http://www.cs.unh.edu/~ruml/cs758
NP-Completeness

- Terms
- Interchangability
- Reductions
- NPC Proofs
- C-SAT is in NP
- C-SAT is NP-Hard
- Break

NP-Completeness
optimization vs decision: if opt were easy, decision would be too

P: solvable in polynomial time

NP: ∃ certificate verifiable in polynomial time

NP-Hard: as hard as any problem in NP (via polytime reduction)

NP-Complete: NP-Hard and in NP

reduce $a$ to $b$: $a \rightarrow b$ in polytime, decide $b$

$b$ hard by reduction from $a$: if $a \rightarrow b$ in polytime and $b$ polytime, could solve $a$
The Power of Reduction

Theorem: If \( B \leq_P A \) for some \( B \in \text{NPC} \), then \( A \) is NP-Hard.

Since \( B \) is NPC, we have \( \forall C \in \text{NP}, C \leq_P B \). Since \( B \leq_P A \), then \( C \leq_P A \) which shows \( A \) is NP-Hard.

If also \( A \in \text{NP} \), then since \( A \in \text{NP} \), we have \( A \in \text{NPC} \).
Reductions

NP-Completeness
- Terms
- Interchangability
- Reductions
  - NPC Proofs
  - C-SAT is in NP
  - C-SAT is NP-Hard
  - Break

SAT

CIRCUIT-SAT
  ↓
  SAT
  ↓
  3-CNF SAT
  ↓
  CLIQUE  SUBSET-SUM
  ↓
  VERTEX-COVER
  ↓
  HAM-CYCLE
  ↓
  TSP
To prove some problem $A$ is NP-Complete:

1. Prove $A \in NP$
2. Pick a known NP-Complete problem $B$
3. Find a translation of instances of $B$ into instances of $A$
4. Show that translated $A$ version is accepted if and only if the original $B$ version should be accepted.
5. Prove that the reduction runs in polynomial time.
Circuit-SAT is in NP

Circuit-SAT: is circuit satisfiable? (otherwise, can be removed)

Certificate is value for every wire.
Simply check that each gate is computed correctly and output is true.
Circuit-SAT is NP-Hard

Need to construct reduction \( f \) from any \( L \in \text{NP} \). Given input \( x \in L \), resulting circuit \( C \in \text{Circuit-SAT} \) iff \( x \in L \). We’ll make \( C \) so it’s SAT iff \( \exists y \) s.t. verification algorithm \( A(x, y) \) for \( L \) gives true. Intuition: for input \( y \), run \( A(x, y) \).

Let \( n = |x| \) and \( T(n) = O(n^k) \) be bound on \( A \)’s running time.

Let \( M \) be a circuit for a stored-program computer (including PC and storage). String \( T(n) \) of them together to form \( C'' \).

\( C \) is \( C'' \) with input hardwired to program for \( A \) and input \( x \), and output hardwired to result of \( A \). Input to \( C \) is \( y \).

Iff \( y \) exists, \( C \) is satisfiable, so we have a reduction.

\( A \) is constant size and uses poly storage. \( M \) is poly size and needs poly steps to run \( A \). \( y \) is poly sized. So \( C'' \) and \( C \) have size polynomial in \( n \) and can be constructed in polynomial time.
NP-Completeness
■ Terms
■ Interchangability
■ Reductions
■ NPC Proofs
■ C-SAT is in NP
■ C-SAT is NP-Hard
■ Break

SAT

- asst 12
- wildcard
To prove some problem $A$ is NP-Complete:

1. Prove $A \in NP$
2. Pick a known NP-Complete problem $B$
3. Find a translation of instances of $B$ into instances of $A$
4. Show that translated $A$ version is accepted if and only if the original $B$ version should be accepted.
5. Prove that the reduction runs in polynomial time.
Consider formula with \( n \) variables and \( m \) connectives.

\[ \text{SAT} \in \text{NP}: \text{given variables assignments, evaluate formula.} \]

\[ \text{SAT is NP-Hard: Reduction from Circuit-SAT. Basic translation fails on shared subcircuits.} \]

\[ \text{Instead, use one variable for each wire and one clause per gate.} \]
\[ \text{Combine clauses with} \ \wedge \ \text{and include} \ \wedge \ x_0 \ (\text{output}). \]
\[ \text{SAT iff wires in circuit have legal values yielding true.} \]
3-CNF SAT

CNF where each clause has exactly 3 literals. Aka 3-SAT.

3-CNF SAT ∈ NP: given variables assignments, evaluate formula.

3-CNF SAT is NP-Hard: Reduction from SAT. Construct expression tree and convert to binary branching.
Assign each node a variable.
Form clause for each internal node’s variable, eg: \( y_3 \leftrightarrow (y_1 \lor y_2) \)
Clauses will have at most 3 literals.
Convert each clause to CNF: form complete truth table, form DNF for false rows, negate and push \( \neg \) inward (using DeMorgan) to get CNF
For each binary clause \((l_1 \lor l_2)\), convert to \((l_1 \lor l_2 \lor p) \land (l_1 \lor l_2 \lor \neg p)\).
For each unit clause \((l)\), convert to \((l_1 \lor p \lor q) \land (l \lor p \lor \neg q) \land (l \lor \neg p \lor q) \land (l \lor \neg p \lor \neg q)\).
Each step preserves satisfiability and is polynomial time.
For example:

- What’s still confusing?
- What question didn’t you get to ask today?
- What would you like to hear more about?

Please write down your most pressing question about algorithms and put it in the box on your way out.

*Thanks!*