Introduction to
Approximation Algorithms

Mong-Jen Kao (高孟駿)

Friday 13:20 – 15:10

Design & <u>Analysis</u>

of Approximation Algorithms

Some general concepts.

Finding & Deriving the Bounds

- A great part of Approximation Algorithms is about ***finding bounds***.
  - ***Upper-bounds*** / ***Lower-bounds***

    for

    Our algorithm / ***Optimal solution***

Takes ***some imaginations*** and ***sometimes deep observations***.

Often more conceivable.

Let's try to *review this part for every algorithm* we are talking about.

Outline

- Metric Steiner Tree
  - Factor-Preserving Reduction
  - MST-based 2-approximation
- Metric Traveling Salesman Problem (TSP)
  - 3/2-approximation
  - A PTAS for the Euclidean TSP*

Metric Traveling Salesman Problem (TSP)

The Traveling Salesman Problem (TSP)

- Given a complete graph $G = (V, E)$ with *nonnegative edge costs*, find a minimum cost cycle *visiting every vertex* exactly once.
  - This is the most general form of the TSP problem.
  - However, this problem *cannot be approximated* at all.

Theorem.

The TSP problem cannot be approximated to a factor of $\alpha(n)$, for any polynomial-time computable function $\alpha(n)$, unless **P = NP**.

The (Metric) Traveling Salesman Problem (TSP)

- Given a complete graph $G = (V, E)$ with *nonnegative edge costs that satisfy* ***the triangle inequality***, find a minimum cost cycle *visiting every vertex* exactly once.

A Simple 2-approximation Algorithm

Algorithm A for Metric TSP

1. Compute an MST T for G.
2. Double every edge of the T to obtain an Eulerian graph.
3. Find an Eulerian tour τ on this graph.
4. Shortcutting τ to obtain a TSP tour C and output C.

An Improved $3/2$-approximation Algorithm

Algorithm A for Metric TSP

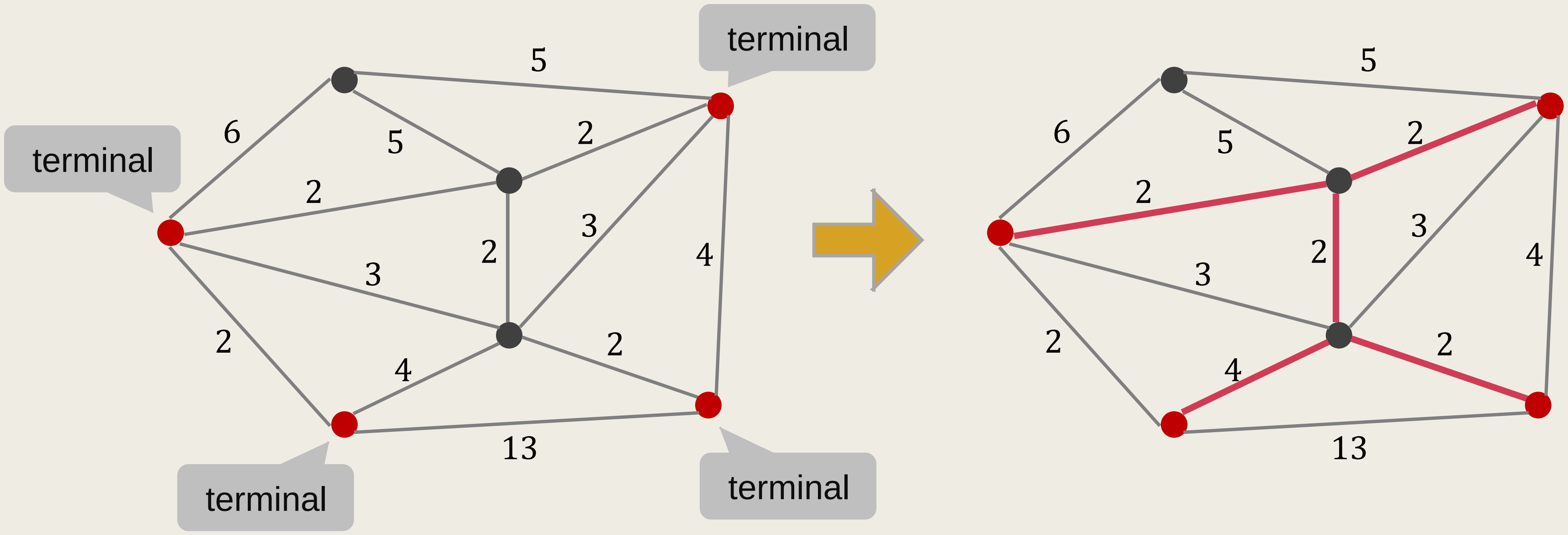
1. Compute an MST T for G.
2. Compute a ***min-cost perfect matching*** M on the set of odd-degree vertices of T.
3. Add M to T to obtain an Eulerian graph.
4. Find an Eulerian tour τ on this graph.
5. Shortcutting τ to obtain a TSP tour C and output C.

