NP-Completeness and Boolean Satisfiability

Mridul Aanjaneya

Stanford University

August 14, 2012

Time-Bounded Turing Machines

- A Turing Machine that, given an input of length n, always halts within T(n) moves is said to be T(n)-time bounded.
  - The TM can be multitape.
  - Sometimes, it can be nondeterministic.
- The deterministic, multitape case corresponds roughly to an $O(\mathrm{T(n)})$ running-time algorithm.

The class P

- If a DTM M is T(n)-time bounded for some polynomial T(n), then we say M is polynomial-time (polytime) bounded.
- And L(M) is said to be in the class P.
- **Important Point:** When we talk of P, it doesn't matter whether we mean by a computer or by a TM.

Polynomial Equivalence of Computers and TM's

- A multitape TM can simulate a computer that runs for time $O(\mathsf{T(n)})$ in at most $O(\mathsf{T}^2(\mathsf{n}))$ of its own steps.
- If $\mathsf{T(n)}$ is a polynomial, so is $\mathsf{T}^2(\mathsf{n})$.

Examples of Problems in P

- Is w in $L(G)$, for a given CFG G?
  - Input w.
  - Use CYK algorithm, which is $O(n^3)$.
- Is there a path from node x to node y in graph G?
  - Input $= x, y$, and G.
  - Use Dijkstra's algorithm, which is $O(n \log n)$ on a graph of n nodes and arcs.

Running Times Between Polynomials

- You might worry that something like $O(n \log n)$ is not a polynomial.
- However, to be in P, a problem only needs an algorithm that runs in time less than some polynomial.
- Surely $O(n \log n)$ is less than the polynomial $O(n^2)$.

A Tricky Case: Knapsack

- The Knapsack problem is: given positive integers i_1, i_2,..., i_n, can we divide them in two sets with equal sums?
- Perhaps we can solve this problem in polytime by a dynamic-programming algorithm:
  - Maintain a table of all the differences we can achieve by partitioning the first j integers.

Knapsack

- **Basis:** $j=0$. Initially, the table has true for 0 and false for all other differences.
- **Induction:** To consider i_j, start with a new table initially all false.
- Then set k to true if, in the old table, there is a value m that was true, and k is either $m+i_j$ or $m-i_j$.

Knapsack

- Suppose we measure running time in terms of the sum of the integers, say m.
- Each table only needs space $O(m)$ to represent all the positive and negative differences we could achieve.
- Each table can be constructed in time $O(n)$.

Knapsack

- Since $n \leq m$, we can build the final table in $O(m^2)$ time.
- From that table, we can see if 0 is achievable and solve the problem.

Subtlety: Measuring Input Size

- Input size has a specific meaning: the length of the representation of the problem as it is input to a TM.
- For the Knapsack problem, you cannot always write the input in a number of characters that is polynomial in either the number of or sum of the integers.

Knapsack - Bad Case

- Suppose we have n integers, each of which is around 2^n.
- We can write integers in binary, so the input takes $O(\mathrm{n}^2)$ space to write down.
- But the tables require space $O(\mathrm{n}2^n)$.
- They therefore require at least that order of time to construct.

Bad Case

- Thus, the proposed polynomial algorithm actually takes time $O(\mathsf{n}^2 2^n)$ on an input of length $O(\mathsf{n}^2)$.
- Or, since we like to use n as the input size, it takes time $O(\mathsf{n}2^{\mathsf{sqrt}}(\mathsf{n}))$ on an input of length n.
- In fact, it appears no algorithm solves Knapsack in polynomial time.

Redefining Knapsack

- We are free to describe another problem, call it Pseudo-Knapsack, where integers are represented unary.
- Pseudo-Knapsack is in P.

The class NP

- The running time of a nondeterministic TM is the maximum number of steps taken along any branch.
- If that time bound is polynomial, the NTM is said to be polynomial-time bounded.
- And its language/problem is said to be in the class NP.

Example: NP

- The Knapsack problem is definitely in NP, even using the conventional binary representation of integers.
- Use nondeterminism to guess one of the subsets.
- Sum the two subsets and compare.

P versus NP

- Originally a curiosity of Computer Science, mathematicians now recognize as one of the most important open problems the question P = NP?
- There are thousands of problems that are in NP but appear not to be in P.
- But no proof that they aren't really in P.

Complete Problems

- One way to address the P = NP question is to identify complete problems for NP.
- An NP-complete problem has the property that if it is in P, then every problem in NP is also in P.
- Defined formally via polytime reductions.

Complete Problems: Intuition

- A complete problem for a class embodies every problem in the class, even if it does not appear so.
- Strange but true: Knapsack embodies every polytime NTM computation.

