Approximation Algorithms for Graph Related Problems

Sandip Das
Indian Statistical Institute, Kolkata
sandipdas@isical.ac.in

NP-completeness

"I can't find an efficient algorithm, but neither can all these famous people."

Optimization Problem

An optimization problem Π is characterized by 3 components :

Instances *D*: a set of input instances.

Solutions *S(I)*: the set of all feasible solutions for an instance $I \in D$.
Value *f*: a function which assigns a value to each solutions.

A minimization problem is: given an instance, find a solution such that its value is minimum among all feasible solution.

Graph Coloring Problem

n vertices: v_1 v_2 v_3 … v_n

Colors: C_1 C_2 … C_k

Optimization: Minimum number of colors required to color the vertices so that adjacent vertices must be of different color

Coping With NP-Hardness

Brute-force algorithms.

- Develop clever enumeration strategies.
- Guaranteed to find optimal solution.
- No guarantees on running time.

Heuristics.

Develop intuitive algorithms.
Guaranteed to run in polynomial time.
No guarantees on quality of solution.

Coping With NP-Hardness

Approximation algorithms.

Guaranteed to run in polynomial time.
Guaranteed to find "high quality" solution.

How do we measure “high quality” solution?

Is solution ≅ Optimum solution ± some constant?
(1 ± some small constant)*Optimum solution?

Obstacle: need to prove a solution’s value is close to optimum, without even knowing what optimum value is!

Performance guarantees

Suppose *A(n)* is the solution of a non optimal algorithm for problem *P* of size *n*.

In case

$|Optimum\ solution - A(n)| <= Some\ constant$

then *A(n)* is an *absolute approximation algorithms*

Performance ratio

$R_A(n) = A(n)/OPT(n)$ in case *P* is a minimization problem

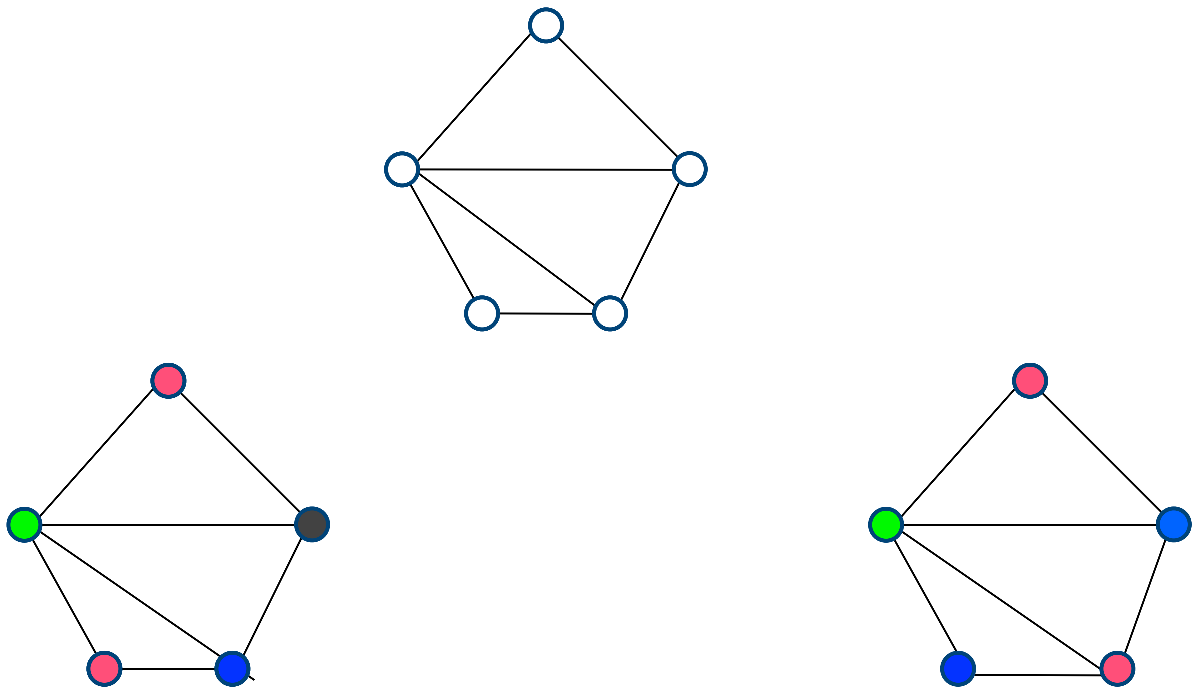
$= OPT(n)/A(n)$ in case *P* is a maximization problem

$R_A(n)$ may be *Constant*, *log n* or any other function

Absolute Approximation Algorithms

Problem: Coloring of the vertices of a graph such that no two adjacent vertices have the same color

Goal: minimize the number of color used.



Absolute Approximation Algorithms

Problem: Coloring of the vertices of a graph such that no two adjacent vertices have the same color

Goal: minimize the number of color used

Decision version: Given some integer k,
is it possible to color the vertices with k colors

Absolute Approximation Algorithms

Problem: Coloring of the vertices of a graph such that no two adjacent vertices have the same color

Goal: minimize the number of color used

Decision version of this problem is NP-hard even if the graph is planar

The problem of deciding whether a planar graph is 3-colorable is NP-complete

It is well-known that any planar graph is 5-colorable

$\Rightarrow$ The performance of the approximation algorithm *A* is such that $|A(G)-OPT(G)| \leq 2$

Vertex Cover

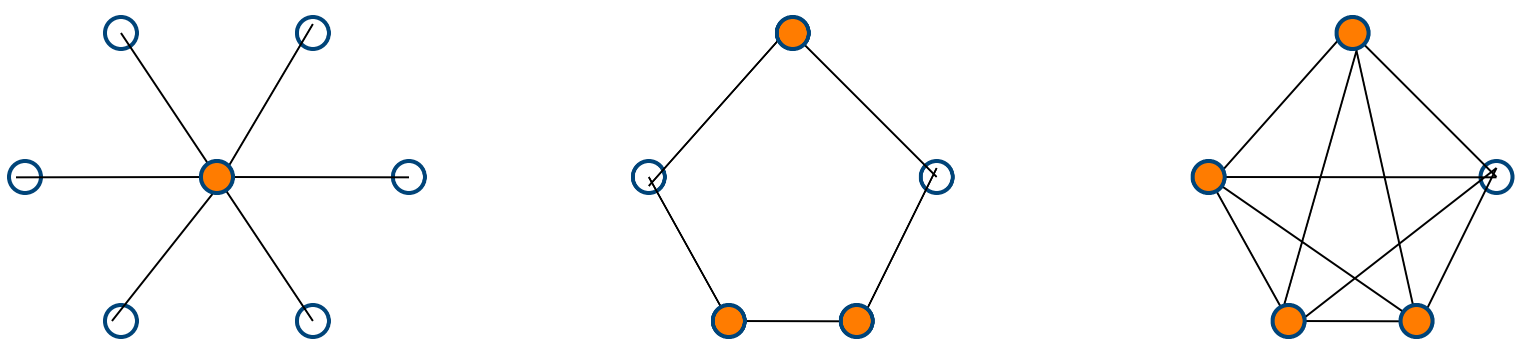
Vertex cover: a subset of vertices which "covers" every edge.
An edge is covered if one of its endpoint is chosen.

The Minimum Vertex Cover Problem:
Find a vertex cover with minimum number of vertices.

Vertex Cover

Vertex cover: a subset of vertices which “covers” every edge.
An edge is covered if one of its endpoint is chosen.

The Minimum Vertex Cover Problem:
Find a vertex cover with minimum number of vertices.



Approximation Algorithms

Key: provably close to optimal.

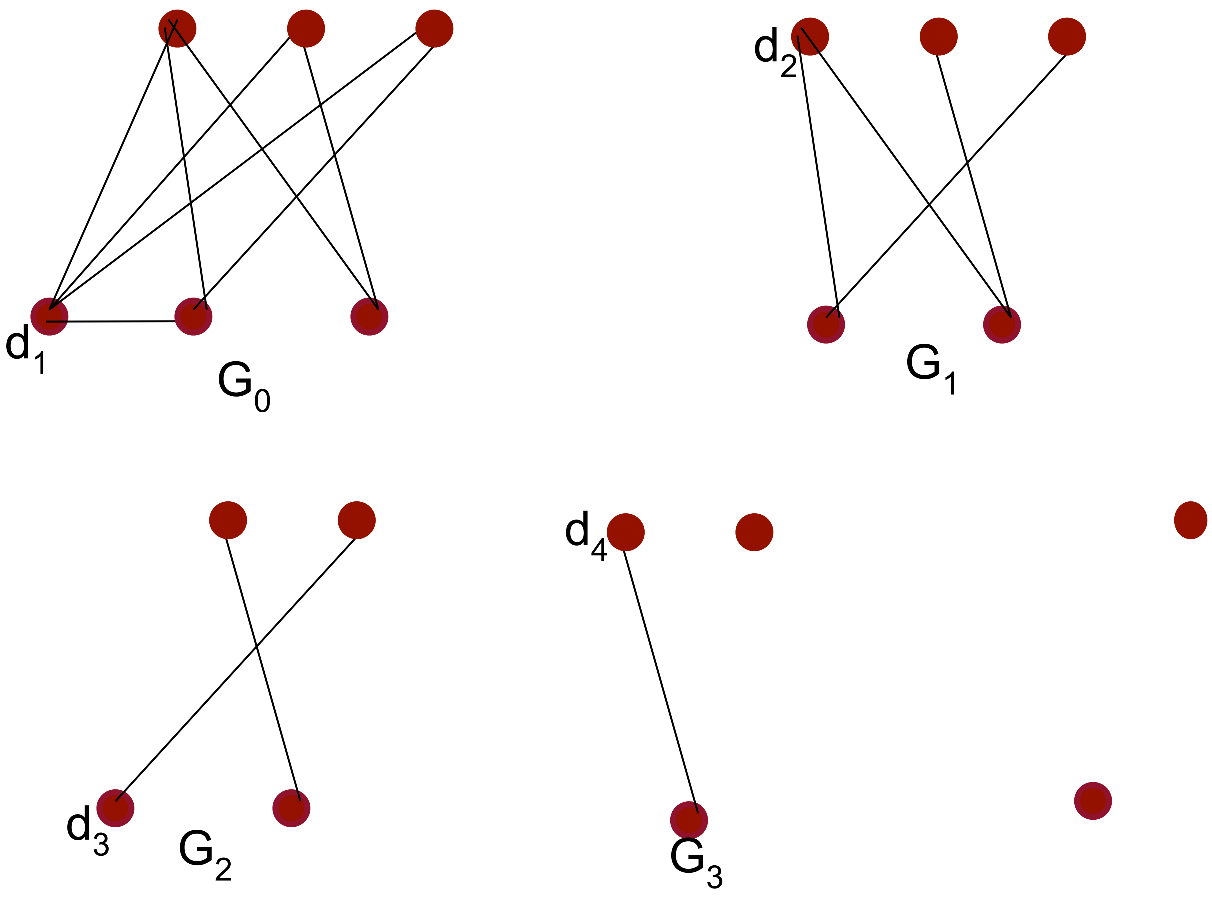
Let OPT be the value of an optimal solution,
and let SOL be the value of the solution that our algorithm returned.

