Greedy Approximation Algorithms

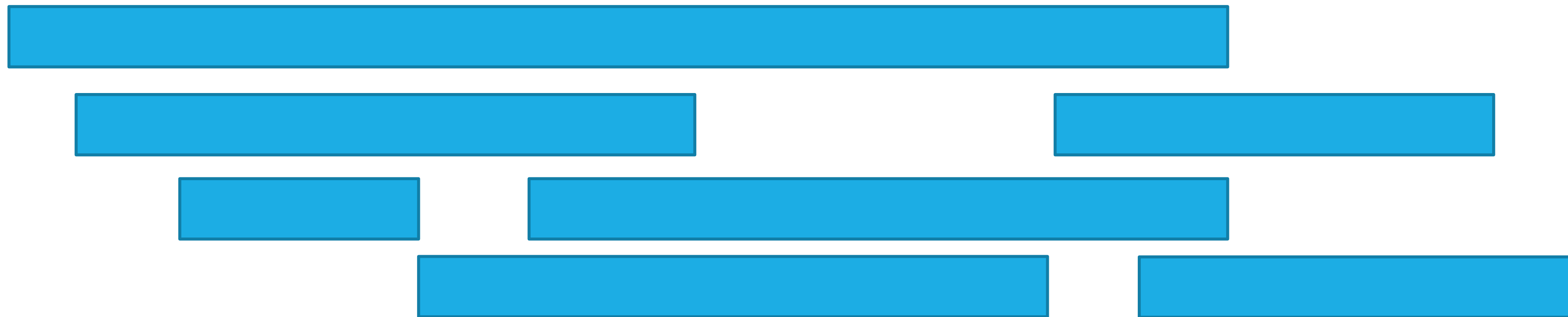
CSE 421 Spring 2026
Lecture 8

Interval Scheduling

You have a single processor, and a set of jobs with fixed start and end times.

Your goal is to maximize the number of jobs you can process.

I.e. choose the maximum number of non-overlapping intervals.



OPT is 3 – there is no way to have 4 non-overlapping intervals;
both the red and purple solutions are equally good.

Greedy Algorithm Ideas (let's narrow down)

Earliest end time

Latest end time

Earliest start time

Latest start time

Shortest interval

Fewest overlaps (with remaining intervals)

Take Earliest Start Time

Take Earliest Start Time – Counter Example

Algorithm finds

Optimum

Taking the one with the earliest start time doesn't give us the best answer.

Greedy Algorithm (narrowing down, 1)

Earliest end time

Latest end time ✖

Earliest start time ✖

Latest start time

Shortest interval

Fewest overlaps (with remaining intervals)

Shortest Interval

Shortest Interval (counter-example)

Taking the shortest interval first doesn't give us the best answer

Greedy Algorithm (narrowing down, 2)

Earliest end time

Latest end time ✖

Earliest start time ✖

Latest start time

Shortest interval ✖

Fewest overlaps (with remaining intervals)

Earliest End Time

Intuition: If u has the earliest end time, and u overlaps with v and w then v and w also overlap.

Why?

Earliest End Time (intuition)

Intuition: If u has the earliest end time, and u overlaps with v and w then v and w also overlap.

Why?

If u and v overlap, then both are "active" at the instant before u ends (otherwise v would have an earlier end time).

Otherwise v would have an earlier end time than u! By the same reasoning, w is also "active" the instant before u ends. So v and w also overlap with each other.

Earliest End Time (strategy)

Can you prove it correct?

Do you want to use

Structural Result

Exchange Argument

Greedy Stays Ahead

Exchange Argument (1)

Let $A = a_1, a_2, \ldots, a_k$ be the set of intervals selected by the greedy algorithm, ordered by endtime

OPT= $o_1, o_2, \ldots, o_\ell$ be the maximum set of intervals, ordered by endtime.

Our goal will be to "exchange" to show A has at least as many elements as OPT.