The Metric Steiner Tree Problem

The Graph Steiner Tree Problem

- Given an undirected graph $G = (V, E)$ with nonnegative edge weight and a subset of vertices $A \subseteq V$, called the terminals,

  the Steiner tree problem is to compute a ***minimum weight tree*** *in* G ***<u>that contains all the terminals</u>*** *of* A.



The Graph Steiner Tree Problem

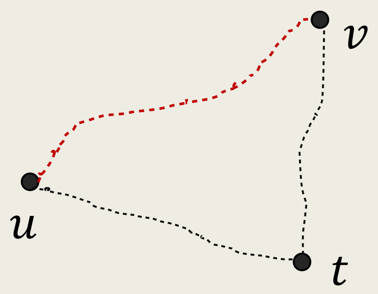
- The graph Steiner tree problem is one type of *min-cost connected subgraph* problems in graphs.
  - When all vertices are terminals, i.e., $A = V$, the problem is exactly the *Minimum Spanning Tree* (MST) problem.
  - When the number of terminals is two, i.e., $|A| = 2$, the problem becomes the *shortest path* (SP) problem.
  - The Steiner tree problem addresses the rest situations in between.

We will see that, *it reduces* to *the Metric Steiner Tree Problem*.

The (Metric) Steiner Tree Problem

- In the (Metric) Steiner Tree problem, we are given as input :
  - An undirected ***complete graph*** $G = (V, E)$,
  - An edge weight function $w : V \to R^{\geq 0}$ that satisfies *the **triangle inequality**, i.e.,*

$$w(u,v) \leq w(u,t) + w(t,v) \quad \forall u,v,t \in V, \text{and}$$



  - A set of terminals $A \subseteq V$,

The goal is to compute a *minimum weight tree in* G that spans all the terminals of A.

A Factor-Preserving Reduction from Graph Steiner Tree to Metric Steiner Tree

Hence, it suffices to consider the metric case.

Approximation Factor Preserving Reduction

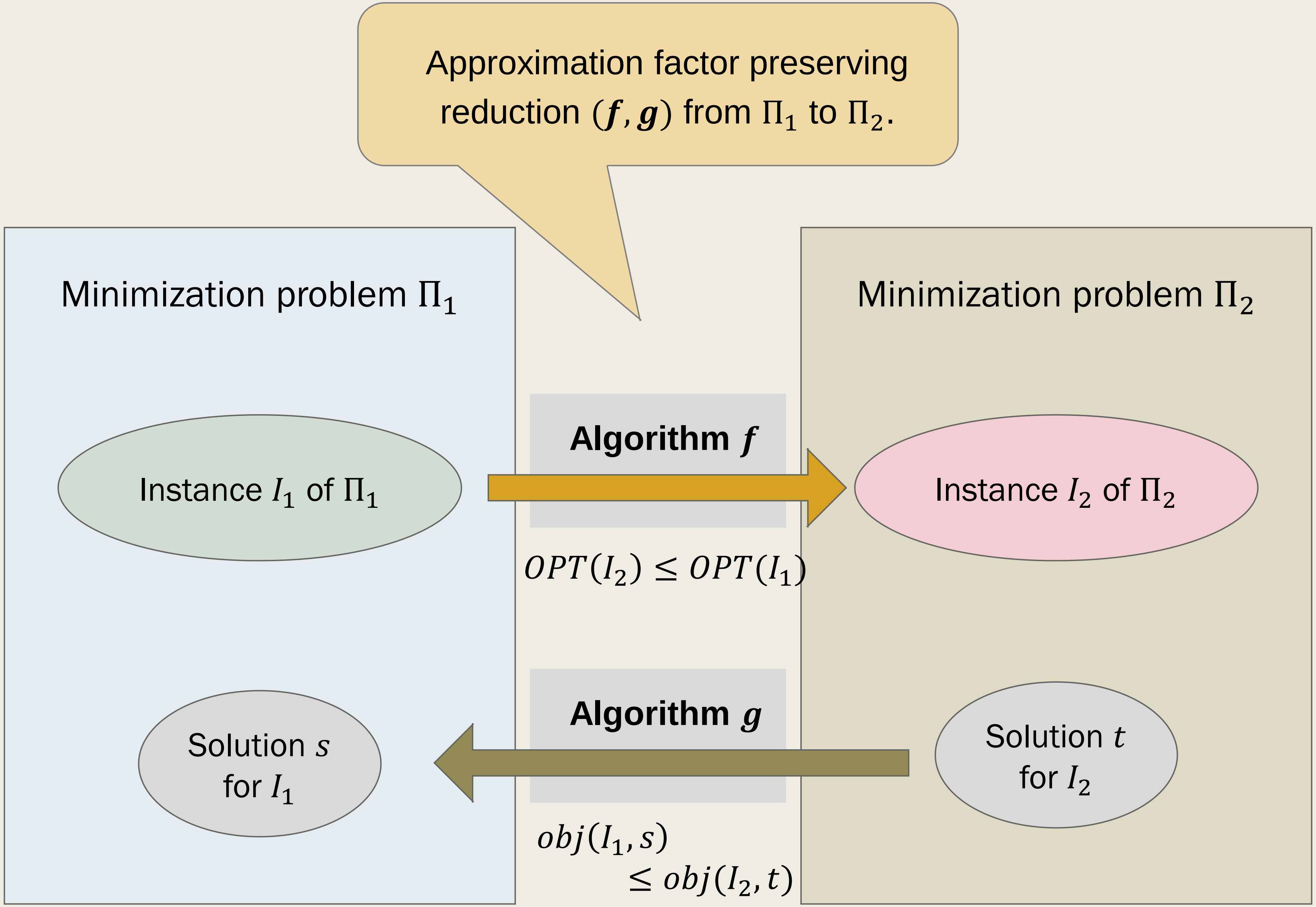
- Let Π_1, Π_2 be two optimization problems.
  An approximation factor preserving reduction from Π_1 to Π_2 consists of two <u>*polynomial-time algorithms*</u> f *and* g, such that

