Greedy Approximation Algorithims

CSE 421 Fall 2022
Lecture 8

Announcements

Starting with HW3, we expect you to fit your submissions into 2 pages.

Our solutions will fit into one-page LaTeXed, so you still have more space than us.

5 page submissions are bad for everyone:

- you're not learning anything new on the bottom of page 4. It would be better to spend your time on other classes, or trying the extra problems in section handouts.
- TAs have limited time to grade, we don't want them spending it skimming a needlessly long proof.

Try formatting algorithms as suggested on the style guide

Key idea (1-2 sentences); algorithm; proof; analysis.

You have HW solutions, our HW1 feedback, and section handouts for examples.

Yesterday's Slides are corrected

"Fewest overlaps" is not an optimal greedy algorithm.

This is a good general lesson; algorithms are hard, we all make mistakes.

(And you'd be shocked how many slightly or totally incorrect I found on the internet this summer).

There's a problem on HW3 that asks you to prove me wrong.

Approximation Algorithms

In 332 you learned about “NP-hard” problems. These are problems where we don’t have (or expect to ever have) polynomial time algorithms.

So what do you do if you really want to solve an NP-hard problem?

Sometimes you want an approximation algorithm!

Approximation Algorithms

What makes an approximation algorithm good?

Speed: We usually require approximation algorithms to run in polynomial time.

Accuracy:

NP-completeness will say that we can't solve the problem exactly in polynomial-time, but (without more thinking/proofs) it doesn't say we couldn't efficiently get a solution that's, say within 1% of the best solution.

Approximation Ratio

For a minimization problem (find the shortest/smallest/least/etc.)

If $OPT(\mathrm{I})$ is the value of the best solution for input I, and $ALG(I)$ is the value that your algorithm finds, then ALG is an α approximation algorithm if for every I,

$$\alpha \cdot OPT(I) \geq ALG(I)$$

i.e. you're always within an α factor of the real best.

Sometimes use big-$\mathcal{O}$ notation on the ratio.

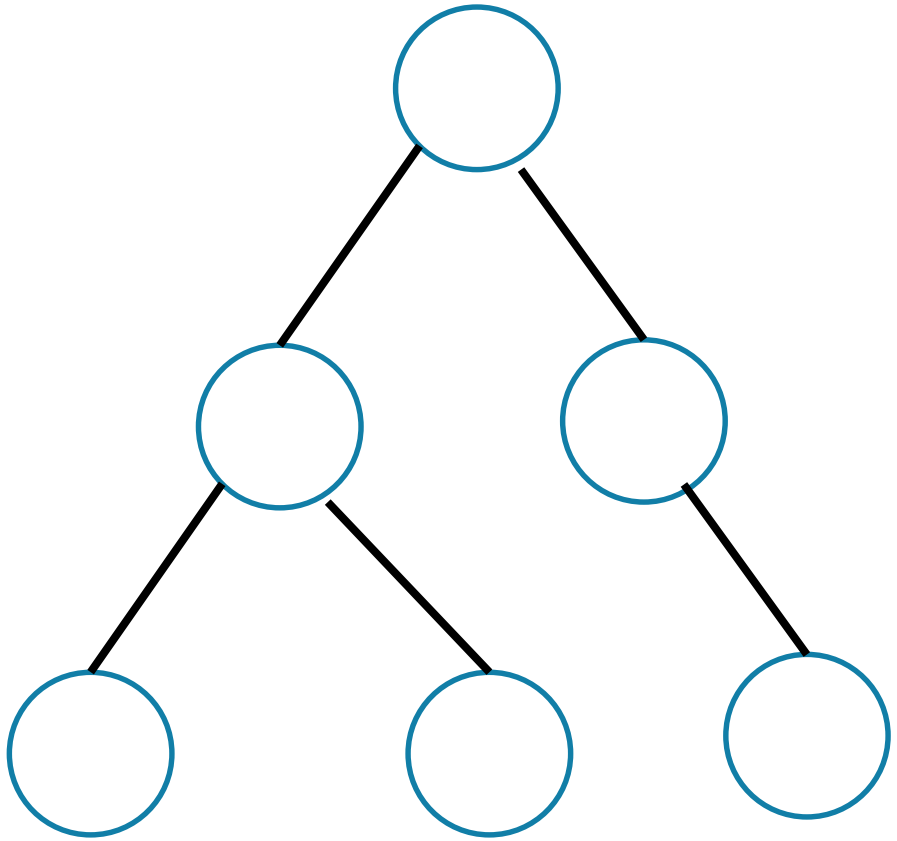
Vertex Cover

Vertex Cover

A set S of vertices is a vertex cover if for every edge (u, v): u is in S, or v is in S, (or both)

Find the minimum vertex cover in a graph.

