One can find multiplicity by repeatedly dividing f by f'.
```
return $\{f'\}$

12 Factorizing Bivariate Polynomials over Finite Fields

Lecture 16: March 24th, 2016

Now, we want to look at the task of factorizing bivariate polynomials over finite fields. So given $f \in \mathbb{F}_q[x, y]$ with $\deg_x(f), \deg_y(f) \leq n$, we want to find its factors in randomized $\text{poly}(n, \log q)$ time.

We first give an outline of what we want the algorithm to look like and then see how we can do each of the steps. Let the input be $f \in \mathbb{F}_q[x, y]$.

1. Pre-processing: We pre-process f so that we may assume, without loss of generality, that f satisfying the following:
   (a) $f \in \mathbb{F}[x, y] \equiv (\mathbb{F}[y])[x]$ is monic in x
   (b) $f(x, y)$ is square-free.
   (c) $f(x, 0)$ is square-free

   What we shall do is change f to a different $\tilde{f}$ that satisfies these properties, such that getting the factors of $\tilde{f}$ is as good as getting the factors of f.
2. Factorise $f(x, 0)$, that is, find g_0, h_0 such that $f = g_0 h_0 \mod \langle y \rangle$. If $f(x, 0)$ is irreducible, then output Irreducible.

3. Use Hensel Lifting to find g_i, h_i such that

$$\begin{aligned}
f &= g_1 \cdot h_1 \mod \langle y^2 \rangle \\
f &= g_2 \cdot h_2 \mod \left\langle y^{2^2} \right\rangle \\
&\vdots \\
f &= g_i \cdot h_i \mod \left\langle y^{2^i} \right\rangle \\
&\vdots \\
f &= g_k \cdot h_k \mod \left\langle y^{2^k} \right\rangle \qquad \text{till } 2^k > 2n^2
\end{aligned}$$

4. Recover if possible a factor of f from g_k. In the recovery step, we solve for $\tilde{f}$ and $\tilde{\ell}$ in

$$\tilde{f} = g_k \cdot \tilde{\ell} \mod \left\langle y^{2^k} \right\rangle$$

such that $\deg_x(\tilde{f}) < n$, $\deg_y(\tilde{f}) \leq n$. If a solution $(\tilde{f}, \tilde{\ell})$ exists, then any such $\tilde{f}$ would have a non-trivial gcd with f. If no such solution exists, output Irreducible.

Before going in to the details of the algorithm, we look at some prerequisites.

12.1 *Hensel Lifting*

Hensel lifting is a way of getting "better and better" approximate factorizations.

Theorem 12.1 (Hensel Lifting)**.** *Let R be a commutative ring R and let I be an ideal in R. Suppose have elements $f, g, h \in R$ such that:*

$$\begin{aligned}
f &= g \cdot h \bmod I \\
\exists a, b \;:\; ag + bh &= 1 \bmod I.
\end{aligned}$$

Then, we have

- (Better factorization) *There are elements $\tilde{g}$ and $\tilde{h}$ such that*

$$f = \tilde{g} \cdot \tilde{h} \bmod I^2,$$

for which,

- (Lifts of previous solutions)

$$\begin{aligned}
\tilde{g} &= g \bmod I \\
\tilde{h} &= h \bmod I.
\end{aligned}$$

- (Uniqueness) *Furthemore, any other g', h' satisfying the above properties, there must be a $u \in I$ such that $f' = \tilde{f} \cdot (1 + u)$ and $g' = \tilde{g} \cdot (1 - u)$.*

Proof. Suppose, $f = gh + q$ where $q \in I$ and $ag + bh = 1 + r$ where $r \in I$. Then, for $g_1 = bq$, $h_1 = aq \in I$; if we take $\tilde{g} = g + g_1$ and $\tilde{h} = h + h_1$, we have

- $\tilde{g} = g \bmod I$
- $\tilde{h} = h \bmod I$

Further,

$$\begin{aligned}\tilde{g}\tilde{h} &= gh + gh_1 + hg_1 + h_1g_1 \\ &= f - q + gh_1 + hg_1 + h_!g_1 \\ &= f - q + q(1 + r) + h_1g_1 \\ &= f \bmod I^2\end{aligned}$$

and

$$\begin{aligned}a\tilde{g} + b\tilde{h} &= ag + bh + ag_1 + bh_1 && \text{where } ag + bh = 1 \bmod I \text{ and } ag_1 + bh_1 \in I \\ &= 1 + r'' && \text{for some } r'' \in I\end{aligned}$$

Now if we take $\tilde{a} = a(1 - r'')$ and $\tilde{b} = b(1 - r'')$, we get

$$\tilde{a}\tilde{g} + \tilde{b}\tilde{h} = (1 - r'')(a\tilde{g} + b\tilde{h}) = (1 - r''^2) = 1 \bmod I^2$$

Finally one can also check that if (g', h') is another solution, then $g' = \tilde{g}(1 + u)$ and $h' = \tilde{h}(1 - u)$ for some $u \in I$. We leave this as an easy exercise. □

Everything in the above theorem is effective — it gives a procedure to get a better approximation.

In our setting, $I = \langle y \rangle$ and $R = \mathbb{F}_q[x, y]$. Then, given a square-free factorization $f = g_0h_0 \bmod \langle y \rangle$, we can lift this to a factorization $f = g_1h_1 \bmod \langle y^2 \rangle$. Even g_1 and h_1 satisfy the hypothesis of the Hensel Lifting and hence can lift it again.

12.2 *Bivariates as univariates over the fraction field*

We'll soon be looking at computing gcds of bivariate polynomials. In the univariate case, we used Euclid's algorithm, which is based on the fact that for $f, g \in \mathbb{F}[x]$, $\gcd(f, g) = \gcd(g, f \bmod g)$. How are we to perform Euclid's algorithm over rings such as $\mathbb{F}_q[x, y]$?

The idea is to view the elements of $\mathbb{F}[x, y]$ as elements of $(\mathbb{F}[y])[x]$. That is, view each bivariate polynomial as a univariate polynomial in x with coefficients being polynomials in y. However, to apply Euclid's algorithm, we would have to divide the coefficients etc. Hence, we will view these as elements in $(\mathbb{F}(y))[x]$ instead of the more usual $(\mathbb{F}[y])[x]$ – univariate polynomials in x with coefficients being rational function entries in y.

However, there are still two things we need to check:

1. Why is $\gcd(f, g)$ in $\mathbb{F}(y)[x]$ same as that in $\mathbb{F}[x, y]$? Is divisibility in $\mathbb{F}(y)[x]$ the same as divisibility in $\mathbb{F}[x][y]$?
2. How does one compute the gcd of of bivariates?

The second question will be part of the problem set. However the first question can be answered using:

Theorem 12.2 (Gauss' Lemma)**.** *Let R be a UFD with field of fractions F and let $f(x) \in R[x]$. If f is reducible over $F[x]$ then it is reducible over $R[x]$.*

The content of a polynomial is the gcd of the coefficients of the polynomial

Proof. We prove a special case for $R = \mathbb{Z}$ and $F = \mathbb{Q}$. So let $f \in \mathbb{Z}[x]$ and without loss of generality, let the content of f be 1. Now, if for $g, h \in \mathbb{Q}[x]$, we can clear all the denominators, and remove common factors to write

$$f = g(x) \cdot h(x) = \frac{m}{n} \cdot G(x)H(x)$$

where $m, n \in \mathbb{Z}$, $G, H \in \mathbb{Z}[x]$ and the contents of G and H are both 1. Thus,

$$n \cdot f = m \cdot G(x)H(x)$$

with the content of the left hand side being n. Is it possible that the content of $G(x)H(x)$ is not 1? Suppose not. Then, there is a prime p which divides the content of $G(x)H(x)$. Since the contents of G and H are both 1, there must be some coefficient of G and some coefficient of H which are not divisible by p. Let i, j be the highest degree of x in G, H respectively for which the coefficients are not divisible by p. Then it is easy to see the the coefficient of x^{i+j} in $G(x)H(x)$ is not divisible by p, which is a contradiction.

Thus, the content of $G(x)H(x)$ is 1 and hence $n = \pm m$ which gives us a factorisation of f in $\mathbb{Z}[x]$. □

In our case, $R = \mathbb{F}[y]$ and $F = \mathbb{F}(y)$ and so Gauss' Lemma gives us that if f is reducible over $\mathbb{F}(y)[x]$, then it is reducible over $\mathbb{F}[y][x]$. Further the proof ensures that the gcd in $\mathbb{F}[x, y]$ and that in $(\mathbb{F}(y))[x]$ differs only by a unit in $\mathbb{F}(y)$ which can be found.

12.3 *The Resultant*

Let R be a UFD (in our case, $R = \mathbb{F}_q[y]$) and let $F = \mathrm{Frac}(R)$ (which in our case is $\mathbb{F}_q(y)$).

Suppose $f, g \in R[x]$ with $\deg(f) = d_1$ and $\deg(g) = d_2$. Consider the linear map $T : F^{d_2}(\equiv F[x]_{\deg \le d_2}) \times F^{d_1}(\equiv F[x]_{\deg \le d_1}) \to F^{d_1 \times d_2}(\equiv F[x]_{\deg \le d_1 + d_2})$ defined by $T(a, b) = af + bg$.

Claim 12.3. *T has a non-trivial kernel if and only if f and g have a non-constant (i.e., not an element of F) common factor.*

Proof. Firstly, if $\gcd(f, g) \neq 1$, then $f = f'\ell$ and $g = g'\ell$ for some f' and g' with $0 < \deg(f') < d_1$, $0 < \deg(g') < d_2$. Thus, $a = -g'$, $b = f'$ is a non-trivial solution to $T(a, b) = 0$ proving that T has a non-trivial kernel.

