CSE 421
Introduction to Algorithms

Lecture 25: Finishing NP Completeness
Dealing with NP-completeness: Approximation Algorithms

See Edstem message re- final exam scheduling

Email/post privately if you CANNOT use extra time

3SAT$\leq_P$Subset-Sum

Given a 3-CNF formula $\boldsymbol{F}$ with $\boldsymbol{m}$ clauses and $\boldsymbol{n}$ variables

- We will create an input for **Subset-Sum** with $\mathbf{2}\boldsymbol{m} + \mathbf{2}\boldsymbol{n}$ numbers that are $\boldsymbol{m} + \boldsymbol{n}$ digits long.
- We will ensure that no matter how we sum them there won't be any carries so each digit in the target $\boldsymbol{W}$ will force a separate constraint.
- Instead of calling them $\boldsymbol{w_1}, \dots, \boldsymbol{w_{2n+2m}}$ we will use mnemonic names:
  - Two numbers for each variable $\boldsymbol{x_i}$
    - $\boldsymbol{t_i}$ and $\boldsymbol{f_i}$ (corresponding to $\boldsymbol{x_i}$ being true or $\boldsymbol{x_i}$ being false)
  - Two extra numbers for each clause $\boldsymbol{C_j}$
    - $\boldsymbol{a_j}$ and $\boldsymbol{b_j}$ (two identical filler numbers to handle number of false literals in clause $\boldsymbol{C_j}$)
- We define them by giving their decimal representation...

$3SAT \leq_p$ Subset-Sum

We include two $n + m$ digit numbers for each Boolean variable x_i

	1	2	3	i	…	n	1	2	3	j	…	m	
$t_i =$	0	0	0	1	…	0	1	0	0	0	…	1	Clauses C_1 and C_m contain x_i
$f_i =$	0	0	0	1	…	0	0	1	0	1	…	0	Clauses C_2 and C_j contain $\neg x_i$

Boolean part in the first n positions:

- Digit i of both t_i and f_i are 1; the rest are 0

Clause part in the next m positions:

- Digit j of t_i is 1 if clause C_j contains literal x_i; the rest are 0
- Digit j of f_i is 1 if clause C_j contains literal $\neg x_i$; the rest are 0

3SAT$\leq_P$Subset-Sum

We also include two extra identical $n + m$ digit numbers for each clause C_j

	1	2	3	i	…	n	1	2	…	j	…	m
$a_j =$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	…	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	…	$\mathbf{1}$	…	$\mathbf{0}$
$b_j =$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	…	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	…	$\mathbf{1}$	…	$\mathbf{0}$

These are:

- All $\mathbf{0}$ in the Boolean columns
- Digit j of both a_j and b_j are $\mathbf{1}$ in the clause columns; the rest are $\mathbf{0}$

To be used for false literals in clause C_j with a satisfying truth asgnt.

3SAT $\leq_P$ Subset-Sum

C_1 C_2 C_3 C_4 C_m

				i						j		
	1	2	3	4	...	n	1	2	3	4	...	m
$t_1 =$	1	0	0	0	...	0	1	0	0	0	...	1
$f_1 =$	1	0	0	0	...	0	0	1	0	1	...	0
$t_2 =$	0	1	0	0	...	0	0	1	0	0	...	0
$f_2 =$	0	1	0	0	...	0	1	0	0	0	...	0
$t_3 =$	0	0	1	0	...	0	1	0	0	0	...	0
$f_3 =$	0	0	1	0	...	0	0	0	1	1	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...
$a_1 =$	0	0	0	0	...	0	1	0	0	0	...	0
$b_1 =$	0	0	0	0	...	0	1	0	0	0	...	0
$a_2 =$	0	0	0	0	...	0	0	1	0	0	...	0
$b_2 =$	0	0	0	0	...	0	0	1	0	0	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...
$W =$	1	1	1	1	...	1	3	3	3	3	...	3

$C_1 = (x_1 \vee \neg x_2 \vee x_3)$

$C_2 = (\neg x_1 \vee x_2 \vee x_5)$

$C_3 = (\neg x_3 \vee x_4 \vee x_7)$

$C_4 = (\neg x_1 \vee \neg x_3 \vee x_9)$

...

$C_m = (x_1 \vee \neg x_8 \vee x_{22})$

Boolean variable part: First n digit positions ensure that exactly one of t_i or f_i is included in any subset summing to W.

Clause part: 1's in each digit position j correspond to the 3 literals that would make clause C_j true.

Every column in the clause part of the block of t's and f's has exactly 3 1's.

The a's and b's add exactly 2 more possible 1's per column

3SAT$\leq_P$Subset-Sum

Boolean variable part:
First n digit positions ensure that exactly one of t_i or f_i is included in any subset summing to W.

				i						j		
	1	2	3	4	…	n	1	2	3	4	…	m
$t_1 =$	1	0	0	0	…	0	1	0	0	0	…	1
$f_1 =$	1	0	0	0	…	0	0	1	0	1	…	0
$t_2 =$	0	1	0	0	…	0	0	1	0	0	…	0
$f_2 =$	0	1	0	0	…	0	1	0	0	0	…	0
$t_3 =$	0	0	1	0	…	0	1	0	0	0	…	0
$f_3 =$	0	0	1	0	…	0	0	0	1	1	…	0
…	…	…	…	…	…	…	…	…	…	…	…	…
$a_1 =$	0	0	0	0	…	0	1	0	0	0	…	0
$b_1 =$	0	0	0	0	…	0	1	0	0	0	…	0
$a_2 =$	0	0	0	0	…	0	0	1	0	0	…	0
$b_2 =$	0	0	0	0	…	0	0	1	0	0	…	0
…	…	…	…	…	…	…	…	…	…	…	…	…
$W =$	1	1	1	1	…	1	3	3	3	3	…	3

$C_1 = (x_1 \vee \neg x_2 \vee x_3)$

$C_2 = (\neg x_1 \vee x_2 \vee x_5)$

$C_3 = (\neg x_3 \vee x_4 \vee x_7)$

$\boxed{C_4 = (\neg x_1 \vee \neg x_3 \vee x_9)}$

…

$C_m = (x_1 \vee \neg x_8 \vee x_{22})$

Key idea of clause columns:
Column j can sum to the target column sum of 3
$\Leftrightarrow$ at least one of the t_i or f_i rows included in the subset contains a 1 in column j

The a's and b's add exactly 2 more possible 1's per column

3SAT$\leq_P$Subset-Sum

If F satisfiable choose one of t_i or f_i depending on the satisfying assignment.
Their sum will have exactly one $\mathbf{1}$ in each of the first n digits and at least one $\mathbf{1}$ in every clause digit position.
Also include 0, 1, or 2 of each a_j, b_j pair to add to W.

				i					j			
	1	2	3	4	…	n	1	2	3	4	…	m
$t_1 =$	1	0	0	0	…	0	1	0	0	0	…	1
$f_1 =$	1	0	0	0	…	0	0	1	0	1	…	0
$t_2 =$	0	1	0	0	…	0	0	1	0	0	…	0
$f_2 =$	0	1	0	0	…	0	1	0	0	0	…	0
$t_3 =$	0	0	1	0	…	0	1	0	0	0	…	0
$f_3 =$	0	0	1	0	…	0	0	0	1	1	…	0
…	…	…	…	…	…	…	…	…	…	…	…	…
$a_1 =$	0	0	0	0	…	0	1	0	0	0	…	0
$b_1 =$	0	0	0	0	…	0	1	0	0	0	…	0
$a_2 =$	0	0	0	0	…	0	0	1	0	0	…	0
$b_2 =$	0	0	0	0	…	0	0	1	0	0	…	0
…	…	…	…	…	…	…	…	…	…	…	…	…
$W =$	1	1	1	1	…	1	3	3	3	3	…	3

If some subset sums to W must have exactly one of t_i or f_i for each i.
Set variable x_i to true if t_i used and false if f_i used.
Must have three $\mathbf{1}$'s in each clause digit column j since things sum to W.
At most two of these can come from a_j, b_j to one of these $\mathbf{1}$'s must come from the choices of the truth assignment $\Rightarrow$ every clause C_j is satisfied so F is satisfiable.

