CSE 421

Approximation Algorithms:
Set Cover

(substituting for Yin Tat Lee)

Approximation Algorithms

How to deal with NP-complete problems

Many fundamental problems in computer science NP-complete: SAT, Independent Set, Travelling Salesman Problem, Vertex Cover, Set Cover, Graph Colouring, …

How to deal with NP-complete problems

Many fundamental problems in computer science NP-complete: SAT, Independent Set, Travelling Salesman Problem, Vertex Cover, Set Cover, Graph Colouring, …

Cannot find optimum solutions in polynomial time.

How to deal with NP-complete problems

Many fundamental problems in computer science NP-complete: SAT, Independent Set, Travelling Salesman Problem, Vertex Cover, Set Cover, Graph Colouring, …

Cannot find optimum solutions in polynomial time. Instead:

- Find (in polynomial time) the optimum solution of special cases (e.g., random inputs)

How to deal with NP-complete problems

Many fundamental problems in computer science NP-complete: SAT, Independent Set, Travelling Salesman Problem, Vertex Cover, Set Cover, Graph Colouring, …

Cannot find optimum solutions in polynomial time. Instead:

- Find (in polynomial time) the optimum solution of special cases (e.g., random inputs)
- Find (in polynomial time) a solution guaranteed to be close to optimal

How to deal with NP-complete problems

Many fundamental problems in computer science NP-complete: SAT, Independent Set, Travelling Salesman Problem, Vertex Cover, Set Cover, Graph Colouring, …

Cannot find optimum solutions in polynomial time. Instead:

- Find (in polynomial time) the optimum solution of special cases (e.g., random inputs)

- Find (in polynomial time) a solution guaranteed to be close to optimal: Approximation Algorithm

Approximation Algorithm

An algorithm has an approximation ratio $\alpha(n)$ if

$$\frac{\text{Cost of computed solution}}{\text{Cost of the optimum}} \leq \alpha(n)$$

for any input of length n. (worst case)

Approximation Algorithm

An algorithm has an approximation ratio $\alpha(n)$ if

$$\frac{\text{Cost of computed solution}}{\text{Cost of the optimum}} \leq \alpha(n)$$

for any input of length n. (worst case)

Goal: For every NP-hard problem, find a polynomial-time approximation algorithm with the best possible approximation ratio.

Approximation Algorithm

An algorithm has an approximation ratio $\alpha(n)$ if

$$\frac{\text{Cost of computed solution}}{\text{Cost of the optimum}} \leq \alpha(n)$$

for any input of length n. (worst case)

Goal: For every NP-hard problem, find a polynomial-time approximation algorithm with the best possible approximation ratio.

Getting a guarantee on the approximation ratio is inherently tricky!

Approximation Algorithm

An algorithm has an approximation ratio $\alpha(n)$ if

$$\frac{\text{Cost of computed solution}}{\text{Cost of the optimum}} \leq \alpha(n)$$

for any input of length n. (worst case)

Goal: For every NP-hard problem, find a polynomial-time approximation algorithm with the best possible approximation ratio.

Getting a guarantee on the approximation ratio is inherently tricky!

Approximation Algorithm

An algorithm has an approximation ratio $\alpha(n)$ if

$$\frac{\text{Cost of computed solution}}{\text{Cost of the optimum}} \leq \alpha(n)$$

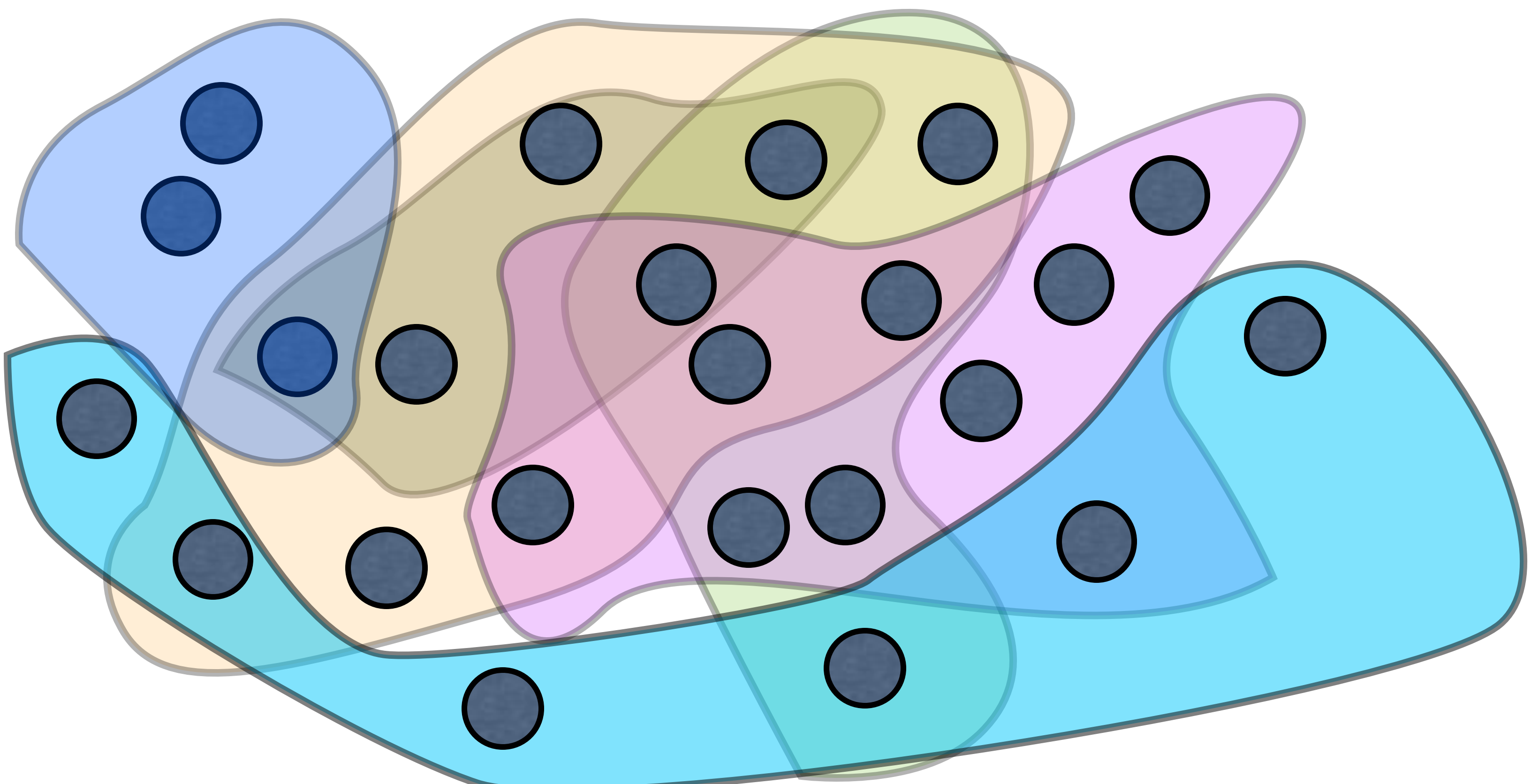
for any input of length n. (worst case)

Goal: For every NP-hard problem, find a polynomial-time approximation algorithm with the best possible approximation ratio.

Getting a guarantee on the approximation ratio is inherently tricky! Many methods to do so: we'll see one example today.

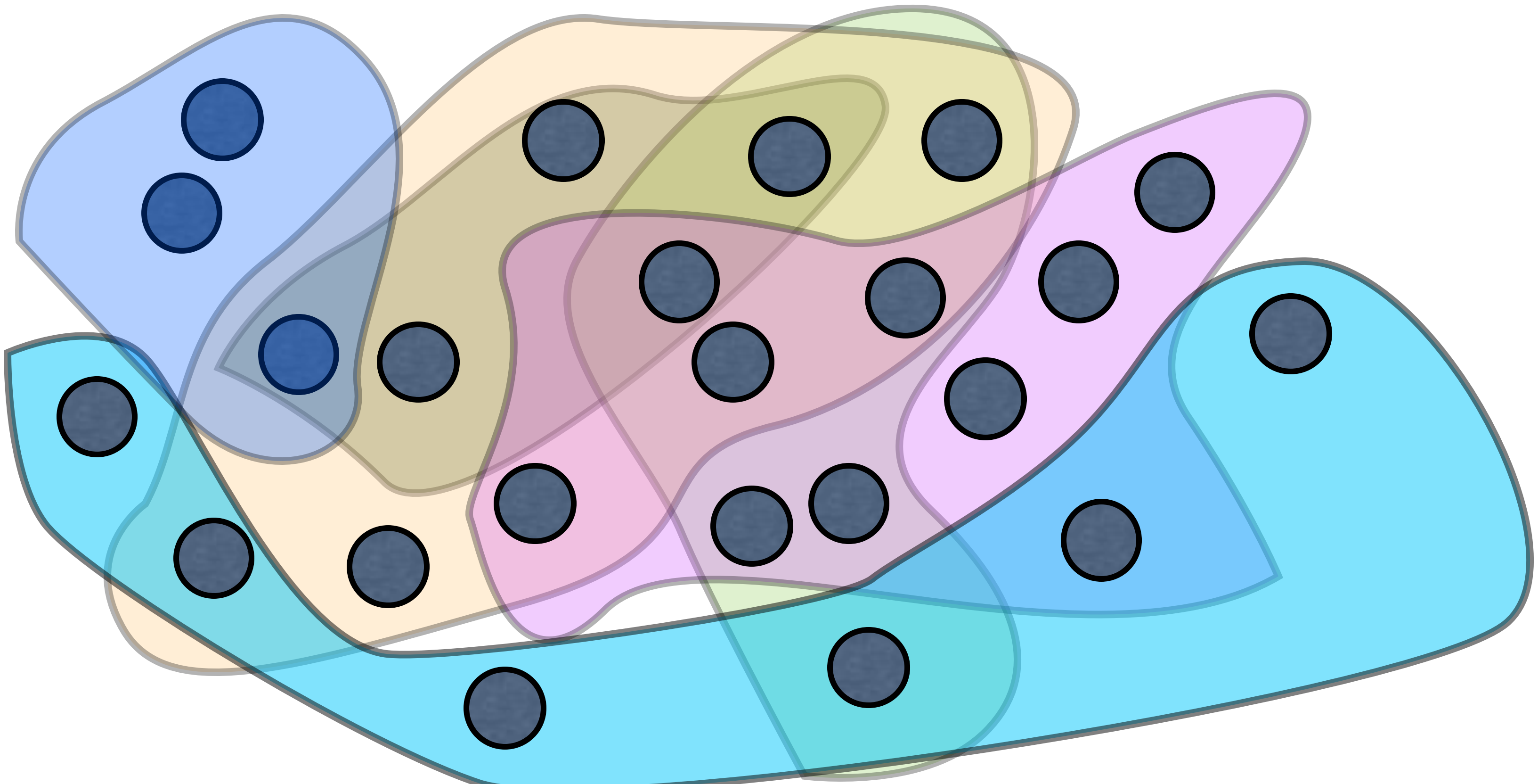
The Set Cover Problem

Problem Statement: Given a set U of n elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of U, find the smallest collection of these sets whose union is U.



