CSE 417 Autumn 2025

Lecture 20: Non-optimal greedy

Glenn Sun

Some interesting greedy ideas

One example from before: A* search

Pick the next point with smallest

distance from source + estimated distance to end

(computed like Dijkstra)

(via distance formula with coordinates, e.g.)

A* search *is* optimal for shortest paths, if you only ever underestimate distance.

Second example: University timetabling

Students submit to a university the courses they would like to take during the next quarter.

The university has k class blocks, and needs to determine if scheduling is possible with no students in two classes at once.

Graph representation:

- Vertices: Courses
- Edges: Courses that cannot be scheduled at the same time

Graph coloring

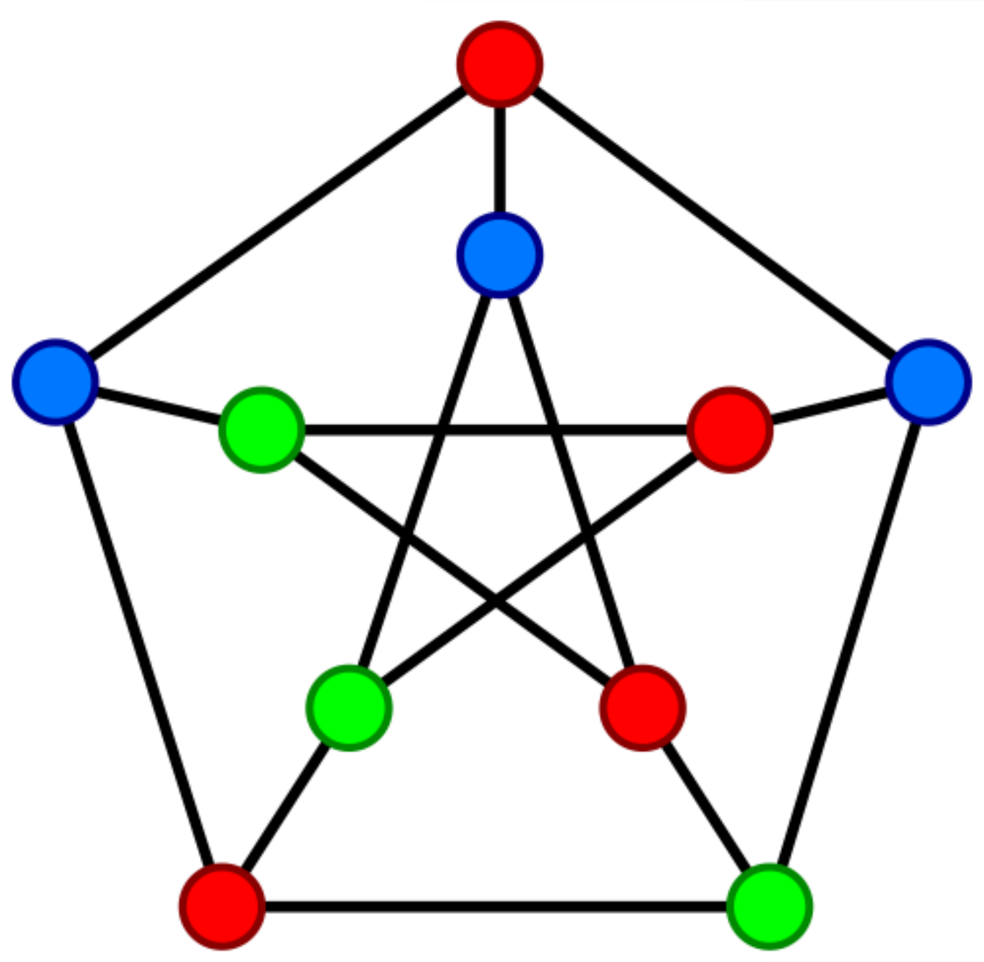
This is an instance of **graph coloring**. A valid graph coloring

- Assigns a color (timeslot) to each vertex (course)
- Such that no two adjacent vertices have the same color

Input: An undirected graph

Goal: What is the minimum number of colors needed?

Graph coloring



Why try a greedy algorithm?

Graph coloring is **hard**.

Mathematicians believe that it is impossible to solve graph coloring optimally in polynomial time: $O(n^c)$ for any c.

