CSE 421
Introduction to Algorithms

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

3SAT$\leq_P$Subset-Sum

Given a 3-CNF formula F with m clauses and n variables

- We will create an input for **Subset-Sum** with $2m + 2n$ numbers that are $m + n$ digits long.
- We will ensure that no matter how we sum them there won't be any carries so each digit in the target W will force a separate constraint.
- Instead of calling them $w_1, \ldots, w_{2n+2m}$ we will use mnemonic names:
  - Two numbers for each variable x_i
    - t_i and f_i (corresponding to x_i being true or x_i being false)
  - Two extra numbers for each clause C_j
    - a_j and b_j (two identical filler numbers to handle number of false literals in clause 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
$a_j =$	0	0	0	0	…	0	0	0	…	1	…	0
$b_j =$	0	0	0	0	…	0	0	0	…	1	…	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 $\mathbf{0}$

3SAT $\leq_P$ Subset-Sum

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

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

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

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

…

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

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.

				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

Clause part:
$\mathbf{1}$'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 $\mathbf{1}$'s.

The a's and b's add exactly 2 more possible $\mathbf{1}$'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

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

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

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

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

…

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

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

The a's and b's add exactly 2 more possible $\mathbf{1}$'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.

	1	2	3	4 (i)	...	n	1	2	3 (j)	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 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 $v_1, \ldots, v_n$ 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 $v_1, \ldots, v_n$ 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?

Hamiltonian-Cycle $\leq_P$ DecisionTSP

Define the reduction given $G = (V, E)$:

- Vertices $V = \{v_1, \dots, v_n\}$ become cities
- Define $d(v_i, v_j) = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 2 & \text{if not} \end{cases}$
- Distance $D = |V|$.

Claim: There is a Hamiltonian cycle in $G \Leftrightarrow$ there is a tour of length $|V|$

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 $2^{\tilde{\Theta}(n^{1/3})}$ 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 $2^{\tilde{\Theta}(n^{1/3})}$) 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

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

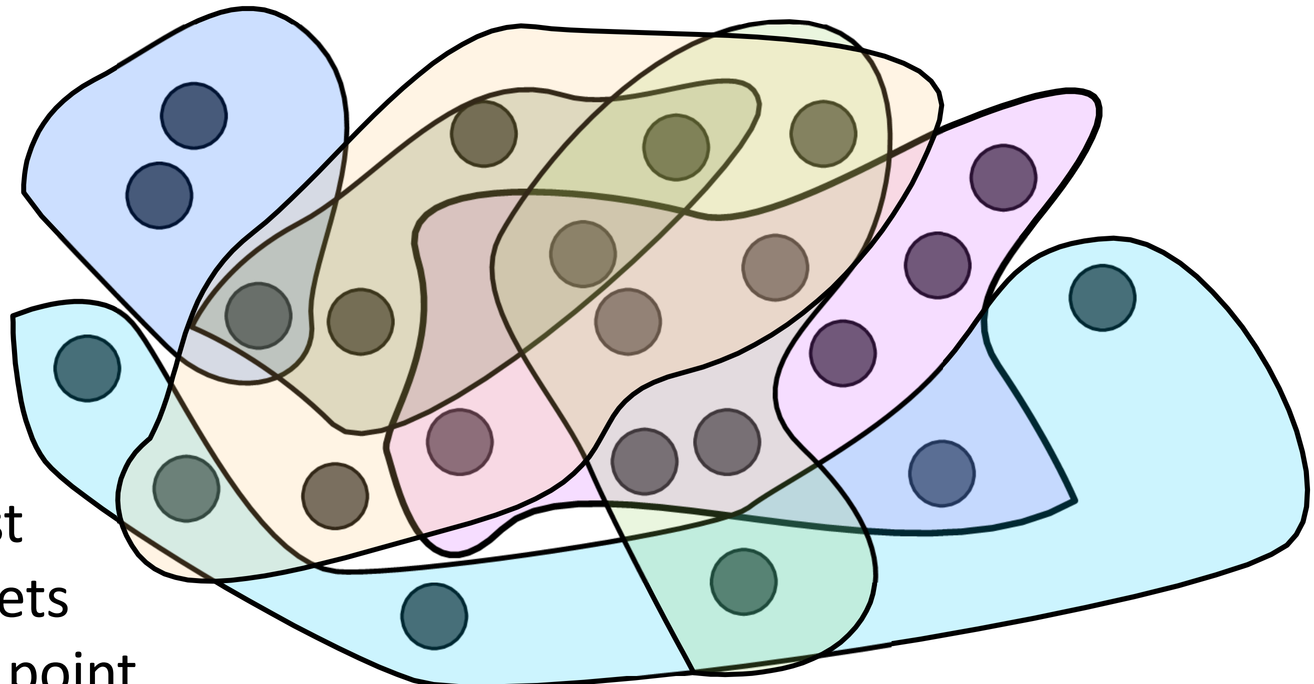
```
On input G = (V, E)
W ← ∅
E' ← E
while E' ≠ ∅
    select any e = (u, v) ∈ E'
    W ← W ∪ {u, v}
    E' ← E' \ {edges e ∈ E' that touch u or v}
```

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

Claim: At most a factor $\mathbf{2}$ larger than the optimal vertex-cover size.

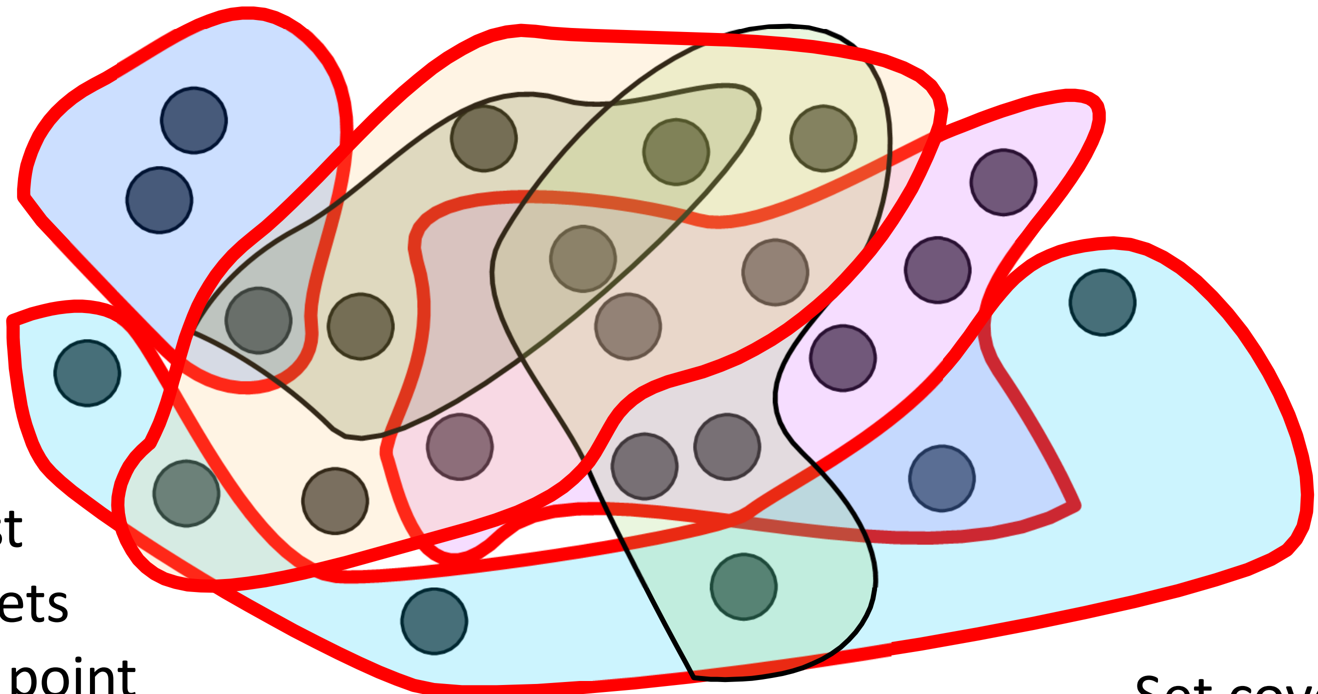
Proof: Edges selected don't share any vertices so any vertex-cover must choose at least one of u or v each time.