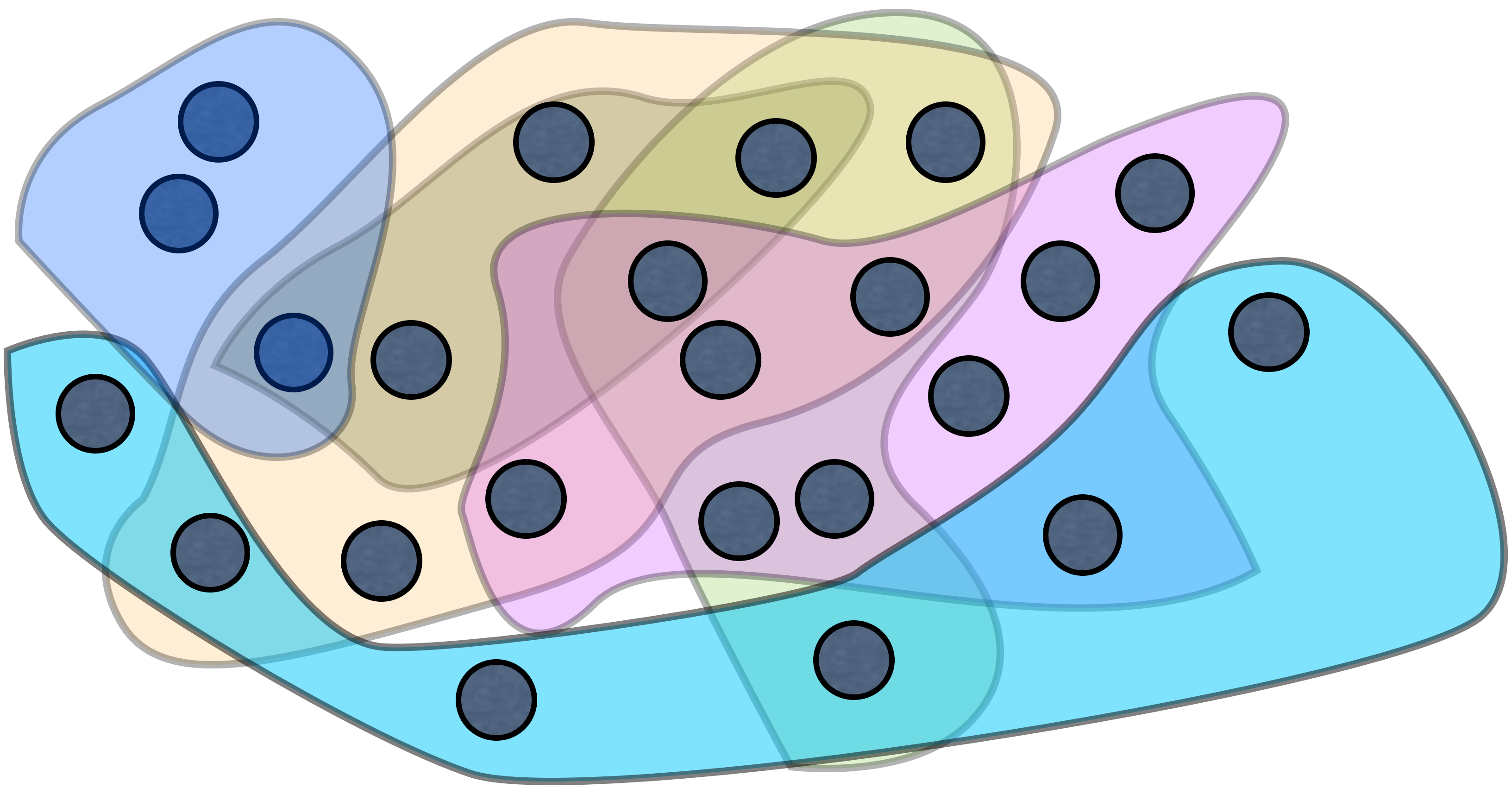
The Set Cover Problem

Problem Statement: Given a set U of n elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of U, find the smallest collection of these sets whose union is U.



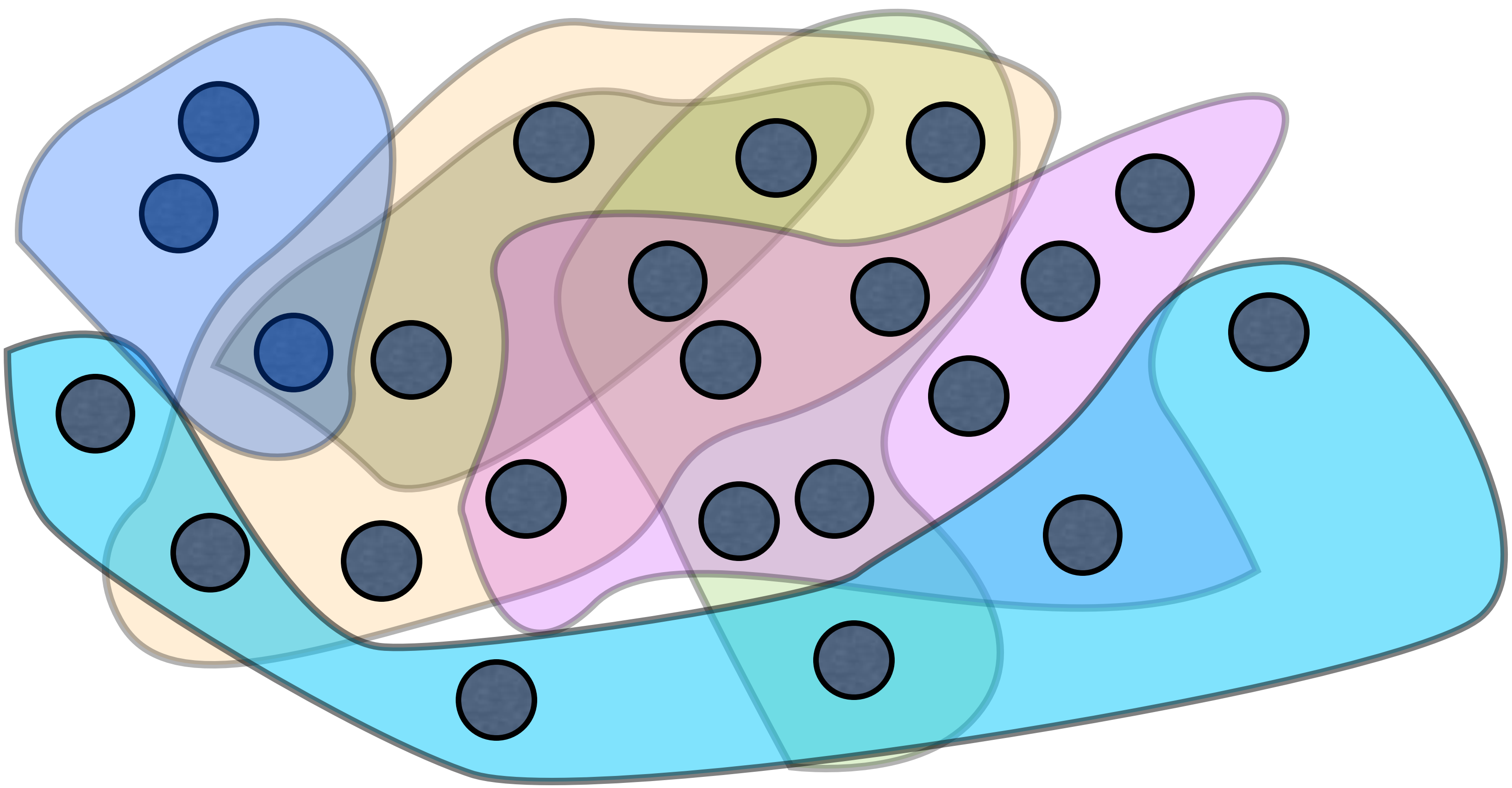
The Set Cover Problem

Applications: Company wants to hire people such that all required skills covered;