Conversely, suppose $\gcd(f, g) = 1$. Then, $\exists a, b \in F[x]$ s.t. $af + bg = 1$ (recall that $R[x]$ is also a UFD). Now suppose $\exists (a', b') \in F[x]_{\deg \le d_2} \times F[x]_{\deg \le d_1}$ such that $T(a', b') = 0$. Then,

$$a'f + b'g = aa'f + ab'g = a'(1 - bg) + ab'g = 0 \Rightarrow a' = (ab' - a'b)g$$

Now since $\deg(a') < \deg(g)$, the above calculation implies $a' = a' \mod g = 0$. Similarly, we can show that $b' = 0$. Hence, $\ker(T)$ is trivial. □

Next we define the Sylvester Matrix to be the matrix of the linear transformation T with respect to the basis $\{(x^i, 0) : 0 \le i < d_2\} \cup \{(0, x^j) : 0 \le j < d_1\}$. This is a $(d_1 + d_2) \times (d_1 + d_2)$ matrix of the form:

$$S = \begin{bmatrix} \overbrace{\begin{matrix} f_0 & & & \\ f_1 & f_0 & & \\ f_2 & f_1 & & \\ \vdots & \vdots & \ddots & \\ f_{d_1} & f_{d_1-1} & \ddots & \\ & f_{d_1} & \ddots & f_0 \\ & & \ddots & f_1 \\ & & \ddots & \vdots \\ & & & f_{d_1} \end{matrix}}^{d_2} & \overbrace{\begin{matrix} g_0 & & & & \\ g_1 & g_0 & & & \\ \vdots & g_1 & \ddots & & \\ g_{d_2} & \vdots & \ddots & \ddots & \\ & g_{d_2} & \ddots & \ddots & \\ & & \ddots & \ddots & g_0 \\ & & \ddots & \ddots & g_1 \\ & & & \ddots & \\ & & & & g_{d_2} \end{matrix}}^{d_1} \end{bmatrix}$$

The resultant of f and g with respect to x is defined as $\mathrm{Res}_x(f, g) = \det(S)$. Note that if $f, g \in \mathbb{F}[x, y] = (\mathbb{F}[y])[x]$, then $\mathrm{Res}_x(f, g) \in \mathbb{F}[y]$.

Observation 12.4. *If $f, g \in \mathbb{F}[x, y]$, then $\mathrm{Res}_x(f, g) = 0$ if and only if f and g share a common factor $h(x, y)$ with $\deg_x(h) \ge 1$.*

Observation 12.5. *If $\deg_y(f), \deg_y(g) \le d$, then $\deg_y(\mathrm{Res}_x(f, g)) \le 2d^2$.*

Lemma 12.6. $\mathrm{Res}_x(f, g) \in \langle f, g \rangle$ *(i.e. $\exists a, b \in \mathbb{F}[x, y]$ s.t. $\mathrm{Res}_x(f, g) = af + bg$*

Proof. By applying the row transformation $R_1 \leftarrow R_1 + xR_2 + \cdots + x^{d_1+d_2-1}R_{d_1+d_2}$, we get a matrix that looks like:

$$\begin{bmatrix} f & xf & \dots & x^{d_2-1}f & g & xg & \dots & x^{d_1-1}g \\ & & & & * & & & \\ & & & & * & & & \\ & & & & \vdots & & & \\ & & & & * & & & \end{bmatrix}$$

Expanding along the first row, we get the required observation. □

Lecture 17: March 27th, 2016

Now we will see the algorithm for factorizing bivariates that we have been building towards. As we saw earlier, the algorithm involves some preprocessing steps to ensure that the input polynomial is in an appropriate form. We will first see how to carry out each of the preprocessing steps and then see the algorithm in detail. Later we will see how a factorization of the *preprocessed* polynomial yields a factorization of the original polynomial.

For the purpose of understanding this algorithm, the input will be $f \in \mathbb{F}_q[x, y]$ with $\deg_x f, \deg_y f \le n$.

Preprocessing

- Make f square-free: This, as discussed before, can be achieved by dividing f by $\gcd\left(f, \frac{\partial f}{\partial x}\right)$. Here we have to observe that $\gcd\left(f, \frac{\partial f}{\partial x}\right)$ being 1 does not guarantee that f is square-free. In such a case we will then have to also check for $\gcd\left(f, \frac{\partial f}{\partial y}\right)$. Again, as before, if the derivatives are zero (assuming $f \neq 0$), it means that f is in fact a polynomial in x^p (or y^p). For the next steps the modified polynomial that is square-free, will be f.

- Ensure $\text{cont}(f) = 1$: Consider the polynomial $f(x, y)$ as an element of $(\mathbb{F}_q[y])[x]$ and just divide by the common factor of the coefficients (which are going to be polynomials in y), if any.

- Extend $\mathbb{F}_q$ to $\mathbb{F}_{q^t}$: This will be achieved by picking a random polynomial of degree t, which with good probability will be irreducible. This t is chosen to be a prime that is *strictly* greater than $2n^2$. As the number $2n^2$ hints, this should be something to do with making sure some resultant stays non-zero.

  Consider $R(y) = \text{Res}_x\left(f, \frac{\partial f}{\partial x}\right)$. We know that $R(y) \neq 0$ as f is square-free. Note that the degree of $R(y)$ is at most $2n^2$. Therefore it can have at most $2n^2$ roots in $\mathbb{F}_{q^t}$. Let α be such that $R(\alpha) \neq 0$. Say, $\tilde{f} = f(x, y - \alpha)$. It is easy to see that a factorization of $\tilde{f}$ yields a factorisation of f. From here we will assume $\tilde{f}$ to be our input, f.

- Make f monic in x:
  Say $f = f_0 + f_1 x + \cdots + f_n x^n$.
  We will take $\tilde{f} = f_n^{(n-1)} \cdot f\left(\frac{x}{f_n}, y\right)$, which is now monic in x.

The $\tilde{f}$ in the final step is the polynomial which we will now assume to be our input, $f \in \mathbb{F}_{q^t}[x, y]$. We will also refer to $\mathbb{F}_{q^t}$ by $\mathbb{F}$. Now f is monic in x, has content 1, is square free and also has the property that $f(x, 0)$ is square free. We will now use Hensel Lifting to factorize such an f over $\mathbb{F}[y][x]$.

Factorization using Hensel lifting

- If we substitute $y = 0$ in f, or equivalently work in $(\mod y)$ world, then we get a univariate polynomial in x. We will begin with a univariate factorization in this setting.

$$\begin{aligned} f &\equiv g_0 h_0 \bmod y \\ &\equiv g_0 h_0 \bmod y^{2^0} \end{aligned}$$

  Note that we can ensure g_0 to be irreducible and monic in x.

- We will use Hensel lifting here, to get to:

$$f \equiv g_k h_k \bmod y^{2^k}$$

In general, such a solution for g_k may not be monic. But recall that any solution must be of the form $g_k(1 \pm u)$ for some $u \in \left\langle y^{2^{k-1}} \right\rangle$. By multiplying by an appropriate $(1 - u)$, we can always ensure that g_k is monic in x.
This lifting process is done enough steps, so that we have $2^k > 2n^2$.

Let us now see how we can obtain factors for the preprocessed f from the final decomposition obtained via Hensel lifting.

Reconstruction

So far, we have obtained a factorization

$$f \equiv g_k h_k \bmod y^{2^k}$$

and g_k is monic in x and $g_k \equiv g_0 \bmod y$. Suppose $f(x, y)$ was indeed reducible, and say had $f = f_1 f_2 \cdots f_r$. When we work modulo $\langle y \rangle$, each of these f_is could have split further. Since g_0 was a monic irreducible factor of $f \bmod y$, we must have g_0 dividing some $f_i \bmod y$. Without loss of generality, let us assume that g_0 divided $f_1 \bmod y$. We would now somehow like to get hold of f_1s and this is what the reconstruction step is trying to do.

Solve for $\tilde{f}, \tilde{\ell}$ such that :

$$\deg_x \tilde{f} < \deg_x f$$
$$\deg_y \tilde{f} \leq \deg_y f$$
$$\tilde{f} = g_k \tilde{\ell} \bmod y^{2^k}$$

In order to prove correctness of this algorithm, we will now have to show that:
(1) Such a $\tilde{f}$ exists for a reducible f, and
(2) Any solution $\tilde{f}$ can be used to get a non-trivial factor of f, i.e. $\gcd(f, \tilde{f}) \neq 1$

Claim 12.7. *If f is reducible then $\tilde{f}$ exists.*

Proof. Suppose $f = f_1 f_2 \cdots f_r$, with all f_is irreducible. As mentioned earlier, g_0 must divide some $f_i \bmod y$ and let us assume it is f_1. Then we know that $f_1 \equiv g_0 \ell_0 \bmod y$, with g_0 monic and irreducible.

We use Hensel lifting to get $f_1 \equiv g'_k \ell'_k \bmod y^{2^k}$; with g_k monic.
Now, if we multiply both sides by $f_2 \cdots f_r$, we get

$$\begin{aligned} f &= f_1 \cdots f_r \\ &= g'_k(\ell'_k f_2 \cdots f_r) \bmod y^{2^k} \\ &= g'_k h'_k \bmod y^{2^k} \end{aligned}$$

By the uniqueness of Hensel Lifting (Theorem 12.1), we get $g'_k = g_k(1 + u)$ for

Formally, one needs to argue in an inductive sense that we start with $g'_0 = g_0$ and that in each subsequent steps this equality holds.

some $u \in \langle y^{2^k} \rangle$. But since both g_k and g'_k are monic in x, we must have $u = 0$ and therefore $g_k = g'_k$. This shows that $\tilde{f} = f_1$ is a valid solution to the system of equations. □

Claim 12.8. *If $\tilde{f}$ is a possible solution in the reconstruction step, then* $\gcd(f, \tilde{f}) \neq 1$.

Proof. Consider $\text{Res}_x(f, \tilde{f}) \in \mathbb{F}[y]$. Clearly, $\text{Res}_x(f, \tilde{f}) = R(y)$(say) has degree $\leq 2n^2$ (Observation 12.5).

Say $R(y) = af + b\tilde{f} \bmod y^{2^k}$, since $R(y) \in \langle f, \tilde{f} \rangle$ (Lemma 12.6).