We're picking a set of ***vertices*** so that the ***vertices*** cover every edge.



Vertex Cover

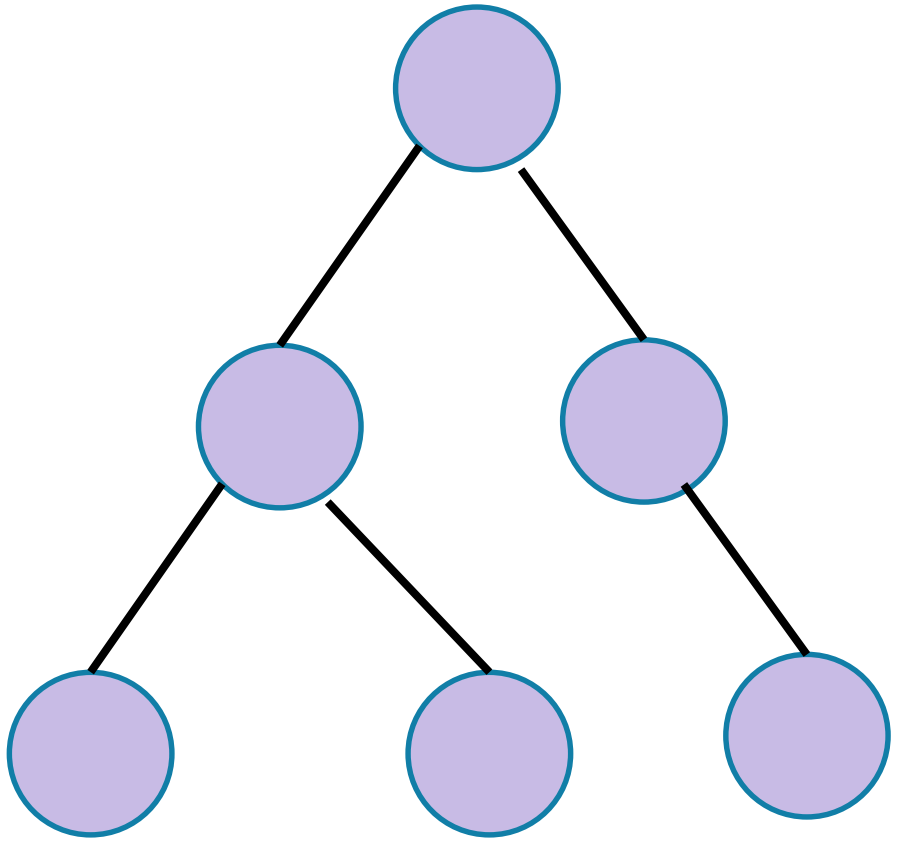
Vertex Cover

A set S of vertices is a vertex cover if for every edge (u, v): u is in S, or v is in S, (or both)

Find the minimum vertex cover in a graph.

We're picking a set of ***vertices*** so that the ***vertices*** cover every edge.

A valid vertex cover! (just take everything)
Definitely not the minimum though.