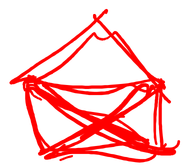
■

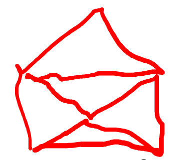
Some other $\mathbf{NP}$-complete examples you should know

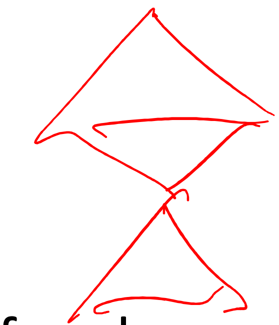
Hamiltonian-Cycle: Given a directed graph $G = (V, E)$. Is there a cycle in G that visits each vertex in V exactly once?

Hamiltonian-Path: Given a directed graph $G = (V, E)$. Is there a path p in G of length $n - 1$ that visits each vertex in V exactly once?

Same problems are also $\mathbf{NP}$-complete for undirected graphs







Note: If we asked about visiting each *edge* exactly once instead of each vertex, the corresponding problems are called **Euler Tour**, **Eulerian-Path** and are polynomial-time solvable.

Travelling-Salesperson Problem (TSP)

Travelling-Salesperson Problem (TSP):

Given: a set of n cities $v_1, \dots, v_n$ and distance function d that gives distance $d(v_i, v_j)$ between each pair of cities

Find the shortest tour that visits all n cities.

DecisionTSP:

Given: a set of n cities $v_1, \dots, v_n$ and distance function d that gives distance $d(v_i, v_j)$ between each pair of cities *and* a distance D

Is there a tour of total length at most D that visits all n cities?

NP-complete problems we've discussed

3SAT → Independent-Set → Clique

↓

Vertex-Cover → 01-Programming → Integer-Programming

↓

Set-Cover

→ 3Color

→ Subset-Sum

→ Hamiltonian-Cycle → DecisionTSP

→ Hamiltonian-Path

Some intermediate problems

Problems reducible to $\mathbf{NP}$ problems not known to be polytime:

Basis for the security of current cryptography:

- **Factoring:** Given an integer N in binary, find its prime factorization.
- **Discrete logarithm:** Given prime p in binary, and g and x modulo p.
  Find y such that $x \equiv g^y \pmod{p}$ if it exists.

Best algorithms known are $2^{\widetilde{\Theta}(n^{1/3})}$ time.

Other famous ones:

- **Graph Isomorphism:** Given graphs G and H, can they be relabelled to be the same? Best algorithm now $n^{\Theta(\log^2 n)}$ (recently improved from $2^{\widetilde{\Theta}(n^{1/3})}$) time.
- **Nash equilibrium:** Given a multiplayer game, find randomized strategies for each player so that no player could do better by deviating.

What to do if the problem you want to solve is NP-hard

1st thing to try:

- You might have phrased your problem too generally
  - e.g., In practice, the graphs that actually arise are far from arbitrary
    - Maybe they have some special characteristic that allows you to solve the problem in your special case
      - For example the **Independent-Set** problem is easy on "interval graphs"
        - Exactly the case for the Interval Scheduling problem!
  - Search the literature to see if special cases already solved

eg Vertex Cover

easy if graph is bipartite!

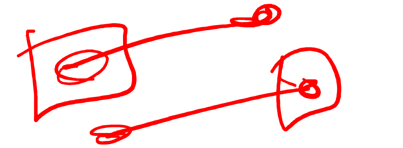
What to do if the problem you want to solve is NP-hard

2nd thing to try if your problem is a minimization or maximization problem

- Try to find a polynomial-time worst-case approximation algorithm
  - For a minimization problem
    - Find a solution with value $\leq \boldsymbol{K}$ times the optimum
  - For a maximization problem
    - Find a solution with value $\geq \mathbf{1}/\boldsymbol{K}$ times the optimum

Want $\boldsymbol{K}$ to be as close to $\mathbf{1}$ as possible.

Greedy Approximation for Vertex-Cover



This is a better approximation factor than the greedy algorithm that repeatedly chooses the highest degree vertex remaining.

On input $G = (V, E)$

$W \leftarrow \emptyset$

$E' \leftarrow E$

while $E' \neq \emptyset$

  select any $e = (u, v) \in E'$

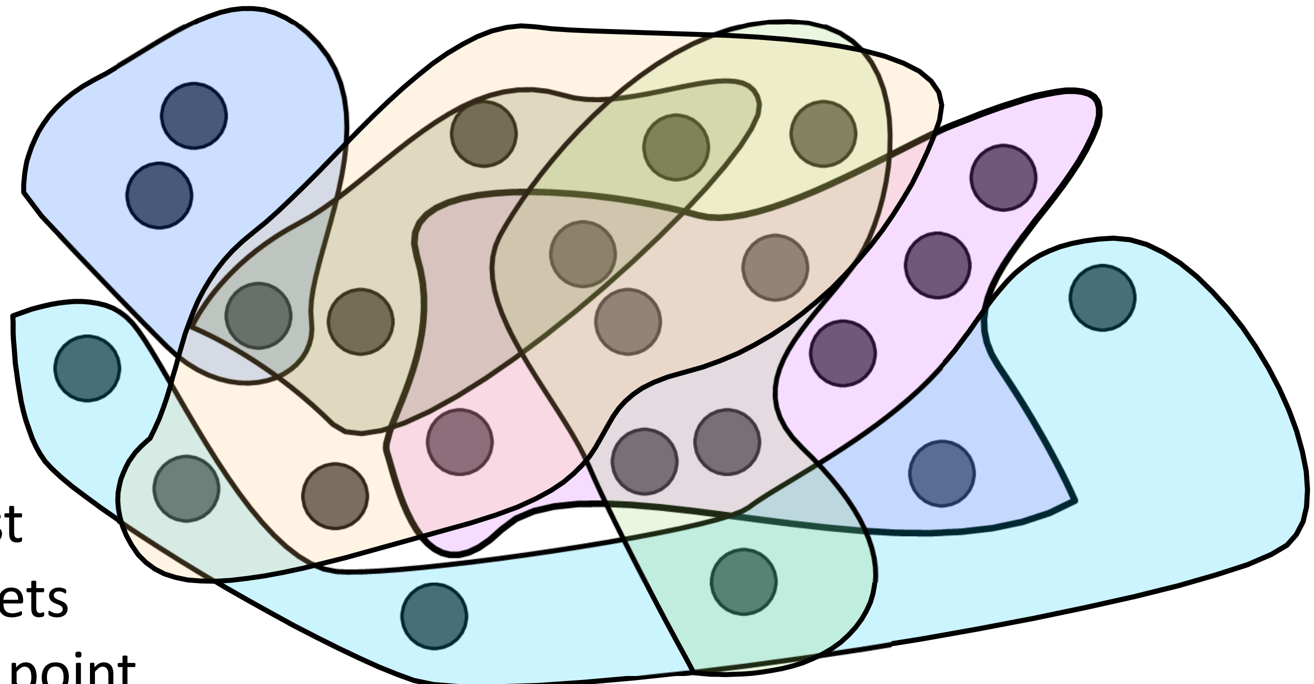
  $W \leftarrow W \cup \{u, v\}$ add both endpoints

  $E' \leftarrow E' \setminus \{\text{edges } e \in E' \text{ that touch } u \text{ or } v\}$

Claim: At most a factor 2 larger than the optimal vertex-cover size.

