27. Algebraic Approaches

Goal: Introduce techniques for proving upper bounds that have algebraic flavor.

27.1 Principle of Inclusion-Exclusion

Goal:
- Count objects from a universe $\mathcal{U}$ that satisfy requirements $A_1, \dots, A_n$.
  $\Rightarrow$ Is hard!
- Translate the problem of computing

$$\left| \bigcap_{i=1}^{n} A_i \right|$$

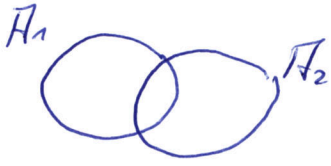
into 2^n computations of the form

$$\left| \bigcap_{i \in X} (\mathcal{U} \setminus A_i) \right|$$

that are easy (polynomial time).

Overall: Compute intersection (union) by means of other operations.

Idea (for the dual version):



$$|A_1 \cup A_2| = |A_1| + |A_2| - \underbrace{|A_1 \cap A_2|}_{\text{the intersection was counted twice}}$$

$$|A_1 \cup A_2 \cup A_3| = |A_1| + |A_2| + |A_3| - |A_1 \cap A_2| - |A_1 \cap A_3| - |A_2 \cap A_3| + \underbrace{|A_1 \cap A_2 \cap A_3|}_{\text{subtracted } A_1 \cap A_2 \cap A_3 \text{ three times}}$$

Theorem (Principle of inclusion-exclusion, union version):

Let $A_1, \dots, A_n$ be finite sets.

Then
$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{\substack{X \subseteq [1,n] \\ X \neq \emptyset}} (-1)^{|X|+1} \left|\bigcap_{i \in X} A_i\right|. \qquad (*)$$

Proof:

Let $e \in \bigcup_{i=1}^{n} A_i$.

Let $A_{i_1}, \dots, A_{i_k}$ be the sets that contain e.

Then for $X \subseteq [1,n]$ with $X \neq \emptyset$ we have

$$e \in \bigcap_{i \in X} A_i \text{ if and only if } X \subseteq \{i_1, \dots, i_k\}. \qquad (**)$$

Hence, what e contributes to the right-hand side of $(*)$ is
$$\sum_{\substack{X \subseteq [1,n] \\ X \neq \emptyset}} (-1)^{|X|+1} \left|\prod_{i \in X} A_i \cap \{e\}\right|$$

$$(\text{By } (**)) = \sum_{\substack{X \subseteq \{i_1, \dots, i_k\} \\ X \neq \emptyset}} (-1)^{|X|+1} \cdot 1$$

$$(\text{Only size of } X \text{ matters, at most } k) = (-1) \sum_{j=1}^{k} \binom{k}{j} (-1)^j$$

$$(\text{Binomial theorem}) = (-1)\left((-1+1)^k - 1\right)$$

$$= 1.$$

Recall that the Binomial theorem states that
$$(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^i b^{n-i}.$$

Hence,

$$(-1+1)^k = \sum_{j=0}^{k} \binom{k}{j} (-1)^j \cdot 1^{k-j}$$

$$= 1 + \sum_{j=1}^{k} \binom{k}{j} (-1)^j .$$

□

Theorem (Principle of inclusion-exclusion, intersection version):
Let $A_1, \dots, A_n$ be finite sets.
Then

$$\left| \bigcap_{i=1}^{n} A_i \right| = \sum_{X \subseteq [1,n]} (-1)^{|X|} \left| \bigcap_{i \in X} (U \setminus A_i) \right|,$$

where $\bigcap_{i \in \emptyset} (U \setminus A_i) := U$.

27.2 Hamiltonian Cycle

HAMILTONIAN:

Given: Directed graph $G = (V, E)$.

Parameter: $n := |V|$ (n is usually the number of edges in a graph).

Question: Does G contain a cycle that visits every vertex exactly once?

Goal: • Even count the number of Hamiltonian cycles.

• Algorithm runs in $2^n \cdot \text{poly}(n)$-time
(can also be achieved with dynamic programming).

• Algorithm uses polynomial space
(dynamic programming does not achieve this).

Theorem:
The number of Hamiltonian cycles in an n-vertex directed graph can be computed in $2^n \cdot \text{poly}(n)$ time and polynomial space.

Proof:

Fix an arbitrary vertex v_0 in G.

Let $\mathcal{U}$ be the set of all cycles of length n of the form

$$v_0 v_1 \dots v_n \quad \text{with } v_n = v_0.$$

For every vertex v, define the requirement

$$\mathcal{A}_v := \text{set of all cycles of length } n \subseteq \mathcal{U} \text{ that}$$

- start in v_0 (as required by $\mathcal{U}$)
- and visit v.

Our goal:

can now be rephrased as computing

$$\left| \bigcap_{v \in V(G)} \mathcal{A}_v \right|.$$

The intersection $\bigcap \mathcal{A}_v$ contains all cycles of length n that visit all n vertices

$\Rightarrow$ and thus every vertex exactly once.

