Subset Sum

- **Problem**
  - **Input:** Set of integers $S = \{w_1, \ldots, w_n\}$, integer $W$
  - **Question:** Is there $S' \subseteq S$ that adds up to exactly $W$?

- **Example**
  - $S = \{1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344\}$, $W = 3754$?
  - Yes!
    - $1 + 16 + 64 + 256 + 1040 + 1093 + 1284 = 3754$
Claim: Subset Sum is in NP

- Recall: We need to show that there is a polynomial-time algorithm which
  - Can accept every YES instance with the right polynomial-size advice
  - Will not accept a NO instance with any advice

- Advice: the actual subset $S'$
- Algorithm: check that $S'$ is indeed a subset of $S$ and sums to $W$
- Simple!
Substitution Sum

- Claim: Exact 3SAT \( \leq_p \) Subset Sum

- Given a formula \( \varphi \) of Exact 3SAT, we want to construct \((S, W)\) of Subset Sum with the same answer.

- In the table in the following slide:
  - Columns are for variables and clauses.
  - Each row is a number in \( S \), represented in decimal.
  - Number for literal \( \ell \): has 1 in its variable column and in the column of every clause where that literal appears.
    - Number selected = literal set to TRUE.
  - "Dummy" rows: can help make the sum in a clause column 4 if and only if at least one literal is set to TRUE.
Subset Sum

- Claim: Exact 3SAT $\leq_p$ Subset Sum

\[
C_1 = \overline{x} \lor y \lor z
\]
\[
C_2 = x \lor y \lor z
\]
\[
C_3 = \overline{x} \lor \overline{y} \lor \overline{z}
\]

- Decimal representation

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

W: 1 1 1 4 4 4 4
Subset Sum

• Note
  ➢ The Subset Sum instance we constructed has "large" numbers
    o Their values are exponentially large (~$10^{\#variables+\#clauses}$)
    o But the number of bits required to write them is polynomial

  ➢ Can we hope to construct Subset Sum instance with numbers whose values are only $poly(\#variables, \#clauses)$ large?
    o Unlikely, as that would prove $P = NP$!
    o Like Knapsack, Subset Sum can be solved in pseudo-polynomial time
      • That is, in polynomial time if the numbers are only polynomially large in value
3-Coloring

• Problem
  ➢ Input: Undirected graph $G = (V, E)$
  ➢ Question: Can we color each vertex of $G$ using at most three colors such that no two adjacent vertices have the same color?
3-Coloring

• Claim: 3-coloring is in NP

➢ Recall: We need to show that there is a polynomial-time algorithm which
  o Can accept every YES instance with the right polynomial-size advice
  o Will not accept a NO instance with any advice

➢ Advice: colors of the nodes in a valid 3-coloring
➢ Algorithm: check that this is a valid 3-coloring
➢ Simple!
3-Coloring

- **Claim: Exact 3SAT $\leq_p$ 3-Coloring**

  - Given an Exact 3SAT formula $\varphi$, we want to construct a graph $G$ such that $G$ is 3-colorable if and only if $\varphi$ has a satisfying assignment.
  - $G$ will have the following nodes:
    - Type 1: true, false, base, one for each $x_i$, one for each $\overline{x_i}$
    - Type 2: additional nodes for each clause $C_j$
  - 1-1 correspondence between valid 3-colorings of type 1 nodes and valid truth assignments:
    - All literals with the same color as “true” node are set to true
    - All literals with the same color as “false” node are set to false
  - **Claim:** Fix any colors of type 1 nodes. There exists a valid 3-coloring of $G$ giving these colors to type 1 nodes if and only if the corresponding truth assignment is satisfying for $\varphi$. 
3-Coloring

- Create 3 new nodes T, F, and B, and connect them in a triangle
- Create a node for each literal, connect it to its negation and to B
- T-F-B must have different colors, and so must B-\( x_i - \overline{x_i} \)
  - Each literal has the color of T or F; its negation has the other color
  - Valid 3-coloring \( \iff \) valid truth assignment (set all with color T to true)
3-Coloring

- We also need valid 3-coloring $\iff$ satisfying truth assignment
  - For each clause, add the following gadget with 6 nodes and 13 edges
  - **Claim:** Clause gadget is 3-colorable $\iff$ at least one of the nodes corresponding to the literals in the clause is assigned color of T
3-Coloring

➢ **Claim:** Valid 3-coloring $\Rightarrow$ truth assignment satisfies $\varphi$
  o Suppose a clause $C_i$ is not satisfied, so all its three literals must be F
  o Then the 3 nodes in top layer must be B
  o Then the first two nodes in bottom layer must be F and T
  o No color left for the remaining node $\Rightarrow$ contradiction!
3-Coloring