Let a_i, o_i be the first two elements where a_i and o_i aren't the same. Since a_{i-1} and o_{i-1} are the same, neither a_i nor o_i overlaps with any of $o_1, \ldots, o_{i-1}$. And by the greedy choice, a_i ends no later than o_i so a_i doesn't overlap with o_{i+1}. So we can exchange a_i into OPT, replacing o_i and still have OPT be valid.

Exchange Argument (2)

Repeat this argument until we have changed OPT into A.

Can OPT have more elements than A?

No! After repeating the argument, we could change every element of OPT to A. If OPT had another element, it wouldn't overlap with anything in OPT, and therefore can't overlap with anything in A after all the swaps. But then the greedy algorithm would have added it to A.

So A has the same number of elements as OPT does, and we really found an optimal

Greedy Stays Ahead (1)

Let $A = a_1, a_2, \ldots, a_k$ be the set of intervals selected by the greedy algorithm, ordered by endtime

OPT= $o_1, o_2, \ldots, o_\ell$ be the maximum set of intervals, ordered by endtime.

Our goal will be to show that for every i, a_i ends no later than o_i.

Proof by induction:

Base case: a_1 has the earliest end time of any interval (since there are no other intervals in the set when we consider a_1 we always include it), thus a_1 ends no later then o_1.

Greedy Stays Ahead (2)

Inductive Hypothesis: Suppose for all $i \leq k$, a_i ends no later than o_i.

IS: Since (by IH) a_k ends no later than o_k, greedy has access to everything that doesn't overlap with a_k. Since a_k ends no later than o_k, that includes o_{k+1}. Since we take the first one that doesn't overlap, a_{k+1} will also end before (or equal to) o_{k+1}.

Therefore a_{k+1} ends no later than o_{k+1}

Wrapping Up: Since every a_i ends no later than o_i, the last interval greedy selects (a_n) is no later than o_n. There cannot be an o_{n+1}, as if it didn't overlap with o_n it also wouldn't overlap with a_n and would have been added by greedy.

Greedy Algorithm (narrowing down, 3)

Earliest end time ✔

Latest end time ✖

Earliest start time ✖

Latest start time

Shortest interval ✖

Fewest overlaps (with remaining intervals)

Other Greedy Algorithms

It turns out latest start time also works.

Latest start time is actually the same as earliest end time (imagine "reflecting" all the jobs along the time axis – the one with the earliest end time ends up having the last start time).

What about fewest overlaps?

Doesn't work. ☹ Counter-examples are a little more complicated than the others.

Greedy Algorithm (all the ideas)

Earliest end time ✓

Latest end time ✗

Earliest start time ✗

Latest start time ✓

Shortest interval ✗

Fewest overlaps (with remaining intervals) ✗

Summary

Greedy algorithms

You'll probably have 2 (or 3…or 6) ideas for greedy algorithms. Check some simple examples before you implement!
Greedy algorithms rarely work.

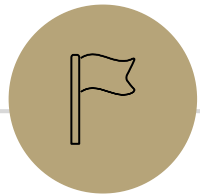
When they work AND you can prove they work, they're great!

Proofs are often tricky

Structural results are the hardest to come up with, but the most versatile.

Greedy stays ahead usually use induction

Exchange start with the **first** difference between greedy and optimal.



Greedy Approximation Algorithms

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 Requirements

What makes an approximation algorithm good?

<u>Speed:</u> We usually require approximation algorithms to run in polynomial time.

<u>Accuracy:</u>

NP-completeness will say that we can't solve the problem <u>exactly</u> 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.