By the inclusion-exclusion principle:

Compute for every $X \subseteq V(G)$ the value

$$\left| \bigcap_{v \in X} (\mathcal{U} \setminus \mathcal{A}_v) \right|.$$

This is the number of cycles of length n from v_0 that do not visit any vertex in X.

Hence, this is the number of cycles of length n from v_0 in the new graph $G' := G \setminus X$.

□

Easy to compute in polynomial time:

Let M be the adjacency matrix of G'.

Compute M^n.

Read off entry (v_0, v_0).

(Recall that entry (v,v') of M^n contains the number of paths of length n from v to v'.)

Altogether: 2^n computations in polynomial time, one for each set X.

Careful:

So far, we have only checked that the number of arithmetic operations is $2^n \cdot \text{poly}(n)$.

What is the cost of each arithmetic operation?

Every number that appears is bounded by $O(n^n)$, the number of paths of length n.

The operations thus act on $O(n \log n)$ bits and take polynomial time.

Moreover, they require $O(n^2)$ space. □

27.2 Steiner Tree

STEINERTREE:

Given: Undirected graph $G=(V,E)$, set of vertices $K \subseteq V$ called terminals, number $\ell \in \mathbb{N}$.

Question: Is there a tree $H \subseteq G$ with at most ℓ edges that connects the terminals?

Theorem: STEINERTREE can be solved in time $2^{|K|} \cdot \text{poly}(n)$ and polynomial space.

Idea: Before: Counting Hamiltonian paths/cycles is hard, counting ordinary paths/cycles is easy.

Here: Counting trees is hard.
Count branching walks instead.

Definition:
Let $G = (V, E)$ be a graph.

- A branching walk in G is a pair
$$B = (H, h),$$
where H is an ordered, rooted tree
and $h : V(H) \to V(G)$ is a homomorphism
$$(xy \in E(H) \Rightarrow h(x)h(y) \in E(G)).$$

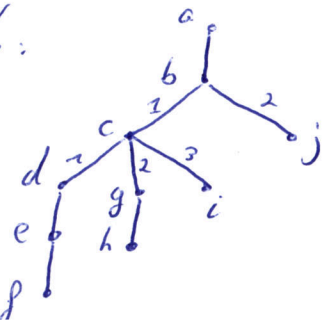
- B is a branching walk from s, if $h(r) = s$, with r the root of H.

- $V(B) := h(V(H))$.

- The length of B is $|E(H)|$.

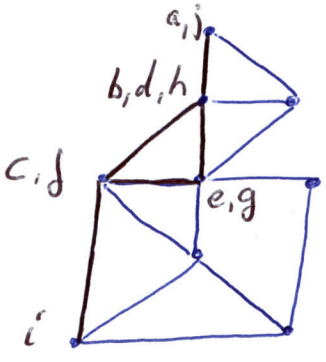
Note:
$h(H)$ is a tree iff h is injective.

Example:



G –

$\rightsquigarrow$



Observation:

Fix a terminal set K.

Graph G contains a tree H
with $K \subseteq V(H)$ and $|E(H)| \leq \ell$

if and only if

G contains a branching walk $B = (H, h)$ from s
such that $K \subseteq V(B)$ and $|E(H)| = \ell$.

Goal: Count branching walks of length ℓ from s
that visit all terminals.
Return true if and only if this number is non-zero.

Our universe $\mathcal{U}$ is the set of all branching walks
of length ℓ from s.

Requirement $R_v := \{B \in \mathcal{U} \mid v \in V(B)\}$.

Hence, our goal is to compute

$$\left| \bigcap_{v \in K} R_v \right|.$$

By the inclusion-exclusion principle,
compute for every $X \subseteq K$ the value $\left| \bigcap_{v \in X} \mathcal{U} \setminus R_v \right|$.

Note, $\bigcap_{v \in X} \mathcal{U} \setminus R_v$ are exactly the branching walks from $\mathcal{U}$
that avoid the vertices of X.

Lemma:

For every $X \subseteq V$, the value $|\bigcap_{v \in X} U[v]|$ can be computed using $O(mnk^3)$ (edges) arithmetic operations.

Proof:

Define $G' := G \setminus X$.

For $a \in V(G')$ and $j \in \{0, \dots, k\}$ let

$b_j(a) :=$ number of length j branching walks from a in G'.

Our goal is to compute $b_k(s)$ (where we return 0 if $s \notin V(G')$).

Use dynamic programming based on the following recursion

$$b_j(a) := \begin{cases} 1, & \text{if } j = 0 \\ \sum_{c \in \text{Neigh}_{G'}(a)} \sum_{j_1 + j_2 = j-1} b_{j_1}(c) \cdot b_{j_2}(a), & \text{else.} \end{cases}$$

For every neighbor c of a, we count the number of branching walks where c is the first child of a.

Note that ordered-ness allows us to just sum up the numbers of branching walks.