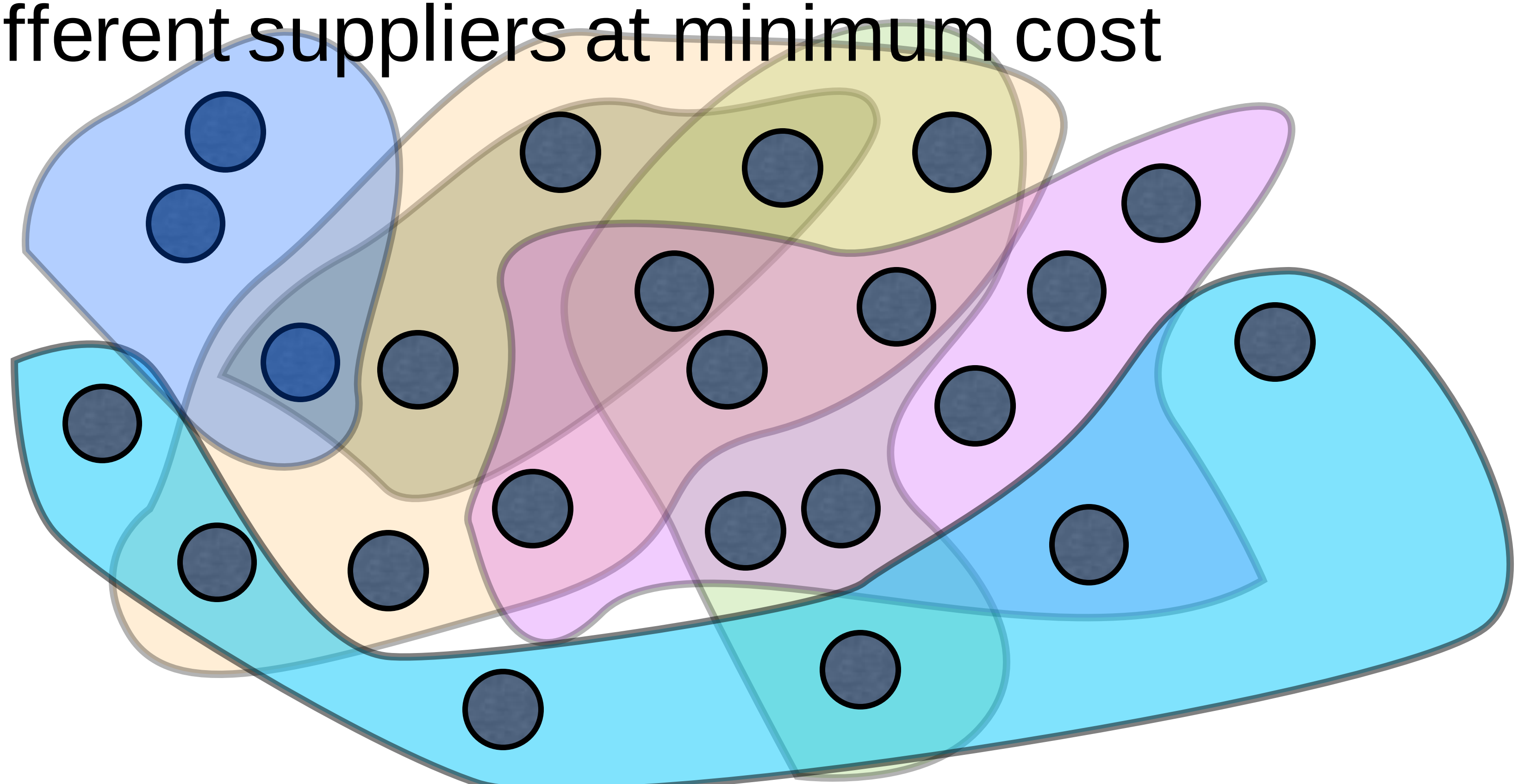
The Set Cover Problem

Applications: Company wants to hire people such that all required skills covered; "Fuzz" testing in software development;



The Set Cover Problem

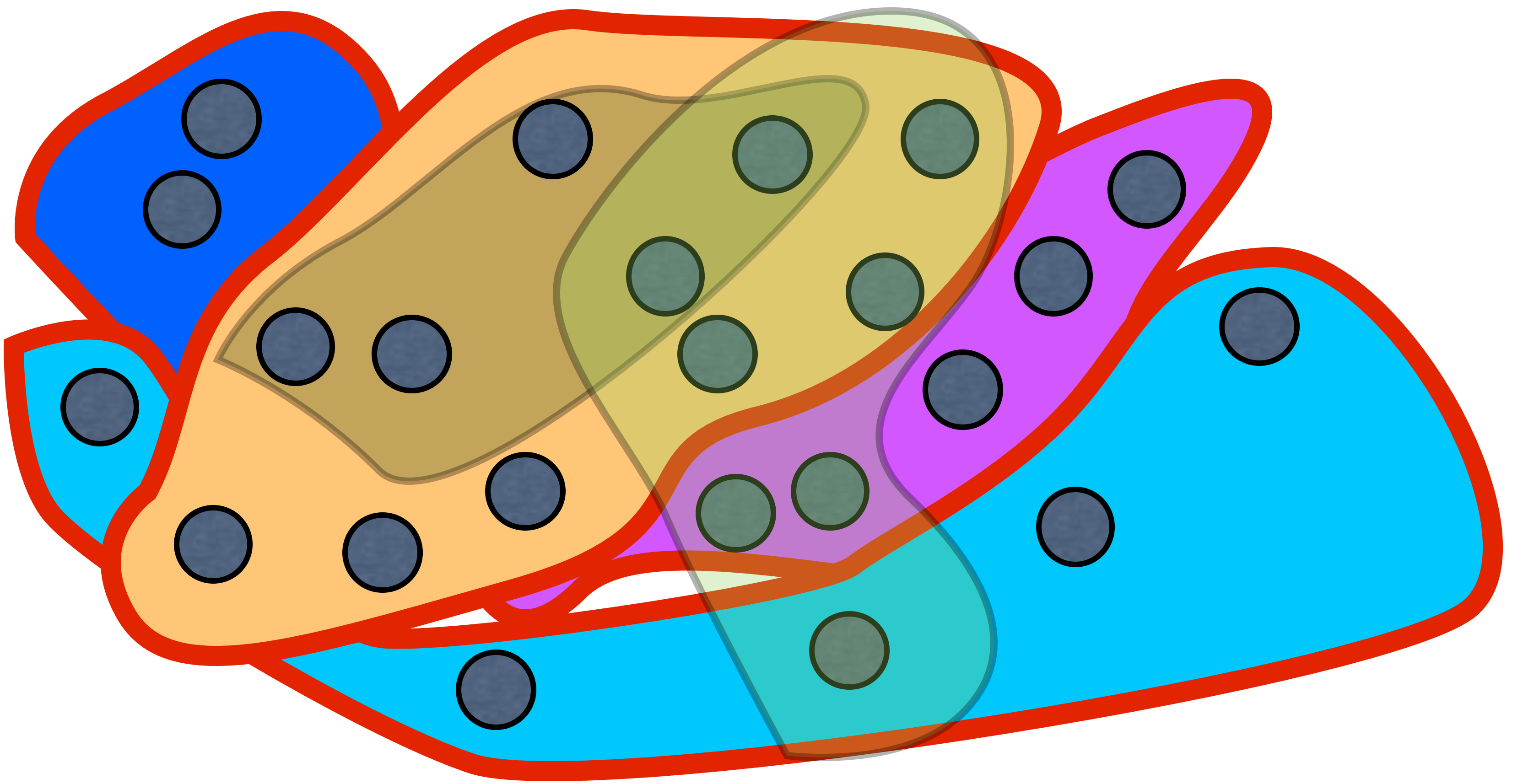
Applications : Company wants to hire people such that all required skills covered; “Fuzz” testing in software development; Manufacturer wants to get all items from different suppliers at minimum cost



