What is an EFFICIENT Algorithm?

• The algorithms we have studied so far have runtimes that are typically bounded by some polynomial in the size of the input

• We call these efficient algorithms

• But for many problems, no polynomial-time algorithm is known

• We suspect that many of these problems can never be solved efficiently
Boolean Satisfiability

• Conjunctive normal form (CNF) = Product of Sums

• A Boolean expression $S$ in CNF is the product (and) of several sums (or)

\[
S = (x + y + z) \cdot (\overline{x} + y + z) \cdot (x + \overline{y} + \overline{z})
\]
Boolean Satisfiability

• Conjunctive normal form (CNF) = Product of Sums

• A Boolean expression $S$ in CNF is the product (*and*) of several sums (*or*)

\[ S = (x + y + z) \cdot (\overline{x} + y + z) \cdot (x + \overline{y} + \overline{z}) \]

• A Boolean expression is *satisfiable* if there exists an assignment of 0’s and 1’s to its variables such that the expression evaluates to 1.
The Satisfiability Problem (SAT)

\[(x + y + z) \cdot (\overline{x} + y) \cdot (x + \overline{z}) \cdot (z + \overline{y}) \cdot (x + \overline{y} + \overline{z})\]

- SAT(F): Given a Boolean formula F in CNF, is there a satisfying assignment of truth values to the variables?
  - This is a “Decision” question: Answer is **YES** or **NO**
  - If F has n variables, there are \(2^n\) different truth assignments

- Answer in above case: ?
A Problem is a “Language”

- **Decision problem:** A problem whose answer is either “yes” or “no”
  - Example: \( \text{TSP}(G,k) \): Does there exist a TSP tour on graph \( G \) that has total length \( \leq k \)?

- A decision problem can be viewed as a *language-recognition problem*
  - Is \((G,k) \in \text{TSP}\)?
  - \( U \): the set of possible inputs to the decision problem
  - \( L \subseteq U \) is the set of inputs which yield “yes”
  - \( L \): the language corresponding to the problem
A Problem is a “Language”

- **Decision problem**: A problem whose answer is either “yes” or “no”
  - A decision problem can be viewed as a *language-recognition problem*
    - Is \((G,k) \in \text{TSP}\) ?
    - \(U\): the set of possible inputs to the decision problem
    - \(L \subseteq U\) is the set of inputs which yield “yes”
    - \(L\): the language corresponding to the problem
A Problem is a “Language”
A Problem is a “Language”
Polynomial-Time Reducibility

- Let $L_1$ and $L_2$ be two languages from the input spaces $U_1$ and $U_2$
- Definition: $L_1$ is **polynomially reducible** to $L_2$ if there exists a polynomial-time algorithm that converts each input $u_1 \in U_1$ to another input $u_2 \in U_2$ such that $u_1 \in L_1$ iff $u_2 \in L_2$

**Recall:**

**Reduction from Sorting to Convex Hull**
- **Input:** an arbitrary instance $I_{\text{SORT}}$ of SORT
- **Transform** $I_{\text{SORT}} = \{x_1, x_2, ..., x_n\}$ into an instance $I_{\text{C-HULL}} = \{(x_1, x_1^2), (x_2, x_2^2), ..., (x_n, x_n^2)\}$ of C-HULL
- Solve the C-HULL problem for instance $I_{\text{C-HULL}}$
- **Transform** solution of $I_{\text{C-HULL}}$ to solution of $I_{\text{SORT}}$
Polynomial-Time Reducibility

• “SORT reduces to CHULL”
  What is $u_1 \in U_1$?
  What is $u_2 \in U_2$?
  such that $u_1 \in L_1$ iff $u_2 \in L_2$
Polynomial-Time Reducibility

$L_1$ is *polynomially reducible* to $L_2$ if there exists a polynomial-time algorithm that converts each input $u_1 \in U_1$ to another input $u_2 \in U_2$ such that $u_1 \in L_1$ iff $u_2 \in L_2$.

