Reductions for search problems

- Problem A is p-reducible to problem B (denoted $A \leq_p B$) if an "oracle" (subroutine) for B can be used to efficiently solve A
- Same definition can be extended to search problems
- Have we seen examples in class before?
- Relation between the search problem Maximum Flow and the search problem Linear Programming?
- The decision problem Circulation and the search problem Maximum Flow?

Self-reducibility

- What about the search vs decision versions of the *same* problem?
- A problem is self-reducible if the search version reduces to the decision version
- SAT is self-reducible (in fact, any NP-complete problem is!)

Cook-Levin Theorem

- We did not prove "the first NP-completeness" result

- Theorem: Exact 3SAT is NP-complete
  - We need to prove this without using any other "known NP-complete" problem
  - We want to directly show that *every problem in NP* can be reduced to Exact 3SAT

- We will first reduce any NP problem to SAT, and then reduce SAT to Exact 3SAT

Cook-Levin Theorem

- We're not going to prove it in this class, but the key idea is as follows

  - If a problem is in NP, then ∃ Turing machine $T(x, y)$ which
    - takes as input a problem instance x and an advice y of size $p(|x|)$
    - verifies in $q(|x|)$ time whether x is a YES instance
    - both p and q are polynomials

  - x is a YES instance iff $\exists y \; T(x, y) = ACCEPT$

Cook-Levin Theorem

NOT IN SYLLABUS

- x is a YES instance iff $\exists y\ T(x, y) = ACCEPT$
  - We need to convert $\exists y\ T(x, y) = ACCEPT$ into whether a SAT formula φ is satisfiable

- Recall that a Turing machine T consists of a memory tape, a head pointer, a state, and a transition function

- What describes T at any given step of its computation?
  - What is written in each cell of its memory tape?
  - Which cell of the tape is the read/write head currently pointing to?
  - What state is the Turing machine in?

Cook-Levin Theorem

NOT IN SYLLABUS

- x is a YES instance iff $\exists y\ T(x, y) = ACCEPT$
  - We need to convert $\exists y\ T(x, y) = ACCEPT$ into $\exists z\ \varphi(z) = TRUE$, where z consists of Boolean variables and φ is a SAT formula

- Variables:
  - $T_{i,j,k}$ = True if machine's tape cell i contains symbol j at step k of the computation
  - $H_{i,k}$ = True if the machine's read/write head is at tape cell i at step k of the computation
  - $Q_{q,k}$ = True if machine is in state q at step k of the computation
  - Cell index i and computation step k only need to be polynomially large as T works in polynomial time

Cook-Levin Theorem

NOT IN SYLLABUS

- x is a YES instance iff $\exists y\ T(x, y) = ACCEPT$
  - We need to convert $\exists y\ T(x, y) = ACCEPT$ into $\exists z\ \varphi(z) = TRUE$, where z consists of Boolean variables and φ is a SAT formula
- Clauses:
  - Express how the variables must be related using the transition function
  - Express that the Turing machine must reach the state $ACCEPT$ at some step of the computation
- This establishes that SAT is NP-complete.
- Next: SAT $\leq_p$ Exact 3SAT.

Cook-Levin Theorem

- Claim: SAT $\leq_p$ Exact 3SAT
  - Take an instance $\varphi = C_1 \wedge C_2 \wedge \cdots$ of SAT
  - Replace each clause with multiple clauses with exactly 3 literals each

  - For a clause with one literal, $C = \ell_1$:
    - Add two variables z_1, z_2, and replace C with four clauses

      $(\ell_1 \vee z_1 \vee z_2) \wedge (\ell_1 \vee \bar{z}_1 \vee z_2) \wedge (\ell_1 \vee z_1 \vee \bar{z}_2) \wedge (\ell_1 \vee \bar{z}_1 \vee \bar{z}_2)$
    - Verify that this is logically equivalent to ℓ_1

  - For a clause with two literals, $C = (\ell_1 \vee \ell_2)$:
    - Add variable z_1 and replace it with the following:

      $$(\ell_1 \vee \ell_2 \vee z_1) \wedge (\ell_1 \vee \ell_2 \vee \bar{z}_1)$$
    - Verify that this is logically equal to $(\ell_1 \vee \ell_2)$

Cook-Levin Theorem

- Claim: SAT $\leq_p$ Exact 3SAT
  - For a clause with three literals, $C = \ell_1 \vee \ell_2 \vee \ell_3$:
    - Perfect. No need to do anything!
  - For a clause with 4 or more literals, $C = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k)$:
    - Add variables $z_1, z_2, \ldots, z_{k-3}$ and replace it with:

      $$(\ell_1 \vee \ell_2 \vee z_1) \wedge (\ell_3 \vee \bar{z}_1 \vee z_2) \wedge (\ell_4 \vee \bar{z}_2 \vee z_3) \wedge \cdots$$
      $$\wedge (\ell_{k-2} \vee \bar{z}_{k-4} \vee z_{k-3}) \wedge (\ell_{k-1} \vee \ell_k \vee \bar{z}_{k-3})$$
    - Check:
      - If any ℓ_i is TRUE, then there exists an assignment of z variables to make this TRUE
      - If all ℓ_i are FALSE, then no assignment of z variables will make this TRUE

NP vs co-NP

- Complements of each other
  - NP = short proof for YES, co-NP = short proof for NO
  - If a problem "Does there exist…" is in NP, then its complement "Does there not exist…" is in co-NP, and vice-versa
  - The same goes for NP-complete and co-NP-complete

- Example
  - SAT is NP-complete ("Does there exist x satisfying φ?")
    - So "Does there exist no x satisfying φ?", i.e., "Is φ always FALSE?" is coNP-complete
  - Then, Tautology ("Is φ always TRUE?") is also coNP-complete

NP ∩ co-NP

- Clearly, P ⊆ NP ∩ co-NP
  - No advice needed; can just solve the problem in polytime
  - Major open question: Is P = NP ∩ co-NP?