Constant factor approximation algorithms:

SOL <= c.OPT for some constant c.

Vertex Cover: Greedy Algorithm 1

Idea: Keep finding a vertex which covers the maximum number of edges.



Vertex Cover: Greedy Algorithm 1

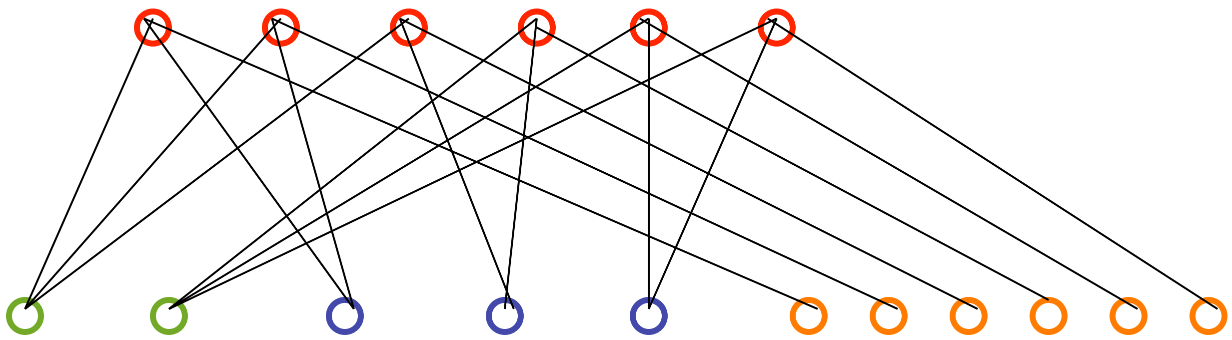
Idea: Keep finding a vertex which covers the maximum number of edges.

Greedy Algorithm 1:

1. Find a vertex *v* with maximum degree.
2. Add *v* to the solution and remove *v* and all its incident edges from the graph.
3. Repeat until all the edges are covered.

How good is this algorithm?

Vertex Cover: Greedy Algorithm 1



OPT = 6, all red vertices.

SOL = 11, if we are unlucky in breaking ties.

First we might choose all the green vertices.

Then we might choose all the blue vertices.

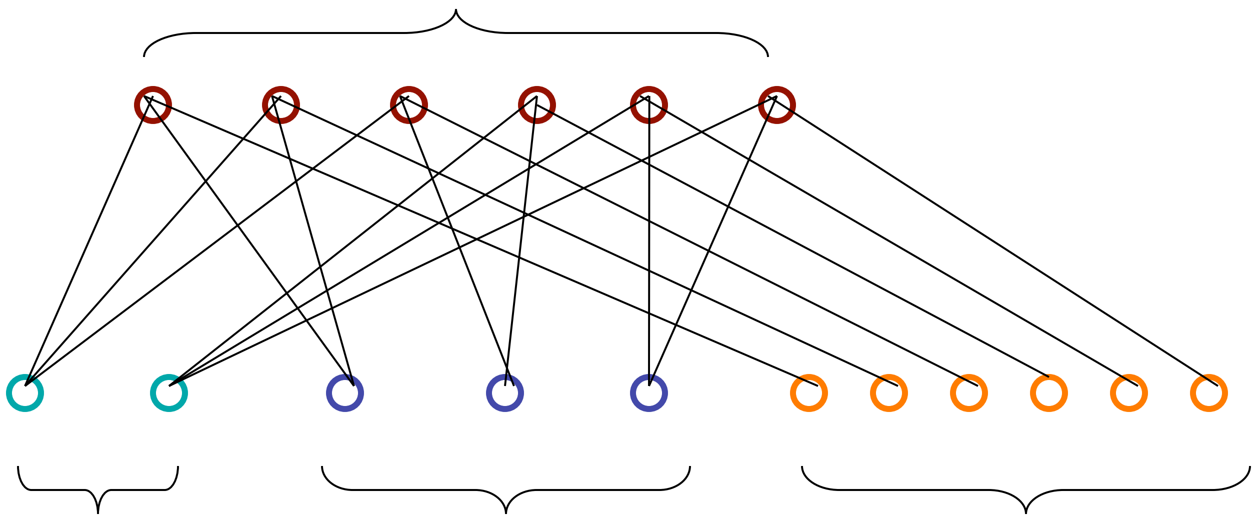
And then we might choose all the orange vertices.

Vertex Cover: Greedy Algorithm 1

Not a constant factor approximation algorithm!

k! vertices of degree k

Generalizing the example!



k!/k vertices of degree k

k!/(k-1) vertices of degree k-1

k! vertices of degree 1

OPT = k!, all top vertices.

SOL = k! (1/k + 1/(k-1) + 1/(k-2) + … + 1) ≈ **k! log(k)**, all bottom vertices.

Vertex Cover: Greedy Algorithm 1

Is the output from this greedy algorithm give an approximation of optimal solution?

From last example we can claim that if it is an approximation algorithm then approximation factor is not better than *O(log n)*

Vertex Cover: Greedy Algorithm 1

Consider	G_i	: remaining graph after the choice of i^{th} vertex in the solution
	d_i	: maximum degree of any node in G_{i-1}
	v_i	: vertex in G_{i-1} with maximum degree

Let C^* denote the optimal vertex cover of G which contain m number of vertices

$|G_{i-1}|$ denote the number of edges in the graph G_{i-1}.

Vertex Cover: Greedy Algorithm 1

$\sum_{v \in c^*} deg(v) \geq |G|$ *and* $|C^*| = m$

Hence,

$$Max_{\, v \in c^*} (deg(v)) \geq |G| / m$$

That is, $d_1 \geq |G_0| / m$

Similarly, $d_2 \geq |G_1| / m, \ \ldots$

Vertex Cover: Greedy Algorithm 1

Then

$$\sum^{m}_{i=1} d_i \geq \sum^{m}_{i=1} |G_{i-1}| /m \qquad As \sum_{v \in c^*} deg_{G_{i-1}}(v) \geq |G_{i-1}|$$

$$\geq \sum^{m}_{i=1} |G_m| /m$$

$$=|G_m|$$

$$\geq |G| - \sum^{m}_{i=1} d_i$$

So, $\sum^{m}_{i=1} d_i \geq |G| /2$

Vertex Cover: Greedy Algorithm 1

In *m* th iterations, algorithm removes at least half the edges of *G*

Thus

after *m*.log |*G*| iterations

all the edges of *G* have been removed

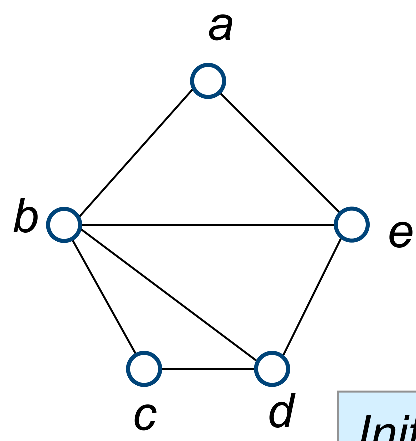
Algorithm 1 computes a vertex cover of size *O(optimum. log n)*

Greedy Algorithm 1 is an *O(log n) approximation algorithm*

Vertex Cover: Algorithm 2

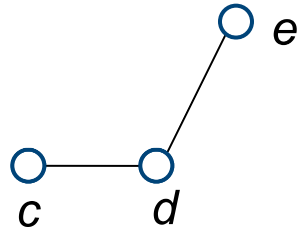
Greedy approach does not always lead to the best approximation algorithm

Vertex Cover: Algorithm 2



Initially C = ϕ
Consider an edge (a,b)
Put "a" and "b" in C, i.e., C = {a,b}

Remove vertices "*a*" and "*b*" frpm the graph

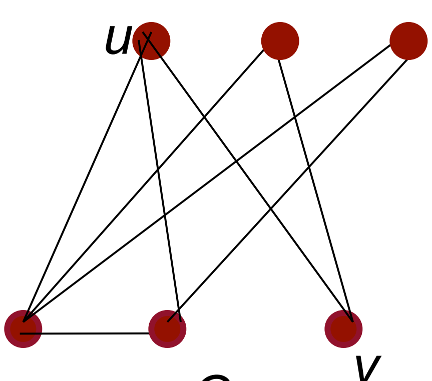


Vertex Cover: Algorithm 2

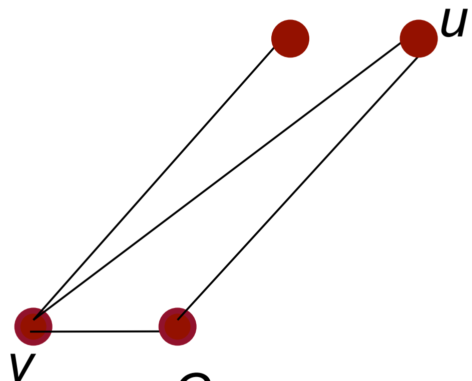
$C = \phi$
while G has atleast one edge
{ (u,v) any edge of G
$G = G \setminus \{u, v\}$
$C = C \cup \{u, v\}$ }
return C

How good is this algorithm?