  – For any instance I_1 of Π_1,
    $I_2 \coloneqq f(I_1)$ is an instance of Π_2 whose optimal value is <u>*no worse than*</u> I_1.

  – For any solution t of I_2,
    $s \coloneqq g(I_1, t)$ is a solution of I_1 whose objective is <u>*no worse than*</u> that of t.

For <u>*minimization problems*</u>, the definition requires

- $OPT_{\Pi_2}(I_2) \le OPT_{\Pi_1}(I_1)$
- $obj_{\Pi_1}(I_1, s) \le obj_{\Pi_2}(I_2, t)$

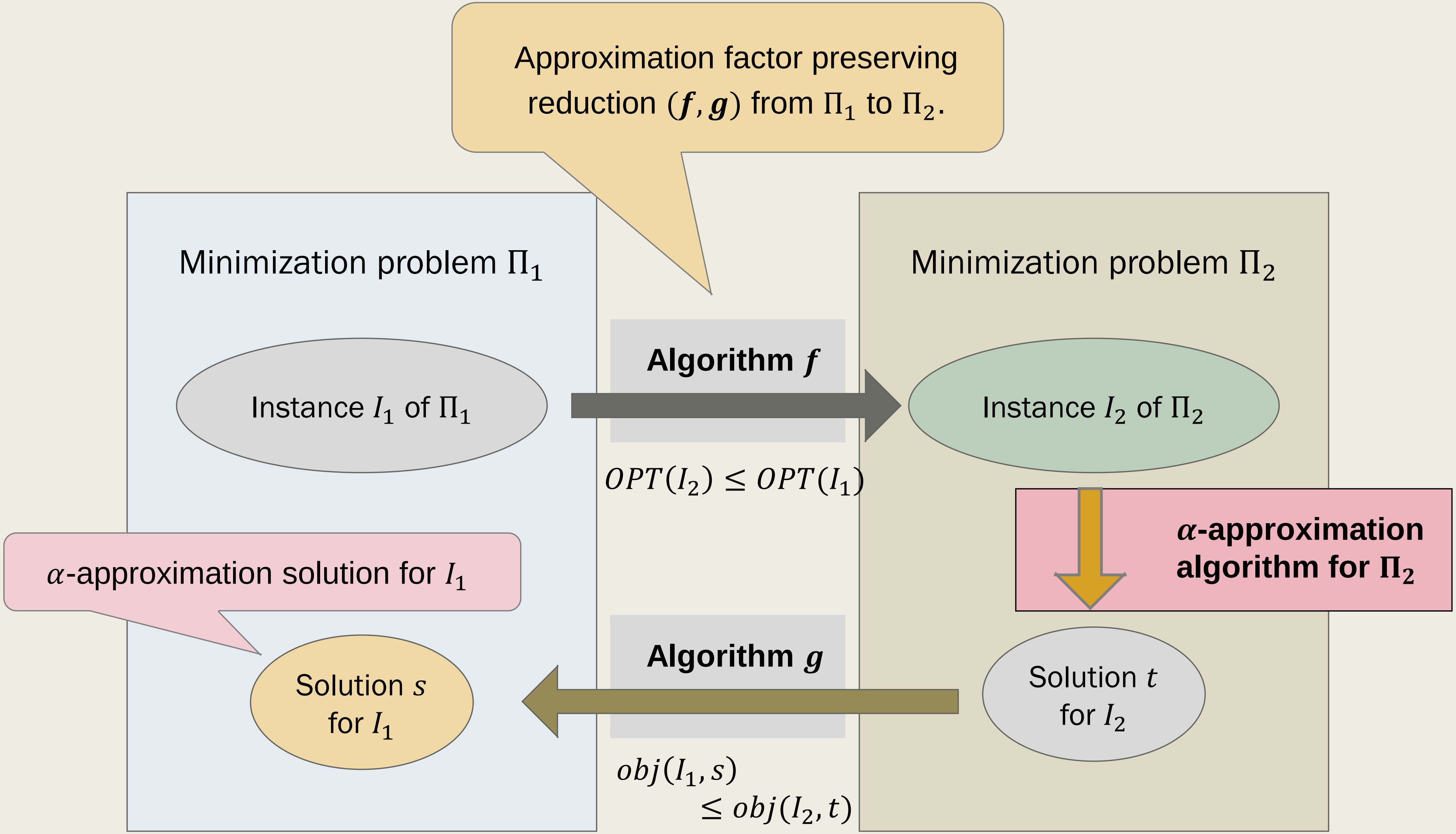


Approximation factor preserving reduction $(\boldsymbol{f}, \boldsymbol{g})$ from Π_1 to Π_2.
Minimization problem Π_1
Minimization problem Π_2
Algorithm $\boldsymbol{f}$
Instance I_1 of Π_1
Instance I_2 of Π_2
$OPT(I_2) \leq OPT(I_1)$
Algorithm $\boldsymbol{g}$
Solution s for I_1
Solution t for I_2
$obj(I_1, s) \leq obj(I_2, t)$

Approximation Factor Preserving Reduction

- Let (f, g) be an approximation factor preserving reduction from Π_1 to Π_2. Then, from the definition, it follows that
  - $OPT_{\Pi_1}(I_1) = OPT_{\Pi_2}(I_2)$, where $I_2 \coloneqq f(I_1)$.
  - An α-approximation algorithm for Π_2 gives an α-approximation solution for Π_1 via g.

- Provided that such a reduction exists,
  to approximate Π_1, it suffices to develop approximation algorithms for Π_2.



Approximation factor preserving reduction $(\boldsymbol{f}, \boldsymbol{g})$ from Π_1 to Π_2.
Minimization problem Π_1
Minimization problem Π_2
Algorithm $\boldsymbol{f}$
Instance I_1 of Π_1
Instance I_2 of Π_2
$OPT(I_2) \leq OPT(I_1)$
$\boldsymbol{\alpha}$-approximation algorithm for $\boldsymbol{\Pi_2}$
α-approximation solution for I_1
Algorithm $\boldsymbol{g}$
Solution s for I_1
Solution t for I_2
$obj(I_1, s) \leq obj(I_2, t)$

Lemma.

There is an approximation factor preserving reduction
from the graph Steiner tree problem to the metric Steiner tree problem.