- NP ∩ co-NP: Short proof of both YES and NO
  - Hunt for problems not known in P but still in NP ∩ co-NP

NP ∩ co-NP

- Linear programming
  - [Gale–Kuhn–Tucker 1948]: LP is in NP ∩ co-NP
  - Question: max objective value ≥ threshold?
  - Proof of YES: Provide a feasible solution with objective ≥ threshold
  - Proof of NO: Provide optimal primal and dual solutions

CHAPTER XIX

LINEAR PROGRAMMING AND THE THEORY OF GAMES [1]

BY DAVID GALE, HAROLD W. KUHN, AND ALBERT W. TUCKER [2]

The basic "scalar" problem of *linear programming* is to maximize (or minimize) a linear function of several variables constrained by a system of linear inequalities [Dantzig, II]. A more general "vector" problem calls for maximizing (in a sense of partial order) a system of linear functions of several variables subject to a system of linear inequalities and, perhaps, linear equations [Koopmans, III]. The purpose of this chapter is to establish theorems of duality and existence for general "matrix" problems of linear programming which contain the "scalar" and "vector" problems as special cases, and to relate these general problems to the theory of zero-sum two-person games.

NP ∩ co-NP

- Linear programming
  - ➢ But later, Khachiyan [1979] proved that LP is in P

ЖУРНАЛ
ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И МАТЕМАТИЧЕСКОЙ ФИЗИКИ

Том 20 Январь 1980 Февраль № 1

УДК 519.852

ПОЛИНОМИАЛЬНЫЕ АЛГОРИТМЫ В ЛИНЕЙНОМ ПРОГРАММИРОВАНИИ

Л. Г. ХАЧИЯН

(Москва)

Построены точные алгоритмы линейного программирования, трудоемкость которых ограничена полиномом от длины двоичной записи задачи.

NP ∩ co-NP

- Primality testing ("Is n a prime?")
  - [Pratt 1975]: PRIMES is in NP ∩ co-NP
  - Proof of NO: Easy, provide a non-trivial factor
  - Proof of YES: relies on interesting math

SIAM J. Comput.
Vol. 4, No. 3, September 1975

EVERY PRIME HAS A SUCCINCT CERTIFICATE*

VAUGHAN R. PRATT†

Abstract. To prove that a number n is composite, it suffices to exhibit the working for the multiplication of a pair of factors. This working, represented as a string, is of length bounded by a polynomial in $\log_2 n$. We show that the same property holds for the primes. It is noteworthy that almost no other set is known to have the property that short proofs for membership or nonmembership exist for all candidates without being known to have the property that such proofs are easy to come by. It remains an open problem whether a prime n can be recognized in only $\log_2^\alpha n$ operations of a Turing machine for any fixed α.

The proof system used for certifying primes is as follows.

Axiom. $(x, y, 1)$.

Inference Rules.

$$R_1: \quad (p, x, a), q \vdash (p, x, qa) \quad \text{provided } x^{(p-1)/q} \not\equiv 1 \pmod p \text{ and } q \mid (p-1).$$

$$R_2: \quad (p, x, p-1) \vdash p \quad \text{provided } x^{p-1} \equiv 1 \pmod p.$$

Theorem 1. *p is a theorem $\equiv$ p is a prime.*

Theorem 2. *p is a theorem $\supset$ p has a proof of $\lceil 4 \log_2 p \rceil$ lines.*

NP ∩ co-NP

- Primality testing ("Is n a prime?")
  - Later, Agrawal, Kayal, and Saxena [2004] proved that PRIMES is in P
    - Milestone result!

Annals of Mathematics, **160** (2004), 781–793

PRIMES is in P

By Manindra Agrawal, Neeraj Kayal, and Nitin Saxena*

Abstract

We present an unconditional deterministic polynomial-time algorithm that determines whether an input number is prime or composite.

NP ∩ co-NP

- Factoring ("Does n have a factor $\leq k$?")
  - FACTOR is in NP ∩ co-NP
    - Proof of YES: Just present such a factor
    - Proof of NO:
      - Present the entire prime factorization of n along with a short proof that each presented factor is a prime
      - Verifier TM can check that each factor is indeed a prime, their product is indeed n, and none of the factors is $\leq k$
        - Actually, proofs of primality are not required anymore since we know the TM can just run the AKS algorithm to check if the factors are prime

NP ∩ co-NP

- Factoring ("Does n have a factor $\leq k$?")
  - Major open question: Is FACTOR in P?
    - Basis of several cryptographic procedures
  - Challenge: Factor the following number.

7403756347956171282804679609742957314259318888923128908493623263897276503402826627689199641962511784399589433050212758537011896809828673317327310893090055250511687706329907239638078671008609696253793465056379635

RSA-704
(A $30,000 prize was claimed in 2012 for this)

NP ∩ co-NP

- Factoring (“Does n have a factor $\leq k$?”)
  - [Shor 1994]: We can factor an n-bit integer in $O(n^3)$ steps on a quantum computer.
  - *Scalable* quantum computers can help
    - 2001: Factored 15 = 3 x 5
    - 2012: Factored 21 = 3 x 7

Other Complexity Classes

- Based on the exact time complexity
  - DTIME(n), NTIME(n^2), …
    - Deterministic / nondeterministic time complexity
- Based on space complexity
  - DSPACE(n), NSPACE($\log n$)
- Using randomization
  - ZPP (expected polynomial time, no errors)
    - Is P = ZPP?
- Allowing probabilistic errors
  - RP (polynomial time, one-sided error)
  - BPP (polynomial time, two-sided erros)