Lecture 15: Approximation Algorithms for Travelling Salesman Problem

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

June 29, 2021

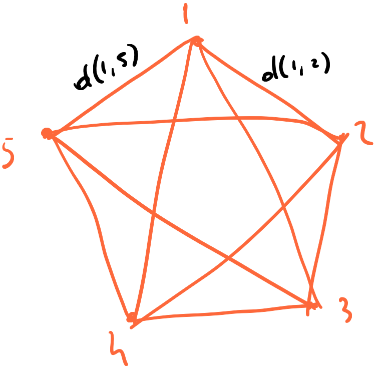
Overview

- Equivalent Versions of Traveling Salesman Problem
- Approximation Algorithms for Traveling Salesman Problem
- Conclusion
- Acknowledgements

Traveling Salesman Problem

- **Input:** set of points X and a symmetric distance function

$$d : X \times X \to \mathbb{R}_{\geq 0}$$



$$d(i,j) = d(j,i) \geq 0$$

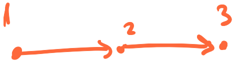
Traveling Salesman Problem

- **Input:** set of points X and a symmetric distance function

$$d : X \times X \to \mathbb{R}_{\geq 0}$$

- For any path $p_0 \to p_1 \to \cdots \to p_t$ in X, *length* of the path is sum of distances traveled

$$\sum_{i=0}^{t-1} d(p_i, p_{i+1})$$



$d(1,2) + d(2,3)$

Traveling Salesman Problem

- **Input:** set of points X and a symmetric distance function

$$d : X \times X \rightarrow \mathbb{R}_{\geq 0}$$

- **Output:** find a cycle that reaches all points in X of shortest length.

Traveling Salesman Problem

- **Input:** set of points X and a symmetric distance function

$$d : X \times X \to \mathbb{R}_{\geq 0}$$

- **Output:** find a cycle that reaches all points in X of shortest length.
- Definitely a problem we would like to solve
  - Efficient route planning (mail system, shuttle bus pick up and drop off...)

Traveling Salesman Problem

- **Input:** set of points X and a symmetric distance function

$$d : X \times X \to \mathbb{R}_{\geq 0}$$

- **Output:** find a cycle that reaches all points in X of shortest length.
- Definitely a problem we would like to solve
  - Efficient route planning (mail system, shuttle bus pick up and drop off...)
- One of the famous NP-complete problems

Traveling Salesman Problem

- **Input:** set of points X and a symmetric distance function

$$d : X \times X \to \mathbb{R}_{\geq 0}$$

- **Output:** find a cycle that reaches all points in X of shortest length.
- Definitely a problem we would like to solve
  - Efficient route planning (mail system, shuttle bus pick up and drop off...)
- One of the famous NP-complete problems
- Comes in many flavours...

Variants of TSP

1. *General* TSP *without* repetitions (General TSP-NR)

Variants of TSP

1. *General* TSP *without* repetitions (General TSP-NR)
    - **Input:** X and symmetric distance function $d : X \times X \to \mathbb{R}_{\geq 0}$
    - **Output:** find a cycle of shortest length that reaches each point of X *exactly once*.

Variants of TSP

1. *General* TSP *without* repetitions (General TSP-NR)
   - **Input:** X and symmetric distance function $d : X \times X \to \mathbb{R}_{\geq 0}$
   - **Output:** find a cycle of shortest length that reaches each point of X *exactly once*.
2. *General* TSP *with* repetitions (General TSP-R)

Variants of TSP

1. *General* TSP *without* repetitions (General TSP-NR)
   - **Input:** X and symmetric distance function $d : X \times X \to \mathbb{R}_{\geq 0}$
   - **Output:** find a cycle of shortest length that reaches each point of X *exactly once*.
2. *General* TSP *with* repetitions (General TSP-R)
   - **Input:** X and a symmetric distance function $d : X \times X \to \mathbb{R}_{\geq 0}$
   - **Output:** cycle that reaches all points in X of shortest length. Cycles may now have a point more than once.