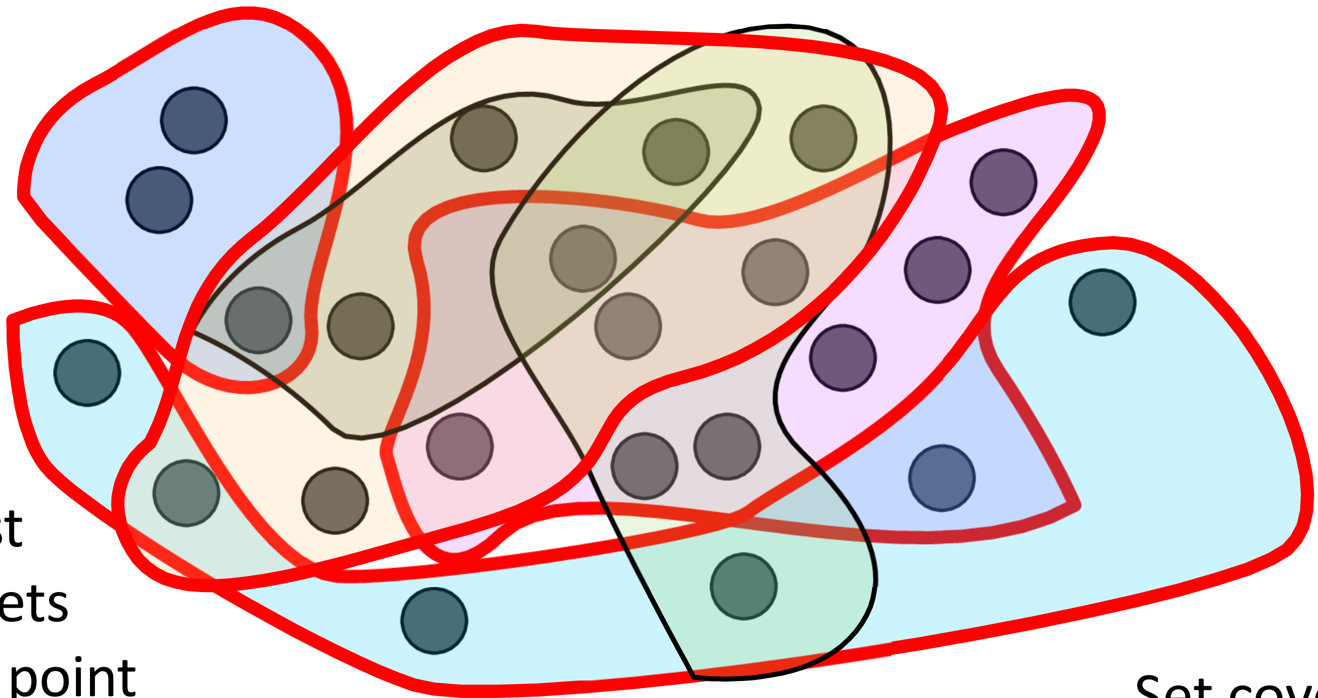
Proof: Edges selected don't share any vertices so any vertex-cover must choose at least one of u or v each time.

