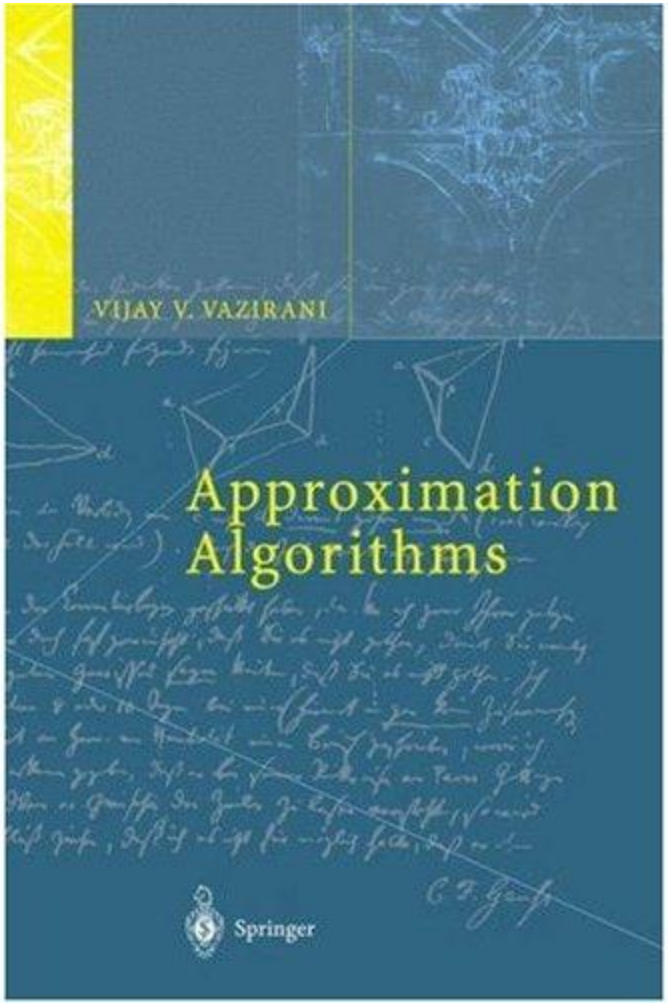
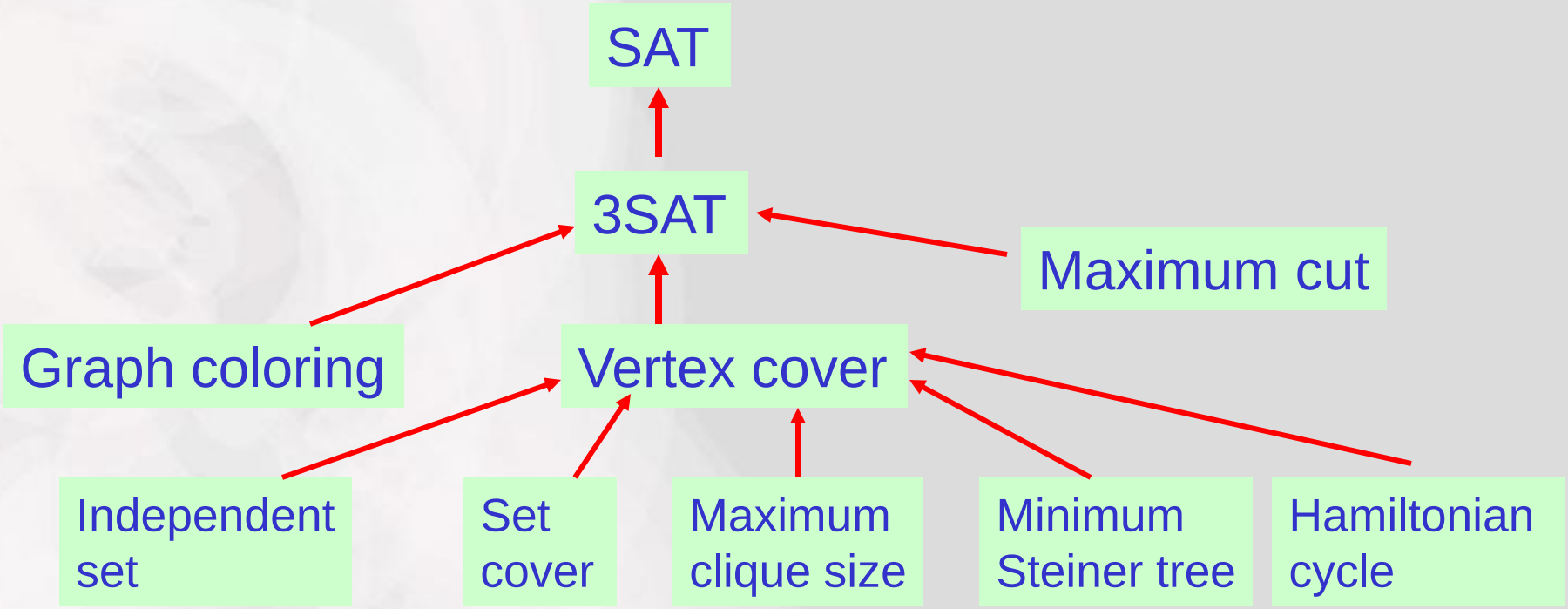


Approximation Algorithms

Examples of NP-complete problems

Summary of some NPc problems



find more NP-complete problems in
http://en.wikipedia.org/wiki/List_of_NP-complete_problems

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.

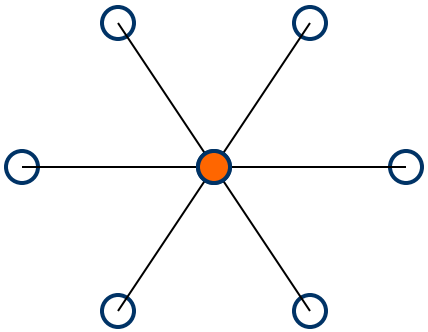
Additive approximation algorithms: SOL <= OPT + c for some constant c.
Very few examples known: edge coloring, minimum maximum-degree spanning tree.

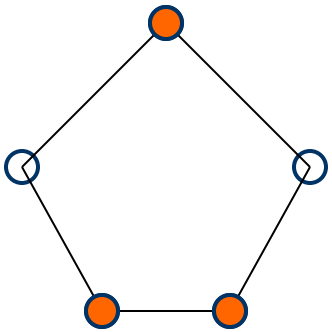
Constant factor approximation algorithms: SOL <= cOPT for some constant c.
Many more examples known.

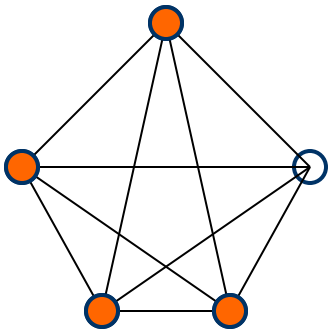
Vertex Cover

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: Greedy Algorithm 1

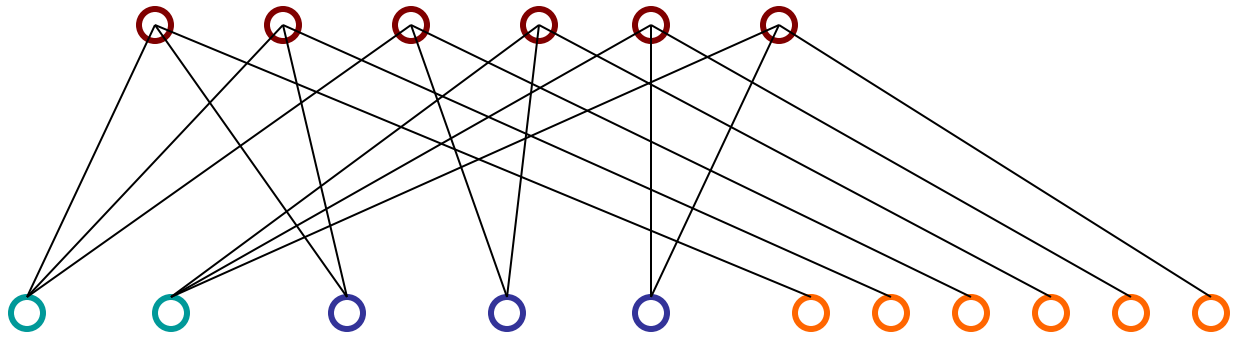
Idea: Keep finding a vertex which covers the maximum number of edges.

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.

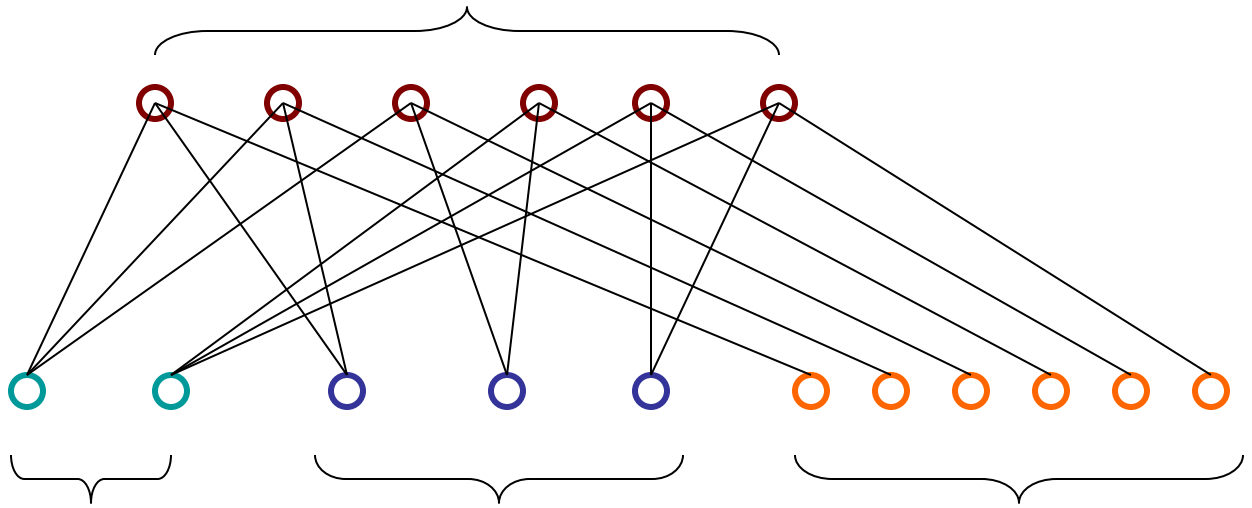
SOL = 11, if we are unlucky in breaking ties.
First we might choose all the green vertices.
Then we might choose all the blue vertices.
And then we might choose all the orange vertices.

Vertex Cover: Greedy Algorithm 1

k! vertices of degree k

Not a constant factor approximation algorithm!

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.

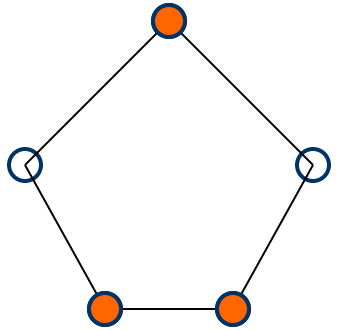
SOL = k! (1/k + 1/(k-1) + 1/(k-2) + … + 1) ≈ **k! log(k)**, all bottom vertices.

Vertex Cover: Greedy Algorithm 2

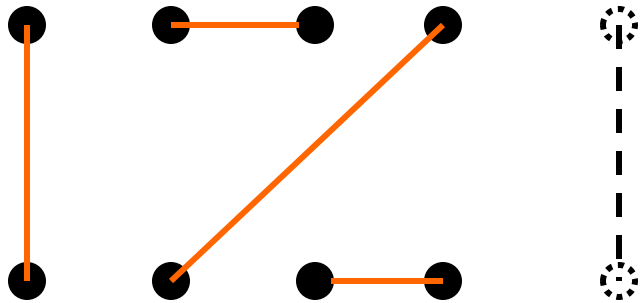
In bipartite graphs, maximum matching = minimum vertex cover.

In general graphs, this is not true.

How large can this gap be?



Vertex Cover: Greedy Algorithm 2



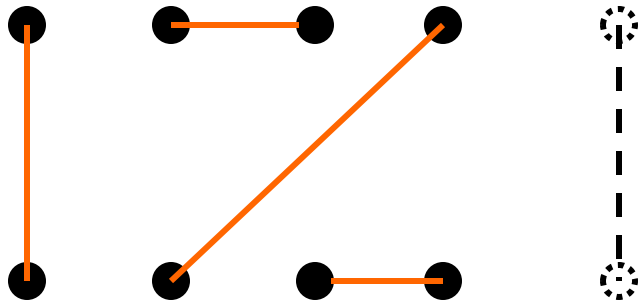
Fix a maximum matching. Call the vertices involved **black**.

Since the matching is maximum, every edge must have a black endpoint.

So, by choosing all the black vertices, we have a vertex cover.

SOL <= 2 * size of a maximum matching

Vertex Cover: Greedy Algorithm 2



What about an optimal solution?

Each edge in the matching has to be covered by a **different** vertex!

OPT >= size of a maximum matching

So, OPT <= 2 SOL, and we have a 2-approximation algorithm!

Vertex Cover

Approximate min-max theorem:

Maximum matching <= minimum vertex cover <= 2*maximum matching

Major open question: Can we obtain a better than 2-approximation algorithm?

Hardness result: It is NP-complete even to ***approximate*** within a factor of 1.36!!

The Traveling Salesman Problem

Traveling Salesman Problem (TSP):
Given a complete graph with nonnegative edge costs,
Find a minimum cost cycle visiting every vertex exactly once.