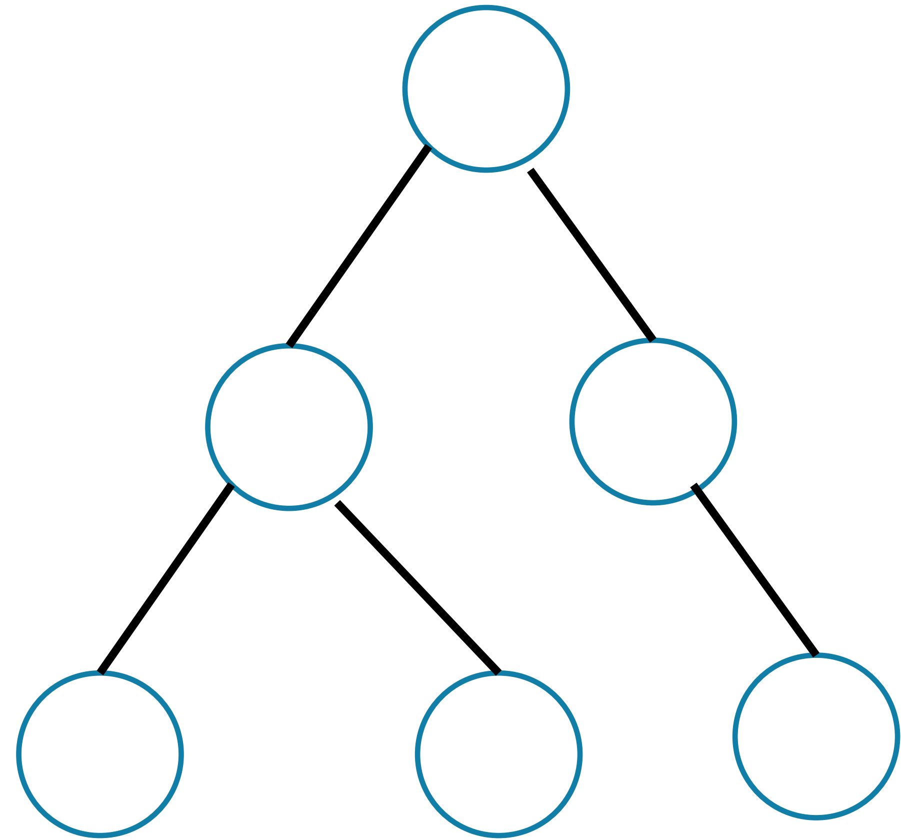
Vertex Cover

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 (ex1)

