CS F364
Design & Analysis of Algorithms

ALGORITHMS - COMPLEXITY

Complexity Classes

- $\mathcal{NP}$-Completeness Via Reductions
  - Examples: SAT

$\mathbb{NP}$-Completeness Via Reductions

- Claim:
  - If π_1 is $\mathbb{NP}$-hard and $\pi_1 \precsim \pi_2$
  - then π_2 is $\mathbb{NP}$-hard.
- Proof:
  - Since π_1 is $\mathbb{NP}$-hard
    - for all π in $\mathbb{NP}$ $\pi \precsim \pi_1$
  - By transitivity
    - for all π in $\mathbb{NP}$ $\pi \precsim \pi_2$
  - i.e. π_2 is $\mathbb{NP}$-hard

ℕℙ-Completeness Via Reductions

- Claim: SAT is ℕℙ-Complete.
- Proof:
  - SAT is in NP
    - Given a formula F of length n in Boolean logic, a certificate for satisfiability would be:
      - a Boolean assignment to its variables for which F will evaluate to true.
    - The length of this certificate is the number of variables (i.e. <= n).
    - The time taken for verifying this certificate is
  - CIRCUIT-SAT $\precsim$ SAT

NP-Completeness Via Reductions

- Claim: CIRCUIT-SAT $\precsim$ SAT
- Proof Attempt:
  - The classic technique of constructing an equivalent formula given a circuit does not work:
    - The time for constructing the table is 2^n (why?), given n inputs to the circuit.
      - The size of the circuit need not be exponential in n i.e. the size of the table may be exponential in the size of the circuit.
      - **Exercise**:
        - Construct an example circuit for which this is true.
    - Can one walk the graph and extract the formula?
      - *When **fan-out** is unlimited, the number of paths walked may be exponential.*

CIRCUIT-SAT ≾ SAT

- We need to map each circuit C to a formula F such that
  - *C is satisfiable* iff *F is satisfiable*
- Given a circuit C:
  - assume each input line is marked with a variable a_j
  - mark each output line of a gate with a variable $b_{i,}$ i>0
  - mark the final output line as b_0
- Construct a formula F:
  - **(AND_i g_i) AND b_0** where each g_i is a formula for gate i:
    - **b_{i1} op b_{i2} <--> b_{i3}** if the gate is binary (i.e. AND or OR)
    - **b_{i1} <--> b_{i3}** if the gate is unary (i.e. NOT) where
      - **b_{i1}** and **b_{i2}** are variables corresponding to input lines
      - **b_{i3}** corresponds to the output line of the gate, and
      - **op** is the operator of the gate.

NP-Completeness Via Reductions – SAT [2]

- Proof [contd.]: CIRCUIT-SAT $\precsim$ SAT
  - (*see previous slide*)
    - we have a mapping of every combinational Boolean circuit C to some Boolean formula F.
  - Claims:
    - F is satisfiable iff C is satisfiable [Why?]
      - i.e. our *mapping is a reduction*.
    - Length of F is linearly proportional to that of C.
      - i.e. our *mapping is a polynomial time reduction*.

CS F364
Design & Analysis of Algorithms

ALGORITHMS - COMPLEXITY

Complexity Classes

- NP-Completeness Via Reductions
  - Examples: CNF-SAT

PROBLEM: CNF-SAT

- CNF-SAT:
  - Given a Boolean expression F in CNF (i.e. Conjunctive Normal Form)
    - *find whether there is an input assignment such that the F is satisfied.*
- Question:
  - Can you design an algorithm to solve CNF-SAT ?
    - Can you design a polynomial time algorithm to solve CNF-SAT?
      - Points to ponder:
        - *Verifying validity of a formula in CNF can be done in polynomial time*.
        - **(Dual):** *Verifying satisfiability of a formula in DNF can be done in polynomial time.*
  - But we argued that SAT is NP-complete:
    - Is there an implication on converting a SAT instance to equivalent CNF (or DNF) ?

$\mathbb{NP}$-Completeness Via Reductions: CNF-SAT

- CNF-SAT:
  - Given a Boolean expression F in CNF
    - *find whether there is an input assignment such that the F is satisfied.*
- CNF-SAT is $\mathbb{NP}$-complete
  1. CNF-SAT is in $\mathbb{NP}$.

     Proof:

     Given an assignment of values (as a certificate) verification can be done in linear time.
  2. CNF-SAT is $\mathbb{NP}$-hard

     i.e. SAT $\precsim$ CNF-SAT

     (*see following slides for a reduction*).

NP-Completeness Via Reductions: CNF-SAT [2]

- SAT $\precsim$ CNF-SAT
  - Is there an algorithm to convert a Boolean expression F to its equivalent CNF?
    - What is the time complexity of the algorithm?
  - We want a mapping

    g: I(SAT) --> I(CNF-SAT)

  that ONLY needs to preserve satisfiability!

