Recall

Summary of Lecture 20

NP-completeness of Independent Set, Vertex Cover, Hamiltonian cycle, TSP

What you should know from Lecture 20:

- how to prove a problem is NP-complete using a polynomial time many-one reduction

Next:

\[
\text{Circuit SAT} \leq_P 3\text{-SAT} \leq_P \text{Ind. Set} \leq_P \text{Vertex Cover} \leq_P \text{Set Cover} \\
\leq_P \text{Ham. cycle} \leq_P \text{TSP} \leq_P \text{Subset Sum}
\]

These are harder proofs. Goal: appreciate trickier constructions; establish the results.
Subset Sum.
Input: Numbers $w_1, \ldots, w_n, W$
Question: Is there a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} w_i = W$

Theorem. Subset Sum is NP-complete.
Proof.

1. Subset Sum is in NP. (done in previous lecture)
2. 3-SAT $\leq_P$ Subset Sum

Assume we have a polynomial time algorithm for Subset Sum. Make a polynomial time algorithm for 3SAT.
Input: A 3-SAT formula $F$ with clauses $C_1 \ldots C_m$ on variables $x_1 \ldots x_n$
Output: Is $F$ satisfiable?
- construct an instance of Subset Sum such that it has a solution iff $F$ is satisfiable
- run the Subset Sum algorithm
- return its answer

We’ve seen how to turn 3-SAT into a packing problem (Independent Set) and into a sequencing problem (Hamiltonian cycle) and now we must turn it into a number problem.
Input: A 3-SAT formula $F$ with clauses $C_1 \ldots C_m$ on variables $x_1 \ldots x_n$
Construct an instance of Subset Sum such that it has a solution iff $F$ is satisfiable

Idea: Choosing numbers in Subset Sum will be choosing True/False. The bits of the numbers will encode information about the clauses.

Create a 0-1 matrix

<table>
<thead>
<tr>
<th></th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$\ldots$</th>
<th>$C_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>1</td>
<td>0</td>
<td>$\ldots$</td>
<td>0</td>
</tr>
<tr>
<td>$\neg x_1$</td>
<td>0</td>
<td>1</td>
<td>$\ldots$</td>
<td>0</td>
</tr>
<tr>
<td>$x_2$</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\neg x_2$</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_3$</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\neg x_3$</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\vdots$</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

E.g., $C_1 = (x_1 \lor \neg x_2 \lor x_3)$
$C_2 = (\neg x_1 \lor x_4 \lor x_5)$

General rule

$M[x_i, C_j] = 1$ if $x_i$ in $C_j$

$M[\neg x_i, C_j] = 1$ if $\neg x_i$ in $C_j$

= 0 otherwise

Regard the rows as binary (or other base) numbers.
Choosing a number = choosing a row. Adding numbers = adding up rows.
Input: A 3-SAT formula $F$ with clauses $C_1 \ldots C_m$ on variables $x_1 \ldots x_n$

Construct an instance of Subset Sum such that it has a solution iff $F$ is satisfiable

Idea: Choosing numbers in Subset Sum will be choosing True/False. The bits of the numbers will encode information about the clauses.

Create a 0-1 matrix

<table>
<thead>
<tr>
<th></th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$\cdots$</th>
<th>$C_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_1</td>
<td>1</td>
<td>0</td>
<td>$\cdots$</td>
<td>0</td>
</tr>
<tr>
<td>\neg x_1</td>
<td>0</td>
<td>1</td>
<td>$\cdots$</td>
<td>0</td>
</tr>
<tr>
<td>x_2</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\neg x_2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x_3</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\neg x_3</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\ddots</td>
<td>\vdots</td>
</tr>
</tbody>
</table>

$C_1 = (x_1 \lor \neg x_2 \lor x_3)$
$C_2 = (\neg x_1 \lor x_4 \lor x_5)$

General rule

$M[x_i, C_j] = 1$ if $x_i$ in $C_j$

$M[\neg x_i, C_j] = 1$ if $\neg x_i$ in $C_j$

We assume no clause contains the same literal twice.

Target sum $\geq 1$ $\cdots$ to ensure we pick $\geq 1$ literal in each clause

Regard the rows as binary (or other base) numbers.
Choosing a number = choosing a row. Adding numbers = adding up rows.
Input: A 3-SAT formula $F$ with clauses $C_1 \ldots C_m$ on variables $x_1 \ldots x_n$
Construct an instance of Subset Sum such that it has a solution iff $F$ is satisfiable

Idea: Choosing numbers in Subset Sum will be choosing True/False. The bits of the numbers will encode information about the clauses.