Set-Cover



Find smallest
collection of sets
containing every point

Set-Cover

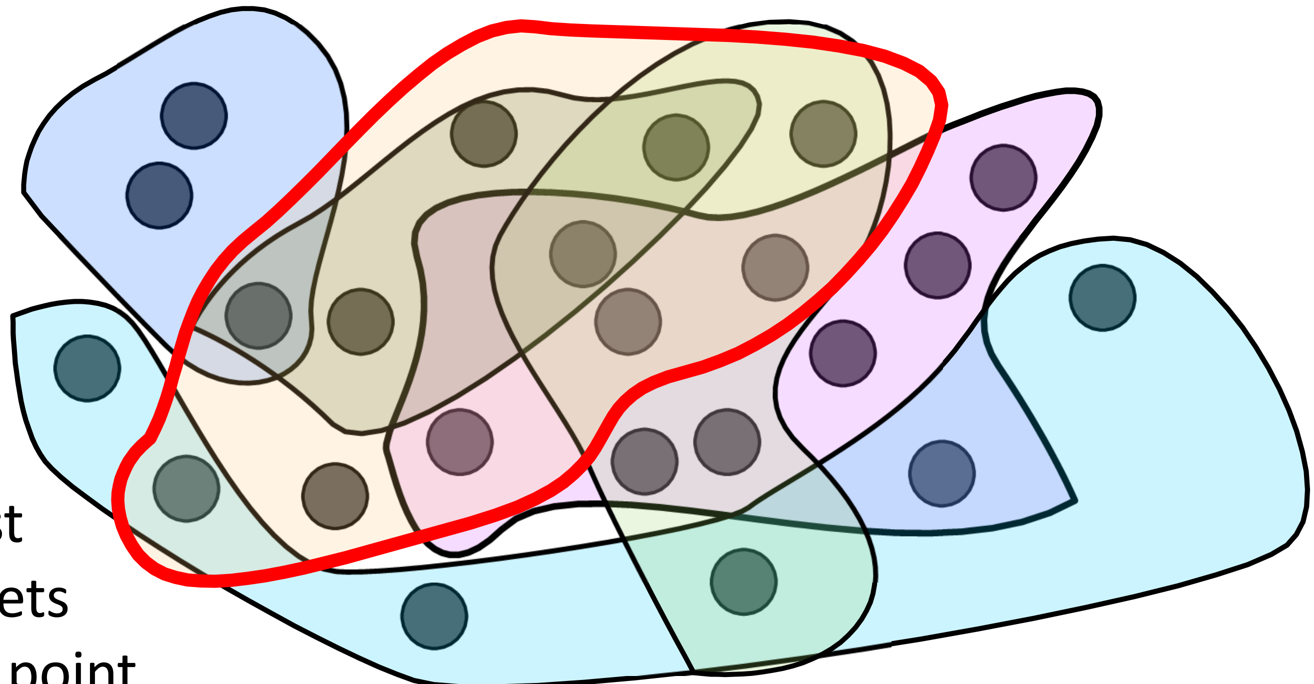


Find smallest collection of sets containing every point

Set cover size **4**

Set-Cover

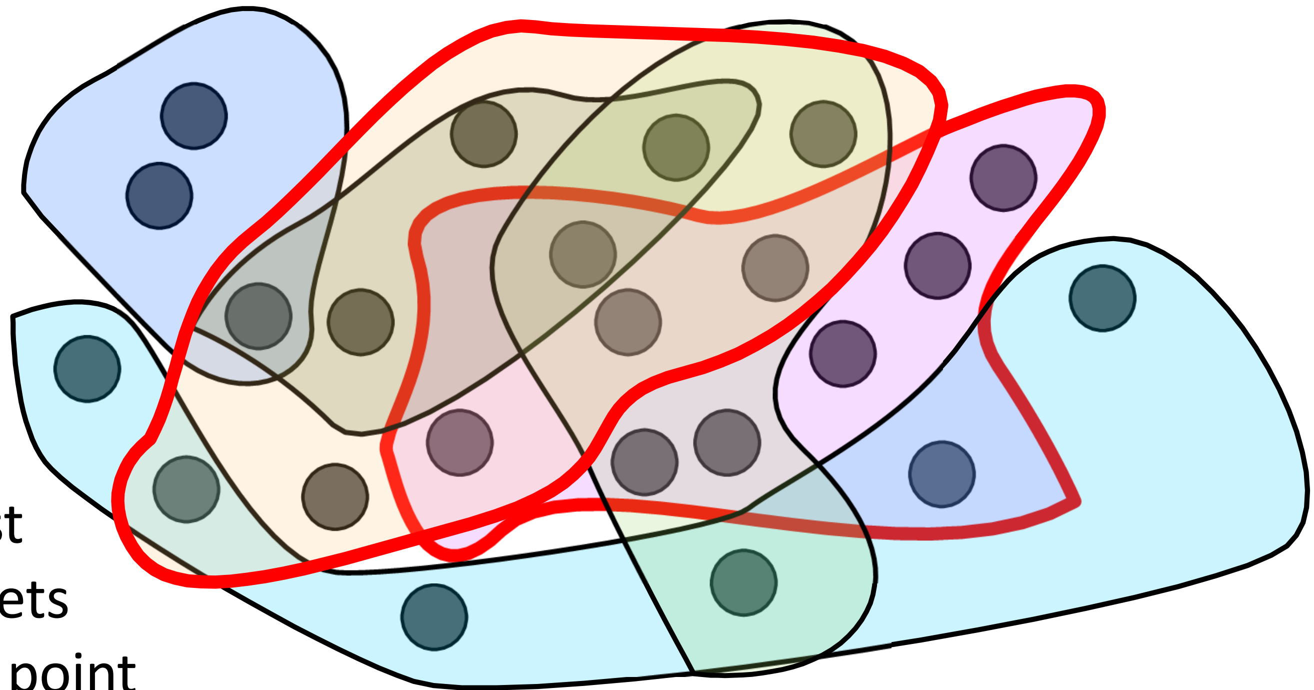
Greedy Set Cover: Repeatedly choose the set that covers the most # of new elements



Find smallest collection of sets containing every point

Set-Cover

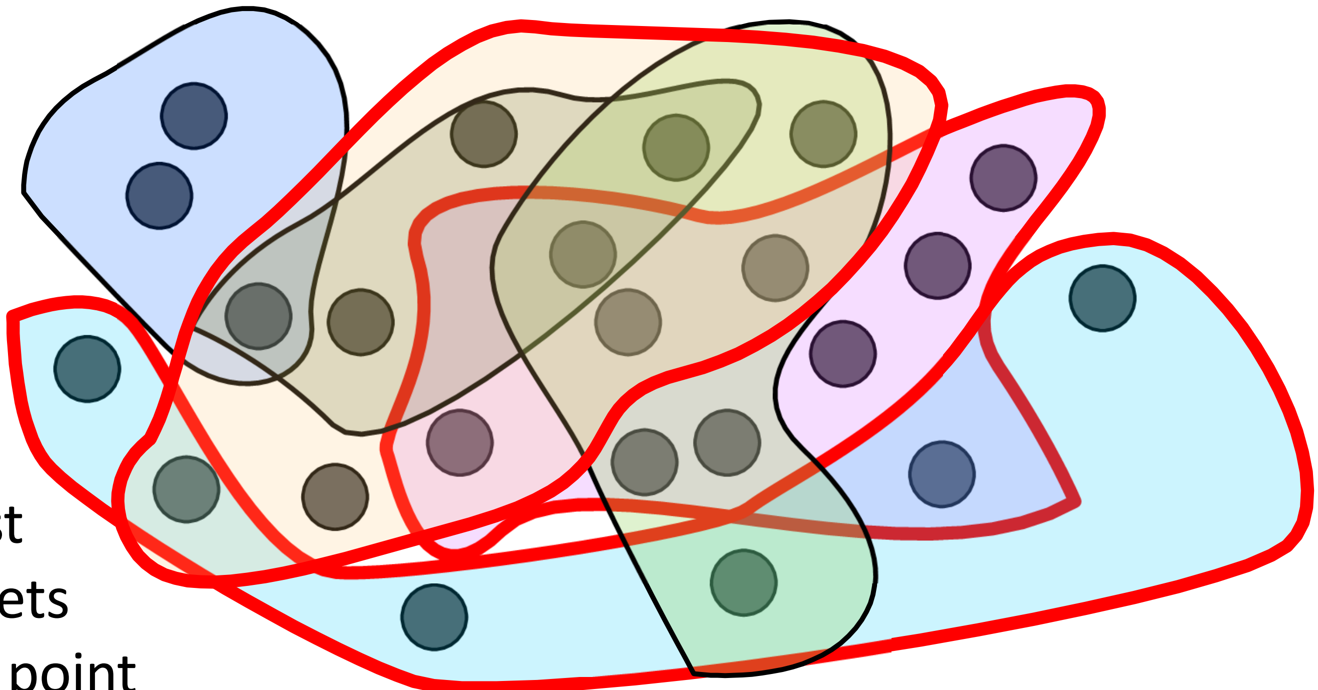
Greedy Set Cover: Repeatedly choose the set that covers the most # of new elements



