Announcements:

- final
  - thursday, 3-5pm.
  - cumulative
  - 1 page handwritten cheat-sheet

Last time:

- pseudo polynomial time
- Knapsack PTAS

Today:

- online algorithms
- ski renter
- secretary

Approximation Algorithms

"show algorithm's solution is always close to optimal solution"

Challenge: for hard problems optimal solution is complex.

Approach:

1. relax constraints and solve relaxed optimally.
2. fix violated constraints.
3. show "fixed solution" is close to "relaxed solution"

Algorithms Flow Chart

```
       [model problem]
             |
             |              yes
     <does greedy work>------------\
             |                     |
             |no                   V
             |         yes
         <similar to> ---->[reduction]
       <another problem>          |
             |                     V
             |no
             |          no                no
      <find subproblem>---><NP hard?>-------[repeat]
             |                    |yes
             |yes                 |
             |               [approximate]
  <independent or dependent?>
        |            |
        | indep      | dep
        |            |
[divide & conquer] [dynamic prog]
        |            |
        V            V

             V
             |
     [problem solved]
```

Course Topics

- Dynamic Programming
- Reductions
  - network flow
  - NP hardness
- Approximaiton Algorithms
  - Greedy algortithms

Online Algorithms

"algorithms that must make decisions without full knowledge of input"

(e.g., if input is events over time, then algorithm doesn't know future)

Ski Renter

input:

- cost to buy skis: B.
- cost to rent skis: R.
- daily weather $d_1, \ldots, d_n$ with
$$d_i = \begin{cases} 1 & \text{if good weather} \\ 0 & \text{if bad weather} \end{cases}$$
(let $k = \sum_i d_i$)

output: schedule for renting or buying skis.

online constraint: on day i do not know $d_{i+1}, \ldots, d_n$.

Note: optimality is impossible because don't know future.

Idea: approximate "optimal offline" algorithm

Algorithm: $\mathrm{O}PT$ (offline)

- if $kR < B$, buy on day 1.
- else, rent on each good day.

Performance: $\mathrm{O}PT = \min(kR, B)$.

Def: an online alg is <u>β-competitive</u> with optimal offline alg, OPT, if on all inputs x for X,

- minimization: $Alg(x) \leq \beta \mathrm{O}PT(x)$.
- maximization: $Alg(x) \geq \mathrm{O}PT(x)/\beta$.

Challenge:

- if we buy first day we ski:
  - for $d = (1, 0, 0, \ldots, 0)$
  - $OPT = R$; $Alg = B \gg R$
- if we rent each time we ski
  - for $d = (1, 1, 1, \ldots, 1)$
  - $OPT = B$; $Alg = Rn \gg B$

Algorithm: "Rent to Buy" "Rent unless total rental cost would exceed buy cost, then buy"

Example: $R = 1, B = 3$

d	1	0	1	1	1	0	1	1	...
Alg	R	/	R	R	B	/	0	0	...

$Alg = \underbrace{3R + B}_{\leq 2B}$, $OPT = B$

Theorem: $Alg \leq 2\mathrm{O}PT$ (Alg is 2-competitive)

Proof:

case 1: $kR \leq B$

- Alg: kR
- OPT: kR

$\Rightarrow Alg = \mathrm{O}PT \leq 2\mathrm{O}PT$.

case 2: $kR > B$

- Alg: total rental $+ B \leq 2B$
- OPT: B

$\Rightarrow Alg \leq 2\mathrm{O}PT$.

Note: competitive analysis gives very strong approximation result.

Secretary Problem

input:

- sequence of candidates $1, \ldots, n$.
- ordering on candidate qualities.

output:

- "hire" / "no hire" decisions.
- to hire best candidate.

online constraint: must make hire/no hire decision for i before seeing $i+1, \ldots, n$.

Fact: "optimal offline" always hires best secretary.

Claim: no deterministic algorithm approximates optimal offline.

Proof: two candidates

case 1: Alg hires 1

- 2 is better.

case 2: Alg doesn't hire 1

- 1 is better.

Idea: consider randomized algorithms.

(maximize probability of hiring the best candidate.)

Claim: randomized algorithm is n-competitive offline.

Proof:

- Alg: for all i, pick ith secretary with probability $1/n$.
- Alg is right with probability $1/n$
- OPT is always right.

$\Rightarrow$ n-competitive.

Claim: no algorithm hires best candidate with probability $\Omega(1/n)$.

Idea: consider randomized inputs.

Assumption: candidates arrive in a uniformly random order.

Example: $n = 3$

1 2 3	1 3 2	3 1 2	2 1 3	2 3 1	3 2 1
(a)	(a)	(b)	(b)	(b)	

Two algs for example:

(a) take i candidate for some i

$\Rightarrow$ $\mathbf{Pr}[\text{success}] = 1/3$

(b) look at 1st, condition choice of 2nd or 3rd.

- if 2nd better than 1st, hire 2nd
- else, hire 3rd.

$\Rightarrow$ $\mathbf{Pr}[\text{success}] = 1/2$

Algorithm: Secretary Alg

- interview k candidates but make no offers
- hire next secretary that is better than any of first k.

Lemma: For $k = n/2$ alg is 4-competitive.

Proof:

- hire best when 2nd best in first half and 1st best in second half.
- Recall: $\mathbf{Pr}[A \& B] = \mathbf{Pr}[A \mid B]\,\mathbf{Pr}[B]$.
- $\mathbf{Pr}[\text{2nd best in first half}] = 1/2$
- $\mathbf{Pr}[\text{1st best in second half} \mid \text{2nd best in first half}] = \frac{n/2}{n-1} \geq 1/2$

$\Rightarrow$ $\mathbf{Pr}$[hire best] $\geq$ $\mathbf{Pr}$[2nd in 1st 1/2] $\mathbf{Pr}$[1st in 2nd 1/2 $|$ 2nd in 1st 1/2] $\geq$ 1/4.

Question: what is best k?

Theorem: for $k = 1/e$ alg is e-competitive
and this is best possible.