COMP 3011
DESIGN AND ANALYSIS OF ALGORITHMS
FALL 2024

Approximation Algorithms

LI Bo
Department of Computing
The Hong Kong Polytechnic University

Definition

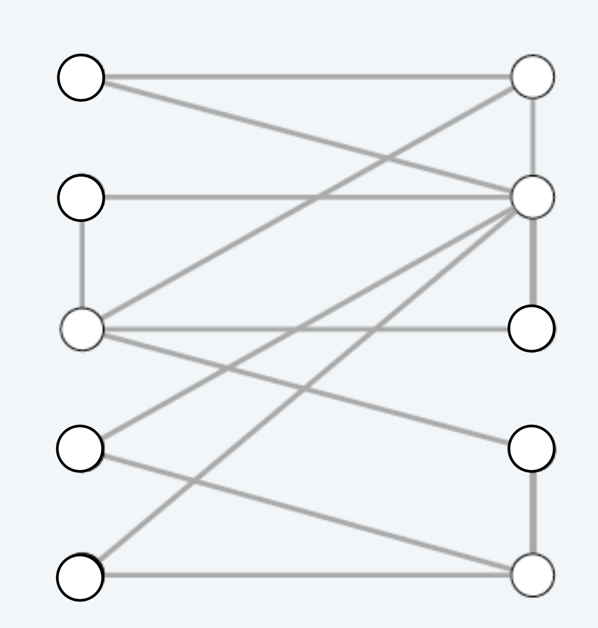
For a maximization problem, an algorithm is called α-approximation if for any input, the algorithm returns a feasible solution S such that

$$\frac{f(S)}{f(O)} \geq \alpha,$$

where O is an optimal solution, and f evaluates the quality of the solution.

INDEPENDENT SET

Independent Set Problem



Given a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$. Find a set of maximum number of vertices such that no two are adjacent?

Intuition

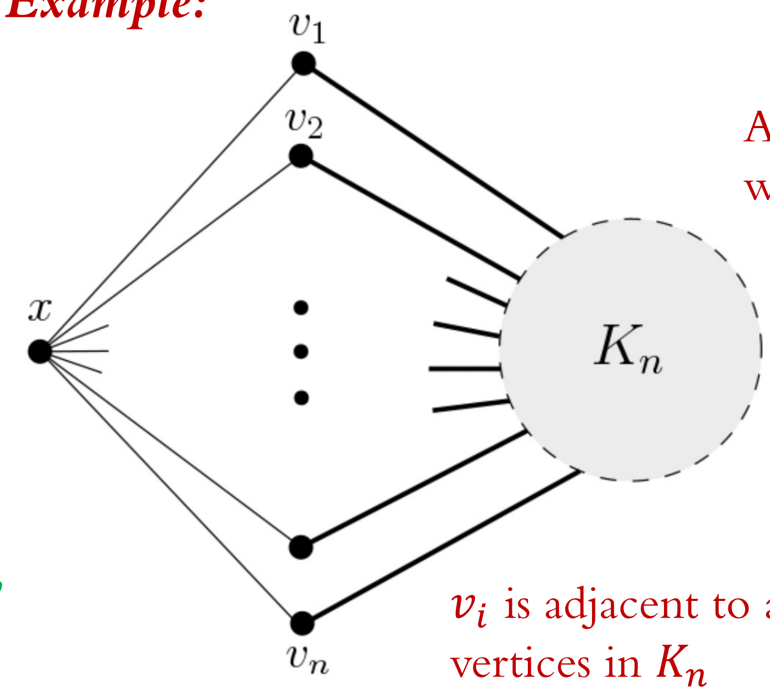
Always select the node with minimum degree.

Greedy Algorithm

Require: a graph $G = (V, E)$
  $W \leftarrow V$
  $S \leftarrow \emptyset$
  while $W \neq \emptyset$ **do**
    Find a vertex $v \in W$ with minimum degree in $G[W]$
    $W \leftarrow W \setminus N_G[v]$
    $S \leftarrow S \cup \{v\}$
  end while
  return S

the subset of vertices adjacent to v and v

A Bad Example:



A complete graph with n vertices

v_i is adjacent to all vertices in K_n

$OPT = n$

$ALG = 2$

Independent Set Problem

Greedy Algorithm

Require: a graph $G = (V, E)$
  $W \leftarrow V$
  $S \leftarrow \emptyset$
  while $W \neq \emptyset$ **do**
    Find a vertex $v \in W$ with minimum degree in $G[W]$
    $W \leftarrow W \setminus N_G[v]$
    $S \leftarrow S \cup \{v\}$
  end while
  return S

***Theorem*.**
The Greedy Algorithm is $(1/(\Delta + 1))$-approximation for graphs with degree at most Δ.

Proof

- Every time a vertex v is picked by Greedy, at most Δ vertices are removed.
- So at the end at most $|S| \cdot (\Delta + 1)$ vertices have been removed.
- All nodes have been removed:

$$n \leq (\Delta + 1) \cdot |S|$$

That is

$$|S| \geq \frac{n}{\Delta + 1} \geq \frac{OPT}{\Delta + 1}$$

LINEAR PROGRAMMING

Linear Programming

- Linear Programming deals with the problem of optimizing a linear objective function subject to linear equality and inequality constraints on the decision variables.
- Linear programming has many practical applications (in transportation, maximum flow, ...).

Example 1: The Diet Problem

- A list of foods is together with the nutrient content and the cost per unit weight of each food.
- A certain amount of each nutrient is required per day.
- Suppose we have two types of grains and three types of nutrients.
- The requirement per day of starch, proteins and vitamins is 8, 15 and 3 respectively.

	Starch	Proteins	Vitamins	Cost ($/kg)
G1	5	4	2	0.6
G2	7	2	1	0.35

Problem: Find how much of each food to consume per day so as to get the required amount per day of each nutrient at minimal cost.