Given a number of cities and the costs of traveling from any city to any other city, what is the cheapest round-trip route that visits each city exactly once and then returns to the starting city?

One of the most well-studied problem in combinatorial optimization.

NP-completeness of Traveling Salesman Problem

Traveling Salesman Problem (TSP):
Given a complete graph with nonnegative edge costs,
Find a minimum cost cycle visiting every vertex exactly once.

Hamiltonian cycle Problem:
Given an undirected graph, find a cycle visiting every vertex exactly once.

The Hamiltonian cycle problem is a special case of the Traveling Salesman Problem.

- For each edge, we add an edge of cost 1.
- For each non-edge, we add an edge of cost 2.

- If there is a Hamiltonian cycle, then there is a cycle of cost n.
- If there is no Hamiltonian cycle, then every cycle has cost at least n+1.

Inapproximability of Traveling Salesman Problem

Theorem: There is no constant factor approximation algorithm for TSP, unless P=NP.

Idea: Use the Hamiltonian cycle problem.

- For each edge, we add an edge of cost 1.
- For each non-edge, we add an edge of cost ~~2~~ **nk**.

- If there is a Hamiltonian cycle, then there is a cycle of cost n.
- If there is no Hamiltonian cycle, then every cycle has cost greater than **nk**.

So, if you have a k-approximation algorithm for TSP,
one just needs to check if the returned solution is **at most nk**.

- If yes, then the original graph has a Hamiltonian cycle.
- Otherwise, the original graph has no Hamiltonian cycle.

Inapproximability of Traveling Salesman Problem

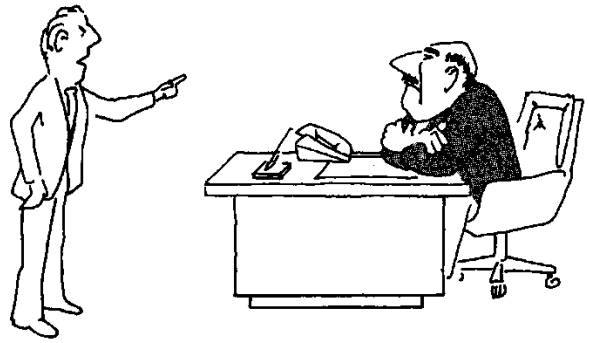
Theorem: There is no constant factor approximation algorithm for TSP, unless P=NP.

This type of theorem is called "hardness result" in the literature.
Just like their names, usually they are very hard to obtain.

- If there is a Hamiltonian cycle, then there is a cycle of cost n.
- If there is no Hamiltonian cycle, then every cycle has cost greater than **nk**.

The strategy is usually like this.
This creates a gap between yes and no instances.
The bigger the gap, the problem is harder to approximate.

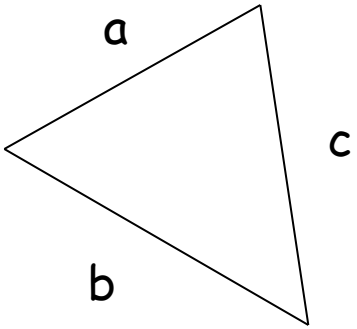
Approximation Algorithm for TSP



"I can't find an efficient algorithm, because no such algorithm is possible!"

There is no constant factor approximation algorithm for TSP, what can we do then?

Observation: in real world situation, the costs satisfy triangle inequality.

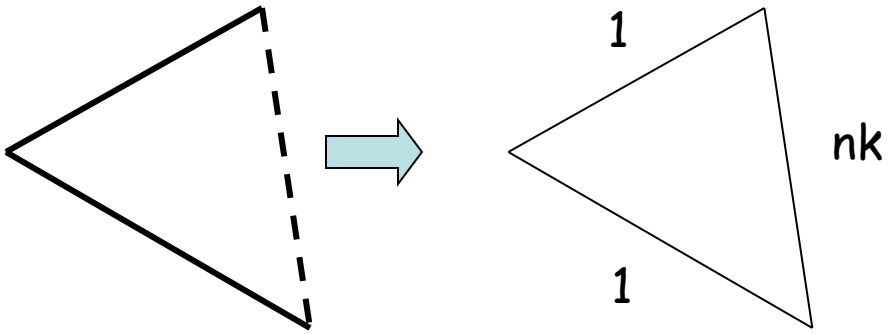


$a + b \geq c$

For example, think of cost of an edge as the distance between two points.

Approximation Algorithm for Metric TSP

Metric Traveling Salesman Problem (metric TSP):
Given a complete graph with edge costs satisfying triangle inequalities,
Find a minimum cost cycle visiting every vertex exactly once.



First check the bad example.

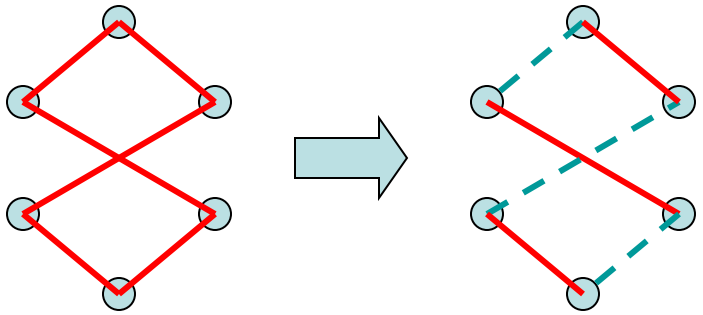
This violates triangle inequality!

How could triangle inequalities help in finding approximation algorithm?

Lower Bounds for TSP

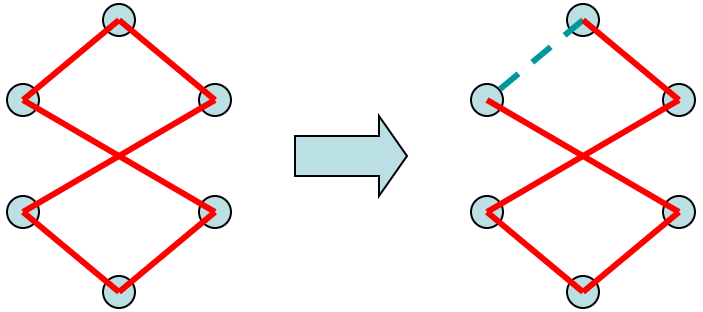
What can be a good lower bound to the cost of TSP?

A tour contains a matching.



Let OPT be the cost of an optimal tour, since a tour contains two matchings, the cost of a **minimum weight perfect matching** is at most OPT/2.

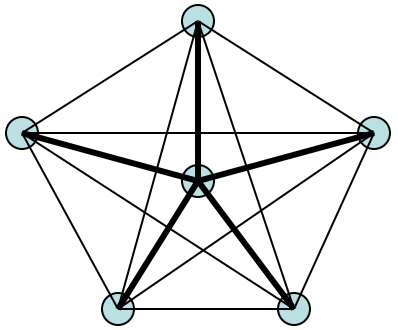
A tour contains a spanning tree.



So, the cost of a **minimum spanning tree** is at most OPT.

Spanning Tree and TSP

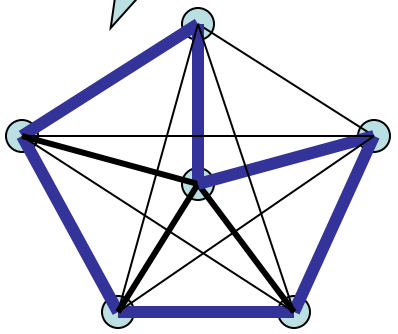
Let the thick edges have cost 1,
And all other edges have cost greater than 1.



So the thick edges form a minimum spanning tree.

But it doesn't look like a Hamiltonian cycle at all!

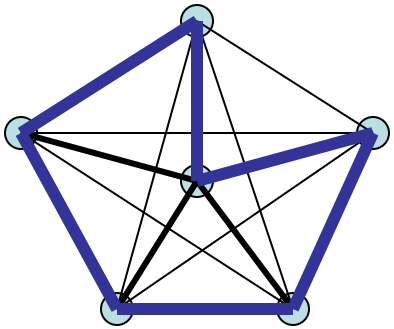
Edge cost at most 2

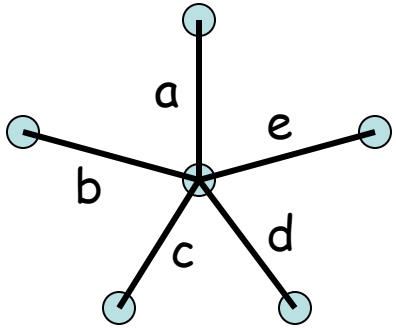


Consider a Hamiltonian cycle.
The costs of the edges which are not in the minimum spanning tree might have very high costs.

Not really! Each such edge has cost at most 2 because of the **triangle inequality**.

Spanning Tree and TSP





Strategy: Construct the TSP tour from a minimum spanning tree.

Use the edges in the minimum spanning tree as many as possible.

The center vertex already has degree 2, skip to the next vertex.

$$\text{Cost} = e + a + \underline{a + b} + \underline{b + c} + \underline{c + d} + \underline{d + e}$$

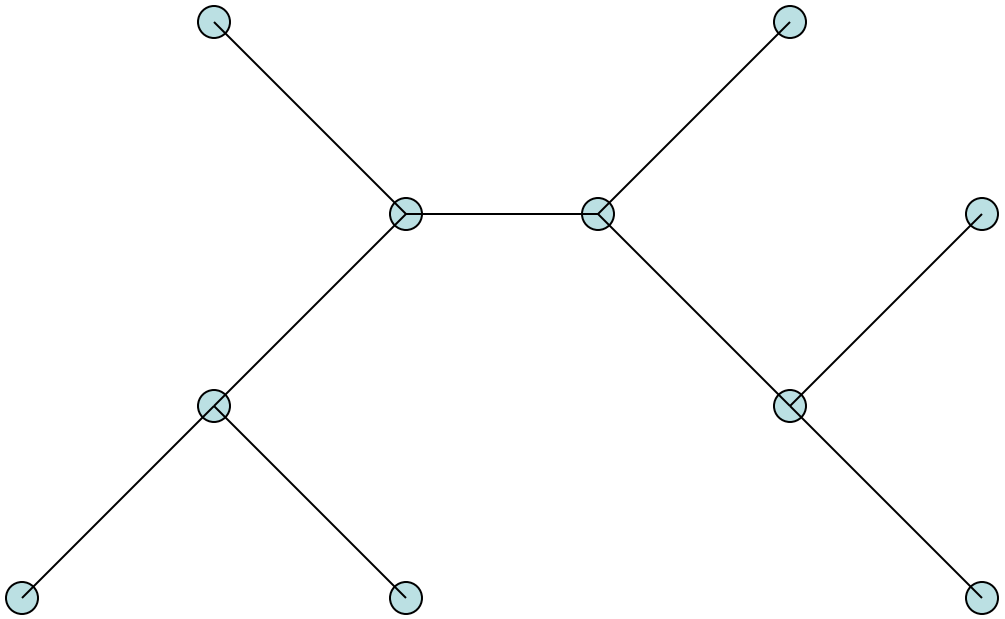
$$= 2a + 2b + 2c + 2d + 2e = 2(a + b + c + d + e)$$

Cost of a minimum spanning tree

$MST \le OPT$ + $SOL \le 2MST$ $\Rightarrow$ $SOL \le 2OPT$

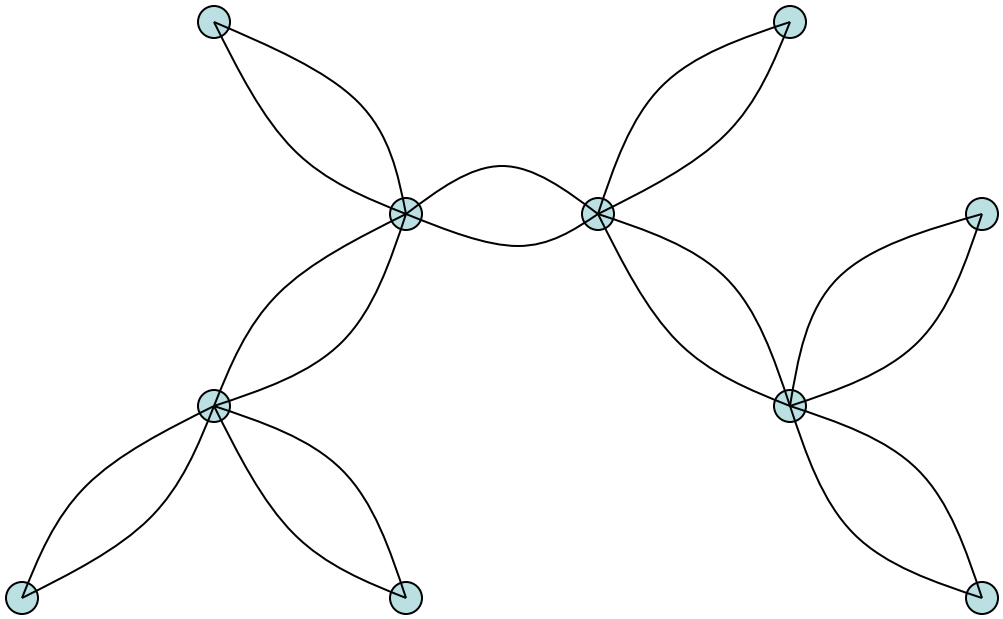
Spanning Tree and TSP

How to formalize the idea of "following" a minimum spanning tree?



Spanning Tree and TSP

How to formalize the idea of "following" a minimum spanning tree?

