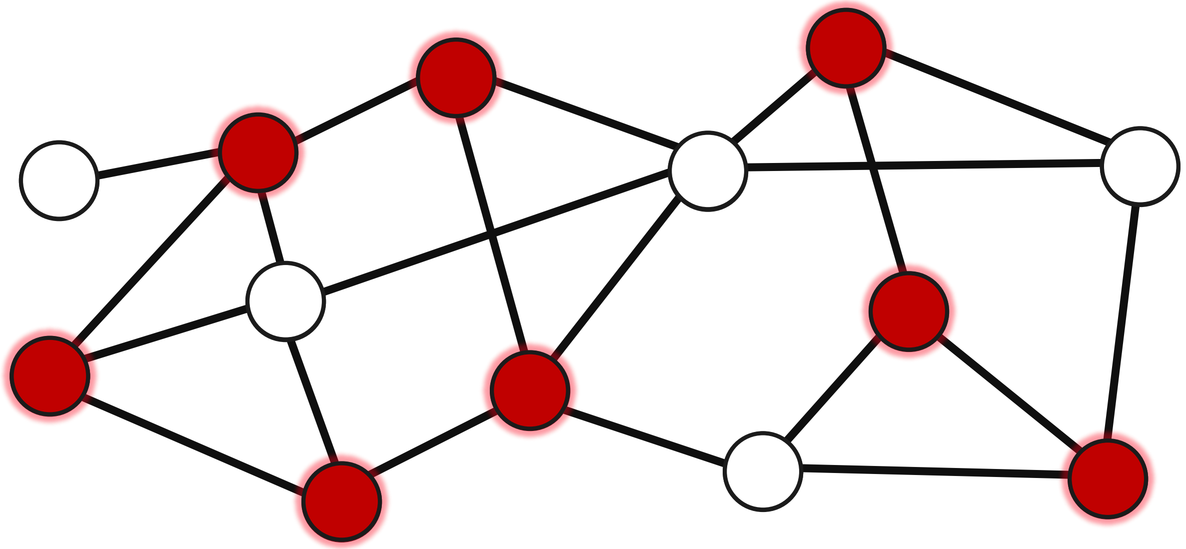
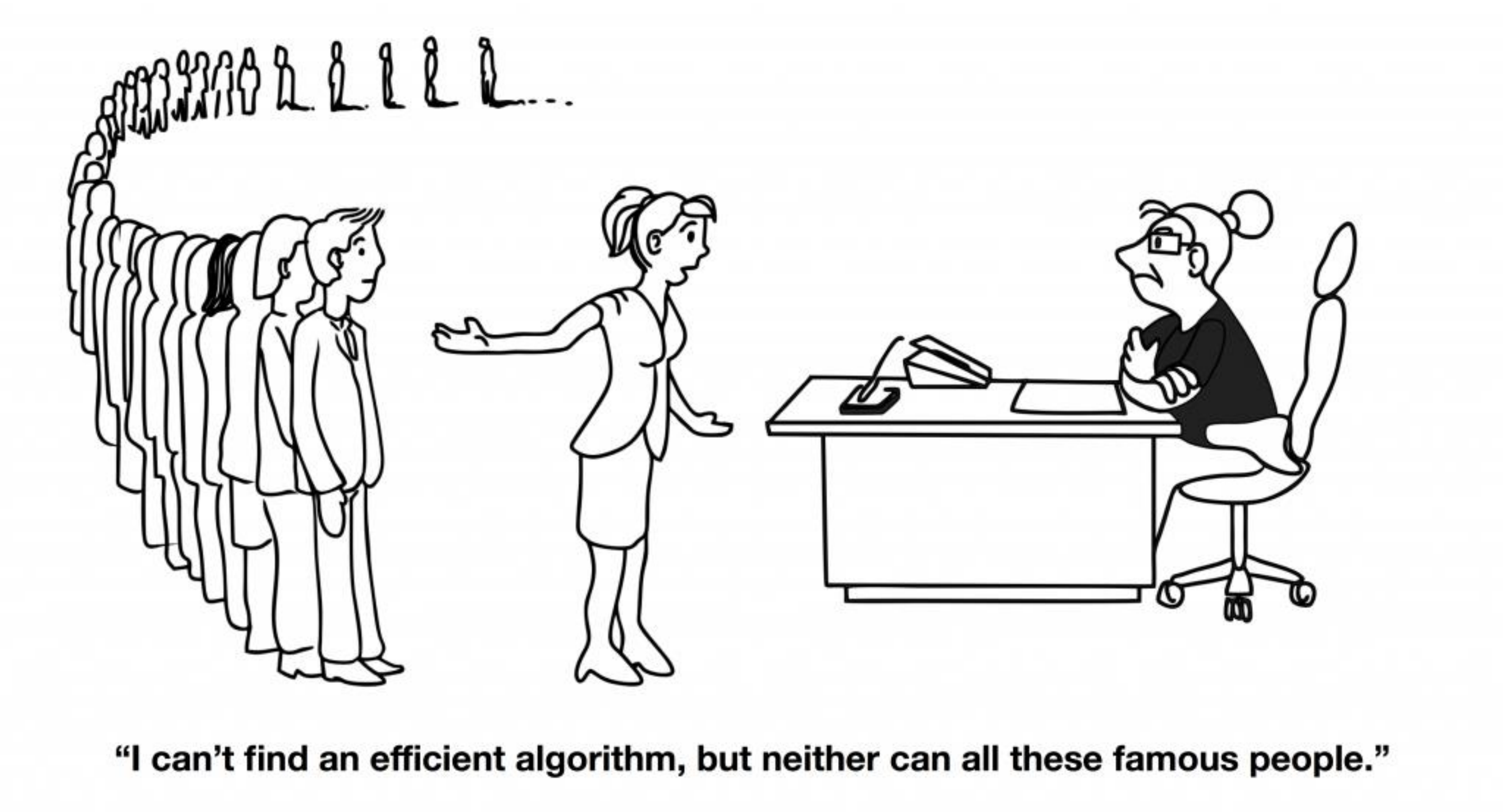


Approximation Algorithms

Yannic Maus

NP-hardness



What do you do? Quit your job & go home? Change problem?

figure by Stefan Szeider, TU Vienna

NP-hard, what can you do? Approximation algorithms!

- **Option 1:** Code a “slow” approach, e.g., knapsack in $O(n \cdot sumOfWeights)$,
  Sometimes this is fast enough
- **Option 2:** Just code some approach without theoretical guarantees
  This is pretty bad if we don’t have any guarantee for our solution, isn’t it?
- **Option 3:** Code simple approach, e.g., greedy, and prove theoretical guarantees

Approximation algorithms (this lecture)

- *Often run in polynomial time.*
- *They don’t give you optimal solutions on all instances, but gives you solutions “close to optimal”.*
- *Provable guarantees on the quality of your solution*
- *Often simple greedy algorithm*

Approximation Algorithm

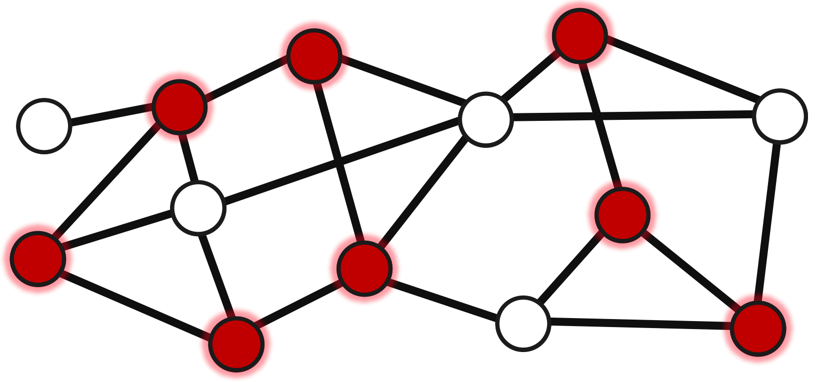
Approximation ratio $\boldsymbol{\rho(n)}$ of an algorithm A: if for any n-sized input the algorithm A produces a solution with value C such that

- $\frac{C}{OPT} \leq \rho(n)$ for a **minimization problem**, and
- $\frac{C}{OPT} \geq \rho(n)$ for a **maximization problem**.