➢ We just proved: valid 3-coloring $\Rightarrow$ satisfying assignment
➢ **Claim:** satisfying assignment $\Rightarrow$ valid 3-coloring
  o Each clause has at least one literal with color T
  o **Exercise:** Regardless of which literal has color T and which color (T/F) the other literals have, the clause widget can always be 3-colored
Review of Reductions

• If you want to show that problem B is NP-complete

• **Step 1: Show that B is in NP**
  - Some polynomial-size advice should be sufficient to verify a YES instance in polynomial time
  - No advice should work for a NO instance

  - Usually, the solution of the “search version” of the problem works
    - But sometimes, the advice can be non-trivial
      - For example, to check LP optimality, one possible advice is the values of both primal and dual variables, as we saw in the last lecture
Review of Reductions

• If you want to show that problem B is NP-complete

• Step 2: Find a known NP-complete problem A and reduce it to B (i.e., show $A \leq_p B$)
  ➢ This means taking an arbitrary instance of A, and solving it in polynomial time using an oracle for B
    o Caution 1: Remember the direction. You are “reducing known NP-complete problem to your current problem”.
    o Caution 2: The size of the B-instances you construct should be polynomial in the size of the original A-instance
  ➢ This would show that if B can be solved in polynomial time, then A can be as well
  ➢ Some reductions are trivial, some are notoriously tricky...
Binary Integer Linear Programming (BILP)

- **Problem**
  - **Input:** $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$, $k \in \mathbb{R}$
  - **Question:** Does there exist $x \in \{0,1\}^n$ such that $c^T x \geq k$ and $Ax \leq b$?

- Decision variant of “maximize $c^T x$ subject to $Ax \leq b$” but instead of any $x \in \mathbb{R}^n$ with $x \geq 0$, we are restricting $x$ to binary.

- Does restricting search space make the problem easier or harder?
  - This is actually NP-complete!
BILP Feasibility

• An even simpler problem
  ➢ Special case where $c = k = 0$, so $c^T x \geq k$ is always true

• Problem
  ➢ Input: $b \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}$
  ➢ Question: Does there exist $x \in \{0,1\}^n$ such that $Ax \leq b$?

  ➢ Just need to find a feasible solution
  ➢ This is still NP-complete!
BILP Feasibility

- Claim: BILP Feasibility is in NP

  - Recall: We need to show that there is a polynomial-time algorithm which
    - Can accept every YES instance with the right polynomial-size advice
    - Will not accept a NO instance with any advice

  - Advice: simply a vector $x$ satisfying $Ax \leq b$
  - Algorithm: Check if $Ax \leq b$
  - Simple!
BILP Feasibility

• Claim: Exact 3SAT \( \leq_p \) BILP Feasibility

- Take any formula \( \varphi \) of Exact 3SAT
- Create a binary variable \( x_i \) for each variable \( x_i \) in \( \varphi \)
  - We’ll represent its negation \( \overline{x}_i \) with \( 1 - x_i \)
- For each clause \( C \), we want at least one of its three literals to be TRUE
  - Just make sure their sum is at least 1
  - E.g., \( C = x_1 \lor \overline{x}_2 \lor \overline{x}_3 \Rightarrow x_1 + (1 - x_2) + (1 - x_3) \geq 1 \)
- Easy to check that
  - this is a polynomial reduction
  - Resulting system has a feasible solution if and only if \( \varphi \) is satisfiable
ILP Feasibility

• Problem
  ➢ Input: $b \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}$
  ➢ Question: Does there exist $x \in \mathbb{Z}^n$ such that $Ax \leq b$?

  ➢ To prove that this is NP-hard, there is an obvious reduction from BILP feasibility to ILP feasibility

  ➢ What about membership in NP?
  ➢ Advice: simply a vector $x$ satisfying $Ax \leq b$
  ➢ Algorithm: Check if $Ax \leq b$
  ➢ Simple?
    o No, not clear if, in every YES instance, there’s a polynomial-length “advice” vector $x$ satisfying $Ax \leq b$
On the Complexity of Integer Programming

CHRISTOS H. PAPADIMITRIOU

Massachusetts Institute of Technology, Cambridge, Massachusetts, and National Technical University, Athens, Greece

ABSTRACT. A simple proof that integer programming is in \( \mathcal{NP} \) is given. The proof also establishes that there is a pseudopolynomial-time algorithm for integer programming with any (fixed) number of constraints.

KEY WORDS AND PHRASES: integer linear programming, \( \mathcal{P}, \mathcal{NP}, \) pseudopolynomial algorithms

CR CATEGORIES: 5 25, 5.3, 5.4
So far...