Find smallest collection of sets containing every point

Set-Cover

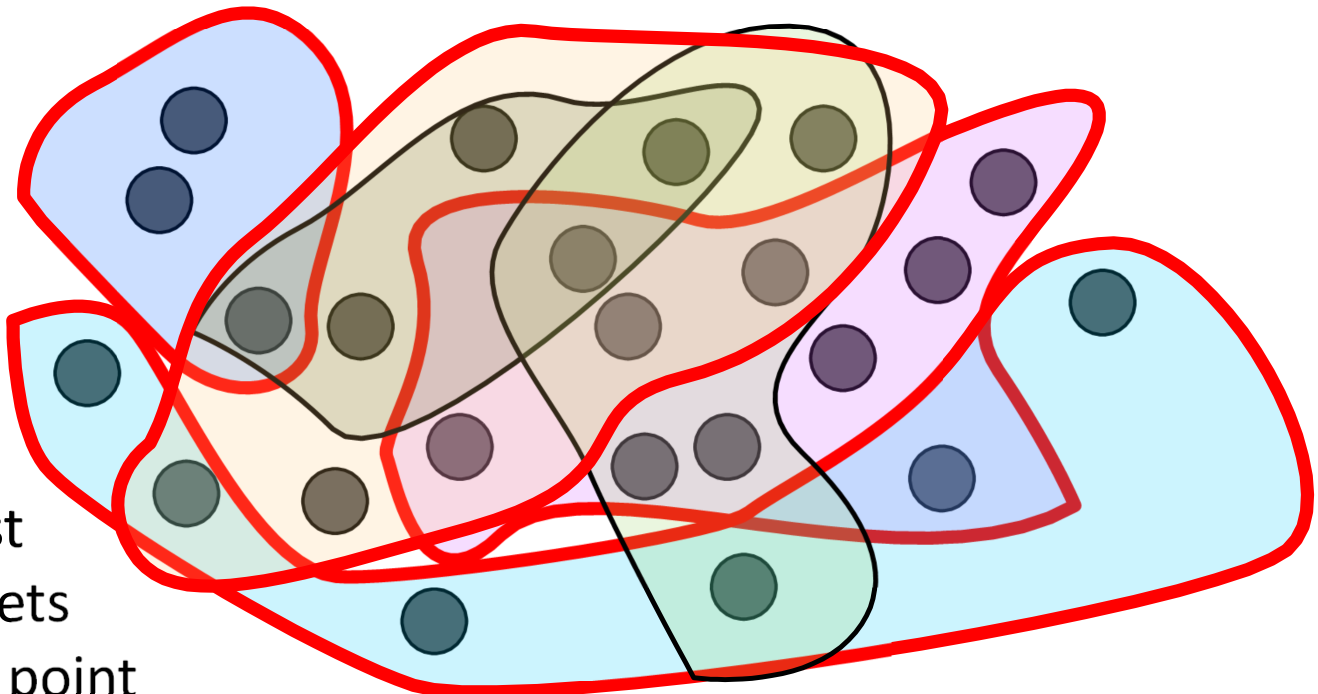
Greedy Set Cover: Repeatedly choose the set that covers the most # of new elements



Find smallest collection of sets containing every point

Set-Cover

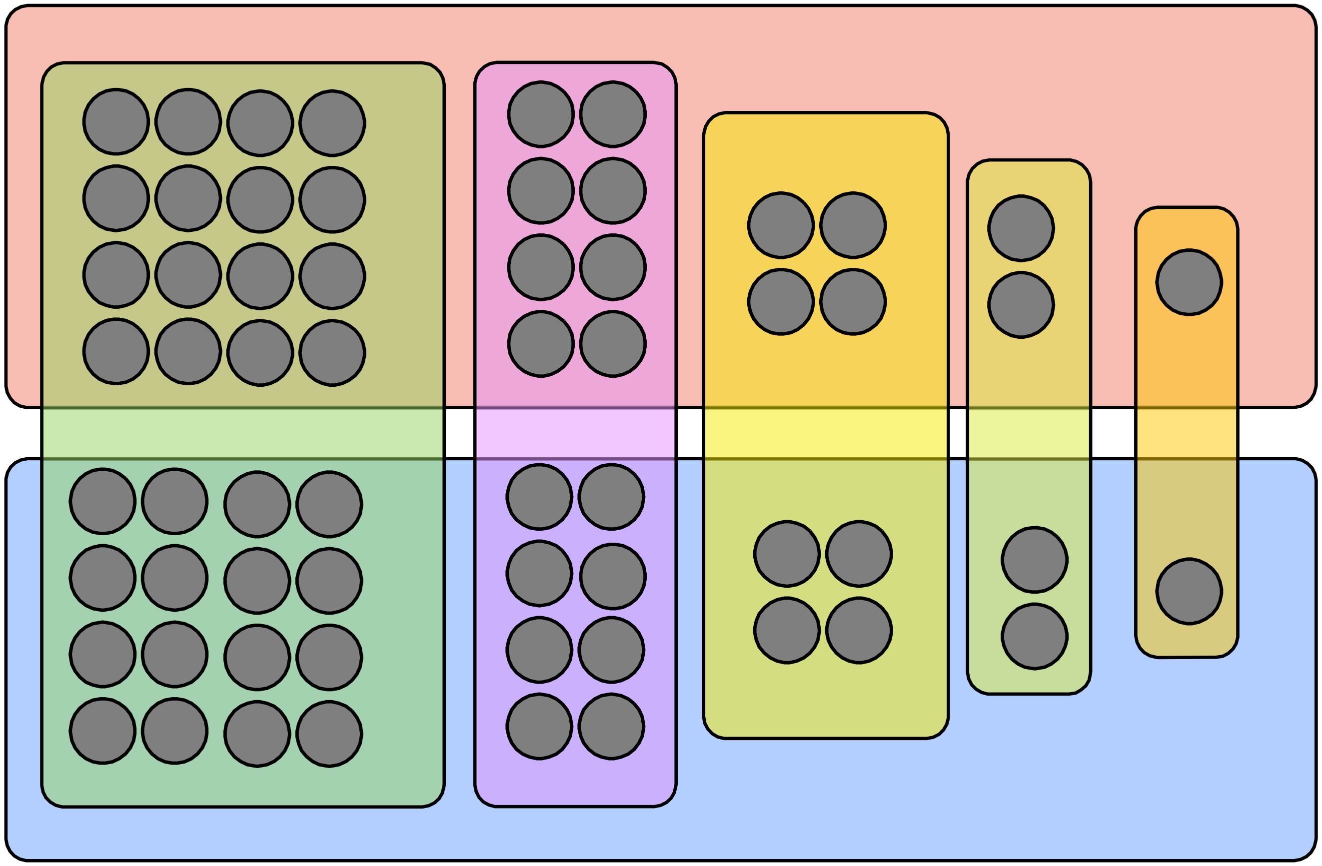
Greedy Set Cover: Repeatedly choose the set that covers the most # of new elements