Example 1: The Diet Problem

	Starch	Proteins	Vitamins	Cost ($/kg)
G1	5	4	2	0.6
G2	7	2	1	0.35

- The requirement per day of starch, proteins and vitamins is 8, 15 and 3 respectively.
- ***Problem***: Find how much of each food to consume per day so as to get the required amount per day of each nutrient at minimal cost.

Formulate the Problem as a Linear Program

- ***Decision variables***: represent the unknowns in the problem.
  - x_1: number of units of grain G1 to be consumed per day;
  - x_2: number of units of grain G2 to be consumed per day.
- ***Objective function***: to be minimized or maximized
  - Min $z = 0.6x_1 + 0.35x_2$
- ***Constraints***: need to be satisfied by the variables
  - $5x_1 + 7x_2 \geq 8$
  - $4x_1 + 2x_2 \geq 15$
  - $2x_1 + x_2 \geq 3$
  - $x_1 \geq 0, x_2 \geq 0$

Minimize $z = 0.6x_1 + 0.35x_2$

subject to:

$$5x_1 + 7x_2 \geq 8$$
$$4x_1 + 2x_2 \geq 15$$
$$2x_1 + x_2 \geq 3$$
$$x_1 \geq 0, x_2 \geq 0.$$

Example 2: The Transportation Problem

- Suppose a company manufacturing widgets has
  - Two factories located at cities F1 and F2 and
  - Three retail centers located at C1, C2 and C3.
- The monthly demand at the retail centers are 8, 5 and 2 respectively.
- The monthly supply at the factories are 6 and 9 respectively.
- It is required that total supply equals the total demand.
- The cost of transportation of 1 widget between any factory and any retail center:

Variables: Let x_{ij} $(i = 1, 2$ and $j = 1, 2, 3)$ be the number of widgets transported from Fi to Cj.

	C1	C2	C3
F1	5	5	3
F2	6	4	1

Goal: to determine the quantity to be transported from each factory to each retail center so as to meet the demand at minimum total shipping cost.

constraints

objective

$$\text{Minimize} \quad 5x_{11} + 5x_{12} + 3x_{13} + 6x_{21} + 4x_{22} + x_{23}$$

subject to:

$$x_{11} + x_{21} = 8$$
$$x_{12} + x_{22} = 5$$
$$x_{13} + x_{23} = 2$$
$$x_{11} + x_{12} + x_{13} = 6$$
$$x_{21} + x_{22} + x_{23} = 9$$
$$x_{11} \geq 0, x_{21} \geq 0, x_{31} \geq 0,$$
$$x_{12} \geq 0, x_{22} \geq 0, x_{32} \geq 0.$$

Linear Programming

- In 1930s, Kantorovich and Koopmans brought new life to linear programming by showing its widespread applicability in resource allocation problems. They jointly received the Nobel Prize in Economics in 1975.
- In 1947, Dantzig invented the first practical algorithm for solving LPs: ***the simplex method***. This essentially revolutionized the use of linear programming in practice.
- In 1979, Khachiyan showed that LPs were solvable in polynomial time using the "***ellipsoid method***". This was a theoretical breakthrough more than a practical one, as in practice the algorithm was quite slow.
- In 1984, Karmarkar developed the "***interior point method***", another polynomial time algorithm for LPs, which was also efficient in practice. Along with the simplex method, this is the method of choice today for solving LPs.

Fractional View of Knapsack

Theorem.
KNAPSACK formulation proves that INTEGER-PROGRAMMING is an ***NP***-hard optimization problem.

KNAPSACK (Optimization)
Given a set of items $X = \{a_1, \cdots, a_n\}$, cost $c_i \geq 0$, values $v_i \geq 0$, a budget B. Find a subset $S \subseteq X$ such that:

$$\sum_{i \in S} c_i \leq B,$$

and $v_i(S)$ is maximized.

LP can be solved in poly time!

Integer liner programming (LP):

$$\max \sum_{i=1}^{n} v_i x_i$$

$$s.t. \sum_{i=1}^{n} c_i x_i \leq B$$

$$\boxed{x_i \in \{0, 1\}}$$

If x^* is optimal solution to *ILP*, then $S = \{a_i \in X : x_i^* = 1\}$ is a max-value feasible subset.

$x_i = 1$ means item a_i is selected
$x_i = 0$ means item a_i is NOT selected

relaxation

Linear programming

$$\max \sum_{i=1}^{n} v_i x_i$$

$$s.t. \sum_{i=1}^{n} c_i x_i \leq B$$

$$0 \leq x_i \leq 1$$

$x_i \in [0,1]$ indicates how much fraction item i is selected

Fractional View of Knapsack

KNAPSACK (Optimization)

Given a set of items $X = \{a_1, \cdots, a_n\}$, cost $c_i \geq 0$, values $v_i \geq 0$, a budget B. Find a subset $S \subseteq X$ such that:

$$\sum_{i \in S} c_i \leq B,$$

and $v_i(S)$ is maximized.

Integer linear programming (ILP):

$x_i \in \{0,1\}$ indicates whether item i is selected

$$\max_{\mathbf{x}} \mathbf{v}^\mathsf{T}\mathbf{x}$$

$$s.t. \sum_{i=1}^{n} c_i x_i \leq B$$

$0 \leq x_i \leq 1$ $\quad \boxed{x_i \in \{0,1\}}$

Linear programming (LP)

Can be solved in polynomial time.