![Diagram of polynomial-time reducibility](image)

*Note: “polynomial-time” in size of input $u_1 \Rightarrow$ size of $u_2$ is also polynomial in the size of $u_1$. (Why?)*
Polynomial-Time Reducibility

• **Definition:** X is *polynomial-time reducible* to Y, written \( X \leq_p Y \), if there exists an algorithm for solving X that would be polynomial if we took no account of the time needed to solve arbitrary instances of Y.

• **Examples:**
  – X(n): Is_It_Prime?
  – Y(n):
    Is_Largest_Factor_Other_Than_Itself_Greater_Than_Or_Equal_To_2?
  – Z(n,k): Is_Number_Of_Distinct_Factors_Greater_Than_Than_k?

**Claim 1:** \( X \leq_p Y \)

**Claim 2:** \( X \leq_p Z \)
Observations About Poly-Time Reducibility

- $L_1 \leq_P L_2$ and $L_2 \leq_P L_3 \implies L_1 \leq_P L_3$

- $L_1 \leq_P L_2$ and $L_2 \leq_P L_1 \implies L_2, L_1$ are polynomially equivalent
Observations About Poly-Time Reducibility

• $L_1 \leq_P L_2$ and $L_2 \leq_P L_3 \Rightarrow L_1 \leq_P L_3$

• $L_1 \leq_P L_2$ and $L_2 \leq_P L_1 \Rightarrow L_2, L_1$ are polynomially equivalent

• Where this is going
  – “Solvable” problems have polynomial-time algorithms
  – Some problems (TSP, SAT, etc.) are “hard” (“non-solvable”)
  – The “hard” problems have equivalent difficulty: they are all polynomially equivalent.
    • $\Rightarrow$ If we can solve any one of them in polynomial time, then we can solve them all in polynomial time
Checkpoint

• Suppose A reduces to B.

  ![Diagram showing the reduction from A to B]

  **Decider of B**

  - If B is easy → A must be easy
  - If A is easy → NOTHING
  - If A is hard → B must be hard
  - If B is hard → NOTHING

  **Decider of A**
4 "non-deterministic steps"

vs. 4! "deterministic steps"
The Classes P and NP

Classes of Decision Problems

- **P**: Problems for which there exists a deterministic polynomial-time algorithm
- **NP**: Problems for which there exists a non-deterministic polynomial-time algorithm

What is “non-deterministic polynomial-time”?
- Solutions are small (== polynomial-size)
- Guessing is free, but solutions must be checkable in polynomial time
  - “Succinct Certificate”

- P ⊆ NP, but whether P = NP is not known
  - Most people believe P ≠ NP, which would imply P ⊊ NP
NP-Hard, NP-Complete

• **NP-Hard:** Problem $X$ is NP-hard if *every* problem in NP is *polynomially reducible* to $X$

• **NP-Complete:** Problem $X$ is NP-complete if:
  – $X$ belongs to NP, and
  – $X$ is NP-hard

• Also, $Y$ is **NP-complete** if $Y \in \text{NP}$ and some NP-complete problem $X$ is *polynomially reducible* to $Y$

• **NP-Complete problems are the hardest problems in NP**
Picture of P, NP, NP-Hard, NP-Complete
Picture of P, NP, NP-Hard, NP-Complete
SAT is NP-Complete (Cook, 1971)

- **Cook’s theorem (1971):** The SAT problem is NP-complete
  - Also, Levin (1972)

- Once we know an NP-complete problem, proving other problems NP-complete becomes easier

- **A new problem Y is NP-complete if:**
  - (1) Y is in NP
  - (2) SAT or any other NP-complete problems, is *polynomially reducible to Y*
    - I.e., “reduce from SAT to Y”
NP-Complete Problems 0, 0’

- **Number Partition**: Given a set of numbers $S = \{x_1, x_2, \ldots, x_n\}$, is there a disjoint partition of $S$ into $S_1, S_2$ (i.e., where $S_1 \cup S_2 = S$, and $S_1 \cap S_2 = \emptyset$) such that the sum of numbers in $S_1 = \text{sum of numbers of } S_2$?

- **k-Tolerant Number Partition**: Given a set of numbers $S = \{x_1, x_2, \ldots, x_n\}$ and a number $k$, is there a disjoint partition of $S$ into $S_1, S_2$ (i.e., where $S_1 \cup S_2 = S$, and $S_1 \cap S_2 = \emptyset$) such that $|\text{sum of numbers in } S_1| - |\text{sum of numbers of } S_2| \leq k$?