Set-Cover



Find smallest collection of sets containing every point

Set-Cover

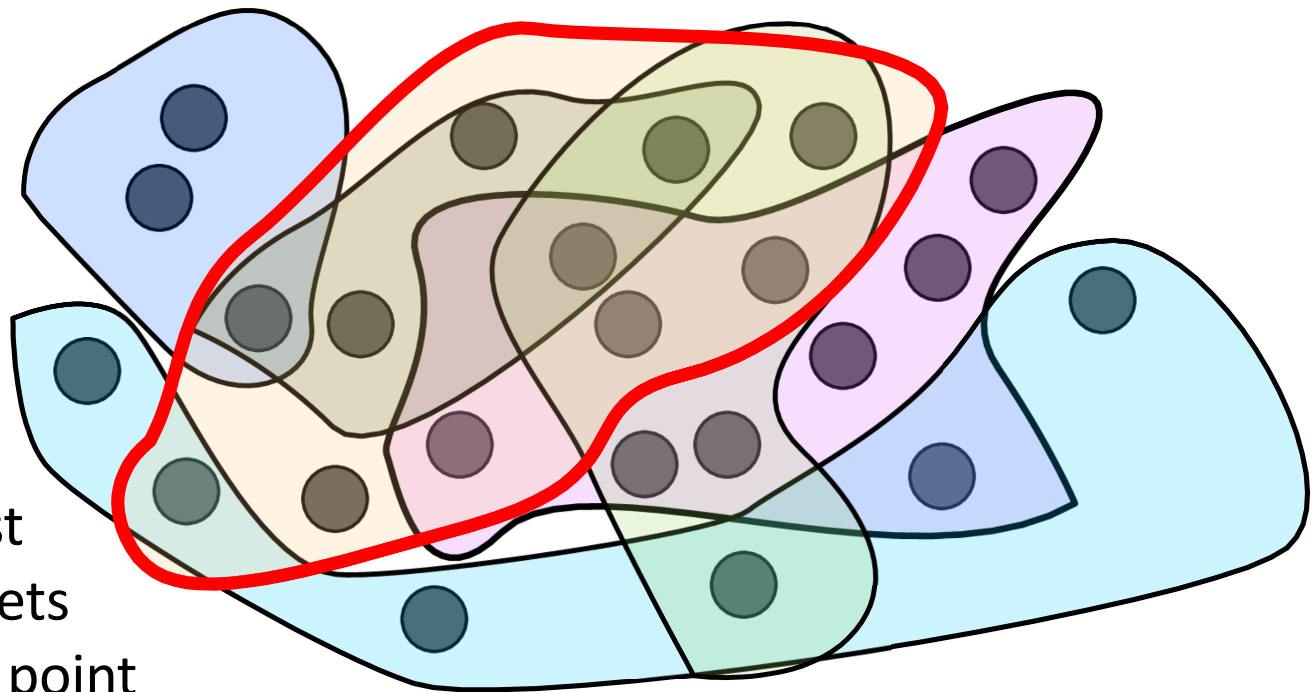


Find smallest collection of sets containing every point

Set cover size **4**

Set-Cover

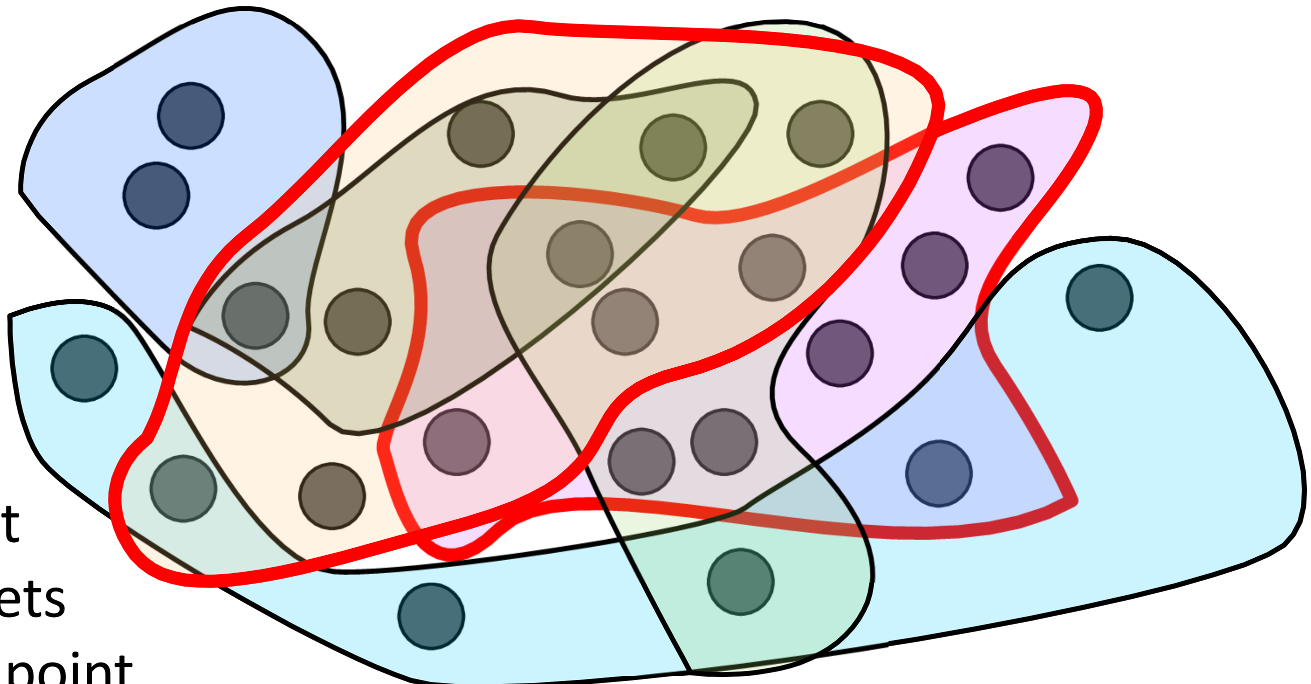
Greedy Set Cover: Repeatedly choose the set that covers the most # of new elements



Find smallest collection of sets containing every point

Set-Cover

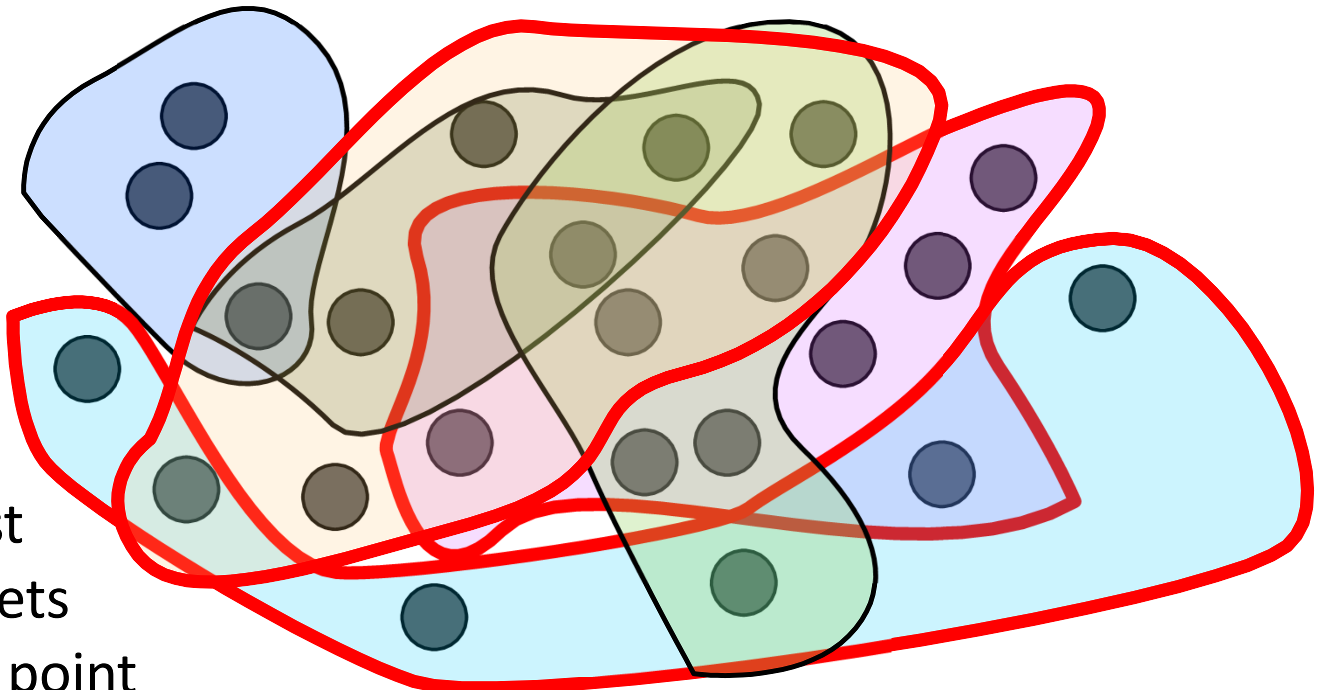
Greedy Set Cover: Repeatedly choose the set that covers the most # of new elements



Find smallest collection of sets containing every point

Set-Cover

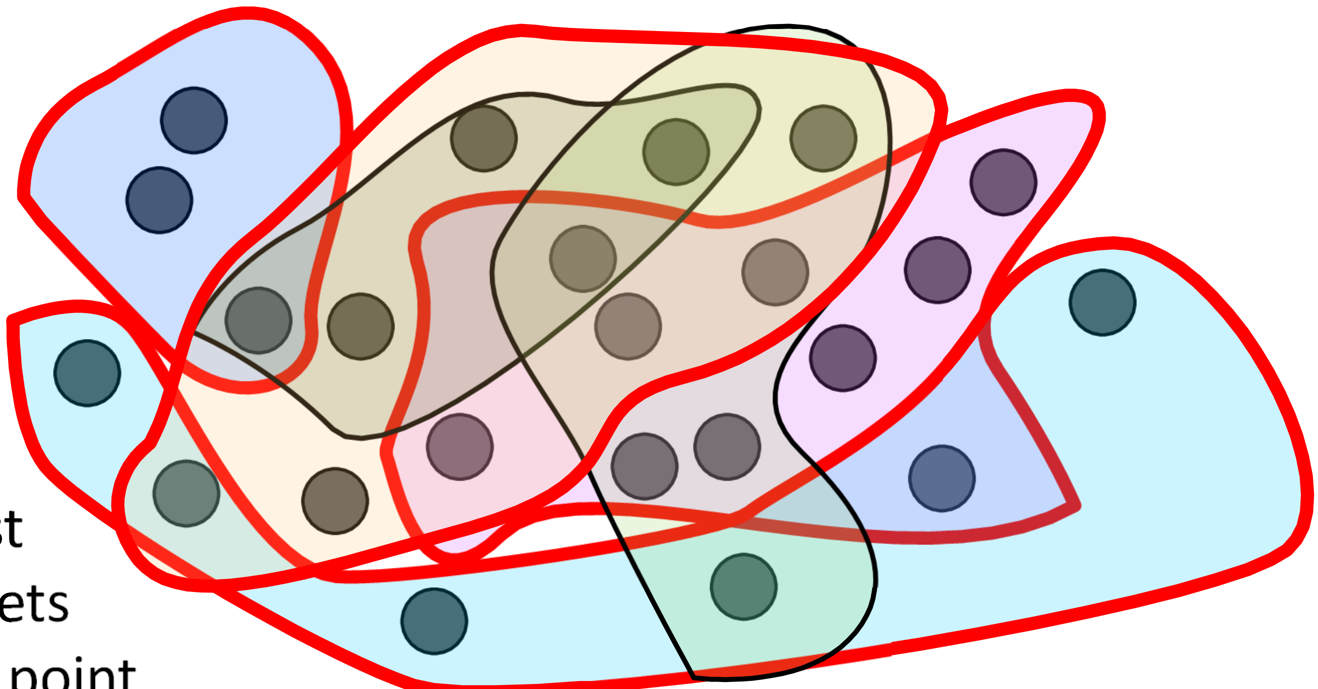
Greedy Set Cover: Repeatedly choose the set that covers the most # of new elements