An ***optimal solution*** to the relaxation LP:

- Sort the items in decreasing order of densities. The density of element i is defined by the ration v_i/c_i (value per cost).
- Add items to the solution one-by-one in this order as long as the sum of the costs does not exceed the budget.
- For the first item i which violates the budget, only add a fraction x_i of it such that the budget constraint is tight:

$$x_i = \frac{B - \sum_{j<i} c_j}{c_i}$$

12

$x_1 = 1$ and $x_2 = \frac{M-1}{M}$

Example:

- item 1 has cost 1 and value 2
- item 2 has cost M and value M (for some $M > 2$)
- the budget is $B = M$

Greedy Algorithm of Knapsack

Greedy Algorithm ALG_1

1: $S \leftarrow \varnothing$
2: **while** $c\left(S \cup \text{argmax}_{i \notin S} \frac{v_i}{c_i}\right) \leq B$ **do**
3: $\quad S \leftarrow S \cup \text{argmax}_{i \notin S} \frac{v_i}{c_i}$
4: **end while**
5: **return** S

Give up the fractional item in OPT_{LP}

Example:

- item 1 has cost 1 and value 2
- item 2 has cost M and value M (for some $M > 2$)
- budget is $B = M$

Not a constant approximation!

Goal: $ALG \geq \alpha \cdot OPT$ for all instances

$$OPT \leq OPT_{LP} = v_1 + v_2 + \cdots v_{i-1} + v_i \cdot x_i$$

$$v_1 + v_2 + \cdots v_{i-1} \leq OPT_{LP} \leq v_1 + v_2 + \cdots v_{i-1} \boxed{+ v_i}$$

$$v_1 + v_2 + \cdots v_{i-1} \leq ALG_1$$

$$v_i \leq ALG_2$$

A single item that can be put in the knapsack

***ALG_2*:** Select the item with largest value

$$OPT \leq OPT_{LP} \leq ALG_1 + ALG_2$$

Algorithm ALG

***ALG*:** Return $\max\{ALG_1, ALG_2\}$

$$OPT_{LP} \leq 2 \cdot \max\{ALG_1, ALG_2\} = 2 \cdot ALG$$

***Theorem*.**
ALG is a ***1/2-approximation poly-time algorithm***.

Vertex Cover Problem

Vertex Cover Problem

VERTEX-COVER.

Given a graph $G = (V, E)$ and an integer k, is there a subset of k (or fewer) vertices such that each edge is incident to at least one vertex in the subset?

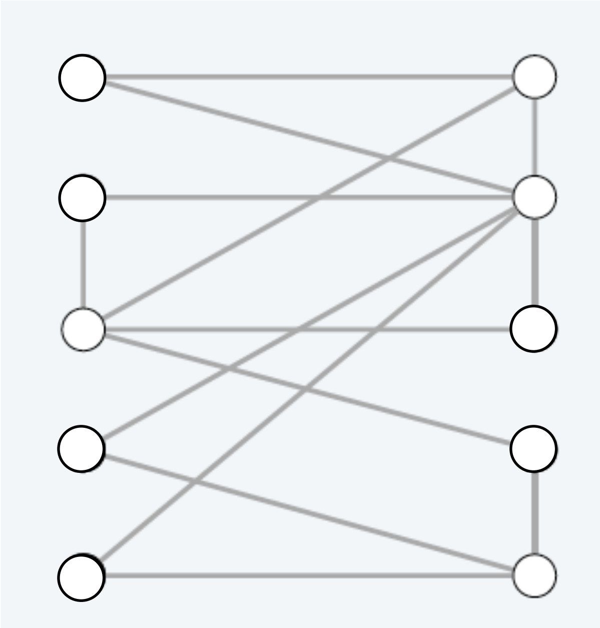
FIND-MIN-VERTEX-COVER.

Given a graph $G = (V, E)$, find a vertex cover with minimum number of vertices.

↓

NP-hard

Approximation Algorithm!



Definition

For a minimization problem, an algorithm is called α-approximation if for any input, the algorithm returns a feasible solution S such that

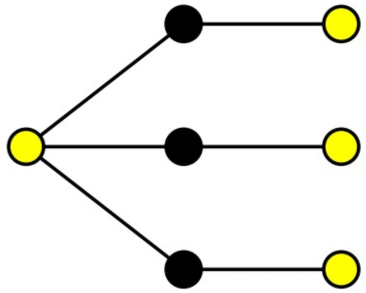
$$\frac{f(S)}{f(O)} \leq \alpha, \qquad \alpha \geq 1$$

where O is an optimal solution, and f evaluates the quality of the solution.

Vertex Cover Problem

Idea 1.
Pick an arbitrary vertex with at least one uncovered edge incident to it, put it into the cover, and repeat.

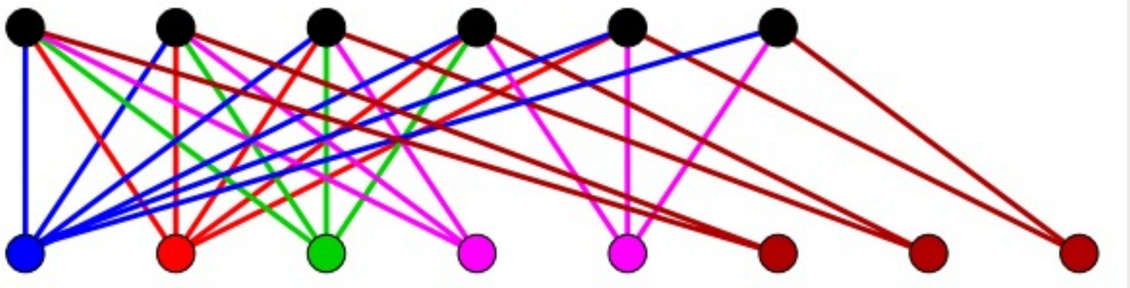
Idea 2.
How about picking the vertex that covers the most uncovered edges?



Cannot be better than $\Omega(\log n)$-approximation!