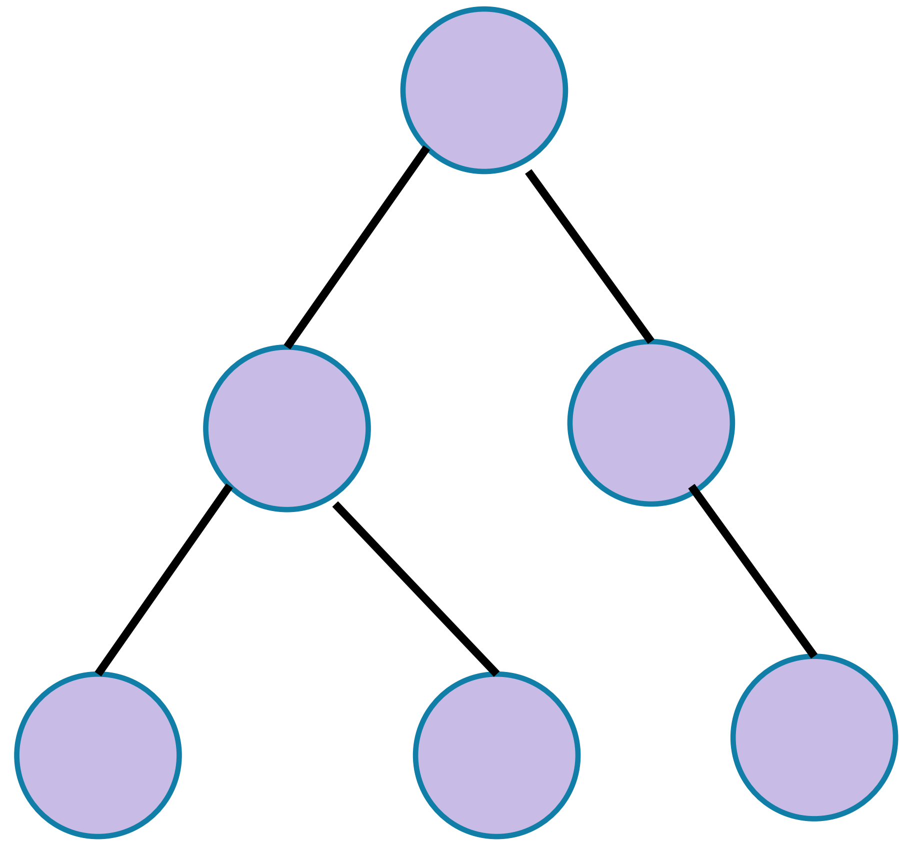
Vertex Cover

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 (ex2)

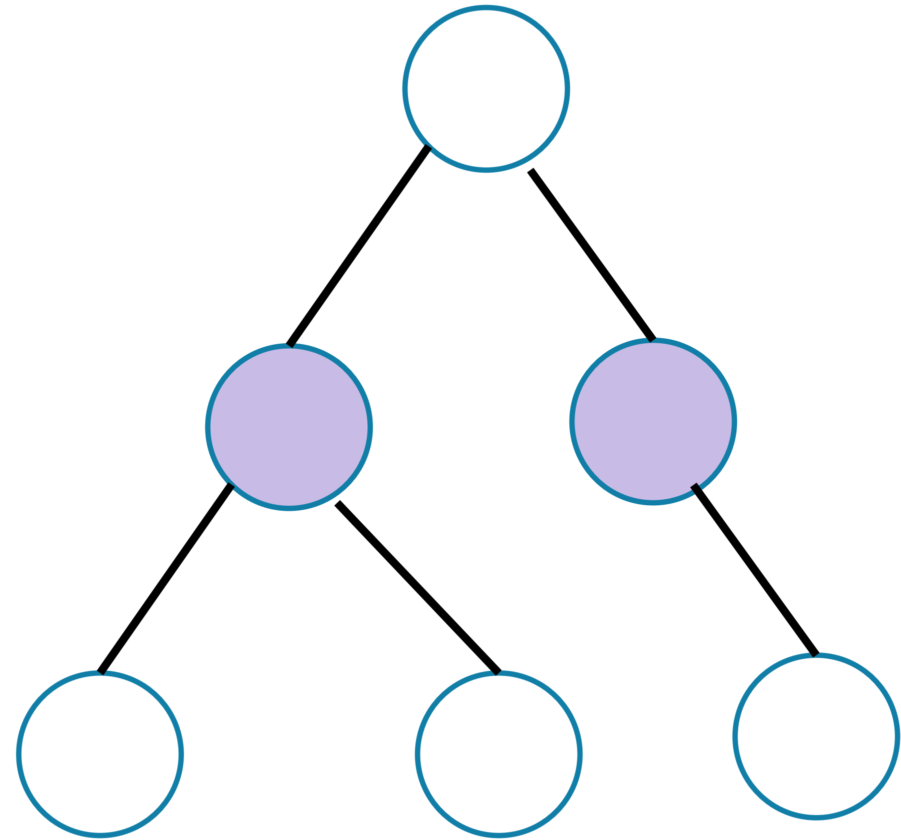
Vertex Cover

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)



Brainstorm

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

Vertex Cover Ideas

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. Put both in your VC. Delete all incident edges.