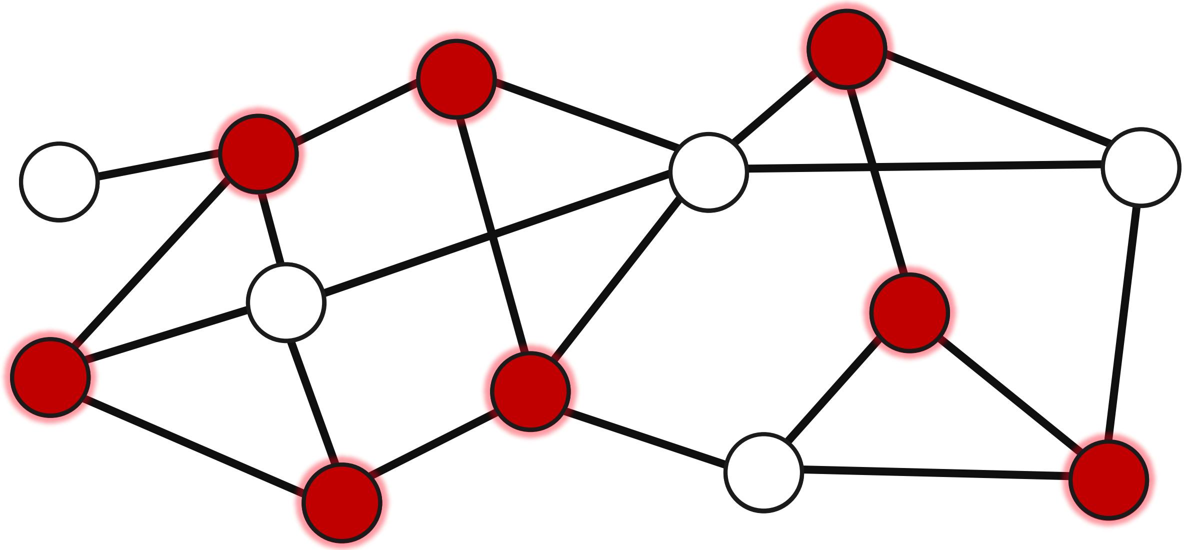
Minimization problem: $\rho(n) \geq 1$. **Maximization problem:** $\rho(n) \leq 1$.

There are different notions of approximation in the literature, e.g., being an additive term away from the optimal solution (above multiplicative factor), or OPT/C instead of C/OPT, or a mix of multiplicative and additive approximation.

Minimum Vertex Cover (MVC)



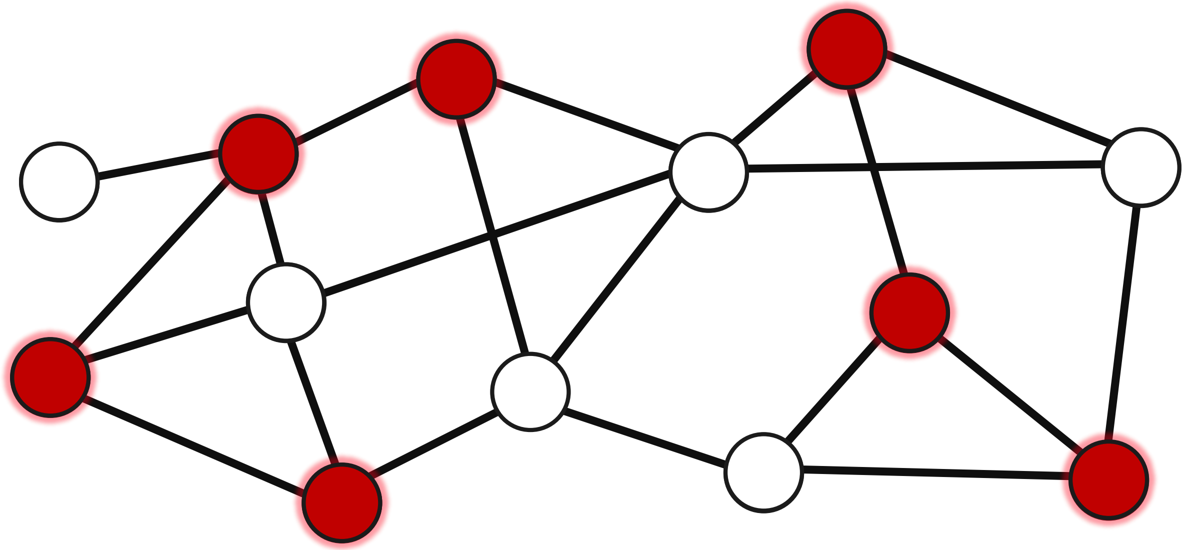
Minimum Vertex Cover (MVC)



A vertex cover of an undirected graph GG is a set S of vertices such that every edge in E(G) is incident to at least one vertex in S. In other words, for every edge (u,v) in GG, at least one of the vertices u or v is in the set S.

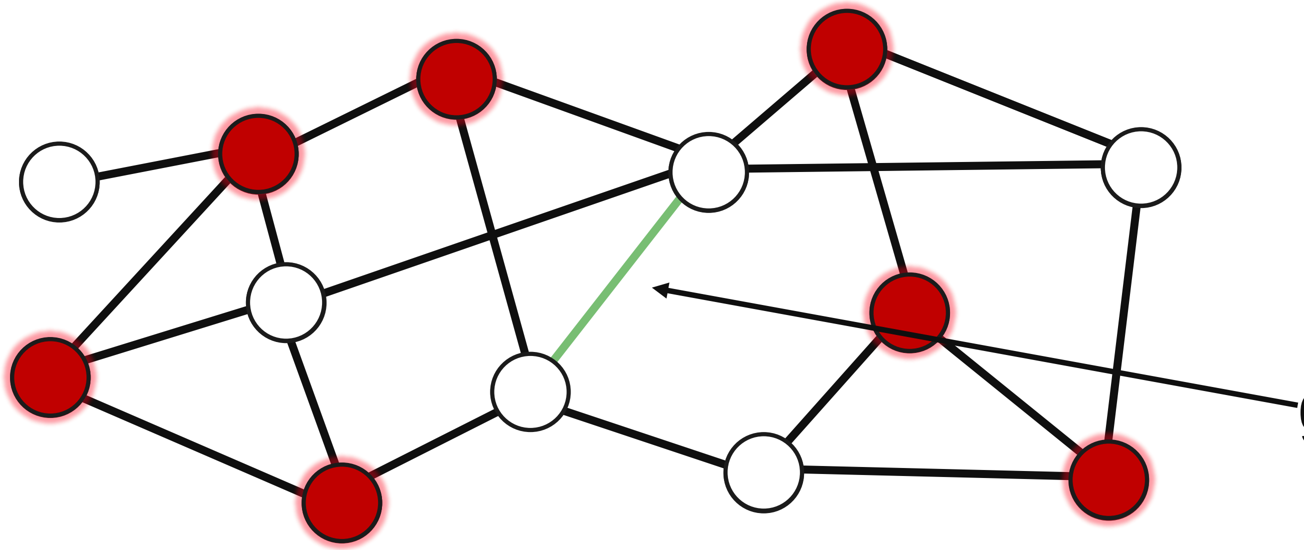
Goal: Compute a vertex cover of **minimum size**.