$$\Rightarrow R(y) = ag_k h_k + bg_k \tilde{\ell} \bmod y^{2^k}$$
$$\Rightarrow R(y) = g_k(ah_k + b\tilde{\ell}) \bmod y^{2^k}$$

But the LHS is a polynomial in y alone, and the RHS is divisible by g_k, which has non-zero degree in x. The only way this is possible is if $(ah_k + b\tilde{\ell}) \equiv 0 \pmod{y^{2^k}}$. This means that $R(y) \equiv 0 \pmod{y^{2^k}}$.

But degree of $R(y) \leq 2n^2 < 2^k$. Therefore $R(y) = 0$ even without the mod, and hence $\gcd(f, \tilde{f}) \neq 1$. □

Undoing the preprocessing

Now that we have a way to obtain a factorization of the preprocessed f, let us see how to obtain the factorization of the original polynomial.

- The last preprocessing step was to make f monic by working with say $f' = f_n^{n-1} \cdot f\left(\frac{x}{f_n}, y\right)$.

  Say we obtained factors of f' to be $g'(x,y)$ and $h'(x,y)$. Define $g(x,y) = g'(f_n x, y)$ and $h(x,y) = h'(f_n x, y)$. Therefore, $g(x,y)h(x,y) = f_n^{n-1} \cdot f(x,y)$. Now we have that $\frac{g(x,y)h(x,y)}{f_n^{n-1}} \in \mathbb{F}[x,y]$, which means that f_n^{n-1} must split between $g(x,y)$ and $h(x,y)$ to give some non-trivial factorization of $f(x,y)$. Since $f_n \in \mathbb{F}[y]$, we can just compute the content of $g(x,y)$ and $h(x,y)$ and divide by the appropriate power of f_n.

- Before the above step, we had extended the field from $\mathbb{F}_q$ to $\mathbb{F}_{q^t}$. We chose a prime $t > 2n^2$ for this case.

  The issue now is that polynomials that were irreducible in $\mathbb{F}_q$ might become reducible over $\mathbb{F}_{q^t}$. We will now prove that with our constraints on t, a factorization in the larger field does indeed yield a factorization in $\mathbb{F}_q$.

  Let us say $f = gh$ in $\mathbb{F}_{q^t}[x,y]$ if we could somehow show that such a g divides f in $\mathbb{F}_q$, we will be done. For this we will need to look at the *Frobenius map* in a finite field.

  Frobenius map σ is defined as, $\sigma : \mathbb{F}_{q^t} \to \mathbb{F}_q$ such that $\sigma(a) = a^q$. This map is extended to polynomials by applying it on every coefficient. Observe that σ fixes all elements in $\mathbb{F}_q$ and that it is linear, since $(a+b)^q = a^q + b^q$ in $\mathbb{F}_{q^t}$. Also, note that if $g|f$ then $\sigma(g)|f$, as $\sigma(f) = f$ since all the coefficients

of f are from $\mathbb{F}_q$.

Say we have $f = g'_1 g'_2 \cdots g'_s$ in $\mathbb{F}_{q^t}[x, y]$. Therefore $g'_1, \sigma(g'_1), \sigma^2(g'_1), \ldots, \sigma^t(g'_1)$ are all factors of f over $\mathbb{F}_{q^t}$. Now $\sigma^t(g'_1) = g'_1$, since we are working in $\mathbb{F}_{q^t}$ and since σ already raises the coefficients to q. Suppose $\sigma^i(g'_1) = g'_1$ for some $i \neq t$, then i must divide t. But we chose t to be a prime and therefore $i = 1$ or $i = t$.

If $i = t$ then we will have t distinct factors for f, which is not possible since the degree of f (at most $2n$) is strictly smaller than t. Hence $i = 1$, which means that g'_1 has to be in $\mathbb{F}_q$, which means our factorization is indeed a factorization over $\mathbb{F}_q[x, y]$.

This finishes our argument. Let us now see the algorithm again in a more compact form.

Algorithm 30: BivariateFactoring

Input : $f \in \mathbb{F}_q[x, y]$ (of degree n in x and y)
Output: A non-trivial factor of f if one exists, or Irreducible otherwise

`// Preprocessing`

Make f square free

Ensure that $\text{cont}(f) = 1$

Extend $\mathbb{F}_q$ to $\mathbb{F}_{q^t}$ with a prime t s.t. $t > 2n^2$

Find a non-root $\alpha \in \mathbb{F}_{q^t}$ of $\text{Res}_x(f, \frac{\partial f}{\partial x})$ to go modulo $(y - \alpha)$ and remain square free. Set $f \leftarrow f(x, y - \alpha)$

Make f monic in x: Say $f = \sum_{i=1}^{n} f_i x^i$ with $f_n \neq 0$, then set $f \leftarrow f_n^{n-1} \cdot f(\frac{x}{f_n}, y))$

`// Hensel Lifting`

Factorize $f(x, 0) = g_0 h_0$ where $g_0(x)$ is irreducible and monic. If no such factorization exists, **return** Irreducible.

Let k be chosen so that $2^k > 2n^2$.

for $i = 1, \cdots, k - 1$ **do**

From the factorization $f = g_i h_i \bmod y^{2^i}$ where g_i is monic in x, use Hensel lifting to obtain

$$f = g_{i+1} h_{i+1} \bmod y^{2^{i+1}}$$

where g_{i+1} is monic in x and $g_{i+1} = g_i \bmod y^{2^i}$

`// Reconstruction`

Solve for $\tilde{f}$ and $\tilde{\ell}$ satisfying

$$\begin{aligned} \tilde{f} &= g_k \tilde{\ell} \bmod y^{2^k} \\ \deg_x(\tilde{f}) &< \deg_x(f) \\ \deg_y(\tilde{f}) &\leq \deg_y(f) \\ \deg_x(\tilde{\ell}) &< \deg_x(f) - \deg_x(g_k). \end{aligned}$$

if *no such solution exists* **then**

return Irreducible.

else

Compute $g = \gcd(f, \tilde{f})$, which would be a non-trivial factor of f. Say $f = g \cdot h$.

Undo the appropriate preprocessing steps to obtain a factor g' for the input polynomial. **return** g'.

13 Factorizing polynomials in $\mathbb{Z}[x]$

Lecture 18: March 31st, 2017

We shall now deal with factorizing univariate polynomials over integer coefficients. This would bear a lot of similarity with factorizing bivariate polynomials over finite fields. This connection might seem mysterious but turns out there is a good reason for it via a chain of prime ideal containments. Basically, we have

$$(0) \subset (p) \subset (p, x) \subset \mathbb{Z}[x],$$

$$(0) \subset (y) \subset (y, x) \subset \mathbb{F}_q[x, y],$$

as the largest chain of prime ideal containments. Even in the algorithm we shall describe, we shall be choosing a prime p to play the role that y had in our bivariate factorization.

The input is a polynomial $f(x) \in \mathbb{Z}[x]$. Unlike earlier, the coefficients are now integers which can potentially be unbounded so the input size will involve the size of the coefficients also. Let us assume that each of the coefficient of f have absolute value at most 2^c (i.e., they are c-bit integers). The goal would be to design an algorithm to compute a non-trivial factor (if any exists) in time $\text{poly}(n, c)$.

Size of factors of $f(x)$

But before we do this, suppose $g(x) \in \mathbb{Z}[x]$ is a factor of f, why should it even be the case that the coefficients of g are small? Is it even reasonable to ask for factorization? Fortunately, we do have a good bound.

Lemma 13.1 (Mignotte's Bound)**.** *Let $f \in \mathbb{Z}[x]$ and say each of its coefficients is bounded by 2^c in absolute value. If $g \in \mathbb{Z}[x]$ is a factor of f, then each coefficient of g is bounded by $2^n \cdot n^n \cdot 2^{cn}$ in absolute value. That is, the coefficients of g are at most $O(n \log n + cn)$ bits long.*

Proof. Suppose $\alpha \in \mathbb{C}$ is a complex root* of f, i.e. $f(\alpha) = 0$, how large can $|\alpha|$ be? If $f = a_0 + a_1 x + \cdots a_n x^n$, then we must have

* - *"Wait, why are we going to complex numbers now?" Patience."*

$$|a_n||\alpha^n| = \left|\sum_{i=0}^{n-1} \alpha^{n-1} a_i\right|$$

$$\implies |\alpha^n| \leq \left|\sum_{i=0}^{n-1} \alpha^{n-1} a_i\right| \leq |\alpha|^{n-1} \left(\sum_{i=0}^{n-1} |a_i|\right)$$

This forces $|\alpha| \leq n \cdot 2^c$.

What was the connection with g? Note that the roots of g is just a subset of the roots of f, and every coefficient of g is a symmetric polynomial in these roots and they happen to be integers! Now how large can any symmetric polynomial over these roots be? For a very crude bound, each symmetric polynomial is a sum of at most 2^n terms, each being a product of at most n of the roots of f. Hence, every coefficient of g is at most $2^n \cdot (n \cdot 2^n)^n$. □

This tells us that asking for the factors of $f(x)$ is not unreasonable. We shall now describe the sketch of the algorithm with a few missing parts that shall be filled in shortly. We'll give the full description at the end of this section.

Informal description of algorithm

1. **Preprocessing step:**
   (a) Make $f(x)$ square-free (by removing any gcd with its derivative).
   (b) Make $f(x)$ content-free (by removing any common factors across all its coefficients).
   (c) Find a prime p such that f mod p remains square free, and $\deg(f)$ does not decrease when considered modulo p (that is, the leading term does not vanish modulo p).
2. **Base factorization and lifting:**
   (a) Factorize $f(x)$ mod p as
   $$f(x) = g_0(x)h_0(x) \bmod p$$
   where $g_0(x)$ is a monic, irreducible polynomial, via Algorithm 28. If there is no non-trivial factorization, output Irreducible.
