EECS 336: Introduction to Algorithms **Lecture 14**
P vs. NP intractability, NP, decision problems

Reading: 8.0-8.3

Last time:

- max flow alg / ford-fulkerson
- duality: max flow = min cut

Today:

- tractibility and intractibility
- P and NP
- decision problems
- INDEP-SET, 3-SAT, TSP, NP, CIRCUIT-SAT

Intractibility and NP-completeness

"when is a problem intractable?"

Def: $\mathcal{P}$ is the class of problems that can be solved in polynomial time.

$X \in \mathcal{P}$ iff

$\exists$ polynomial $p(\cdot)$,

$\exists$ alg $\mathcal{A}$,

$\forall$ instances x of X,

$\Rightarrow$ $\mathcal{A}$ solves x and in time $O(p(|x|))$

Note: easy to show $X \in \mathcal{P}$, just give $\mathcal{A}$ and prove poly runtime.

Examples: network-flow, matching, interval scheduling, etc.

Three Infamous Problems

Problem 1: Independent Set (INDEP-SET)

input: $G = (V, E)$

output: $S \subset V$

- satisfying $\forall v \in S,\ (u, v) \notin E$
- maximizing $|S|$

Problem 2: Satisfiability (SAT)

input: boolean formula $f(\mathbf{z})$

e.g., $f(\mathbf{z}) = (z_1 \vee \bar{z}_2 \vee x_3) \wedge (z_2 \vee \bar{z}_5 \vee z_6) \wedge \cdots$

output:

- "Yes" if assignment $\mathbf{z}$ with $f(\mathbf{z}) = T$ exists

  e.g., $\mathbf{z} = (T, T, F, T, F, \ldots)$

- "No" otherwise.

Problem 3: Traveling Salesman (TSP)

input:

- $G = (V, E)$, complete graph.
- $c(\cdot)$ = costs on edges.

output: cycle C that

- passes through all vertices exactly once.
- minimizes total cost $\sum_{e \in C} c(e)$.

No polynomial time algorithm is known for any of these problems!

Theory of Intractability

Goal: formal way to argue that no polynomial time algorithm exists (or "unlikely to exist"), i.e., $X \notin \mathcal{P}$.

Challenge: must show that all algorithms fail!

Idea: to show X is difficult, reduce notoriously hard problem Y to X, i.e., <u>reduce from Y</u>.

Example: to show new problem X is hard, e.g., reduce TSP to X, i.e., reduce from TSP.

Def: Y reduces to X in polynomial time (notation: $Y \leq_{\mathcal{P}} X$ if any instance of Y can be solved in a polynomial number of computational steps and a polynomial number of calls to black-box that solves instances of X.

Consequences of $Y \leq_{\mathcal{P}} X$:

1. if X can be solved in polynomial time then so can Y.

   > Example: X = network-flow; Y = bipartite matching.

2. if Y cannot be solved in polynomial time then neither can X.

Decision Problems

Goal: show SAT, INDEP-SET, TSP equivalently hard.

Challenge: SAT, INDEP-SET, TSP problem solutions are very different.

Idea: focus on decision version of problem.

Def: A decision problem asks "does a feasible solution exist?"

Example: satisfiability.

Def: an optimization problem asks "what is the min (or max) value of a feasible solution?"

Def: the decision problem X_d for optimization problem X is has input (x, D) = "does instance x of X have a feasible solution with value at most (or at least) D?"

Examples:

INDEP-SET$_d$: set S with $|S| \geq D$

SAT$_d$: $\mathbf{z}$ such that $f(\mathbf{z}) = T$.

TSP$_d$: tour C with $\sum_{c \in C} c(e) \leq D$

Deciding is as hard as optimizing

Theorem: $X \leq_{\mathcal{P}} X_d$

Proof: (reduction via binary search)

- given
  - instance x of X
  - black-box $\mathcal{A}$ to solve X_d
- search(A, B) = find optimal value in $[A, B]$.
  - $D = (A + B)/2$
  - run $\mathcal{A}(x, D)$
  - if "yes", search(A, D)
  - if "no", search(D, B)

Finding solution is as hard as deciding

Example: satisfiability

1. if f is satisfiable $\exists \mathbf{z}$ s.t. $f(\mathbf{z}) = T$
2. guess $z_n = T$
3. let $f'(z_1, .., z_{n-1}) = f(z_1, .., z_{n-1}, T)$
4. if f' is satisfiable, repeat (2) on f'
5. if f' is unsatisfiable, repeat (2) on $f''(z_1, \ldots, z_{n-1}) = f(z_1, \ldots, z_{n-1}, F)$.