Find smallest collection of sets containing every point

Set-Cover

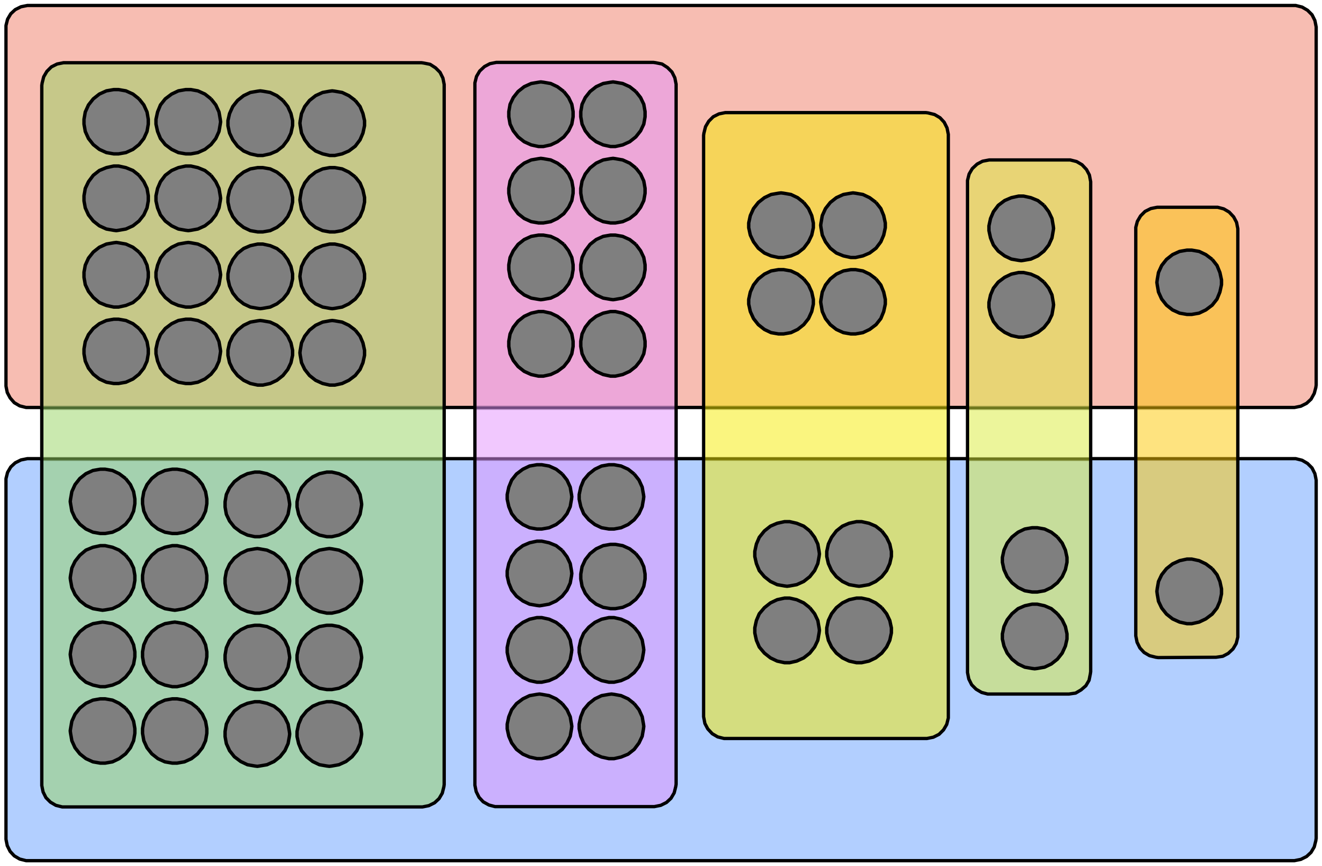
Greedy Set Cover: Repeatedly choose the set that covers the most # of new elements



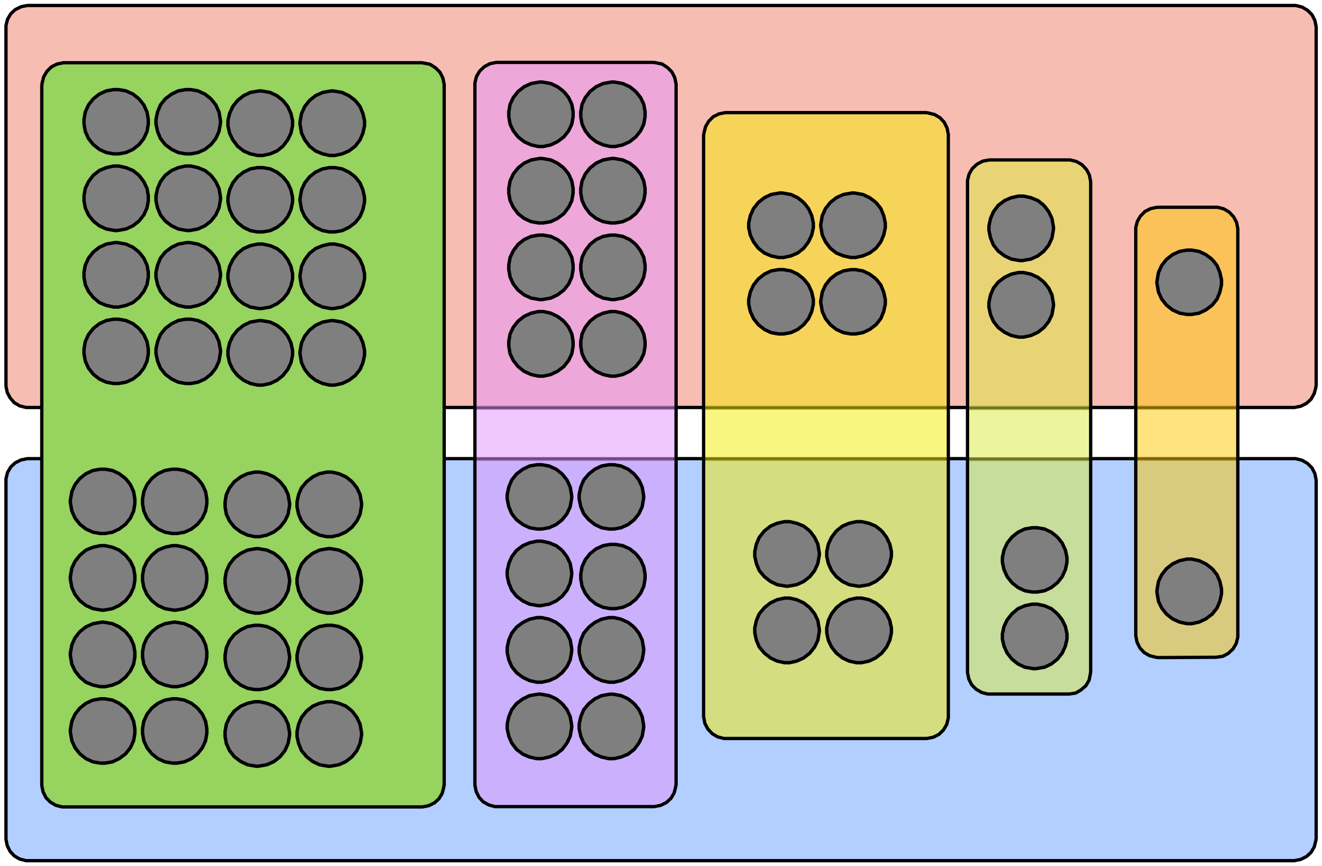
Find smallest collection of sets containing every point