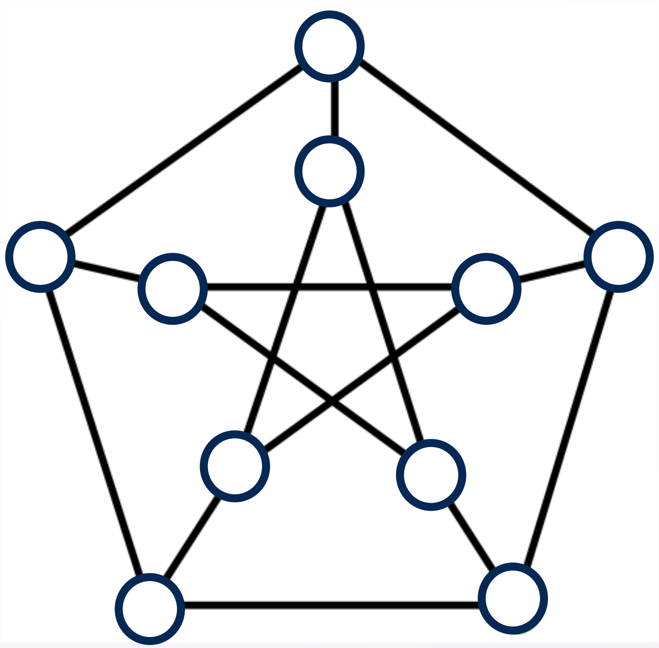
More about this later in course: graph coloring is **NP-complete**.

Easy optimal exponential time algorithm:

- For every k, check every way to color n vertices with k colors.
- But there are k^n possible colorings each iteration!

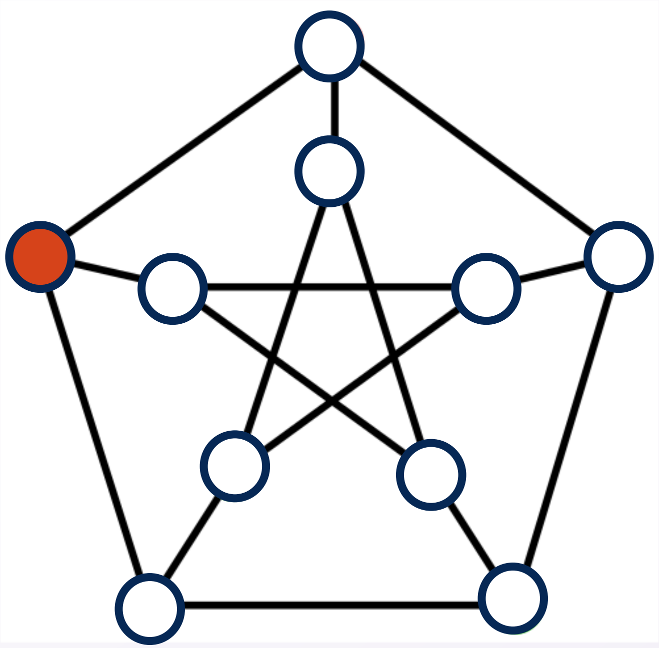
Greedy approach to graph coloring

1. Think of colors as numbers 1, 2, 3, …
2. **while** some vertex is not yet colored,
3. Color it with the smallest number that is not used by its neighbors.



Greedy approach to graph coloring

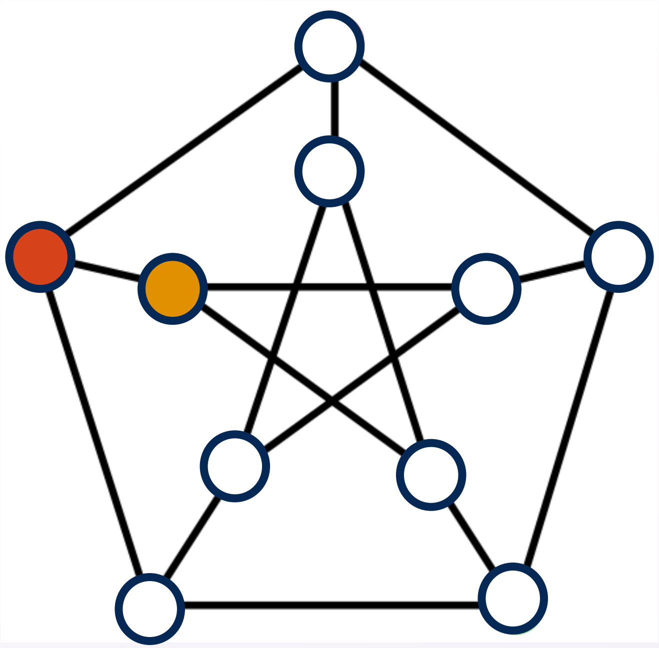
1. Think of colors as numbers 1, 2, 3, …
2. **while** some vertex is not yet colored,
3. Color it with the smallest number that is not used by its neighbors.