Vertex Cover

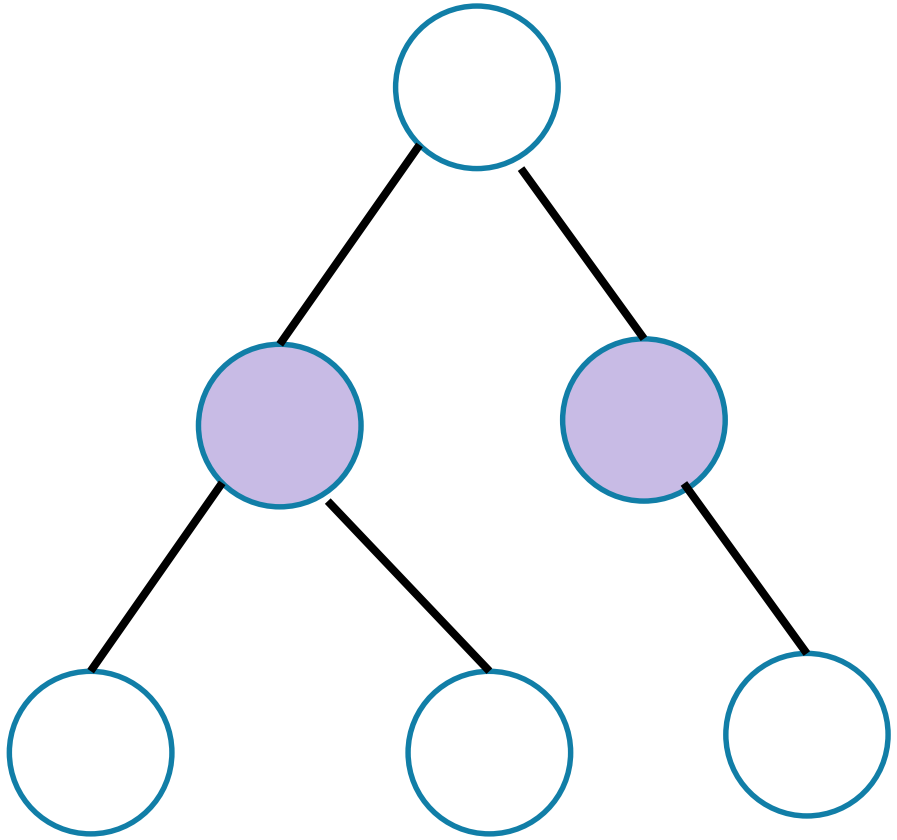
Vertex Cover

A set S of vertices is a vertex cover if for every edge (u, v): u is in S, or v is in S, (or both)

Find the minimum vertex cover in a graph.

We're picking a set of ***vertices*** so that the ***vertices*** cover every edge.

A better vertex cover – size 2 (only 2 vertices)



Vertex Cover

Take a moment, think of greedy ideas that might work for finding a small vertex cover.

Vertex Cover

Take a moment, think of greedy ideas that might work for finding a small vertex cover.

Idea 1: Maximize the "edges covered to vertices taken" ratio

Take a vertex of highest degree remaining in the graph, add it to the VC.