The Set Cover Problem

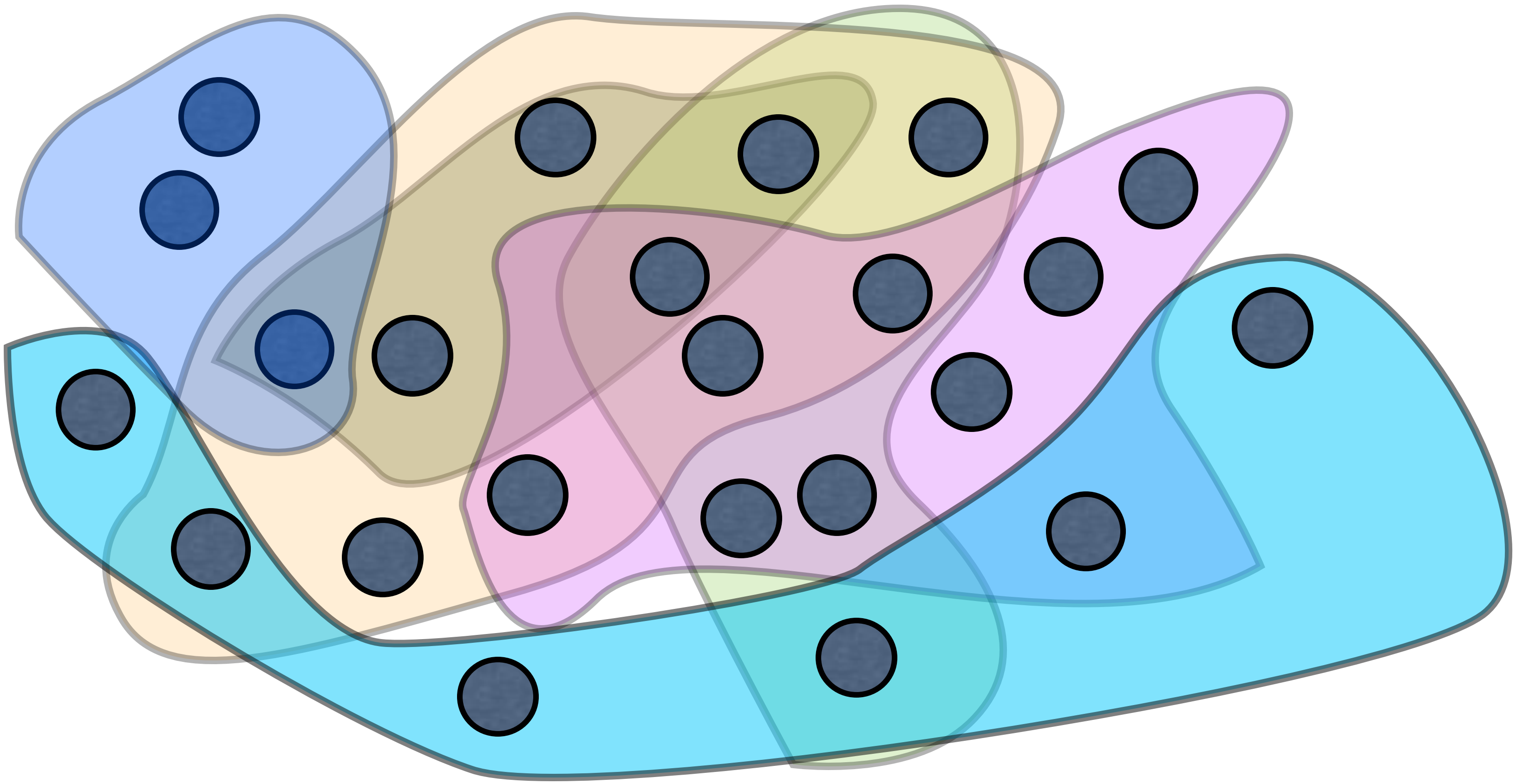
Example 1

Optimal set cover = 4



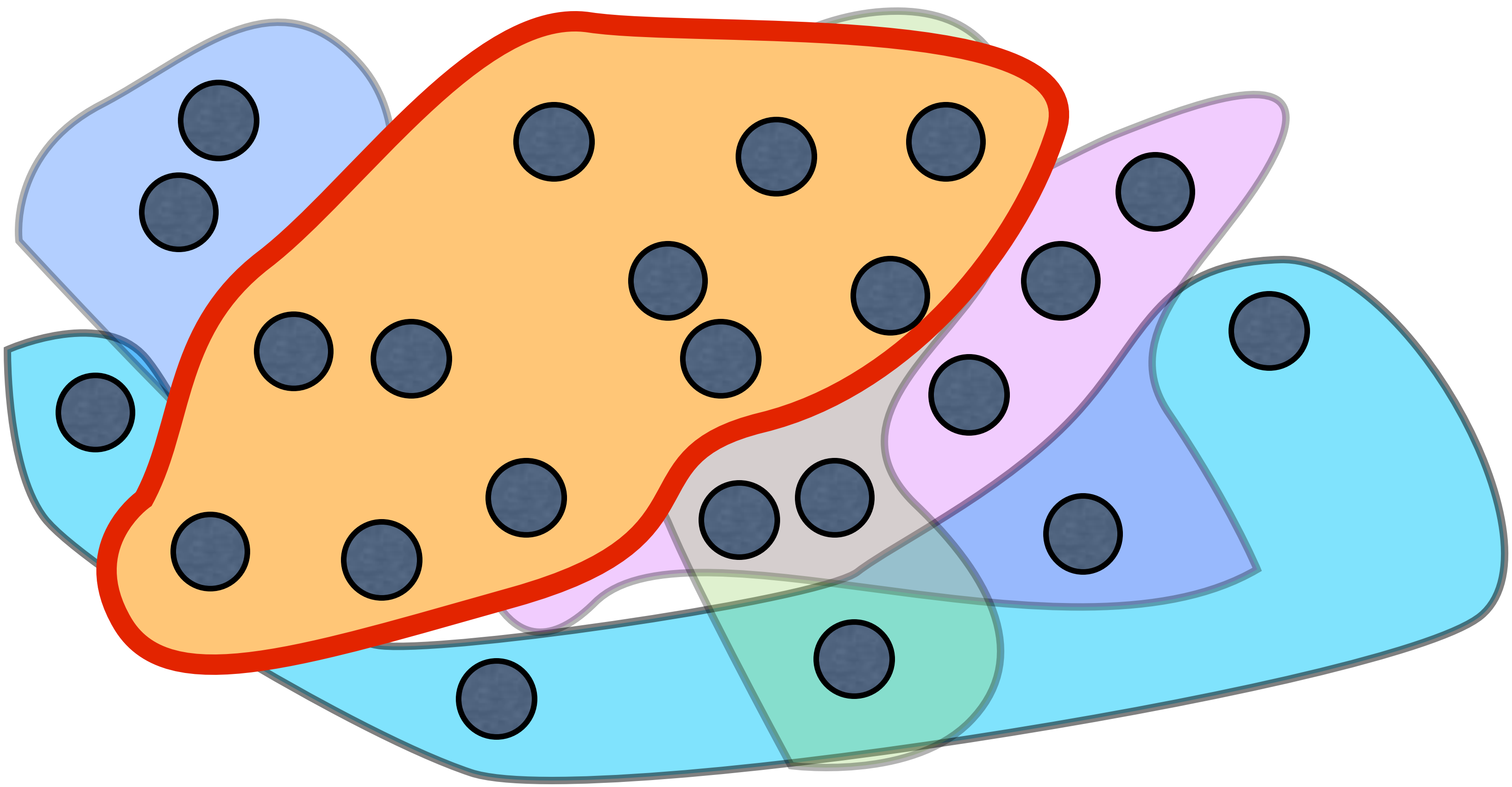
A Greedy Algorithm

Pick the set that maximizes the number of new elements covered



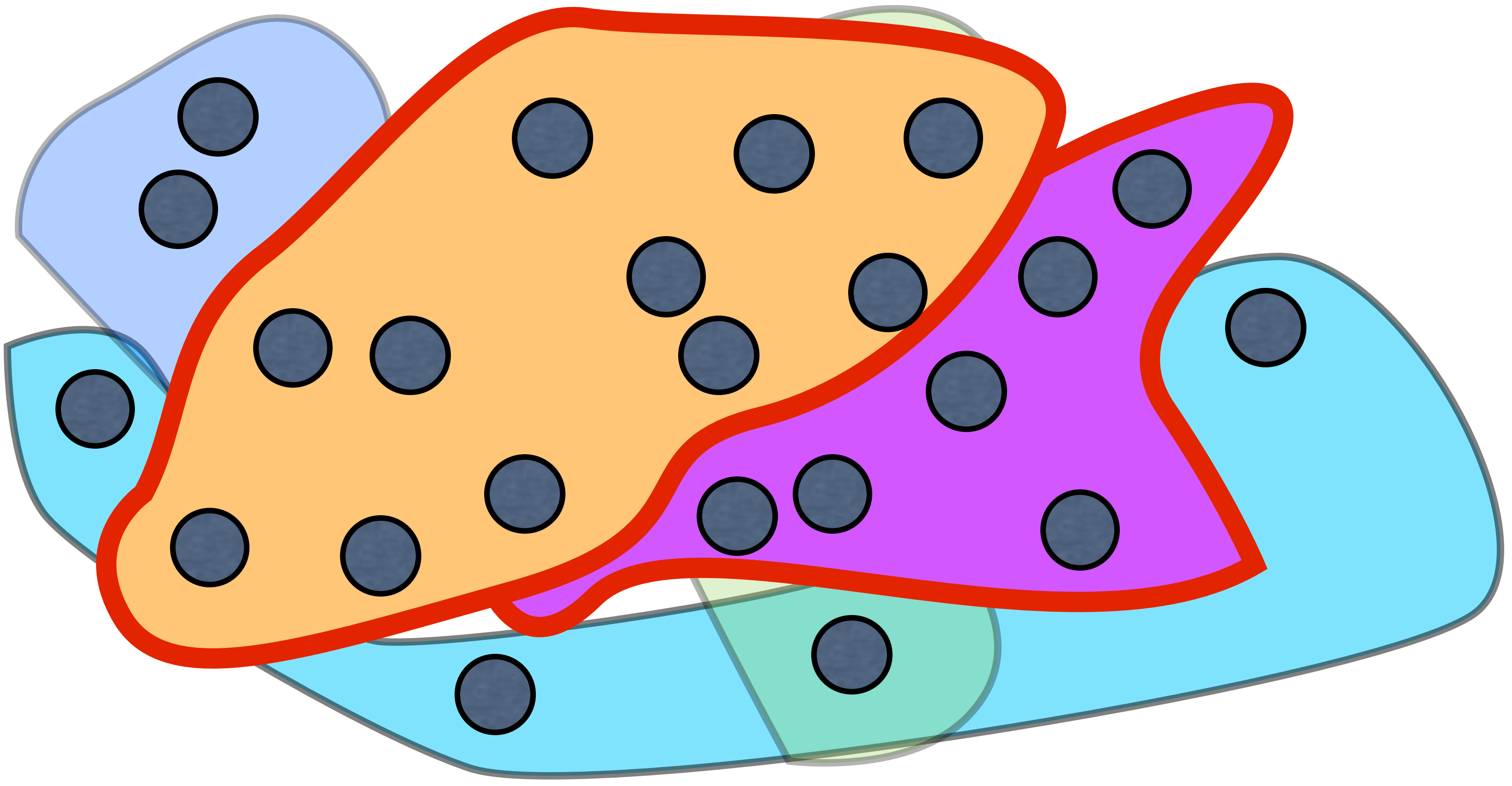
A Greedy Algorithm

Pick the set that maximizes the number of new elements covered



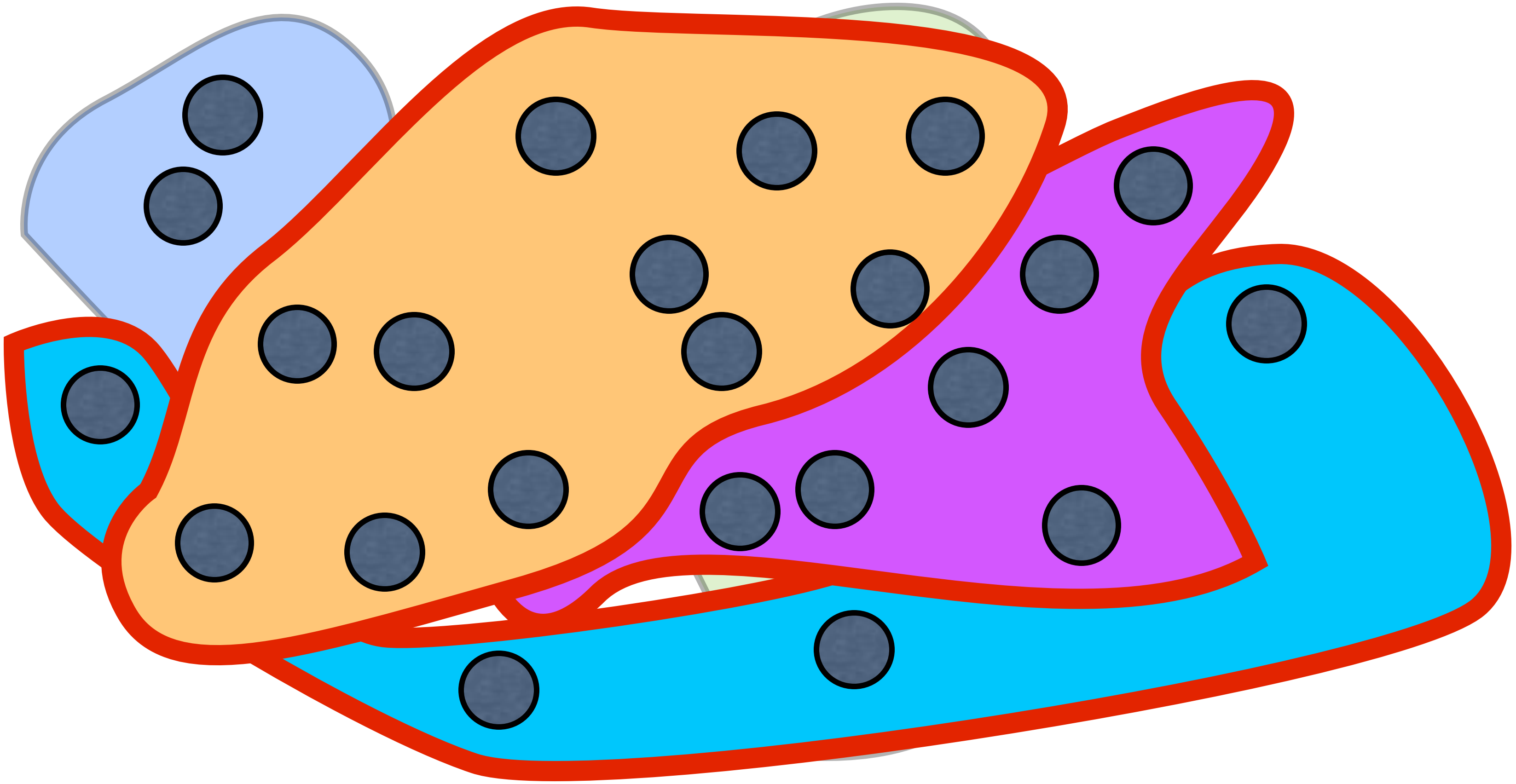
A Greedy Algorithm

Pick the set that maximizes the number of new elements covered



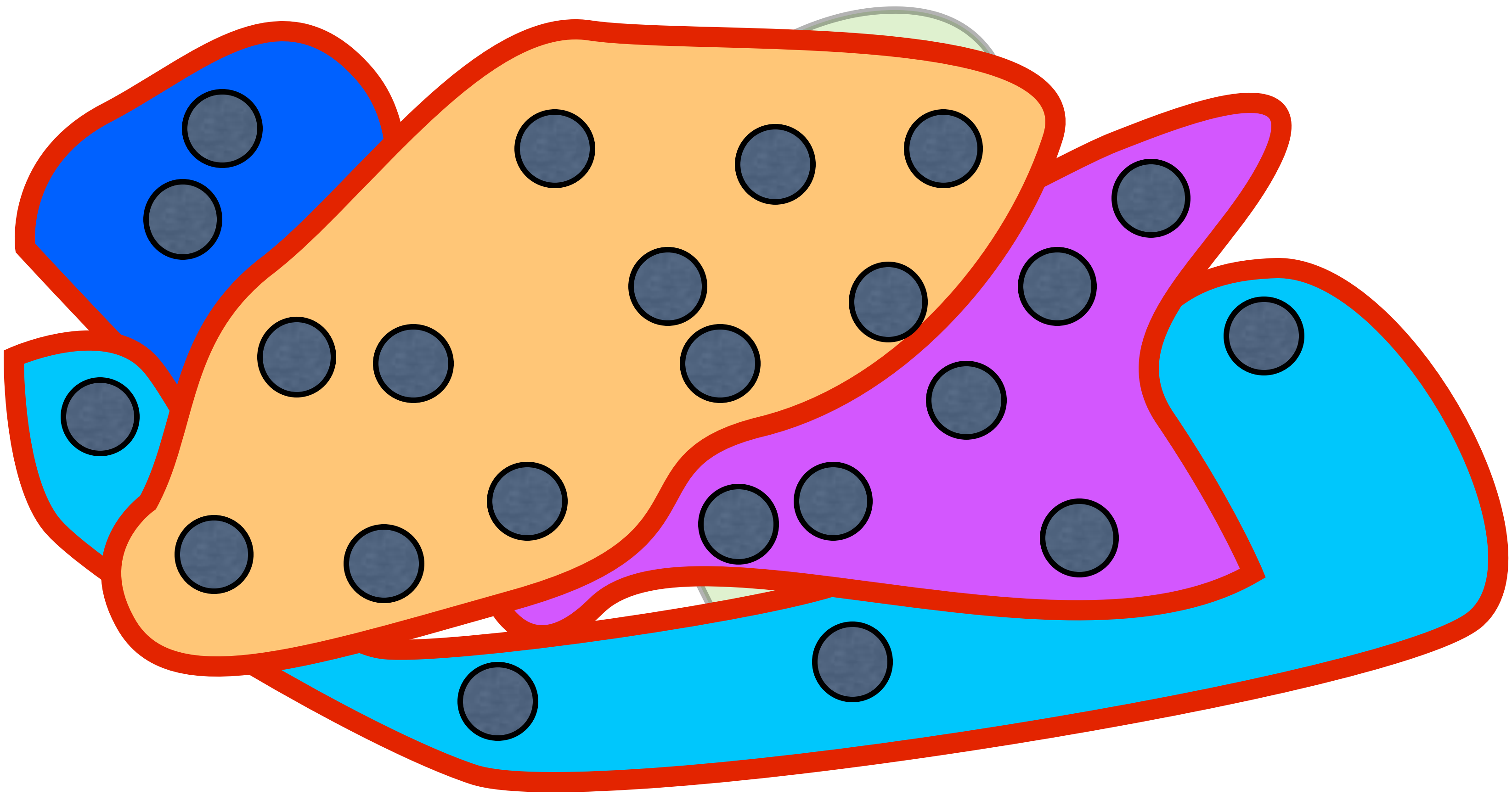
A Greedy Algorithm

Pick the set that maximizes the number of new elements covered



A Greedy Algorithm

Pick the set that maximizes the number of new elements covered



Greedy Algorithm

Example 2

$S_1 = \{1, 2, 3, 8, 9, 10\}$,

$S_2 = \{1, 2, 3, 4, 5\}$,