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

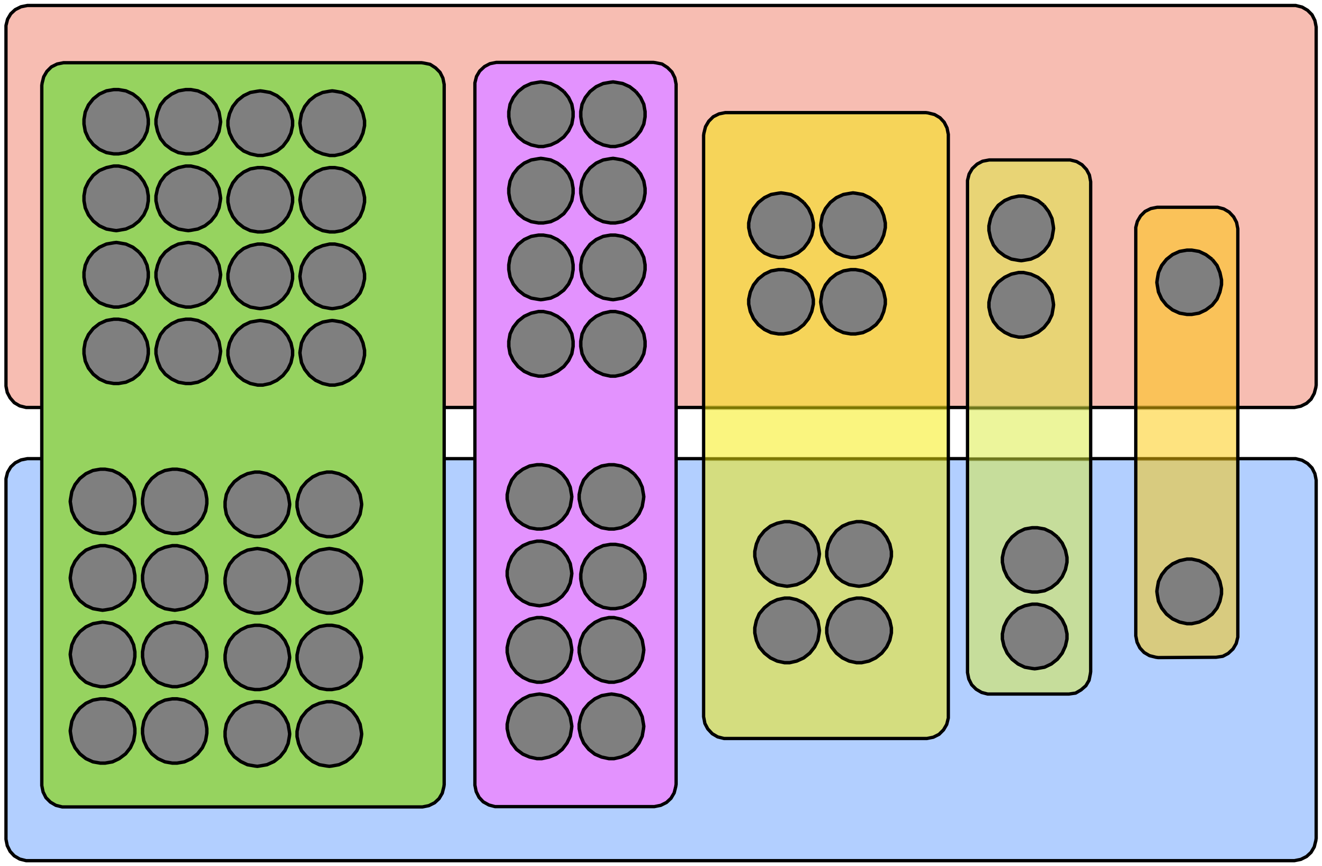
Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered



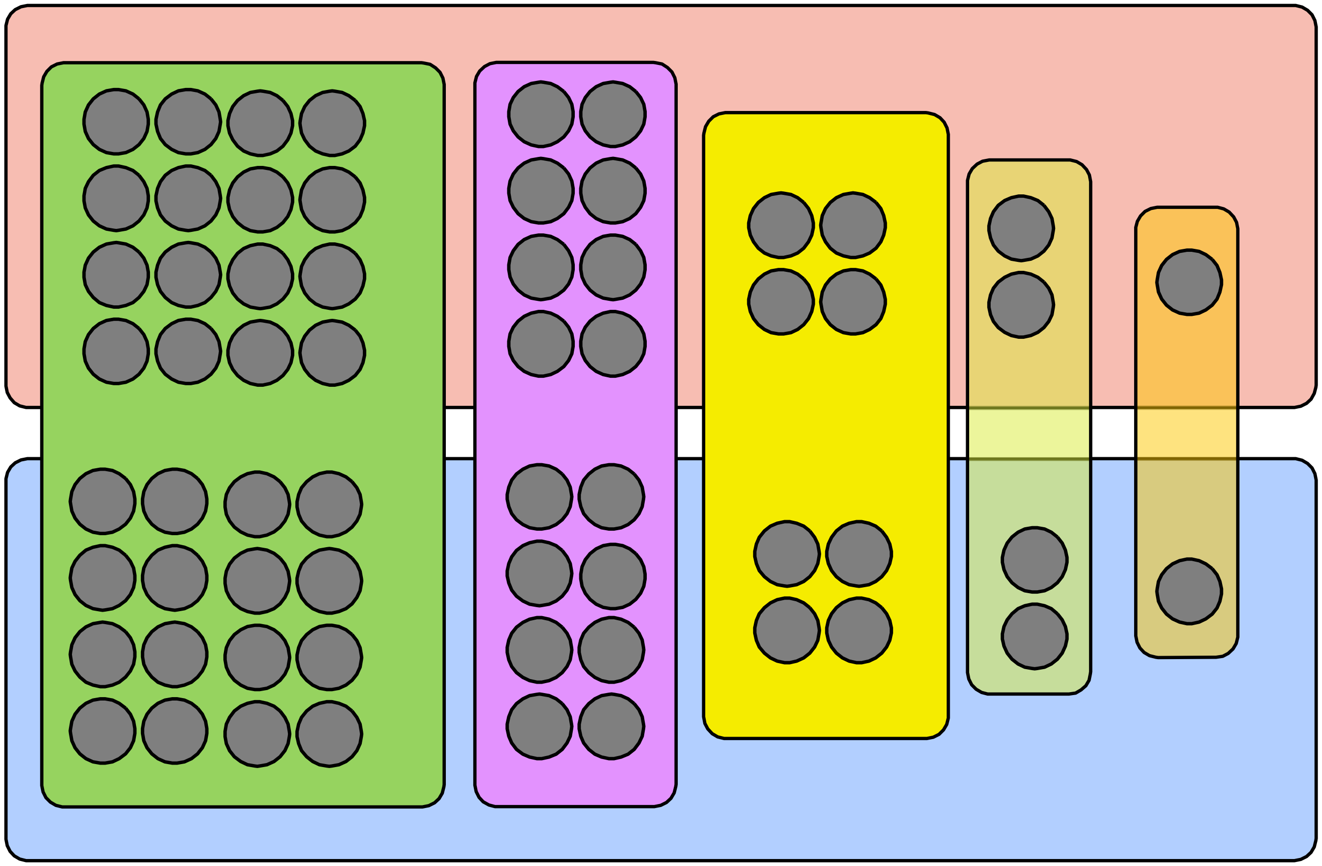
Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered



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

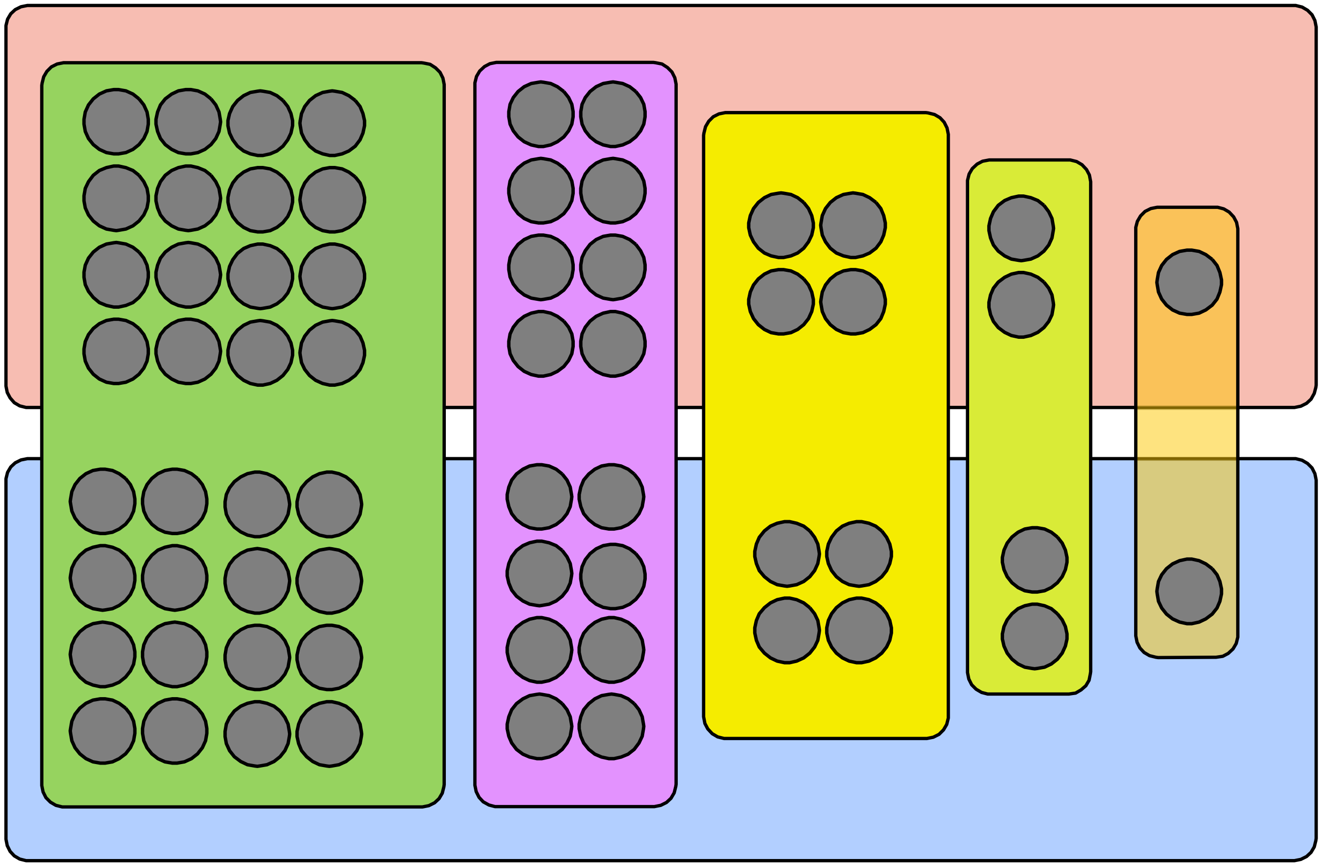


Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered

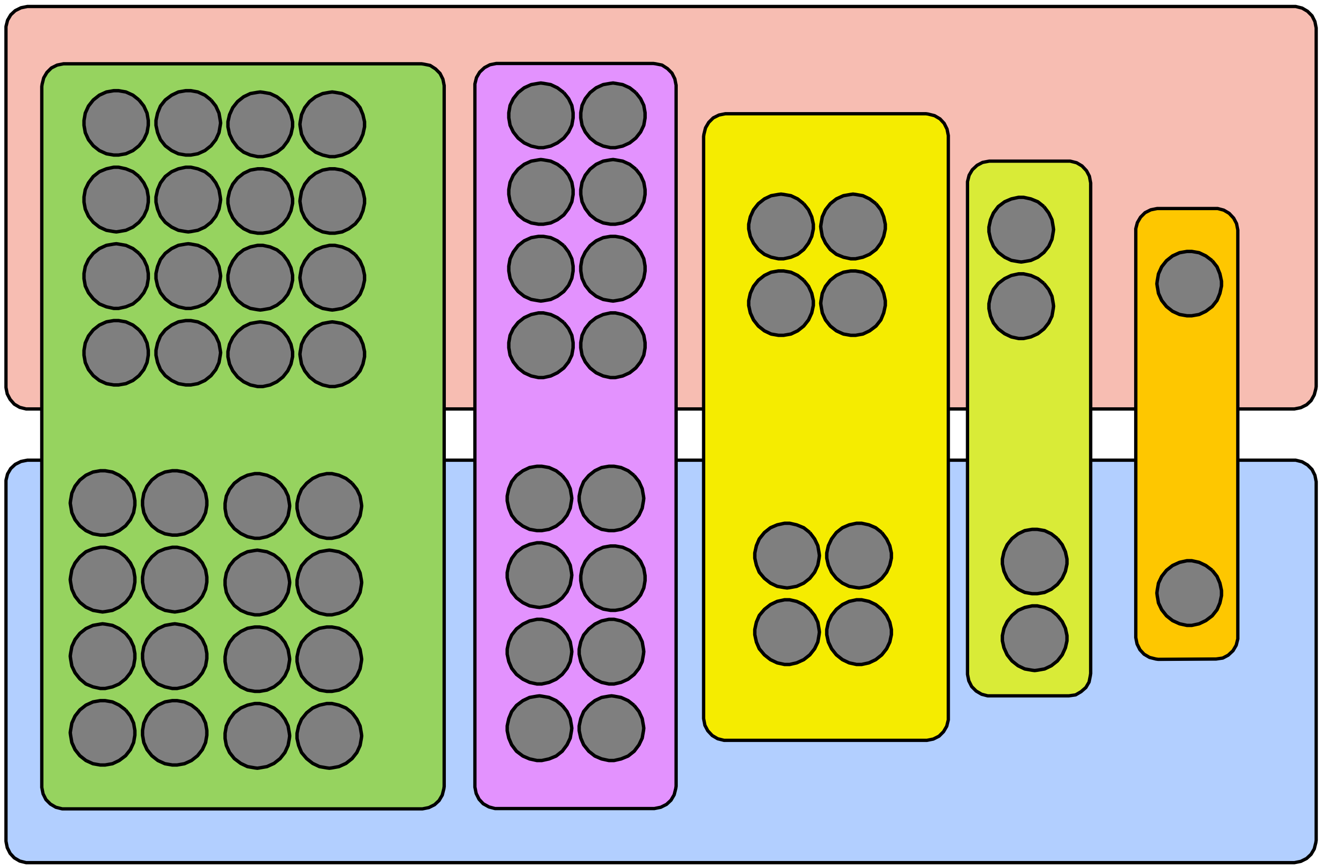


PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING

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



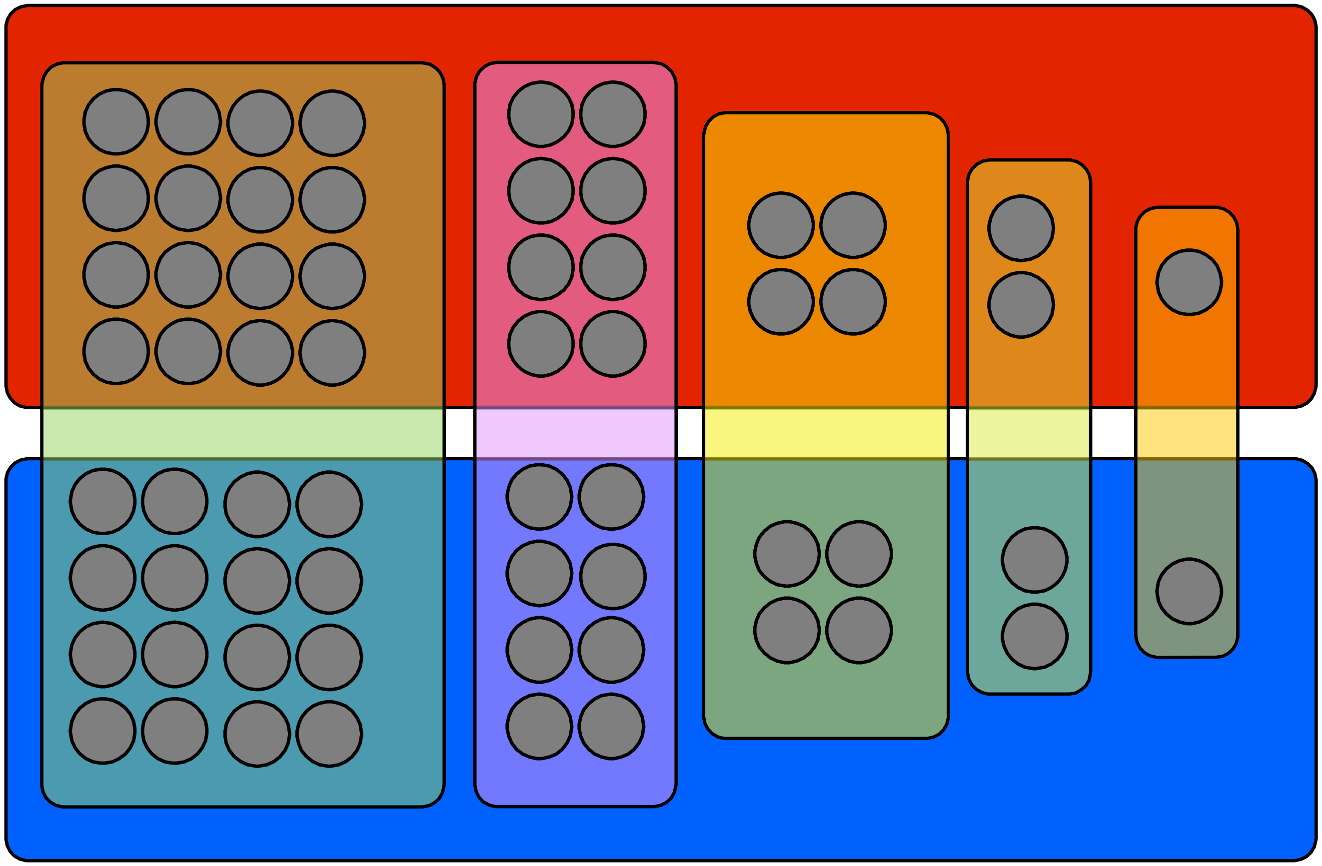
Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered



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 $\boldsymbol{k}$ then the greedy set cover has size $\leq \boldsymbol{k} \ln \boldsymbol{n}$.

Proof: Suppose that there is a set cover of size $\boldsymbol{k}$.

At each step all elements remaining are covered by these $\boldsymbol{k}$ sets.

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

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

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

$\mathbf{1} - \boldsymbol{x} < \boldsymbol{e}^{-\boldsymbol{x}}$ for $\boldsymbol{x} > \mathbf{0}$

Travelling-Salesperson Problem (TSP)

Travelling-Salesperson Problem (TSP):

Given: a set of n cities $v_1, \ldots, v_n$ 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.