Vertex Cover: Algorithm 2



G



G_1

G_2

Vertex Cover: Algorithm 2

For edge *(u, v)*, at least one of the vertex *u* or *v* must be in any optimal cover

IT FOLLOWS IT IS A 2 APPROXIMATION ALGORITHM

Conclusion: Greedy approach does not always lead to the best approximation algorithm

Traveling Salesman

Traveling salesman problem

asks for the shortest Hamiltonian cycle in a weighted undirected graph.

Traveling salesman problem is NP hard

Traveling Salesman : A Special Case

Edge lengths satisfy triangular inequality

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

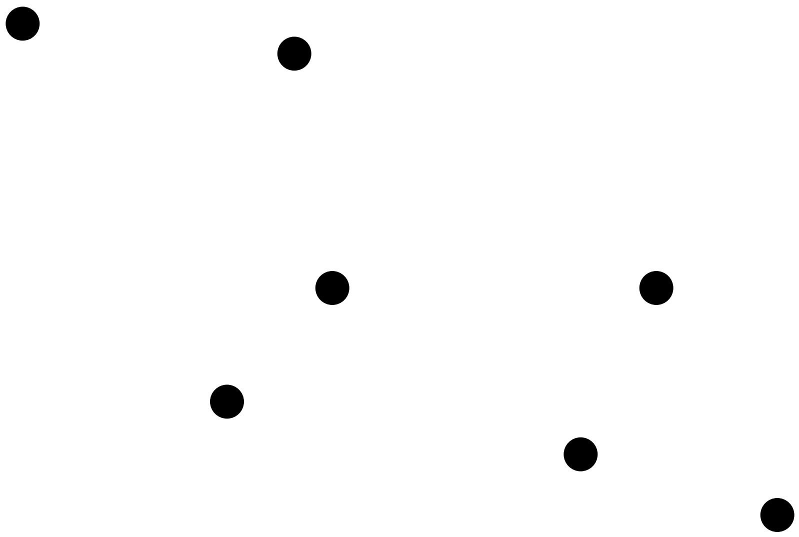
This is true for geometric graph

Consider the following algorithm :

Compute minimum spanning tree *T* of the weighted input graph
Depth first traversal of *T*
Numbering the vertices in order that we first encounter them
Return the cycle obtained by visiting the vertices according to this numbering