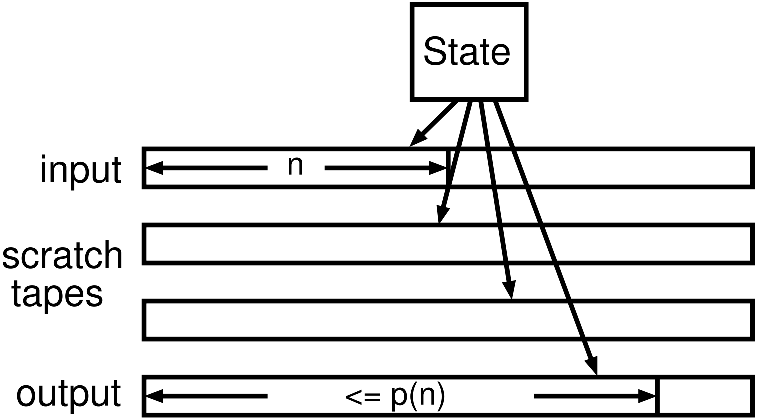
Polytime Reductions

- **Goal:** Find a way to show problem $\mathcal{X}$ to be NP-complete by reducing every language/problem in NP to $\mathcal{X}$ in such a way that if we had a deterministic polynomial time algorithm for $\mathcal{X}$, then we could construct a deterministic polynomial time algorithm for any problem in NP.

Polytime Reductions

- We need the notion of a polytime transducer - a TM that:
  1. Takes an input of length n.
  2. Operates deterministically for some polynomial time $p(n)$.
  3. Produces an output on a separate output tape.
- **Note:** output length is at most $p(n)$.

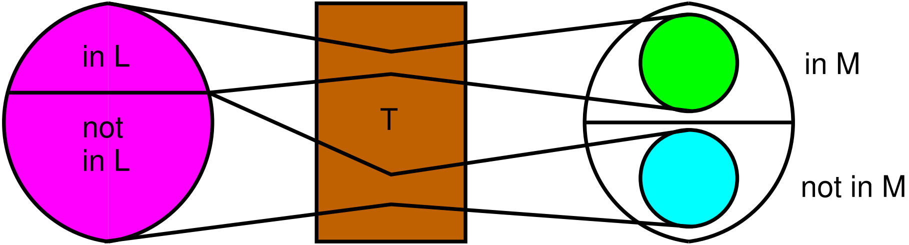
Polytime Reductions



Polytime Reductions

- Let L and M be languages.
- Say L is polytime reducible to M if there is a polytime transducer T such that for every input w to T, the output $x = T(w) \in M$ if and only if $w \in L$.

Picture of Polytime Reductions



NP-complete Problems

- A problem/language M is said to be NP-complete if for every language $L \in NP$, there is a polytime reduction from L to M.
- **Fundamental Property:** If M has a polytime algorithm, then L also has a polytime algorithm.
  - i.e., if $M \in P$, then every $L \in NP$ is also in P, or $P = NP$.

Proof that Polytime Reductions Work

- Suppose M has an algorithm of polynomial time q(n).
- Let L have a polytime transducer T to M, taking polynomial time p(n).
- The output of T, given an input of length n, is at most of length p(n).
- The algorithm for M on the output of T takes time at most q(p(n)).

Proof that Polytime Reductions Work

- We now have a polytime algorithm for L:
  1. Given w of length n, use T to produce x of length $\leq p(n)$, taking time $\leq p(n)$.
  2. Use the algorithm for M to tell if $x \in M$ in time $\leq q(p(n))$.
  3. Answer for w is whatever the answer for x is.
- Total time $\leq p(n) + q(p(n)) =$ a polynomial.

Boolean Expressions

- Boolean, or propositional-logic expressions are built from variables and constants using the operators AND, OR, and NOT.
    - Constants are true and false, represented by 1 and 0, respectively.
    - We'll use concatenation for AND, + for OR, - for NOT.

Example: Boolean Expressions

- (x + y)(-x + -y) is true only when variables x and y have opposite truth values.
- **Note:** parentheses can be used at will, and are needed to modify the precendence order NOT (highest), AND, OR.

The Satisfiability Problem (SAT)

- Study of boolean functions generally is concerned with the set of truth assignments (assignments of 0 or 1 to each of the variables) that make the function true.
- NP-completeness needs only a simpler question (SAT): does there exist a truth assignment making the function true?

Example: SAT

- (x + y)(-x + -y) is satisfiable.
- There are, in fact, two satisfying truth assignments:
  1. x=0; y=1.
  2. x=1; y=0.
- x(-x) is not satisfiable.

SAT is in NP

- There is a multitape NTM that can decide if a Boolean formula of length n is satisfiable.
- The NTM takes $O(n^2)$ time along any path.
- Use nondeterminism to guess a truth assignment on a second tape.
- Replace all variables by guessed truth values.
- Evaluate the formula for this assignment.
- Accept if true.

Cook's Theorem

SAT is NP-complete.