$\delta \le n$

L with size n



R_6 R_5 R_4 R_3 R_2 R_1

$R = R_1 \cup \cdots \cup R_n$ with size $n \log n$

$\delta_n = n$

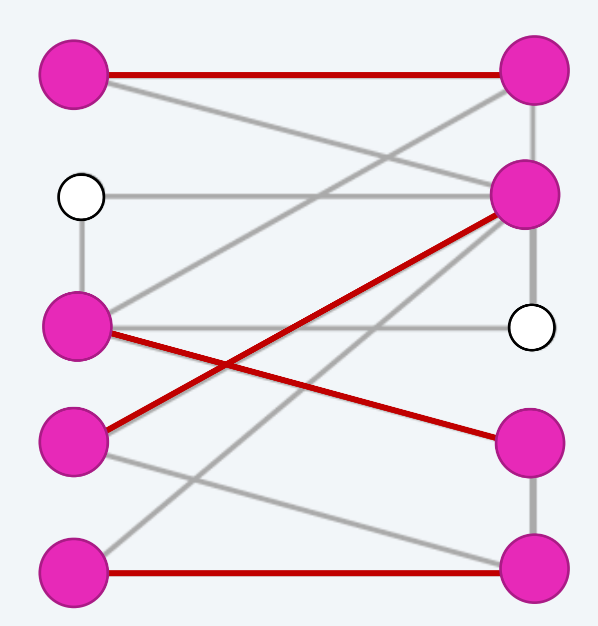
$\delta_{n-2} = n - 2$

$\delta_{n-1} = n - 1$

For $i = n, n-1, \cdots, 1$
- Each R_i has size $\left\lfloor \frac{n}{i} \right\rfloor$
- Each node in R_i is connected to i different nodes in L

Idea 2 will remove all the vertices of $R_1, \cdots, R_1$ and put them into the vertex-cover.

M



In any vertex cover, we have to select at least one endpoint of each edge in M.

Vertex Cover Problem

Algorithm 1.

- Pick an arbitrary edge.
- We know any vertex cover must have at least 1 endpoint of it, so let's take both endpoints.
- Then, throw out all edges covered and repeat until there are no uncovered edges left.

Algorithm 1: APPROX-VERTEX-COVER(G)

```
1 C←∅
2 while E ≠ ∅
      pick any {u, v} ∈ E
      C ← C ∪ {u, v}
      delete all eges incident to either u or v

  return C
```

Theorem. Algorithm 1 is 2-approximation algorithm.

Observation: The set of edges picked by, denoted by M, Algorithm 1 is a ***matching***, i.e., no 2 edges touch each other (edges disjoint).

Claim 1: This algorithm gives a vertex cover.

Claim 2: This vertex cover has size no more than twice of the minimum size (optimal solution).

Proof

- The optimum vertex cover must cover every edge in M. So, it must include at least one of the endpoints of each edge ∈ M, where no 2 edges in M share an endpoint.

$$OPT \geq |M|$$

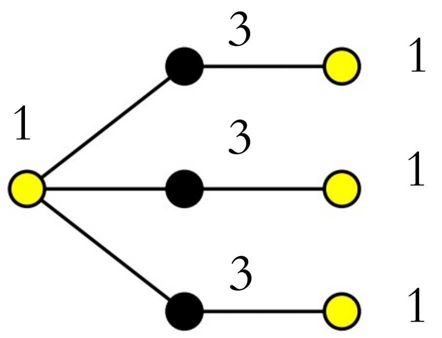
- But the algorithm A return a vertex cover of size $2|M|$, so for any instance I, we have

$$ALG = 2|M| \leq 2OPT$$

Weighted Vertex Cover Problem

Weighted Vertex-Cover.

Given a graph $G = (V, E)$ and each vertex $i \in V$ has a weight $w_i \geq 0$. Find a min-weight subset of vertices $S \subseteq V$ such that every edge is incident to at least one vertex in S.



If x^* is optimal solution to *ILP*, then $S = \{i \in V : x_i^* = 1\}$ is a min-weight vertex cover.

$$\begin{array}{llll} \min & \sum_{i \in V} w_i x_i & & \\ \text{s.t.} & x_i + x_j \geq 1 & & (i,j) \in E \\ & x_i \in \{0, 1\} & & i \in V \end{array}$$

Integer linear programming formulation

- Model inclusion of vertex i using a 0/1 variable x_i:

$$x_i = \begin{cases} 0 & \text{if vertex } i \text{ is not in vertex cover} \\ 1 & \text{if vertex } i \text{ is in vertex cover} \end{cases}$$

- Vertex covers is 1–1 correspondence with 0/1 assignments: $S = \{i \in V : x_i = 1\}$.
- Objective function:

$$\text{Minimize } \sum_i w_i \cdot x_i$$

- Constraints: For every edge (i, j), must take either vertex i or j (or both):

$$x_i + x_j \geq 1$$

Weighted Vertex Cover Problem

$$\begin{array}{llcll} \min & \sum_{i \in V} w_i x_i & & & \\ \text{s.t.} & x_i + x_j & \geq & 1 & (i,j) \in E \\ & \boxed{x_i \;\in\; \{0,\ 1\}} & & & i \in V \end{array}$$

Linear programming relaxation

$$\begin{array}{llcll} \min & \sum_{i \in V} w_i x_i & & & \\ \text{s.t.} & x_i + x_j & \geq & 1 & (i,j) \in E \\ & \boxed{x_i \;\geq\; 0} & & & i \in V \end{array}$$

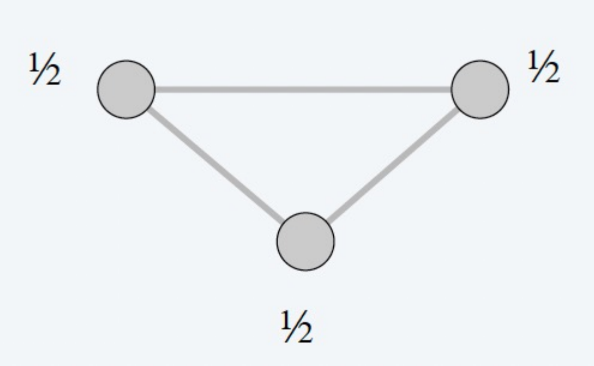
Lemma.