   (b) Do Hensel lifting for k steps (to be determined soon) to obtain
   $$f(x) = g_k(x)h_k(x) \bmod p^{2^k}$$
   where $g_k = g_0 \bmod p$, and g_k is monic.
3. **Reconstruction step:**
   (a) Solve for a polynomial $\tilde{f}$ with $\deg \tilde{f} < n$ and *each coefficient of* $\tilde{f}$ *being small* such that
   $$\tilde{f} = g_k \cdot \tilde{\ell} \bmod p^{2^k}$$
   (b) If no such solution $\tilde{f}$ exists, output Irreducible. Else, output $\gcd(f, \tilde{f})$.

The parts that are marked in red need to be elaborated and we shall do the easy steps first.

13.1 *Finding a good prime*

Suppose f was a square-free polynomial with its coefficients bounded by 2^c. Then clearly, the number of primes that divide the leading term is at most c. These are primes that we cannot take. But there may be other bad primes that f mod p non-square-free even though f is square-free. When can this happen? Recall that f is square-free if and only if f has a non-trivial gcd with f'. Thus, f is square-free if and only if $\mathrm{Res}_x(f, f') \neq 0$. Note that the resultant of univariate

polynomials over integers is also an integer (just a determinant of an integer matrix). Hence, if we choose a prime that does not divide this resultant, we would be done.

Lemma 13.2. *Let f and g be integer polynomials of degree at most n that have coefficients bounded by 2^c. Then, $|\mathrm{Res}_x(f,g)| \leq (2n)! \cdot 2^{2cn}$.*

Proof. The resultant of f and g is the determinant of a matrix of size at most $2n \times 2n$ made up of entries bounded by 2^c. Hence, its determinant can be at most $(2n)! \cdot 2^{2cn}$. □

Corollary 13.3. *There is a prime within the first $(cn)^2$ integers that neither divides $\mathrm{Res}_x(f,f')$ nor the leading coefficient of f.*

Proof. The number of primes that divide either the resultant or the leading coefficient of f is at most $O(cn + n\log n)$. The first $(cn)^2$ integers has more primes than this. □

Thus, we can just run through the first $(cn)^2$ primes and choose a prime p that works and proceed to the next steps.

13.2 *How large should k be?*

In the bivariate case, $\mathrm{Res}_x(f,\tilde{f})$ was a polynomial in y, and we chose our k so that 2^k was bigger than the degree. Here, we just need to work with the magnitude instead as the resultants here are just numbers.

But in order to get a bound on the size of $\mathrm{Res}_x(f,\tilde{f})$, we need a bound on the coeffcients of $\tilde{f}$. This is promised by the reconstruction step, though we do not know what precisely "small" means there.

Lemma 13.4. *Suppose we have an $\tilde{f}$ for whch there is some $\tilde{\ell} \in \mathbb{Z}[x]$ such that $\tilde{f}$ to $\tilde{f} = g_k\tilde{\ell} \bmod p^{2^k}$ with $\deg \tilde{f} < n$ and each coefficient of $\tilde{f}$ has absolute value at most 2^r. If $p^{2^k} > (2n)! \cdot 2^{2n(r+c)}$, then $\gcd(f,\tilde{f}) \neq 1$.*

Proof. Consided $\mathrm{Res}_x(f,\tilde{f})$. By Lemma 12.6, there must be polynomials $a(x), b(x)$ such that

$$\begin{aligned} & \mathrm{Res}_x(f,\tilde{f}) = af + b\tilde{f} \\ \implies & \mathrm{Res}_x(f,\tilde{f}) = af + b\tilde{f} \bmod p^{2^k} \\ & \qquad\qquad = ag_kh_k + bg_k\tilde{\ell} \bmod p^{2^k} \\ & \qquad\qquad = g_k(ah_k + b\tilde{\ell}) \bmod p^{2^k}. \end{aligned}$$

But the LHS is a number modulo p^{2^k} but the RHS is a multiple by g_k, which has non-zero degree in x. Hence, this forces

$$\mathrm{Res}_x(f,\tilde{f}) = 0 \bmod p^{2^k}.$$

But notice that we chose k so that p^{2^k} is bigger than the larest possible value of $\mathrm{Res}_x(f, \tilde{f})$. Hence, $\mathrm{Res}_x(f, \tilde{f})$ is zero even without the mod. Hence, f and $\tilde{f}$ must share a common factor. □

And as seen earlier Claim 12.7, we know that a solution does indeed exist as g_0 must divide some honest-to-God factor of f, and that is a valid choice for $\tilde{f}$.

What is left therefore is to see how to find a *short* vector $\tilde{f}$ that is divisible by g_k. This is where lattices come in.

13.3 *Lattices*

Motivation

The reconstruction step required that we look for a polynomial $\tilde{f}$ of degree at most $n-1$ and "small coefficients" such that g_k divides it mod p^{2^k}. Since we have some weird condition on the coefficients, let us try and rewrite it somehow. Say $\deg g_k = m$. As we run over possibilities for $\tilde{\ell} = a_0 + a_1 x + \cdots + a_{n-1-m}x^{n-1-m}$, the polynomial $g_k\tilde{\ell}$ is

$$\sum_{i=0}^{n-m-1} a_i x^i g_k(x).$$

What we want is, after we reduce each coefficient modulo p^{2^k}, each coefficient is small. We can express this as

$$\sum_{i=0}^{n-m-1} a_i x^i g_k(x) - \sum_{i=0}^{n} b_i p^{2^k} x^i,$$

where we are basically removing any multiple of p^{2^k} in each "coordinate" if possible. We can restate our goal in the following form:

> Find an integer linear combination of the vectors
>
> $$\left\{g_k(x), x g_k(x), \ldots, x^{n-m-1} g_k(x), p^{2^k}, p^{2^k}x, \cdots, p^{2^k}x^{n-1}\right\}$$
>
> (where each polynomial is thought of as a vector of length n with its coefficients listed out) of "small length".

This is what a *lattice*. Formally, a lattice over $\mathbb{Z}$ is defined as follows.

Definition 13.5 (Lattice)**.** *Let $\mathbf{b}_1, \ldots, \mathbf{b}_m \in \mathbb{Z}^n$. The* lattice generated by them, *denoted by $\langle \mathbf{b}_1, \ldots, \mathbf{b}_m \rangle_{\mathbb{Z}}$ is the set of all integer linear combinations of these vectors. That is,*

$$\langle \mathbf{b}_1, \ldots, \mathbf{b}_m \rangle_{\mathbb{Z}} = \left\{ \sum_i \alpha_i \mathbf{b}_i \ : \ \alpha_i \in \mathbb{Z} \text{ for all } i \right\}. \qquad \lozenge$$

Remark 13.6. *The subspace spanned by the generating set consists of $\mathbb{Q}$-linear combinations of these vectors whereas the lattice is only with integer combinations. In fact,*

turns out that there is always a generating set of a lattice whose size is equal to the rank *of the subspace spanned by them (you'll see why in Problem Set 2). Furthermore, such a generating set can be found efficiently.*

So we shall assume from now on that whenever we are given a generating set $\{\mathbf{b}_1, \ldots, \mathbf{b}_m\} \subset \mathbb{Z}^n$*, we will assume that* $m = \operatorname{rank}\left(\operatorname{span}_{\mathbb{Q}} \{\mathbf{b}_1, \ldots, \mathbf{b}_m\}\right)$. ◊

The Shortest Vector Problem (SVP): Given input vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ that generate a lattice $\mathcal{L} \subseteq \mathbb{Z}^n$, find a vector $\mathbf{b}_{\text{opt}} \in \mathcal{L}$ such that $\|\mathbf{b}_{\text{opt}}\| = \min_{0 \neq \mathbf{b} \in \mathcal{L}} \|\mathbf{b}\|$.

Turns out, finding the shortest vector in a given lattice is NP-hard. So it is infeasible to solve the SVP problem in order to use it for our factorization algorithm. However, we do not really need the shortest possible solution. All we need is a *short enough* solution. A beautiful algorithm of Lenstra, Lenstra and Lovász give an approximation algorithm for the SVP problem.

Theorem 13.7 (Lenstra, Lenstra, Lovász)**.** *Given integer vectors* $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\} \subseteq \mathbb{Z}^n$*, there is a deterministic polynomial time (in n and the size of coordinates of* $\mathbf{b}_i$*s) algorithm that computes a vector* $\mathbf{0} \neq \mathbf{b} \in \mathcal{L} := \langle \mathbf{b}_1, \ldots, \mathbf{b}_n \rangle_{\mathbb{Z}}$ *such that*

$$\|\mathbf{b}\| \leq 2^{(n-1)/2} \cdot \min_{\mathbf{0} \neq \mathbf{b}' \in \mathcal{L}} \|\mathbf{b}'\|.$$

That is, it gives a $2^{(n-1)/2}$*-factor approximation to the shortest vector problem.*

Before we can describe the algorithm, we would need a few preliminaries, namely the Gram-Schmidt orthogonalization.

13.4 *Gram-Schmidt Orthogonalization*

The idea of the Gram-Schmidt process is, given a basis $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_m\} \subseteq \mathbb{Q}^n$, the Gram-Schmidt process output a different basis $B^* = \{\mathbf{b}_1^*, \ldots, \mathbf{b}_m^*\}$ that generate the same space but are pairwise orthogonal. The algorithm is pretty standard and is described below.

Algorithm 31: Gram-Schmidt

Input : A full rank basis $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_m\} \subseteq \mathbb{Q}^n$
Output: An orthogonal basis for the space generated by B

1 **for** $i = 1, \ldots, m$ **do**
2 Compute

$$\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^* \quad , \quad \text{where } \mu_{i,j} = \frac{\left\langle \mathbf{b}_i, \mathbf{b}_j^* \right\rangle}{\left\langle \mathbf{b}_j^*, \mathbf{b}_j^* \right\rangle}.$$

3 Define the matrix

$$M = \begin{bmatrix} 1 & & & \\ \mu_{2,1} & 1 & & \\ \vdots & \ddots & \ddots & \\ \mu_{m,1} & \cdots & \mu_{m,m-1} & 1 \end{bmatrix}.$$

return $B^* = \left\{\mathbf{b}_1^*, \ldots, \mathbf{b}_m^*\right\}$ *and* M.