Create a 0-1 matrix

<table>
<thead>
<tr>
<th></th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$\ldots$</th>
<th>$C_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>1</td>
<td>0</td>
<td>$\ldots$</td>
<td>0</td>
</tr>
<tr>
<td>$\neg x_1$</td>
<td>0</td>
<td>1</td>
<td>$\ldots$</td>
<td>0</td>
</tr>
<tr>
<td>$x_2$</td>
<td>0</td>
<td>0</td>
<td>$\ldots$</td>
<td>0</td>
</tr>
<tr>
<td>$\neg x_2$</td>
<td>1</td>
<td>0</td>
<td>$\ldots$</td>
<td>0</td>
</tr>
<tr>
<td>$x_3$</td>
<td>1</td>
<td>0</td>
<td>$\ldots$</td>
<td>0</td>
</tr>
<tr>
<td>$\neg x_3$</td>
<td>0</td>
<td>0</td>
<td>$\ldots$</td>
<td>0</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\ldots$</td>
<td>$\vdots$</td>
</tr>
</tbody>
</table>

$e.g.$ $C_1 = (x_1 \lor \neg x_2 \lor x_3)$
$C_2 = (\neg x_1 \lor x_4 \lor x_5)$

General rule

$M[x_i, C_j] = 1$ if $x_i$ in $C_j$
$M[\neg x_i, C_j] = 1$ if $\neg x_i$ in $C_j$

We assume no clause contains the same literal twice.

Target sum $\geq 1 \geq 1 \ldots$ to ensure we pick $\geq 1$ literal in each clause.

Regard the rows as binary (or other base) numbers.
Choosing a number = choosing a row. Adding numbers = adding up rows.

Issues: (1) ensure we don’t choose row $x_i$ and row $\neg x_i$
(2) how can we ensure sum $\geq 1$? What can the sum be? 1 or 2 or 3.
Add slack rows of 1 and 2 so sum can always be 4.
Finally:

$$W = \text{interpret last row in base 10 numbers} = \text{one for each row, interpreting the row in base 10}$$

What is the size of the Subset Sum Problem?

$$2n + 2m \text{ numbers}$$

$$\text{each with } n+m \text{ base 10 digits.}$$

Why base 10?
Large enough to avoid carries. And familiar.

**Claim.** Polynomial time.
Claim. \( F \) is satisfiable iff there is a subset of the numbers with sum \( W \).

Proof.

⇒ Suppose \( F \) is satisfiable. If \( x_i \) is True, pick row \( x_i \). If \( x_i \) is False, pick row \( \neg x_i \).
Then column \( x_i \) adds up to its target 1, and column \( C_j \) adds to 1, 2, or 3.
Next we choose some slack rows \( S_{j,1} \) and \( S_{j,2} \) to increase the sum to 4:

\[
\begin{align*}
1 + S_{j,1} + S_{j,2} & = 4 \\
2 + S_{j,2} & = 4 \\
3 + S_{j,1} & = 4
\end{align*}
\]

This gives a set of rows (i.e. numbers) that sum to \( W \).

⇐ Suppose there is a subset with sum \( W \).
Note that any whole column sum is \( \leq 6 \), so no carries occur, and column sums must give the target digits.
Because \( x_i \) column sum is 1, we must have chosen row \( x_i \) or row \( \neg x_i \) (not both) — set the variable accordingly.
Because column \( C_j \) sum is 4 and slacks sum to \( \leq 3 \), we must have chosen a literal to satisfy clause \( C_j \). Thus \( F \) is satisfiable.
Summary of Lecture 21, Part 1

Subset Sum is NP-complete

What you should know from Lecture 21, Part 1:

- appreciate that NP-hardness proofs can be tricky, and that we can use numbers to encode things

Next:

\[
\begin{align*}
\text{Circuit SAT} & \leq_p 3\text{-SAT} \\
\text{ind. Set} & \leq_p \text{Vertex Cover} \leq_p \text{Set Cover} \\
\text{Ham. cycle} & \leq_p \text{TSP}
\end{align*}
\]

\[\leq_p \text{Subset Sum}\]
The first NP-completeness proofs.

**Circuit Satisfiability**

A *circuit* is a directed acyclic graph with:

- *sources* (no edge entering), labelled with variables or 0 or 1 — inputs
- one *sink* (no edge leaving) — output
- *internal nodes*

A circuit *computes an output* (in the obvious way) when values are given for the input variables.
Circuit Satisfiability

Input: A circuit $C$

Question: Is there an assignment of values to inputs such that the output is 1? i.e., is $C$ satisfiable?

Theorem. Circuit SAT is NP-complete.

Proof.

1. Circuit SAT is in NP. (easy, details omitted)

2. this is the first NP-completeness proof so we must prove that
   for every $Y$ in NP, $Y \leq_p$ Circuit SAT
   i.e. for every $Y$ in NP, there is an algorithm that maps any input $y$ for $Y$ to a circuit $C$ s.t. $y$ is a YES input iff $C$ is satisfiable.

High level idea only.