Optimal value of *LP* is ≤ optimal value of *ILP*.

$OPT_{LP} \leq OPT_{ILP}$

Proof. *LP* has fewer constraints.

LP solution x^* may not correspond to a vertex cover.
(even if all weights are 1)



How can solving *LP* help us find a low-weight vertex cover?

Solve *LP* and round fractional values in x^*.

LP Rounding Algorithm

Linear programming relaxation

$$\begin{array}{llcll} \min & \sum_{i\in V} w_i x_i & & & \\ \text{s.t.} & x_i + x_j & \geq & 1 & (i,j)\in E \\ & x_i & \geq & 0 & i \in V \end{array}$$

LP solution x^* may not correspond to a vertex cover.
(even if all weights are 1)

For any edge $(i,j)\in E$, $x_i^* + x_j^* \geq 1$,

$x_i^* \geq \frac{1}{2}$ or $x_j^* \geq \frac{1}{2}$ or both

***Theorem*.**
The rounding algorithm is a ***2-approximation*** algorithm.

Integer solution: $y_i = \begin{cases} 1, \text{ if } i \in S \text{ or } x_i^* \geq \frac{1}{2} \\ 0, \text{ if } i \notin S \text{ or } x_i^* < \frac{1}{2} \end{cases}$

Rounding

Given *LP* solution x^*, Set

$$S = \left\{ i \in V \,\middle|\, x_i^* \geq \frac{1}{2} \right\}.$$

Let $W(S) = \sum_{i\in S} w_i$.

***Claim*.** S is a vertex cover

***Proof*.** For any $(i,j)\in E$, at least one of i,j is in S.

***Claim*.** $W(S) \leq 2 \cdot OPT_{LP} = 2 \cdot \sum_{i\in V} x_i^* \cdot w_i$.

***Proof*.** $\sum_{i\in V} x_i^* \cdot w_i \geq \sum_{i\in S} x_i^* \cdot w_i \geq \frac{1}{2}\sum_{i\in S} w_i$.

SET COVER PROBLEM

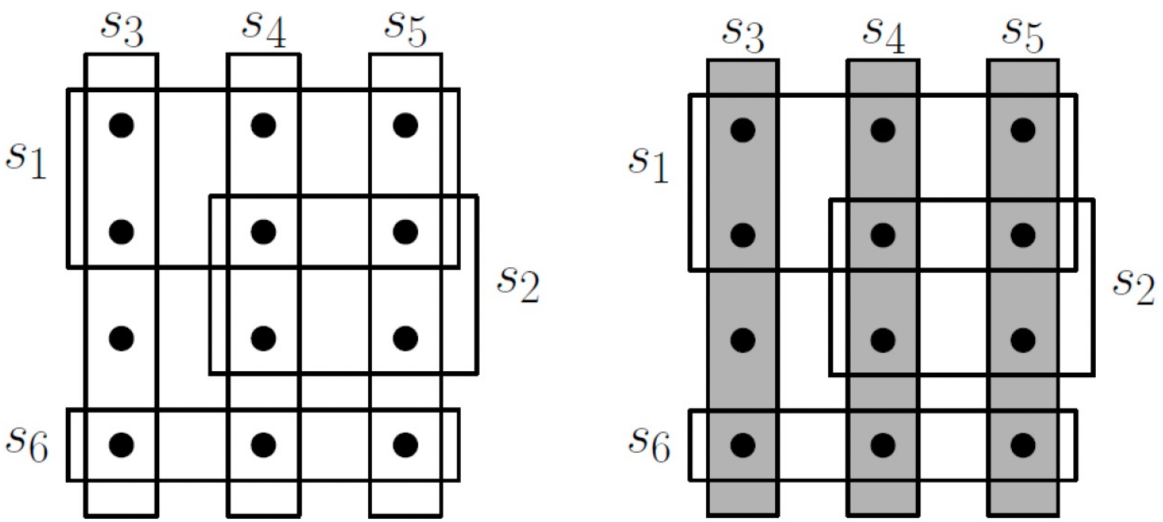
Set Cover Problem

SET-COVER.

Given a set X of elements (points), a collection S of subsets of X, and an integer k, are there $\leq k$ of these subsets whose union is equal to X?

FIND-MIN-SET-COVER.

Given a set X of elements (points), a collection S of subsets of X, find the fewest number of these subsets whose union is equal to X?



OPT=3

Greedy Algorithm

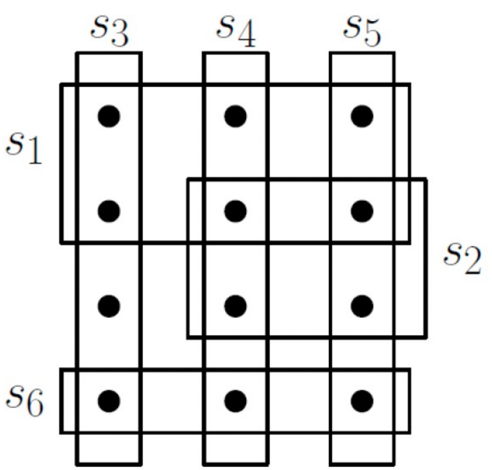
- Pick the set that covers the most points.
- Throw out all the points covered.
- Repeat.