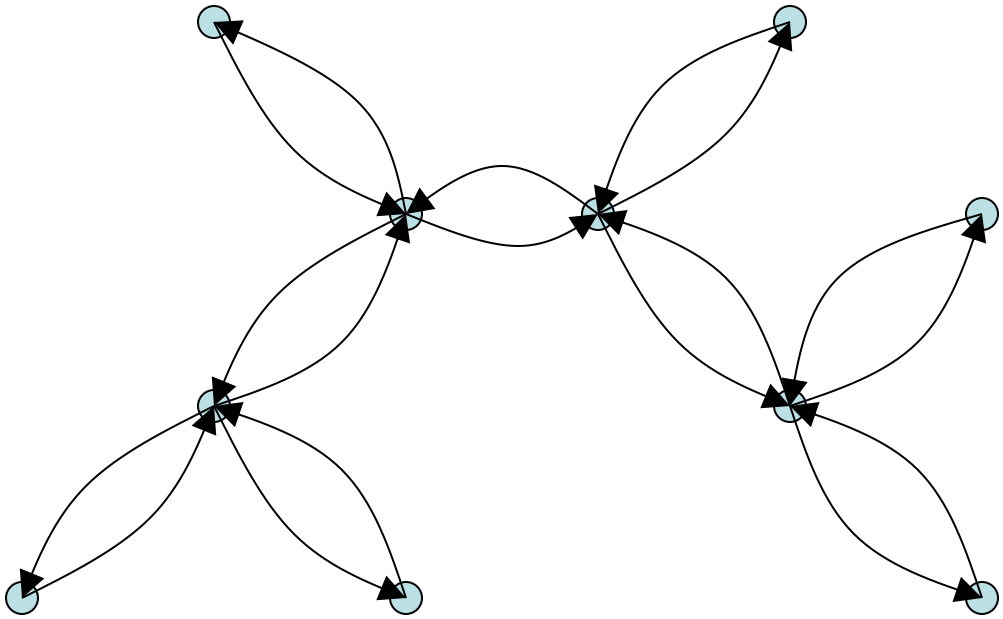


Key idea: double all the edges and find an Eulerian tour.

This graph has cost 2MST.

Spanning Tree and TSP

How to formalize the idea of "following" a minimum spanning tree?

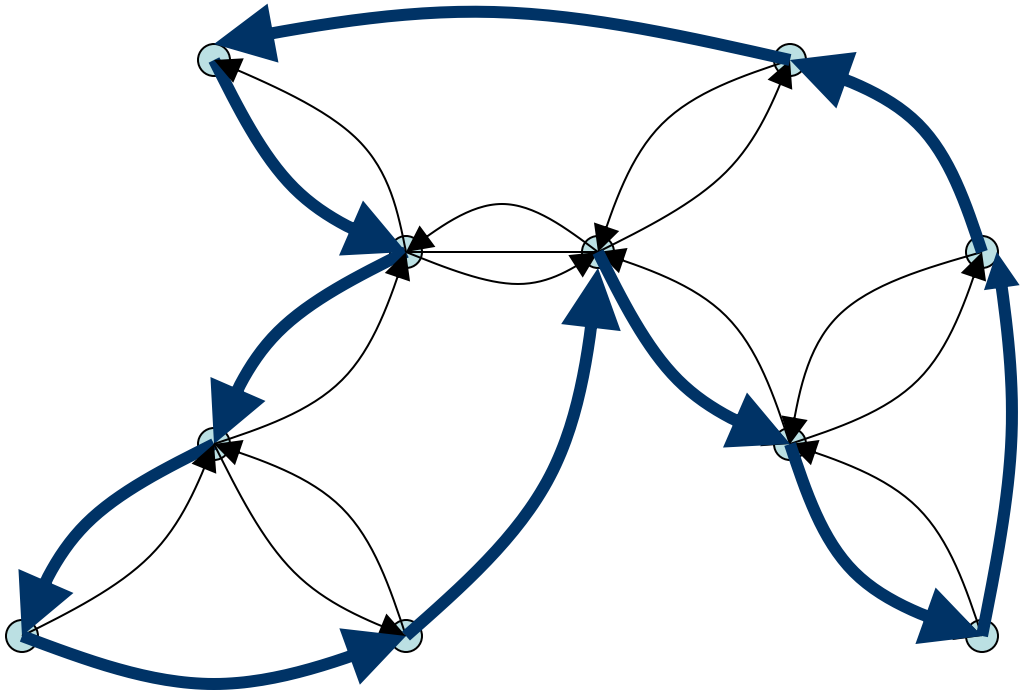


Key idea: double all the edges and find an Eulerian tour.

This graph has cost 2MST.

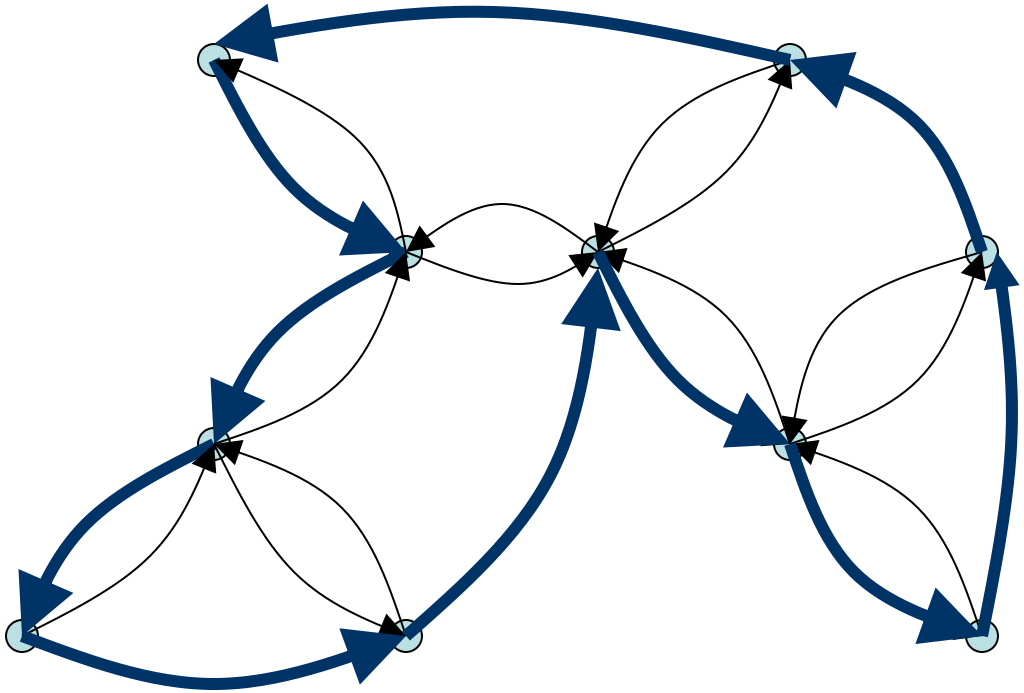
Spanning Tree and TSP

Strategy: shortcut this Eulerian tour.



Spanning Tree and TSP

By **triangle inequalites**, the shortcut tour is not longer than the Eulerian tour.



Each directed edge is used exactly once in the shortcut tour.

A 2-Approximation Algorithm for Metric TSP

(Metric TSP – Factor 2)

1. Find an MST, T, of *G*.
2. Double every edge of the MST to obtain an Eulerian graph.
3. Find an Eulerian tour, T*, on this graph.
4. Output the tour that visits vertices of *G* in the order of their first appearance in T*. Let *C* be this tour.
   (That is, shortcut T*)

Analysis:

1. cost(T) ≤ OPT (because MST is a lower bound of TSP)
2. cost(T*) = 2cost(T) (because every edge appears twice)
3. cost(*C*) ≤ cost(T*) (because of triangle inequalities, **shortcutting**)
4. So, cost(*C*) ≤ 2OPT

Better approximation?

There is a 1.5 approximation algorithm for metric TSP.

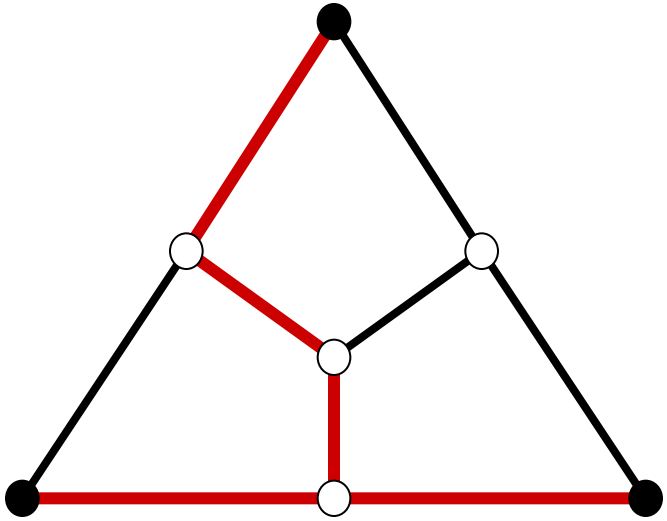
Hint: use a minimum spanning tree and a maximum matching (instead of double a minimum spanning tree).

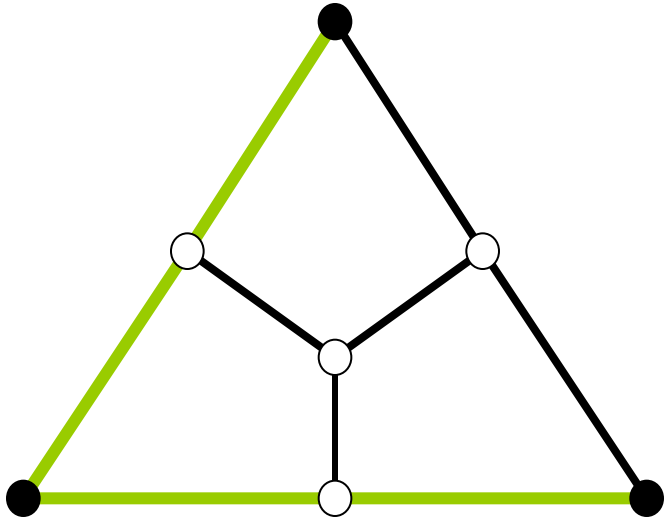
Major open problem: Improve this to 4/3?

An aside: hardness result is not an excuse to stop working, but to guide us to identify interesting cases.

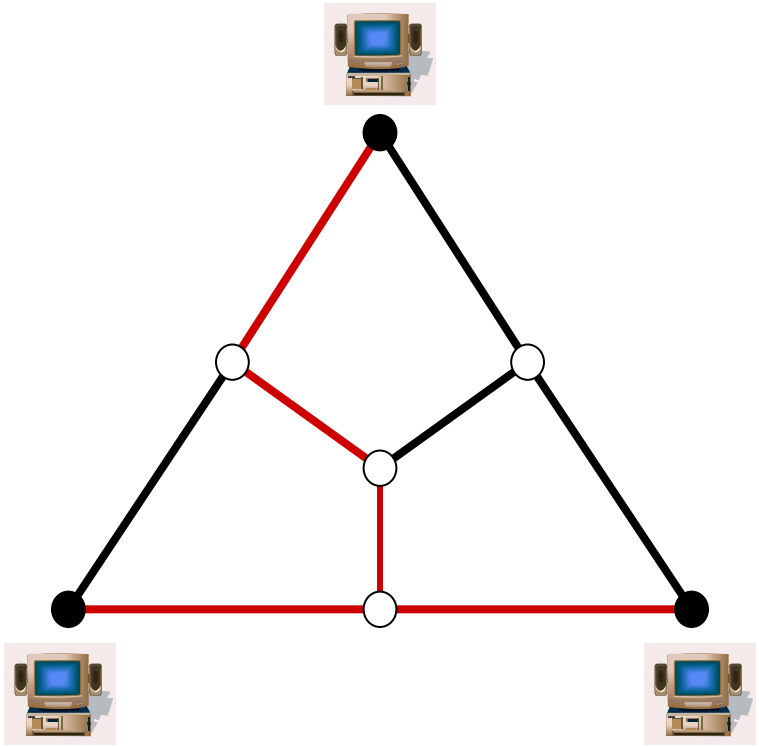
Minimum Steiner Tree

The Minimum Steiner Tree Problem:
Given an undirected graph G with nonnegative edge costs and whose vertices are partitioned into two sets, required vertices and Steiner vertices, find a minimum cost tree in G that contain all the required vertices and any subset of the Steiner vertices.