- Let $I = (G = (\mathrm{V}, \mathrm{E}), w, A)$ be an instance of the graph Steiner tree problem.
- We create an instance $\boldsymbol{I'} = (\boldsymbol{G'}, \boldsymbol{w'}, \boldsymbol{A})$ for the metric Steiner tree problem as follows.
  - Let G' be the complete graph defined on V.
  - For each $u, v \in V$, define $w'(u, v) \coloneqq d_w(u, v)$, where $d_w(u, v)$ is the <u>*shortest distance*</u> between u and v in G with respect to w.
    - That is, we define (G', w') to be the *closure* of (G, w).

Lemma.

There is an approximation factor preserving reduction
from the graph Steiner tree problem to the metric Steiner tree problem.

- Clearly, the construction can be done in polynomial time.
- Let T be an optimal Steiner tree for I.

  Then,

$$w'(T) = \sum_{(u,v)\in T} w'(u,v) \leq \sum_{(u,v)\in T} w(u,v) = w(T)\,.$$

  - Since T is also a Steiner tree for I',
    for any optimal Steiner tree T' for I', we have $w'(T') \leq w'(T)$.
  - Hence, $OPT(I') = w'(T') \leq w(T) = OPT(I)$.

Lemma.

There is an approximation factor preserving reduction from the graph Steiner tree problem to the metric Steiner tree problem.

- Let T' be a Steiner tree for I'.
- From T', construct a Steiner tree T for I as follows.
  1. Replace each edge of T', say, edge (u, v), by a shortest path between u and v in G with respect to w.
     Let H be the resulting graph.
  2. Break cycles in H arbitrarily to get a tree.
     Let it be T.
- Clearly, the construction is in polynomial time.

Lemma.

There is an approximation factor preserving reduction
from the graph Steiner tree problem to the metric Steiner tree problem.

- By the construction of T,

$$A \subseteq V(T') \subseteq V(H) = V(T)$$

  and T is a Steiner tree for I.

- We also have

$$w(T) \leq w(H) \leq \sum_{u,v \in V} w'(u,v) = w'(T').$$

- Hence $obj(I,T) \leq obj(I',T')$.

This completes the reduction.