Note: since $X_d =_{\mathcal{P}} X$, we write "X" but we mean "X_d"

A notoriously hard problem

Note: all example problem have <u>short certificates</u> that could easily verify "yes" instance.

Def: $\mathcal{NP}$ is the class of problems that have short (polynomial sized) certificates that can easily (in polynomial time) verify "yes" instances.

Historical Note: $\mathcal{NP}$ = <u>non-deterministic polynomial time</u>

"a nondeterministic algorithm could guess the certificate and then verify it in polynomial time"

Note: Not all problems are in $\mathcal{NP}$.

E.g., unsatisfiability.

Def:

- Problem <u>X is in $\mathcal{NP}$</u> if exists short easily-verifiable certificate.
- Problem <u>X is $\mathcal{NP}$-hard</u> if $\forall Y \in \mathcal{NP}$, $Y \leq_{\mathcal{P}} X$.
- Problem <u>X is $\mathcal{NP}$-complete</u> if $X \in \mathcal{NP}$ and X is $\mathcal{NP}$-hard.

Lemma: INDEP-SET $\in \mathcal{NP}$.

Lemma: SAT $\in \mathcal{NP}$.

Lemma: TSP $\in \mathcal{NP}$.

Goal: show INDEP-SET, SAT, TSP are $\mathcal{NP}$-complete.

Notorious Problem: NP

input:

- decision problem verifier program VP.
- polynomial $p(\cdot)$.
- decision problem instance: x

output:

- "Yes" if exists certificate c such that $VP(x,c)$ has "verified = true" at computational step $p(|x|)$.
- "No" otherwise.

Fact: NP is $\mathcal{NP}$-complete.

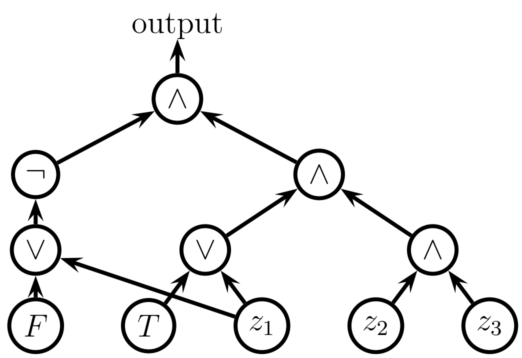
Note: Unknown whether $\mathcal{P} = \mathcal{NP}$.

Note: $\leq_{\mathcal{P}}$ is transitive: if $Y \leq_{\mathcal{P}} X$ and $X \leq_{\mathcal{P}} Z$ then $Y \leq_{\mathcal{P}} Z$.

Plan: NP $\leq_{\mathcal{P}}$ CIRCUIT-SAT $\leq_{\mathcal{P}}$ SAT.

Circuit Satisfiability

Example:



Problem 4: CIRCUIT-SAT

input: boolean circuit $Q(\mathbf{z})$

- directed acyclic graph $G = (V, E)$
- internal nodes labeled by logical gates:

  "and", "or", or "not"

- leaves labeled by variables or constants

  $T, F, z_1, \ldots, z_n$.

- root r is output of circuit

output:

- "Yes" if exists $\mathbf{z}$ with $Q(\mathbf{z}) = T$
- "No" otherwise.

Lemma: CIRCUIT-SAT is $\mathcal{NP}$-hard.

Proof: (reduce from NP)

- goal: convert NP instance (VP, p, x) to CIRCUIT-SAT instance Q
- $VP(\cdot,\cdot)$ polynomial time

  $\Rightarrow$ computer can run it in poly steps.

- each step of computer is circuit.
- output of one step is input to next step
- unroll $p(|x|)$ steps of computation

  $\Rightarrow \exists$ poly-size circuit $Q'(\mathbf{x}, \mathbf{c}) = VP(x, c)$

- hardcode $\mathbf{x}$: $Q(\mathbf{c}) = Q'(\mathbf{x}, \mathbf{c})$
- Conclusion: Q is sat iff exists c with $VP(x, c) =$ "verified".

QED