Examples for Vertex Cover

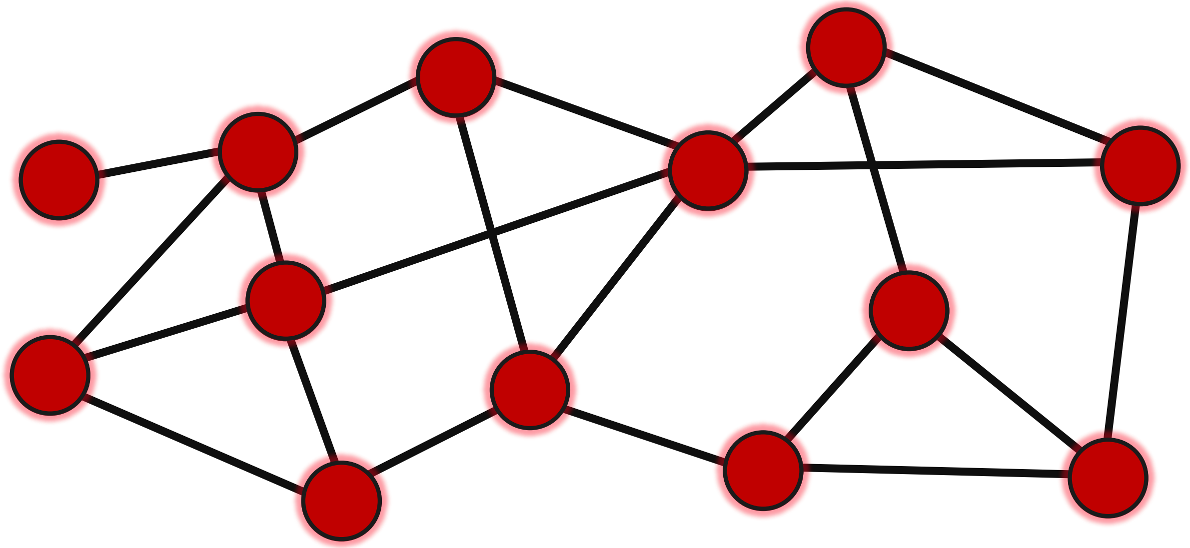


Not a valid cover.

Examples for Vertex Cover



Not a valid cover.
green edge not covered



Valid vertex cover,
but a really large one

Minimum Vertex Cover is NP-complete

Decision variant: Given graph G and k, is there a VC of G with size $\leq k$?

NP containment:

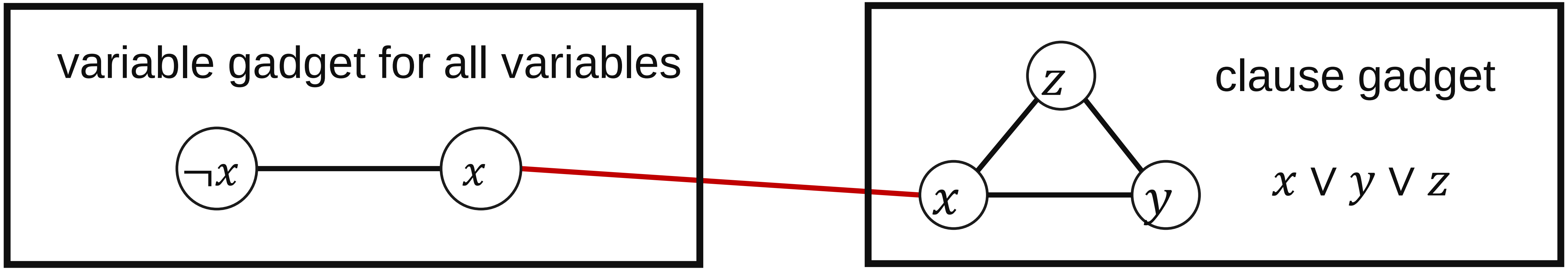
Polynomially verifiable whether a given subset $S \subseteq V$ is a valid vertex cover of size k.

Decision variant: Given graph G and k, is there a MVC of G with size $\leq k$?

NP-hardness:

Reduce 3-SAT to the decision variant of MVC (not part of this lecture)

Given 3-sat formular ϕ with m variables, l clauses, create graph G as follows:



An edge between “clause literal” each the corresponding variable literal

Exercise: 3-SAT formular ϕ *is satisfiable iff* G *has a MVC of size* $k = m + 2l$

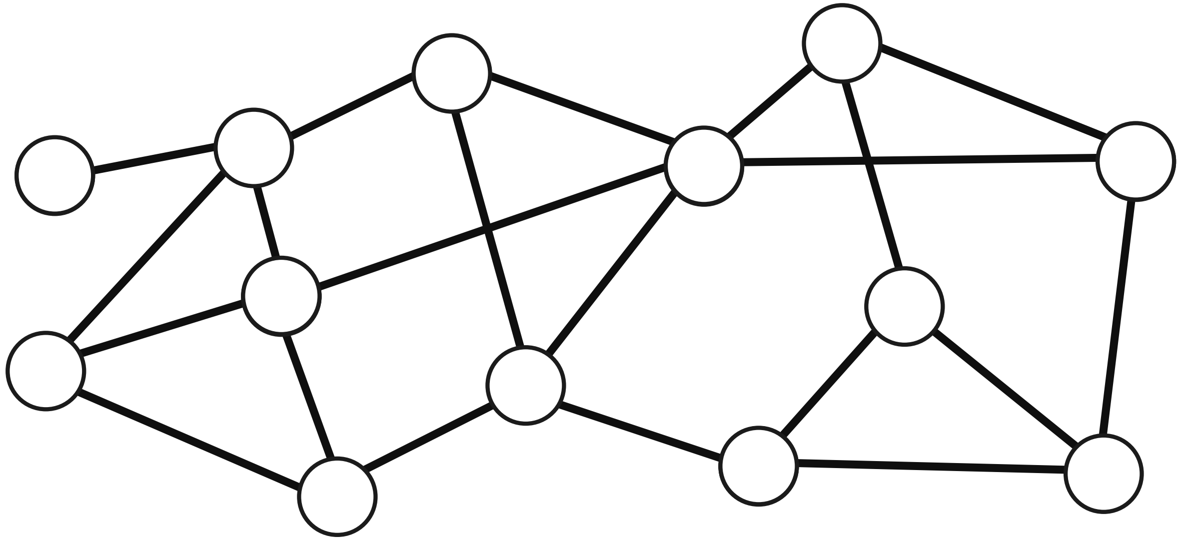
How do you solve the problem?

Greedy!