(Brief)

Status of the Steiner Tree Problem

The Steiner Tree Problem

- The Steiner tree problem is *NP-hard*.
  - It is also *APX-complete*, which means that, unless *P* = *NP*, it is not possible to approximate this problem arbitrarily close to 1.
- This problem can be approximated to $\ln(4) \approx 1.39$ by Linear Programming (LP) and iterative randomized rounding techniques. [Byrka et al., *STOC*, 2010]
  - Approximating this problem within a ratio $96/95 \approx 1.0105$ is *NP-hard*.
- This problem is an *important fundamental problem* and has practical applications in circuit layout and network designs.

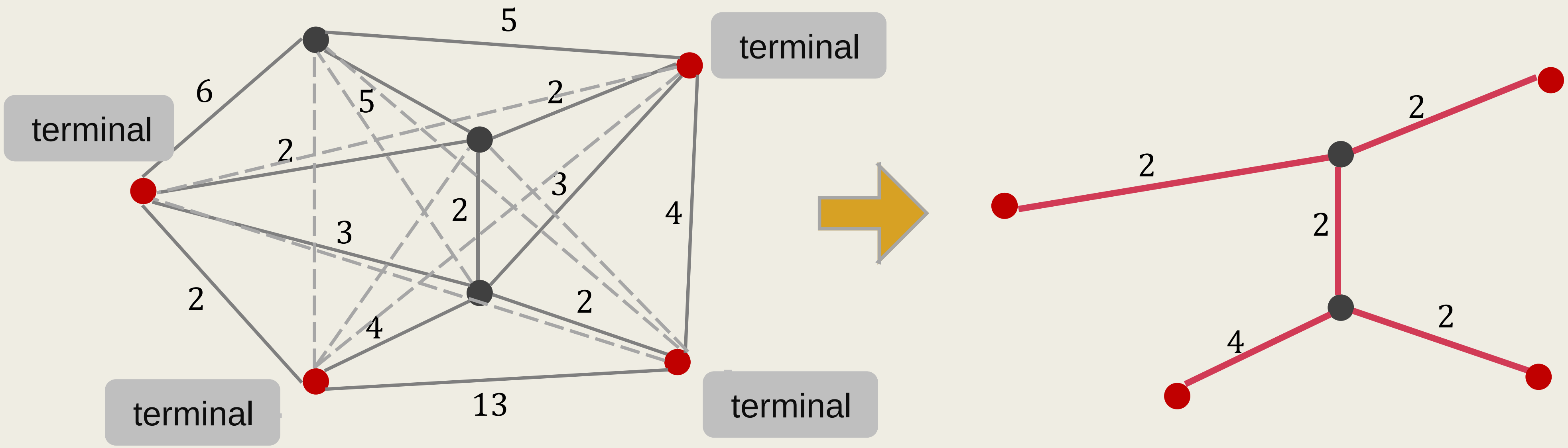
The Steiner Tree Problem

- When the underlying metric is Euclidean,
  i.e., the Euclidean Steiner tree, there is a PTAS.
- Many special cases and further generalizations have been considered.
  For example, the rectilinear Steiner tree, further connectivity constraints, parameterized complexity, etc.
- In this lecture,
  we will examine a simple 2-*approximation* via an *MST-based* algorithm.

MST-Based 2-Approximation

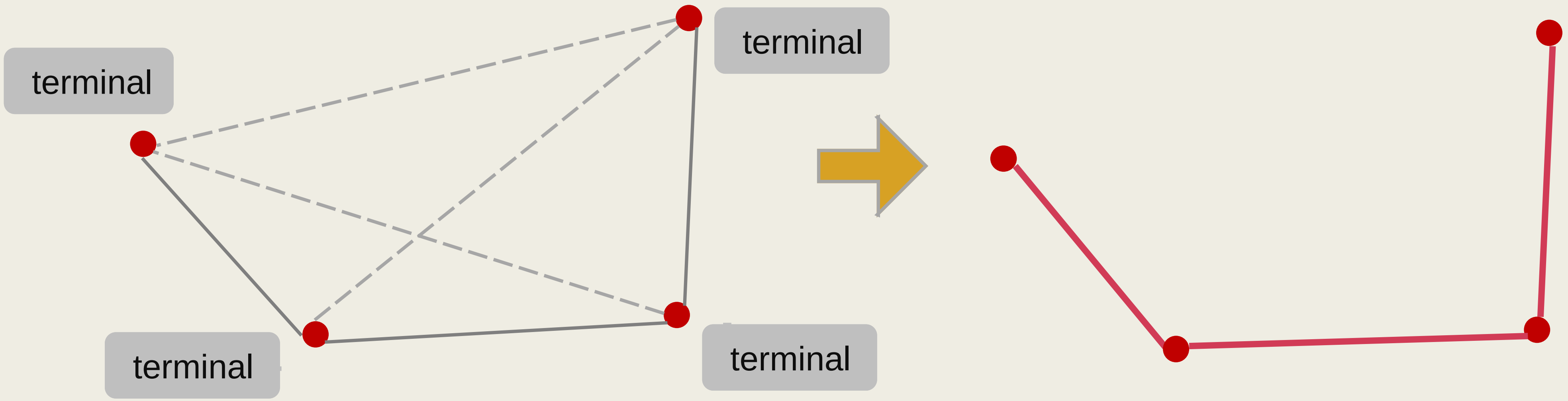
Connecting the Terminals – How?