Traveling Salesman : A Special Case

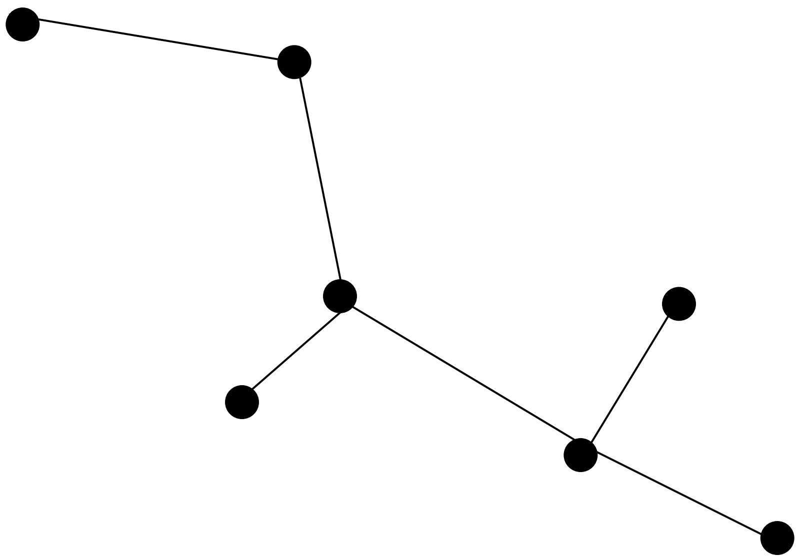
Demonstration



Set of points distributed in 2D

Traveling Salesman : A Special Case

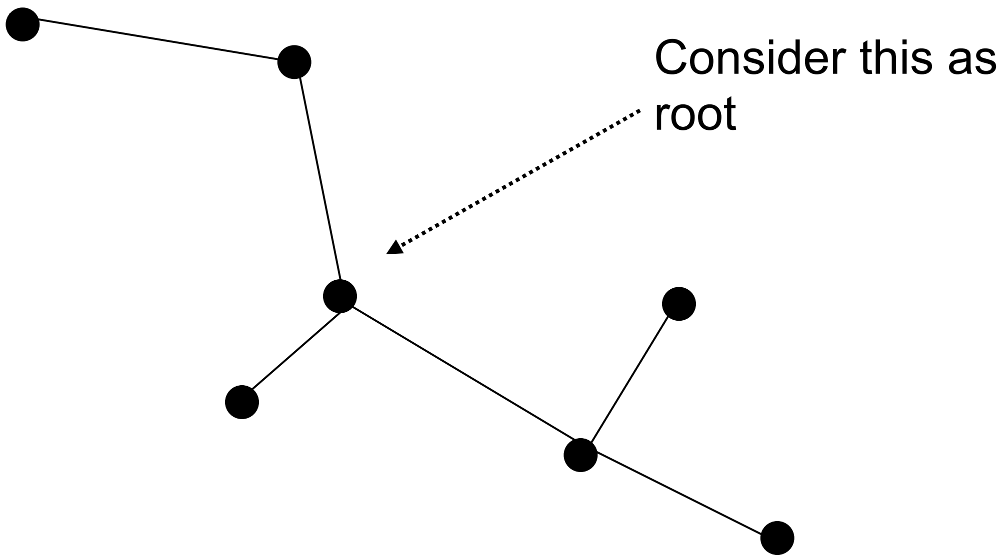
Demonstration



Minimum spanning tree

Traveling Salesman : A Special Case

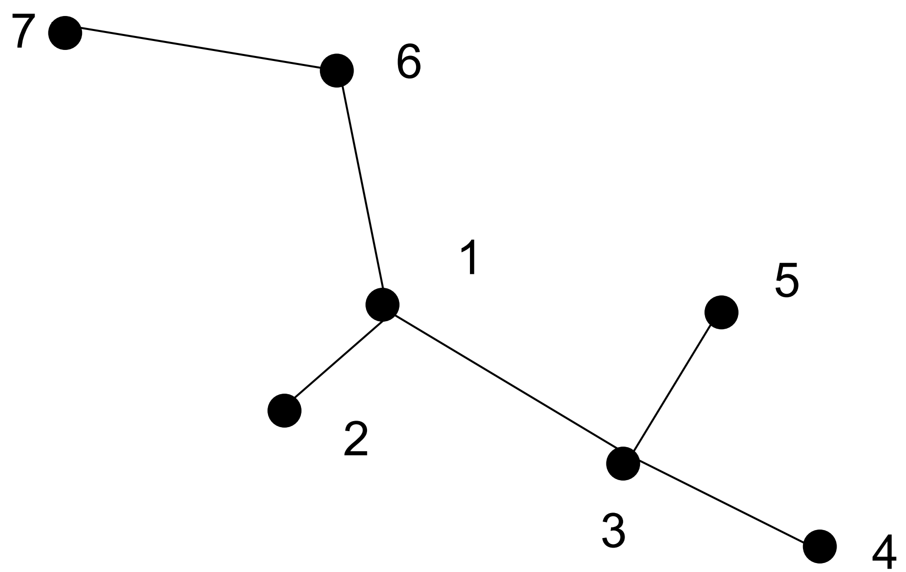
Demonstration



Depth first traversal

Traveling Salesman : A Special Case

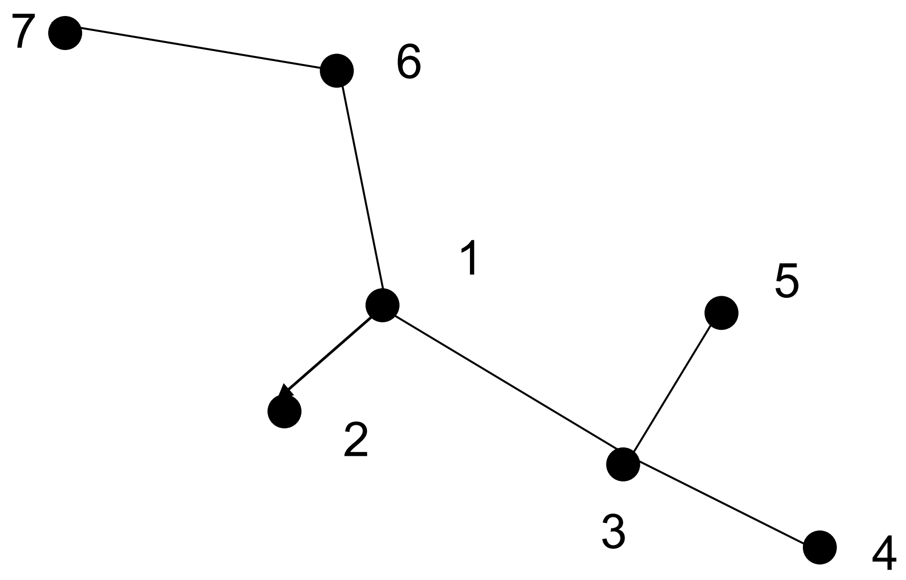
Demonstration



Depth first traversal and
numbering of vertices

Traveling Salesman : A Special Case

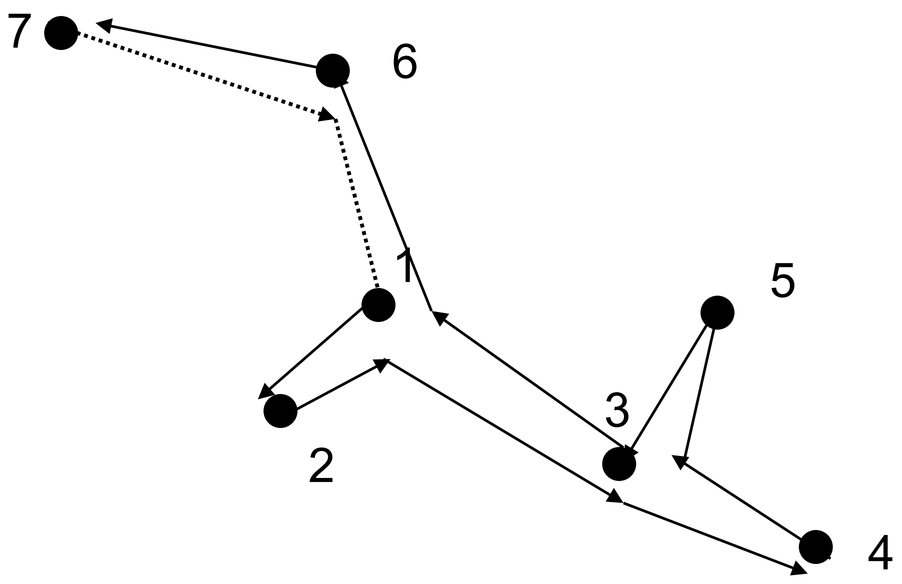
Demonstration



Traveling salesman tour

Traveling Salesman : A Special Case

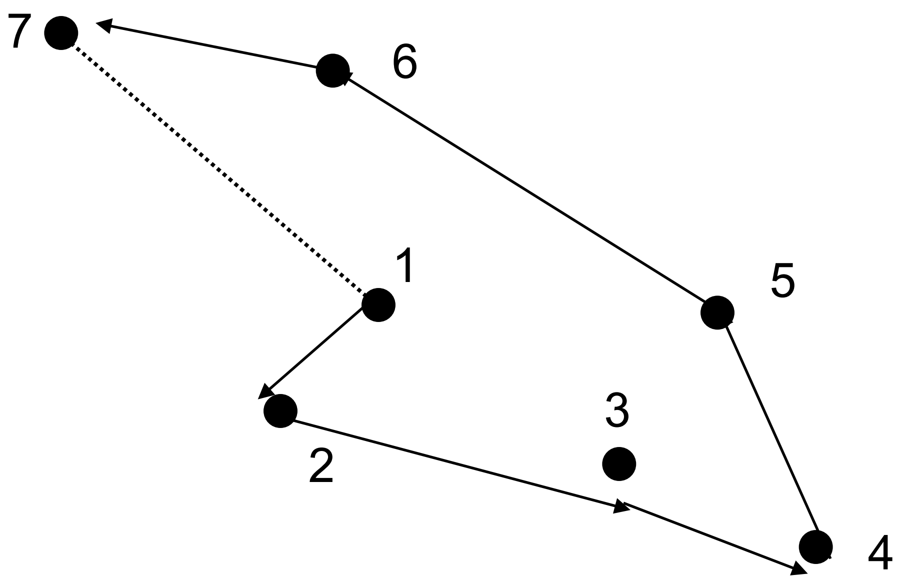
Demonstration



Traveling salesman tour
with cost 2.*MST*

Traveling Salesman : A Special Case

Demonstration



Traveling salesman tour
with reduced cost ≤ *2.MST*

Traveling Salesman : A Special Case

Output quality :

Cost of the tour using this algorithm

≤ 2* cost of minimum spanning tree

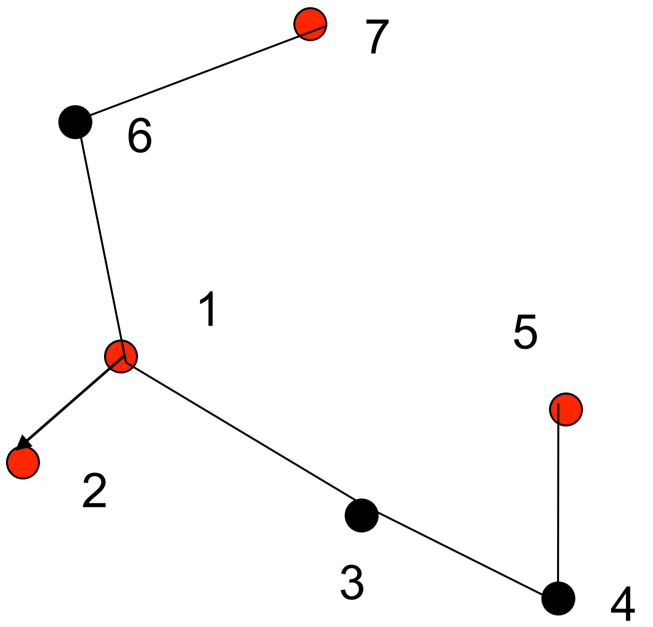
≤ 2* cost of optimal solution

Conclusion: The algorithm outputs 2 approximation of the minimum traveling salesman problem

Traveling Salesman : A Improved heuristic

Christofides 1976

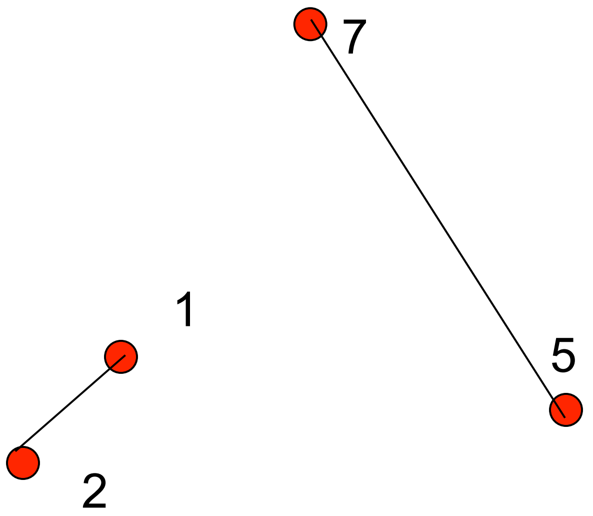
Locate odd degree vertices in minimum spanning tree



Number of odd degree vertices is even

Traveling Salesman : A Improved heuristic

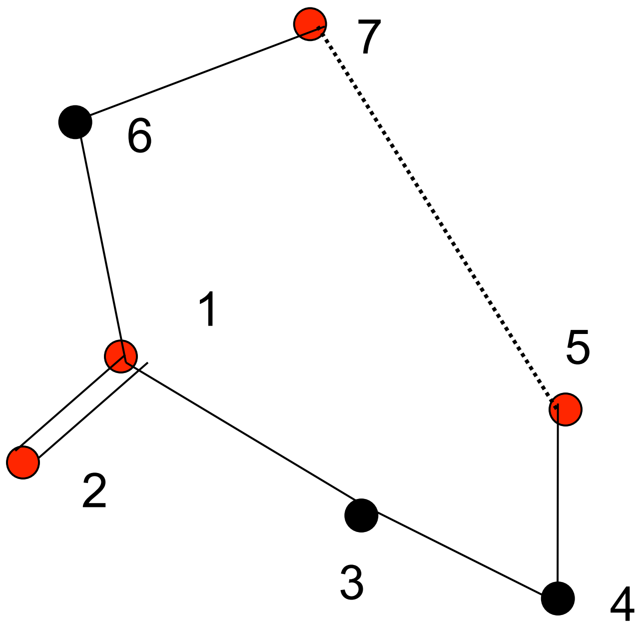
Compute a *minimum cost perfect matching* of these odd degree vertices



Perfect matching of odd degree vertices

Traveling Salesman : A Improved heuristic

Merge the perfect matching with minimum spanning tree allowing multi edges



Merging the perfect edges with MST

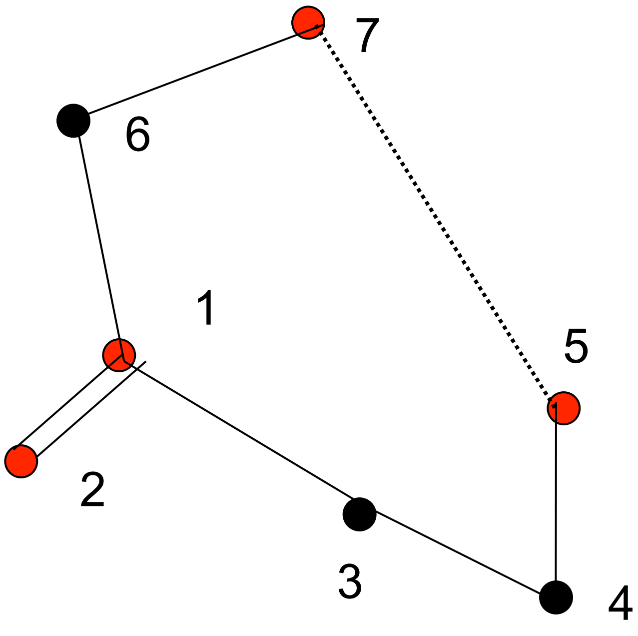
Traveling Salesman : A Improved heuristic

Observations:

In this new multigraph, every vertex has even degree

Thus it contains an *Eulerian circuit*

In *O(n)* time we can compute a closed walk that uses every edge exactly once

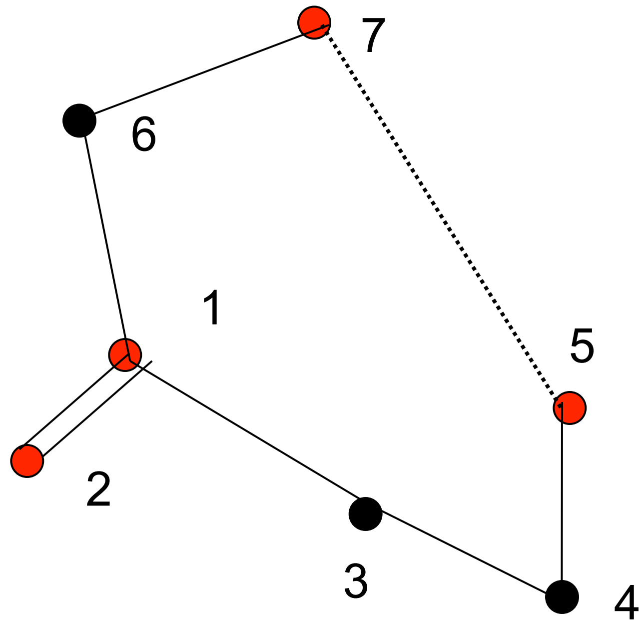


Traveling Salesman : A Improved heuristic

Observations:

Cost of the tour =
cost of minimum spanning tree +
cost of minimum odd vertex matching

Cost of minimum odd vertex matching ≤
½ * optimal traveling salesman tour



Result: *Given a weighted graph that obeys triangular inequality, the Christofides heuristic computes a (3/2)-approximation of the minimum traveling salesman tour*