- For a fixed c, we consider all possibilities for the size j_1 of the subtree rooted at c.
  Number j_2 is for the remaining edges from a.
  Note that we subtracted one for the edge ac.

*-

- Note that considering branching walks instead of trees pays off precisely in this formula:
  We do not require the branching walks from a and c to be vertex disjoint.
  Hence, we can combine any two of them,
  and thus multiply $b_{j_1}(c)$ and $b_{j_2}(a)$.

- Dynamic programming computes the values $b_j(a)$ bottom-up.
  There are n neighbors,
  and every number j can be split in k^2 many ways.
  The table has $n \cdot k$ entries that have to be filled. □

The numbers have $O(\log k + \log n + \log n)$ bits
and thus the arithmetic operations take polynomial time.

Theorem:
STEINERTREE can be solved in time $2^{|K|} \cdot \text{poly}(\text{input})$
and in polynomial space.

17.3 Chromatic Number

A *k-coloring* of a graph G
is a function
$$c: V(G) \to [1, k]$$
where $c(u) \neq c(v)$ whenever u and v are adjacent.

CHROMATIC NUMBER
Given: Graph G, $k \in \mathbb{N}$.
Question: Does G have a k-coloring?

Note that each class of vertices in a k-coloring is an independent set (no two vertices are adjacent). Moreover, subsets of independent sets are again independent sets.

Observation:

G has a k-coloring if and only if

There is a cover of $V(G)$ by k independent sets

(there are independent sets $I_1, \dots, I_k$ with $\bigcup_{i=1}^{k} I_i = V(G)$).

Hence: Compute the number of covers of $V(G)$ by independent sets.

Universe $\mathcal{U}$:= Set of tuples $(I_1, \dots, I_k)$ of independent sets, not necessarily disjoint, nor even different.

Requirement: Need to cover every vertex

$$A_v := \{ (I_1, \dots, I_k) \in \mathcal{U} \mid v \in \bigcup_{i=1}^{k} I_i \}.$$

Then

$\left| \bigcap_{v \in V(G)} A_v \right|$ is the number of covers of G by k independent sets.

- Define $s(Y)$:= number of independent sets in $G[Y]$ (subgraph of G induced by Y).

- By inclusion-exclusion, computing $\left| \bigcap_{v \in V(G)} A_v \right|$

reduces to computing, for every $X \subseteq V(G)$,
the value

$$|\{(I_1, \dots, I_k) \in \mathcal{U} \mid I_1, \dots, I_k \subseteq V(G) \setminus X\}| = s(V(G) \setminus X)^k.$$

- We do not know how to compute $s(V(G) \setminus X)^k$
in polynomial time.
But $s(Y)$ can be computed simultaneously for all $Y \subseteq V(G)$
using dynamic programming and the formula:

$$s(Y) := \underbrace{s(Y \setminus \{y\})}_{\text{independent sets without } y} + \underbrace{s(Y \setminus Neigh(y))}_{\text{independent sets with } y},$$

where y is an arbitrary but fixed element from Y.

- Computing these values takes
$O(2^n)$ arithmetic operations
on $O(n)$-bit numbers,
and takes space $2^n \cdot poly(input)$.

- Once $s(Y)$ has been computed
for all subsets $Y \subseteq V(G)$ // up front
the values $s(V(G) \setminus X)^k$ can be determined
in polynomial time
by $O(\log k)$ multiplications
of $O(nk)$-bit numbers.

Theorem: CHROMATIC NUMBER can be solved in time and space $2^n \cdot \text{poly}(\text{input})$.

Remark:

- A $2^n \cdot \text{poly}(\text{input})$-time algorithm that uses polynomial space is currently not known.
- Let us trade running time for space efficiency.
  - ↳ Cannot afford to tabulate the number of independent sets $s(A)$ for all $A \subseteq V(G)$.
  - ↳ Instead, compute $s(A)$ by recursion, without storing intermediary values.
    Takes time $2^{|A|} \cdot \text{poly}(\text{input})$ for each subset $A \subseteq V(G)$.
- This yields a running time of

$$\sum_{X \subseteq V(G)} \left(2^{|V(G) \setminus X|} \text{poly}(\text{input}) \right)$$

$$= \left(\sum_{X \subseteq V(G)} 2^{|V(G) \setminus X|} \right) \text{poly}(\text{input})$$

(Only size of X matters) $= \left(\sum_{k=0}^{n} \binom{n}{k} 2^{n-k} \, 1^k \right) \text{poly}(\text{input})$

(Binomial theorem) $= 3^n \cdot \text{poly}(\text{input})$.

- There are better branching algorithms for finding the number of independent sets.

The currently best known algorithm take $1{,}238^n \cdot \text{poly}(\text{input})$.
With the Binomial theorem, we arrive at the following.

Theorem:
CHROMATIC NUMBER can be solved
in time $2{,}238^n \cdot \text{poly}(\text{input})$ and polynomial space.