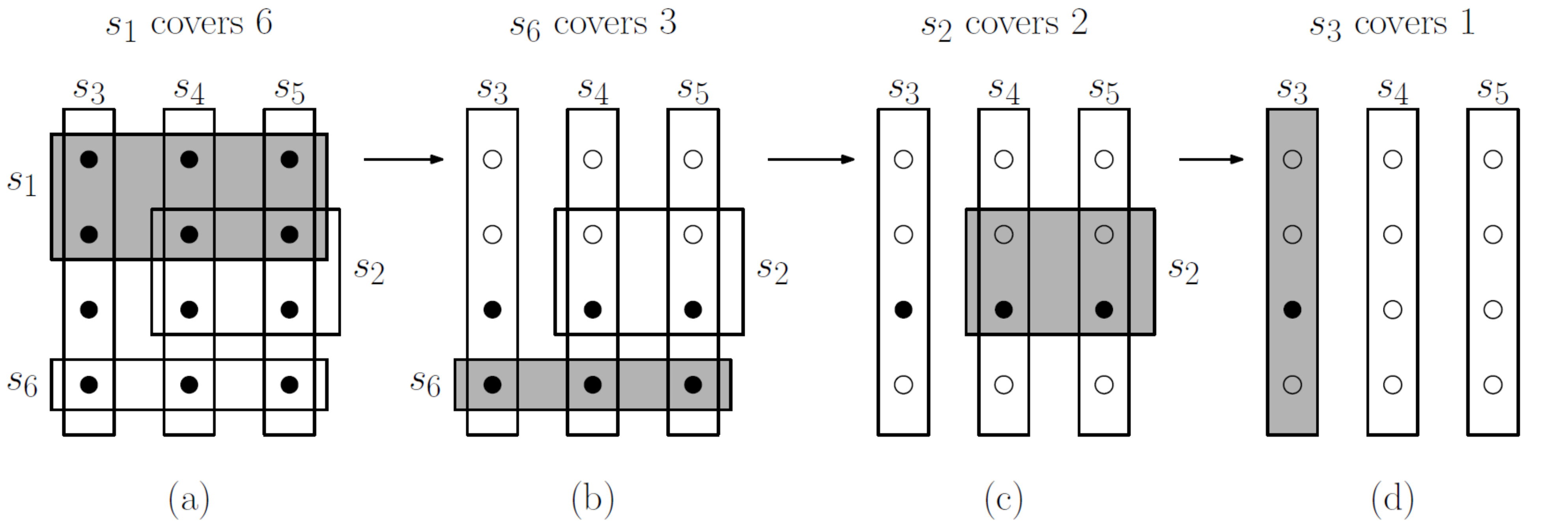
What is ALG for this instance?

Set Cover Problem

Greedy Algorithm ALG

- Pick the set that covers the most points.
- Throw out all the points covered.
- Repeat.





OPT=3

ALG=4

What is the approximation ratio?

Cannot be better than $\Omega(\log n)$!!!

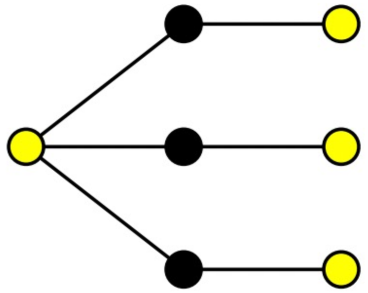
Vertex Cover Problem

Idea 1.

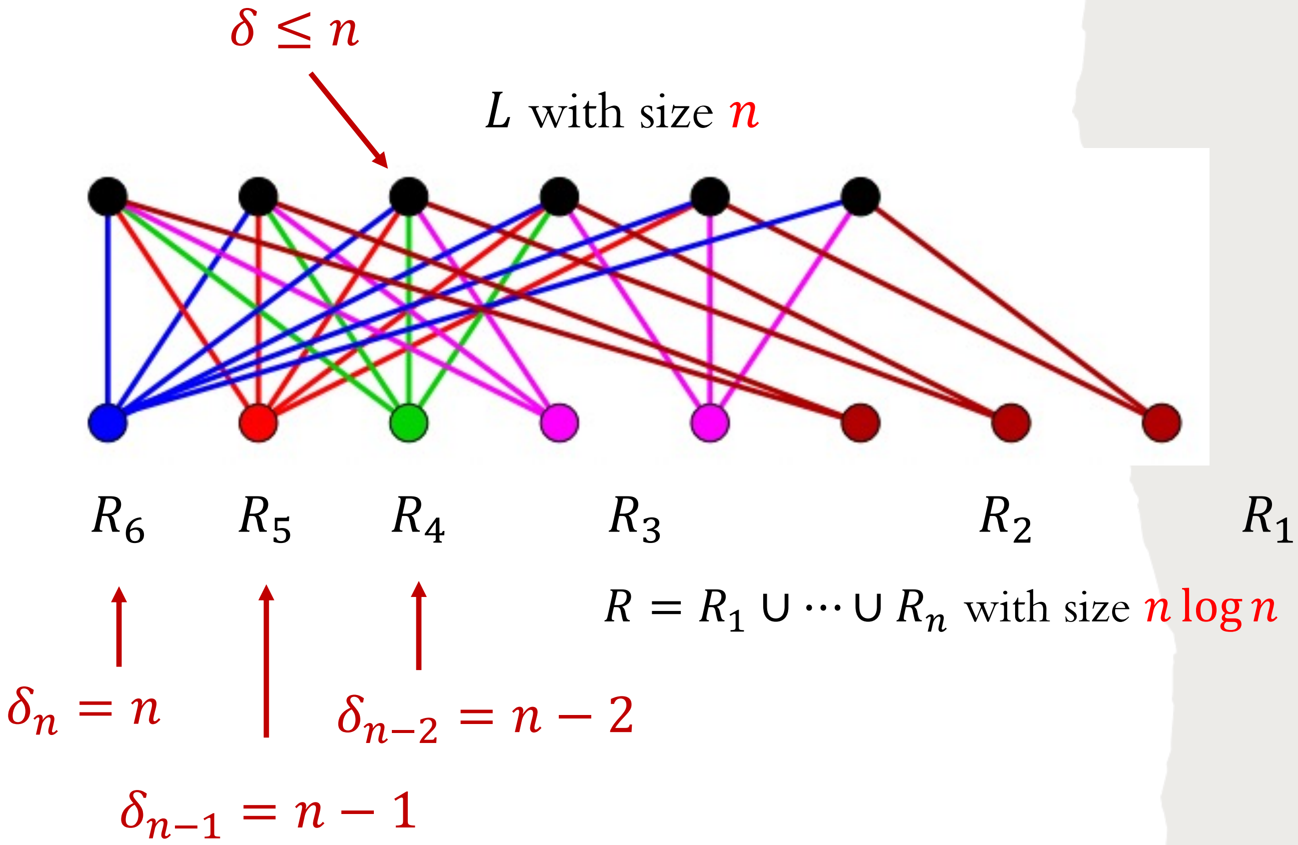
Pick an arbitrary vertex with at least one uncovered edge incident to it, put it into the cover, and repeat.