Traveling Salesman

Consider G be an arbitrary undirected graph with *n* vertices

Length function $l(e) = \begin{cases} 1 & \text{if } e \text{ is an edge in } G \\ 2 & \text{otherwise} \end{cases}$ for K_n

G has a Hamiltonian cycle then
there is an Hamiltonian cycle in K_n whose length is exactly *n*

Traveling salesman problem is NP hard even if all the edge lengths are 1 or 2

Due to polynomial time reduction from Hamiltonian cycle to this type of Traveling salesman problem

Traveling Salesman

We can replace the values in length function by any values we like

$$\text{Length function} \qquad l(e) = \begin{cases} 1 & \text{if } e \text{ is an edge in } G \\ n & \text{otherwise} \end{cases}$$

G has a Hamiltonian cycle then
there is an Hamiltonian cycle in K_n whose length is exactly *n* or has length at least *2n*

Thus if we can approximate

the shortest traveling salesman tour within a factor of 2 in polynomial time

we would have a polynomial time algorithm for the Hamiltonian cycle problem

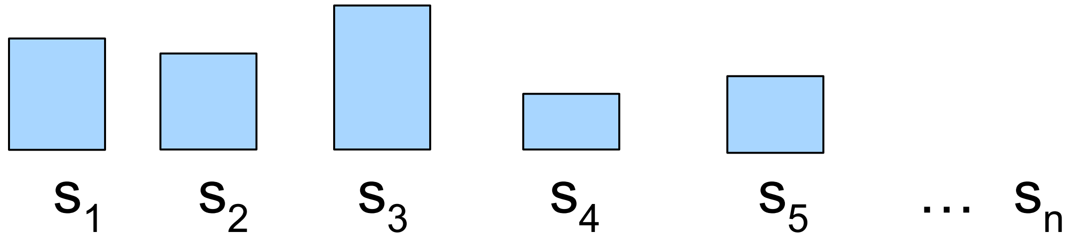
Traveling Salesman

We have the following negative results

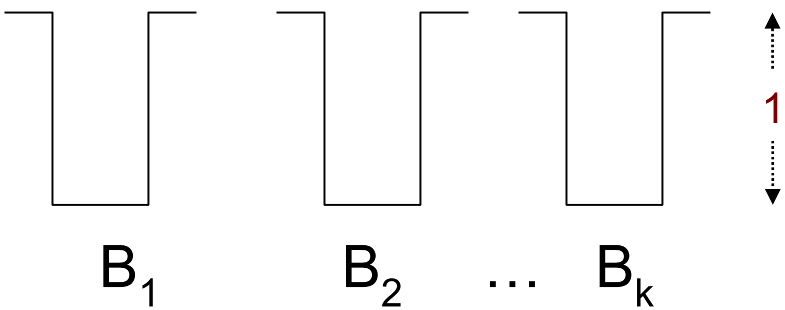
For any function *f(n)* that can be computed in polynomial in *n,* there is no polynomial time *f(n)* approx fo TSP on general weighted graph unless P=NP.

Bin Packing

n items:



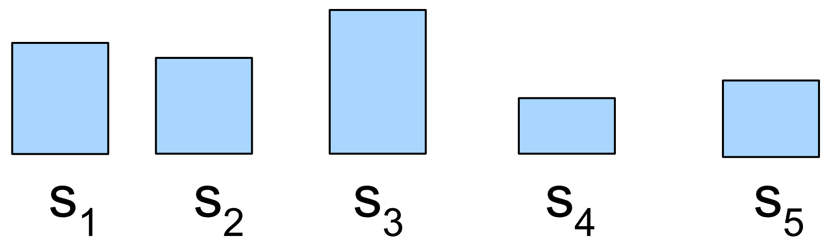
Bins:



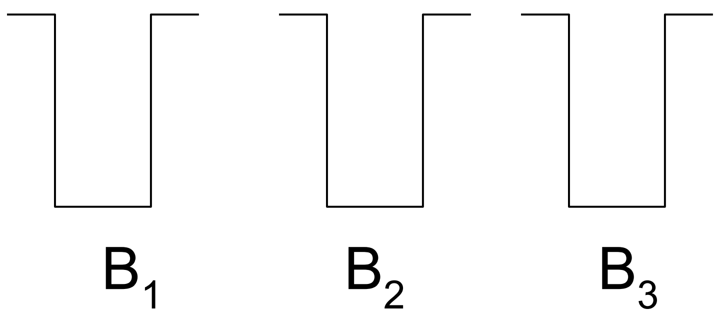
Optimization: Pack items in minimum number of bins

Bin Packing: First fit

5 items:



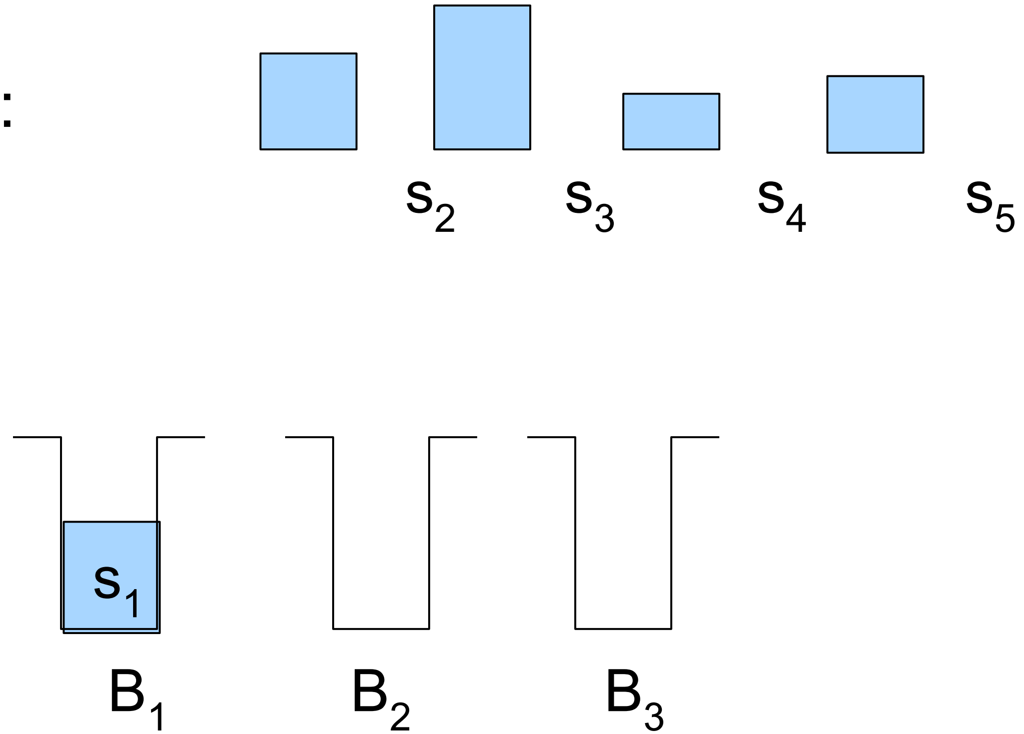
Bins:



Bin Packing: First fit

5 items:

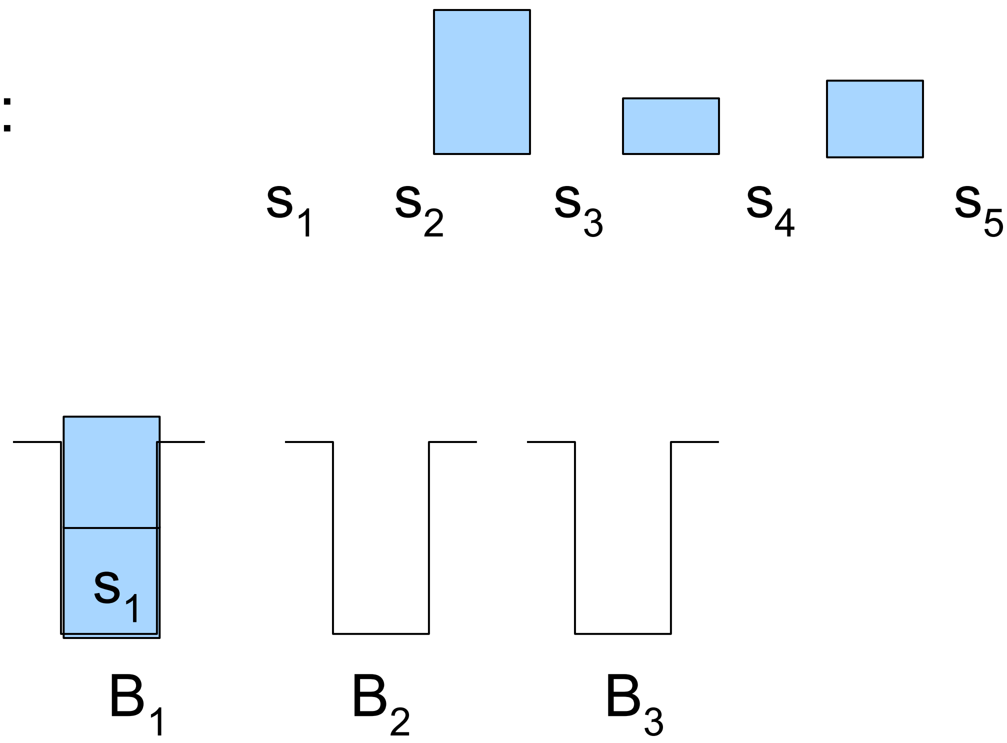
Bins:



Bin Packing: First fit

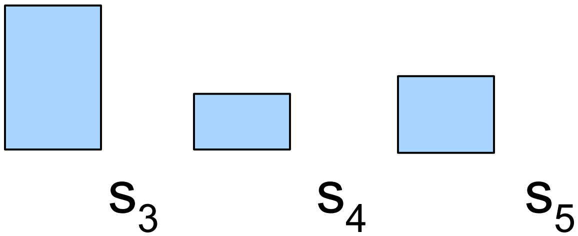
5 items:

Bins:

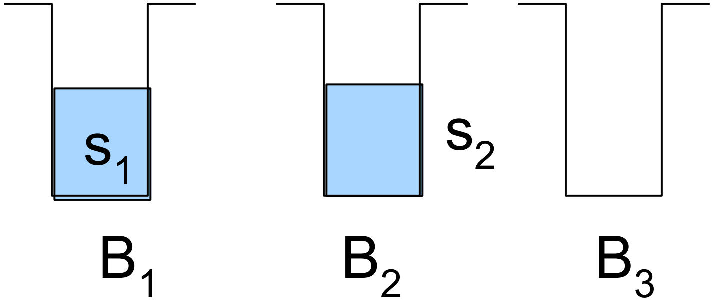


Bin Packing: First fit

5 items:

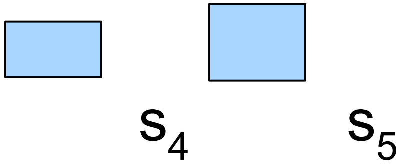


Bins:

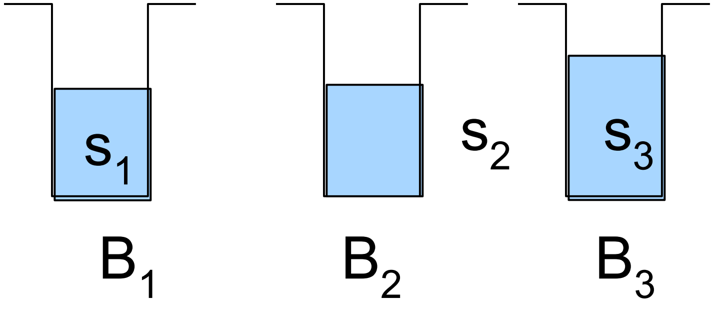


Bin Packing: First fit

5 items:



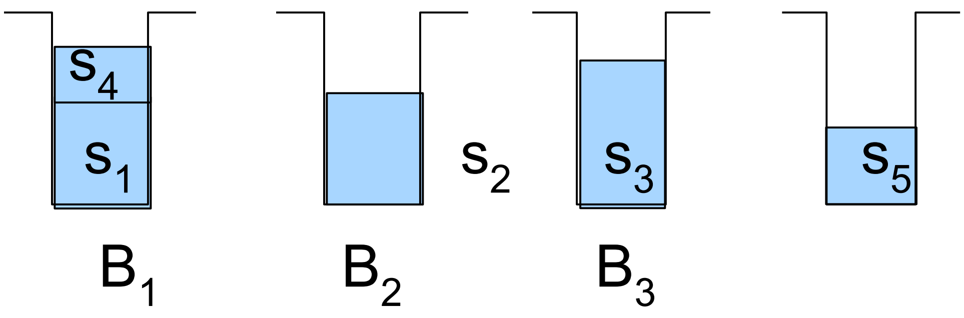
Bins:



Bin Packing: First fit

5 items:

Bins:



Bin Packing: First fit

How good is this algorithm?

Claim: The number of bins < $\lceil 2 * \Sigma_i \text{size of item } s_i \rceil$

Observation: at most one bin is more than half empty

Conclusion: Approximation factor less than equal to 2

Bin Packing: First fit

Is approximation factor less than 5/3 ?

Consider

6m items of size 1/7 + 0.0001
6m items of size 1/3 + 0.0001
6m items of size 1/2 + 0.0001

First fit will distribute the items as

$$\left.\begin{array}{l}\text{m bins with 6 items of size 1/7 + 0.0001 each}\\ \text{3m bins with 2 items of size 1/3 + 0.0001 each}\\ \text{6m bins with 1 items of size 1/2 + 0.0001 each}\end{array}\right\}\ \text{10m}$$

It can be placed in 6m bins

Bin Packing:

Any better algorithm than *first fit*?

May be best fit, first fit decreasing, best fit decreasing, ...

How much good are they?

Is any algorithm having a guarantee of $3/2 - \varepsilon$ approximation?

If such an algorithm exists

It will report optimal solution in case optimal solution is one or two.

IS IT?

Bin Packing:

Consider the partitioning problem:

n nonnegative numbers $a_1, a_2, a_3, \ldots, a_n$

decide whether there exist a partition into two sets each adding up to $\frac{1}{2}\sum a_i$

For example,

11 non-negative integers

6,9,15,12,8,16, 12, 19, 23, 12, 22

Total 154

decide whether there exist a partition into two sets each adding up to 77

Bin Packing:

Consider the partitioning problem:

n nonnegative numbers $a_1, a_2, a_3, \ldots, a_n$

decide whether there exist a partition into two sets each adding up to $\frac{1}{2}\sum a_i$

Can be mapped into bin packing problem with bin size $\frac{1}{2}\sum a_i$

Any 3/2 -ε approximation algorithm solve this NP-hard problem!!!

Idea:

If we allow to the high degree polynomial complexity of our approximation algorithm,
can we get better approximation bound?

Sometimes it may be possible with polynomial time complexity

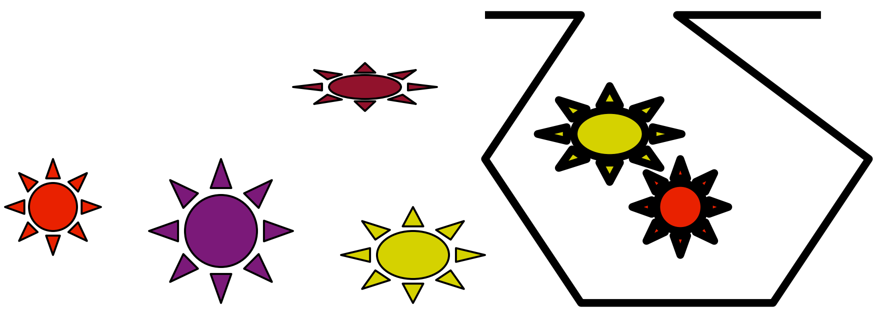
Thank You

Knapsack Problem

Items: $U=\{u_1, u_2, u_3, ..., u_n\}$ and u_i has size s_i with profit p_i
Capacity of knapsack: B

Solution: Choose subset U' of U s.t $\sum_{ui \in U'} s_i \leq B$

Objective: maximize the net profit $\sum_{ui \in U'} p_i$



Knapsack Problem

Here $OPT(I) >= Approx(I)$

Looking for $(1- \varepsilon)OPT(I) <= Approx(I)$

In case $\varepsilon=0$, then our approx algorithm is optimum

But complexity?

Let us consider $\varepsilon=0.0001$ then what wil be the complexity?

What if we vary ε?

Knapsack Problem: Greedy Algorithm

Profit density : p_i/s_i

1. Sort items in non-increasing order of their profit densities

2. $U' = \emptyset$

3. for $i = 1, 2, ..., n$

   If $\sum_{u_j \in U'} s_j \le B - s_i$ then $U' = U' + u_i$

But it does not do well

Knapsack Problem: polynomial approximation scheme

Choose a subset S of at most k elements

Run greedy algorithm using the remaining items

Repeat the process for all possible choice of k-set S

Complexity : $O(n^{k+1} \log n)$

Knapsack Problem: polynomial approximation scheme

All possible permutation of size k or small

$\{u'_1, u'_2, u'_3, \ldots, u'_k\}$

Items: $U=\{u_1, u_2, u_3, \ldots, u_n\}$

$\{u'_1, u'_2, u'_3, \ldots, u'_k\}$

Choose one permutation

Consider size of bin B = original size $- \sum_{ui \in U'} s_i$

Run greedy algorithm with this bin size and with remaining items.

Knapsack Problem: polynomial approximation scheme

How good is this algorithm?

Let one of the optimal solution is $X=\{u_1, u_2, u_3, \ldots, u_r\}$

If $r \le k$?

Suppose r>k

Let Optimal solution = $\{u'_1, u'_2, u'_3, \ldots, u'_k\} + \{u'_{k+1}, u'_{k+2}, \ldots, u'_r\}$

Items with larger profits in *X*

Execute greedy algorithm

Knapsack Problem: polynomial approximation scheme

Optimal solution =

$\{u'_1, u'_2, \ldots, u'_k\} + \{u'_{k+1}, u'_{k+2}, \ldots, u'_l\} + \{u'_{l+1}, u'_{l+2}, \ldots, u'_r\}$

Items with larger profits in *X* *As initial choice Of k-set*	Items selected from remaining that matches with *X*	Items not selected by greedy algorithm

So,

approx solution =

$\{u'_1, u'_2, \ldots, u'_k\} + \{u'_{k+1}, u'_{k+2}, \ldots, u'_l\} + \{v_1, v_2, \ldots, v_t\}$

Items with larger profits in *X*	Items selected from remaining that matches with *X*	Items selected but not matching with *X*

Knapsack Problem: polynomial approximation scheme

approx solution =

$$u'_1,\ u'_2,\ \ldots,\ u'_k, u'_{k+1},\ v_{1,..}\ u'_{k+2},\ \ \ldots,\ v_{7,\ldots} u'_l,\ \ldots\ v_{m'},\ v_{m'+1},\ \ldots,\ v_t$$

In optimum set X

Not in optimum X set

$$u'_1,\ u'_2,\ \ldots,\ u'_k, u'_{k+1},\ v_{1,..}\ u'_{k+2},\ \ \ldots,\ v_{7,\ldots} u'_l,\ \ldots\ v_{m'},\qquad v_{m'+1},\ \ldots,\ v_t$$

u'_{l+1} is in this place,
but the algorithm cannot select it

Knapsack Problem: polynomial approximation scheme

approx solution =

$\{u'_1, u'_2, \ldots, u'_k\} + \{u'_{k+1}, u'_{k+2}, \ldots, u'_l\} + \{v_1, v_2, \ldots, v_{m'}\} + \{v_{m'+1}, \ldots, v_t\}$

Here each of these Items has profit density at least p_{l+1}/s_{l+1}

Here total size is less than s_{l+1}

Profit in approximation solution $\geq \sum_{i=1}^{l} profit(u'_i) + \sum_{i=1}^{m'} profit(v_i)$

