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 \( A \),
- \( \forall \) instances \( x \) of \( X \),
  \( \Rightarrow A \) solves \( x \) and in time \( O(p(|x|)) \)

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

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

Three Infamous Problems

Problem 1: Independent Set (INDEPENDENT-SET)

input: \( G = (V, E) \)

output: \( S \subseteq V \)
  - satisfying \( \forall v \in S, (u, v) \notin E \)
  - maximizing \( |S| \)

Problem 2: Satisfiability (SAT)

input: boolean formula \( f(z) \)

\( \text{e.g., } f(z) = (z_1 \lor \overline{z}_2 \lor x_3) \land (\overline{z}_2 \lor \overline{z}_5 \lor z_6) \land \cdots \)

Problem 3: Traveling Salesman (TSP)

input:
  - \( G = (V, E) \), complete graph.
  - \( c(\cdot) = \text{costs on edges.} \)

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., reduce from \( Y \).

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: $Y \leq_p X$ 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 $Y \leq_p X$:

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$_d$: $z$ such that $f(z) = T$.

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

Deciding is as hard as deciding

Theorem: $X \leq_p X_d$

Proof: (reduction via binary search)

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

Finding solution is as hard as deciding

Example: satisfiability

1. if $f$ is satisfiable $\exists z$ s.t. $f(z) = T$
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 $f''(z_1, .., z_{n-1}) = f(z_1, .., z_{n-1}, F)$.

Note: since $X_d =_p X$, we write “$X$” but we mean “$X_d$”
A notoriously hard problem

Note: all example problems have short certificates 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: \( \mathcal{NP} \) = non-deterministic polynomial time

“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 \( X \) is in \( \mathcal{NP} \) if exists short easily-verifiable certificate.
- Problem \( X \) is \( \mathcal{NP} \)-hard if \( \forall Y \in \mathcal{NP}, Y \leq_{p} X \).
- Problem \( X \) is \( \mathcal{NP} \)-complete if \( X \in \mathcal{NP} \) and \( X \) is \( \mathcal{NP} \)-hard.

Lemma: \( \text{INDEP-SET} \in \mathcal{NP} \).

Lemma: \( \text{SAT} \in \mathcal{NP} \).

Lemma: \( \text{TSP} \in \mathcal{NP} \).

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

Notorious Problem: \( \mathcal{NP} \)

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: \( \mathcal{NP} \) is \( \mathcal{NP} \)-complete.

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

Note: \( \leq_{p} \) is transitive: if \( Y \leq_{p} X \) and \( X \leq_{p} Z \) then \( Y \leq_{p} Z \).

Plan: \( \mathcal{NP} \leq_{p} \text{CIRCUIT-SAT} \leq_{p} \text{SAT} \).
Circuit Satisfiability

Example:

\[
\begin{array}{c}
\text{output} \\
\land \\
\neg \\
\lor \\
F \\
T \\
z_1 \\
z_2 \\
z_3
\end{array}
\]

Problem 4: CIRCUIT-SAT

input: boolean circuit \(Q(z)\)

- 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 \(z\) with \(Q(z) = T\)
- “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

\[\Rightarrow \exists\ \text{poly-size circuit } Q'(x, c) = VP(x, c)\]

- hardcode \(x\): \(Q(c) = Q'(x, c)\)
- Conclusion: \(Q\) is sat iff exists \(c\) with \(VP(x, c) = \text{“verified”}\).

QED