Delete all its incident edges

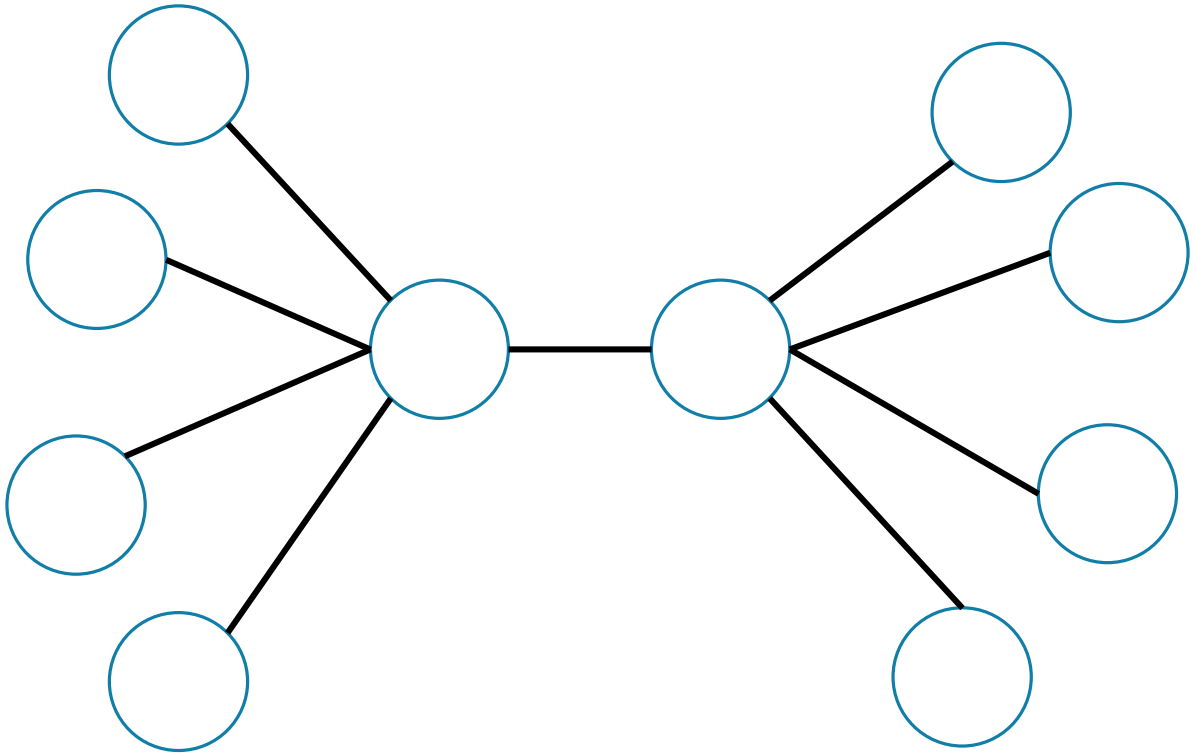
Idea 2: At least one good one

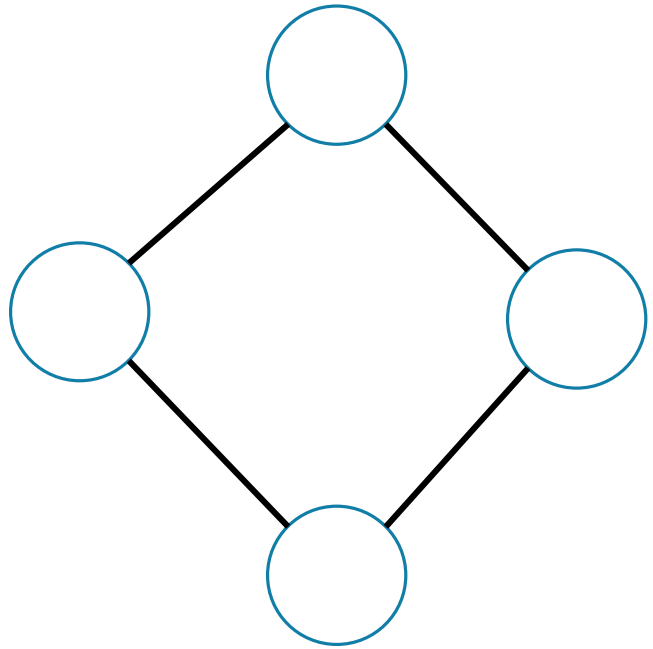
Choose an (arbitrary) edge (u, v). At least one of u, v is in the minimum VC. But both in your VC. Delete all incident edges.