Knapsack Problem: polynomial approximation scheme

Profit in approximation solution≥

$$\sum_{i=1}^{l} profit(u'_i) + p_{l+1}/s_{l+1} \sum_{i=1}^{m'} size(v_i)$$

Optimal solution= $\sum_{i=1}^{l} profit(u'_i) + \sum_{i=l+1}^{r} profit(u'_i)$

$$\le \text{profit of approximation sol} - p_{l+1}/s_{l+1} \sum_{i=1}^{m'} size(v_i)$$

$$+(B - \sum_{\iota=1}^{l} size(u'_i))p_{l+1}/s_{l+1}$$

$$\le \text{profit of approximation sol}$$

$$+(B - \sum_{i=1}^{l} size(u'_i) - \sum_{i=1}^{m'} size(v_i))p_{l+1}/s_{l+1}$$

Knapsack Problem: polynomial approximation scheme

Optimal solution

$\leq$ *profit of approximation solution* $+ \; p_{l+1}$

Knapsack Problem: polynomial approximation scheme

Optimal solution - *profit of approximation sol*

$\leq p_{l+1} \leq$ *profit of approximation sol / k*

Optimal solution - *profit of approximation sol*

$\leq$ *Optimal sol / k*

(1-1/k)Optimal Solution $\leq$ *profit of approximation sol*

K-center Clustering

Given :	a set $P = \{p_1, p_2, ..., p_n\}$ of n points in the plane an integer k
Objective:	find a collection of k circles that collectively enclose all the points
such that	radius of the largest circle is as small as possible

K-center Problem is NP hard for $k \geq 2$

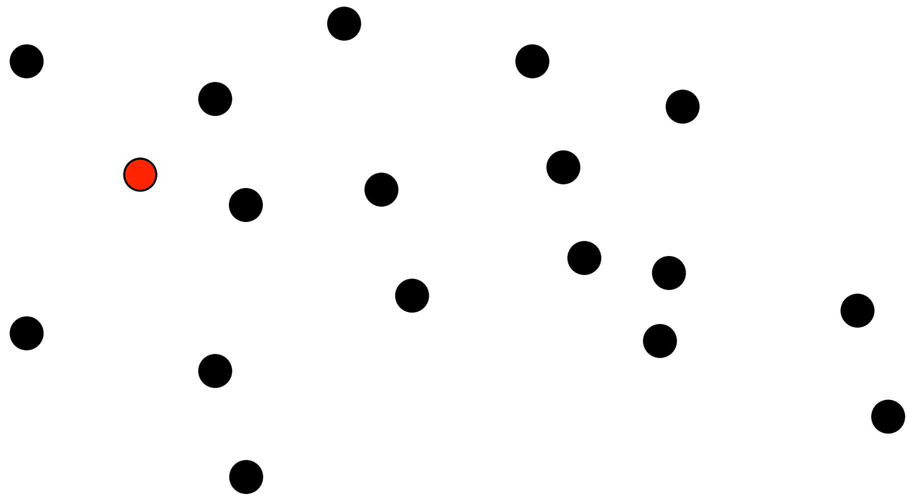
It is NP hard even to approximate within a factor of roughly 1.8

K-center Clustering : A Greedy Strategy

Teofilo Gonzalez 1985

Choose the *k* center points one at a time

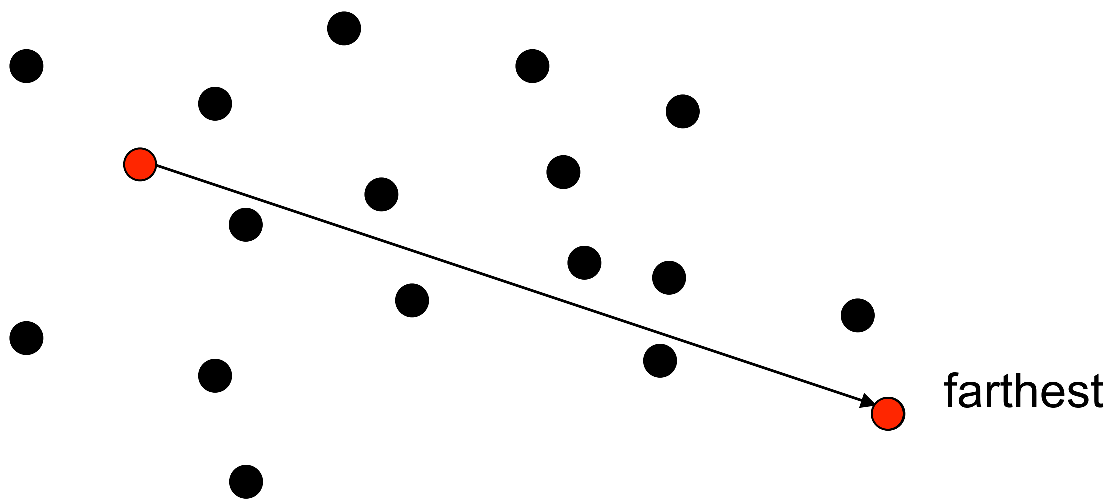
Starting with an arbitrary input point as the first center



K-center Clustering : A Greedy Strategy

Teofilo Gonzalez 1985

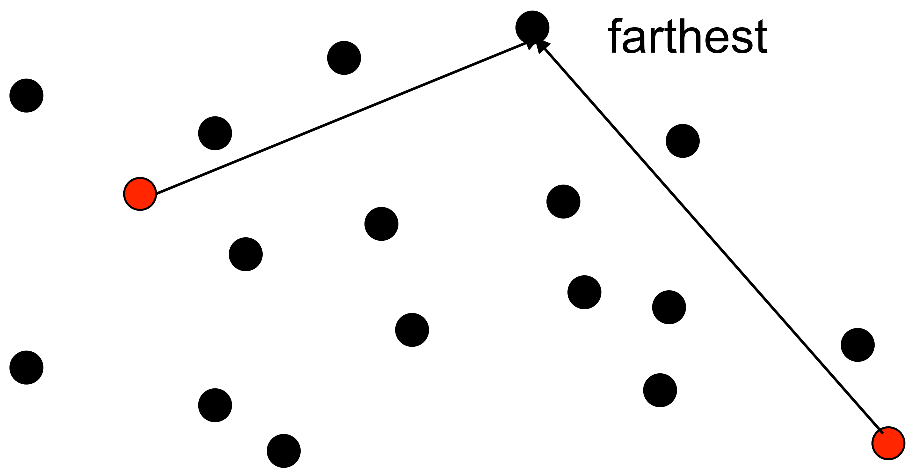
In each iteration,
choose the input point that is farthest from
any earlier center point to be the next center point



K-center Clustering : A Greedy Strategy

Teofilo Gonzalez 1985

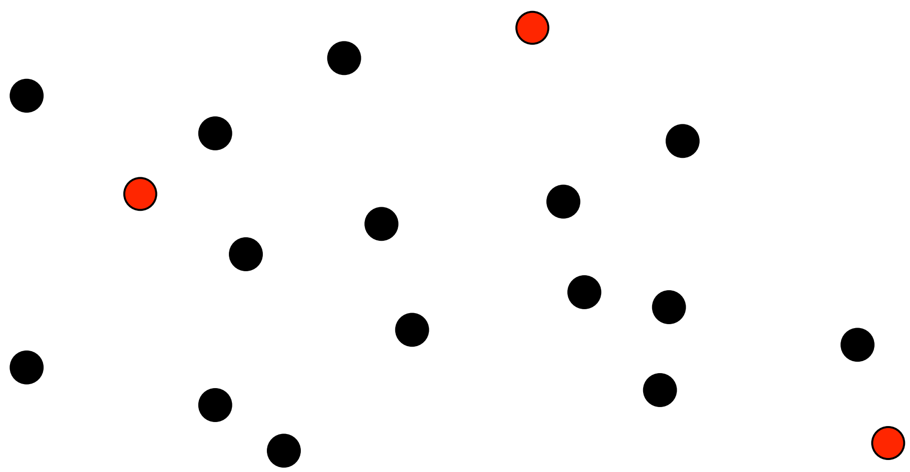
In each iteration,
choose the input point that is farthest from
any earlier center point to be the next center point



K-center Clustering : A Greedy Strategy

Teofilo Gonzalez 1985

In each iteration,
choose the input point that is farthest from
any earlier center point to be the next center point



K-center Clustering : A Greedy Strategy

Teofilo Gonzalez 1985

Performance

Let

r^* : optimal k-center clustering radius

r : clustering radius obtained by this algorithm for $k+1$ center

If $r > 2r^*$ then

any ball of radius r^* contains at most one of these $k+1$ center points

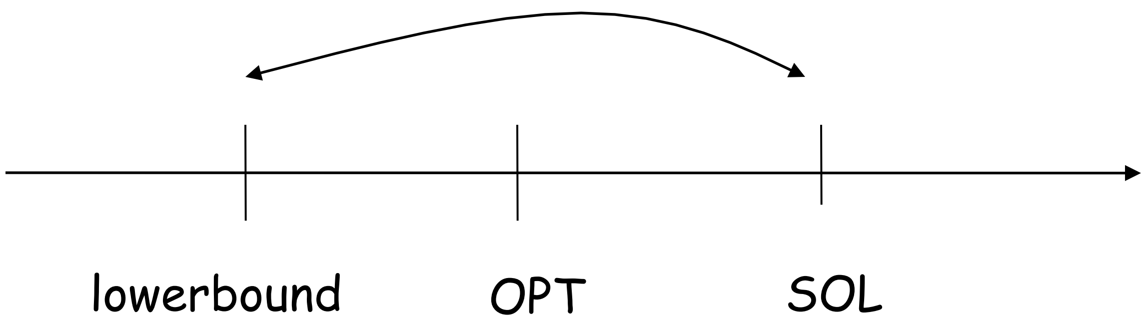
Conclusion : Algorithm computes a 2-approximation to the optimal k-center clustering

Lower bound and Approximation Algorithm

For NP-complete problem, we can't compute an optimal solution in polytime.