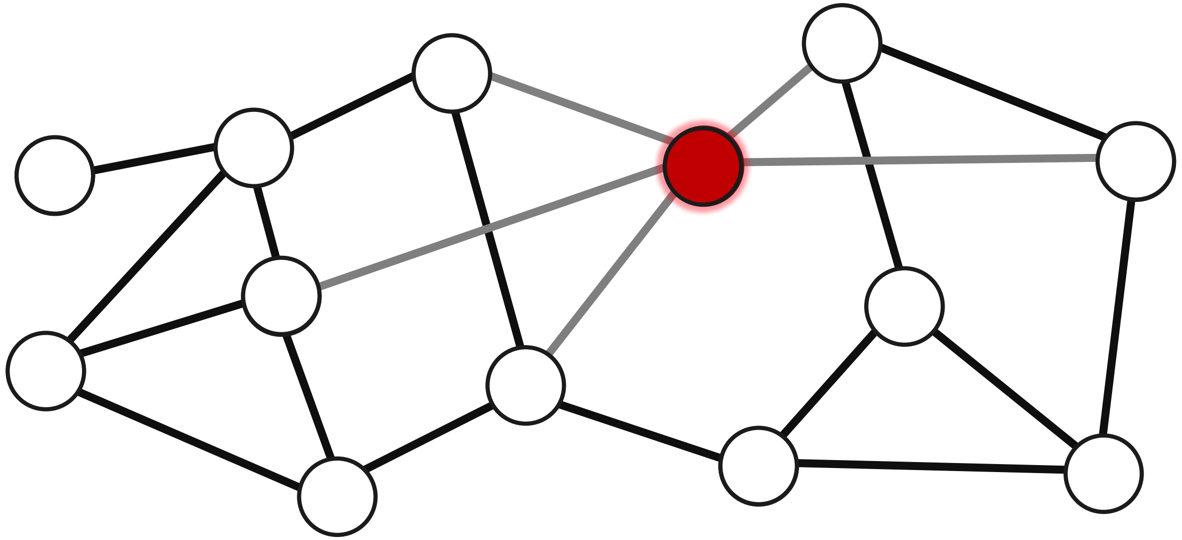
“Vertex greedy” for MVC

```
Input: Graph G = (V, E)
Output: Vertex cover C
C = empty set
F = E(G)
while F is not empty:
   choose largest degree vertex v in G=(V-C,F)
   add v to C
   remove all edges incident on v from F
return C
```