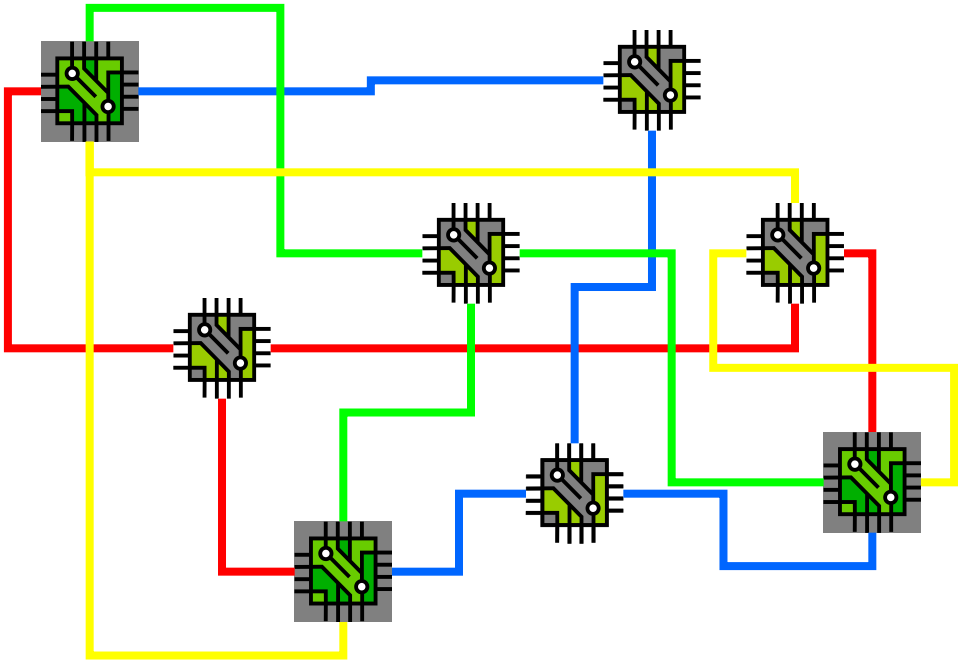




Applications

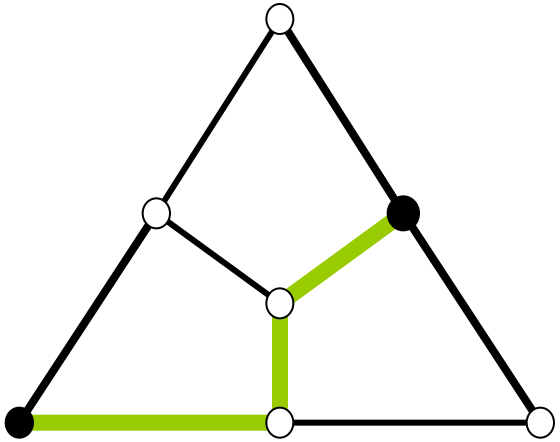


Computer networks

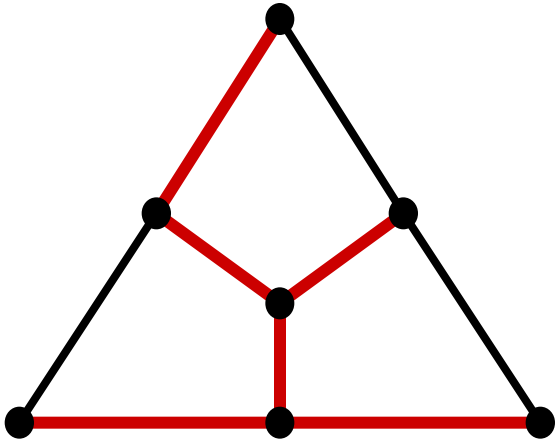


VLSI circuit design

Special Cases



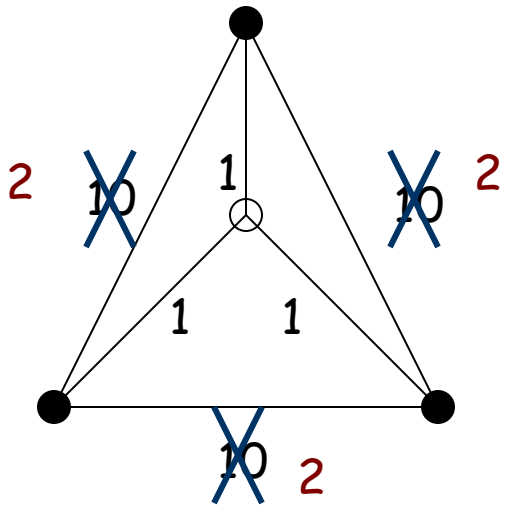
Only two required vertices,
just a shortest path.



Every vertex is required,
just a minimum spanning tree.

The general problem is NP-complete.

Useless Edges

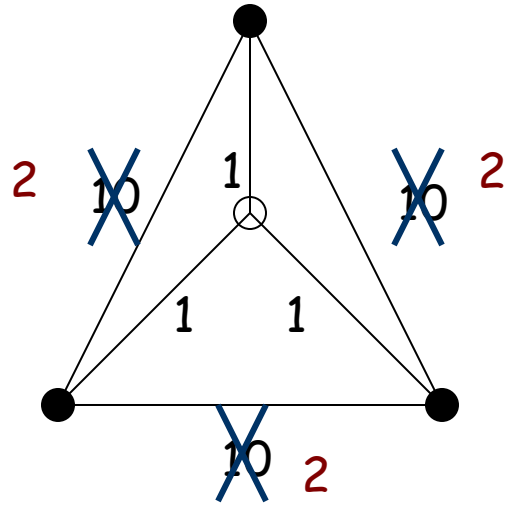


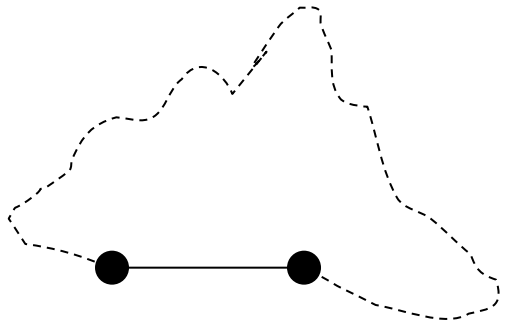
We must use Steiner vertices in order to construct a good solution, because edges connecting required vertices could be very expensive (or even don't exist).

Would I ever use an edge of weight 10 (an expensive edge)?

The purpose of using such an edge is to connect two required vertices, but we can always use a path of length 2 (which passes through the Steiner vertices and has a total cost of 2) for the same purpose!

Metric Closure



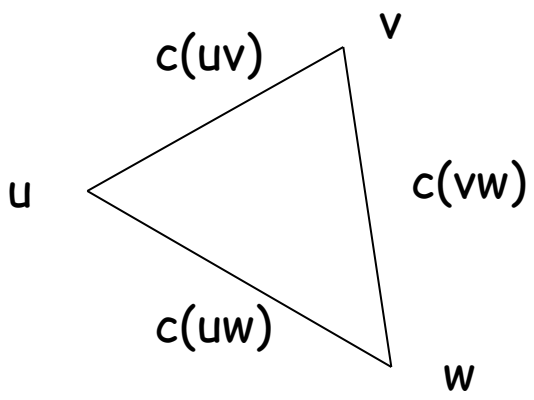


In general, we can always replace the cost of an edge by the cost of a **cheapest (shortest) path**.

We call the resulting graph the **metric closure** of the original graph.

Metric Closure

Claim. In the metric closure, the edge costs satisfy triangle inequalities.



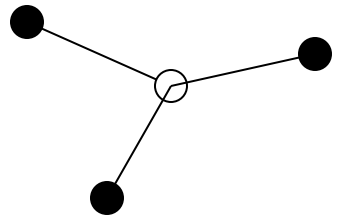
Note that the metric closure is a complete graph.

(Proof by contradiction:)
Suppose $c(vw) > c(uv) + c(uw)$. There is a path P(uv) between u and v with cost c(uv), and a path P(uw) between u and w with cost c(uw).
The union of P(uv) and P(uw) contains a path P* from v to w with cost at most $c(uv) + c(uw)$. This contradicts that c(vw) represents the cost of a shortest path from v to w (since P* is shorter).

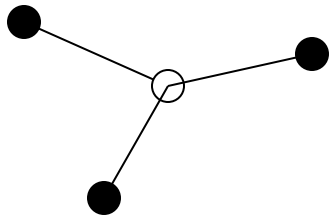
Metric Closure

Lemma The cost of a minimum Steiner tree in a graph G = the cost of a minimum Steiner tree in its metric closure.

First The cost of a minimum Steiner tree in a graph $G \geq$ the cost of a minimum Steiner tree in its metric closure.



in G

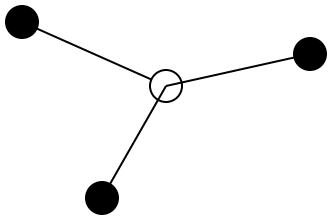


Just take the same tree in the metric closure

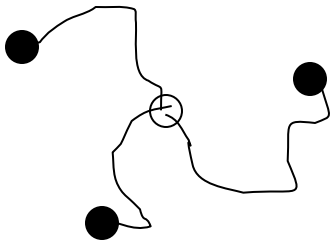
(Proof): Let T be a minimum Steiner tree in G. For each edge uv, The cost of uv in the metric closure is at most the cost of uv in G. So, the same T in the metric closure is no more expensive.

Metric Closure

Next The cost of a minimum Steiner tree in a graph $G \leq$ the cost of a minimum Steiner tree in its metric closure.



in metric closure



in G

take the union of the shortest paths

(Proof): Let T be a minimum Steiner tree in the metric closure. Consider the union U of the shortest paths in the original graph G. The union U has cost at most the cost of T. Clearly, U connects all the required vertices. So, U contains a Steiner tree T*. Hence, T* has cost at most the cost of T.

Metric Steiner Tree

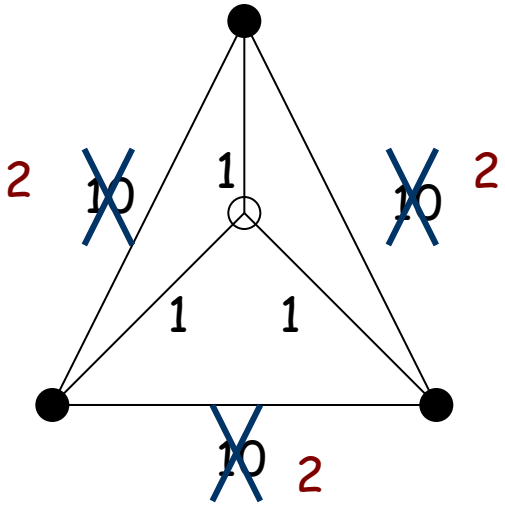
Lemma The cost of a minimum Steiner tree in a graph G = the cost of a minimum Steiner tree in its metric closure.

Claim. In the metric closure, the edge costs satisfy triangle inequalities.

Without loss of generality, we can consider the metric TSP problem.

Metric Steiner Tree Problem:
Given a complete graph with edge costs satisfying triangle inequalities, find a minimum Steiner tree (which connect all the required vertices).

Metric Steiner Tree



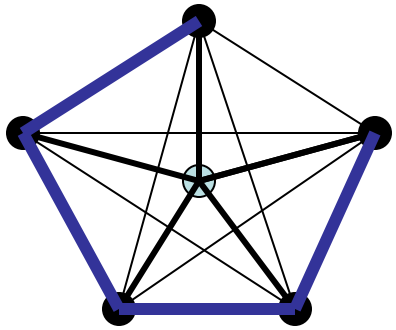
A Steiner tree connects all the required vertices.

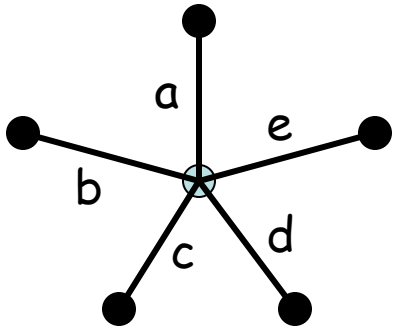
Can we just find a minimum spanning tree connecting all the required vertices?

In other words, forget about all Steiner vertices!

How bad can it be?

Spanning Tree and Steiner Tree





$MST \le \underline{a + b} + \underline{b + c} + \underline{c + d} + \underline{d + e}$

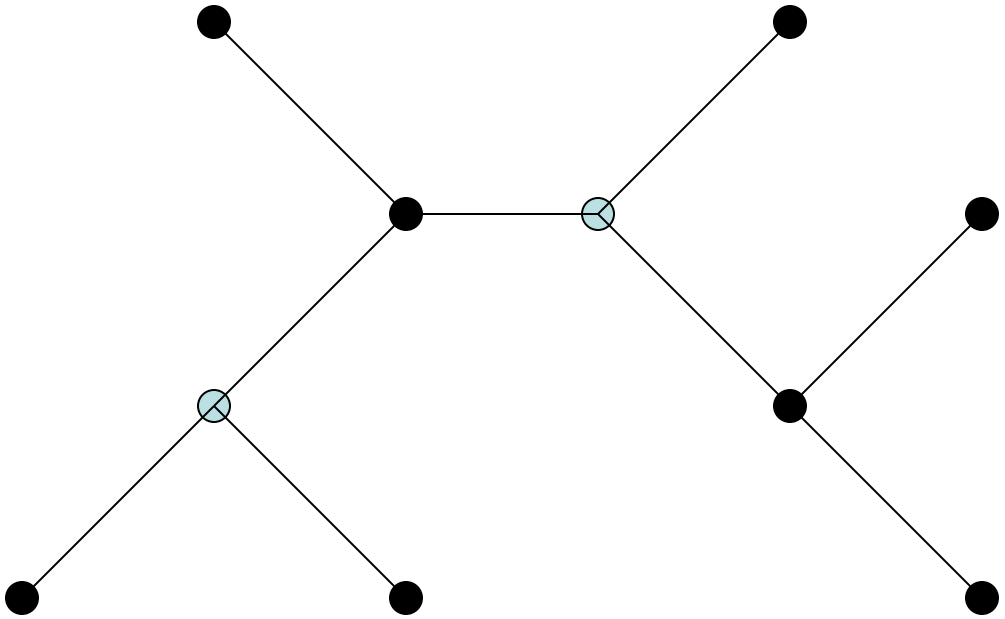
$\le 2a + 2b + 2c + 2d + 2e = 2(a + b + c + d + e)$

Cost of a minimum Steiner tree

$MST \le 2OPT$

Spanning Tree and Steiner Tree

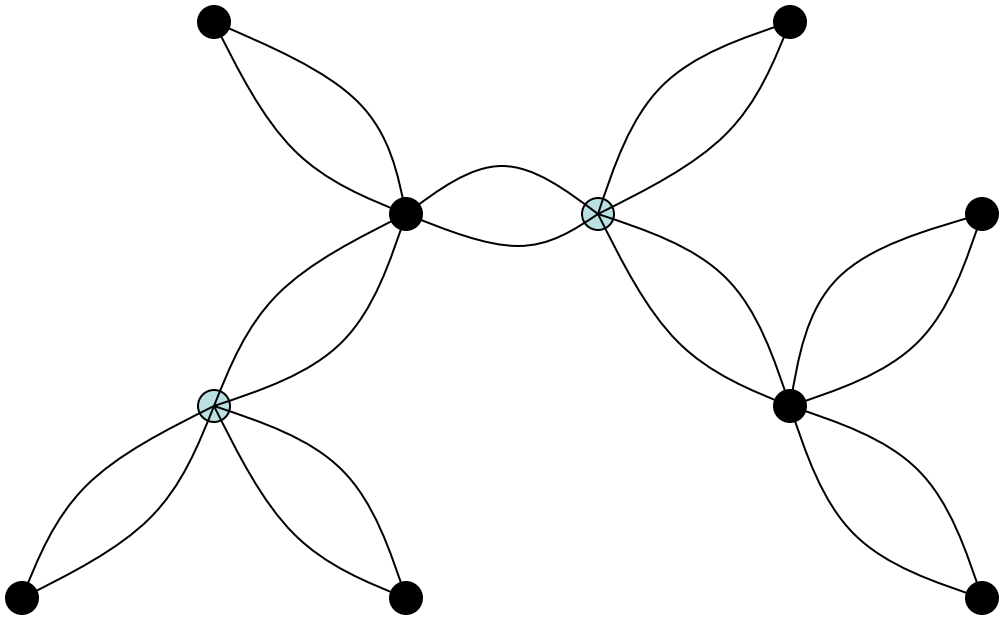
How to formalize the idea?