- The goal of the Steiner tree problem is to connect the terminals in G, using a tree structure.
  - Non-terminal nodes can be used if necessary.



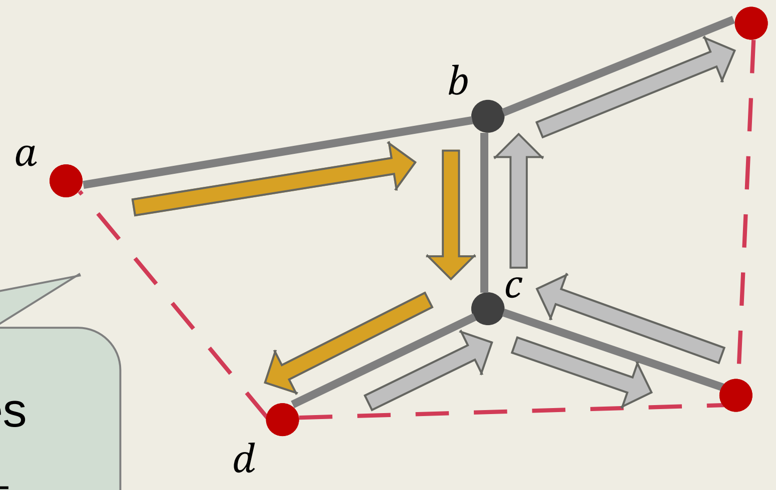
Connecting the Terminals – How?

- Minimum spanning tree (MST) connects the given set of vertices *using the minimum cost possible*.
  - Intuitively, *with triangle inequality*, the cost of MST should not be too bad compared to the optimal Steiner tree.



The Price of Ignoring All Non-terminal Nodes

- Locally, triangle inequality guarantees low-cost when non-terminal nodes are ignored.
  - Globally, ignoring all non-terminal nodes doesn't seem to behave too bad, either.



Triangle inequality promises that $\overline{ad} \leq \overline{ab} + \overline{bc} + \overline{cd}$.

This key observation leads us to a 2-approximation guarantee.

2-Approximation Algorithm for Steiner Tree

- Let $I = (G = (\mathrm{V}, \mathrm{E}), w, A)$ be an instance of Steiner tree.
- The algorithm goes as follows.

1. Compute an MST T of the induced subgraph of A in G, i.e., the graph consists of the vertices in A and the edges.
2. Output T as the approximate Steiner tree for I.

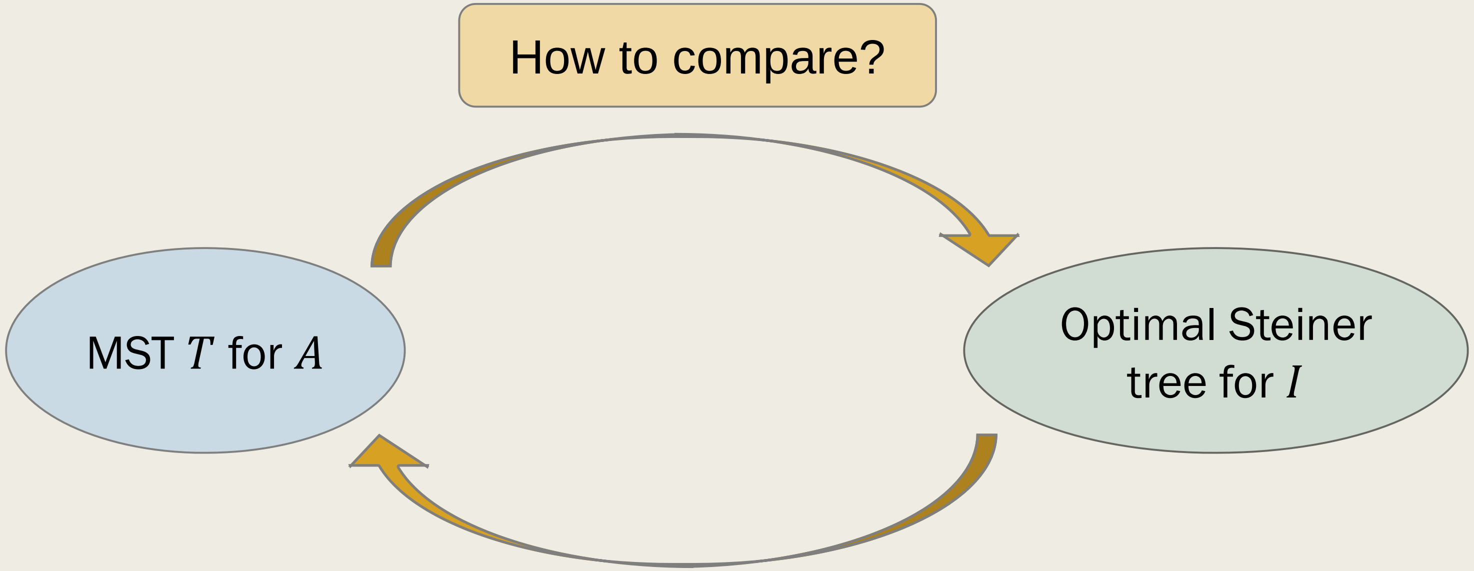
<u>Question</u>:

How do we bound the cost of T in terms of OPT_I ?

How do we compare T to OPT_I ?

- Intuitively, we feel that, *MST does not perform too bad*.

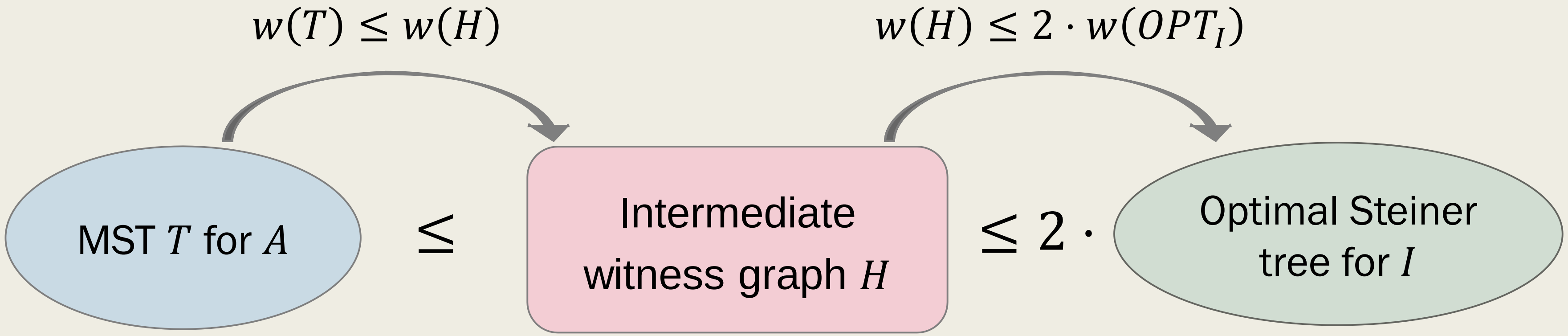
  How do we *formally establish a bound* ?



Use an *Intermediate Witness*

- We will show that, we can *construct from* OPT_I a witness graph H, such that

$$w(T) \leq w(H) \leq 2 \cdot w(OPT_I).$$



Use an *Intermediate Witness*

- We will show that, we can *construct from* OPT_I a witness graph H, such that

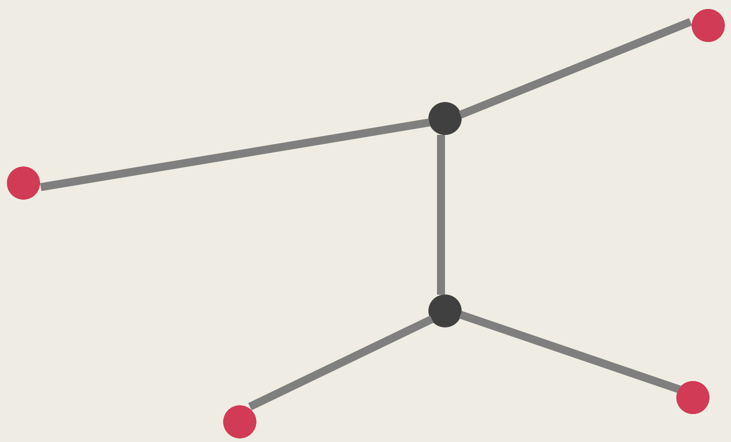
$$w(T) \leq w(H) \leq 2 \cdot w(OPT_I)\,.$$

- Note that, the construction of H is imaginary, and the graph H is *only used in the analysis*.
  - Since OPT_I *is unknown*,
    we cannot actually construct anything from it.