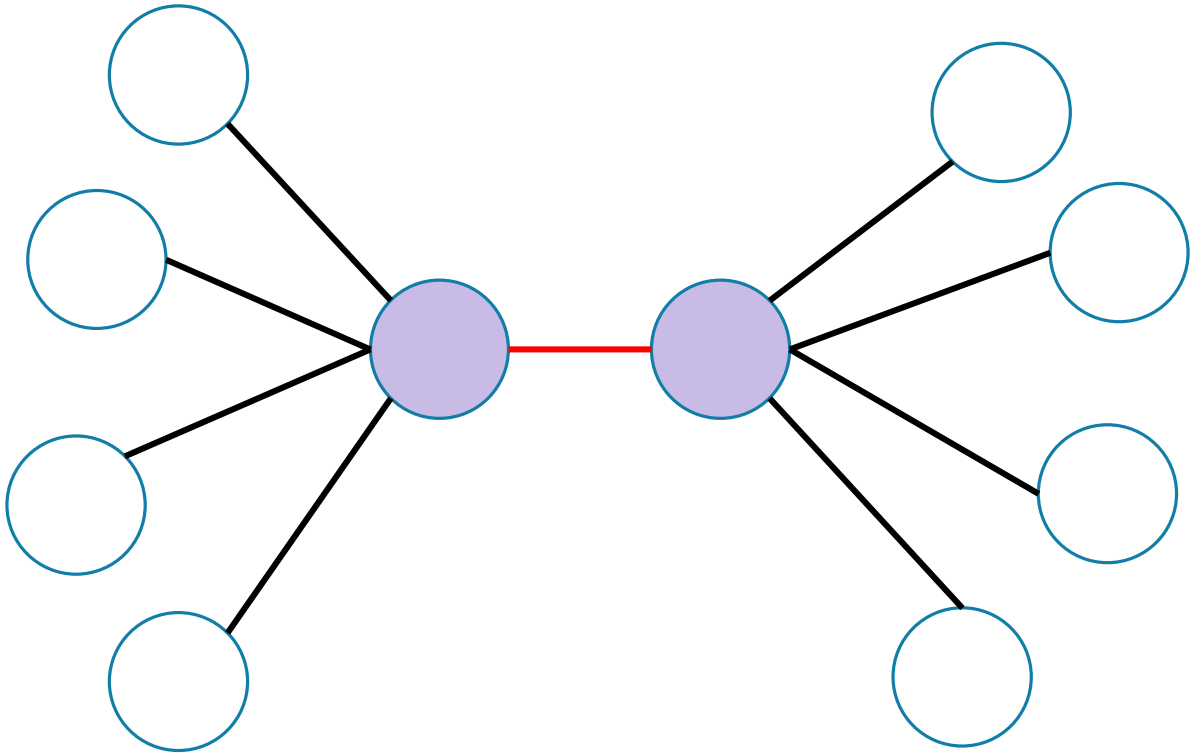
Non-optimal

You'll show on the homework that idea 1 isn't optimal.

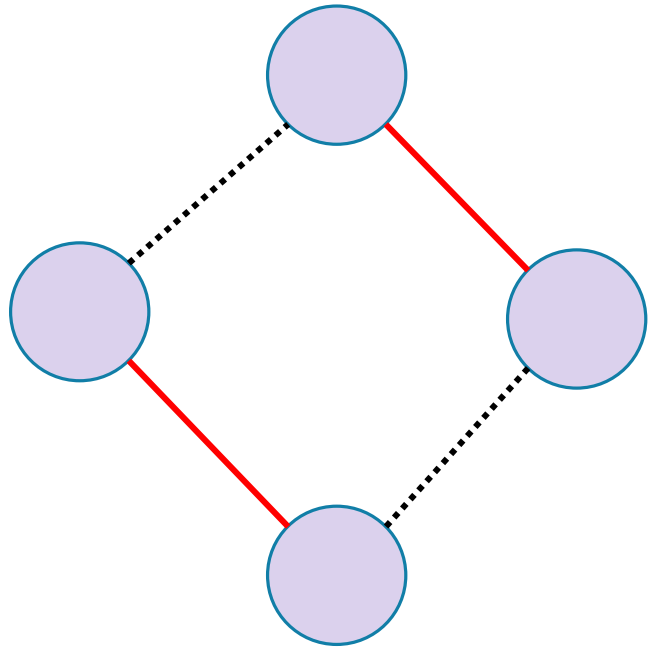
Focus on idea 2, come up with a graph where it could give you the optimal VC, and another where it doesn't.







If the algorithm selects the central edge, we get the optimal VC.



Any first edge will add two to the VC, and leave the opposite edge uncovered.
The algorithm finds a VC of size 4.
The optimal is 2.

Approximation Ratio

The ratio between $\mathbf{ALG}$ and $\mathbf{OPT}$ on that graph was only $\mathbf{2}$ $(\mathbf{4/2})$.

So is the approximation ratio for the algorithm $\mathbf{2}$?

We don't know that yet!

Remember approximation ratio is a **worst-case** measure

It's also asymptotic (we want n arbitrarily large, not $n = 11$)

But it turns out it is $\mathbf{2}$. Let's see why.

How do we analyze an approximation algorithm?

Need to find an α so it's always true that $\alpha \cdot \mathrm{OPT} \leq \mathrm{ALG}$.

These proofs aren't always easy to write!

We usually don't "understand" OPT very well. If we did, we'd run an algorithm to find it!

Take 521 (or do undergrad research!) to learn more

We're going to see one example proof today.

Finding an approximation for Vertex Cover

Here's Idea 2

```
While(G still has edges)
     Choose any edge (u,v)
     Add u to VC, and v to VC
     Delete u v and any edges touching them
EndWhile
```

Why? At least one of u, v is in the vertex cover. We know we're not getting the exact optimal, so…don't try. At least one of the two was a good decision.

Does it work?

Do we find a vertex cover? (Is the solution valid?)

Is it close to the smallest one? (Is the solution good?)

But first, let's notice – this is a polynomial-time algorithm!

If we're going to take exponential time, we can get the exact answer. We want something fast if we're going to settle for a worse answer.

Do we find a vertex cover?