Let the cost of this Steiner tree be OPT.

Spanning Tree and Steiner Tree

How to formalize the idea?

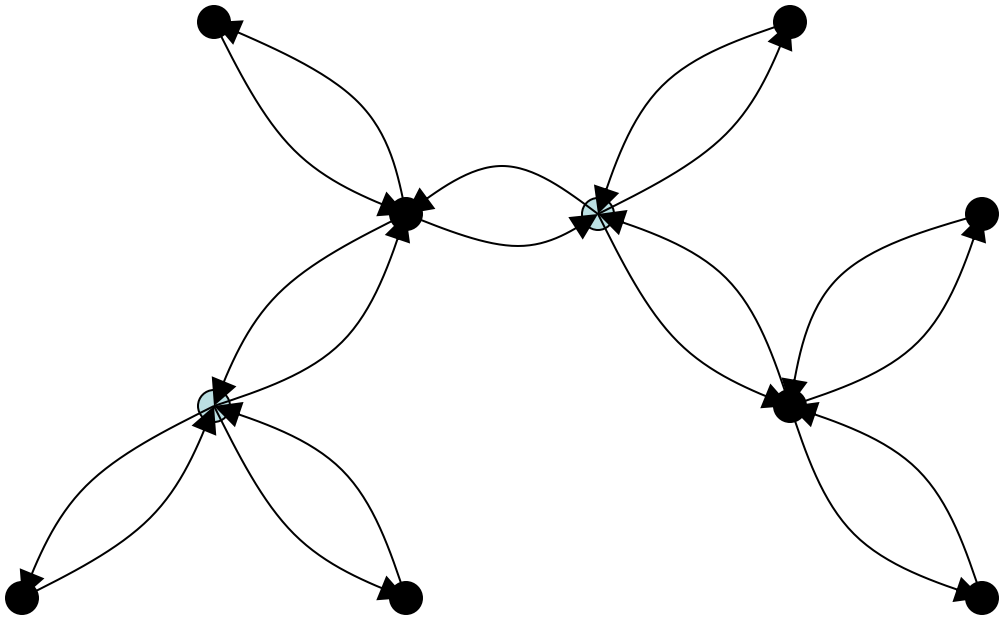


Key idea: double all the edges and find an Eulerian tour.

This graph has cost 2OPT.

Spanning Tree and Steiner Tree

How to formalize the idea?

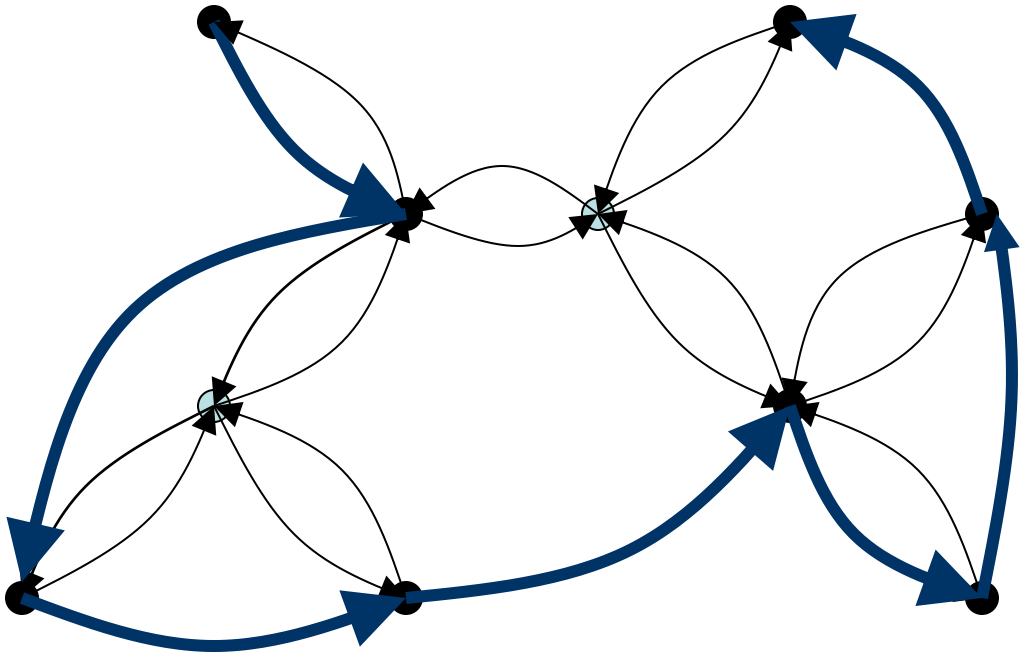


Key idea: double all the edges and find an Eulerian tour.

This graph has cost 2OPT.

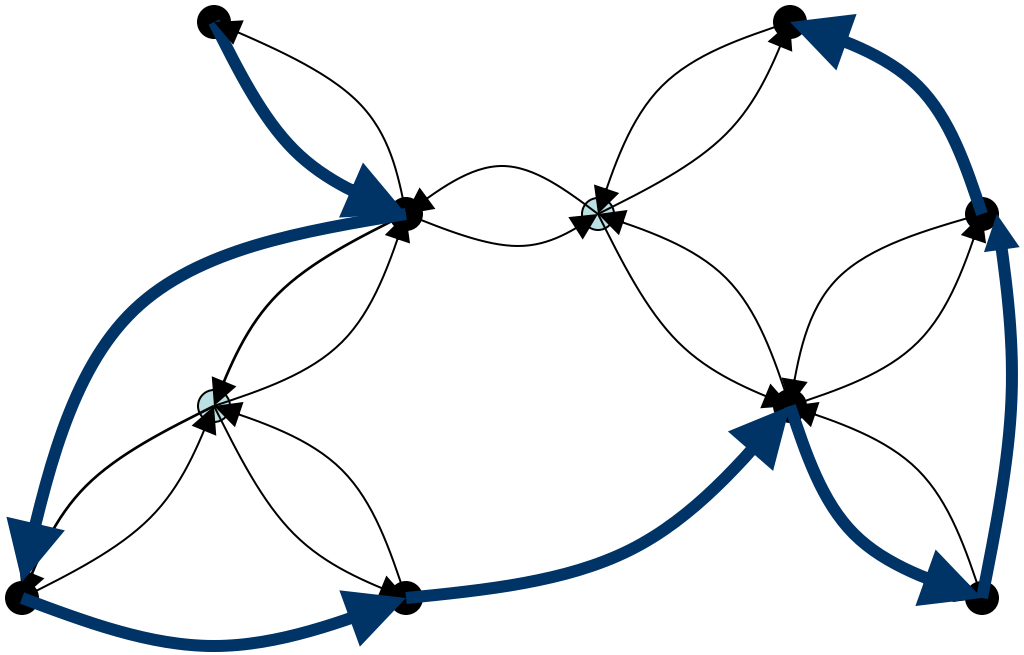
Spanning Tree and Steiner Tree

Strategy: shortcut this Eulerian tour.



Spanning Tree and Steiner Tree

By **triangle inequalites**, the shortcut tour is not longer than the Eulerian tour.



So, the cost of MST ≤ the cost of Eulerian graph ≤ 2OPT.

A 2-Approximation for Metric Steiner Tree

(Metric Steiner Tree – Factor 2)

1. Compute the metric closure G^* of G.
2. Find a minimum spanning tree T^* of G^*.
3. Construct the union U of the shortest paths in G which correspond to edges in T^*.
4. Output the Steiner tree T contained in U.

Analysis:

1. $cost(T) \leq cost(U)$ (because T is contained in U)
2. $cost(U) \leq cost(T^*)$ (because G^* is the metric closure of G)
3. $cost(T^*) \leq 2OPT$ (because of triangle inequalities, **shortcutting**)
4. So, $cost(T) \leq 2OPT$

Better approximation?

(Robin Zelikovsky)
There is a 1.55 approximation algorithm for minimum Steiner tree.

Just for fun 1: Show that the 2-approximation algorithm is equivalent to the following algorithm:

- Find a cheapest path that connects two required vertices which are not yet connected.
- Repeat until all required vertices are connected.

Just for fun 2: Design a polynomial time algorithm to find a minimum Steiner tree for k required vertices, when k is fixed.

Polynomial Time Approximation Scheme (PTAS)

We have seen the definition of an additive approximation algorithm, and the definition of a constant factor approximation algorithm.

The following is something even better.

An algorithm ***A*** is an **approximation scheme** if for every $\epsilon > 0$,

A runs in polynomial time (which may depend on ϵ) and return a solution:

- $SOL \leq (1+\epsilon)OPT$ for a minimization problem
- $SOL \geq (1-\epsilon)OPT$ for a maximization problem

For example, ***A*** may run in time $n^{100/\epsilon}$.

There is a time-accuracy tradeoff.

Knapsack Problem

A set of items, each has different size and different value.

We only have one knapsack.

Goal: to pick a subset which can fit into the knapsack and **maximize the value** of this subset.

Knapsack Problem

(The Knapsack Problem) Given a set $S = \{a_1, \ldots, a_n\}$ of objects, with specified sizes and profits, $\text{size}(a_i)$ and $\text{profit}(a_i)$, and a knapsack capacity B, find a subset of objects whose total size is bounded by B and total profit is maximized.

Assume $\text{size}(a_i)$, $\text{profit}(a_i)$, and B are all integers.

We'll design an approximation scheme for the knapsack problem.

Greedy Methods

General greedy method:
Sort the objects by some rule,
and then put the objects into the knapsack according to this order.

Sort by object size in non-decreasing order: 

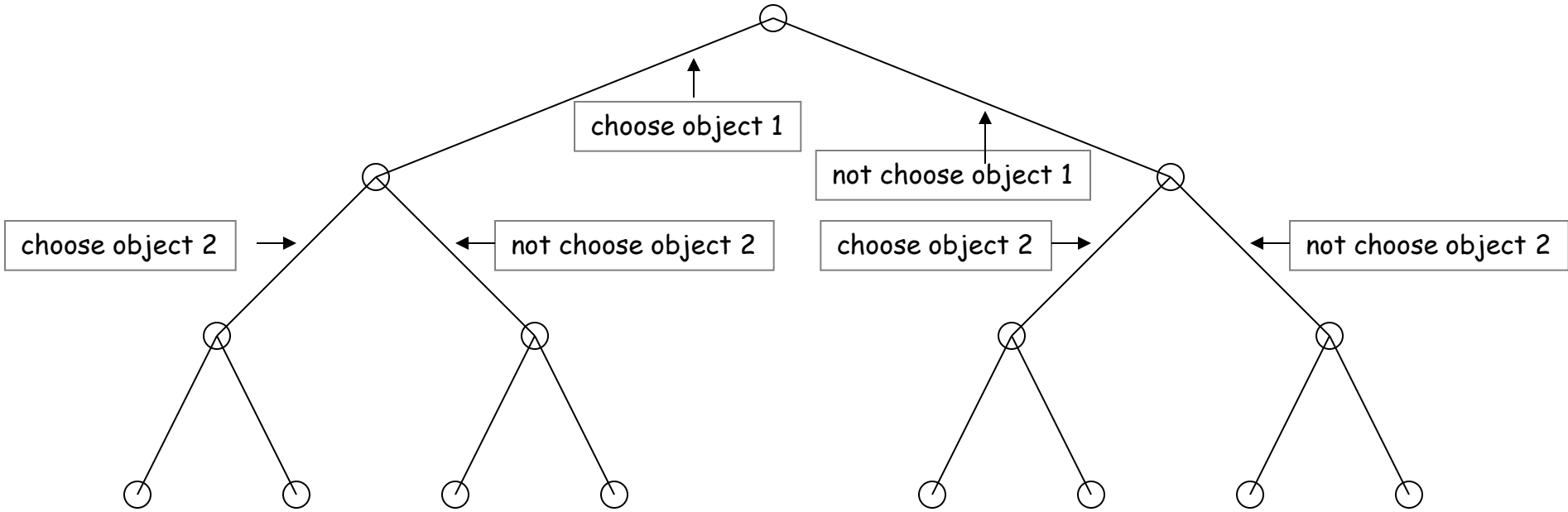
Sort by profit in non-increasing order: 

Sort by profit/object size in non-increasing order: 

Greedy won't work.