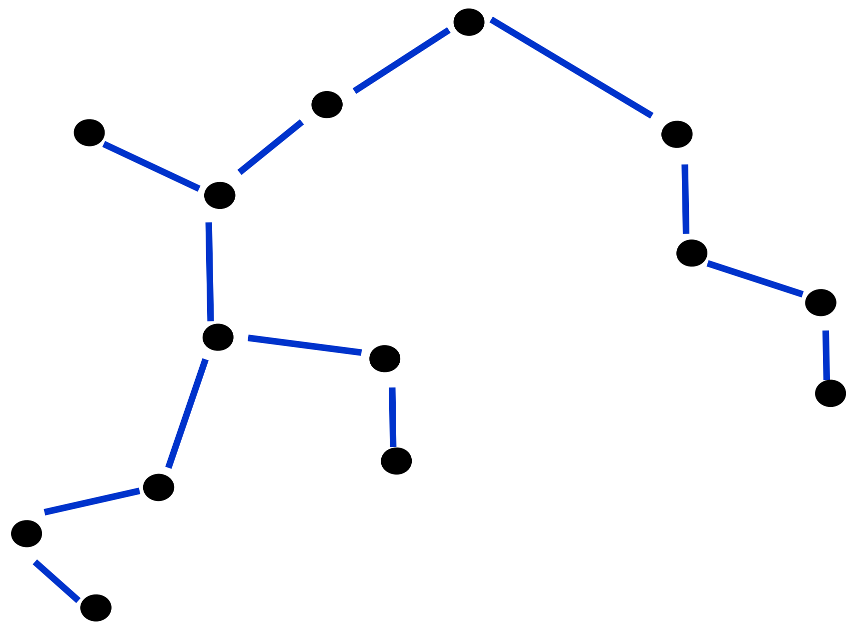
MetricTSP:

The distance function d satisfies the triangle inequality:

$$d(u, w) \leq d(u, v) + d(v, w)$$

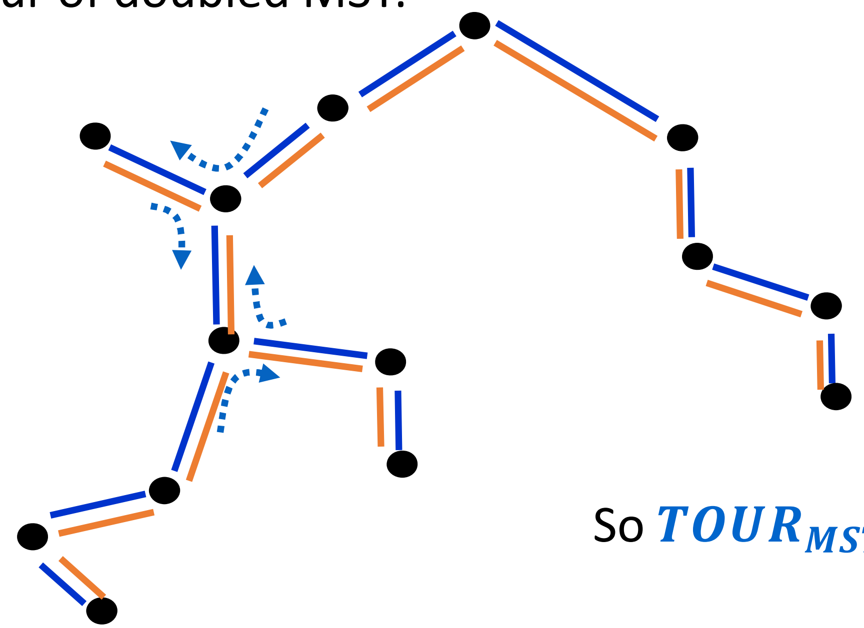
Proper tour: visit each city exactly once.

Minimum Spanning Tree Approximation: Factor of 2



TSP: Minimum Spanning Tree Factor 2 Approximation

Euler Tour of doubled MST:



Euler tour covers each edge twice
so $\boldsymbol{TOUR_{MST}(G) = 2\ MST(G)}$

Any tour contains a spanning tree
so $\boldsymbol{MST(G) \leq TOUR_{OPT}(G)}$

So $\boldsymbol{TOUR_{MST}(G) = 2\ MST(G) \leq 2\ TOUR_{OPT}(G)}$

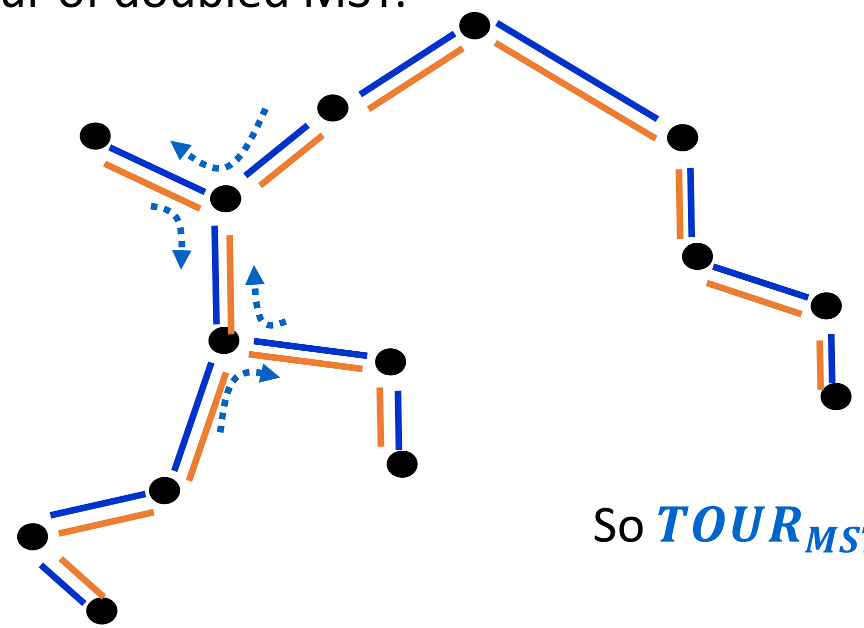
This visits each node more than once, so not a proper tour.

Why did this work?

- We found an Euler tour on a graph that used the edges of the original graph (possibly repeated).
- The weight of the tour was the total weight of the new graph.
- Suppose now
  - All edges possible
  - Weights satisfy the triangle inequality (MetricTSP)

MetricTSP: Minimum Spanning Tree Factor 2 Approximation

Euler Tour of doubled MST:



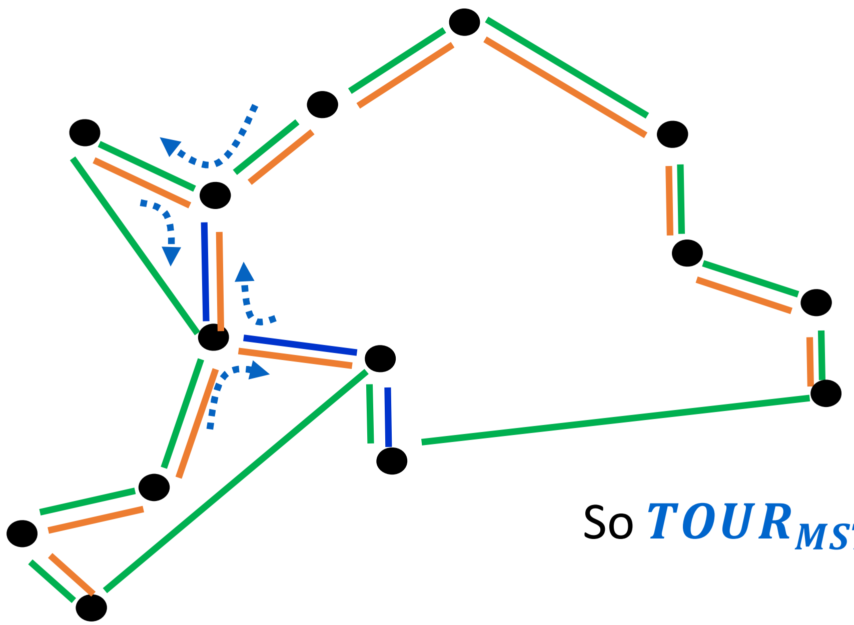
Euler tour covers each edge twice
so $\boldsymbol{TOUR_{MST}(G) = 2\ MST(G)}$

Any tour contains a spanning tree
so $\boldsymbol{MST(G) \leq TOUR_{OPT}(G)}$

So $\boldsymbol{TOUR_{MST}(G) = 2\ MST(G) \leq 2\ TOUR_{OPT}(G)}$

Instead: take shortcut to next unvisited vertex on the Euler tour
By triangle inequality this can only be shorter.

MetricTSP: Minimum Spanning Tree Factor 2 Approximation



Euler tour covers each edge twice
so $\boldsymbol{TOUR_{MST}(G) = 2\ MST(G)}$

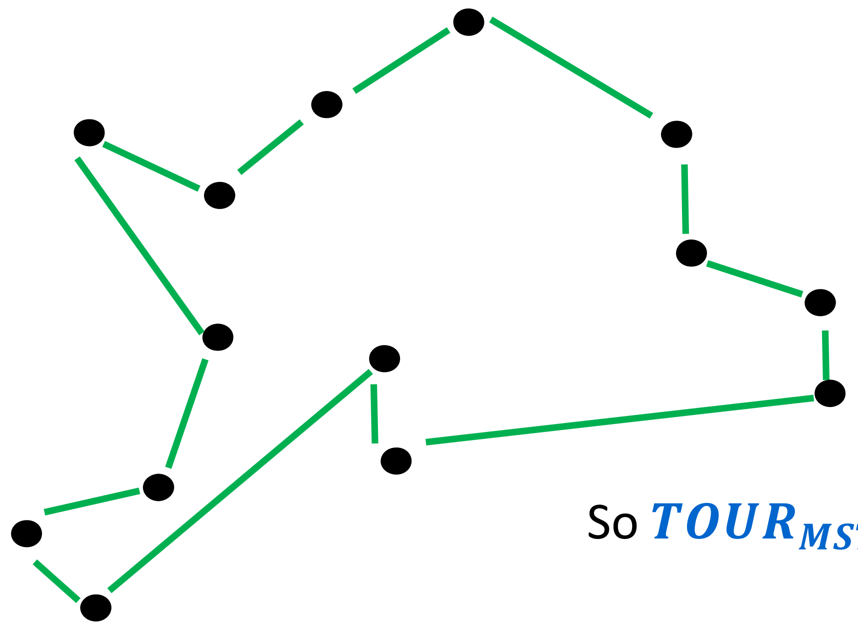
Any tour contains a spanning tree
so $\boldsymbol{MST(G) \leq TOUR_{OPT}(G)}$

So $\boldsymbol{TOUR_{MST}(G) = 2\ MST(G) \leq 2\ TOUR_{OPT}(G)}$

Instead: take shortcut to next unvisited vertex on the Euler tour
By triangle inequality this can only be shorter.