Observe that we only delete an edge after we have ensured it will be covered.

When we delete an edge, the edge is covered by whichever endpoint caused it to be deleted (because we added both vertices of the chosen edge to the vertex cover).

And we only stop the algorithm when every edge has been deleted. So every edge is covered (i.e. we really have a vertex cover).

How big is it?

Let OPT be a **minimum** vertex cover.

Key idea: "charge" or "assign" each vertex we add to ALG's solution to a vertex in OPT. If every vertex in OPT gets "assigned" at most k vertices of OPT then it must be that $k \cdot OPT \leq ALG$.

You've seen this proof technique before! Finding a bijection between two sets says they are the same size. That's $k = 1$ with the technique above.

What assignment do we pick?

Key is finding what in OPT we are going to charge to…

Claim: For every (u, v) from ALG,

Why? (u, v) was an edge! OPT covers (u, v) so at least one is in OPT.

Charge both (u, v) to whichever was in OPT.

We assign at most two vertices of ALG to every one in OPT, so we have a 2-approximation.

Greedy Approximation Algs

Greedy Algorithms are a very common source for approximation algorithms!

Since you're making an optimal "local" choice, it's not likely to be a terrible solution (even if it's rarely the absolute best one).

It's still simple to implement and fast!

And the proofs aren't nearly as hard anymore!!

Set Cover

To make a VC problem look like set cover:
U is the set of edges
Elements of $\mathcal{F}$ are all the edges touching a single vertex.

A generalization of vertex cover:

Let U be a universe,

And $\mathcal{F} \subseteq \mathcal{P}(U)$ be a family of subsets of U.

For example, $U = \{1,2,3,4,5,6\}$
$\mathcal{F} = \{\ \{1,2,3\}, \{3,4,5\}, \{5,6\}, \{2,4,6\}\ \}$.

$\mathcal{S} \subseteq \mathcal{F}$ is a cover if $\bigcup_{S \in \mathcal{S}} S = \mathcal{F}$

For example $\{\ \{1,2,3\}, \{5,6\}, \{2,4,6\}\}$ is a cover.

The size of a cover is the number of sets in $\mathcal{S}$.

The cover above is size 3.

Set Cover

Not clear how to adapt idea 2; we're using something specific to graphs (that every element of U appears in exactly two of the elements of $\mathcal{F}$)

Idea 1 still would work though:

Idea 1: Maximize [Elements of U] covered to [sets] taken" ratio

Take a [Set with maximum number of uncovered elements], add it to the [SC]

Delete all [Newly covered elements of U]