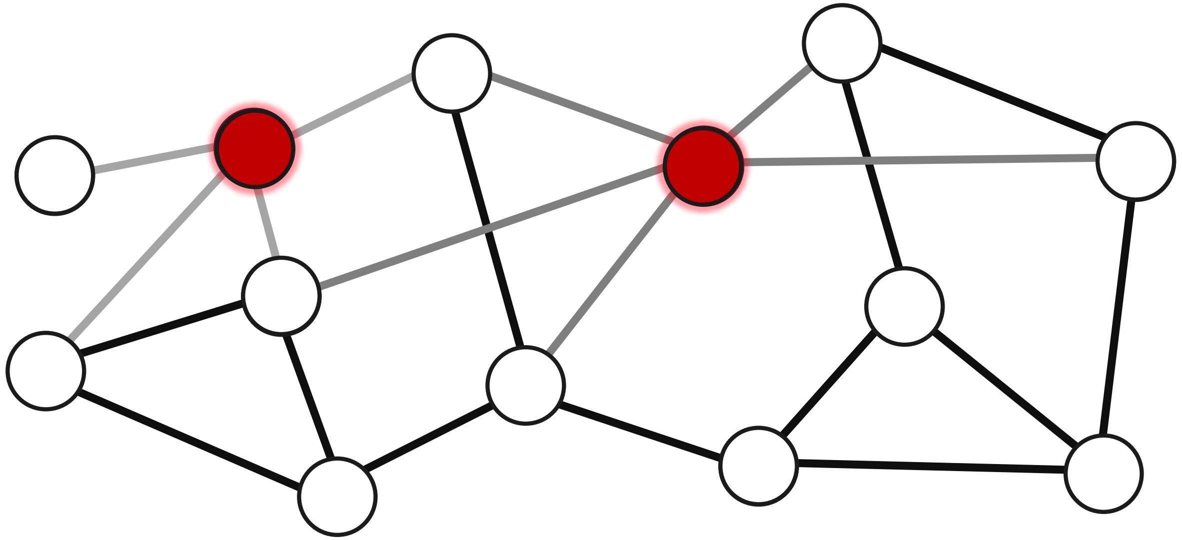
Example: Greedily adding vertices



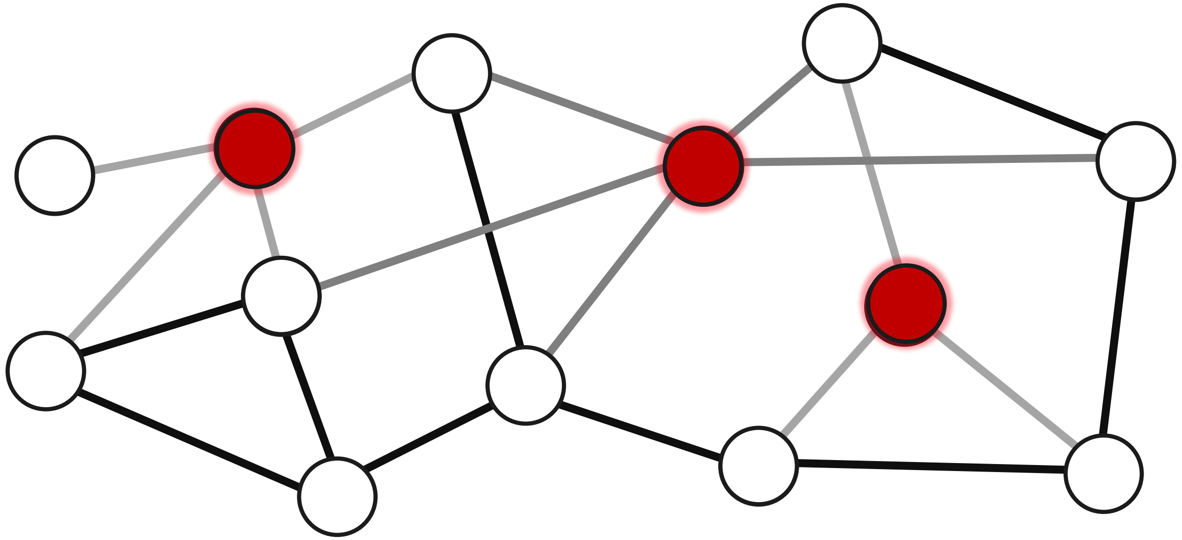
Example: Greedily adding vertices



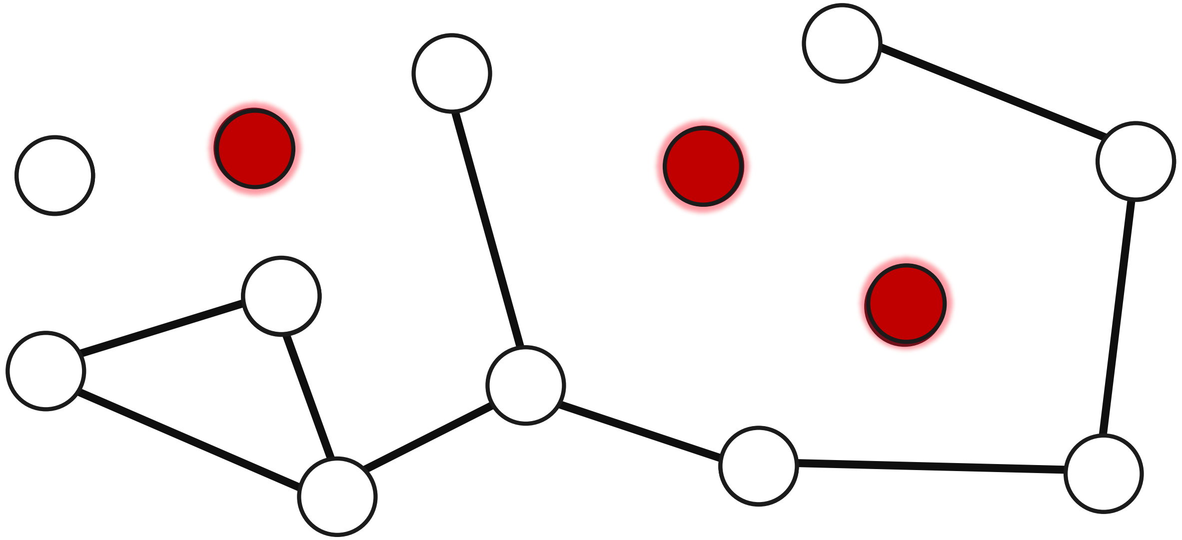
Example: Greedily adding vertices



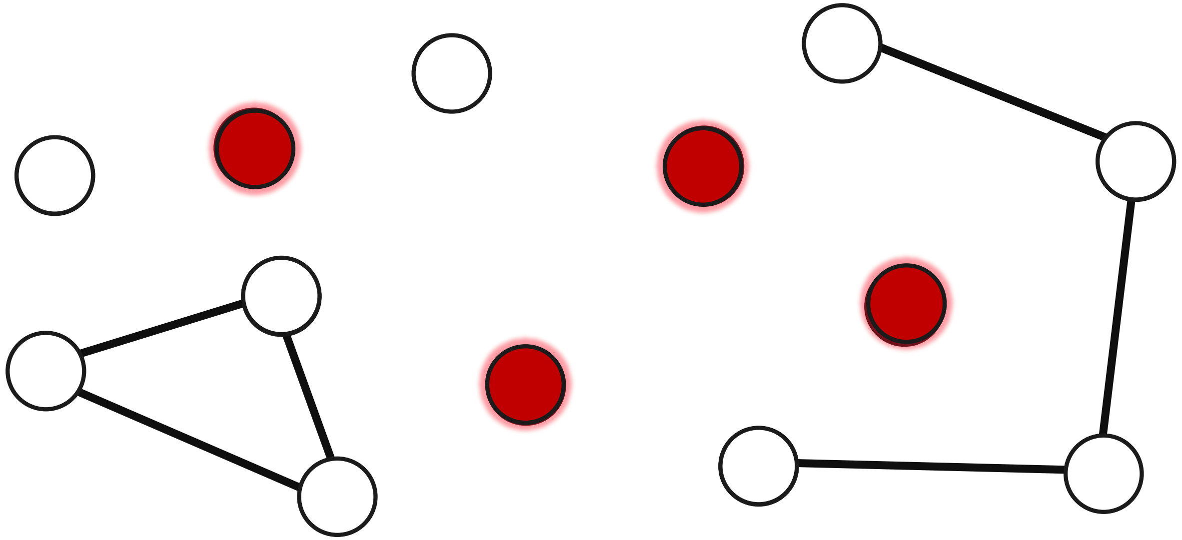
Example: Greedily adding vertices



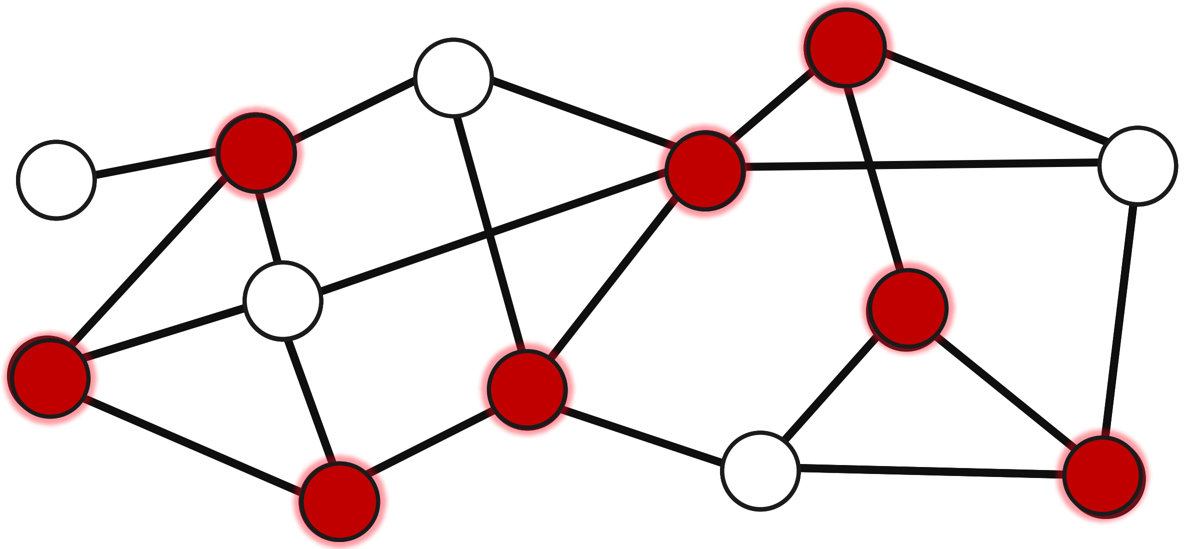
Example: Greedily adding vertices



Example: Greedily adding vertices



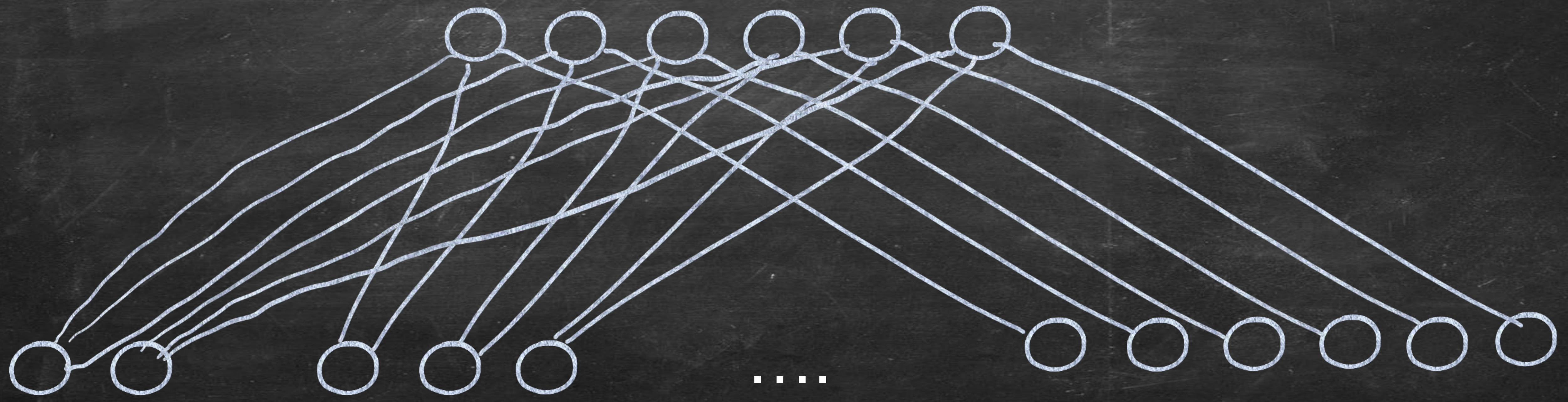
Example: Greedily adding vertices



vertex cover of size 8

Theorem: Greedily adding the largest degree vertex (in the graph induced by uncovered edges) is not a constant factor approximation for MVC.