SAT $\precsim$ CNF-SAT

- Mapping: **g(F) = F'** where **F'** is in CNF.
  - Construct the parse tree of the given expression F.
  - Label each edge with a variable (*including the one incoming edge to the root*)
    - Label each leaf with the input variable
  - Construct formula **F'** as $\mathbf{r_0}$ **AND (AND$_i$ v$_i$)**
    - where each $\mathbf{v_i}$ corresponds to a vertex of the tree and is of the form:
      - $\mathbf{e_o}$ **<--> $\mathbf{e_{i1}}$ op $\mathbf{e_{i2}}$** if **op** is *binary*
      - $\mathbf{e_o}$ **<--> op $\mathbf{e_i}$** if **op** is *unary*
        - for output edge $\mathbf{e_o}$ and input edges $\mathbf{e_{i1}}$ and $\mathbf{e_{i2}}$

SAT ≾ CNF-SAT [2]

- Claims (A and B):
  A. The mapping **g** can be computed in polynomial time:
     1. Time taken for constructing a parse tree, given a formula of length **n**
        1. Time taken for parsing: **O(n³)**

           refer to **CYK algorithm (Dynamic Programming)**
        2. Time taken for construction of tree: **O(n)**
     2. Time taken for constructing the formula from the tree:
        1. Time taken for traversing the tree: **O(n)**
        2. Time taken for writing the formula: **O(k*n)** where k is the (constant) length of each clause

  B. **g(F) is satisfiable iff F is satisfiable**.

CS F364
Design & Analysis of Algorithms

Algorithms - Complexity

Complexity Classes

- NP-Completeness Via Reductions
  - Examples: kSAT

PROBLEMS: KSAT

- kSAT
  - Satisifiability problem, where input instances are in CNF with exactly k distinct literals in clause.
- 2SAT is kSAT for k=2
- 2SAT can be solved in polynomial time.

2SAT IS IN $\mathbb{P}$

- Exercise: Reduce 2SAT to a problem on directed graphs that is efficiently solvable:
  1. Note that L1 | L2 is equivalent to !L1 --> L2.
  2. Given a formula F in 2CNF, construct a directed graph G:
     1. add vertices labeled x and !x for each variable x occurring in the formula.
     2. add an edge from L1 to L2 if L1 --> L2 is a clause in F
  3. Argue that:

     there is a path in G from x to !x for some variable x

     iff

     F is not satisfiable.

PROBLEMS: 2SAT VS. HORN-SAT

- 2SAT can be solved in polynomial time.
- Exercise:
  1. Can you reduce 2SAT to HORN-SAT?
- HORN-SAT is *satisfiability of Horn formulas*.
  - A Horn formula is a conjunction of Horn clauses and
  - a Horn clause is of the form x --> y (i.e. !x | y)

[Note that HORN-SAT is solvable in polynomial time.
Refer to *Huth & Ryan: Logic in CS* for an algorithm for HORN-SAT.]

PROBLEM: 3SAT

- 3CNF-SAT (*which is commonly referred to as* 3SAT):
  - Given a Boolean expression in CNF with *exactly 3 distinct literals* in each clause,
  - find whether there is an input assignment such that the expression is satisfied.
- Solving 3SAT:
  - There is no known polynomial time algorithm to solve 3SAT.

3SAT IS $\mathbb{NP}$-COMPLETE

- 3SAT is $\mathbb{NP}$-complete
  - 3SAT is in $\mathbb{NP}$
    - Proof: Trivial
  - 3SAT is $\mathbb{NP}$-hard
    - Proof: CNF-SAT $\precsim$ 3SAT
    - Reduction:
      - Map each clause in the CNF expression to one or more 3-literal clauses.
      - (*see next slide for the mapping*)

CNF-SAT $\precsim$ 3SAT

Mapping M: Let the input formula F be the conjunction $AND_i\ C_i$

Then for each i:

1. case **C_i is L** : Replace C_i with
   $(L \mid p \mid q)\ \&\ (L \mid !p \mid q)\ \&\ (L \mid p \mid !q)\ \&\ (L \mid !\ \mid !q)$
2. case **C_i is $L_1 \mid L_2$** : Replace C_i with
   $(L_1 \mid L_2 \mid p)\ \&\ (L_1 \mid L_2 \mid !p)$
3. case **C_i is $L_1 | L_2 | L_3$** : Replace C_i with C_i
4. case **C_i is $L_1 | L_2 | L_3 | L_4$:** Replace C_i with
   $(p \mid L_3 \mid L_4)\ \&\ (!p \mid L_1 \mid L_2)\ \&\ (p \mid !L_1)\ \&\ (p \mid !L_2)$
   and recursively handle the last two clauses.
1. case **C_i is $L_1 \mid \ldots \mid L_k$** (k>4): Replace C_i with
   $(p \mid L_3 \mid L_4 \mid \ldots \mid L_k)\ \&\ (!p \mid L_1 \mid L_2)\ \&\ (p \mid !L_1)\ \&\ (p \mid !L_2)$
   and recursively handle the first clause and the last two clauses.

p and q are new variables introduced.

- Note:
  - The last three clauses in the replaced expression for the fourth case correspond to $(p \text{ <--> } (L_1 \mid L_2))$.

CNF-SAT ≾ 3SAT [2]

- Claim (about the mapping M described in the previous slide):
  - M is a polynomial time algorithm that
    - given an input F outputs F' in 3CNF such that
      - F' is satisfiable iff F is satisfiable and
      - the size of F' is a polynomial function of the size of F.
- Exercises:
  - Implement the algorithm M.
  - Derive the time complexity of M.
  - Estimate the size of the output of M as a function of the size of its input.

kSAT IS $\mathbb{NP}$-COMPLETE

- Exercise:
  - Prove that kSAT is $\mathbb{NP}$-complete