

M, 11/04/19

Fall'19 CSCE 629

Analysis of Algorithms

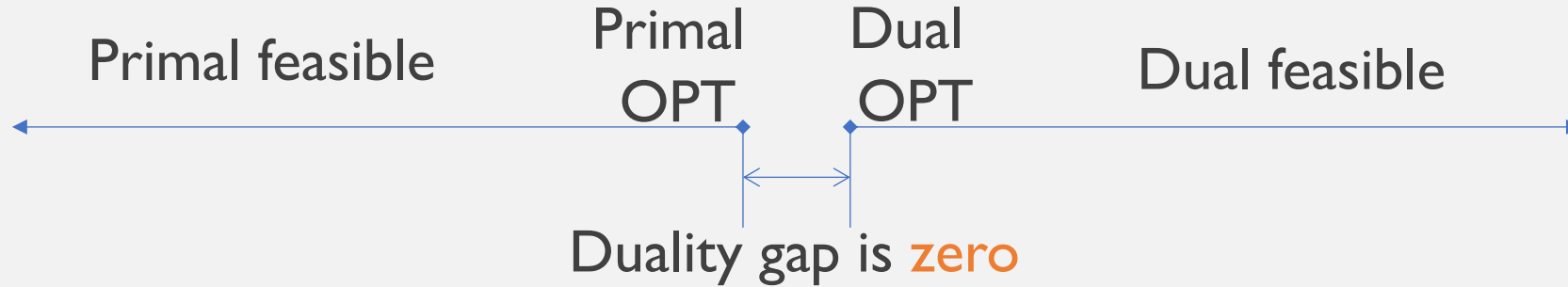
Fang Song
Texas A&M U

Lecture 25

- Computational intractability

Credit: based on slides by A. Smith & K. Wayne

Fundamental theorem of linear programming



(Primal) Max $c^T x$
Subject to:
 $Ax \leq b$
 $x \geq 0$

(Dual) Min $y^T b$
Subject to:
 $y^T A \geq c^T$
 $y \geq 0$

- **Weak duality.** If x is a feasible solution for a linear program Π , and y is a feasible solution for its dual \sqcup , then $c^T x \leq y^T Ax \leq y^T b$.
- **Strong duality.** Π has an optimal solution and x^* **if and only if** its dual \sqcup has an optimal solution y^* such that $c^T x = y^T Ax = y^T b$.

Duality example

(P) Maximize: $x_1 + 5x_2$
Subject to:

$$0 \leq x_1 \leq 12$$

$$0 \leq x_2 \leq 15$$

$$x_1 + x_2 \leq 24$$

Max = 84, $x_1 = 9, x_2 = 15$

(D) Minimize: $12y_1 + 15y_2 + 24y_3$
Subject to:

$$y_1 + y_3 \geq 1$$

$$y_2 + y_3 \geq 5$$

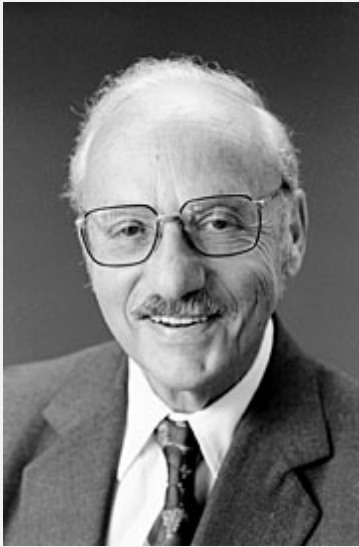
$$y_1, y_2, y_3 \geq 0$$

Min = 84, $y_1 = 0, y_2 = 4, y_3 = 1$

(magic) multipliers

Certificate: $x_1 + 5x_2 = 4 \cdot x_2 + 1 \cdot (x_1 + x_2) \leq 4 \cdot 15 + 24 = 84$

A dialogue between Dantzig & von Neumann



George Dantzig

Let me show you my exciting finding: simplex algorithm for LP
... .. [next 30 mins]

Get to the point, please!

OK! Em... To be concise ... [next 3 mins]

Ah, that!



John von Neumann

[next 60 mins]
.... (convexity)... (fixed point) ... (2-player game) ...
so, there is duality which'd follow by my min-max theorem ...

For any matrix A , $\min_x \max_y xAy = \max_y \min_x xAy$.

A reflection on the algorithmic journey

■ So far: algorithm design triumph

- Divide-and-conquer
- Greedy
- Dynamic programming
- Linear programming (duality)
- Local search
- Randomization
- ...

Examples

- $O(n \log n)$ Merge sort
- $O(n \log n)$ interval scheduling
- $O(n^2)$ edit distance
- $O(n^3)$ bipartite matchin

■ New goal: understand what is hard to compute

Computational intractability

- **Computability:** can you solve it, in principle?

Halting problem is **uncomputable** [Given program code, will this program terminate or loop indefinitely?]