$S_3 = \{4, 5, 7\}$,

$S_4 = \{5, 6, 7\}$,

$S_5 = \{6, 7, 8, 9, 10\}$.

Greedy Algorithm

Example 2

$S_1 = \{1, 2, 3, 8, 9, 10\}$,

$S_2 = \{1, 2, 3, 4, 5\}$,

$S_3 = \{4, 5, 7\}$,

$S_4 = \{5, 6, 7\}$,

$S_5 = \{6, 7, 8, 9, 10\}$.

Greedy Algorithm

Example 2

$S_1 = \{1, 2, 3, 8, 9, 10\}$,

$S_2 = \{1, 2, 3, 4, 5\}$,

$S_3 = \{4, 5, 7\}$,

$S_4 = \{5, 6, 7\}$,

$S_5 = \{6, 7, 8, 9, 10\}$.

Greedy Algorithm

Example 2

$S_1 = \{1, 2, 3, 8, 9, 10\}$,

$S_2 = \{1, 2, 3, 4, 5\}$,

$S_3 = \{4, 5, 7\}$,

$S_4 = \{5, 6, 7\}$,

$S_5 = \{6, 7, 8, 9, 10\}$.

Greedy Algorithm

Example 2

$S_1 = \{1, 2, 3, 8, 9, 10\}$,

$S_2 = \{1, 2, 3, 4, 5\}$,

$S_3 = \{4, 5, 7\}$,

$S_4 = \{5, 6, 7\}$,

$S_5 = \{6, 7, 8, 9, 10\}$.

Is this the optimal solution?

How well did Greedy do in this case?

