CS 341 W22 Lecture 21
NP-Hardness of Subset-Sum and Circuit SAT

R. Peng, University of Waterloo
Summary of Lecture 20

• NP-Completeness of
  • Independent Set
  • Vertex Cover
  • Hamiltonian Cycle
  • TSP

• What you should know: proving a problem is NP-complete using a polynomial time many-one reduction
Today

- These are the harder proofs
- Goals:
  - Formally establish NP-completeness
  - Example of more involved proofs
Subset Sum

• Input: n numbers $w_1...w_n$, goal $W$
• Output: is there a subset $S \subseteq \{1...n\}$ such that $\Sigma_{i \in S} w_i = W$?

**Theorem**: Subset Sum is NP-complete

• Subset sum is in NP: certificate is subset ... (rest exercise)
• 3-SAT $\leq_p$ Subset sum: assume access to a polynomial time algorithm for Subset sum, use it to give a polynomial time algorithm for 3-SAT

Alternatively: given a 3-SAT instance, output an instance of Subset Sum that’s true iff the 3-SAT instance is satisfiable

We have seen how to turn 3-SAT into graph problems, now we turn it into a number problem.
3-SAT $\leq_P$ Subset Sum

Recall construction of a 3-SAT formula $F$: $m$ clauses $C_1 \ldots C_m$, each the or of 3 literals on variables $x_1 \ldots x_n$.

Idea of conversion / reduction:

- Each digit of the sum encodes one clause
- Build a number corresponding to the effect of setting $x_i$ true

<table>
<thead>
<tr>
<th></th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>...</th>
<th>$C_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\neg x_1$</td>
<td>0</td>
<td>1</td>
<td></td>
<td></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>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Assume no carries,

general rule: want $\geq 1$ in each clause/digit

So if $C_j$ contains $x_i$, $M[x_i, C_j] = 1$,

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

e.g. $C_1 = (x_1 \lor \neg x_2 \ldots)$
Issues

• How to enforce $\geq 1$, instead of $= 1$:
  allow extra term of 1 & 2, require sum = 4

• How to ensure that we don’t pick both $x_i$ and $\neg x_i$
  create an extra digit where both $x_i$ and $\neg x_i$ are 1s on, and require a sum of 1 on that digit
**Larger Matrix**

- 4 numbers per variable
- \( m + n \) digits
- Goal: \( m \) digits 4, \( n \) digit of 1
- Work in base 10 so there is no possibilities of carries

Slack rows to allow either 1, 2, 3 from sums of rows correspond to \( x_i \) / \( \neg x_i \)

To ensure we don’t pick both \( x_i \) and \( \neg x_i \)
Correctness

• Poly-time: resulting instance has $O(n + m)$ numbers, each with $O(n + m)$ digits.

• If $F$ is satisfiable, then the Subset Sum instance has solution: if $x_i$ true, pick row $x_i$, else pick row $\neg x_i$. The last $n$ digits are good. For the first $m$ digits, if the sum is 1, add both $ja$ and $jb$, if 2, add in $jb$, if 3 add in $ja$.

• If there is a subset with sum $W$, then $F$ has solution:
  • first, note that because each digit sum is at most 6, and we work in base 10 there are no carries.
  • Last $m$ digits give that we pick exactly one of $x_i$ or $\neg x_i$
  • Claim: that corresponding assignment to $x_i$ satisfies $F$: for each clause $j$, the digit has sum 4, but the extra sum to at most 3, so one of the literals in the clause must be satisfied
Summary of Lecture 21, Part 1

We showed:
• Subset Sum is NP-complete

What you should know from Lecture 21 Part 1:
• NP-completeness proofs can be tricky
• Numbers can encode a lot of things
The First NP-Completeness Proofs

Circuit Satisfiability
A circuit is a directed acyclic graph with:
• Edges passing values on as inputs
• Sources (no edges entering) / inputs labeled with True/False
• One sink (no edges leaving), output
• Internal nodes labeled \( \land \) “AND”, \( \lor \) “OR”, \( \neg \) “NOT”

Given inputs, a circuit computes an output by having all nodes evaluate their values in topological order.