It is clear from the description of the algorithm that if we were to list the input basis vectors as rows of a matrix B, and if B^* is thought of as the output vectors listed as rows, then

$$M \cdot B^* = B.$$

Here are a few simple but important observations.

Observation 13.8. *For all $i \leq m$, the space generated by $\{\mathbf{b}_1, \ldots, \mathbf{b}_i\}$ is the same as the space generated by $\left\{\mathbf{b}_1^*, \ldots, \mathbf{b}_i^*\right\}$.*

Observation 13.9. *If $S_i = \operatorname{span}\{\mathbf{b}_1, \ldots, \mathbf{b}_i\}$, and if $S_i^\perp$ is its orthogonal complement, then $\mathbf{b}_{i+1}^*$ is just the projection of $\mathbf{b}_{i+1}$ on to $S_i^\perp$.*

Observation 13.10. *The output of Algorithm 31 would vary if the input vectors are rearranged.*

The idea of the LLL algorithm would be the following. We would like to just do the Gram-Schmidt orthogonalization and output the smallest basis vector, however some of the $\mu_{i,j}$s may not be integers so these transformations do not stay within the lattice. However, what we shall do is as much as we can of the Gram-Schmidt process within the lattice by which we end up with a new basis that is sort-of-orthogonal. And then will come the magical Lovász step of checking if a particular condition holds and swapping vectors and restarting the whole process. The algorithm is completely described below, but we'll prove the correctness etc. in the next class.

Algorithm 32: LLL Algorithm

Input : A full rank basis $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_m\} \subseteq \mathbb{Z}^n$
Output: A (short enough) vector $\mathbf{b} \in \langle \mathbf{b}_1, \ldots, \mathbf{b}_m \rangle_{\mathbb{Z}}$

1 **repeat**
2 Using Algorithm 31, compute M, B^* such that $MB^* = B$.
`// Do integer row operations on both sides.`
3 **for** $i = m, \cdots, 1$ **do**
4 **for** $j = i-1, \cdots, 1$ **do**
5 $\mathbf{b}_i \leftarrow \mathbf{b}_i - \lceil \mu_{i,j} \rfloor \mathbf{b}_j$, where $\lceil \mu_{i,j} \rfloor$ is the integer nearest to $\mu_{i,j}$.
6 Update row i of M accordingly.
`// We again have` $MB^* = B$ `with now` M `being lower triangular and all off-diagonal entries having absolute value at most` $\frac{1}{2}$.
7 **if** *for some* $1 \leq i \leq m-1$ *we have* $\left(\frac{3}{4}\right) \left\|\mathbf{b}_i^*\right\|^2 > \left\|\mathbf{b}_{i+1}^* + \mu_{i+1,i}\mathbf{b}_i^*\right\|^2$ **then**
8 Swap $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$.
9 **else**
10 **return** $\mathbf{b}_1$
11 **until** *until you return something*

This seems to be a mysterious algorithm, not just for the unbounded repeat loop but a rather curious criterion under which we swap some two vectors and repeat the process. It might seem a bit weird at first, and we will look at it in detail shortly but at the moment, the following exercise should throw some light on the weird condition.

Exercise 1. *On input* $\{\mathbf{b}_1, \ldots, \mathbf{b}_m\}$, *suppose Gram-Schmidt outputs* $\{\mathbf{b}_1^*, \ldots, \mathbf{b}_m^*\}$. *Now consider applying the Gram-Schmidt process after swapping the vectors* $\mathbf{b}_i$ *and* $\mathbf{b}_{i+1}$ *and suppose the output vectors are* $\{\hat{\mathbf{b}_1}, \ldots, \hat{\mathbf{b}_m}\}$. *Then, the following holds:*

- *For all* $j < i$, *we have* $\hat{\mathbf{b}_j} = \mathbf{b}_j^*$.
- *For all* $j > i+1$, *we have* $\hat{\mathbf{b}_j} = \mathbf{b}_j^*$.
- $\hat{\mathbf{b}_i} = \mathbf{b}_{i+1}^* + \mu_{i+1,i}\mathbf{b}_i^*$.

Therefore, line 7 of Algorithm 32 is asking — Does swapping two successive vectors lead to a significant drop in the length of the i-th Gram-Schmidt vector? If there is such a significant drop (by a factor of 3/4), then we do such a swap.

To show that the above algorithm works, we need to show two things — (1) If it does terminate, whatever it outputs is a good approximation of the shortest vector in the lattice, and (2) the algorithm does terminate in polynomially many steps.

Lecture 19: April 10th, 2017

If the algorithm terminates, it is right

Lemma 13.11. *Suppose $\{\mathbf{b}_1^*, \ldots, \mathbf{b}_m^*\}$ is an orthogonal system of vectors (as obtained by Algorithm 31) on input $\{\mathbf{b}_1, \ldots, \mathbf{b}_m\}$ such that*

$$M \cdot B^* = B,$$

where B and B^* are matrices whose rows are $\{\mathbf{b}_1, \ldots, \mathbf{b}_m\}$ and $\{\mathbf{b}_1^*, \ldots, \mathbf{b}_m^*\}$ respectively, and M is a lower-triangular matrix with 1s on the diagonal. If $\mathbf{b}_{opt}$ is the shortest vector in $\mathcal{L}$, the lattice generated by $\{\mathbf{b}_1, \ldots, \mathbf{b}_m\}$, then $\|\mathbf{b}_{opt}\| \geq \min_i \|\mathbf{b}_i^*\|$.

Proof. Say $\mathbf{b}_{\text{opt}} = \sum_i \alpha_i \mathbf{b}_i$ such that $\alpha_i \in \mathbb{Z}$, and let k be the largest index such that $\alpha_i \neq 0$. Now, if we express $\mathbf{b}_i = \mathbf{b}_i^* + \sum_{j<i} \mu_{i,j} \mathbf{b}_j^*$, we obtain:

$$\mathbf{b}_{\text{opt}} = \alpha_k \mathbf{b}_k^* + (\text{terms involving } \mathbf{b}_i^* \text{ for } i < k).$$

Using Pythagoras on the previous equation gives

$$\|\mathbf{b}_{\text{opt}}\|^2 \geq |\alpha_k|^2 \cdot \|\mathbf{b}_k^*\|^2 \geq \|\mathbf{b}_k^*\|^2$$

as α_k is a non-zero integer and therefore $|\alpha_k| \geq 1$. □

At any time in Algorithm 32 before the swap-step, we do have an equation of the form $MB^* = B$ and hence we may claim that $\|\mathbf{b}_{\text{opt}}\| \geq \min_i \|\mathbf{b}_i^*\|$. However, the issue is that we must output a vector in the lattice $\mathcal{L}$, and from the rows of B^*, only $\mathbf{b}_1^*$ is guaranteed to lie in $\mathcal{L}$. This is exactly where the termination condition helps us.

Whenever Algorithm 32 terminates, the termination condition ensures the following: For all $i \in \{1, \ldots, m-1\}$,

$$\begin{aligned}
\frac{3}{4}\|\mathbf{b}_i^*\|^2 &\leq \|\mathbf{b}_{i+1}^* + \mu_{i+1,i}\mathbf{b}_i^*\|^2 \\
&= \|\mathbf{b}_{i+1}^*\|^2 + \mu_{i+1,i}^2 \|\mathbf{b}_i^*\|^2 && \text{(Pythagoras)} \\
&\leq \|\mathbf{b}_{i+1}^*\|^2 + \frac{1}{4}\|\mathbf{b}_i^*\|^2 && \left(|\mu_{i+1,i}| \leq \frac{1}{2}\right) \\
\implies \|\mathbf{b}_{i+1}^*\|^2 &\geq \frac{\|\mathbf{b}_i^*\|^2}{2}
\end{aligned}$$

An orthogonal basis that satisfies $\|\mathbf{b}_{i+1}^*\|^2 \geq \|\mathbf{b}_i^*\|^2/2$ is called a *reduced Basis*. Combining the above with Lemma 13.11 makes the following plain.

Proposition 13.12. *If $\{\mathbf{b}_1^*, \ldots, \mathbf{b}_m^*\}$ is a reduced basis then $\|\mathbf{b}_1\|^2 \leq 2^{n-1}\|\mathbf{b}_{opt}^*\|^2$.* □

The algorithm terminates in polynomially many steps

So, what have we established till now? We know that if Algorithm 32 does terminate then, by Proposition 13.12, $\|\mathbf{b}_1\| = \|\mathbf{b}_1^*\| \leq 2^{\frac{n-1}{2}}\|\mathbf{b}_{\text{opt}}\|$. Thus, we may safely

output $\mathbf{b}_1$, to achieve an approximation ratio of $2^{\frac{n-1}{2}}$.

Now, let us understand why Algorithm 32 does terminate in a polynomial number of steps (in n, m and the size of the coordinates of $\mathbf{b}_i$'s). A priori, it is not clear that the algorithm stops at all. To this end suppose at line 7 we find that for some $1 \leq i \leq m-1$ we have $\frac{3}{4}\left\|\mathbf{b}_i^*\right\|^2 > \left\|\mathbf{b}_{i+1}^* + \mu_{i+1,i}\mathbf{b}_i^*\right\|^2$, then we swap $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$. Now, $\{\mathbf{b}_1, \ldots, \mathbf{b}_{i+1}, \mathbf{b}_i, \ldots, \mathbf{b}_m\}$ becomes the input to Algorithm 31 at line 2. Suppose the output vectors of the Gram-Schmidt process are $\left\{\hat{\mathbf{b}_1}, \ldots, \hat{\mathbf{b}_m}\right\}$, represented as rows in the matrix $\hat{B}$. Observe that the subsequent run through lines 3–6 does not perturb $\hat{B}$. Therefore, by Exercise 1 we obtain that $\left\{\hat{\mathbf{b}_1}, \ldots, \hat{\mathbf{b}_m}\right\}$ and $\left\{\mathbf{b}_1^*, \ldots, \mathbf{b}_m^*\right\}$ differ only on the $(i)^{th}$ and $(i+1)^{th}$ vector, and further $\hat{\mathbf{b}_i} = \mathbf{b}_{i+1}^* + \mu_{i+1,i}\mathbf{b}_i^*$. This means that $\frac{3}{4}\left\|\mathbf{b}_i^*\right\|^2 > \left\|\hat{\mathbf{b}_i}\right\|^2$.