Picture so far

- We have one NP-complete problem: SAT.
- In the future, we shall do polytime reductions of SAT to other problems, thereby showing them to be NP-complete.
- **Why?** If we polytime reduce SAT to $\mathcal{X}$, and $\mathcal{X} \in$ P, then so is SAT, and therefore so is all of NP.

Conjunctive Normal Form

- A Boolean formula is in Conjunctive Normal Form (CNF) if it is the AND of clauses.
- Each clause if the OR of literals.
- Each literal if either a variable or the negation of a variable.

CSAT

Is a boolean formula in CNF satisfiable?

NP-completeness of CSAT

- You can convert any formula to CNF.
- If may exponentiate the size of the formula and therefore take time to write down that is exponential in the size of the original formula, but these numbers are all fixed for a given NTM M and independent of n.

k-SAT

- If a boolean formula is in CNF and every clause consists of exactly k literals, we say that the boolean formula is an instance of k-SAT.
  - Say the formula is in k-CNF.
- **Example:** 3-SAT formula:

$$(x + y + z)(x + -y + z)(x + y + -z)(x + -y + -z)$$

k-SAT Facts

- Every boolean formula has an equivalent CNF formula.
  - But the size of the CNF formula may be exponential in the size of the original.
- Not every boolean formula has a k-SAT equivalent.
- 2-SAT is in P, 3-SAT is NP-complete.

Proof: 2-SAT is in P (sketch)

- Pick an assignment for some variable, say x = true.
- Any clause with -x forces the other literal to be true.
  - **Example:** (-x + -y) forces y to be false.
- Keep seeing what other truth values are forced by variables with known truth values.

Proof: 2-SAT is in P (sketch)

- One of three things can happen:
  1. You reach a contradiction (e.g., z is forced to be both true and false).
  2. You reach a point where no more variables have their truth value forced, but some clauses are not yet made true.
  3. You reach a satisfying truth assignment.

Proof: 2-SAT is in P (sketch)

- **Case 1:** (Contradition) There can be only a satisfying assignment if you use the other truth value for x.
  - Simplify the formula by replacing x by this truth value and repeat the process.
- **Case 3:** You found a satisfying assignment, so answer yes.

Proof: 2-SAT is in P (sketch)

- **Case 2:** (You force values for some variables, but other variables and clauses are not affected.)
    - Adopt these truth values, eliminate the clauses that they satisfy, and repeat.
- In cases 1 and 2 you have spend $O(n^2)$ time and have reduced the length of the formula by ≥ 1, so $O(n^3)$ total.

3-SAT

- This problem is NP-complete.
- Clearly it is in NP, since SAT is.
- It is not true that every boolean formula can be converted to an equivalent 3-CNF formula, even if we exponentiate the size of the formula.

3-SAT

- But we don't need equivalence.
- We need to reduce every CNF formula F to some 3-CNF formula that is satisfiable if and only if F is.
- Reduction involves introducing new variables into long clauses, so that we can split them apart.

Reduction of CSAT to 3-SAT

- Let $(x_1 + x_2 + \ldots + x_n)$ be a clause in some CSAT instance, with $n \geq 4$.
  - **Note:** the x's are literals, not variables; any of them could be negated variables.
- Introduce new variables $y_1, \ldots, y_{n-3}$ that appear in no other clause.

Reduction of CSAT to 3-SAT

- Replace $(x_1 + x_2 + \ldots + x_n)$ by $(x_1+x_2+y_1)(x_3+y_2+ -y_1)\ldots$ $(x_{n-2}+y_{n-3}+ -y_{n-4})(x_{n-1}+x_n+ -y_{n-3})$.
- If there is a satisfying assignment of the x's for the CSAT instance, then one of the literals x_i must be made true.
- Assign y_i = true if $j <$ i-1 and y_j = false for larger j.

Reduction of CSAT to 3-SAT

- We are not done.
- We also need to show that if the resulting 3SAT instance is satisfiable, then the original CSAT instance was satisfiable.

Reduction of CSAT to 3-SAT

- Suppose $(x_1+x_2+y_1)(x_3+y_2+ -y_1)\ldots(x_{n-2}+y_{n-3}+ -y_{n-4})(x_{n-1}+x_n+ -y_{n-3})$ is satisfiable, but none of the x's is true.
- The first clause forces $y_1 =$ true.
- Then the second clause forces $y_2 =$ true.
- And so on . . . all the y's must be true.
- But then the last clause is false.

Reduction of CSAT to 3-SAT

- There is a little more to the reduction, for handling clauses of 1 or 2 literals.
- Replace (x) by $(x+y_1+y_2)(x+y_1+ -y_2)(x+ -y_1+y_2)(x+ -y_2+ -y_2)$.
- Replace (w+x) by (w+x+y)(w+x+ -y).

CSAT to 3-SAT Running Time

- This reduction is surely polynomial.
- In fact, it is linear in the length of the CSAT instance.
- Thus, we have polytime-reduced CSAT to 3-SAT.
- Since CSAT is NP-complete, so is 3-SAT.