Greedy approach to graph coloring

1. Think of colors as numbers 1, 2, 3, …
2. **while** some vertex is not yet colored,
3. Color it with the smallest number that is not used by its neighbors.

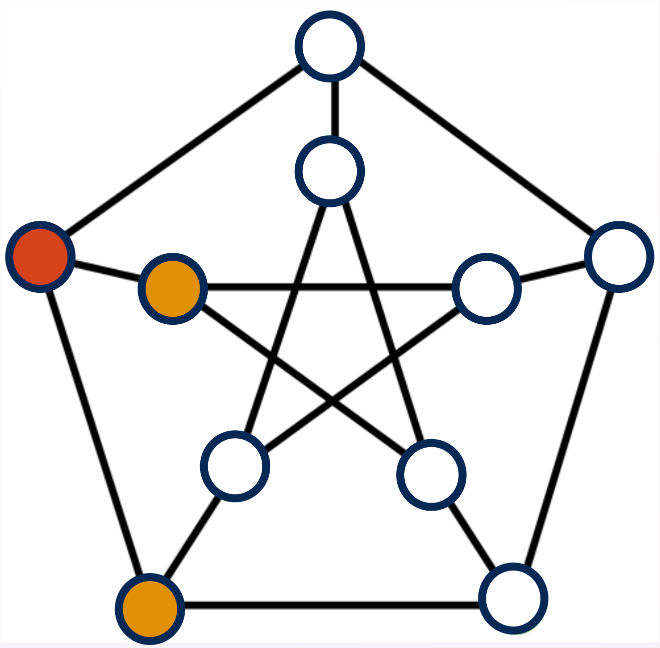
1 2 3



Greedy approach to graph coloring

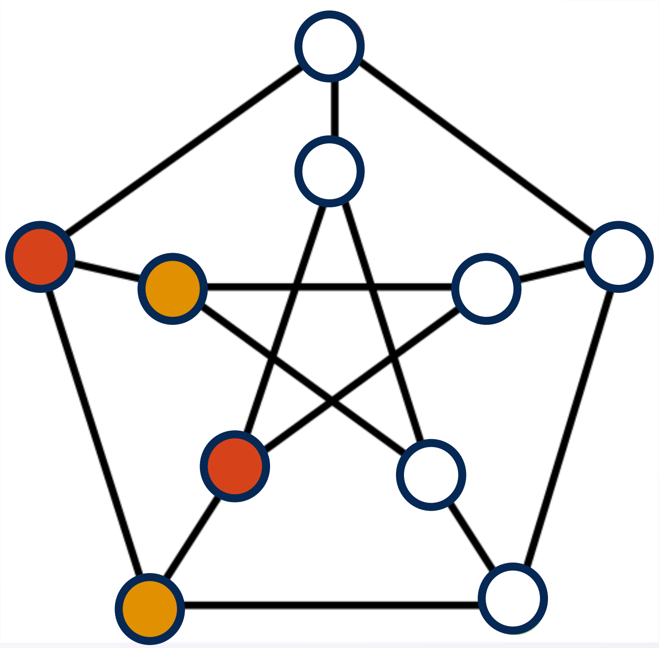
1. Think of colors as numbers 1, 2, 3, …
2. **while** some vertex is not yet colored,
3. Color it with the smallest number that is not used by its neighbors.

1 2 3



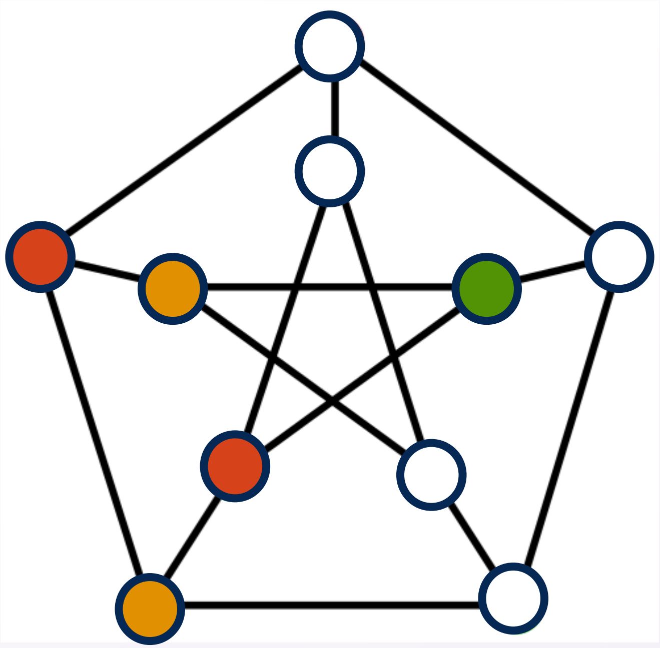
Greedy approach to graph coloring

1. Think of colors as numbers 1, 2, 3, …
2. **while** some vertex is not yet colored,
3. Color it with the smallest number that is not used by its neighbors.