Example 2

$S_1 = \{1, 2, 3, 8, 9, 10\}$,

$S_2 = \{1, 2, 3, 4, 5\}$,

$S_3 = \{4, 5, 7\}$,

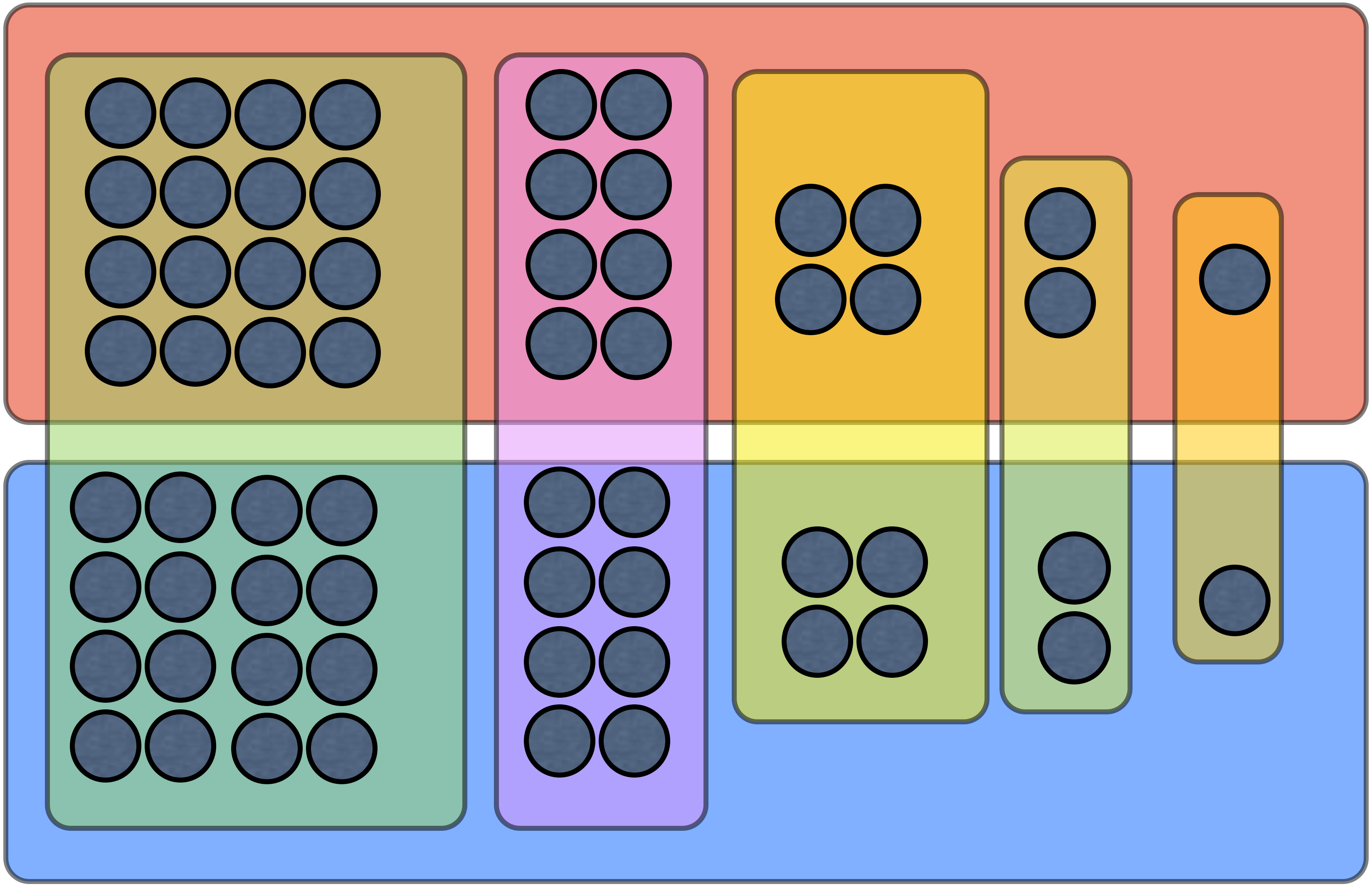
$S_4 = \{5, 6, 7\}$,

$S_5 = \{6, 7, 8, 9, 10\}$.