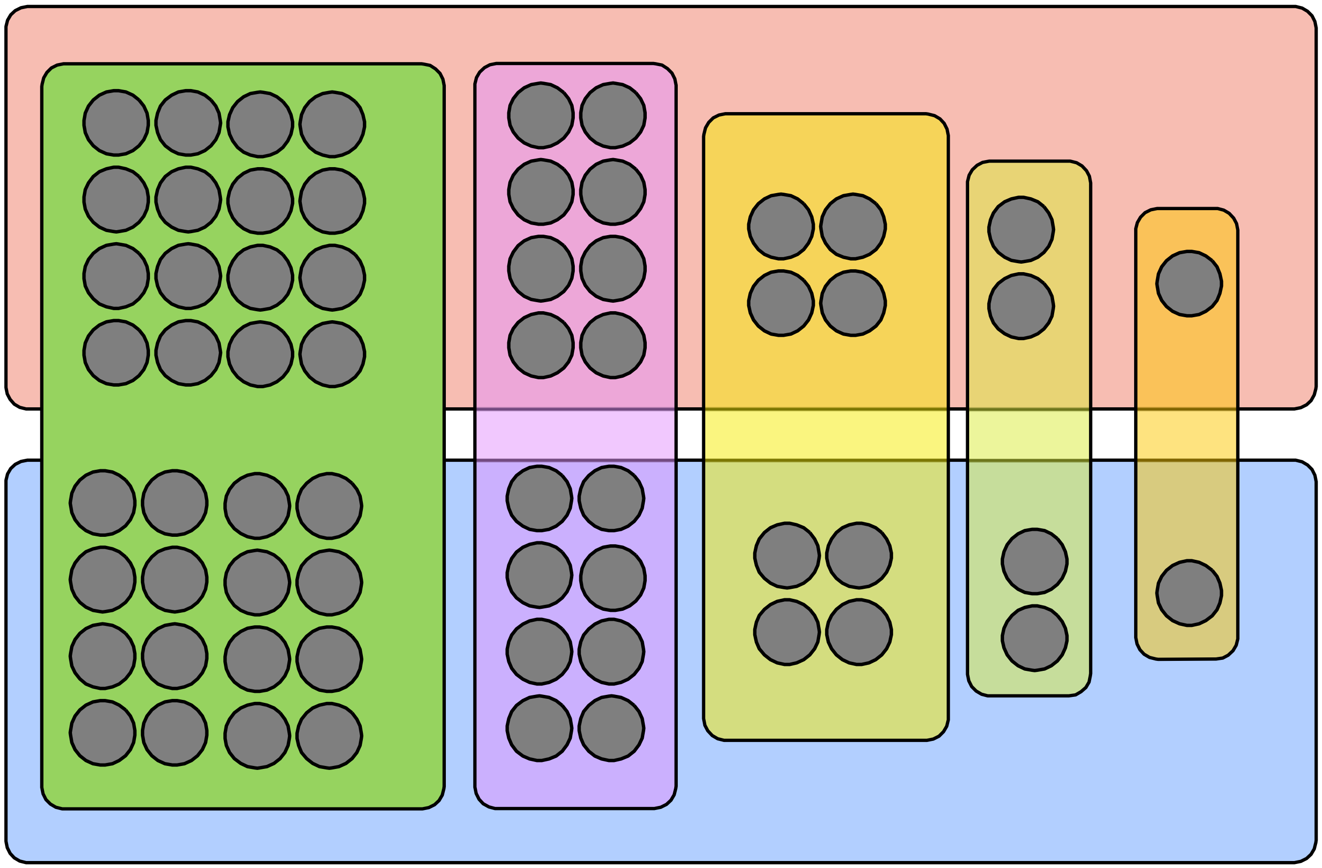
Find smallest collection of sets containing every point

Theorem: Greedy finds best cover up to a factor of $\ln n$.

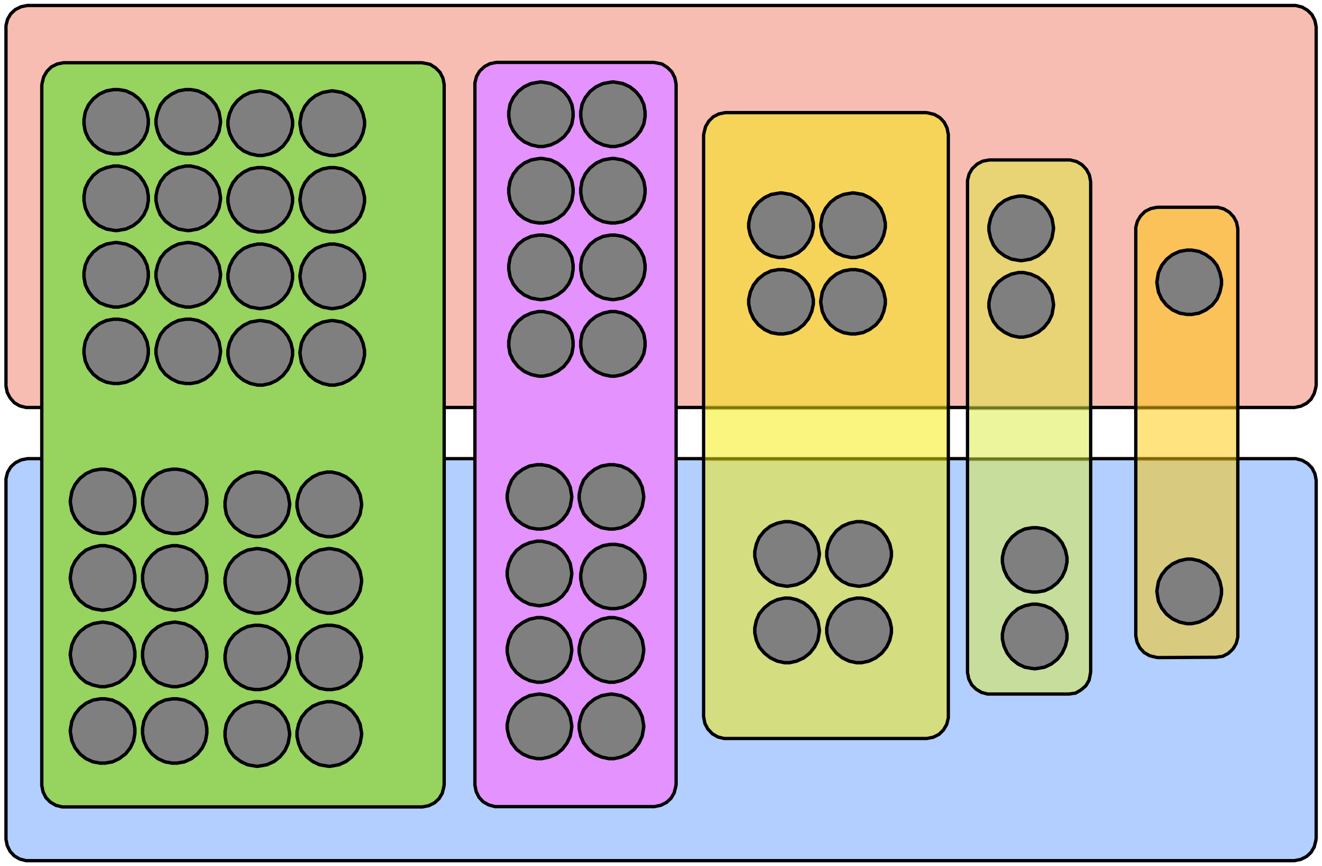
Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered



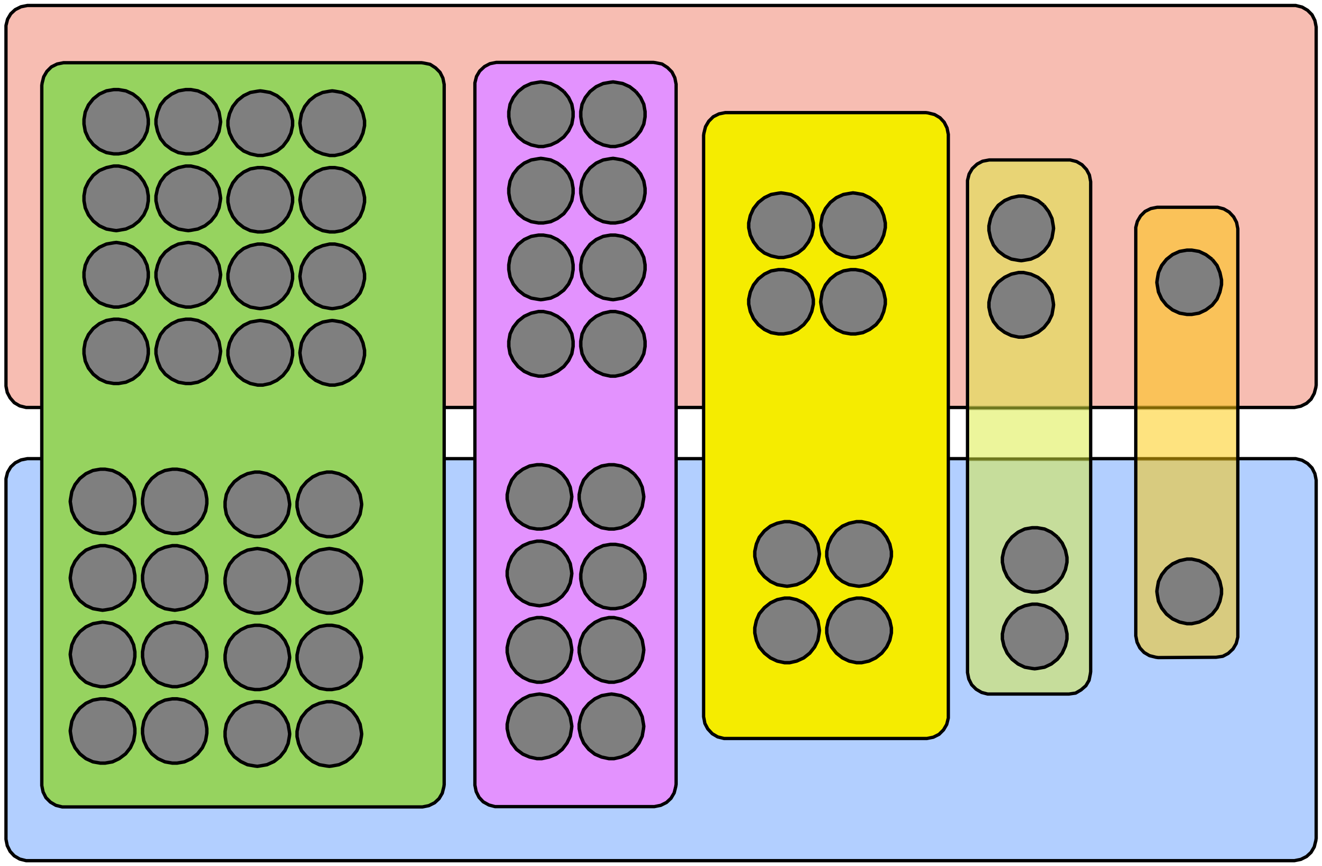
Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered



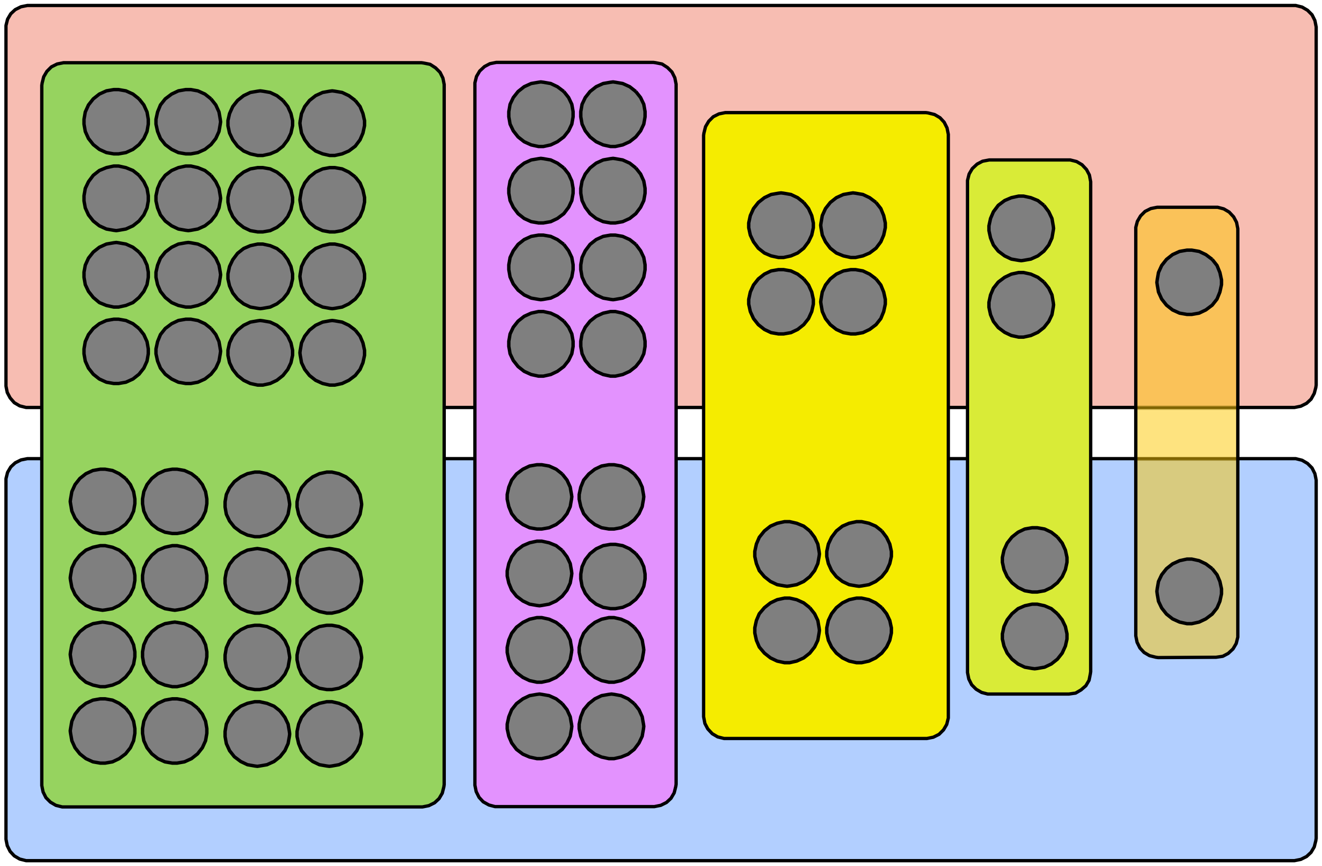
Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered



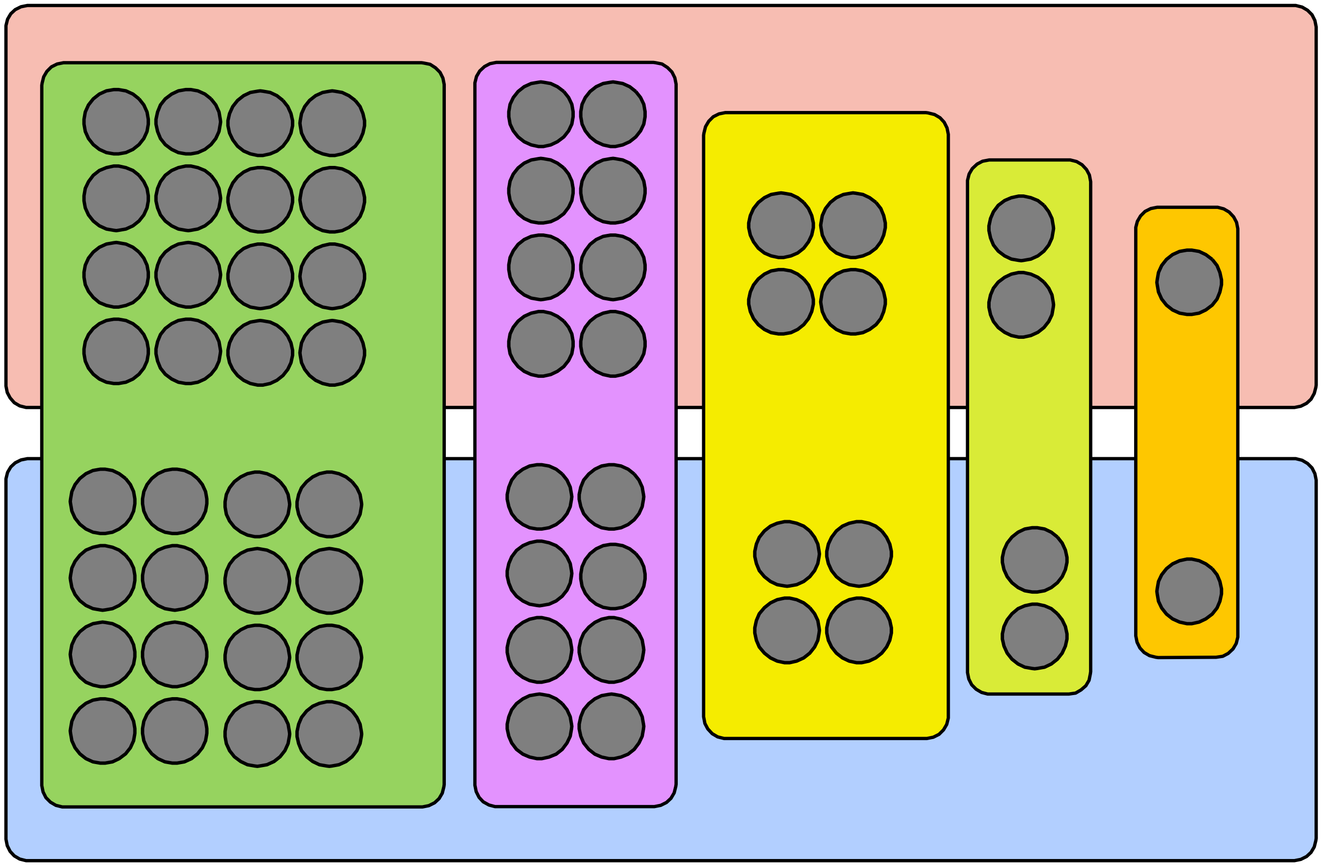
Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered



Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered



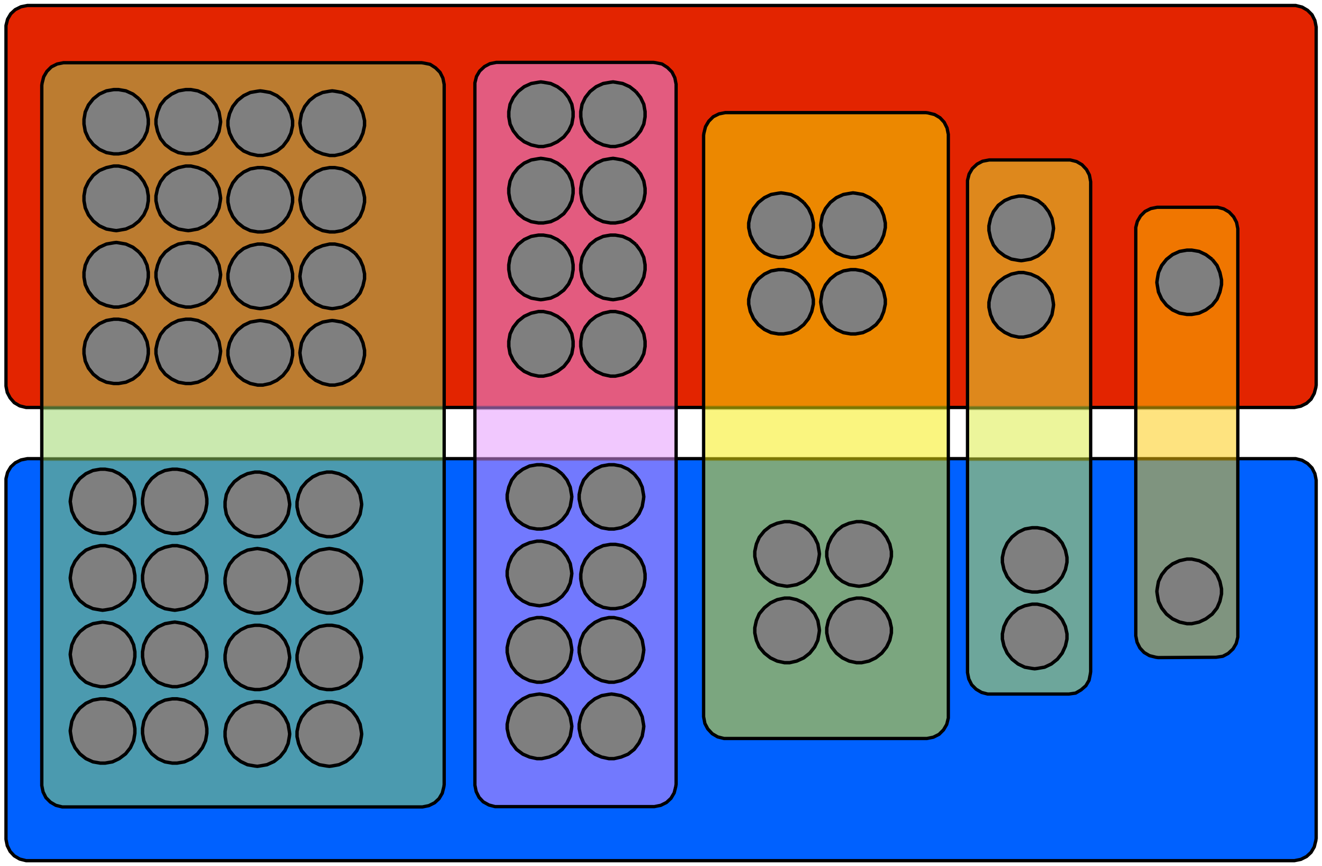
Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered



Greedy solution: **5** sets

Greedy solution: $\sim \log_2 n$ sets

Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered



Optimal:
2 sets

Greedy Approximation to Set-Cover

Theorem: If there is a set cover of size k then the greedy set cover has size $\leq k \ln n$.

Proof: Suppose that there is a set cover of size k.

$n - n/k$

$n \cdot (1 - 1/k)$

At each step all elements remaining are covered by these k sets.

So always a set available covering $\geq 1/k$ fraction of remaining elts.

So # of uncovered elts after i sets $\leq \left(1 - \frac{1}{k}\right) \times$ (# uncovered after $i - 1$ sets).

Total after t sets $\leq n\left(1 - \frac{1}{k}\right)^t < n \cdot e^{-t/k} = 1$ for $t = k \ln n$. ■

$(e^{-1/k})^t$

$\boxed{1 - x < e^{-x} \text{ for } x > 0}$