• To establish NP-completeness of problem B, we always reduced Exact 3SAT to B
  ➢ But we can reduce any other problem A that we have already established to be NP-complete
  ➢ Sometimes this might lead to a simpler reduction because A might already be “similar” to B

• Let’s see an example!
**Vertex Cover**

**Problem**
- **Input:** Undirected graph $G = (V, E)$, integer $k$
- **Question:** Does there exist a vertex cover of size $k$?
  - That is, does there exist $S \subseteq V$ with $|S| = k$ such that every edge is incident to at least one vertex in $S$?

**Example:**
- Does this graph have a vertex cover of size 4?
  - Yes!
- Does this graph have a vertex cover of size 3?
  - No!
Vertex Cover

• **Problem**
  - **Input:** Undirected graph $G = (V, E)$, integer $k$
  - **Question:** Does there exist a vertex cover of size $k$?
    - That is, does there exist $S \subseteq V$ with $|S| = k$ such that every edge is incident to at least one vertex in $S$?

**Question:**
- Did we see this graph in the last lecture?
  - Yes!
  - For independent set of size 6

= vertex cover

= independent set
**Vertex Cover**

**Problem**
- **Input:** Undirected graph $G = (V, E)$, integer $k$
- **Question:** Does there exist a vertex cover of size $k$? That is, does there exist $S \subseteq V$ with $|S| = k$ such that every edge is incident to at least one vertex in $S$?

**Question:**
- Did we see this graph in the last lecture?
  - Yes!
  - For independent set of size 6

= vertex cover

= independent set
Vertex Cover

• Vertex cover and independent set are intimately connected!

• **Claim**: $G$ has a vertex cover of size $k$ if and only if $G$ has an independent set of size $n - k$

• **Stronger claim**: $S$ is a vertex cover if and only if $V \setminus S$ is an independent set
Vertex Cover

• **Claim:** $S$ is a vertex cover if and only if $V \setminus S$ is an independent set

• **Proof:**
  - $S$ is a vertex cover
  - IFF: For every $(u, v) \in E$, at least one of $\{u, v\}$ is in $S$
  - IFF: For every $(u, v) \in E$, at most one of $\{u, v\}$ is in $V \setminus S$
  - IFF: No two vertices of $V \setminus S$ are connected by an edge
  - IFF: $V \setminus S$ is an independent set  ■
Vertex Cover

• Claim: Independent Set $\leq_p$ Vertex Cover

- Take an arbitrary instance $(G, k)$ of Independent Set
- We want to check if there is an independent set of size $k$
- Just convert it to the instance $(G, n - k)$ of Vertex Cover
- Simple!
  - A reduction from Exact 3SAT would have basically repeated the reduction we already did for Exact 3SAT $\leq_p$ Independent Set

- Note: I didn’t argue that Vertex Cover is in NP
  - This is simple as usual. Just give the actual vertex cover as the advice.
Set Cover

• Problem
  ➢ Input: A universe of elements $U$, a family of subsets $S$, and an integer $k$
  ➢ Question: Do there exist $k$ sets from $S$ whose union is $U$?

• Example
  ➢ $U = \{1, 2, 3, 4, 5, 6, 7\}$
  ➢ $S = \{\{1, 3, 7\}, \{2, 4, 6\}, \{4, 5\}, \{1\}, \{1, 2, 6\}\}$
  ➢ $k = 3$? Yes! $\{\{1, 3, 7\}, \{4, 5\}, \{1, 2, 6\}\}$
  ➢ $k = 2$? No!
Set Cover

• Claim: Set Cover is in NP

   ➢ Easy. Let the advice be the actual \( k \) sets whose union is \( U \).

• Claim: Vertex Cover \( \leq_p \) Set Cover

   ➢ Given an instance of vertex cover with graph \( G = (V, E) \) and integer \( k \), create the following set cover instance
     o Set \( U = E \)
     o For each \( v \in V \), \( S \) contains a set \( S_v \) of all the edges incident on \( v \)
     o Selecting \( k \) set whose union is \( U = \) selecting \( k \) vertices such that union of their incident edges covers all edges
     o Hence, the two problems obviously have the same answer
Polynomial-Time Reductions

3-SAT

INDEPENDENT SET

VERTEX COVER

SET COVER

constraint satisfaction

3-SAT reduces to
INDEPENDENT SET

DIR-HAM-CYCLE

HAM-CYCLE

TSP

packing and covering

SEQUENCING

partitioning

numerical

GRAPH 3-COLOR

PLANAR 3-COLOR

SCHEDULING

SUBSET-SUM

Dick Karp (1972)
1985 Turing Award
Cook-Levin Theorem

• We did not prove “the first NP-completeness” result

• Theorem: Exact 3SAT is NP-complete
  ➢ We need to prove this without using any other “known NP-complete” problem
  ➢ We want to directly show that every problem in NP can be reduced to Exact 3SAT