What can we use? Just that $Y \in$ NP,
   i.e., there is a poly time verification algorithm A for $Y$. A takes two inputs $y, g$, ($g =$ certificate or “guess”) and outputs YES/NO. Property of A:
   $y$ is a YES instance for $Y$ iff $\exists g$ (of poly size) s.t. $A(y,g)$ outputs YES
2. this is the first NP-completeness proof so we must prove that
   for every $Y$ in NP, $Y \leq_P \text{Circuit SAT}$

   What can we use? Just that $Y \in \text{NP}$,
i.e., there is a poly time verification algorithm $A$ for $Y$. $A$ takes two inputs $y, g$,
($g =$ certificate or “guess”) and outputs YES/NO. Property of $A$:
   
   $y$ is a YES instance for $Y$ iff $\exists g$ (of poly size) s.t. $A(y, g)$ outputs YES

**Idea:** Convert algorithm $A$ with known input $y$ and unknown input $g$
to a circuit $C$ with input variables $=$ bits of $g$
such that $C$ is satisfiable iff $\exists g$ s.t. $A(y, g)$ outputs YES

Write a program for algorithm $A$. Compile it. Assemble . . .
At the hardware level, $A$ is implemented by $\&$, $\lor$, $\neg$ gates.
We get a circuit $C$.

Inputs to $C$: bits of $y$ (known), bits of $g$ (variables)
Internal nodes of circuit: memory locations after each time step of algorithm $A$.

Because $\text{size}(g)$ is polynomial and $A$ runs in polynomial time, the circuit has polynomial size.

Is there an algorithm to convert $A$, $y$ to $C$? Yes: compiler, assembler, etc.
and this takes polynomial time.
Summary of Lecture 21, Part 2

Circuit SAT is NP-complete — the first NP-completeness proof (at least the idea)

Next:

\[
\begin{align*}
\text{Circuit SAT} & \leq_P 3\text{-SAT} & \leq_P \text{Ind. Set} & \leq_P \text{Vertex Cover} & \leq_P \text{Set Cover} \\
& & \leq_P \text{Ham.cycle} & \leq_P \text{TSP} \\
& & & \leq_P \text{Subset Sum}
\end{align*}
\]
**Theorem.** 3-SAT is NP-complete.

**Proof.**

1. 3-SAT is in NP. (easy, details omitted)

2. Circuit SAT $\leq_p$ 3-SAT

   Assume we have a polynomial time algorithm for 3-SAT. Make a polynomial time algorithm for Circuit SAT.

   Input: A circuit $C$
   Output: Is $C$ satisfiable?
   - construct a 3-SAT formula $F$ such that $C$ is satisfiable iff $F$ is satisfiable
   - run the 3-SAT algorithm
   - return its answer

   Intuitively (or from CS 245), circuits and formulas are equivalent. Just convert circuit $C$ to formula $F$. 
Convert circuit $C$ to formula $F$.

the obvious way:

\[(\neg x_1 \lor x_3) \land x_3 \lor \neg (x_2 \land x_3)\]

Caution: Is this polynomial size?

$\text{No!}$
Convert circuit \( C \) to formula \( F \).

the better way: make a variable \( x_u \) for each node \( u \) in the circuit

\[
\overbrace{x_u \equiv x_v \lor x_w}^{\text{as 3-SAT clauses}} \land \overbrace{(\neg x_u \lor x_v \lor x_w)}^{\text{as 3-SAT clauses}} \land (x_u \lor \neg x_v) \land (x_u \lor \neg x_w)
\]

Note: \( a \equiv b \) means \( (\neg a \lor b) \land (a \lor \neg b) \)

Claim. We can turn clauses of 2 literals into clauses of 3 literals.

Final formula: \( F = \bigwedge \) of all clauses \( \land x_{\text{output}} \)
Convert circuit $C$ to formula $F$.

the better way: make a variable $x_u$ for each node $u$ in the circuit

\[ x_u \equiv x_v \lor x_w \]

as clauses:
\[ \neg x_u \lor (x_v \land \neg x_u) \lor (x_u \lor \neg x_v) \lor (x_u \lor x_v) \]

\[ x_u \equiv x_v \land x_w \]

\[ \neg x_u \lor (x_v \lor x_u) \lor (x_u \lor \neg x_v) \lor (x_u \lor \neg x_w) \]

Note: $a \equiv b$ means $\neg a \lor b \land a \lor \neg b$

Claim. We can turn clauses of 2 literals into clauses of 3 literals.

Final formula: $F = \land$ of all clauses $\land x_{\text{output}}$
Claim 1. $F$ has polynomial size and can be computed in polynomial time.

Claim 2. $F$ is satisfiable iff $C$ is satisfiable.

Proof.
\(\Leftarrow\) Suppose $C$ is satisfiable. Then assigning True/False to variables of $F$ according to $C$'s computation will satisfy $F$.

\(\Rightarrow\) Suppose $F$ is satisfiable. Then there is an assignment of True/False to the variables (original inputs + new variables for circuit nodes) that makes $F$ True. For circuit $C$, use the same values for the input variables. By construction, the variables for the circuit nodes capture the evaluation of $C$. And $x_{\text{output}} = 1$ (True). Therefore $C$ is satisfiable.
What you should know from Lecture 21.

Appreciate NP-completeness proofs. Know some basic NP-complete problems.

Next:

A glimpse of more recent results on NP-completeness.