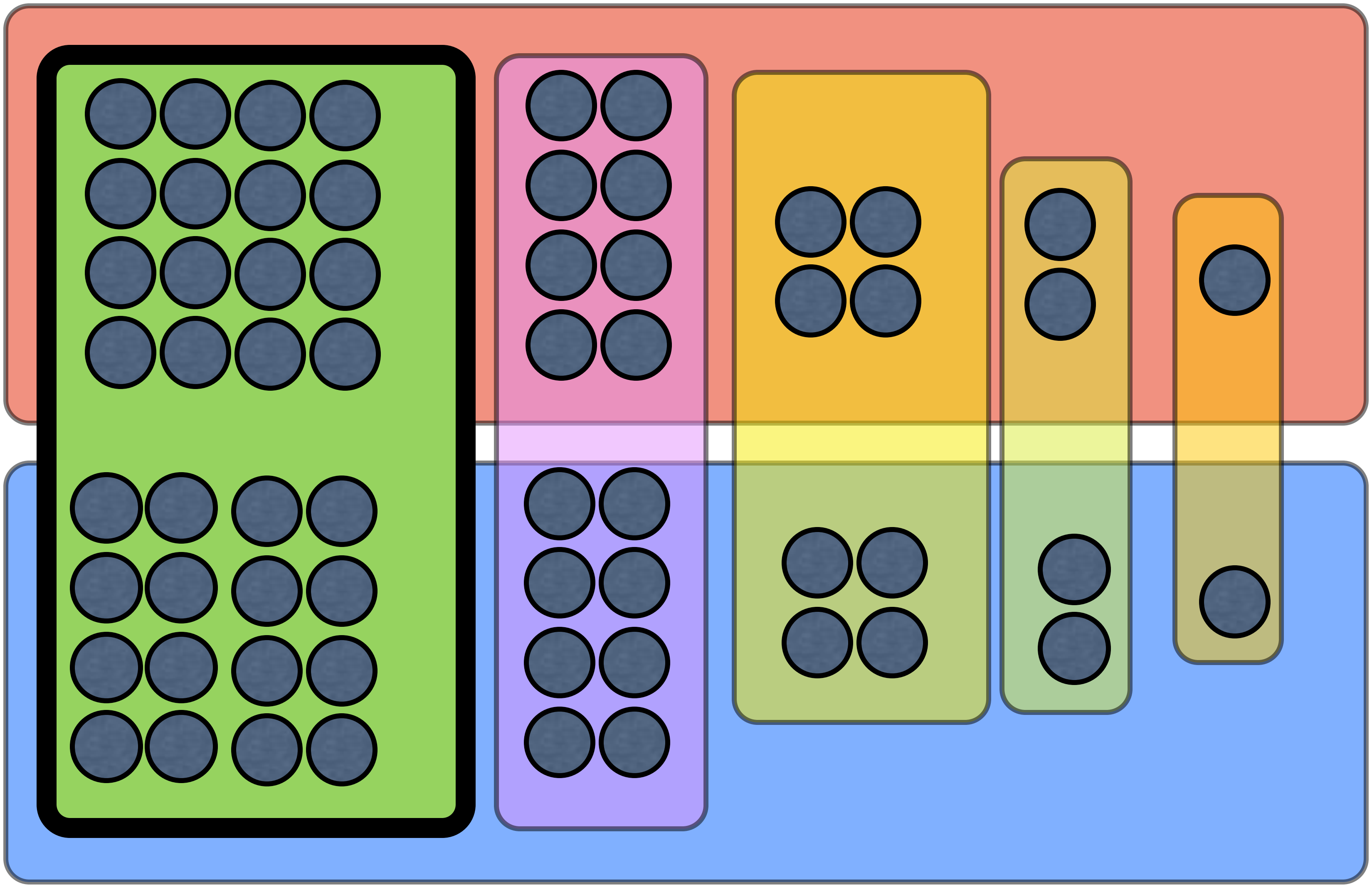
Greedy: 3

Optimum: 2

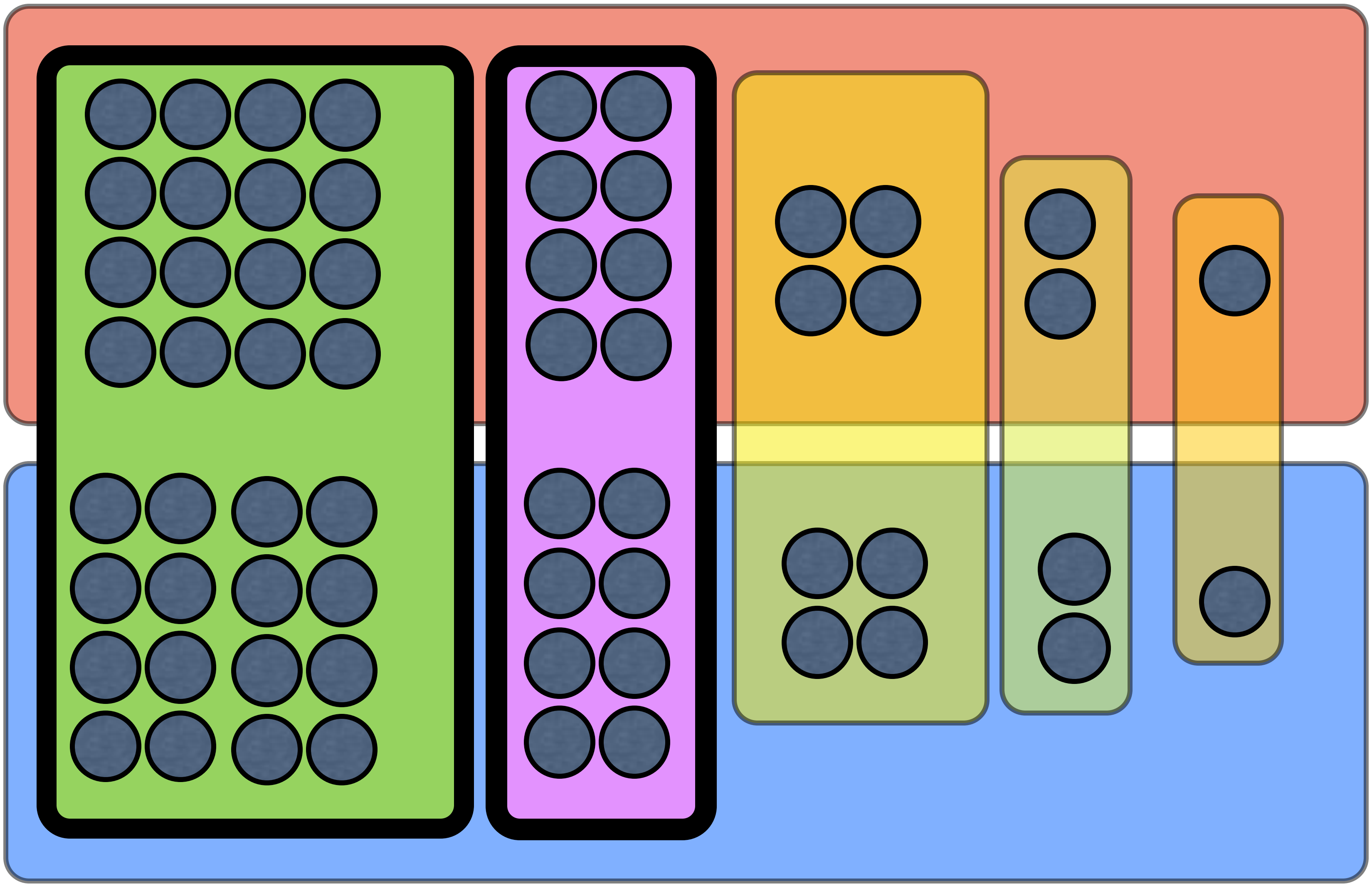
A Really Bad Example for Greedy



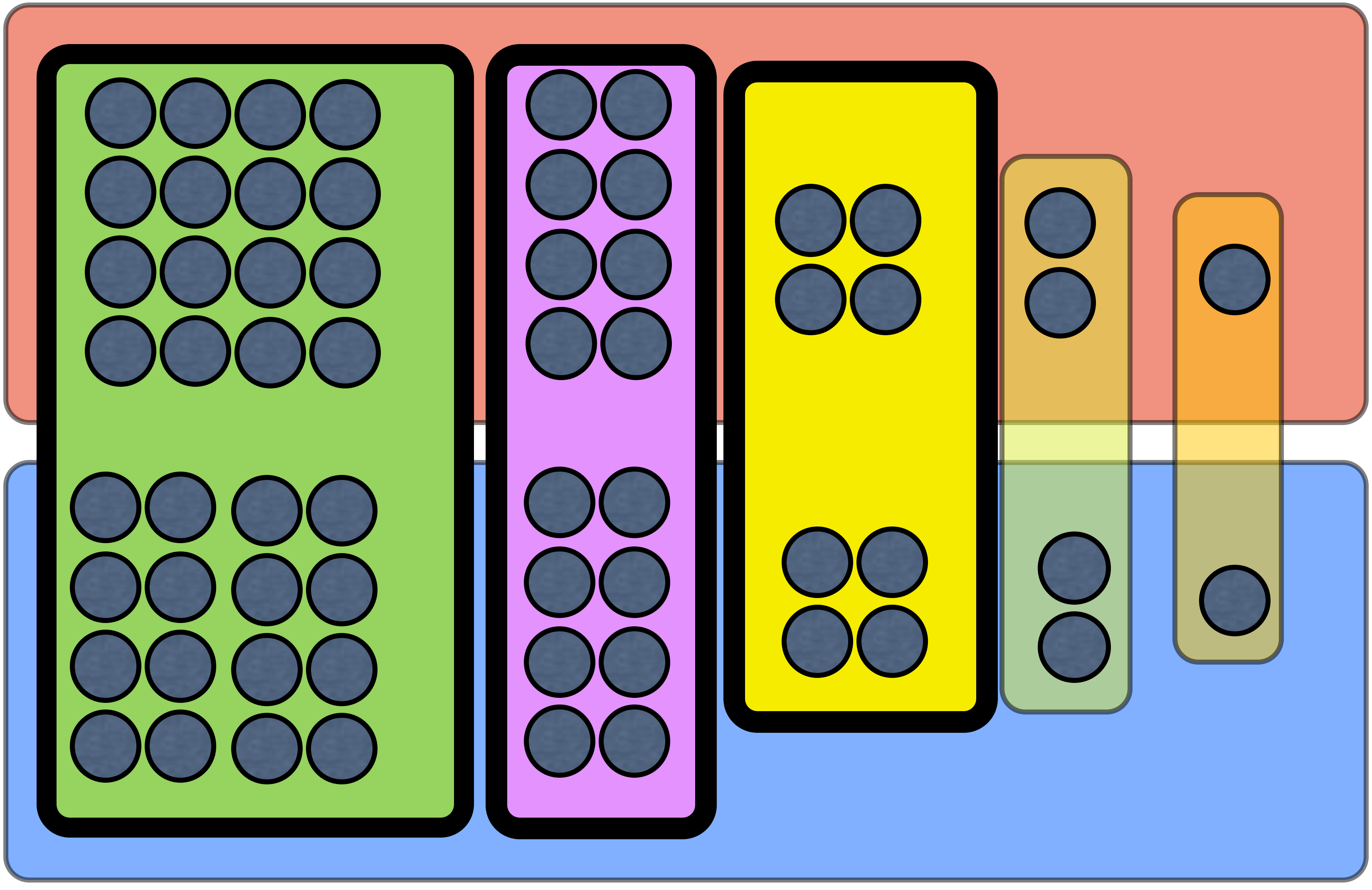
A Really Bad Example for Greedy



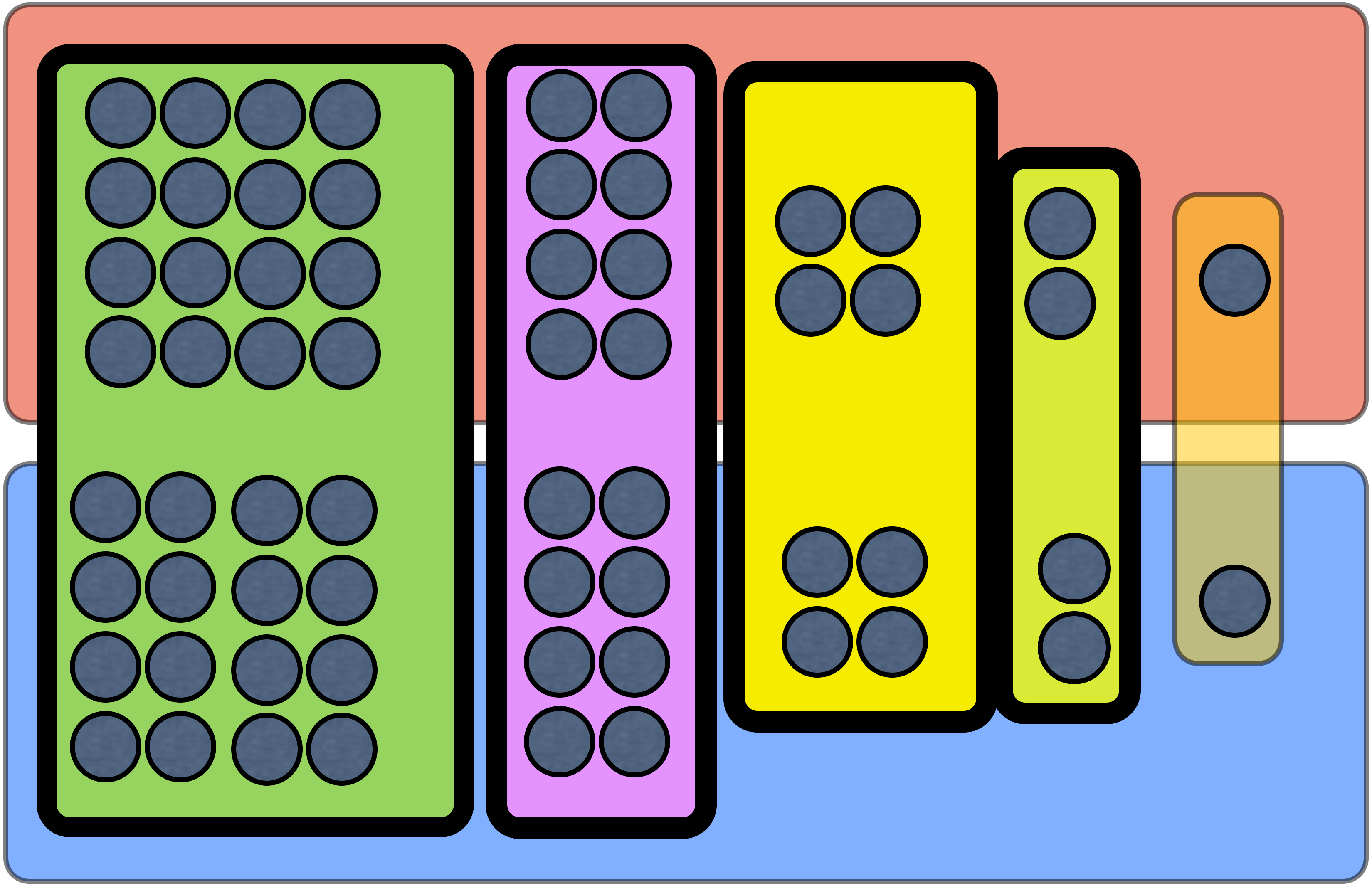
A Really Bad Example for Greedy



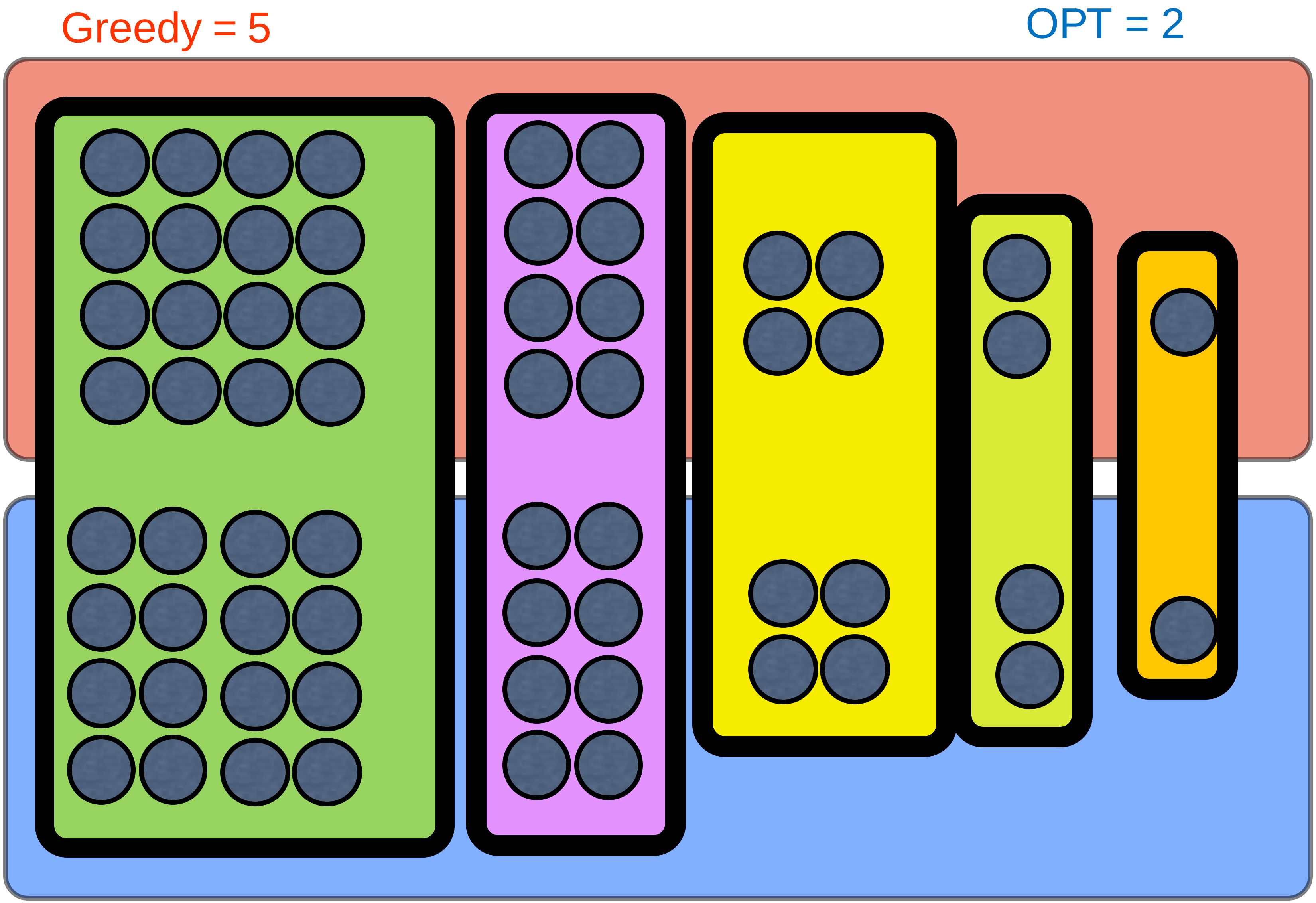
A Really Bad Example for Greedy



A Really Bad Example for Greedy



A Really Bad Example for Greedy



Greedy gives $O(\log(n))$ approximation

Thm: If $OPT = k$, greedy finds at most $k \ln(n)$ sets.

Greedy gives $O(\log(n))$ approximation

Thm: If $OPT = k$, greedy finds at most $k \ln(n)$ sets.

Pf: Since $OPT = k$, there exists a set that covers at least $1/k$ of the remaining elements.

Greedy gives $O(\log(n))$ approximation

Thm: If $OPT = k$, greedy finds at most $k \ln(n)$ sets.

Pf: Since $OPT = k$, there exists a set that covers at least $1/k$ of the remaining elements. If not, $OPT > k$.

Greedy gives $O(\log(n))$ approximation

Thm: If $OPT = k$, greedy finds at most $k \ln(n)$ sets.

Pf: Since $OPT = k$, there exists a set that covers at least $1/k$ of the remaining elements. If not, $OPT > k$.

Number of elements left after first set chosen:

$$n_1 \leq n - n/k = n(1 - 1/k)$$

Greedy gives $O(\log(n))$ approximation

Thm: If $OPT = k$, greedy finds at most $k \ln(n)$ sets.

Pf: Since $OPT = k$, there exists a set that covers at least $1/k$ of the remaining elements. If not, $OPT > k$.

Number of elements left after first set: $n_1 \leq n(1 - 1/k)$

Need to cover n_1 elements. There exists a set with at least $n_1/(k-1)$ elements

Greedy gives $O(\log(n))$ approximation

Thm: If $OPT = k$, greedy finds at most $k \ln(n)$ sets.

Pf: Since $OPT = k$, there exists a set that covers at least $1/k$ of the remaining elements. If not, $OPT > k$.

Number of elements left after first set: $n_1 \leq n(1 - 1/k)$

Need to cover n_1 elements. There exists a set with at least $n_1/(k-1)$ elements, otherwise $OPT > k$.

Greedy gives $O(\log(n))$ approximation

Thm: If $OPT = k$, greedy finds at most $k \ln(n)$ sets.

Pf: Since $OPT = k$, there exists a set that covers at least $1/k$ of the remaining elements. If not, $OPT > k$.

Number of elements left after first set: $n_1 \leq n(1 - 1/k)$

Need to cover n_1 elements. There exists a set with at least $n_1/(k-1)$ elements, otherwise $OPT > k$.

Number of elements left uncovered: $n_2 \leq n_1(1 - 1/(k-1))$

Greedy gives $O(\log(n))$ approximation

Thm: If $OPT = k$, greedy finds at most $k \ln(n)$ sets.

Pf: Since $OPT = k$, there exists a set that covers at least $1/k$ of the remaining elements. If not, $OPT > k$.

Number of elements left after first set: $n_1 \leq n(1 - 1/k)$

Need to cover n_1 elements. There exists a set with at least $n_1/(k-1)$ elements, otherwise $OPT > k$.

Number of elements left uncovered: $n_2 \leq n_1(1 - 1/(k-1))$

Greedy gives $O(\log(n))$ approximation

Thm: If $OPT = k$, greedy finds at most $k \ln(n)$ sets.

Pf: Since $OPT = k$, there exists a set that covers at least $1/k$ of the remaining elements. If not, $OPT > k$.

Number of elements left after first set: $n_1 \leq n(1 - 1/k)$

Need to cover n_1 elements. There exists a set with at least $n_1/(k-1)$ elements, otherwise $OPT > k$.

Number of elements left: $n_2 \leq n(1 - 1/k)(1 - 1/(k-1))$

Greedy gives $O(\log(n))$ approximation

Thm: If $OPT = k$, greedy finds at most $k \ln(n)$ sets.

Pf: Since $OPT = k$, there exists a set that covers at least $1/k$ of the remaining elements. If not, $OPT > k$.

Number of elements left after first set: $n_1 \leq n(1 - 1/k)$