• We will first reduce any NP problem to SAT, and then reduce SAT to Exact 3SAT
Cook-Levin Theorem

• We’re not going to prove it in this class, but the key idea is as follows

  ➢ If a problem is in NP, then ∃ Turing machine $T(x, y)$ which
    o takes as input a problem instance $x$ and an advice $y$ of size $p(|x|)$
    o verifies in $q(|x|)$ time whether $x$ is a YES instance
    o both $p$ and $q$ are polynomials

  ➢ $x$ is a YES instance iff $\exists y T(x, y) = ACCEPT$
Cook-Levin Theorem

• $x$ is a YES instance iff $\exists y T(x, y) = ACCEPT$
  - We need to convert $\exists y T(x, y) = ACCEPT$ into whether a SAT formula $\varphi$ is satisfiable

• Recall that a Turing machine $T$ consists of a memory tape, a head pointer, a state, and a transition function

• **What describes $T$ at any given step of its computation?**
  - What is written in each cell of its memory tape?
  - Which cell of the tape is the read/write head currently pointing to?
  - What state is the Turing machine in?
Cook-Levin Theorem

• $x$ is a YES instance iff $\exists y T(x, y) = ACCEPT$
  - We need to convert $\exists y T(x, y) = ACCEPT$ into $\exists z \varphi(z) = TRUE$, where $z$ consists of Boolean variables and $\varphi$ is a SAT formula

• Variables:
  - $T_{i,j,k} = True$ if machine’s tape cell $i$ contains symbol $j$ at step $k$ of the computation
  - $H_{i,k} = True$ if the machine’s read/write head is at tape cell $i$ at step $k$ of the computation
  - $Q_{q,k} = True$ if machine is in state $q$ at step $k$ of the computation
  - Cell index $i$ and computation step $k$ only need to be polynomially large as $T$ works in polynomial time
Cook-Levin Theorem

• $x$ is a YES instance iff $\exists y \ T(x, y) = ACCEPT$
  ➢ We need to convert $\exists y \ T(x, y) = ACCEPT$ into $\exists z \ \varphi(z) = TRUE$, where $z$ consists of Boolean variables and $\varphi$ is a SAT formula

• Clauses:
  ➢ Express how the variables must be related using the transition function
  ➢ Express that the Turing machine must reach the state $ACCEPT$ at some step of the computation

• This establishes that SAT is NP-complete.
• Next: SAT $\leq_p$ Exact 3SAT.
Cook-Levin Theorem

- **Claim**: SAT \(\leq_p\) Exact 3SAT
  - Take an instance \(\varphi = C_1 \land C_2 \land \cdots\) of SAT
  - Replace each clause with multiple clauses with exactly 3 literals each
    - For a clause with one literal, \(C = \ell_1\):
      - Add two variables \(z_1, z_2\), and replace \(C\) with four clauses
        \[ (\ell_1 \lor z_1 \lor z_2) \land (\ell_1 \lor \bar{z}_1 \lor z_2) \land (\ell_1 \lor z_1 \lor \bar{z}_2) \land (\ell_1 \lor \bar{z}_1 \lor \bar{z}_2) \]
      - Verify that this is logically equivalent to \(\ell_1\)
    - For a clause with two literals, \(C = (\ell_1 \lor \ell_2)\):
      - Add variable \(z_1\) and replace it with the following:
        \[ (\ell_1 \lor \ell_2 \lor z_1) \land (\ell_1 \lor \ell_2 \lor \bar{z}_1) \]
      - Verify that this is logically equal to \((\ell_1 \lor \ell_2)\)
Cook-Levin Theorem

• Claim: SAT $\leq_p$ Exact 3SAT

  ➢ For a clause with three literals, $C = \ell_1 \lor \ell_2 \lor \ell_3$:
    o Perfect. No need to do anything!

  ➢ For a clause with 4 or more literals, $C = (\ell_1 \lor \ell_2 \lor \cdots \lor \ell_k)$:
    o Add variables $z_1, z_2, \ldots, z_{k-3}$ and replace it with:
      $$(\ell_1 \lor \ell_2 \lor z_1) \land (\ell_3 \lor \bar{z}_1 \lor z_2) \land (\ell_4 \lor \bar{z}_2 \lor z_3) \land \cdots \land (\ell_{k-2} \lor \bar{z}_{k-4} \lor z_{k-3}) \land (\ell_{k-1} \lor \ell_k \lor \bar{z}_{k-3})$$
    o Check:
      • If any $\ell_i$ is TRUE, then there exists an assignment of $z$ variables to make this TRUE
      • If all $\ell_i$ are FALSE, then no assignment of $z$ variables will make this TRUE