Non-optimal

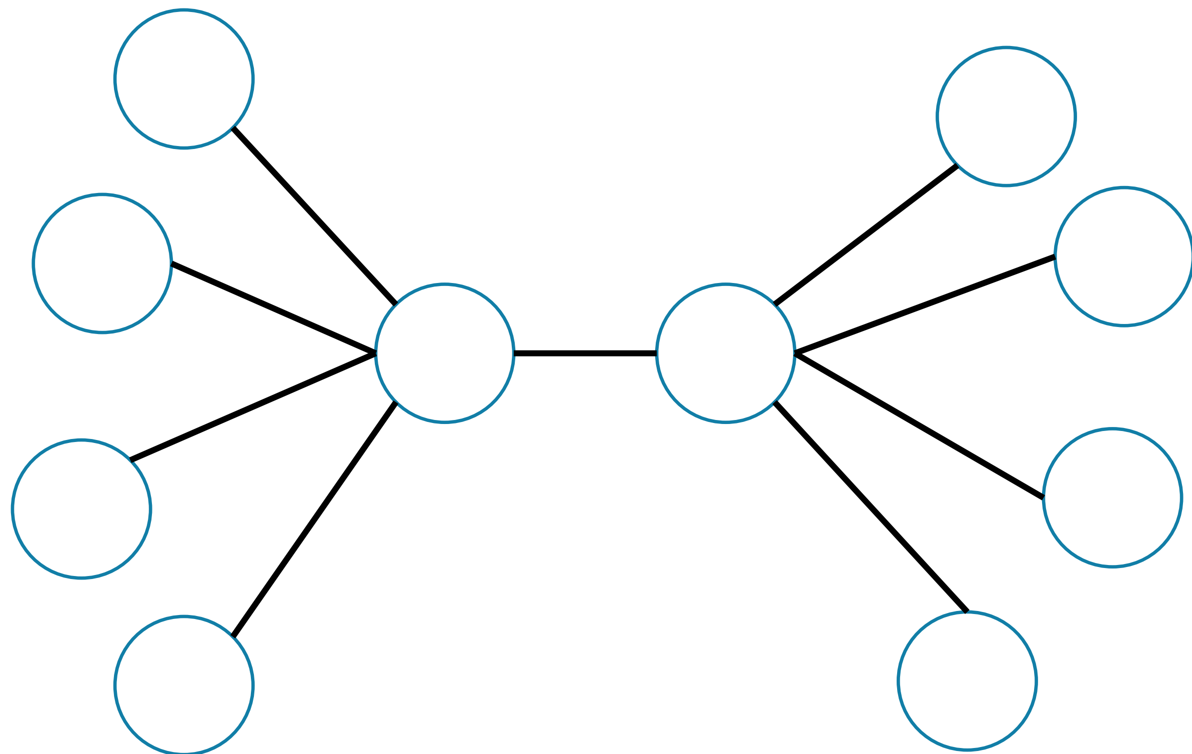
Showing idea 1 doesn't work is a good exercise!

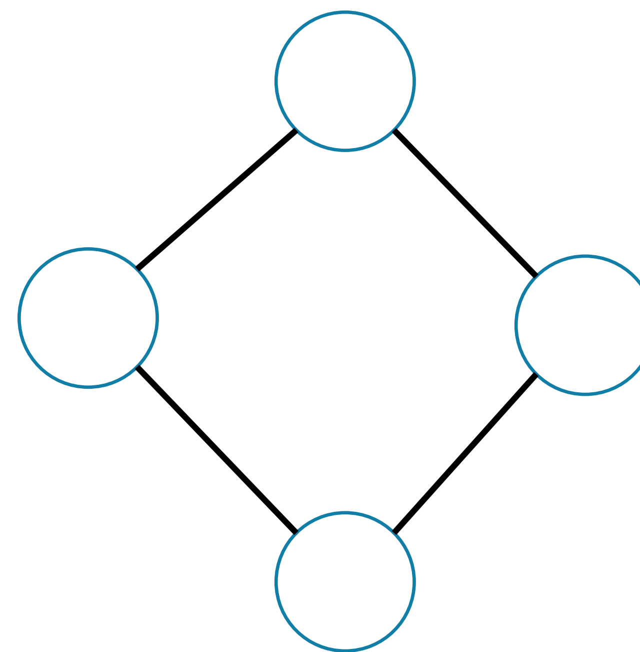
Focus on idea 2, come up with a graph where it could give you the optimal VC, and another where it doesn't.