Greedy approach to graph coloring

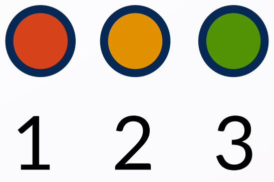
1. Think of colors as numbers 1, 2, 3, …
2. **while** some vertex is not yet colored,
3. Color it with the smallest number that is not used by its neighbors.

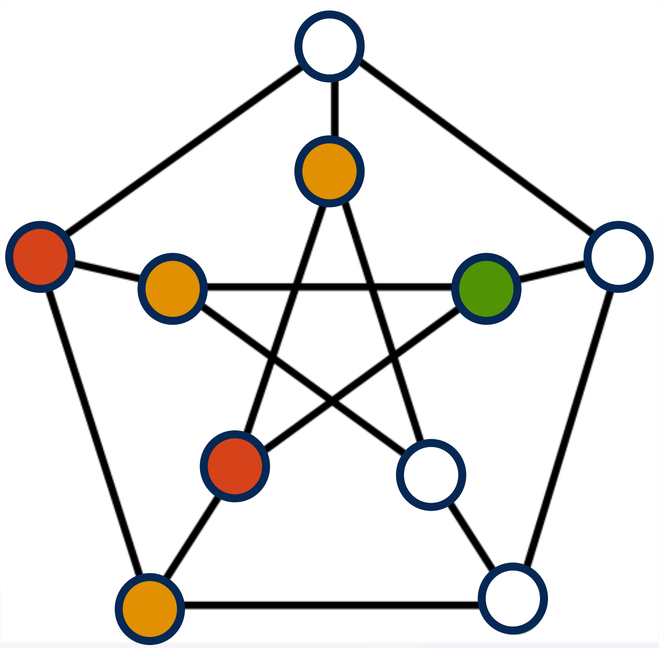


1 2 3

Greedy approach to graph coloring

1. Think of colors as numbers 1, 2, 3, …
2. **while** some vertex is not yet colored,
3. Color it with the smallest number that is not used by its neighbors.