This figure: $k = 3$

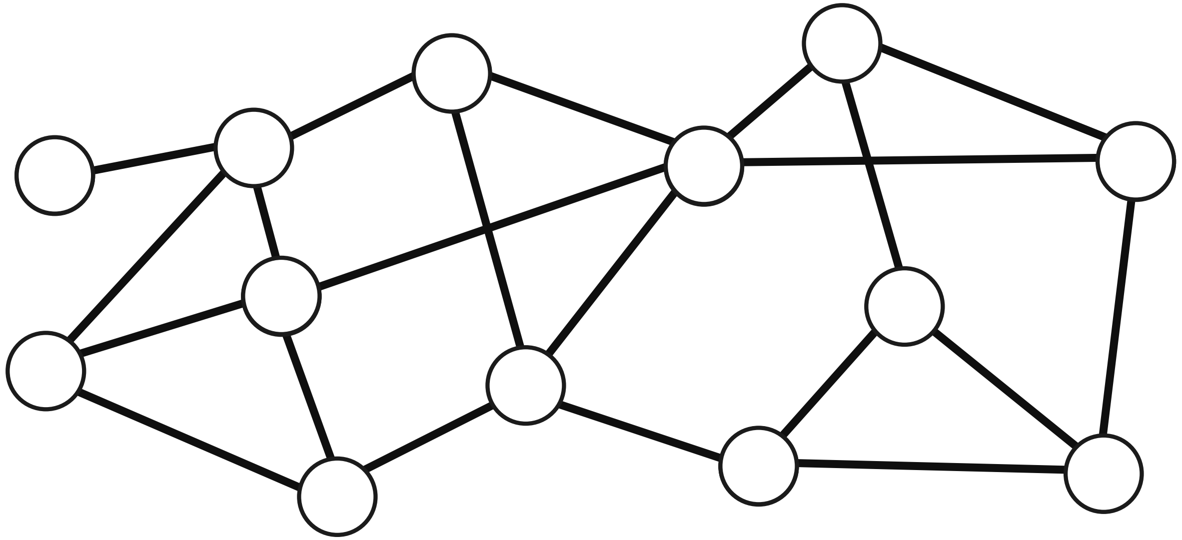


For $x = k, k-1, \ldots, 1$: $k!/x$ vertices degree x

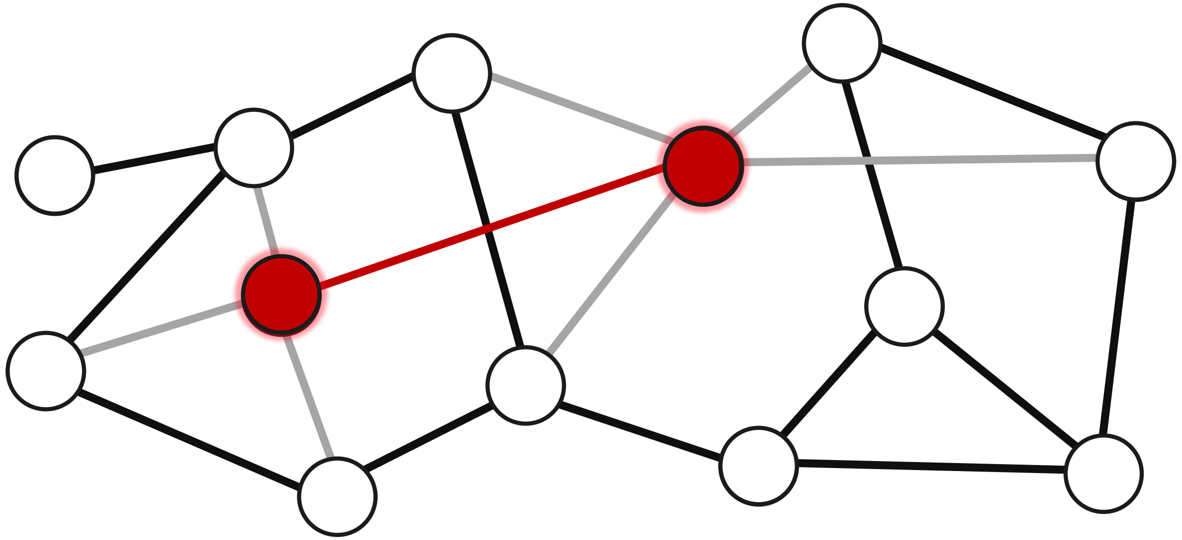
“Edge greedy” for MVC

```
Algorithm: ApproximateVertexCover(G)
Input: Graph G = (V, E)
Output: Vertex cover C
C = empty set
F = E(G)
while F is not empty:
    choose any edge (u, v) from F
    add u and v to C
    remove all edges incident on u and v from F
return C
```