Idea 2.

How about picking the vertex that covers the most uncovered edges?



Cannot be better than $\Omega(\log n)$-approximation!



For $i = n, n-1, \cdots, 1$

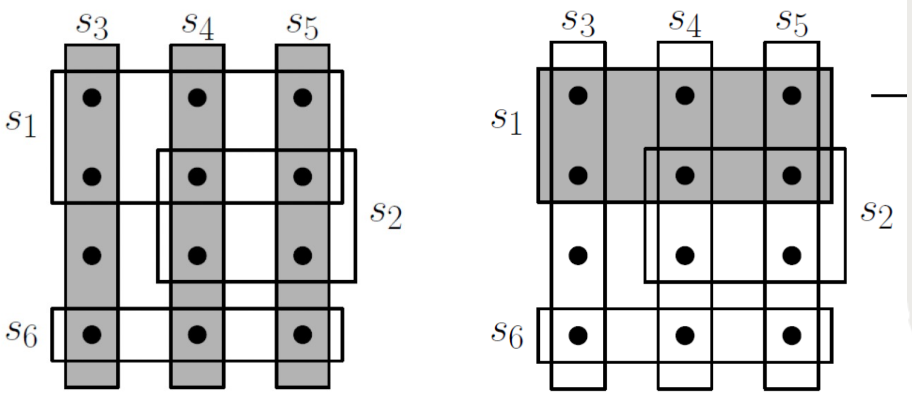
- Each R_i has size $\left\lfloor \frac{n}{i} \right\rfloor$
- Each node in R_i is connected to i different nodes in L

Idea 2 will remove all the vertices of $R_1, \cdots, R_1$ and put them into the vertex-cover.

Set Cover Problem

***Theorem*.**

If the optimal solution uses k sets, the greedy algorithm finds a solution with at most $k \cdot \ln n$ sets.

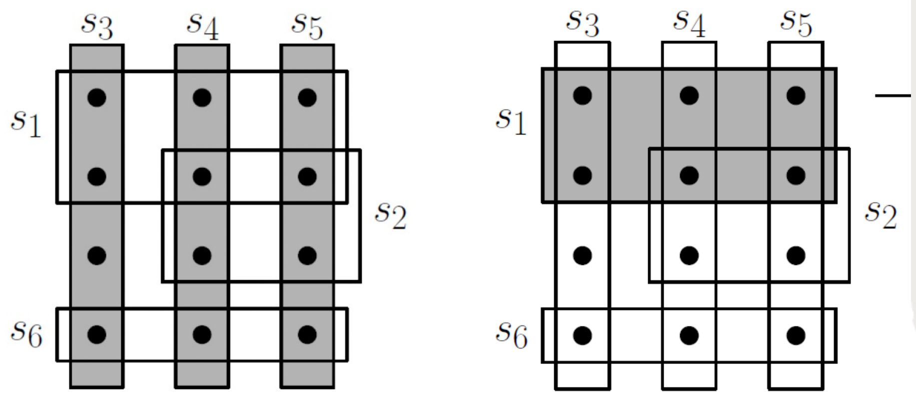


If lucky, we may choose one set in OPT and so there are actually $k-1$ sets covering the remainder, but we can't count on this.

Proof.

- Since $\mathrm{OPT} = k$, there must be a set that covers at least a $1/k$ fraction of the points.
- ***ALG*** chooses the set that covers the most points, so it covers at least $1/k$ fraction of the points.
- Therefore, after the first iteration, there are at most $n \cdot (1 - \frac{1}{k})$ points left.
- Again, since $\mathrm{OPT} = k$, there must be a set that covers at least a $1/k$ fraction of the remainder.
- So, again, since ***ALG*** chooses the set that covers the most points remaining, after the 2nd iteration, there are at most $n \cdot \left(1 - \frac{1}{k}\right)^2$ points left.

Set Cover Problem



***Theorem*.**

If the optimal solution uses k sets, the greedy algorithm finds a solution with at most $k \cdot \ln n$ sets.

Proof.

