Introduction to approximation and randomized algorithms
NDMI084

Assumed knowledge: essentials of complexity theory

linear programming

probability

What is this course about?

Think about an NP-hard problem Q. Unless P=NP, there is no algorithm that

1 in polynomial time RELAX ⇒ 1 heuristics, exp. time algorithms

2 for any instance of Q 2 special instances, e.g. planar

3 finds an optimal solution 3 provably "almost" optimal

= approximation algorithms

Randomized algorithms - exploit randomness

quality of the solution or the runtime depend on the random choices

on average, the solution is good (fast)

Why randomized algorithms?

- no deterministic ones (e.g., cryptography, distributed algorithms)
- simplicity
- efficiency

Example: n friends want to compute their average salary or SUM

but nobody wants to reveal his/her own salary

Assumptions: salaries - non-negative integers bounded by B

no outside trusted party

secure private communication for all pairs of friends

Requirement: even if k friends collude, they will not learn more then the sum of the remaining n-k salaries

How to do it?

Friends $F_1, F_2, \dots, F_n$

Salaries $S_1, S_2, \dots, S_n \le B$

Protocol 1

$N = n \cdot B$

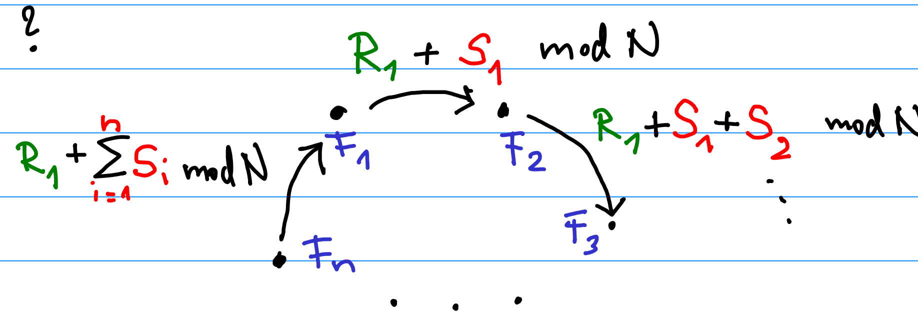
F_1 picks a random number $R_1 \in [0, N)$
and passes $S_1 + R_1 \bmod N$ to F_2

F_i, $i > 1$, adds S_i to the number from F_{i-1}
and passes the result mod N to F_{i+1} (to F_1, in case of F_n)

F_1 substracts R_1 from the result, mod N
and announces the outcome to everyone

Any difficulty?

$X \bmod N = X - \left\lfloor \frac{X}{N} \right\rfloor \cdot N$



Protocol 2

F_i chooses $n-1$ integers $R_{i,1}, R_{i,2}, \dots R_{i,n-1}$, ind., rand, $\in [0, N)$
calculates $R_{i,n}$ s.t. $R_{i,1} + R_{i,2} + \dots + R_{i,n} = S_i \pmod N$

F_i sends $R_{i,j}$ to F_j, for $i = 1, \dots, n$

F_i announces $R_{1,i} + R_{2,i} + \dots + R_{n,i} \bmod N$

F_i sums the numbers announced by all $F_1, \dots, F_n$, again mod N

$$
\begin{array}{lccccccccl}
F_1 & R_{11} & + & R_{12} & + & \cdots & + & R_{1n} & = & S_1 \pmod N \\
F_2 & R_{21} & + & R_{22} & + & \cdots & + & R_{2n} & = & S_2 \pmod N \\
\vdots & & & & & \vdots & & & & \\
F_n & R_{n1} & + & R_{n2} & + & \cdots & + & R_{nn} & = & S_n \pmod N \\
 & \downarrow & & \downarrow & & & & \downarrow & & \\
 & F_1 & & F_2 & & & & F_n & &
\end{array}
$$