Hmmm ... so the norms of all the vectors except the $(i)^{th}$ and $(i+1)^{th}$ stay the same and the norm of the $(i)^{th}$ one drops by a constant factor. If only, we had a handle on the norm of the $(i+1)^{th}$ vector, we could argue that some potential function, like $\prod_{i=1}^{m}\left\|\mathbf{b}_i^*\right\|^2$ is decreasing each time the swap at line 7 is performed, and hence Algorithm 32 must terminate within so-and-so number of steps. Great! Unfortunately*, $\prod_{i=1}^{m}\left\|\mathbf{b}_i^*\right\|^2$ stays constant throughout Algorithm 32!

** or fortunately!*

To see this, observe the elementary row operations performed on the input matrix B: as we noted earlier, each operation is either a swap (line 7) or of the form $R_i \to R_i - \sum_j \beta_j R_j$, where $\beta_j \in \mathbb{Z}$. These are all invertible operations on the lattice. Consider $B' = RB$ where B' is any intermediate form that B acquires and R is the corresponding $m \times m$ matrix representing the accrued Row-elementary operations. R is an integer matrix and invertible; further R^{-1} is also and integer matrix!* Since, $\det(R)$ and $\det\left(R^{-1}\right)$ are both integers and $\det(R) \times \det\left(R^{-1}\right) = 1$, we obtain $\det(R) = \pm 1$. Such matrices (R such that $\det R = \pm 1$) are infact called *unimodular matrices* and these are precisely transformations we can perform on the lattice basis without changing the lattice.

** Why? Think about reversing all the Row-elementary operations performed.*

Observation 13.13. *If the rows of an $m \times n$ integer matrix B is a basis of some lattice and B' is another $m \times n$ integer matrix. Then, rows of B and the rows of B' generate the same lattice if and only if $B' = RB$ where R is an $m \times m$ unimodular integer matrix.*

Therefore, we'd like to say that $|\det B| = |\det B'|$ throughout, but wait, these matrices aren't even square matrices. But let us make them square matrices by looking at BB^T and $B'B'^T$ instead. Hence, $\det BB^T = \det B'B'^T$ as $B' = RB$ and $\det R = \pm 1$.

Let us couple this observation with the fact that $MB^* = B$, and M is lower triangular with all the diagonal entries 1 implying that $\det(M) = 1$. Thus, by the orthogonality of the rows of B^*, we establish that

$$\det\left(B^*B^{*^T}\right) = \prod_{i=1}^{m}\left\|\mathbf{b}_i^*\right\|^2 = \det\left(BB^T\right)$$

stays invariant.

All this effectively tells us that using $\prod_{i=1}^{m} \left\|b_i^*\right\|^2$ as a potential function is useless because this never changes throughout the algorithm. Now, it is time for more magic. Instead of looking at the potential function $\prod_{i=1}^{m} \left\|\mathbf{b}_i^*\right\|^2$, we look at the following variant:

$$\begin{aligned} D_i &:= \prod_{j=1}^{i} \left\|\mathbf{b}_j^*\right\|^2 \\ \Phi &:= \prod_{i=1}^{m} D_i \\ &:= \left\|\mathbf{b}_1^*\right\|^{2m} \left\|\mathbf{b}_2^*\right\|^{2m-2} \cdots \left\|\mathbf{b}_m^*\right\|^2 \end{aligned}$$

Lemma 13.14. *If B is an integer matrix, then each D_i is also an integer and is non-zero.*

Proof. Consider the Gram-Schmidt equation $MB^* = B$ where B is an upper-triangular, if we focus on just the first i rows of B and B^*, then we can restrict ourselves to the $i \times i$ principal minor of M to get an equation $M_i B_i^* = B_i$. Since M was upper-triangular with 1s on the diagonal, M_i continues to share that property. Therefore,

$$\begin{aligned} \det B_i B_i^T &= \det B_i^* (B_i^*)^T \\ &= \left\|\mathbf{b}_1^*\right\|^2 \cdots \left\|\mathbf{b}_m^*\right\|^2 \\ &=: D_i \end{aligned}$$

And since we are starting with a full-rank basis, none of the $\mathbf{b}_i^*$s are zero and hence D_i is non-zero. □

Lemma 13.15. *Every swap at line 7 of Algorithm 32 reduces Φ by at least $\frac{3}{4}$.*

Proof. Suppose, in some iteration, at line 7 we swap $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$, and after a run through lines 2–6, we obtain $\{\hat{\mathbf{b}_1}, \ldots, \hat{\mathbf{b}_m}\}$. Let $D_i' = \prod_{j=1}^{i} \left\|\hat{\mathbf{b}_j}\right\|^2$ Then, by our earlier observations and Exercise 1 it is clear that for $j < i$, we have $D_j = D_j'$. Additionally, $D_i' \leq \frac{3}{4} D_i$.

We also have $D_{i+1} = D_{i+1}'$. This is because swapping rows i and $i+1$ does not change $\det(B_{i+1} B_{i+1}^T) = D_{i+1}$. Similarly, for $j > i+1$, $D_j = D_j'$.

Hence, only D_i reduces by a factor of 3/4, and all other D_js stay the same. Therefore, $\prod_j D_j' \leq \left(\frac{3}{4}\right) \prod_j D_j$. □

Now we know that every swap step reduces our potential function by at least a factor of 3/4. We also know from Lemma 13.14 that Φ is always a positive integer. This should let us upper-bound the number of swaps throughout if we have a bound on how large Φ is in the beginning of the algorithm.

Proposition 13.16. *Φ is maximum at the beginning of Algorithm 32, where $|\Phi| \leq (m! \times (n 2^{2c})^m)^m$, where c is the maximum length of the bit representations of the coefficients of the input vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_m\}$.*

Proof. As seen in Lemma 13.14, $D_i = \det B_i B_i^T$. If each of the entries of B are integers of magnitude at most 2^c, each entry of the $m \times m$ matrix $B_i B_i^T$ has magnitude at most $n2^{2c}$. Therefore, $D_i = \det(B_i B_i^T)$ is at most $m!(n2^{2c})^m$. □

Proposition 13.17. *Algorithm 32 performs at most* $O\left(m^2 c + m^2 \log(m) + m \log(n)\right)$ *swap operations at line 7, and hence runs in time polynomial in n, m and the size of the coordinates of* $\mathbf{b}_i$*'s.*

And that completes the analysis of the LLL algorithm and hence Theorem 13.7, and hence concludes the algorithm to factorize polynomials in $\mathbb{Z}[x]$.

Algorithm 33: Factorization over integers

Input : $f \in \mathbb{Z}[x]$ of degree n, and coefficients bounded by 2^c
Output: A non-trivial factor of f if one exists, or Irreducible otherwise

`// Preprocessing`
1 Make f square free
2 Ensure that $\text{cont}(f) = 1$
3 Find a prime p such that $f \bmod p$ is square-free and $\deg(f)$ and p does not divide the leading coefficient of f.

`// Hensel Lifting`
4 Factorize $f(x) = g_0 h_0 \bmod p$ where $g_0(x)$ is irreducible and monic. If no such factorization exists, **return** Irreducible.
5 Let k be chosen so that $p^{2^k} > 2^{3cn^3}$.
6 **for** $i = 1, \cdots, k-1$ **do**
7 From the factorization $f = g_i h_i \bmod p^{2^i}$ where g_i is monic, use Hensel lifting to obtain

$$f = g_{i+1} h_{i+1} \bmod p^{2^{i+1}}$$

where g_{i+1} and $g_{i+1} = g_i \bmod p^{2^i}$

`// Reconstruction`
8 If $\deg g_k = m$, construct the lattice

$$\mathcal{L} := \left\langle g_k(x), x g_k(x), \ldots, x^{n-m-1} g_k(x), p^{2^k}, p^{2^k} x, \ldots, p^{2^k} x^{n-1} \right\rangle_{\mathbb{Z}}.$$

Using Algorithm 32, compute a short vector $\tilde{f}(x) \in \mathcal{L}$.

9 **if** $\left\|\tilde{f}\right\| > 2^{3cn^2}$ **then**
`// If f was reducible, then LLL ought to find an` $\tilde{f}$ `with coefficients at most` $2^{2cn^2 + O(n)}$.
10 **return** Irreducible.
11 **else**
12 Compute $g = \gcd(f, \tilde{f})$, which would be a non-trivial factor of f. Say $f = g \cdot h$.
13 Undo the appropriate preprocessing steps to obtain a factor g' for the input polynomial. **return** g'.

14 Kaltofen's black-box factorization algorithm

Lecture 20: April 10th, 2017

The algorithms we have seen in section 12 that we can use an algorithm for k-variate factorization and boot-strap it with the Hensel Lifting approach to obtain a factorization algorithm for $(k+1)$-variate polynomials. This works of course, but the running time degrades *very badly* in k. Hence, using this approach for factorizing multivariate polynomials (where the number of variables n grows) is infeasible.

The other question is, if we are dealing with polynomials over many variables, we could potentially have many monomials for such a polynomial. How is the input provided to us?

Kaltofen came up with a remarkable algorithm where given just blackbox access to the polynomial (that is, all we know are its degree, number of variables, and we are only allowed to evaluate the polynomial at any chosen point of ours), we can construct blackboxes for each of its factors as well! Firstly, it is not at all clear that the factors should even be efficiently computable in the first place! Kaltofen's result shows that they indeed can be, and also construct a randomized algorithm to find blackboxes for the factors given a blackbox for the polynomial.

A key ingredient in the algorithm is the Hilbert's Irreducibility theorem Theorem 14.1 which gives a test for irreducibility of multivariate polynomials.