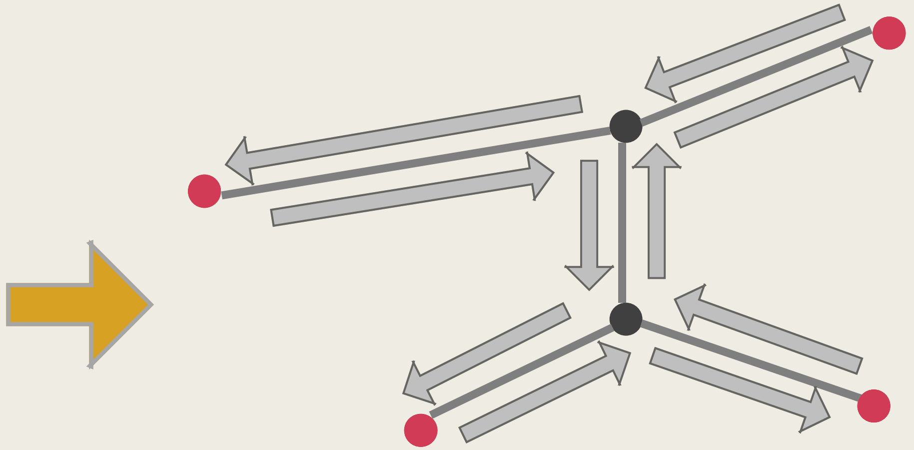
Constructing the graph H

Clearly, $w(K) = 2 \cdot w(OPT_I)$, since each edge in OPT_I is traversed exactly most twice.

- Let OPT_I be an optimal Steiner tree for I.
    1. Use DFS to compute an Eulerian trail of OPT_I. Let it be K.



OPT_I

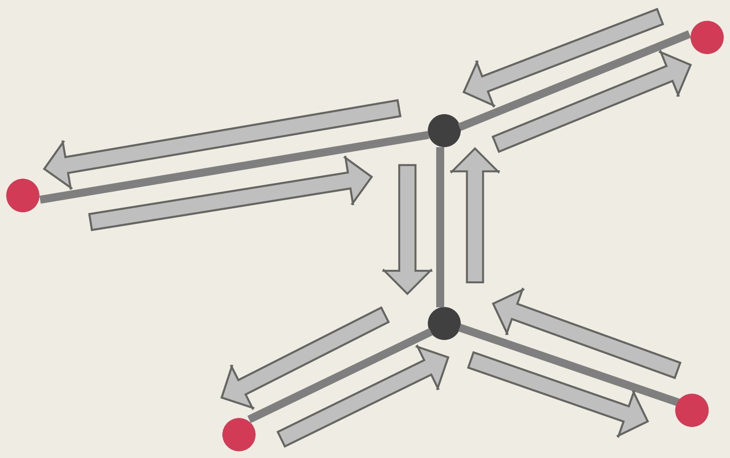


The Eulerian trail K

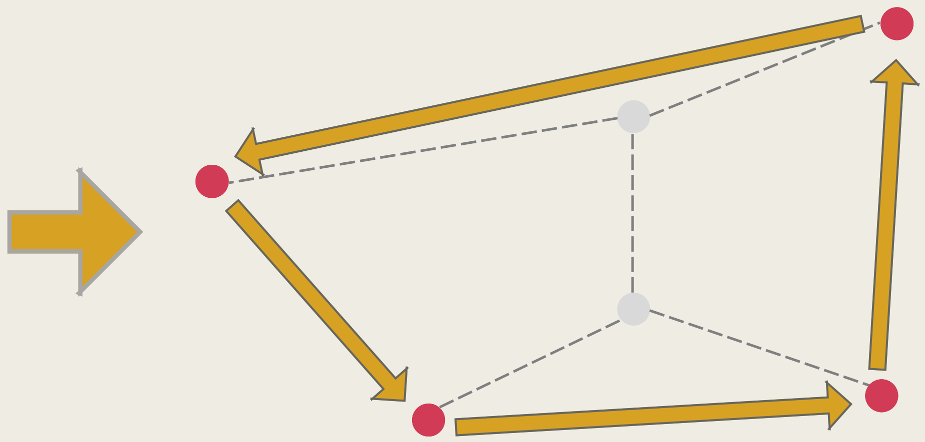
- Let OPT_I be an optimal Steiner tree for I.
  1. Use DFS to compute an Eulerian trail of OPT_I.
     Let it be K.
  2. *Shortcut the trail* K by discarding the non-terminal vertices between terminal vertices.
     Let H be the resulting *Hamiltonian cycle* on A.

Clearly, $w(K) = 2 \cdot w(OPT_I)$, since each edge in OPT_I is traversed exactly most twice.

Clearly, $w(H) \leq w(K) = 2 \cdot w(OPT_I)$.



The Eulerian trail K



The Hamiltonian cycle H on A

The cost of H

- We have created a Hamiltonian cycle H on A from OPT_I with $w(H) \leq 2 \cdot w(OPT_I)$.
  - Since T is an MST for A, we must have $w(T) \leq w(H)$.
  - Hence, $w(T) \leq 2 \cdot w(OPT_I)$,
    and T is a 2-approximation for I.

What we have actually done

- From our key observation, we have shown that,
  there exists a Hamiltonian cycle of A with cost at most $2 \cdot w(OPT_I)$.
- Then, MST always performs no worse than H.