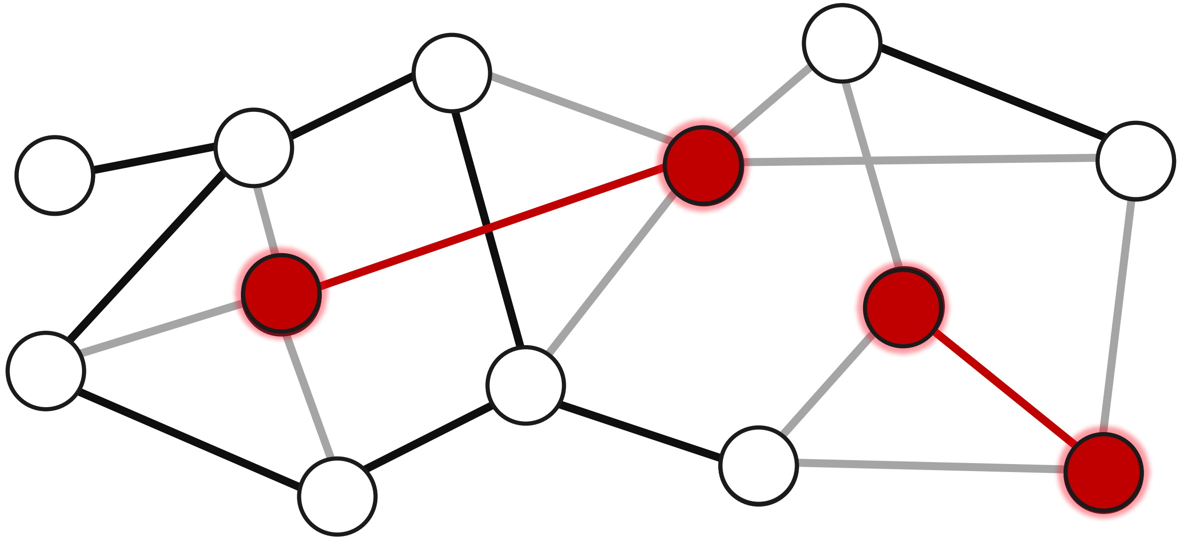
Example: Greedily adding vertices



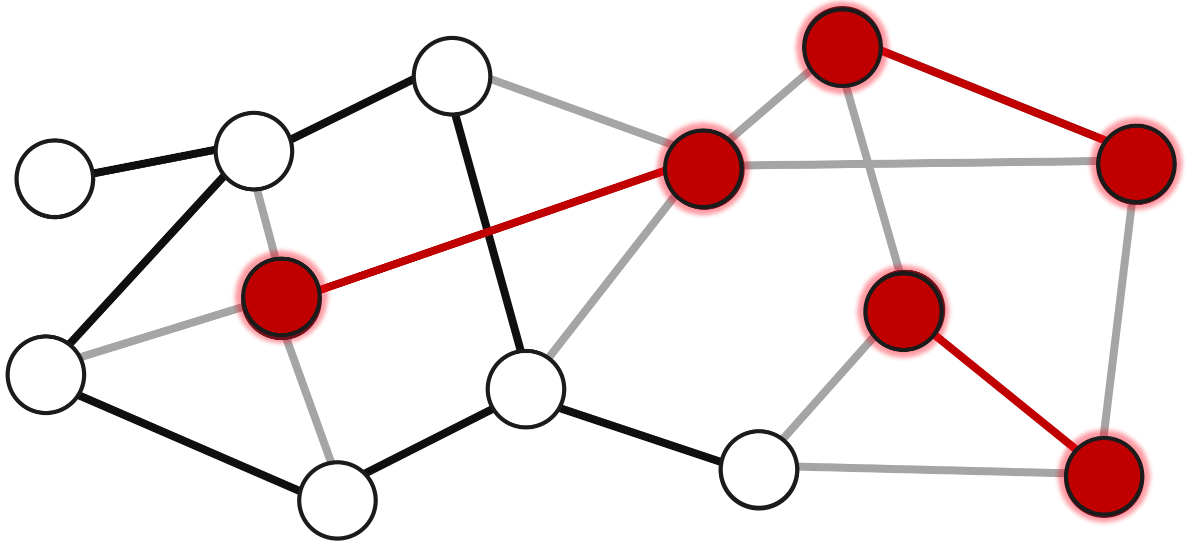
Example: Greedily adding vertices



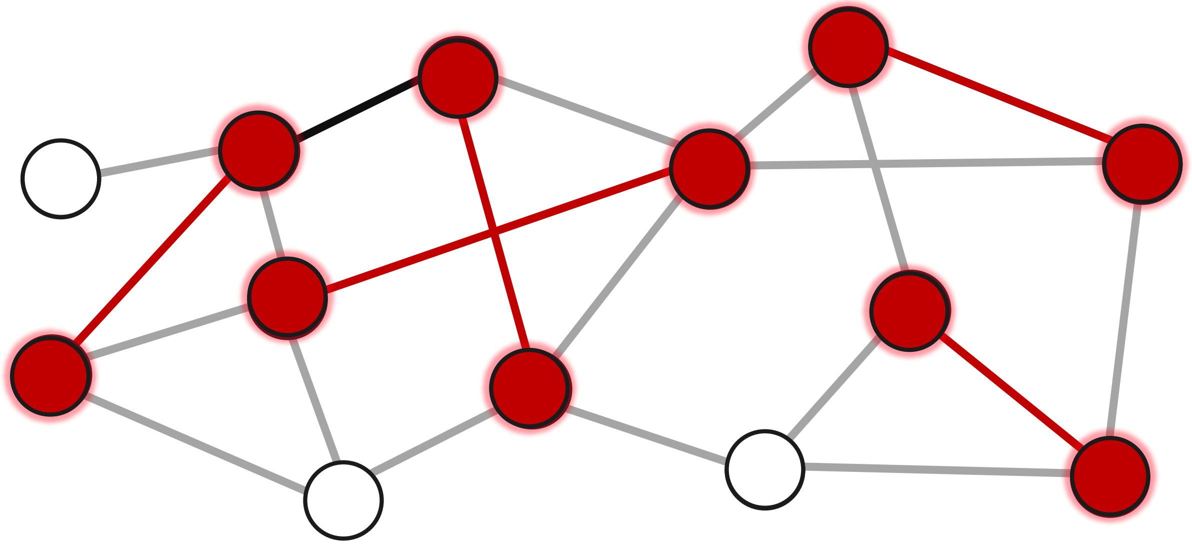
Example: Greedily adding vertices



Example: Greedily adding vertices



Example: Greedily adding vertices



vertex cover of size 10 (worse than previous VC)

Theoretically, “greedy edge picking” has a better approximation guarantee (next slide) than “greedy vertex picking”

MVC: Greedy adding edges is a 2-approximation

Theorem: Greedily adding both vertices of any still uncovered edge is a 2-approximation for the minimum vertex cover problem.

Proof:

F: the set of edges that our greedy algorithm picked. **F is a matching in $\boldsymbol{G}$.** Why?
$C = V(F)$ be the computed vertex cover.

- Clearly C is a vertex cover, as we continue adding vertices until all edges are covered.
- We have $|C| = 2|F|$ (no overlap, as F is a matching in G)

Let C_{OPT} be any optimal solution to MVC.
$|\boldsymbol{C_{OPT}}| \geq \boldsymbol{F}$, because C_{OPT} has to cover every edge, including all edges in F.
But no vertex of G can cover more than a single edge of F, as F is a matching.

$$\Rightarrow |C| = 2\,|F| \leq 2|C_{OPT}|.$$

Minimum Vertex Cover: Conclusion

- exact MVC is NP-complete
- “vertex greedy” does not give a constant approximation factor
- “edge greedy” gives 2-approximation
- that does not mean that “vertex greedy” is always worse than “edge greedy”

Hardness of approximation (not covered in this lecture):

- It is NP-hard to compute anything better than a $\sqrt{2} \approx 1.41$ –approximation

  [Khot, Minzer, Shafra, 2017]
- It is conjectured that it is NP-hard to compute anything better than a 2-eps approximation for any constant eps (unique games conjecture) [its details go well beyond the scope of this lecture!]

  [Khot, Regev, 2008]

Set Cover

Set Cover: Definition & Example

Input:

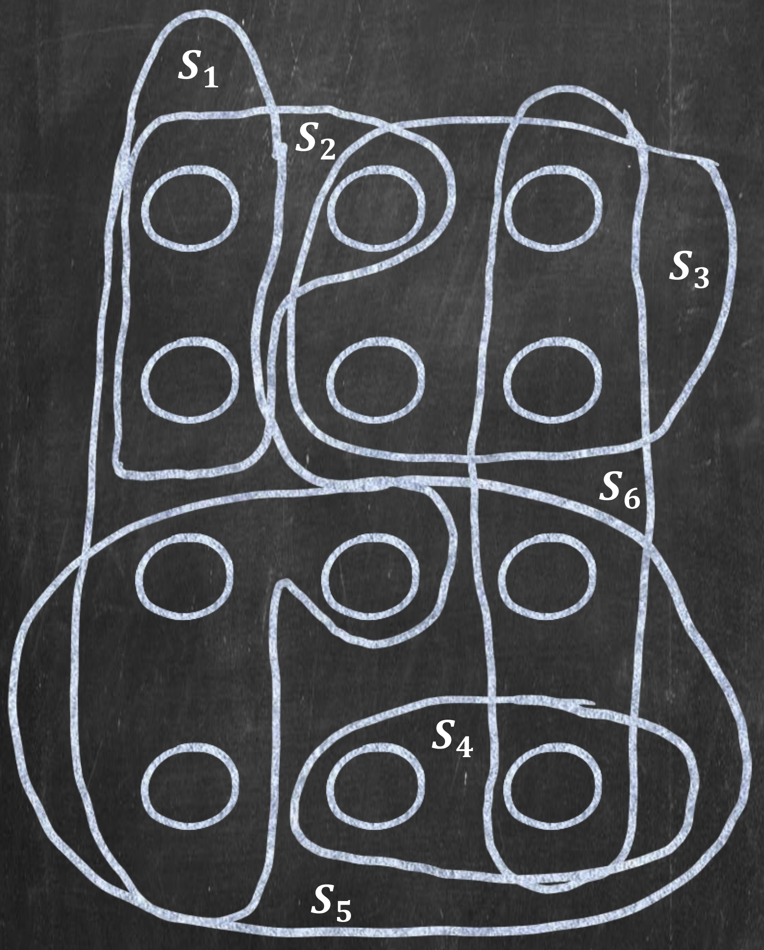
- **Universe** $X = \{x_1, \dots, x_n\}$ of n elements
- **Collection of Sets** $S = \{S_1, \dots, S_k\}$, each $S_i \subseteq X$

Set Cover: a collection of sets (indices) $I \subseteq \{1, \dots k\}$ s.t. all elements are covered, i.e.,

$$X \subseteq \bigcup_{i \in I} S_i$$

Goal: Select a minimum size set cover (minimize $|I|$).

(Minimum) set cover is NP-complete.



Optimal solution: S_1, S_3, S_5

How do you solve the problem?

Greedy!