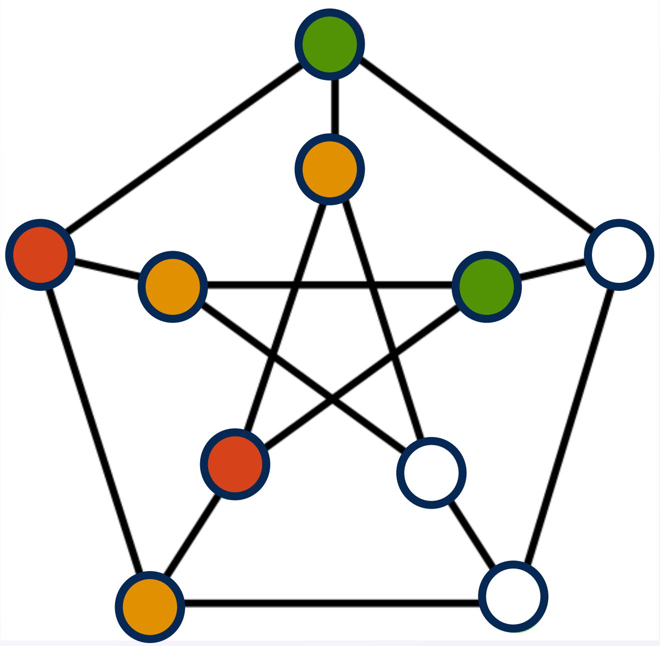




Greedy approach to graph coloring

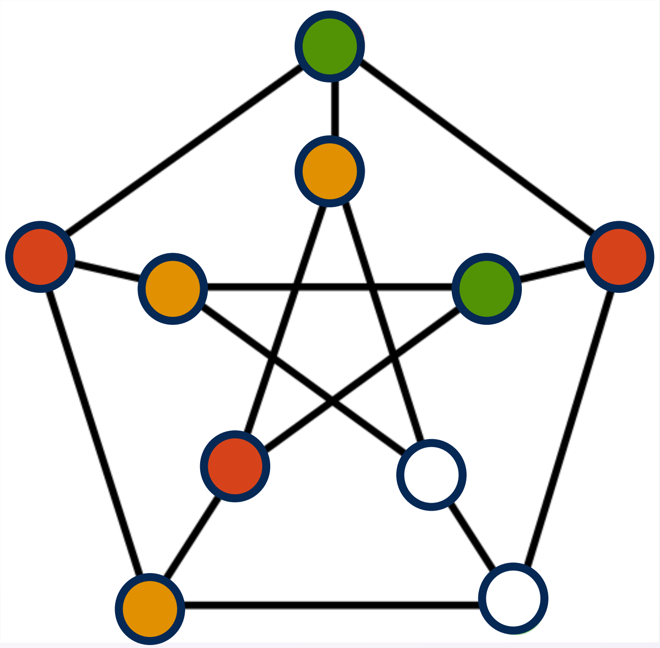
1. Think of colors as numbers 1, 2, 3, …
2. **while** some vertex is not yet colored,
3. Color it with the smallest number that is not used by its neighbors.

1 2 3



Greedy approach to graph coloring

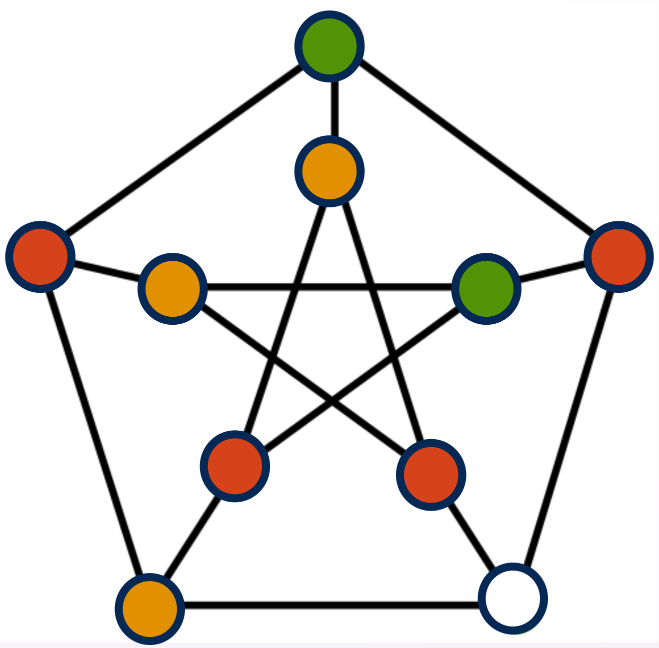
1. Think of colors as numbers 1, 2, 3, …
2. **while** some vertex is not yet colored,
3. Color it with the smallest number that is not used by its neighbors.



Greedy approach to graph coloring

1. Think of colors as numbers 1, 2, 3, …
2. **while** some vertex is not yet colored,
3. Color it with the smallest number that is not used by its neighbors.

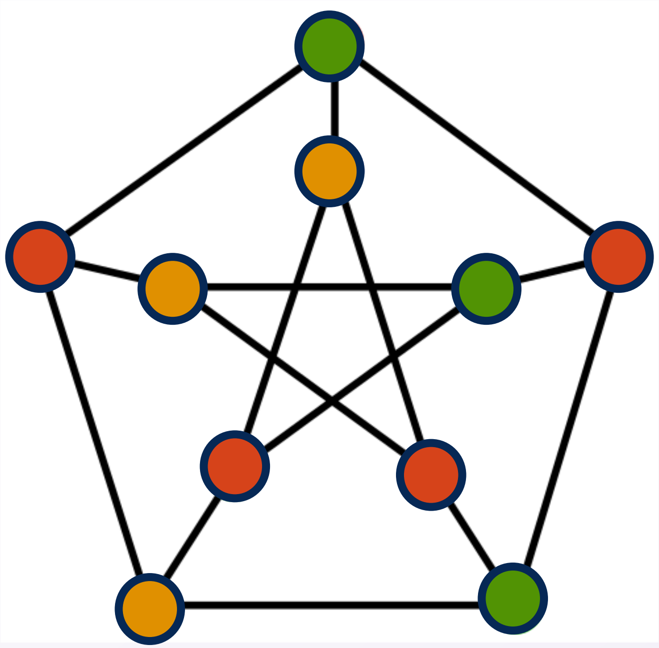
1 2 3



Greedy approach to graph coloring

1. Think of colors as numbers 1, 2, 3, …
2. **while** some vertex is not yet colored,
3. Color it with the smallest number that is not used by its neighbors.