<u>Idea 2</u>: At least one good one

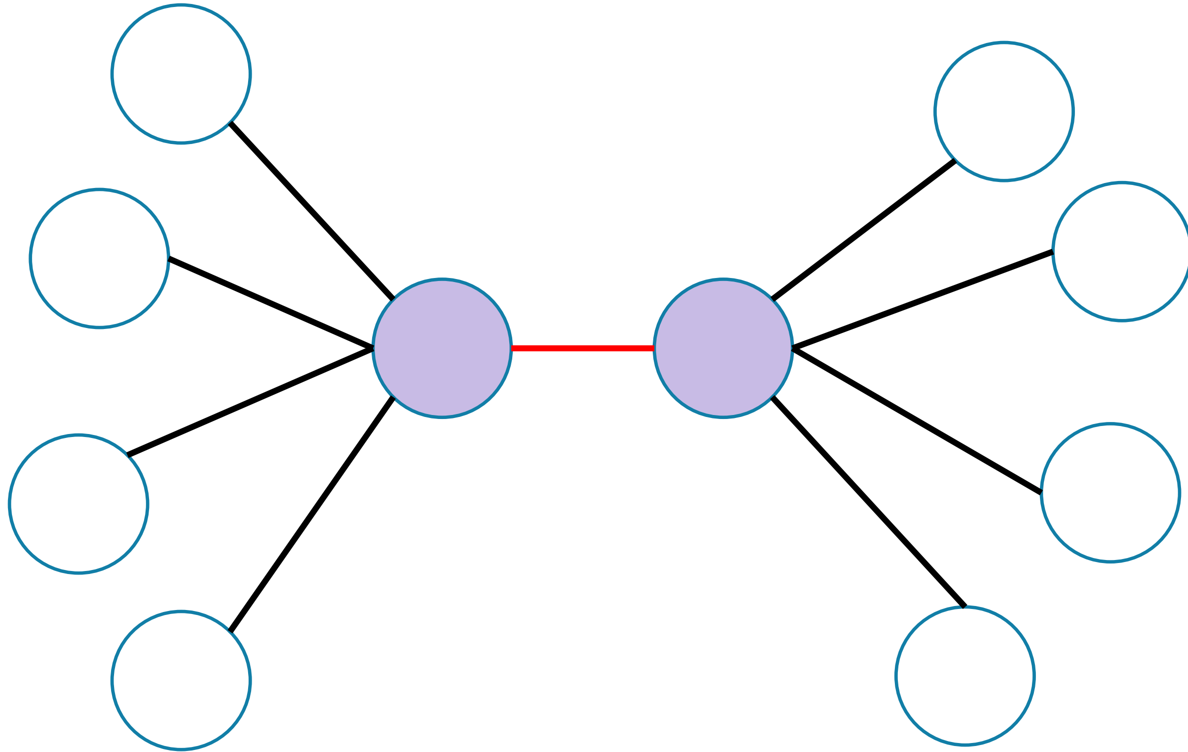
Choose an (arbitrary) edge (u, v). At least one of u, v is in the minimum VC. Put both in your VC. Delete all incident edges.