Set Cover: Algorithm

```
GreedySetCover(Universe, Sets):
    I = {}
while X is not empty:
    MaxSet = argmax(Set in Sets, |Set ∩ X|)
    I = I∪{MaxSet}
    X = X - MaxSet
return I
```

Set Cover: Definition & Example

Input:
- **Universe** $X = \{x_1, \dots, x_n\}$ of n elements
- **Collection of Sets** $S = \{S_1, \dots, S_k\}$, each $S_i \subseteq X$

Set Cover: a collection of sets (indices) $I \subseteq \{1, \dots k\}$
s.t. all elements are covered, i.e.,

$$X \subseteq \bigcup_{i \in I} S_i$$

Goal: Select a minimum size set cover (minimize $|I|$).

(Minimum) set cover is NP-complete.

Set Cover: Analysis (Correctness)

Proof (Correctness):

The greedy algorithm computes a valid cover, as we keep adding sets until all elements are covered.

Let C_{OPT} be an optimal cover, let $t = |C_{OPT}|$
Let X_k be the elements in iteration k. $X_0 = X$

Claim: For all $0 \leq k$, the set X_k can be covered with t sets.
Proof: The original set X can be covered with t sets, so the same is true for $X_k \subseteq X$

In step k, there exists a set that covers at least $|X_k|/t$ elements (pigeonhole principle)

$\Rightarrow$ in step k the greedy algorithm is going to pick a set of size at least $|X_k|/t$

For all k, we have $|X_{k+1}| \leq \left(1 - \frac{1}{t}\right) |X_k|$

By induction for all $k \geq 0$: $|X_k| \leq \left(1 - \frac{1}{t}\right)^k |X_0| = \left(1 - \frac{1}{t}\right)^k \cdot |X|$

Greedy Set Cover: Analysis (Approximation factor)

Let C_{OPT} be an optimal cover, let $t = |C_{OPT}|$
Let X_k be the elements in iteration k. $X_0 = X$

By induction for all $k \geq 0 : |X_k| \leq \left(1 - \frac{1}{t}\right)^k |X_0| = \left(1 - \frac{1}{t}\right)^k \cdot |X|$

When do we stop? How many sets do we choose?

We stop when $X_k = \emptyset$ $(|X_k| < 1)$, chosen at most k sets
For $k^* = t \cdot (\lceil \log|X| \rceil + 1)$, *we obtain*

$$|X_{k^*}| \leq \left(1 - \frac{1}{t}\right)^{k^*} \cdot |X| \leq e^{-\frac{k^*}{t}} \cdot |\mathrm{X}| = \mathrm{e}^{-\frac{k^*}{t} + \log|X|} \leq e^{-1} < 1$$

At most $\boldsymbol{k^* = t \cdot (\lceil \log|X| \rceil + 1)}$ *sets, so we have a* $\boldsymbol{(\lceil \log|X| \rceil + 1)}$*-approximation.*

Conclusion: Set Cover

Theorem: Greedily adding the set that covers most uncovered elements is a $(\lceil \log|X| \rceil + 1)$-approximation for the (minimum) set cover problem

- This is not a constant approximation factor! The more elements, the worse the approximation!
- Still, the problem and algorithm appear are used quite often

Hardness of approximation (not covered in this lecture):

- It is known that computing a constant-factor approximation is NP-hard
  → (difficult) research area hardness of approximation

[Raz, Safra '97]

- Computing a better than (1-o(1))ln n-approximation is NP-hard

[Alon, Moshkovitz, Safra '06], [Dinur, Steuerer '13]

Partition

Partition Problem

Input:

n positive integers $s_1, \dots s_n$

Goal:

Partition the set of integers (integer indices) into two sets $A, B \subseteq \{1, \dots n\}$ to minimize

$$\max\left\{\sum_{i \in A} s_i , \sum_{i \in B} s_i\right\}$$

- NP-complete
- Dynamic programming: $O(n \cdot \sum s_i)$ (does not contradict NP-hardness)
- Bruteforce by trying all combinations: $O(2^n)$

Partition: Constant approximation? Trivial!

2-Approximation: Any distribution is a 2-approximation. Not very helpful

😢

Approximation Scheme (PTAS)

Polynomial time approximation scheme (PTAS): For each $\epsilon > 0$:
- Computes a $(1 \pm \epsilon)$-approximation
- Runtime is polynomial in input for fixed ϵ.

- The runtime can be different for different ϵ
- So $O(n^{1/\epsilon})$ is fine, e.g. for $\epsilon = 0.01$, this is $O(n^{100})$.
- Even $n^{\exp(\frac{1}{\epsilon})}$ is also fine, but maybe not useful in practice

A PTAS for Partition

Fix: $m = \left\lceil \frac{1}{\epsilon} \right\rceil - 1$

Order from largest to smallest $s_1 \geq s_2 \geq \cdots \geq s_n$.

Compute an **optimal solution (A,B)** for the first m elements.

Greedy for the rest:

```
For i=m+1,…,n
    If weight(A) ≤ weight (B) :
        add i to A,
    else
        add i to B.
```

Runtime: $O(n \cdot \log n + 2^{\frac{1}{\epsilon}+1} + n)$

Proof of Approximation factor

Notation:
A, B: Sets at the end of the algorithm
A_k, B_k: Sets after the k-th step

Proof: Wlog assume at the end we have $weight(A) \geq weight\ (B)$
Let s_k be the last element added to A.

Look at the snapshot after adding k:

We only need to prove $\mathrm{w}(\mathrm{A}) = w(A_k) \leq (1+\epsilon)OPT$

but we don't know OPT, how can we compare to it?

Compare with a lower bound to OPT

As we do not know OPT, we show that our solution is even a good approximation of L

$$\boldsymbol{L = \frac{1}{2}\sum s_i \leq OPT}$$

lower bound for OPT (for a maximization problem we would need an upper bound)

Proof of Approximation factor

Case 1 (k was added in the first phase) $\Rightarrow \boldsymbol{k \leq m}$

After the first phase, A was optimal for the smaller problem of adding the first m elements. Later, we never add anything to A. We obtain:

$$w(A) = w(A_k) \leq OPT(s_1, \dots, s_m) \leq OPT(s_1, \dots, s_n)$$

➔ Approximation ratio in this case is 1

Case 2 (k was added in the second phase) ⇒ $\boldsymbol{k > m}$

Claim: $s_k \leq 2L/(m+1)$

Proof: As $s_1 \geq s_2 \geq \cdots \geq s_k$ we get:

$$2L \geq \sum_{i=1}^{k} s_i \geq \sum_{i=1}^{k} s_k \geq (m+1) \cdot s_k,$$

claim follows by dividing by m+1.

$w(A) \leq w(B_{k-1}) + s_k$ (we added k to A because of w$(A_{k-1}) \leq w(B_{k-1})$, and never changed A afterwards)

$$w(A) \leq w(B_{k-1}) + s_k \leq w(B) + s_k = 2L - w(A) + s_k$$

$$w(A) \leq L + 0.5 \cdot s_k \leq (1+\epsilon)L \leq (1+\epsilon)OPT$$

PTAS for the partition problem

Theorem: For any $\epsilon > 0$ there exists an algorithm that computes a $(1+\epsilon)$-approximation of the partition problem in time $O(n \cdot \log n + 2^{\frac{1}{\epsilon}+1} + n)$.

Conclusion & Tips

- Often simple greedy algorithms provide good approximations, but not always
- Sometimes it is hard to compare the algorithms performance with OPT, instead compare with a "lower bound" on OPT (as the L value in the partition proof, or $|OPT| \geq |Matching|$ in MVC)
- PTAS: can approximate arbitrarily well, but one pays for it in terms of runtime
- Not discussed: often approximation also makes sense for problems that are not NP-complete

In practice: *If you use a "heuristic" or "greedy" approach, you should ask yourself whether you provide a theoretical guarantee for it. Unfortunately, often this is not possible (e.g., in many machine learning algorithms)*