MetricTSP: Minimum Spanning Tree Factor 2 Approximation

Final:



Euler tour covers each edge twice
so $\boldsymbol{TOUR_{MST}(G) = 2\ MST(G)}$

Any tour contains a spanning tree
so $\boldsymbol{MST(G) \leq TOUR_{OPT}(G)}$

So $\boldsymbol{TOUR_{MST}(G) = 2\ MST(G) \leq 2\ TOUR_{OPT}(G)}$

Instead: take shortcut to next unvisited vertex on the Euler tour
By triangle inequality this can only be shorter.

Christofides Algorithm: A factor 3/2 approximation

Any subgraph of the weighted complete graph that has an Euler Tour will work also!

Fact: To have an Euler Tour it suffices to have all degrees even.

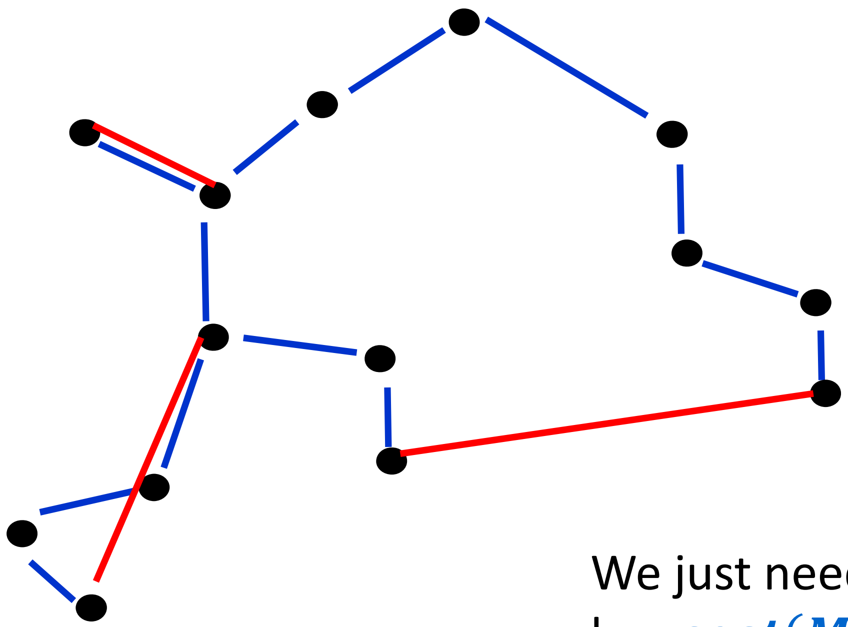
Christofides Algorithm:

- Compute an MST $\boldsymbol{T}$
- Find the set $\boldsymbol{O}$ of odd-degree vertices in $\boldsymbol{T}$
- Add a minimum-weight perfect matching* $\boldsymbol{M}$ on the vertices in $\boldsymbol{O}$ to $\boldsymbol{T}$ to make every vertex have even degree
  - There are an even number of odd-degree vertices!
- Use an Euler Tour $\boldsymbol{E}$ in $\boldsymbol{T} \cup \boldsymbol{M}$ and then shortcut as before

Theorem: $\boldsymbol{Cost(E) \leq 1.5\, TOUR_{OPT}}$

*Requires finding optimal matchings in general graphs, not just bipartite ones

Christofides Approximation

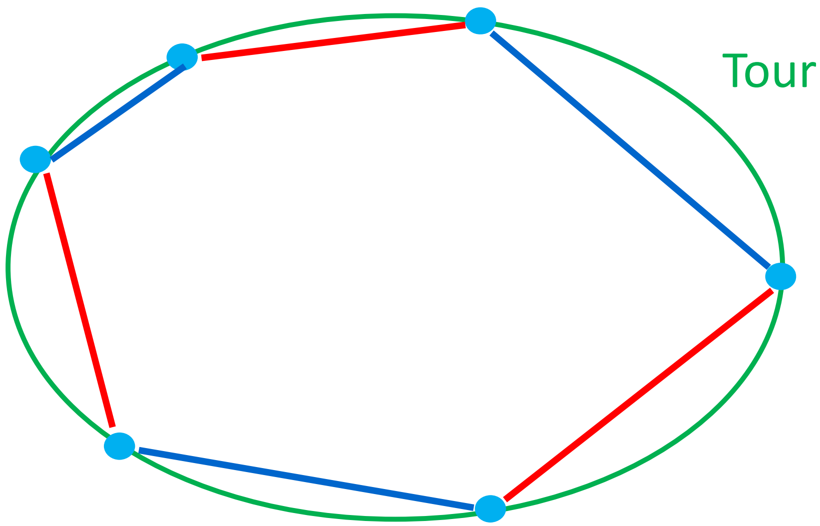


Any tour contains a spanning tree so $\boldsymbol{MST \leq TOUR_{OPT}}$

We just need to show that the matching $\boldsymbol{M}$ has $\boldsymbol{cost(M) \leq TOUR_{OPT}/2}$

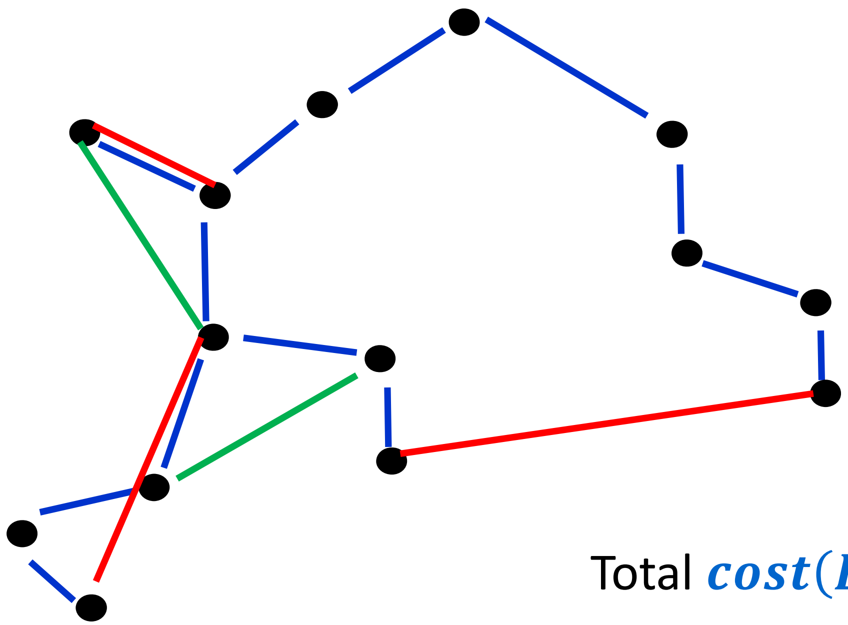
Christofides Approximation

Any tour costs at least the cost of two matchings $\boldsymbol{M_1}$ and $\boldsymbol{M_2}$ on $\boldsymbol{O}$



$$\boldsymbol{2\ cost(M) \leq cost(M_1) + cost(M_2) \leq TOUR_{OPT}}$$

Christofides Approximation Final Tour



Total $\boldsymbol{cost(E) \leq 3\ TOUR_{OPT}/2}$

Max-3SAT Approximation

Max-3SAT: Given a 3CNF formula $\boldsymbol{F}$ find a truth assignment that satisfies the maximum possible # of clauses of $\boldsymbol{F}$.

Observation: A single clause on 3 variables only rules out $\mathbf{1/8}$ of the possible truth assignments since each literal has to be false to be ruled out.

$\Rightarrow$ a random truth assignment will satisfy the clause with probability $\mathbf{7/8}$.

So in expectation, if $\boldsymbol{F}$ has $\boldsymbol{m}$ clauses, a random assignment satisfies $\mathbf{7}\boldsymbol{m}\mathbf{/8}$ of them.

A greedy algorithm can achieve this: Choose most frequent literal appearing in clauses that are not yet satisfied and set it to true.

If $\mathbf{P} \neq \mathbf{NP}$ no better approximation is possible

Knapsack Problem

Each item has a value $\boldsymbol{v_i}$ and a weight $\boldsymbol{w_i}$.

Maximize $\sum_{i \in S} \boldsymbol{v_i}$ with $\sum_{i \in S} \boldsymbol{w_i} \leq \boldsymbol{W}$.

Theorem: For any $\boldsymbol{\varepsilon > 0}$ there is an algorithm that produces a solution within $\boldsymbol{(1 + \varepsilon)}$ factor of optimal for the Knapsack problem with running time $O(\boldsymbol{n^2/\varepsilon^2})$

"Polynomial-Time Approximation Scheme" or PTAS

Algorithm: Maintain the high order bits in the dynamic programming solution.

Hardness of Approximation

Polynomial-time approximation algorithms for $\mathbf{NP}$-hard optimization problems can sometimes be ruled out unless $\mathbf{P} = \mathbf{NP}$.

Easy example:

Coloring: Given a graph $G = (V, E)$ find the smallest k such that G has a k-coloring.