Two Examples

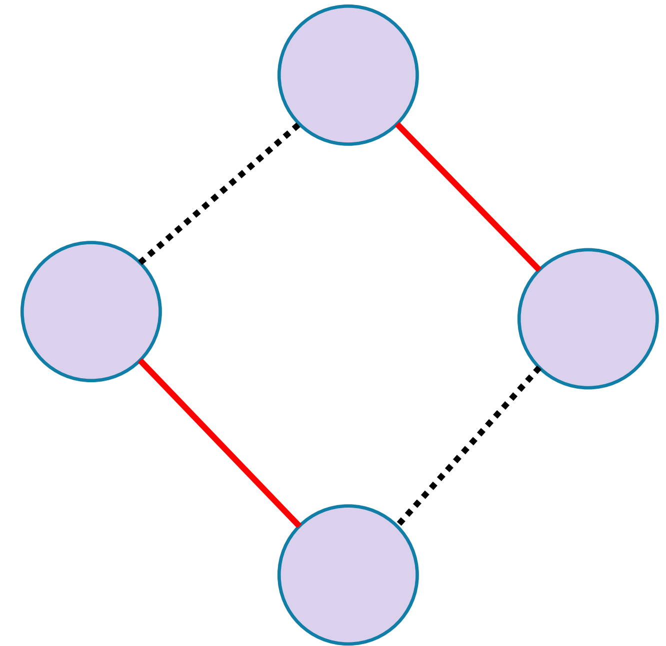




Two Examples (alg results)



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 for VC

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

So is the approximation ratio for the algorithm 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 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, at least one was in OPT

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.

When the algorithm adds (u, v), it deletes u, v and all incident edges, so u,v will never be charged again (as we only charge along an edge in the remaining graph. We assign at most two vertices of ALG to every one in OPT, so we have a 2-approximation.

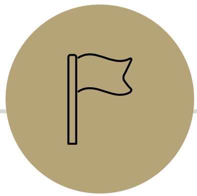
Greedy Approximation Algs

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!!



Optional Content

A similar problem: Set cover

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 Algorithm Idea

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:

<u>Idea 1</u>: 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 (1)

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 (2)

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 (3)

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? (1)

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? (2)

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

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

$$n = e^{i/k}$$

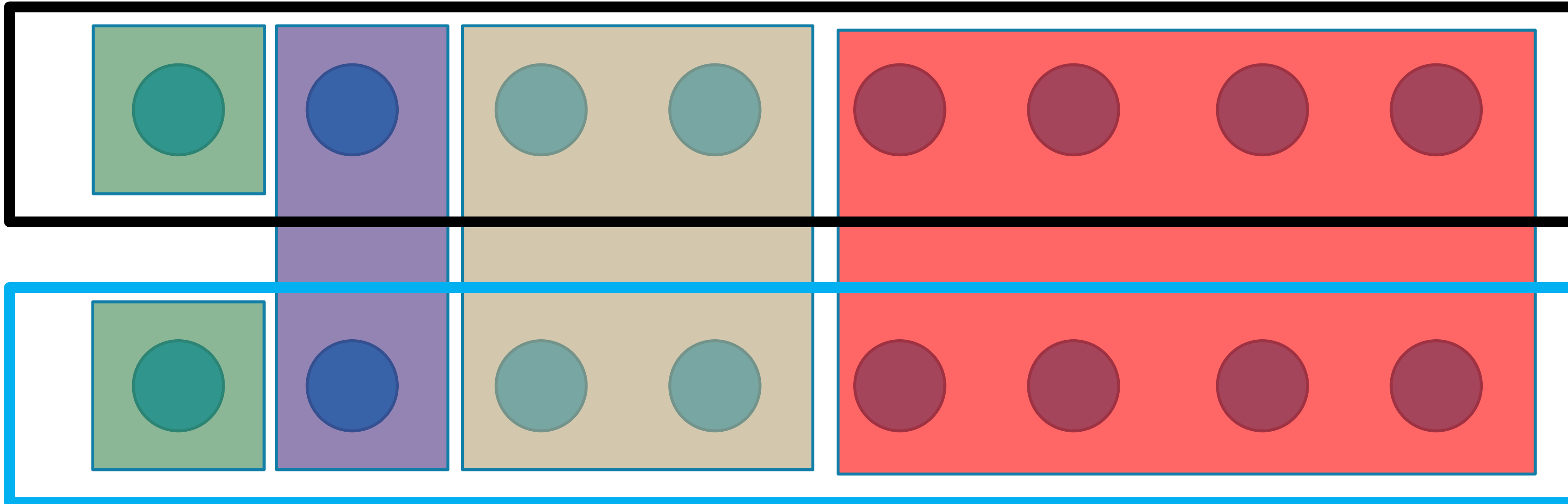
$$\ln(n) = i/k$$

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

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.