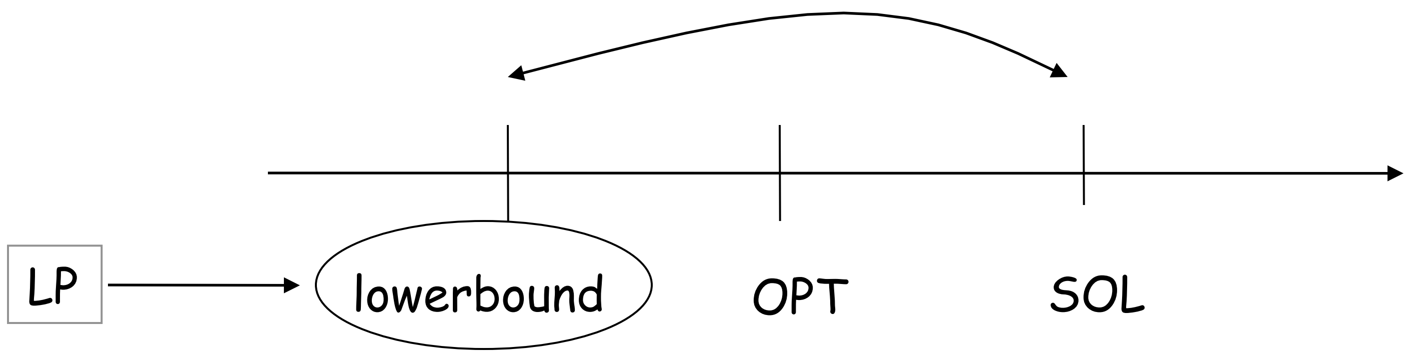
The key of designing a polytime approximation algorithm is
to obtain a good (lower or upper) bound on the optimal solution.

The general strategy (for a minimization problem) is:



SOL $\leq c \cdot$ lowerbound $\Rightarrow$ SOL $\leq c \cdot$ OPT

Linear Programming and Approximation Algorithm



Linear programming: a general method to compute a lowerbound in polytime.

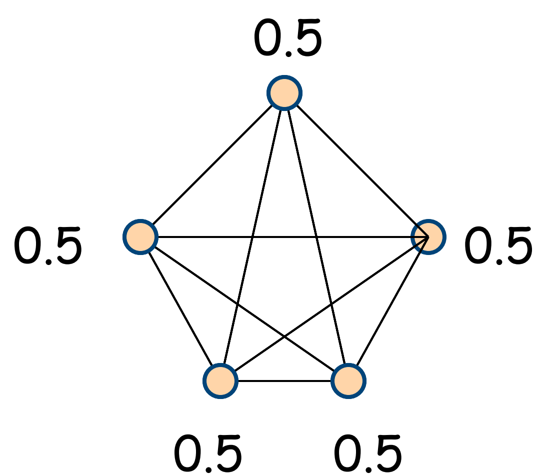
To computer an approximate solution,
we need to return an (integral) solution
close to an optimal LP (fractional) solution.

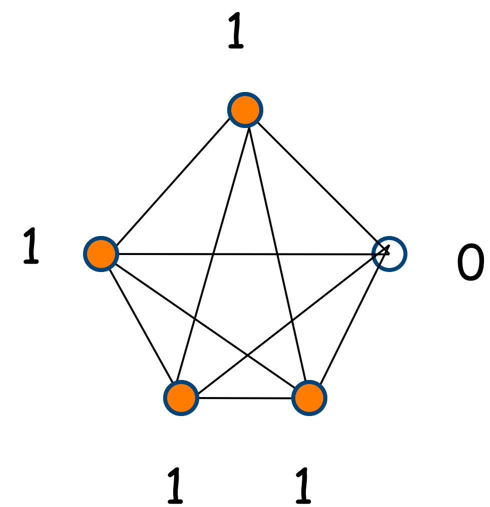
An Example: Vertex Cover

Integrality gap: $= \max \dfrac{\text{Optimal integer solution.}}{\text{Optimal fractional solution.}}$

Over all instances.

In vertex cover, there are instances where this gap is almost 2.





Linear Programming Relaxation for Vertex Cover

$$\min \sum_{v \in V(G)} y_v$$

$$\sum_{\{u,v\}=e} y_u + y_v \geq 1$$

$$y_v \geq 0$$

Theorem: For the vertex cover problem,
every vertex (or basic) solution of the LP
is half-integral, i.e. x(v) = {0, ½, 1}

Linear Programming Relaxation for Set Cover

$$\min \sum_{S \in S^*} c(S) x_S$$

$$\sum_{S : e \in S} x_S \geq 1 \qquad \text{for each element e.}$$

$$x_S \geq 0 \qquad \text{for each subset S.}$$

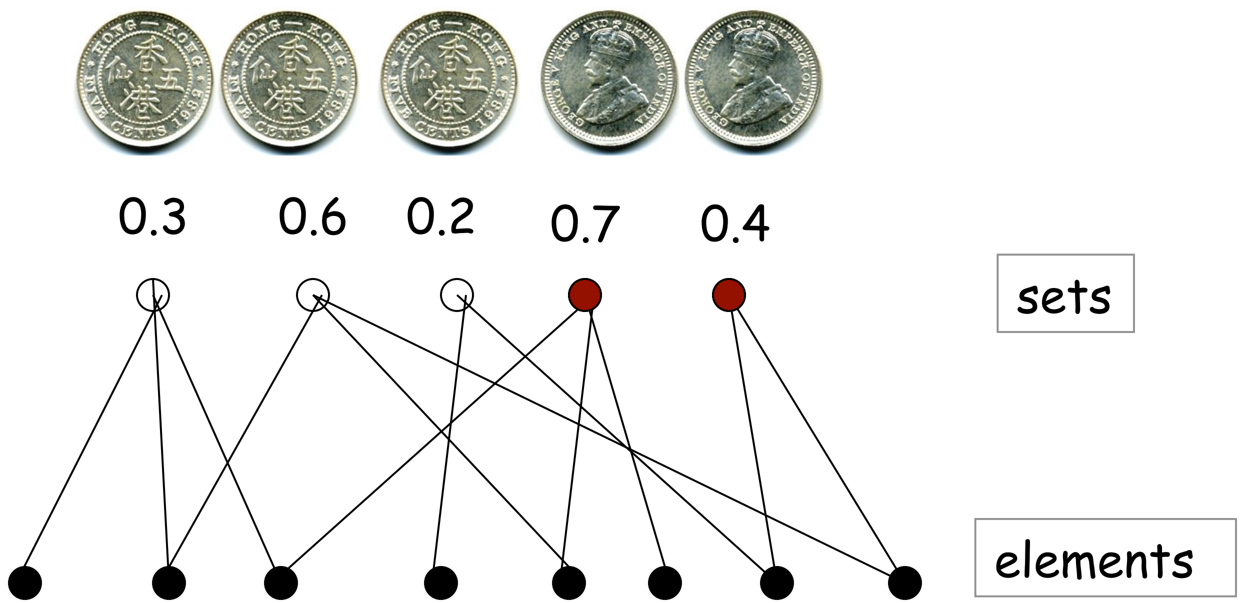
How to “round” the fractional solutions?

Idea: View the fractional values as probabilities, and do it randomly!

Algorithm

First solve the linear program to obtain the fractional values x*.

Then flip a (biased) coin for each set with probability x*(S) being "head".



Add all the "head" vertices to the set cover.

Repeat log(n) rounds.

Performance

Theorem: The randomized rounding gives an O(log(n))-approximation.

Claim 1: The sets picked in each round have an expected cost of at most LP.

Claim 2: Each element is covered with high probability after O(log(n)) rounds.

So, after O(log(n)) rounds, the expected total cost is at most O(log(n)) LP, and every element is covered with high probability, and hence the theorem.

Remark: It is NP-hard to have a better than O(log(n))-approximation!

Cost

Claim 1: The sets picked in each round have an expected cost of at most LP.

$$
\begin{aligned}
& \mathbf{E}[\text{total cost}] \\
= & \sum_{S \in S^*} \mathbf{E}[\text{cost of } S] \\
= & \sum_{S \in S^*} \mathbf{Pr}[S \text{ is picked}] \cdot c(S) \\
= & \sum_{S \in S^*} x_S \cdot c(S) \\
= & LP
\end{aligned}
$$

Q.E.D.

Feasibility

Claim 2: Each element is covered with high probability after $O(\log(n))$ rounds.

First consider the probability that an element e is covered after one round.

Let say e is covered by $S1, \ldots, Sk$ which have values $x1, \ldots, xk$.

By the linear program, $x1 + x2 + \ldots + xk >= 1$.

Pr[e is not covered in one round] = $(1 - x1)(1 - x2)\ldots(1 - xk)$.

This is maximized when $x1=x2=\ldots=xk=1/k$, why?

Pr[e is not covered in one round] $<= (1 - 1/k)^k$

Feasibility

Claim 2: Each element is covered with high probability after O(log(n)) rounds.

First consider the probability that an element e is covered after one round.

Pr[e is not covered in one round] <= $(1 - 1/k)^k$

So, $$\mathbf{Pr}[e \text{ is covered in one round}] \geq 1 - \left(1 - \frac{1}{k}\right)^k \geq 1 - \frac{1}{e}$$

What about after O(log(n)) rounds?

$$\mathbf{Pr}[e \text{ is not covered}] \leq \left(\frac{1}{e}\right)^{O(\log n)} \leq \frac{1}{4n}$$

Feasibility

Claim 2: Each element is covered with high probability after O(log(n)) rounds.

$$\mathbf{Pr}[e \text{ is not covered}] \le \left(\frac{1}{e}\right)^{O(\log n)} \le \frac{1}{4n}$$

So,

$$\mathbf{Pr}[\text{some element is not covered}] \le n \cdot \frac{1}{4n} \le \frac{1}{4}$$

So,

$$\mathbf{Pr}[\text{a set cover is returned}] \ge \frac{3}{4}$$

Remark

Let say the sets picked have an expected total cost of at most clog(n) LP.

Claim: The total cost is greater than 4clog(n) LP with probability at most $\frac{1}{4}$.

This follows from the Markov inequality, which says that:

$$\mathbf{Pr}[X \geq t] \leq \frac{\mathbf{E}[X]}{t}$$

Proof of Markov inequality:

$$\mathbf{E}[X] = |X| \cdot Pr[X \geq t] + |X| \cdot Pr[X < t] \geq t \cdot Pr[X \geq t]$$

The claim follows by substituting E[X]=clog(n)LP and t=4clog(n)LP

Wrap Up

$$\mathbf{Pr}[\text{a set cover is returned}] \geq \frac{3}{4}$$

$$\mathbf{Pr}[\text{cost} \leq O(\log n)] \geq \frac{3}{4}$$

Theorem: The randomized rounding gives an O(log(n))-approximation.

This is the only known rounding method for set cover.

Randomized rounding has many other applications.