Note:

$$
\underbrace{\sum_{i=1}^{n} \Big(\overbrace{\sum_{j=1}^{n} R_{ij} \bmod N}^{S_i} \Big)}_{\text{sum of salaries}} = \underbrace{\sum_{j=1}^{n} \Big(\overbrace{\sum_{i=1}^{n} R_{ij} \bmod N}^{\to F_j} \Big) \bmod N}_{\text{outcome of Protocol 2}}
$$

BASIC DEFINITIONS

Def. Optimization problem a quadraple $(\mathcal{I}, \mathcal{F}, f, g)$ where

$\mathcal{I}$ is the set of all valid instances .... inputs

$\mathcal{F}$ is a function specifying for each instance $I \in \mathcal{I}$ a set of feasible solutions $\mathcal{F}(I)$

f is a function specifying for each $I \in \mathcal{I}$ and each feasible solution $S \in \mathcal{F}(I)$ the cost of the solution ... objective function

g is min or max

Example: the shortest path problem

$\mathcal{I}$ ... set of pairs $(G(V,E), \{s,t\} \in V^2)$

$\mathcal{F}$ ... for each $G(V,E)$ and $\{s,t\} \in V^2$, set of all s-t-paths

$f(P)$ ... the lenght of the path P (= # of edges)

g = min

Def. NP-Optimization problem is an optimization problem s.t.

1. instances are finite strings
2. for each $I \in \mathcal{I}$ and each $S \in \mathcal{F}(I)$, $|S| \le \mathrm{poly}(|I|)$
3. there exists a poly-time decision procedure that tests for each I and $S \in \Sigma^*$ whether $S \in \mathcal{F}(I)$
4. f is poly-time computable

For an instance I of an optimization problem

$OPT(I)$ denotes the optimal value of a feasible solution, i.e., $\min_{S \in \mathcal{F}(I)} f(I,S)$ (or max ....)

For an algorithm A for an optimization problem, $A(I)$ denotes the value of the objective function f for the solution of the algorithm

For randomized algorithms, $A(I)$ is a random value

Def. Algorithm A has approximation ratio R if R ... constant or function

A is a poly-time algorithm that for each I finds a feasible solution s.t.

$A(I) \le R \cdot OPT(I)$ if g=min, or $A(I) \ge \frac{OPT(I)}{R}$ if g=max

For randomized algorithms, replace $A(I)$ by $E[A(I)]$ the expected value

TRAVELING SALESMAN PROBLEM

Input: the set of cities $V = \{1, 2, \dots, n\}$

and a non-negative function specifying the distances between cities,

$d : V \times V \to \mathbb{R}_0^+$ non-existing edges $d(\cdot) = \infty$

Output: a permutation of the cities $v_1, v_2, \dots, v_n$ specifying the Hamiltonian cycle

Goal: minimize $d(v_n, v_1) + \sum_{i=1}^{n-1} d(v_i, v_{i+1})$

Note: an NP-optimization problem $\mathcal{I} = \dots,\ \mathcal{F} = \dots,\ f = \dots,\ g = \min$

HOW WELL CAN YOU APPROXIMATE OPT ?

Assume there exists an $\alpha(n)$-approximation algorithm for TSP.

Think about an instance $G = (V, E)$ of the Hamiltonian cycle problem.

Can we solve it using the $\alpha(n)$-approximation algorithm for TSP?

Reduction Ham. $\to$ TSP

$\{u,v\} \in E \quad \longrightarrow \quad d(u,v) = 1$

$\{u,v\} \notin E \quad \longrightarrow \quad d(u,v) = n \cdot \alpha(n)$

Observe: YES instance $\longrightarrow$ OPT $= n$

NO instance $\longrightarrow$ OPT $> n \cdot \alpha(n)$

$\Rightarrow$ for YES ... $A(I) \le n \cdot \alpha(n)$

NO ... $A(I) > n \cdot \alpha(n)$

That is, the $\alpha(n)$-approx. alg. can distinguish between YES and NO instances of Hamilt. prob.

– NOT possible, unless P=NP.

Theorem: Let $\alpha(n)$ be a poly-time computable function (e.g., n^3, 2^n). Unless P=NP, there is no $\alpha(n)$-approximation algorithm for TSP.