NP-Complete Problems So Far

- 3-SAT
- Indep-Set
- Vertex-Cover
- Set-Cover
- Circuit-SAT
- Ham-Cycle
- Traveling-Salesman

Arrows show reductions discussed in class.
We could construct a polynomial reduction between any pair.

NP-Completeness and Reductions

Careful, direction of reduction matters!

A \leq_P B: A reduces to B (A “no harder” than B)

From arbitrary instance of A, construct instance of B
Reduction and construction is one-way

Problem instances are equivalent (both ways):

Yes_A \implies Yes_B
Yes_B \implies Yes_A (same as No_A \implies No_B)

B is NP-complete means:
1. B is in NP: can check solution in polynomial time
   (“easy enough”)
2. B is NP-hard: some NP-complete A reduces to B: A \leq_P B
   (“hard enough”). We also say: reduce from A.

Clicker

Which of the following graph problems are in NP?

A. Length of longest simple path is \leq k
B. Length of longest simple path is = k
C. Length of longest simple path is \geq k
D. Find length of longest simple path.
E. All of the above.

Numerical problems

Subset Sum decision problem: given n items with weights
w_1, ..., w_n, is there a subset of items whose weight is exactly W?

Dynamic programming: O(nW) pseudo-polynomial time algorithm
(not polynomial in input length n \log W)

Subset Sum Warmup

Does this instance have a solution?

- w_1 1010
- w_2 1001
- w_3 0110
- w_4 0101
- W 1111

A. Yes
B. No
Subset Sum Warmup

For which nonzero values of $y$ does this instance have a solution?

10010
10001
01001
01010
00111
00100
-----
1113y

A. $y = 1$
B. $y = 1, 2$
C. $y = 1, 2, 3$

Subset Sum

Theorem. Subset sum is NP-complete.

Reduction from 3-SAT. ($n$ variables, $m$ clauses).

Subset Sum Reduction

$$ (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor x_3) $$

<table>
<thead>
<tr>
<th>Item</th>
<th>variable digits</th>
<th>clause digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>$f_1$</td>
<td>1 0 0</td>
<td>0 1 1</td>
</tr>
<tr>
<td>$t_2$</td>
<td>0 1 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>$f_2$</td>
<td>0 1 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0 0 1</td>
<td>0 1 0</td>
</tr>
<tr>
<td>$f_3$</td>
<td>0 0 1</td>
<td>0 1 0</td>
</tr>
<tr>
<td>$W$</td>
<td>1 1 1</td>
<td>3 3 3</td>
</tr>
</tbody>
</table>

- Items $t_i, f_i$ for each $x_i$: correspond to truth assignment
- Weights $\Rightarrow$ select exactly one
- (Numbers are base 10)

- Clause digit equal to 1 iff $x_i$ assignment satisfies $C_j$
- Total for clause digit > 0 iff assignment satisfies $C_j$
- Goal: all clause digits > 0. How to set $W$ to enforce this?
  - Total could be 1, 2, 3 for satisfied clause.

Set all clause digits of $W$ to 3... then add dummy items to increase total by at most two.
Chapter 8 NP and Computational Intractability

### Subset Sum Reduction

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

#### Item | variable digits | clause digits
--- | --- | ---
\(t_1\) | 1 0 0 | 1 0 0
\(f_1\) | 1 0 0 | 0 1 1
\(t_2\) | 0 1 0 | 0 1 0
\(f_2\) | 0 1 0 | 1 0 1
\(f_3\) | 0 0 1 | 0 1 0
\(W\) | 1 1 1 | 3 3 3

- Two dummy items per clause \(\Rightarrow\) can increase total by up to 2
- Can make total exactly 3 iff total of non-dummy items is \(> 0\)

### Subset Sum Reduction: Details (Review on Own)

- All weights have \(n + m\) digits
- For variable \(x_i\), create two items \(t_i, f_i\)
  - Both have \(i\)th digit equal to 1
  - All other items have zero in this digit
  - \(i\)th digit of \(W = 1\) \(\Rightarrow\) select exactly one of \(t_i, f_i\)
- The \(n + j\)th digit corresponds to clause \(C_j\)
  - If \(x_i \in C_j\), set \(n + j\)th digit of \(t_i = 1\)
  - If \(\neg x_i \in C_j\), set \(n + j\)th digit of \(f_i = 1\)
  - Everything else 0

### Subset Sum Proof

- All numbers (including \(W\)) are polynomially long.
- If \(\Phi\) satisfiable,
  - Select \(t_i\) if \(x_i = 1\) in satisfying assignment else select \(f_i\)
  - Take \(y_j, z_j\) as needed.
- If subset exists with sum \(W\)
  - Either \(t_i\) or \(f_i\) is chosen. Assign \(x_i\) accordingly.
  - For each clause, at least one term must be selected, otherwise clause digit is \(< 3\).

### Graph Coloring

**Def.** A \(k\)-coloring of a graph \(G = (V, E)\) is a function \(f : V \rightarrow \{1, \ldots, k\}\) such that for all \((u, v) \in E\), \(f(u) \neq f(v)\).

**Problem.** Given \(G = (V, E)\) and number \(k\), does \(G\) have a \(k\)-coloring?

**Many applications**

- Actually coloring maps!
- Scheduling jobs on machine with competing resources.
- Allocating variables to registers in a compiler.

**Claim.** 2-COLORING \(\in P\) (equivalent to bipartite testing)

**Theorem.** 3-COLORING is NP-Complete.

### 3-Color: Gadget for Variables

- Reduce from 3-SAT.

3 colors: True, False, “Base”

3 special nodes in a clique \(T, F, B\).

For each variable \(x_i\), two nodes \(v_{i0}, v_{i1}\).

Edges \((v_{i0}, B), (v_{i1}, B), (v_{i0}, v_{i1})\).

Either \(v_{i0}\) or \(v_{i1}\) colored \(T\), the other colored \(F\).
Reduction: Clause Gadget

For clause $x_i \lor \neg x_j \lor x_k$

Top node can be colored iff not all three $v$-nodes are $F$.

Proof

- Graph is polynomial in $n + m$.
- If satisfying assignment
  - Color $B, T, F$ then $v_{1i}$ as $T$ if $\phi(x_i) = 1$.
  - Since clauses satisfied, can color each gadget.
- If graph 3-colorable
  - One of $v_{10}, v_{11}$ must get $T$ color.
  - Clause gadget colorable iff clause satisfied.

Question. What about $k$-coloring?

Clicker Question

Which of the following is true?

A. If we can reduce 3-coloring to $k$-coloring, then $k$-coloring is NP-complete
B. $k$-coloring is NP-complete since any 3-coloring is also a $k$-coloring for $k \geq 3$
C. $k$-coloring is not NP-complete since 3-coloring is the hardest case, for $k > 3$ the coloring is easier
D. $k$-coloring is not NP-complete because the 4-color theorem has been proved

NP-Completeness Recap

Types of hard problems:

- 3-SAT
- Independent-Set
- Vertex-Cover
- Set-Cover
- Circuit-SAT
- Hamilton-Cycle
- Hamilton-Path
- Traveling-Salesman
- Subset-Sum
- 0-1 Knapsack
- Graph-Coloring
- Constraint satisfaction
- Partitioning
- Numerical
- Sequencing
- Packing
- Covering

... and many others. See book or other sources for more examples. You can use any known NP-complete problem to prove a new problem is NP-complete.