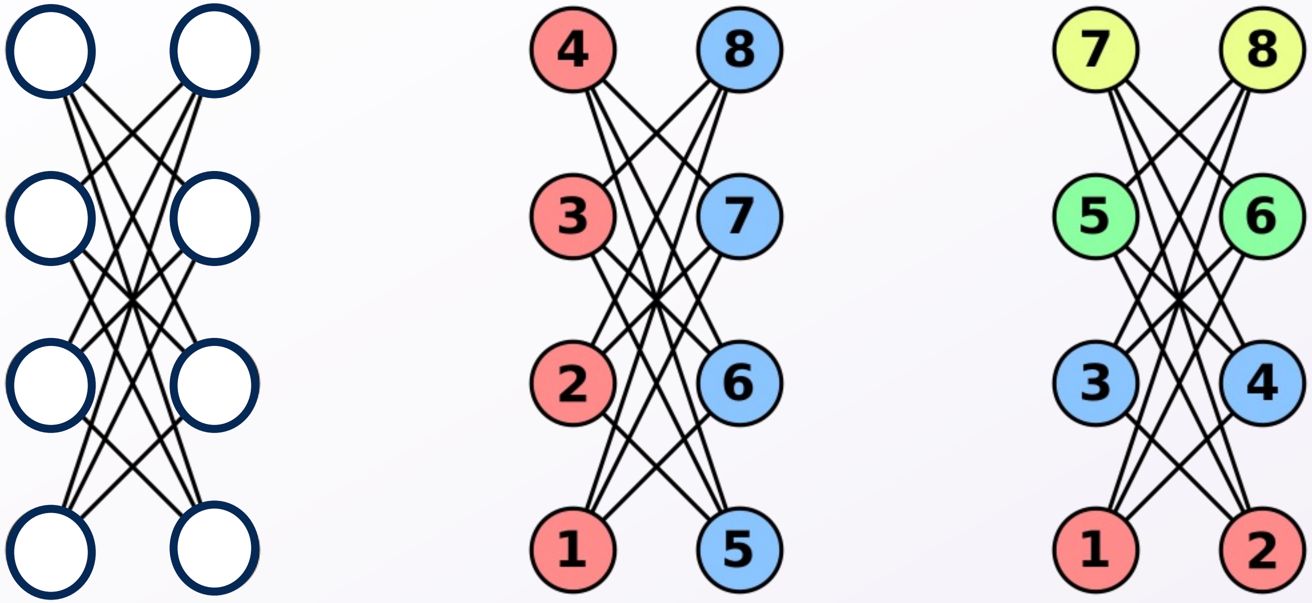
1 2 3



How bad can greedy be?

A **family** of counterexamples: crown graphs

- Take two set of vertices $x_1, \dots, x_n$ and $y_1, \dots, y_n$
- Connect every pair (x_i, y_j) except when $i = j$



Greedy: $n/2$ colors

Optimal: 2 colors

Gap: $n/4 = \Omega(n)$

How good is greedy?

The **degree** of a vertex is the number of neighbors it has.

Observation: The greedy algorithm uses at most 1 more color than the maximum degree of the graph!

- Because we pick the smallest number different from all neighbors.

Approximation algorithms

What is an approximation algorithm?

Suppose you are trying to solve a problem that asks you to **maximize** some quantity. On any particular input,

- Let $\boldsymbol{a}$ be the value obtained by your algorithm.
- Let $\boldsymbol{o}$ be the optimal value.

The **approximation ratio** for this particular input is $\boldsymbol{a/o}$.

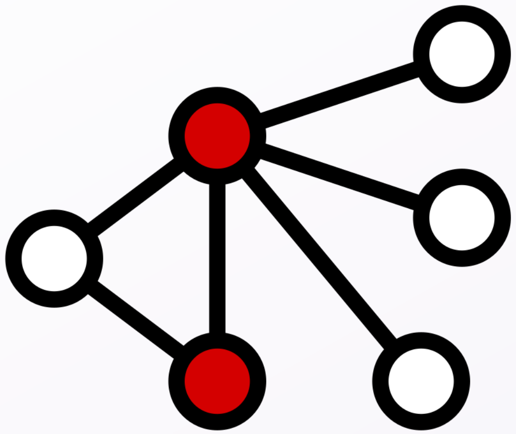
The approximation ratio for your overall algorithm is the largest $\boldsymbol{a/o}$ for any possible input.