Exhaustive Search

n objects, total 2^n possibilities, view it as a search tree.

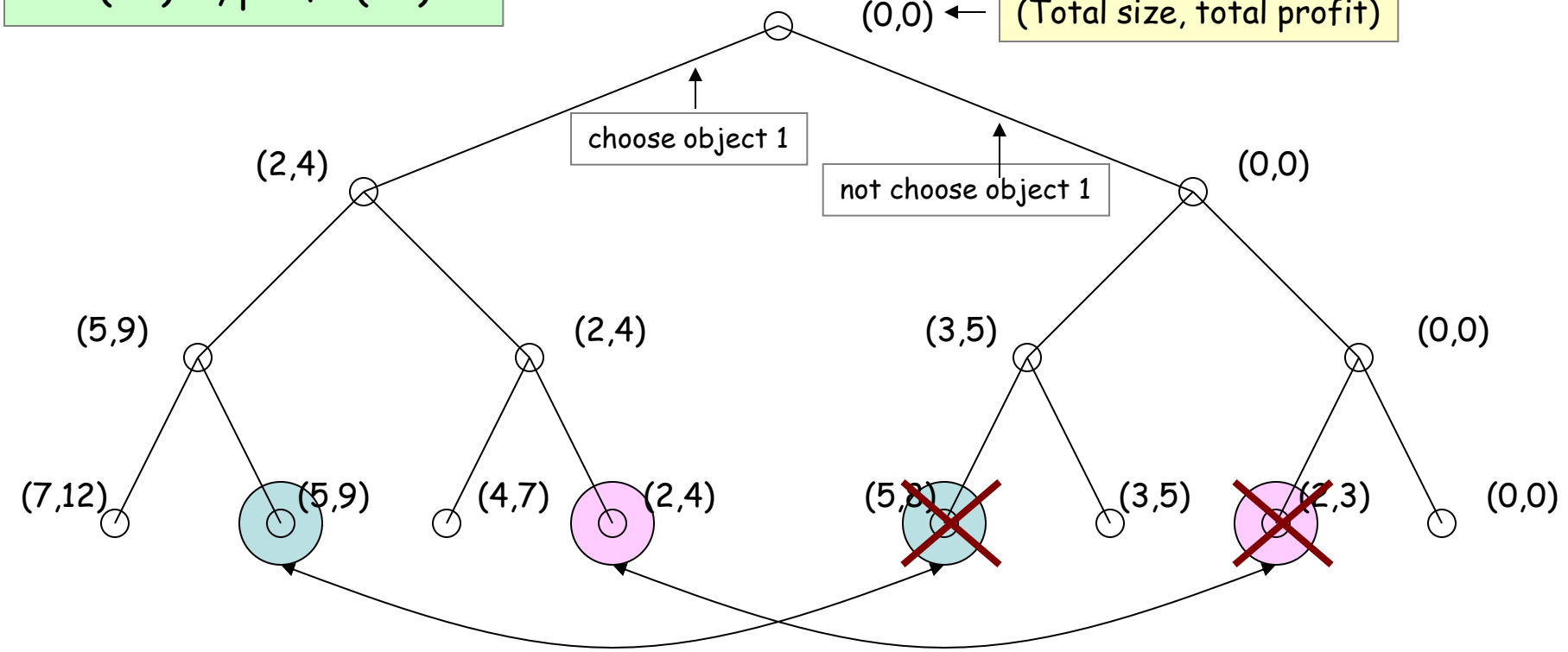


At the bottom we could calculate the total size and total profit, and choose the optimal subset.

Exhaustive Search

size(a1)=2, profit(a1)=4
size(a2)=3, profit(a2)=5
size(a3)=2, profit(a3)=3
size(a4)=1, profit(a4)=2

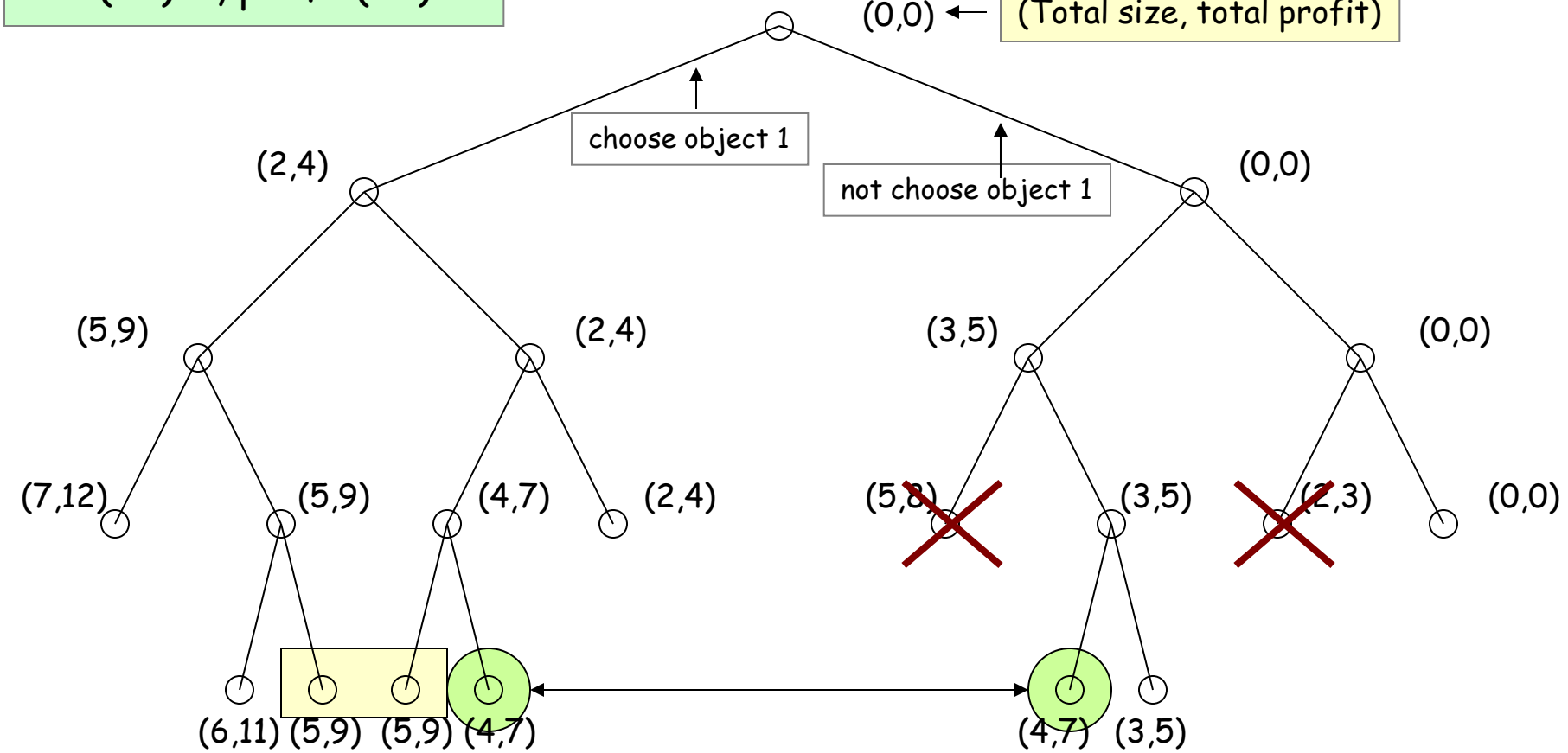
There are many redundancies.



Exhaustive Search

size(a1)=2, profit(a1)=4
size(a2)=3, profit(a2)=5
size(a3)=2, profit(a3)=3
size(a4)=1, profit(a4)=2

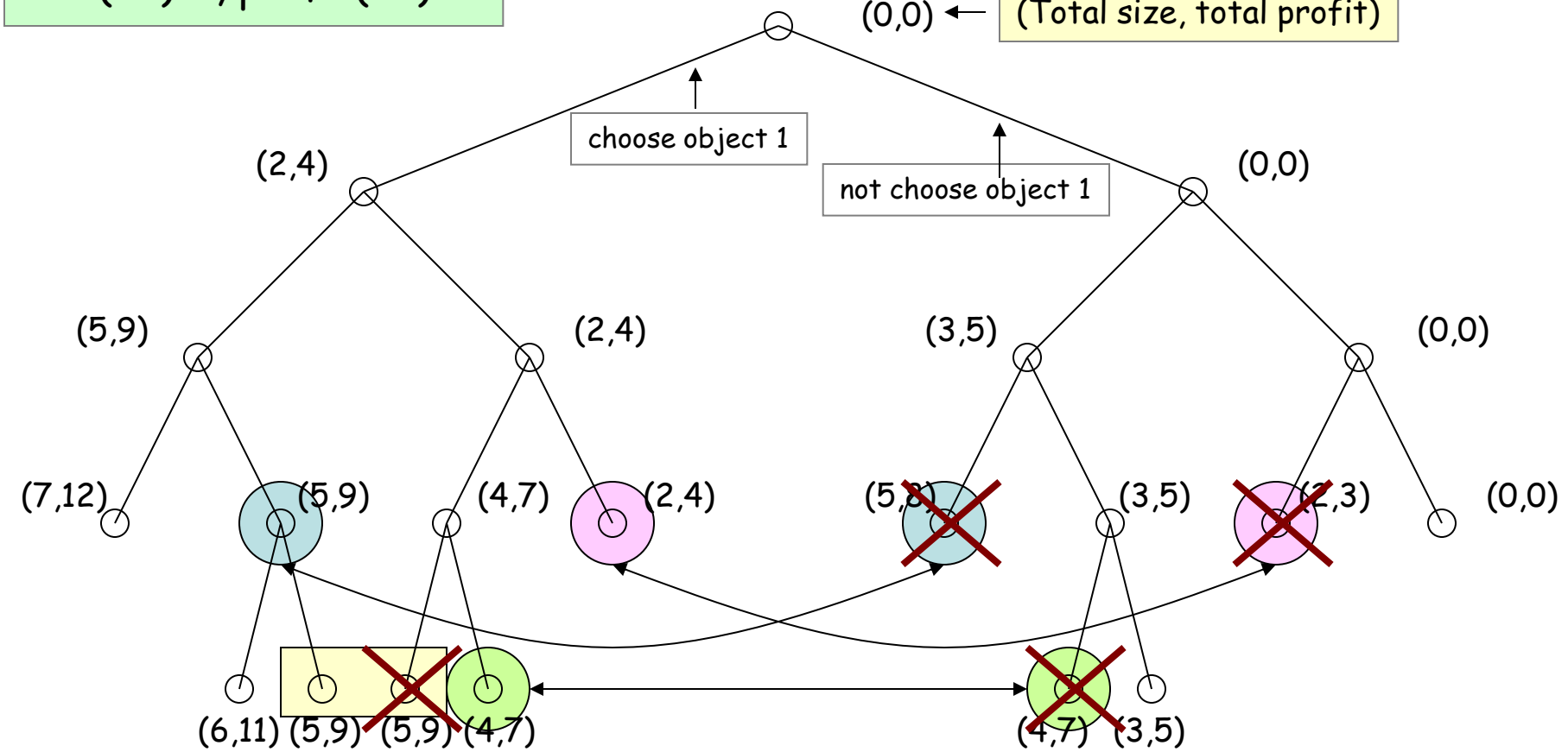
There are many redundancies.



Exhaustive Search

size(a1)=2, profit(a1)=4
size(a2)=3, profit(a2)=5
size(a3)=2, profit(a3)=3
size(a4)=1, profit(a4)=2

There are many redundancies.



The Ideas

Consider two subproblems P and Q at the same level (i.e. same number of objects have been considered).

- If size(P)=size(Q) and profit(P)=profit(Q), just compute either one.
- If size(P)=size(Q) and profit(P)>profit(Q), just compute P.
- If profit(P)=profit(Q) and size(P)>size(Q), just compute Q.

Important: the history doesn't matter (i.e. which subset we chose to achieve profit(P) and size(P)).

Dynamic Programming

Dynamic programming is just exhaustive search with polynomial number of subproblems.

We only need to compute each subproblem once,
and each subproblem is looked up at most a polynomial number of times,
and so the total running time is at most a polynomial.

Dynamic Programming for Knapsack

Suppose we have considered object 1 to object i.
We want to remember what profits are achievable.
For each achievable profit, we want to minimize the size.

Let S(i,p) denote a subset of {a1,…,ai} whose total profit is **exactly** p and total size is **minimized**.

Let A(i,p) denote the size of the set S(i,p)

(A(i,p) = ∞ if no such set exists).

For example, A(1,p) = size(a1) if p=profit(a1),
Otherwise A(1,p) = ∞ (if p ≠ profit(a1)).

Recurrence Formula

Remember: $A(i,p)$ denote the minimum size to achieve profit p using objects from 1 to i.

How to compute $A(i+1,p)$ if we know $A(i,q)$ for all q?

Idea: we either choose object $i+1$ or not.

If we do not choose object $i+1$: then $A(i+1,p) = A(i,p)$.