Church-Turing Thesis. A function can be computed in any *reasonable* model of computation **iff.** it is computable by a **Turing machine**.

- **Complexity:** can you solve it, under resource constraints?

Extended Church-Turing Thesis. A function can be computed **efficiently** in any *reasonable* model of computation **iff.** it is efficiently computable by a **Turing machine**.

Disprove ECT???



Quantum supremacy using a programmable superconducting processor

Central ideas in complexity

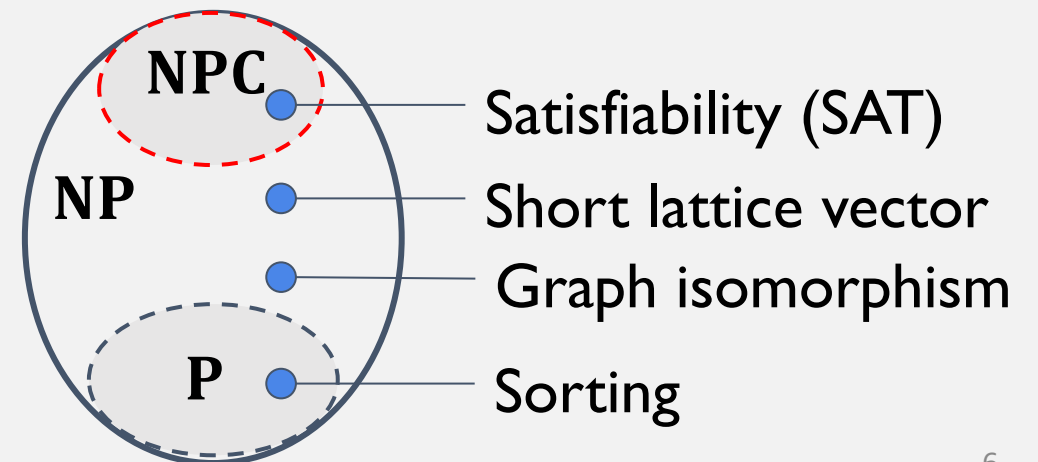
■ Poly-time as “feasible”

- Most natural problems either are easy (e.g., n^3) or no poly-time alg. known

■ Reduction : relating hardness ($A \leq B \Rightarrow A$ no harder than B)

■ Classify problems by “hardness”

- P = {problems that are easy to answer}
- NP = {problems that are easy to **verify** given **hint**} [lots of examples, stay tuned!]
- **Complete** problems: “hardest” in a class



What'd be considered “feasible”?

Q. Which problems will we be able to solve in practice?

A. Those with poly-time algorithms. [von Neumann1953,Godel1956, Cobham1964,Edmonds1965,Rabin1966]

YES	Probably No
Shortest path	Longest path
Matching	3D-matching
Min cut	Max cut
2-SAT	3-SAT
Planar 4-color	Planar 3-color
Bipartite vertex cover	Vertex cover
Primality	Factoring

Classify problems

Desiderata. Classify problems as those that can be solved in polynomial-time and those that cannot.

- **Provably require exponential time.**

- Given a **Turing machine**, does it HALT in at most k steps?
- Given a board position in an $n \times n$ generalization of chess, can black win?

Roughly: C program on machine with infinite memory

☹️ **Frustrating news:** Huge number of fundamental problems have defied classification for decades.

- We will show: these problems are “computationally equivalent ” and appear to be different manifestations of one hard problem.

Tool: polynomial-time reduction

Desiderata'. Suppose we can solve Y in poly-time. What else could we solve in polynomial time?

- **Reduction.** Problem X **polynomial reduces to** Problem Y if **arbitrary** instance of X can be solved using:
 - Polynomial number of standard computation steps
 - & polynomial number of calls to **oracle** that solves A

Notation. $X \leq_{P, Cook} Y$ (or $X \leq_P Y$)

! Mind your direction, don't confuse $X \leq_P Y$ with $Y \leq_P X$

N.B. We pay for time to write down instances to oracle
 \Rightarrow instances of Y must be of polynomial size.

What polynomial-time reductions buy us

- **Design algorithms.** If $X \leq_p Y$ and Y can be solved in poly-time, then X can also be solved in polynomial time.
- **Establish intractability.** If $X \leq_p Y$ and X **cannot** be solved in poly-time, then Y cannot be solved in polynomial time.
- **Establish equivalence.** If $X \leq_p Y$ and $Y \leq_p X$, then $X \equiv_p Y$.

Bottomline. Reductions classify problems acc. to **relative** difficulty

Quiz

■ Which of the following poly-time reductions are known?

- A. $\text{FIND-MAX-FLOW} \leq_P \text{FIND-MIN-CUT}$
- B. $\text{FIND-MIN-CUT} \leq_P \text{FIND-MAX-FLOW}$
- C. Both A and B
- D. Neither A nor B

VALUES VS. ACTUAL FLOW/CUT