- More generally, after t rounds, there are at most $n \cdot \left(1 - \frac{1}{k}\right)^t$ points left.
- After $t = k \ln n$ rounds, there are at most

$$n \cdot \left(1 - \frac{1}{k}\right)^{k \ln n} < n \cdot \left(\frac{1}{e}\right)^{\ln n} = 1$$

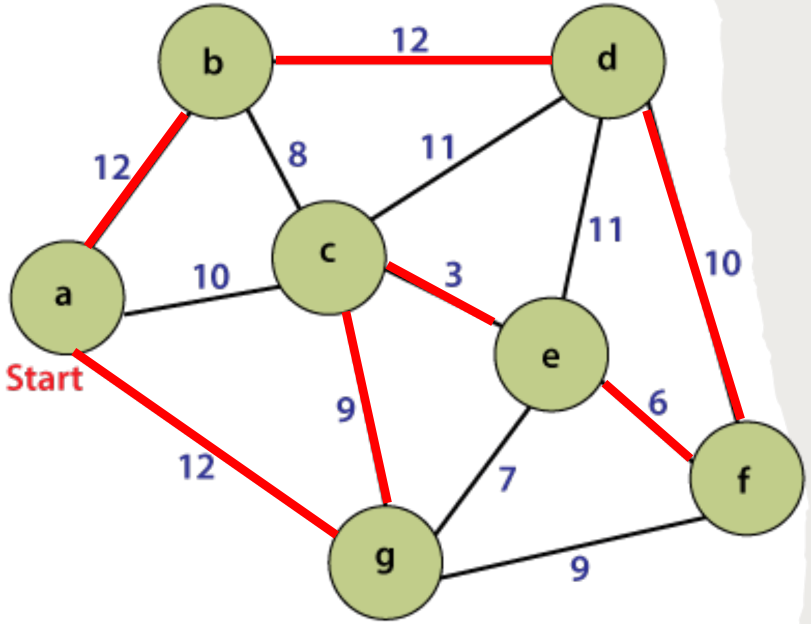
points left, which means we must be done.

TRAVELLING SALESMAN PROBLEM

Travelling Salesman Problem (TSP)

We are given n cities $1, \cdots, n$, and a nonnegative integer distance $l(i, j)$ between any two cities i and j (assume that the distances are symmetric, that is, $l(i, j) = l(j, i)$ for all i and j).

We are asked to find the shortest tour of the cities -- that is, the permutation π such that $\sum_{i=1}^{n} l(\pi(i), \pi(i+1))$ (where by $\pi(n+1)$ we mean $\pi(1)$) is as small as possible.



Removing any edge from any tour is a spanning tree!

The minimum one can be found in poly-time

TSP ≥ MST

A special case is when edge lengths to satisfy ***triangle inequality***

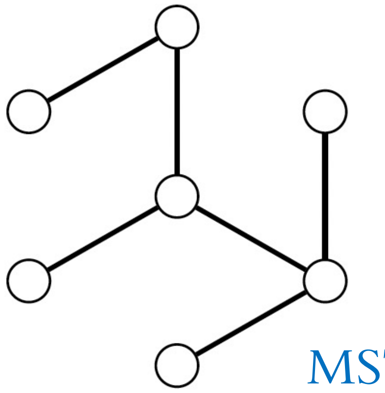
$$l(u, w) \le l(u, v) + l(v, w) \text{ for any vertices } u, v, w$$

$$l(u, w) \le l(u, v_1) + l(v_1, v_2) + \cdots + l(v_{k-1}, v_k) + l(v_k, w) \text{ for any vertices } u, v_1, \cdots, v_k, w$$

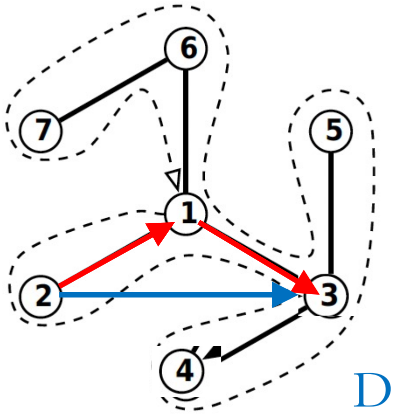
Travelling Salesman Problem (TSP)

Algorithm ALG

- Compute the minimum spanning tree (MST) T of G.
- Perform a depth-first search of T, numbering the vertices in the order that we first encounter them. (every vertex is numbered).
- Return the cycle obtained by visiting the vertices according to this numbering.

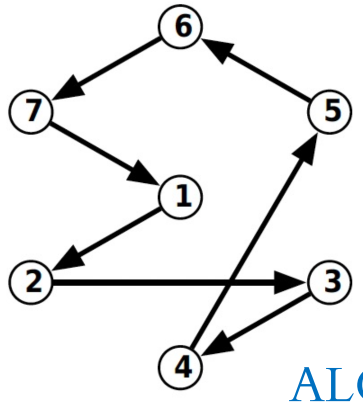


MST



DFS

moving directly from each node to the next unvisited node



ALG

Theorem. A depth-first ordering of the MST gives a 2-approximation of the shortest TSP tour.

- OPT: the cost of the optimal TSP tour
- MST: the total length of the MST
- ALG: the length of the tour by algorithm

$$OPT \geq MST$$

Removing any edge from OPT is a spanning tree

$$ALG \leq 2 \cdot MST$$

DFS traverses every edge in MST exactly twice = $2 \cdot MST$

Triangle inequality: $ALG \leq DFS = 2 \cdot MST$

SUMMARY

Summary

Algorithm Design Techniques

- ***Greedy***
  - E.g. Independent Set Problem
- ***Dynamic Programming***
  - E.g. Knapsack Problem
- ***Linear Programming (+ Rounding)***
  - E.g. Vertex Cover Problem
- ***Structures + Relationships with P problems***
  - E.g. Vertex Cover, Metric TSP
- ***Combination of Multiple Algorithms***
  - E.g. Knapsack Problem
- More like ***Divide-and-Conquer, Local Search, Reductions***

Tips

- Understand the structures of your algorithms
- Understand the difficulty of your algorithms

  NP-complete or P?

- Start from common/familiar techniques
- Prove your approximation ratios
- Is the approximation ratio good enough for your algorithms?

Use examples to prove your conjectures

Use examples to disprove your conjectures

- Improve your approximations