If we choose object $i+1$:

then $A(i+1,p) = \text{size}(a_{i+1}) + A(i,p-\text{profit}(a_{i+1}))$ if $p > \text{profit}(a_{i+1})$.

$A(i+1,p)$ is the minimum of these two values.

An Example

Remember: A(i,p) denote the minimum size to achieve profit p using objects from 1 to i.

Optimal Solution: max{ p | A(n,p) ≤ B} where B is the size of the knapsack.

size(a1)=2, profit(a1)=4; size(a2)=3, profit(a2)=5;
size(a3)=2, profit(a3)=3; size(a4)=1, profit(a4)=2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	∞	∞	∞	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

An Example

$A(i+1,p) = \min\{A(i,p), size(a_{i+1}) + A(i,p-profit(a_{i+1}))\}$.

$A(2,p) = \min\{A(1,p), \quad A(1,p-5)+3\}$.

size(a1)=2, profit(a1)=4; size(a2)=3, profit(a2)=5;
size(a3)=2, profit(a3)=3; size(a4)=1, profit(a4)=2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	∞	∞	∞	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

An Example

$A(i+1,p) = \min\{A(i,p), size(a_{i+1}) + A(i,p-profit(a_{i+1}))\}$.

$A(3,p) = \min\{A(2,p), A(2,p-3)+2\}$.

size(a1)=2, profit(a1)=4; size(a2)=3, profit(a2)=5;
size(a3)=2, profit(a3)=3; size(a4)=1, profit(a4)=2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	∞	∞	∞	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	0	∞	∞	∞	**2**	**3**	∞	∞	∞	**5**	∞	∞	∞	∞	∞
3	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

An Example

$A(i+1,p) = \min\{A(i,p), size(a_{i+1}) + A(i,p-profit(a_{i+1}))\}$.

$A(4,p) = \min\{A(3,p), A(3,p-2)+1\}$.

size(a1)=2, profit(a1)=4; size(a2)=3, profit(a2)=5;
size(a3)=2, profit(a3)=3; size(a4)=1, profit(a4)=2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	∞	∞	∞	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	0	∞	∞	∞	**2**	**3**	∞	∞	∞	**5**	∞	∞	∞	∞	∞
3	0	∞	∞	**2**	**2**	**3**	∞	**4**	**5**	**5**	∞	∞	**7**	∞	∞
4	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

An Example

$A(i+1,p) = \min\{A(i,p), size(a_{i+1}) + A(i,p-profit(a_{i+1}))\}$.

$A(4,p) = \min\{A(3,p), A(3,p-2)+1\}$.

size(a1)=2, profit(a1)=4; size(a2)=3, profit(a2)=5;
size(a3)=2, profit(a3)=3; size(a4)=1, profit(a4)=2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	∞	∞	∞	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	0	∞	∞	∞	**2**	**3**	∞	∞	∞	**5**	∞	∞	∞	∞	∞
3	0	∞	∞	**2**	**2**	**3**	∞	**4**	**5**	**5**	∞	∞	**7**	∞	∞
4	0	∞	**1**	**2**	**2**	**3**	**3**	**4**	**5**	**5**	**6**	**6**	**7**	∞	**8**

An Example

Remember: A(i,p) denote the minimum size to achieve profit p using objects from 1 to i.

Optimal Solution: max{ p | A(n,p) ≤ B} where B is the size of the knapsack.

For example, if B=8, OPT=14, if B=7, OPT=12, if B=6, OPT=11.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	∞	∞	∞	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	0	∞	∞	∞	**2**	**3**	∞	∞	∞	**5**	∞	∞	∞	∞	∞
3	0	∞	∞	**2**	**2**	**3**	∞	**4**	**5**	**5**	∞	∞	**7**	∞	∞
4	0	∞	**1**	**2**	**2**	**3**	**3**	**4**	**5**	**5**	**6**	**6**	**7**	∞	**8**

Running Time

The input has 2n numbers, say each is at most P.
So the input has total length 2nlog(P).

For the dynamic programming algorithm,
there are n rows and at most nP columns.
Each entry can be computed in constant time (look up two entries).
So the total time complexity is $O(n^2P)$.

The running time is not polynomial if P is very large (compared to n).

Approximation Algorithm

We know that the knapsack problem is NP-complete.

Can we use the dynamic programming technique to design approximation algorithm?

Scaling Down

Idea: scale down the numbers and
compute the optimal solution in this modified instance

- Suppose $P \geq 1000n$.
- Then $OPT \geq 1000n$.
- Now scale down each element by 100 times (profit*:=profit/100).
- Compute the optimal solution using this new profit.
- Can't distinguish between element of size, say 2199 and 2100.
- Each element contributes at most an error of 100.
- So total error is at most 100n.
- This is at most 1/10 of the optimal solution.
- However, the running time is 100 times faster.

Approximation Scheme

Goal: to find a solution which is at least $(1-\epsilon)OPT$ for any $\epsilon > 0$.

Approximation Scheme for Knapsack

1. Given $\epsilon > 0$, let $K = \epsilon P/n$, where P is the largest profit of an object.
2. For each object a_i, define $profit^*(a_i) = \lfloor profit(a_i)/K \rfloor$.
3. With these as profits of objects, using the dynamic programming algorithm, find the most profitable set, say S'.
4. Output S' as the approximate solution.

Quality of Solution

Theorem. Let S denote the set returned by the algorithm. Then, $\text{profit}(S) \geq (1-\epsilon)OPT$.

Proof.

Let O denote the optimal set.

For each object a, because of rounding down,

$K \cdot \text{profit}^*(a)$ can be smaller than $\text{profit}(a)$, but by not more than K.

Since there are at most n objects in O,

$$\text{profit}(O) - K \cdot \text{profit}^*(O) \leq nK.$$

Since the algorithm return an optimal solution under the new profits,

$$\text{profit}(S) \geq K \cdot \text{profit}^*(S) \geq K \cdot \text{profit}^*(O) \geq \text{profit}(O) - nK$$

$$= OPT - \epsilon P \geq (1 - \epsilon)OPT$$

because $OPT \geq P$.

Running Time

For the dynamic programming algorithm,

there are n rows and at most $n\lfloor P/K \rfloor$ columns.

Each entry can be computed in constant time (look up two entries).

So the total time complexity is $O(n^2\lfloor P/K \rfloor) = O(n^3/\epsilon)$.

Therefore, we have an approximation scheme for Knapsack.

Approximation Scheme

Quick Summary

1. Modify the instance by rounding the numbers.
2. Use dynamic programming to compute an optimal solution S in the modified instance.
3. Output S as the approximate solution.

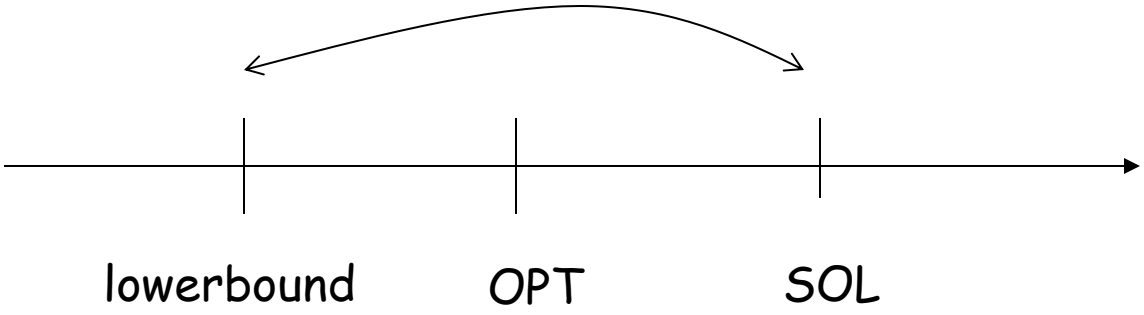
Other examples: bin packing, Euclidean TSP.

Lower bound and Approximation Algorithm

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

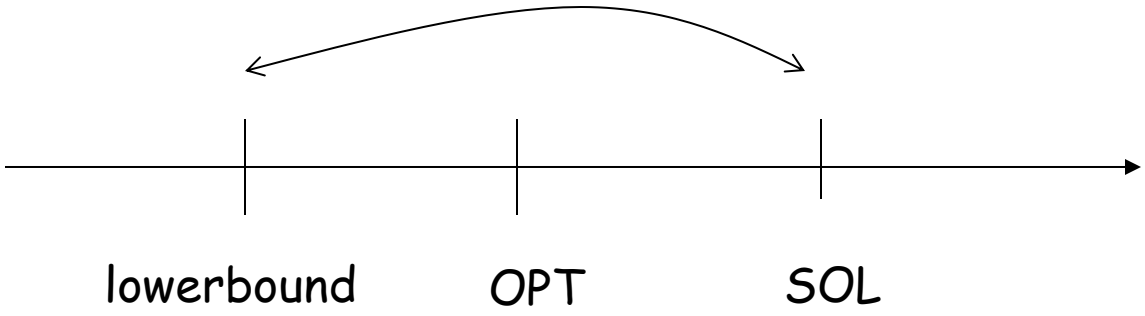
The key of designing a polytime approximation algorithm is to obtain a good (lower or upper) bound on the optimal solution.

The general strategy (for a minimization problem) is:



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

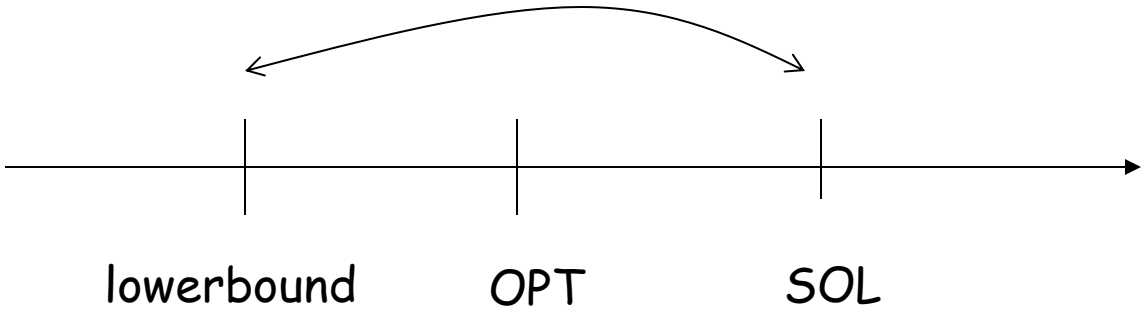
Lower bound and Approximation Algorithm



To design good approximation algorithm, we need a good lowerbound.

For example, if $100 \cdot \text{lowerbound} \leq \text{OPT}$ for some instance,
then if we compare SOL to lowerbound to analyze the performance,
we could not achieve anything better than an 100-approximation algorithm.

Lower bound and Approximation Algorithm

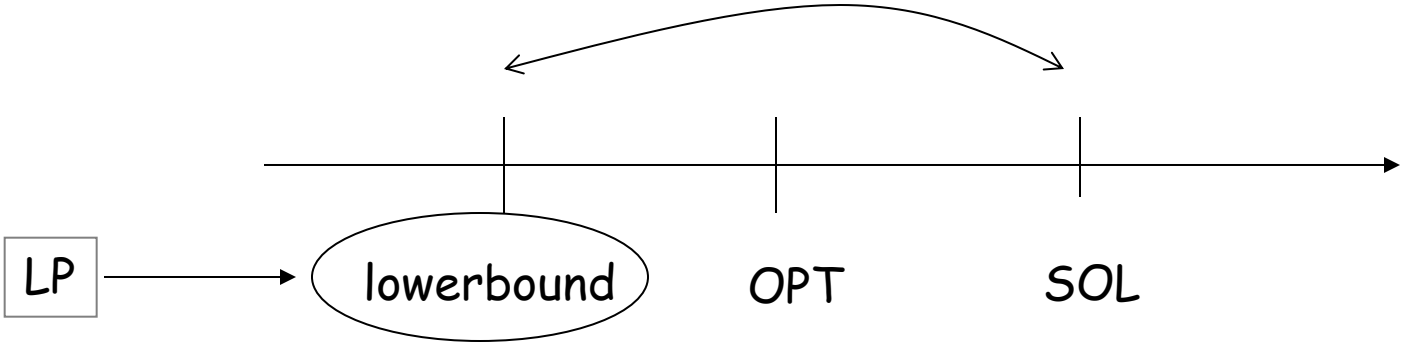


Goal: to find a lowerbound as close to OPT as possible.

In metric TSP and the minimum Steiner tree problem, we use minimum spanning tree as a lowerbound.

In general, it is often difficult to come up with a good lowerbound.

Linear Programming and Approximation Algorithm



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

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

An Example: Vertex Cover

Vertex cover: find a minimum subset of vertices which cover all the edges.

A linear programming relaxation of the vertex cover problem.

$$LP = \min \sum_{v \in V(G)} c_v \cdot x_v$$

$$x_u + x_v \geq 1$$

$$0 \leq x_v \leq 1$$

Clearly, LP is a lowerbound on the optimal vertex cover, because every vertex cover corresponds to a feasible solution of this LP.

An Example: Vertex Cover

$$LP = \min \sum_{v \in V(G)} c_v \cdot x_v$$

$$x_u + x_v \geq 1$$

$$0 \leq x_v \leq 1$$

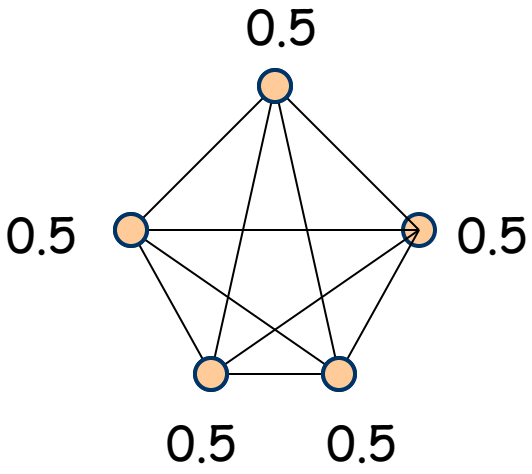
How bad is this LP?