Hilbert's irreducibility theorem

If we wish to come up with a blackbox factorization algorithm, at the very least we should be able to check if a given polynomial is irreducible or not. We know how to test for irreducibility for univariate polynomials using any of the factoring algorithms we have seen previously. A natural approach seems to be to reduce to this case, by substituting each variable x_i with a polynomial function $a_i(t)$ and checking for the irreducibility of the univariate polynomial $f(a_1(t), \ldots, a_n(t))$. However there is an obvious reason why this idea won't work because if the field is $\mathcal{C}$, every univariate polynomial factorizes into linear factors.

The following remarkable theorem shows that a random *bivariate substitution* preserves irreducibility:

$$(x, y_1, \ldots, y_n) \mapsto (x, a_1 t + b_1, \ldots, a_n t + b_n).$$

For simplicity, we shall state the theorem for polynomials that are monic in x, but as we have seen earlier, this can very easily be ensured by minor massaging.

Theorem 14.1 (Hilbert's Irreducibility Theorem)**.** *Let $f(x, y_1, \ldots, y_n)$ be a degree d polynomial which is monic in x. Suppose $S \subseteq \mathbb{F}$ and*

$$\Pr_{\bar{a}, \bar{b} \in S^n} [\, f(x, a_1 t + b_1, \ldots, a_n t + b_n) \text{ is reducible }] \geq \frac{O(d^5)}{|S|}$$

then $f(x, y_1, \ldots, y_n)$ is reducible.

We shall prove this shortly but for now let us assume this and complete Kaltofen's algorithm.

Algorithm

Here is an algorithm for black box factorization assuming Theorem 14.1. We will see a proof of 14.1 in the next section.

The algorithm has two steps: In step 1, given the blackbox for f, the algorithm outputs the factorization pattern of f. In step 2, given an index i and a point $(\alpha, \beta_1, \ldots, \beta_n)$ it outputs the evaluation of the i-th factor at the point i.e., $f_i(\alpha, \beta_1, \ldots, \beta_n)$. Fix a set $S \in \mathbb{F}$ large enough.

Algorithm 34: Blackbox Factorization

Input : A blackbox computing $f \in \mathbb{F}[x, y_1, \ldots, y_n]$ that is monic in x and of degree at most d
Output: Factorization pattern of f

1 Pick $\bar{a}, \bar{b}$ uniformly at random from S^n.
2 Compute the monomial representation of $f_{\bar{a},\bar{b}}(x,t) = f(x, a_1 t + b_1, \ldots, a_n t + b_n)$ by evaluating it at sufficiently many points and interpolating.
3 Factorize $f_{\bar{a},\bar{b}}(x,t)$ to obtain

$$f_{\bar{a},\bar{b}}(x,t) = f'_1(x,t)^{e_1} \cdots f'_r(x,t)^{e_r}.$$

4 **return** $(e_1, e_2, \ldots, e_r)$ as the factorization pattern

Input : An index $i \in [r]$ and a point $(\alpha, \beta_1, \ldots, \beta_n)$
Output: The evaluation of the i-th factor of f at $(\alpha, \beta_1, \ldots, \beta_n)$

5 Compute the polynomial

$$\tilde{f}(x, t_1, t_2) = f(x, a_1 t_1 + t_2(\beta_1 - b_1) + b_1, \ldots, a_n t_1 + t_2(\beta_n - b_n) + b_n).$$

6 Factorize this polynomial as $\tilde{f} = \tilde{f}_1^{e'_1} \cdots \tilde{f}_r^{e'_r}$
7 Find the j for which $\tilde{f}_j(x, t, 0) = f'_i(x, t)$.
8 **return** $f_j(\alpha, 0, 1)$

Theorem 14.2. *Algorithm 34 outputs the right answer with probability at least* $1 - O(d^6)/|S|$.

Proof. Suppose the true irreducible factors of f were $\{f_1, \ldots, f_r\}$. We shall say that the projection $\bar{a}, \bar{b}$ is *bad* for f_i if the corresponding bivariate projection of f_i fails to be irreducible. By Theorem 14.1, the probability that $\bar{a}, \bar{b}$ is bad for any fixed f_i is at most $O(d^5)/|S|$. By taking a union bound over all factors, the probability that it is bad for *any* f_i is bounded by $O(d^6)/|S|$. We will show that if the chosen projection $\bar{a}, \bar{b}$ is not bad for any f_i, then the algorithm outputs the right answer.

Since the chosen projection was not bad, it is clear that the factorization pattern of $f_{\bar{a},\bar{b}}(x,t)$ is the correct factorization pattern for f. Further more, the

polynomial $\tilde{f}(x, t_1, t_2)$ is constructed so that

$$\tilde{f}(x, t, 0) = f_{\bar{a},\bar{b}}(x, t) \quad \text{and} \quad \tilde{f}(\alpha, 0, 1) = f(\alpha, \beta_1, \ldots, \beta_n).$$

The only issue is that when we factorize $\tilde{f}$, eventhough every factor of $\tilde{f}$ corresponds to a factor of $f_{\bar{a},\bar{b}}(x, t)$, the order might be shuffled. But since they are all irreducible and distinct, the correct order can be found by checking for the right j so that $\tilde{f}_j(x, t, 0) = f'_i(x, t)$. □

The probability can also be amplified by trying out many projections $\{\bar{a}, \bar{b}\}$ and choosing one that yields the coarsest factorization of $f_{\bar{a},\bar{b}}(x, t)$.

14.1 *Proof of Hilbert's Irreducibility theorem*

We would need the following standard fact that a non-zero polynomial cannot be zero too often.

This is called the Schwartz Lemma or the Schwartz-Zippel Lemma or the Schwartz-Zippel-DeMillo-Lipton Lemma or Øre's Lemma.

Lemma 14.3 (Øre,Schwartz,Zippel,DeMillo-Lipton)**.** *If f is a non-zero polynomial of degree at most d and $S \subseteq \mathbb{F}$ is a finite set then*

$$\Pr_{\bar{a} \in S^n}[f(\bar{a}) = 0] \leq \frac{d}{|S|}.$$

We are given that a random bivariate projection $f_{\bar{a},\bar{b}}(x, t)$ is reducible very often and we would like to show that f must be reducible. As a warm-up, we will first show that if $f_{\bar{0},\bar{b}}(x)$ is not square-free too often, then f cannot be square-free.

Not square-free too often implies reducible

Lemma 14.4. *Suppose f is a polynomial monic in x and say $\deg(f) \leq d$. If it is the case that $\Pr[f(x, b_1, \ldots, b_n)$ not square-free $] \geq \frac{2d^2}{|S|}$ then $f(x, y_1, \ldots, y_n)$ is reducible.*

Proof. Note that $f(x, b_1, \ldots, b_n) \in \mathbb{F}[x]$ is a univariate polynomial in x. Consider the polynomial $R(\bar{(y)}) := \mathrm{Res}_x\left(f, \frac{\partial f}{\partial x}\right) \in \mathbb{F}[\bar{y}]$. For every $\bar{b}$ such that $f(x, b_1, \ldots, b_n)$ is not square-free, we know that $R(\bar{b})) = 0$. This implies that $R(\bar{y})$ is a polynomial of degree at most $2d^2$ but is zero with probability more than $2d^2/|S|$ on S^n and hence by Lemma 14.3 we must have that $R(y) = 0$.

This then implies that f and $\frac{\partial f}{\partial x}$ have a non-trivial gcd in $\mathbb{F}(\bar{y})[x]$, which by Gauss's Lemma (Theorem 12.2) means that they have a non-trivial gcd in $\mathbb{F}[x, \bar{y}]$. This of course implies that $f(x, \bar{y})$ not just reducible, but in fact not square-free. □

Therefore, we may assume that there are many points in $\bar{b} \in S^n$ where $f(x, b_1, \ldots, b_n)$ is square-free (for otherwise, Lemma 14.4 already shows that f is reducible). Without loss of generality, let us assume that $f(x, 0, \ldots, 0)$ is square-free. This essentially allows us to forget the $\bar{b}$ and instead work with $f(x, a_1 t, \ldots, a_n t)$. The problem therefore has reduced to the following — we are

given a monic (in x) polynomial $f(x, y_1, \ldots, y_n)$ of degree at most d such that $f(x, 0, \ldots, 0)$ is square-free and also

$$\Pr_{\bar{a} \in S^n} [f(x, a_1 t, \ldots, a_n t) \text{ is reducible}] > \frac{O(d^5)}{|S|},$$

and we want to conclude that f must be reducible.

The main sub-case: $f(x, 0, \ldots, 0)$ is square-free

Define the following auxiliary polynomials which will be useful in arguing the reducibility of f

$$p(x, t, y_1, \ldots, y_n) := f(x, a_1 t, \ldots, a_n t).$$

For every $\bar{a} \in \mathbb{F}^n$ define a bivariate polynomial

$$f_{\bar{a}}(x, t) := f(x, a_t, \ldots, a_n t).$$

Notice that $p(x, t, a_1, \ldots, a_n) = f_{\bar{a}}(x, t)$ and $p(x, 1, y_1, \ldots, y_n) = f(x, y_1, \ldots, y_n)$. The following lemma says that it suffices to show that $p(x, t, \bar{y})$ is reducible.