- Task: Reduce k-Tolerant Number Partition to Number Partition
NP-Complete Problems 0, 0’

• **Number Partition:** Given a set of numbers $S = \{x_1, x_2, \ldots, x_n\}$, is there a disjoint partition of $S$ into $S_1, S_2$ (i.e., where $S_1 \cup S_2 = S$, and $S_1 \cap S_2 = \emptyset$) such that the sum of numbers in $S_1 = \text{sum of numbers of } S_2$?

• **k-Tolerant Number Partition:** Given a set of numbers $S = \{x_1, x_2, \ldots, x_n\}$ and a number $k$, is there a disjoint partition of $S$ into $S_1, S_2$ (i.e., where $S_1 \cup S_2 = S$, and $S_1 \cap S_2 = \emptyset$) such that $|\text{sum of numbers in } S_1| - |\text{sum of numbers of } S_2| \leq k$?

• **Task:** Reduce k-Tolerant Number Partition to Number Partition
Two Parts To An NP-Completeness Proof

• Problem is in NP

• A known NP-complete problem reduces to it
NP-Complete Problems 0, 0’

• Reduce $k$-Tolerant Number Partition to Number Partition
NumPart, Bin-Packing

- **NumPart(S):** Given a set of numbers $S = \{x_1, x_2, \ldots, x_n\}$ with $0 < x_i \leq 1$, there a disjoint partition of $S$ into $S_1, S_2$ (i.e., where $S_1 \cup S_2 = S$, and $S_1 \cap S_2 = \emptyset$) such that $\sum_{x_i \in S_1} = \sum_{x_j \in S_2}$?

- **Bin-Packing(L,B):** Given a list of items $L = \{x_1, x_2, \ldots, x_n\}$ with $0 < x_i \leq 1$, and a number $B$, can $L$ be packed into $B$ unit-capacity bins?

**Given that NumPart is NP-complete, prove that Bin-Packing is NP-complete**
NP-Complete Problems #1, #2, #3

The following five problems are NP-complete

(Manber, Introduction to Algorithms: A Creative Approach)

• **Definition:** A vertex cover of $G = (V, E)$ is $V' \subseteq V$ such that every edge in $E$ is incident to some $v \in V'$.

• **VC(G,k):** Given undirected $G = (V, E)$ and integer $k$, does $G$ have a vertex cover with $\leq k$ vertices?

• **CLIQUE(G,k):** Does $G$ contain a clique of size $\geq k$?

• **3COLOR(G):** Is $G$ properly colorable with 3 colors?
NP-Complete Problems #1, #2, #3

• **VC(G,k):** Given undirected $G = (V, E)$ and integer $k$, does $G$ have a vertex cover with $\leq k$ vertices?

• **CLIQUE(G,k):** Does $G$ contain a clique of size $\geq k$?

• **3COLOR(G):** Is $G$ properly colorable with 3 colors?
NP-Complete Problems #4, #5

• **Definition:** A dominating set \( D \) of \( G = (V, E) \) is \( D \subseteq V \) such that every \( v \in V \) is either in \( D \) or adjacent to at least one vertex of \( D \).

• **DS(G,k):** given \( G, k \), does \( G \) have a dominating set of size \( \leq k \)?

• **3SAT(F):** Given Boolean expression \( F \) in CNF such that each clause has *exactly* 3 variables, is \( F \) satisfiable?
All NP problems

→ SAT

Clique

→ 1

Vertex Cover

→ 2

Dominating Set

→ 4

3SAT

→ 3

3-Colorability

→ 5

To prove NP-completeness of a new problem:
– Prove that the problem belongs to NP
– Show polynomial reducibility from a known NP-complete problem

This slide shows the order in which we will develop some classic reductions
NP-Completeness Proof: Vertex Cover (VC)

• **Problem:** Given undirected $G = (V, E)$ and integer $k$, does $G$ have a vertex cover with $\leq k$ vertices?
• **Theorem:** The VC problem is NP-complete.
• **Proof:**
  – **Approach:** Reduction from CLIQUE
  – I.e., given that CLIQUE is NP-complete, we will prove that VC is NP-complete
  – **VC is in NP.** This is true since we can guess a cover of size $\leq k$ and check it in polynomial time
  – **Goal:** Transform arbitrary CLIQUE instance into VC instance such that CLIQUE answer is “yes” iff VC answer is “yes”
NP-Completeness Proof: Vertex Cover (VC)

– **Claim**: CLIQUE(G, k) has *same* answer as VC \((\overline{G}, n-k)\), where \(n = |V|\).