Need to cover n_1 elements. There exists a set with at least $n_1/(k-1)$ elements, otherwise $OPT > k$.

Number of elements left: $n_2 \leq n(1 - 1/k)(1 - 1/k)$

Greedy gives $O(\log(n))$ approximation

Thm: If $OPT = k$, greedy finds at most $k \ln(n)$ sets.

Pf: Since $OPT = k$, there exists a set that covers at least $1/k$ of the remaining elements. If not, $OPT > k$.

Number of elements left after first set: $n_1 \leq n(1 - 1/k)$

Need to cover n_1 elements. There exists a set with at least $n_1/(k-1)$ elements, otherwise $OPT > k$.

Number of elements left: $n_2 \leq n(1 - 1/k)^2$

Greedy gives $O(\log(n))$ approximation

Thm: If $OPT = k$, greedy finds at most $k \ln(n)$ sets.

Pf: Since $OPT = k$, there exists a set that covers at least $1/k$ of the remaining elements. If not, $OPT > k$.

Number of elements left after first set: $n_1 \leq n(1 - 1/k)$

Need to cover n_1 elements. There exists a set with at least $n_1/(k-1)$ elements, otherwise $OPT > k$.

In general, number of elements left: $n_i \leq n(1 - 1/k)^i$

Greedy gives $O(\log(n))$ approximation

Thm: If $OPT = k$, greedy finds at most $k \ln(n)$ sets.

Pf: Since $OPT = k$, there exists a set that covers at least $1/k$ of the remaining elements. If not, $OPT > k$.

Number of elements left after first set: $n_1 \leq n(1 - 1/k)$

Need to cover n_1 elements. There exists a set with at least $n_1/(k-1)$ elements, otherwise $OPT > k$.

Useful upper bound: for all real x, we have $1 + x \leq e^x$. (Lots of proofs exist for this; a simple one is using the Taylor expansion of e^x.)

Greedy gives $O(\log(n))$ approximation

Thm: If $OPT = k$, greedy finds at most $k \ln(n)$ sets.

Pf: Since $OPT = k$, there exists a set that covers at least $1/k$ of the remaining elements. If not, $OPT > k$.

Number of elements left after first set: $n_1 \leq n(1 - 1/k)$

Need to cover n_1 elements. There exists a set with at least $n_1/(k-1)$ elements, otherwise $OPT > k$.

In general, number of elements left: $n_i \leq n(1 - 1/k)^i \leq ne^{-\frac{i}{k}}$

Greedy gives $O(\log(n))$ approximation

Thm: If $OPT = k$, greedy finds at most $k \ln(n)$ sets.

Pf: Since $OPT = k$, there exists a set that covers at least $1/k$ of the remaining elements. If not, $OPT > k$.

Number of elements left after first set: $n_1 \leq n(1 - 1/k)$

Need to cover n_1 elements. There exists a set with at least $n_1/(k-1)$ elements, otherwise $OPT > k$.

In general, number of elements left: $n_i \leq ne^{-\frac{i}{k}}$

So after $\mathrm{i} = k \ln n$ steps, # uncovered elements < 1.

Approximation Algorithm Summary

- The best known approximation algorithm for set cover is the greedy.
  - It is NP-Complete to obtain better than ln(n) approximation ratio for set cover.

Approximation Algorithm Summary

- The best known approximation algorithm for set cover is the greedy.
  - It is NP-Complete to obtain better than ln(n) approximation ratio for set cover.
- The best known approximation algorithm for vertex cover is the greedy.
  - It has been open for 40 years to obtain a polynomial time algorithm with approximation ratio better than 2

Approximation Algorithm Summary

- The best known approximation algorithm for set cover is the greedy.
  - It is NP-Complete to obtain better than ln(n) approximation ratio for set cover.
- The best known approximation algorithm for vertex cover is the greedy.
  - It has been open for 40 years to obtain a polynomial time algorithm with approximation ratio better than 2
- There is a long list of problems for which we do not know the best approximation algorithms! Very active area of research (including in our Theory group!)