Claim 14.5. *If $p(x, t, \bar{y})$ is reducible, then $f(x, \bar{y})$ is reducible.*

Proof. If p is reducible, then p can be written as

$$p(x, t, \bar{y}) = A(x, t, \bar{y}) \cdot B(x, t, \bar{y})$$

Recall that $f(x, \bar{y}) = p(x, 1, \bar{y})$. Therefore $f(x, \bar{y}) = A(x, 1, \bar{y}) \cdot B(x, 1, \bar{y})$. The only thing to be checked is if $A(x, 1, \bar{y})$ or $B(x, 1, \bar{y})$ becomes a constant. But this cannot happen as p is also monic in x and hence so are A and B. Therefore $A(x, 1, \bar{y}), B(x, 1, \bar{y})$ are non-constants and f is reducible. □

Hensel lifting a bivariate factorization

Just like in section 12, we would be using Hensel lifting heavily. The first step is to get better factorizations for $f_{\bar{a}}(x, t)$. We start with

$$f_{\bar{a}}(x, t) = g_0(x) \cdot h_0(x) \bmod t$$

where $g_0(x)$ is monic in x and is coprime with $h_0(x)$. This can be done as we know $f_{\bar{a}}(x, t) \bmod t = f(x, 0, \ldots, 0)$ which is square-free. Starting from this solution, we apply Hensel lifting repeatedly to get

$$\begin{aligned} f_{\bar{a}}(x, t) &= g_0(x) \cdot h_0(x) \bmod t \\ &\vdots \\ f_{\bar{a}}(x, t) &= g_{k,\bar{a}}(x, t) \cdot h_{k,\bar{a}}(x, t) \bmod t^k, \end{aligned}$$

where $g_{k,\bar{a}}(x, t)$ is monic in x, and again we shall lift until $k > 2d^2$.

At the face of it, if we were to start with a different choice of $\bar{a}$, the polynomial $g_{k,\bar{a}}(x,t)$ could be completely different. The following lemma says that the different $g_{k,\bar{a}}(x,t)$s are globally correlated.

Lemma 14.6. *There is a polynomial $g_k(x,t,\bar{y})$ such that for every $\bar{a}$ we have $g_{k,\bar{a}}(x,t) = g_k(x,t,\bar{a})$).*

Proof. Consider obtaining factorizations of $f(x,y_1,\cdots,y_n)$ via Hensel Lifting, by starting with

$$f(x,y_1,\ldots,y_n) = g_0(x)\cdot h_0(x) \bmod \langle y_1,\cdots,y_n\rangle.$$

We have the same start as $f(x,y_1,\ldots,y_n) \bmod \langle y_1,\cdots,y_n\rangle = f(x,0,\ldots,0)$. Apply Hensel lifting here gives

$$f(x,y_1,\ldots,y_n) = g'_k(x,\bar{y})\cdot h'_k(x,\bar{y}) \bmod \langle y_1,\cdots,y_n\rangle^k.$$

Bear with the $'$ on the RHS; that's only because these are not quite the g_k we are after. What the above equation means is that if we consider the polynomial

$$f(x,y_1,\ldots,y_n) - g'_k\cdot h'_k,$$

then every monomial in this polynomial has y-degree at least k. Therefore, if we scale the y-variables by t

$$f(x,ty_1,\ldots,ty_n) - g'_k(x,ty_1,\ldots,ty_n)\cdot h'_k(x,ty_1,\ldots,ty_n)$$

now is divisible by t^k. In other words,

$$\begin{aligned} f(x,ty_1,\ldots,ty_n) &= g'_k(x,ty_1,\ldots,ty_n)\cdot h'_k(x,ty_1,\ldots,ty_n) \bmod t^k \\ \implies f_{\bar{a}}(x,t) &= g'_k(x,ta_1,\ldots,ta_n)\cdot h'_k(x,ta_1,\ldots,ta_n) \bmod t^k \\ &:= g_k(x,t,\bar{a})h_k(x,t,\bar{a}) \bmod t^k \end{aligned}$$

where $g_k(x,t,\bar{y}) := g'_k(x,ty_1,\cdots,ty_n)$. Observe that $g_k(x,t,\bar{a})$ is also monic in x, and is equal to $g_0(x) \bmod t$ and we also have a lifted equation

$$f_{\bar{a}}(x,t) = g_{k,\bar{a}}(x,t)\cdot h_{k,\bar{a}}(x,t) \bmod t^k.$$

By the uniqueness of Hensel Lifting (Theorem 12.1) we have that $g_{k,\bar{a}}(x,t) = g_k(x,t,\bar{a})$ for all $\bar{a}$. □

Reconstructing a factor of $p(x,t,\bar{y})$

Like in Algorithm 30, the last step would be to reconstruct a factor of $p(x,t,\bar{y})$ from a sufficiently lifted factor $g_k(x,t,\bar{y})$. As in the bivariate case, we would like to build a system of equations that should have a solution if $p(x,t,\bar{y})$ is reducible.

We shall interpret $g_k(x,t,\bar{y})$ as a bivariate polynomial, by thinking of it as an element of $\mathbb{F}(\bar{y})[x,t]$. The system of equations solves for a possible $\tilde{f}(x,t),\tilde{\ell}(x,t) \in$

$\mathbb{F}(\bar{y})[x,t]$ such that $\tilde{f}(x,t) = g_k \cdot \tilde{\ell}(x,t) \bmod t^k$. The indeterminates would be the coefficients of $\tilde{f}$ and $\tilde{\ell}$, where

$$\tilde{f}(x,t) = \sum_{i=0}^{\deg_x(p)-1} \sum_{j=0}^{\deg_t(p)} \alpha_{i,j} x^i y^j$$

$$\tilde{\ell}(x,t) = \sum_{i=0}^{\deg_x(p)-\deg_x(g_k)-1} \sum_{j=0}^{k-1} \beta_{i,j} x^i y^j$$

The number of interminates is $m = O(kd) = O(d^3)$ and so is the number of constraints is also $O(d^3)$ (but bigger than m). Therefore, this entire system can be represented as a matrix equation

$$M(\bar{y}) \cdot \left(\bar{\alpha}\ \bar{\beta}\right)^T = \bar{0}$$

where M is an $O(m) \times m$ matrix made up of polynomials in $\bar{y}$. At the moment, it is completely unclear as to why there should even be a solution over $\mathbb{F}(y_1, \ldots, y_n)$. If we can show that $\operatorname{rank}_{\mathbb{F}(\bar{y})}(M(\bar{y})) < m$, then we would at least have some solution over $\mathbb{F}(y_1, \ldots, y_n)$.

Claim 14.7. *If* $\Pr_{\bar{a}}[f_{\bar{a}}(x,t)$ *is reducible*$] > O(d^5)/|S|$*, then* $\operatorname{rank}_{\mathbb{F}(\bar{y})}(M(\bar{y})) < m$.

Proof. Suppose not, then there is some $m \times m$ submatrix of M that has non-zero determinant. Let's call this submatrix $M'(\bar{y})$.

For every $\bar{a}$ such that $f_{\bar{a}}(x,t)$ is reducible, if we were to write down the equations for

$$\tilde{f}_a(x,t) = g_k(x,t,\bar{a}) \cdot \tilde{\ell}_{k,a}(x,t) \bmod t^k$$

with the same degree constriants etc, we have the matrix equation

$$M(\bar{a}) \cdot \left(\bar{\alpha}\ \bar{\beta}\right)^T = \bar{0}$$

and since $f_{\bar{a}}(x,t)$ was reducible, this equation does indeed have a solution. But then, this means that $\det(M'(\bar{a})) = 0$ as we know that $\operatorname{rank} M(\bar{a}) < m$. But $\det(M'(\bar{y}))$ is a polynomial of degree at most $O(km) = O(d^5)$ and somehow is zero with probability more than $O(d^5)/|S|$. Therefore, by Lemma 14.3, we must have $\operatorname{rank} M(y) < m$. □

Since $\operatorname{rank}(M(\bar{y})) < m$, there exists a solution in $\mathbb{F}(y_1, \ldots, y_n)[x,t]$ for $\tilde{f}(x,t)$ and $\tilde{\ell}(x,t)$ but these involve coefficients that have denominators as polynomials in $\bar{y}$. But we are working with a homogeneous system of equations and hence we can clear out all denominators to get $\tilde{f}(x,t,\bar{y})$ and $\tilde{\ell}(x,t,\bar{y})$ that are indeed polynomials in $\mathbb{F}[x,t,\bar{y}]$, with $\deg_x \tilde{f} < \deg_x(p) \le d$ and $\deg_t \tilde{f} \le \deg_t(p) \le d$.

As in Claim 12.8, we can once again show that $\gcd(p, \tilde{f}) \neq 1$ and this gcd must be a proper factor of p as $\deg_x \tilde{f} < \deg_x p$. This shows that $p(x,t,\bar{y})$ is reducible, and hence $f(x,\bar{y})$ is reducible (Claim 14.5), thus completing the proof

of Theorem 14.1. □(Theorem 14.1)

Part III: Very Basic Algebraic Geometry

This part of the course is not going to be scribed but below is just a list of topics covered in each lecture. Hopefully the notes here would be completed in one of the future offerings of this course.

- Lecture 21: Introduction to varieties, existence of Gröbner bases, and Buchberger's algorithm
- Lecture 22: Analysis of Buchberger's algorithm (partial; rest left as exercise) and proof of Hilbert's Nullstellensatz.
- Lecture 23: Explicit degree bounds for ideal membership, and geometry of elimination.
- Lecture 24: Quantifier elimination, introduction to ideal-variety correspondence, computing ideal intersections
- Lecture 25: Ideal variety correspondence, irreducible varieties and prime ideals, prime decomposition of a radical ideal
- Lecture 26: Hilbert polynomials and the dimension of a variety

List of scribes

- Lecture 1, 2: Ramprasad Saptharishi
- Lecture 3: Somnath Chakraborty
- Lecture 4: Anamay Tengse
- Lecture 5: Prerona Chatterjee
- Lecture 6: Suhail Sherif
- Lecture 7: Gunjan Kumar
- Lecture 8: Siddharth Bhandari
- Lecture 9: Prerona Chatterjee
- Lecture 10/11: Tulasi mohan Molli
- Lecture 12: Ramprasad Saptharishi
- Lecture 13: Kshitij Gajjar
- Lecture 14: Somnath Chakraborty
- Lecture 15: Suhail Sherif
- Lecture 16: Prerona Chatterjee
- Lecture 17: Anamay Tengse
- Lecture 18: Ramprasad Saptharishi
- Lecture 19: Siddharth Bhandari
- Lecture 20: Tulasi mohan Molli & Ramprasad Saptharishi