Midterm

Congratulations! You did great in the midterm
Median ~ 81%
• I did terrible in midterm, can I still get 3.9 or 4.0? Yes!
• If you are way below median below 50% try harder
Final will be harder
Mid-quarter evaluations

• HW problems are too hard for me
  • We have resources to prepare for HW (problem solving section, OH, etc.). You can also practice with exercises in the book.
  • Difficult HW problems make you prepared for real world algorithm design problems

• Grading rules are too strict
  • Every week I spent hours to train TAs how to grade. The well-defined rubric is my effort to have a systematic grading guidelines that all TAs can follow. Without it everybody grades arbitrarily.
  • Everything is not about grade! We are here to learn.

• TAs have not responded to my re-grade requests
  • Send me an email or come to OH, I’ll look into your request

• What is the point of this course after all? Why do you have to prove correctness of an algorithm?
  • Often algorithms that we design are incorrect.
Approximation Algorithms
How to deal with NP-complete Problem

Many of the important problems in real world are NP-complete.

   SAT, Set Cover, Graph Coloring, TSP, Max IND Set, Vertex Cover, …

So, we cannot find optimum solutions in polynomial time. What to do instead?

• Find optimum solution of special cases (e.g., random inputs)

• Find near optimum solution in the worst case
Approximation Algorithm

Polynomial-time Algorithms with a guaranteed approximation ratio.

\[ \alpha = \frac{\text{Cost of computed solution}}{\text{Cost of the optimum}} \]

worst case over all instances.

**Goal**: For each NP-hard problem find an approximation algorithm with the best possible approximation ratio.
Vertex Cover

Given a graph $G=(V,E)$, Find smallest set of vertices touching every edge
Greedy algorithms are typically used in practice to find a (good) solution to NP-hard problems

**Strategy (1):** Iteratively, include a vertex that covers most new edges

Q: Does this give an optimum solution?
A: No,
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most

Greedy Vertex cover = 20
OPT Vertex cover = 8
Greedy (1): Pick vertex that covers the most

$n$ vertices. Each vertex has one edge into each $B_i$

Greedy pick bottom vertices $= n + \frac{n}{2} + \frac{n}{3} + \cdots + 1 \approx n \ln n$

OPT pick top vertices $= n$
A Different Greedy Rule

**Greedy 2**: Iteratively, pick both endpoints of an uncovered edge.

Vertex cover = 6
Greedy 2: Pick Both endpoints of an uncovered edge

Greedy vertex cover = 16

OPT vertex cover = 8
**Greedy (2) gives 2-approximation**

**Thm:** Size of greedy (2) vertex cover is at most twice as big as size of optimal cover

**Pf:** Suppose Greedy (2) picks endpoints of edges $e_1, \ldots, e_k$. Since these edges do not touch, every valid cover must pick one vertex from each of these edges!

i.e., $OPT \geq k$.

But the size of greedy cover is $2k$. So, Greedy is a 2-approximation.
Set Cover

Given a number of sets on a ground set of elements,

**Goal:** choose minimum number of sets that cover all.

e.g., a company wants to hire employees with certain skills.
Set Cover

Given a number of sets on a ground set of elements, 

**Goal:** choose minimum number of sets that cover all.

Set cover = 4
A Greedy Algorithm

Strategy: Pick the set that maximizes # new elements covered
A Greedy Algorithm

**Strategy:** Pick the set that maximizes # new elements covered
A Greedy Algorithm

**Strategy:** Pick the set that maximizes \# new elements covered
A Greedy Algorithm

**Strategy**: Pick the set that maximizes # new elements covered
A Greedy Algorithm

**Strategy:** Pick the set that maximizes # new elements covered

**Thm:** Greedy has $\ln n$ approximation ratio
A Tight Example for Greedy
A Tight Example for Greedy
A Tight Example for Greedy
A Tight Example for Greedy
A Tight Example for Greedy
A Tight Example for Greedy

Greedy = 5

OPT = 2
**Greedy Gives O(\log(n)) approximation**

**Thm:** If the best solution has \( k \) sets, greedy finds at most \( k \ln(n) \) sets.

**Pf:** Suppose OPT=\( k \)
There is set that covers \( 1/k \) fraction of remaining elements, since there are \( k \) sets that cover all remaining elements.
So in each step, algorithm will cover \( 1/k \) fraction of remaining elements.

\[
\text{#elements uncovered after } t \text{ steps} \leq n \left(1 - \frac{1}{k}\right)^t \leq ne^{-\frac{t}{k}}
\]

So after \( t = k \ln n \) steps, \# uncovered elements < 1.
Approximation Alg Summary

• To design approximation Alg, always find a way to lower bound OPT

• The best known approximation Alg for vertex cover is the greedy.
  – It has been open for 50 years to obtain a polynomial time algorithm with approximation ratio better than 2

• The best known approximation Alg for set cover is the greedy.
  – It is NP-Complete to obtain better than \( \ln n \) approximation ratio for set cover.
Single Source Shortest Path

Given an (un)directed graph $G=(V,E)$ with non-negative edge weights $c_e \geq 0$ and a start vertex $s$

Find length of shortest paths from $s$ to each vertex in $G$
Dijkstra(s)

- Set all vertices $v$ undiscovered, $d(v) = \infty$.
- Set $d(s) = 0$, mark $s$ discovered.

while there is edge from discovered vertex to undiscovered vertex,

- let $(u,v)$ be such edge minimizing $d(u) + c_{u,v}$

- set $d(v) = d(u) + c_{u,v}$, mark $v$ discovered.