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

True iff \( x_1 = x_2 \)
Circuit Satisfiability

**Input**: a circuit C

**Output**: is the circuit satisfiable, aka. is there an assignment of values to the inputs such that the output is True

Theorem: Circuit SAT is NP-complete

1. Circuit SAT is in NP: certificate is the input, simulate circuit
2. This is the first NP-completeness proof, so we must prove for every Y in NP, $Y \leq_p$ Circuit SAT.

High level ideas only: only fact we can use is Y is in NP, i.e. that it has a poly-time verifier algorithm A
Y \leq P \text{ Circuit SAT for every } Y \in NP

Y \in NP: there is an algorithm A that takes two inputs y & g, and outputs YES/NO such that y is a YES instance of Y if and only if there exists a g of poly size such that A(y, g) outputs YES.

**Idea: convert A to a circuit with input variables being g**

How? A lot of handwaving for now:

- Code A, compile + assemble, look at things at hardware level
- Internal nodes of circuit: states of memory at each time step of running A
- \(|g|, \text{Time}(A)\) both \(\text{poly}(|y|)\), so C has polynomial size

Obtain a circuit C such that \(C(g) = A(y, g)\). Reduction algorithm is basically a compiler.
Summary of Lecture 21, Part 2

We showed:

• Circuit SAT is NP-complete --- the first NP-completeness proof, or at least the idea

What you should know from Lecture 21 Part 2:

• Checkers can be encoded as circuits, after which we ‘solve’ for an input.
Theorem: 3-SAT is NP-Complete

1. 3-SAT is in NP: previously covered, exercise

2. Circuit SAT $\leq_p$ 3-SAT:
   • Assume we have a polynomial time algorithm for 3-SAT, use it to solve circuit SAT.
   • Given a circuit, construct a 3-SAT instance that’s satisfiable iff the circuit is satisfiable.
   • Intuitively (or from CS245): circuits and formulas are equivalent. Convert circuit to formula.
The Obvious Way

Chain together formulas of input nodes

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

This is not polynomial time / sized
One Variable Per Node

Local conditions become clauses with at most 3 literals

- \( x_u \) is true only if both \( x_v \) & \( x_w \) are true:
  - \( x_v \) false \( \rightarrow \) \( x_u \) false: cannot have \( x_v = F \) and \( x_u = T \): clause \( x_v \lor \neg x_u \)
  - Similarly, get \( x_w \lor \neg x_u \) so that \( xw = F \) forces \( x_u \) to be \( T \)
  - Finally, can’t have \( x_u = F \) when both \( x_v \) and \( x_w \) are \( T \), so a clause with three literals: \( \neg x_v \lor \neg x_w \lor x_u \)
One Variable Per Node

Local formulas become clauses with at most 3 literals

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

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

\[(-x_v \lor x_u) \land (x_v \lor \neg x_u)\]
Circuit SAT \leq_p 3-SAT

Final Formula: F = \wedge \text{all clauses} \wedge x_{\text{output}}

- F is poly-sized and can be computed in poly-time.
- F is satisfiable iff C is satisfiable

\(\Leftarrow\) If C is satisfiable, then assigning T/F to the variables in F according to their values in C’s computation satisfies F.

\(\Rightarrow\) If F is satisfiable, then there is assignment of True/False to the variables that makes F true. Use the same values on input variables.

By construction, the values of the intermediate nodes are what we get from evaluating C on those inputs, and we also have the output is True from the extra clause in F.
Summary of Lecture 21

We showed:

• 3-SAT $\leq_p$ Subset Sum
• Every problem in NP $\leq_p$ Circuit SAT
• Circuit SAT $\leq_p$ 3-SAT

What you should know from Lecture 21:

• NP-completeness proofs can get very interesting
• Everything in NP reduce to circuits, and SAT instances