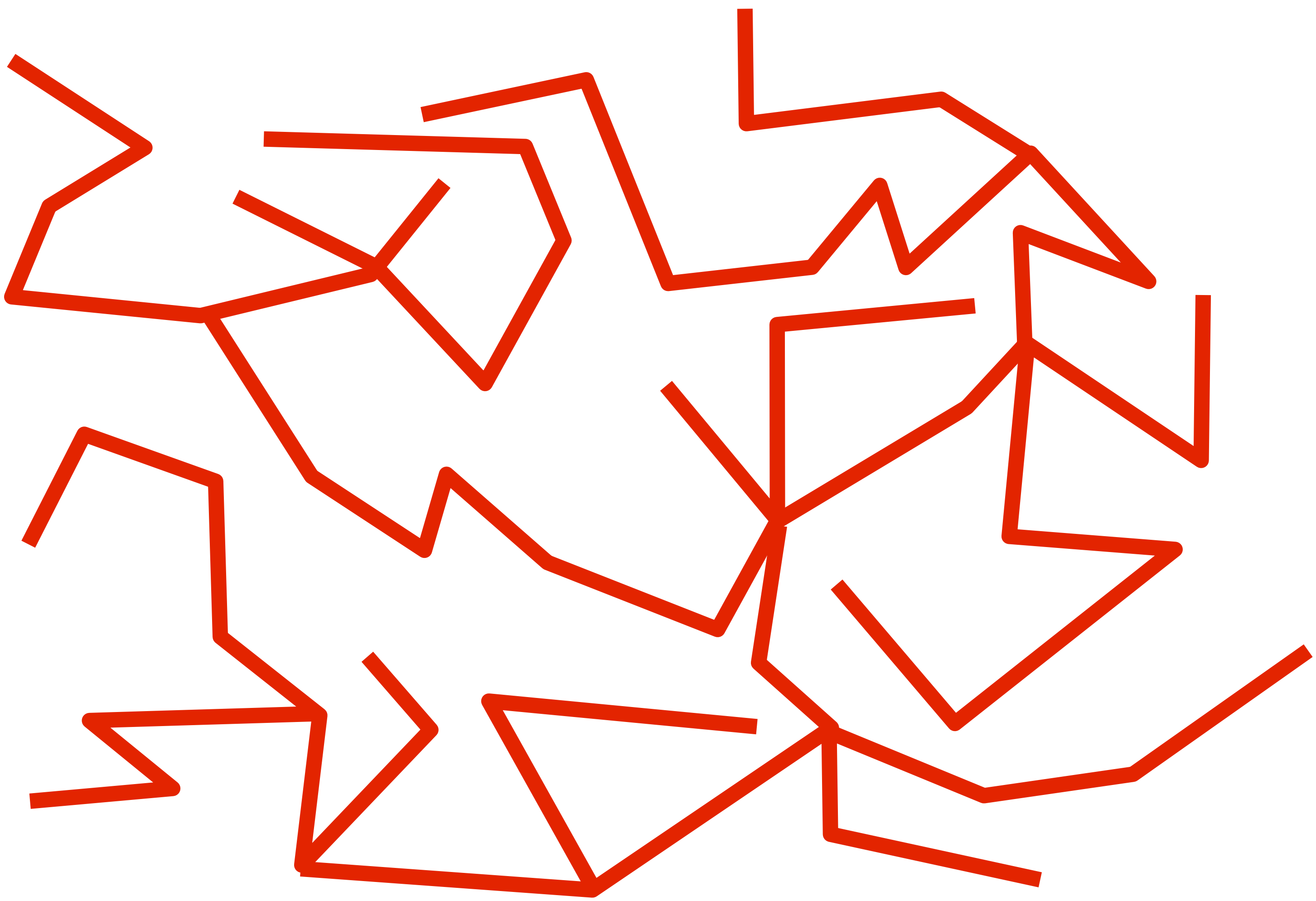
Because $\mathbf{3}$-coloring is $\mathbf{NP}$-hard, no approximation ratio better than $4/3$ is possible unless $\mathbf{P} = \mathbf{NP}$ because you would have to be able to figure out if a $\mathbf{3}$-colorable graph can be colored in < 4 colors. i.e. if it can be $\mathbf{3}$-colored.

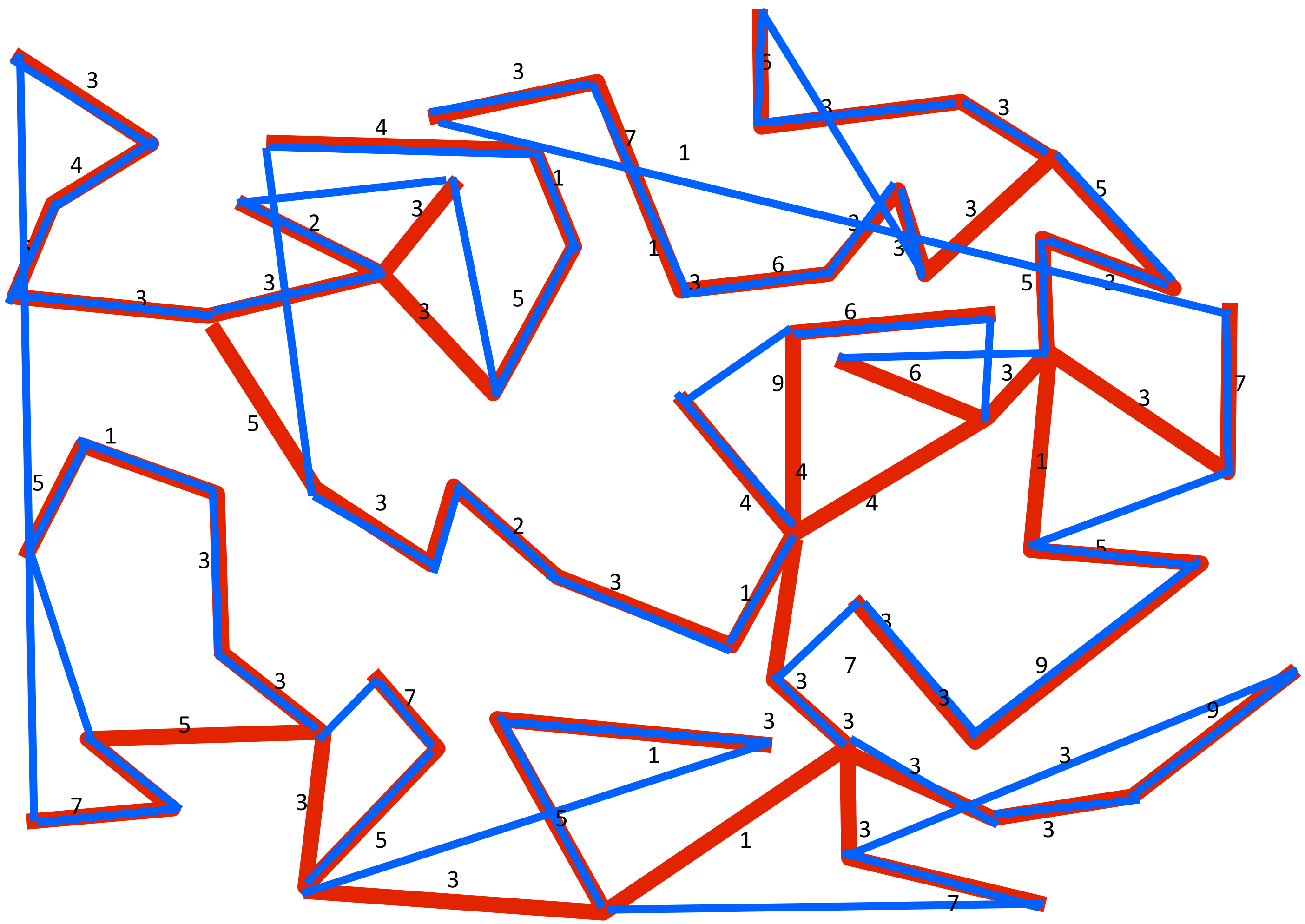
- We now know a huge amount about the hardness of approximating $\mathbf{NP}$ optimization problems if $\mathbf{P} \neq \mathbf{NP}$.
- Approximation factors are very different even for closely related problems like **Vertex-Cover** and **Independent-Set**.

Approximation Algorithms/Hardness of Approximation

Research has classified many problems based on what kinds of polytime approximations are possible if $\mathbf{P} \neq \mathbf{NP}$

- **Best:** $(\mathbf{1} + \boldsymbol{\varepsilon})$ factor for any $\boldsymbol{\varepsilon} > \mathbf{0}$. (PTAS)
  - packing and some scheduling problems, TSP in plane
- Some fixed constant factor $> \mathbf{1}$. e.g. $\mathbf{2}, \mathbf{3}/\mathbf{2}, \mathbf{8}/\mathbf{7}, \mathbf{100}$
  - Vertex Cover, Max-3SAT, MetricTSP, other scheduling problems
  - Exact best factors or very close upper/lower bounds known for many problems.
- $\Theta(\log \boldsymbol{n})$ factor
  - Set Cover, Graph Partitioning problems
- **Worst:** $\Omega(\boldsymbol{n}^{\mathbf{1}-\boldsymbol{\varepsilon}})$ factor for every $\boldsymbol{\varepsilon} > \mathbf{0}$.
  - Clique, Independent-Set, Coloring




3 5
3
3 3
4
7
1
4
1
5
3 3
2 3
1 3
6
3 3 5 2
3 5
3 6
6 3
9 7
3
5
1
1
4
5
3 4 4
2
5
3
3
1
3
7 9
3 3
7 3
9
3 3
5
1 3
3
3
7
5
3 3
5 1
3
7