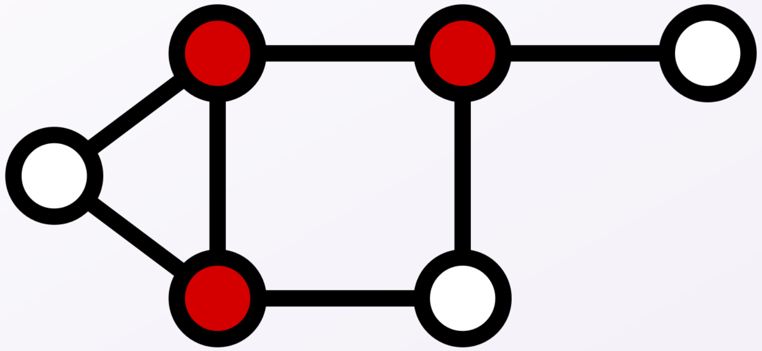
Use $\boldsymbol{o/a}$ for minimization problems.

Vertex cover

Input: An undirected graph

Goal: Select a smallest set of vertices so that every edge is covered

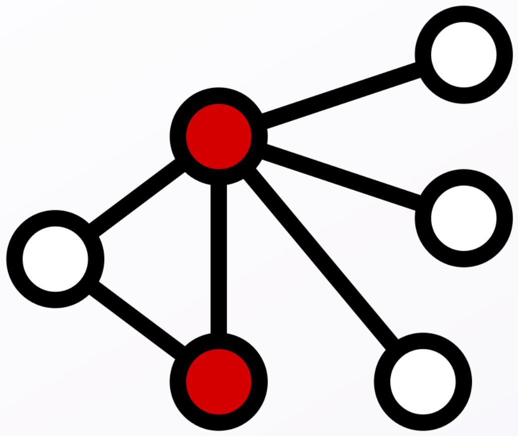




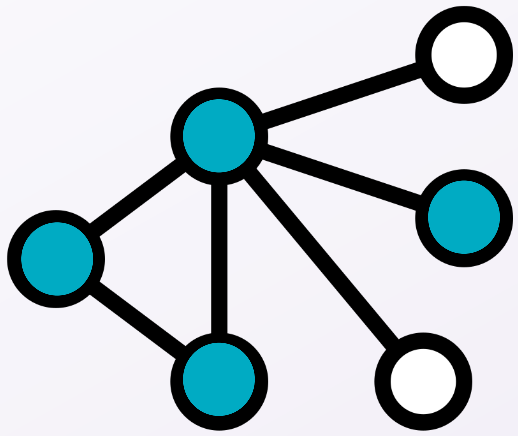
A simple greedy approach

1. **while** we don't have a vertex cover (some edge is uncovered),
2. Pick any uncovered edge and select ***both*** of its endpoints.

Optimal



Greedy



Calculating the approximation ratio

Claim. The greedy algorithm achieves an approximation ratio of 2.

- Because we picked uncovered edges, the chosen edges don’t touch each other.
- Any vertex cover must cover all the chosen edges.
- Because they don’t touch, every vertex can only cover one.
- Thus, any vertex cover must use at least half of what we used!

Load balancing

Input: There are m computers and n jobs, taking time $t_1, \dots, t_n$.

The **makespan** is the time it takes to finish on all computers.

Goal: Distribute the jobs to minimize makespan.

1:	$t_1 = 2$	$t_4 = 6$
2:	$t_2 = 3$	$t_5 = 2$
3:	$t_3 = 4$	$t_6 = 3$

A greedy approach

1. **for each** job in the order we received it,
2. Use the first available machine.

Greedy:

1:	$t_1 = 2$	$t_4 = 6$
2:	$t_2 = 3$	$t_5 = 2$
3:	$t_3 = 4$	$t_6 = 3$

Optimal:

1:	$t_4 = 6$		
2:	$t_2 = 3$	$t_5 = 2$	$t_1 = 2$
3:	$t_3 = 4$	$t_6 = 3$	

Calculating the approximation ratio

Take some time to

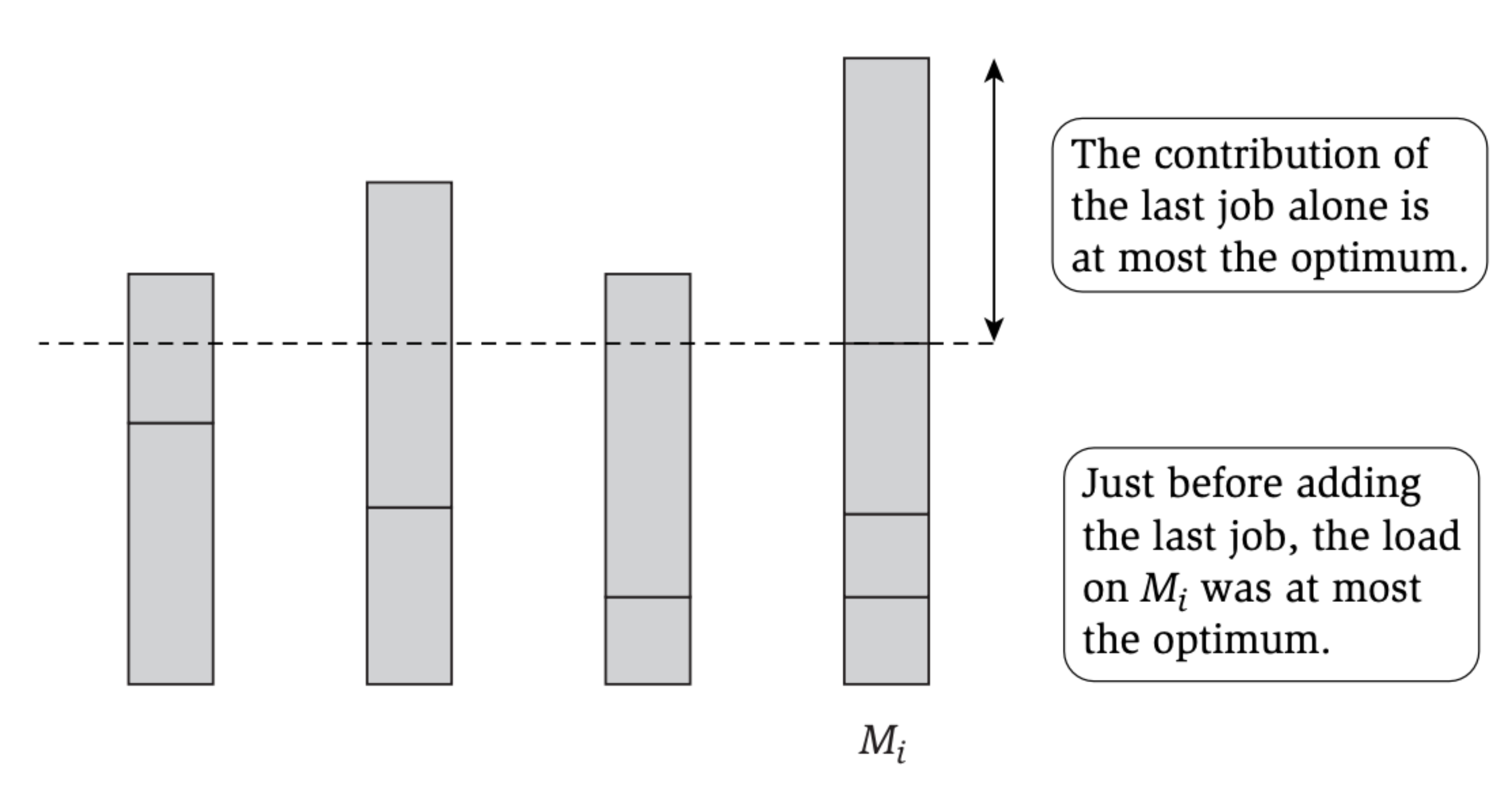
- Come up with some more examples where greedy is suboptimal, and try to make them as bad as possible.
- Think about how you might come up with an approximation ratio for this algorithm

A few observations

- If the input is a bunch of tasks of the same size, we are pretty close to optimal.
  - The optimal makespan is $\geq \frac{1}{m}(t_1 + \cdots + t_n)$.
- If the input has just one big task, we are actually optimal!
  - The optimal makespan is at least the maximum t_i.

Calculating the approximation ratio

Claim. The greedy algorithm achieves an approximation ratio of 2.



Final reminders

HW4 (Graphs) resubmissions close tonight @ 11:59pm!

I have OH now-12:30pm:

- Meet at front of classroom, we'll walk over together
- CSE (Allen) 214 if you're coming later

Nathan has online OH 12–1pm:

- https://washington.zoom.us/my/nathanbrunelle