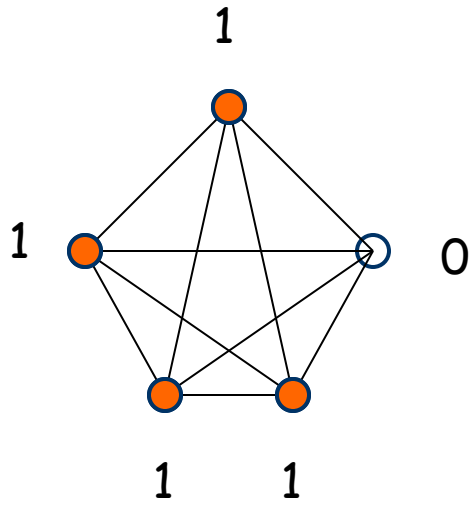
LP = 2.5

LP = n/2

OPT = 4

OPT = n-1





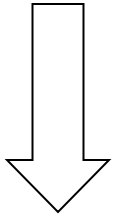
An Example: Vertex Cover

Integrality gap: $= \max_{\text{over all instances.}} \frac{\text{Optimal integer solution.}}{\text{Optimal fractional solution.}}$

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

Half-integrality

Theorem: For the vertex cover problem, every vertex (or basic) solution of the LP is half-integral, i.e. $x(v) = \{0, \frac{1}{2}, 1\}$



There is a 2-approximation algorithm for the vertex cover problem.

The integrality gap of the vertex cover LP is at most 2.

Survivable Network Design

Input

- An undirected graph $G = (V,E)$,
- A cost $c(e)$ on each edge,
- A connectivity requirement $r(u,v)$ for each pair u,v.

Output

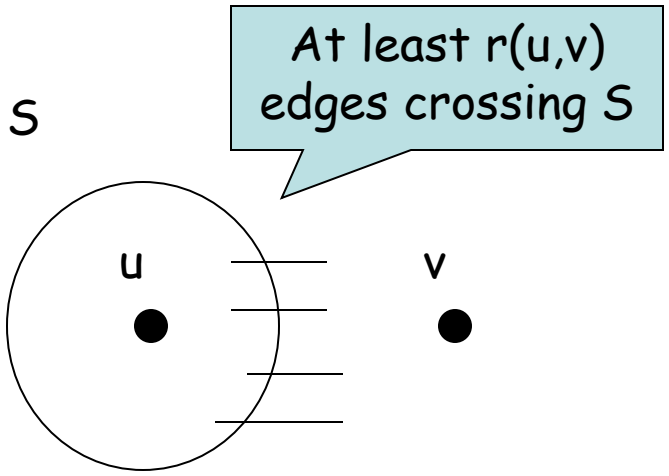
- A minimum cost subgraph H of G which has $r(u,v)$ edge-disjoint paths between each pair u,v.
  (That is, H satisfies all the connectivity requirements.)

Survivable Network Design

- **Minimum spanning tree**: r(u,v) = 1 for all pairs.
- **Minimum Steiner tree**: r(u,v) = 1 for all pair of required vertices.
- **Hamiltonian cycle**: r(u,v) = 2 for all pairs and every edge has cost 1.
- **k-edge-connected subgraph**: r(u,v) = k for all pairs.
- **Minimum cost k-flow**: r(s,t) = k for the source s and the sink t.

Survivable Network Design is NP-complete.

Linear Programming Relaxation



For each set S separating u and v, there should be at least r(u,v) edges "crossing" S.
Let f(S) = max{ r(u,v) | S separates u and v}.

$$LP = \min \sum_{e \in E(G)} c_e \cdot x_e$$

$$\sum_{e \in \delta(S)} x_e \geq f(S)$$

← for each subset S of V

$$x_e \geq 0$$

Separation Oracle

$$LP = \min \sum_{e \in E(G)} c_e \cdot x_e$$

$$\sum_{e \in \delta(S)} x_e \geq f(S) \quad \leftarrow \text{for each subset S of V}$$

$$x_e \geq 0$$

There are exponentially many constraints, but this LP can still be solved in polynomial time by the ellipsoid method. The reason is that we can design a polynomial time separation oracle to determine if x is a feasible solution of the LP.

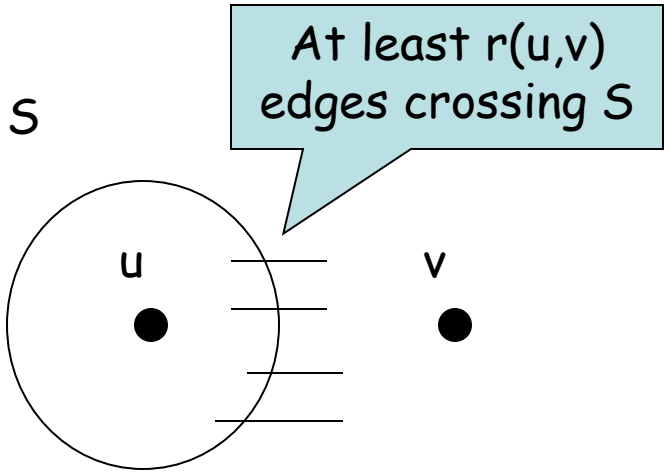
Separation Oracle

$$LP = \min \sum_{e \in E(G)} c_e \cdot x_e$$

$$\sum_{e \in \delta(S)} x_e \geq f(S)$$ ← for each subset S of V

$$x_e \geq 0$$

Remember: f(S) = max{ r(u,v) | S separates u and v}.



Max-Flow Min-Cut

Every (u,v)-cut has at least r(u,v) edges

if and only if

there are r(u,v) flows from u to v.

Separation oracle: check if each pair u,v has a flow of r(u,v)!

Special Case: Minimum Spanning Tree

$$LP = \min \sum_{e \in E(G)} c_e \cdot x_e$$

$$\sum_{e \in \delta(S)} x_e \geq 1$$

$$x_e \geq 0$$

for each subset S of V

How bad is this LP?

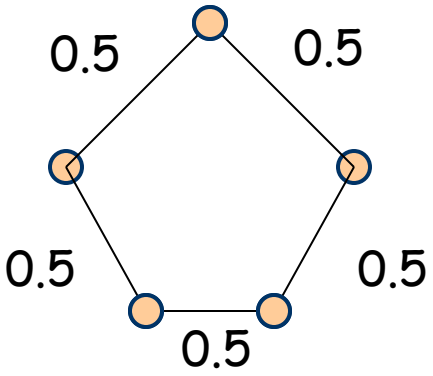
LP = 2.5

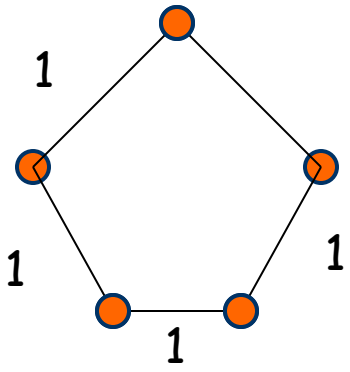
LP = n/2

OPT = 4

OPT = n-1

Cannot even solve minimum spanning tree!

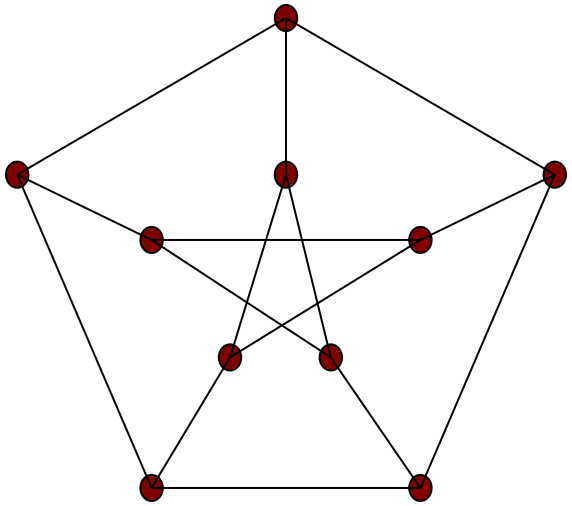




Half-integrality

Does the LP has half-integral optimal solution?

Consider the minimum spanning tree problem, i.e. f(S)=1 for all S.



Peterson Graph

All 1/3 is a feasible solution and has cost 5.

Any half-integral solution having cost 5 must be a Hamiltonian cycle.

But Peterson graph does not have a Hamiltonian cycle!

So, **no** half-integral optimal solution!

Linear Programming Relaxation

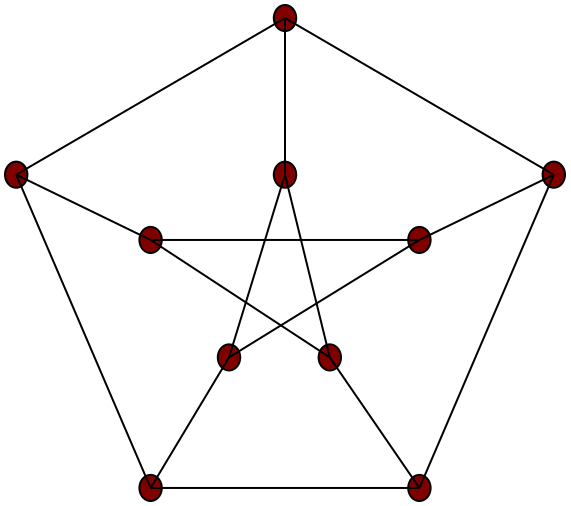
$$LP = \min \sum_{e \in E(G)} c_e \cdot x_e$$

$$\sum_{e \in \delta(S)} x_e \geq f(S)$$ ← for each subset S of V

$$x_e \geq 0$$

What is the integrality gap of this LP?

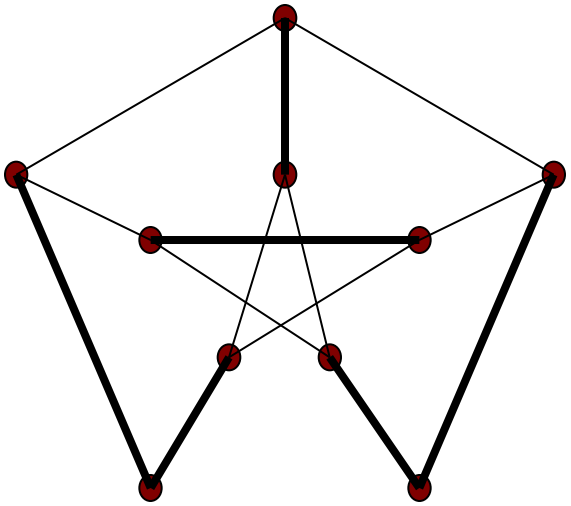
Moment of Inspiration



All 1/3 is a feasible solution.

But this is not a vertex solution!

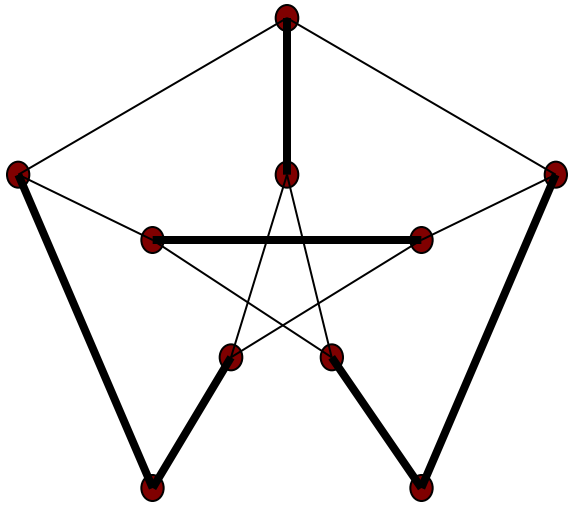
This is a vertex solution.



Thick edges have value 1/2;
Thin edges have value 1/4.

Structural Result of the LP

Kamal Jain



Theorem. Every vertex solution has an edge with value at least 1/2

Corollary. There is a 2-approximation algorithm for survivable network design.

Iterative Rounding

Initialization: $H = \emptyset$, $f' = f$.

While $f' \neq 0$ do:

- Find a vertex solution, x, of the LP with function f'.
- Add every edge with $x(e) \geq 1/2$ into H.
- Update f' for every set S, set $f'(S) \leftarrow f(S) - |\delta_H(S)|$.

Output H.

A new vertex solution is computed in each iteration

Guaranteed to exist

Update the connectivity requirements.

Analysis

Corollary. There is a 2-approximation algorithm for survivable network design.

Intuitive reason: we only pick an edge when the LP picks at least half.

Proof: Let say we pick an edge e.

Key: LP-c(e)x(e) is a feasible solution for the next iteration.

$$\begin{aligned} \text{cost}(H) &= c(e) + \text{cost}(H') \le 2 \cdot c(e)x(e) + \text{cost}(H') \\ &\le 2 \cdot c(e)x(e) + 2(LP - c(e)x(e)) \le 2LP \\ &\le 2OPT. \end{aligned}$$

Some Remarks

1. The iterative rounding algorithm performs very well in practice.

2. No combinatorial algorithm has a performance ratio better than $O(\log n)$.