Run the greedy algorithm

Let $U = \{1,2,3,4,5,6,7,8\}$

$S = \{\ \{1,2,3\}, \{2,4\}, \{6,8\}, \{3,5,7\}, \{5,7,8\}, \{2,5,6\}, \{4\}\ \}$

Run the greedy algorithm

Let $U = \{1,2,3,4,5,6,7,8\}$

$S = \{ \{1,2,3\}, \{2,4\}, \{6,8\}, \{3,5,7\}, \{5,7,8\}, \{2,5,6\}, \{4\} \}$

Take $\{1,2,3\}$

Remaining to cover $\{4,5,6,7,8\}$

$S = \{ \{1,2,3\}, \{2,4\}, \{6,8\}, \{3,5,7\}, \{5,7,8\}, \{2,5,6\}, \{4\} \}$

Run the greedy algorithm

Let $U = \{1,2,3,4,5,6,7,8\}$

$S = \{ \{1,2,3\}, \{2,4\}, \{6,8\}, \{3,5,7\}, \{5,7,8\}, \{2,5,6\}, \{4\} \}$

Take $\{1,2,3\}$

Remaining to cover $\{4,5,6,7,8\}$

$S = \{ \{1,2,3\}, \{2,4\}, \{6,8\}, \{3,5,7\}, \{5,7,8\}, \{2,5,6\}, \{4\} \}$

Take $\{5,7,8\}$

Remaining to cover $\{4,6\}$

$S = \{ \{1,2,3\}, \{2,4\}, \{6,8\}, \{3,5,7\}, \{5,7,8\}, \{2,5,6\}, \{4\} \}$

Run the greedy algorithm

Let $U = \{1,2,3,4,5,6,7,8\}$

$S = \{ \{1,2,3\}, \{2,4\}, \{6,8\}, \{3,5,7\}, \{5,7,8\}, \{2,5,6\}, \{4\} \}$

Take $\{1,2,3\}$

Remaining to cover $\{4,5,6,7,8\}$

$S = \{ \{1,2,3\}, \{2,4\}, \{6,8\}, \{3,5,7\}, \{5,7,8\}, \{2,5,6\}, \{4\} \}$

Take $\{5,7,8\}$

Remaining to cover $\{4,6\}$

$S = \{ \{1,2,3\}, \{2,4\}, \{6,8\}, \{3,5,7\}, \{5,7,8\}, \{2,5,6\}, \{4\} \}$

Take $\{2,4\}$ then $\{6,8\}$.

Approximation Ratio?

Let k be the size of OPT.

Claim: If there are m elements remaining, then some set covers at least m/k of them.

That applies at every step! So after i selections, the number of elements remaining to be covered is at most

$$n\left(1-\frac{1}{k}\right)^{i} = n\left[\left(1-\frac{1}{k}\right)^{k}\right]^{i/k} \leq ne^{-i/k}$$

Approximation Ratio?

After i steps, there are $ne^{-i/k}$ elements remaining. There will be 1 left when

$ne^{-i/k} = 1$

$n = e^{i/k}$

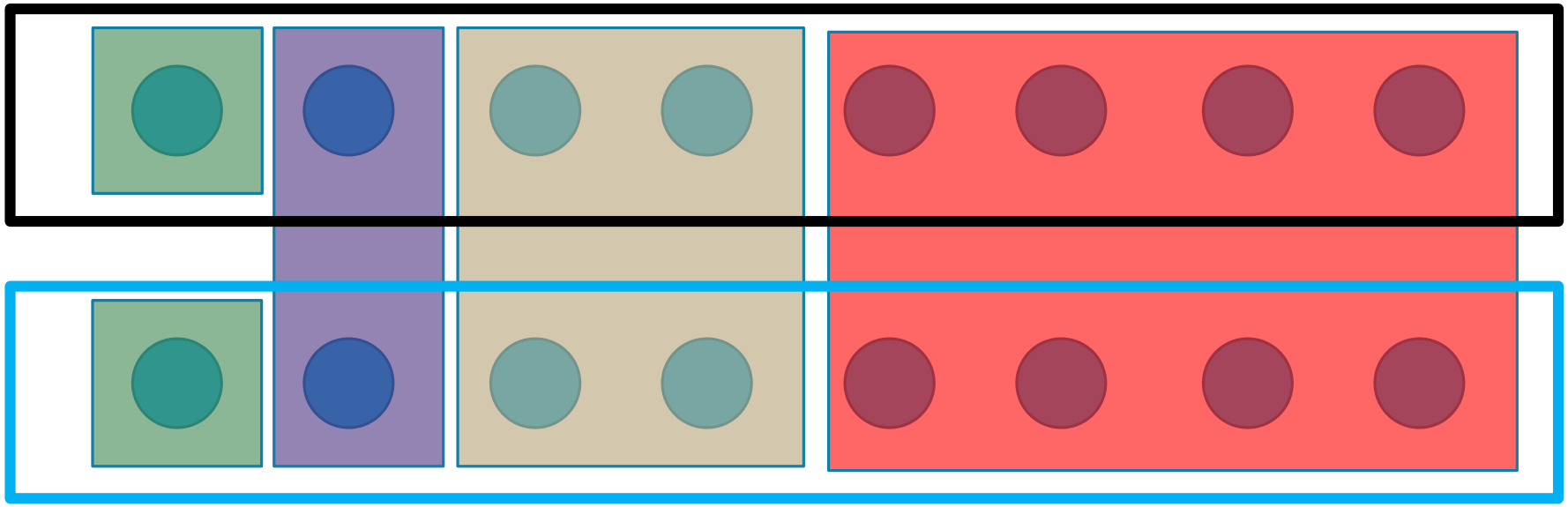
$\ln(n) = i/k$

$i = k \cdot \ln(n)$.

So the approximation ratio is about $\ln(n)$

(ignoring some off-by-one errors to get here).

A Bad Example



Greedy might take: Red, gold, purple, green, green (boxes spanning both rows)

OPT will take each row.

Example is 5 to 2.

Doubling the instance adds 1 to ALG, 0 to OPT
That gives us a $\log(n)$ ratio.