– **Observe**: If \(C = (u, F)\) clique in \(G\), then \(v-u\) is VC in \(\overline{G}\).
NP-Completeness Proof: Vertex Cover (VC)

– **Observe:** If D is a VC in $\overline{G}$, then $\overline{G}$ has no edge between vertices in V-D.

So, we have that a k-clique in G $\iff$ a size (n-k) VC in $\overline{G}$

– Can transform in *polynomial time*. 
NP-Completeness Proof: DS

Definition: A dominating set $D$ of $G = (V, E)$ is $D \subseteq V$ such that every $v \in V$ is either in $D$ or adjacent to at least one vertex of $D$

- **Problem**: given $G$, $k$ does $G$ have a dominating set of size $\leq k$?
- **Theorem**: DS is NP-complete
- **Proof**: (reduction from VC)
  - $DS \in NP$
  - Given instance $(G,k)$ of VC, make every edge of $G$ into a triangle and answer DS.
NP-Completeness Proof: DS

• G’ has DS D of size k iff G has VC C of size k.
  – Without loss of generality, assume $D \subseteq V$, D is a DS $\Rightarrow$ every edge hits some vertex in D $\Rightarrow$ D is a VC in G
  – C is a VC $\Rightarrow$ new and old vertices dominated
NP-Completeness Proof: 3SAT

• **3SAT**: Given a Boolean expression in CNF such that each clause has *exactly* 3 variables, determine satisfiability.

• **Theorem**: 3SAT is NP-Complete

• **Proof**:
  • $3\text{SAT} \in \text{NP}$
  • Let $E$ be an arbitrary instance of SAT
    
    $C = (x_1 + x_2 + ... + x_k)$ arbitrary clause of $E$

    For $k \geq 4$
    
    $C' = (x_1 + x_2 + y_1) \cdot (x_3 + \overline{y_1} + y_2) \cdot (x_4 + \overline{y_2} + y_3) \cdot ...
    \cdot (x_{k-1} + x_k + \overline{y_{k-3}})$
NP-Completeness Proof: 3SAT

C’ is satisfiable iff C is satisfiable:
- C’ = 1 \implies \text{at least one } x_i = 1 \implies C = 1
- C = 1 \implies \text{at least one } x_i = 1 \implies \text{can set } y_i \text{'s so that } C’ = 1

\[ C = (x_1 + x_2) \]
\[ C = (x_1 + x_2 + y) \bullet (x_1 + x_2 + \bar{y}) \]
\[ C = (x_1) \]
\[ C = (x_1 + y_1 + y_2) \bullet (x_1 + \bar{y}_1 + y_2) \bullet (x_1 + y_1 + \bar{y}_2) \bullet (x_1 + \bar{y}_1 + \bar{y}_2) \]
NP-Completeness Proof: CLIQUE

• **Problem:** Does $G=(V,E)$ contain a clique of size $k$?
• **Theorem:** Clique is NP-Complete. (reduction from SAT)
• **Idea:** Make “column” of $k$ vertices for each clause with $k$ variables.
  – No edge within a column.
  – All other edges present except between $x$ and $\bar{x}$
NP-Completeness Proof: CLIQUE

• Example: $E = (x + y + z) \cdot (\overline{x} + \overline{y} + z) \cdot (y + \overline{z})$

$G = \begin{array}{c}
\overline{z} \\
y \\
x \\
\overline{x} \\
z \\
y \\
x \\
\overline{z}
\end{array}$

• $G$ has $m$-clique (m is the number of clauses in $E$), iff $E$ is satisfiable.

(Assign value 1 to all variables in clique)