Variants of TSP

1. *General* TSP *without* repetitions (General TSP-NR)
   - **Input:** X and symmetric distance function $d : X \times X \to \mathbb{R}_{\geq 0}$
   - **Output:** find a cycle of shortest length that reaches each point of X *exactly once*.
2. *General* TSP *with* repetitions (General TSP-R)
   - **Input:** X and a symmetric distance function $d : X \times X \to \mathbb{R}_{\geq 0}$
   - **Output:** cycle that reaches all points in X of shortest length. Cycles may now have a point more than once.
3. *Metric* TSP *without* repetitions (Metric TSP-NR)

Variants of TSP

1. *General* TSP *without* repetitions (General TSP-NR)
   - **Input:** X and symmetric distance function $d : X \times X \to \mathbb{R}_{\geq 0}$
   - **Output:** find a cycle of shortest length that reaches each point of X *exactly once*.
2. *General* TSP *with* repetitions (General TSP-R)
   - **Input:** X and a symmetric distance function $d : X \times X \to \mathbb{R}_{\geq 0}$
   - **Output:** cycle that reaches all points in X of shortest length. Cycles may now have a point more than once.
3. *Metric* TSP *without* repetitions (Metric TSP-NR)
   - **Input:** X and a symmetric distance function $d : X \times X \to \mathbb{R}_{\geq 0}$ which satisfies triangle inequality (thus gives a metric on X)
   - **Output:** cycle of shortest length that reaches each point of X *exactly once*.

$$d(x,y) \leq d(x,z) + d(z,y) \quad \forall z \in X$$

Variants of TSP

1. *General* TSP *without* repetitions (General TSP-NR)
    - **Input:** X and symmetric distance function $d : X \times X \to \mathbb{R}_{\geq 0}$
    - **Output:** find a cycle of shortest length that reaches each point of X *exactly once*.
2. *General* TSP *with* repetitions (General TSP-R)
    - **Input:** X and a symmetric distance function $d : X \times X \to \mathbb{R}_{\geq 0}$
    - **Output:** cycle that reaches all points in X of shortest length. Cycles may now have a point more than once.
3. *Metric* TSP *without* repetitions (Metric TSP-NR)
    - **Input:** X and a symmetric distance function $d : X \times X \to \mathbb{R}_{\geq 0}$ which satisfies triangle inequality (thus gives a metric on X)
    - **Output:** cycle of shortest length that reaches each point of X *exactly once*.
4. *Metric* TSP *with* repetitions (Metric TSP-R)

Variants of TSP

1. *General* TSP *without* repetitions (General TSP-NR)
   - **Input:** X and symmetric distance function $d : X \times X \to \mathbb{R}_{\geq 0}$
   - **Output:** find a cycle of shortest length that reaches each point of X *exactly once*.
2. *General* TSP *with* repetitions (General TSP-R)
   - **Input:** X and a symmetric distance function $d : X \times X \to \mathbb{R}_{\geq 0}$
   - **Output:** cycle that reaches all points in X of shortest length. Cycles may now have a point more than once.
3. *Metric* TSP *without* repetitions (Metric TSP-NR)
   - **Input:** X and a symmetric distance function $d : X \times X \to \mathbb{R}_{\geq 0}$ which satisfies triangle inequality (thus gives a metric on X)
   - **Output:** cycle of shortest length that reaches each point of X *exactly once*.
4. *Metric* TSP *with* repetitions (Metric TSP-R)
   - **Input:** X and symmetric distance function $d : X \times X \to \mathbb{R}_{\geq 0}$ giving metric (satisfies Δ- inequality)
   - **Output:** cycle that reaches all points in X of shortest length. Cycles may now have a point more than once.

Facts about variants

1. *General* TSP *without* repetitions (General TSP-NR)

Facts about variants

1. *General* TSP *without* repetitions (General TSP-NR)
   - if $P \neq NP$ then there is no poly-time constant-approximation algorithm for General TSP-NR.

Facts about variants

1. *General* TSP *without* repetitions (General TSP-NR)
   - if $P \neq NP$ then there is no poly-time constant-approximation algorithm for General TSP-NR.
   - More generally, if there is any function $r : \mathbb{N} \rightarrow \mathbb{N}$ such that $r(n)$ computable in polynomial time, then it is hard to $r(n)$-approximate General TSP-NR if we assume that $P \neq NP$

$r(n) = 2^n$

$r(n) = n!$

$r(n) = 2^{2^n}$ (repeated squaring)

no hopes of obtaining any reasonable approximation to this problem

Facts about variants

1. *General* TSP *without* repetitions (General TSP-NR)
   - if $P \neq NP$ then there is no poly-time constant-approximation algorithm for General TSP-NR.
   - More generally, if there is any function $r : \mathbb{N} \to \mathbb{N}$ such that $r(n)$ computable in polynomial time, then it is hard to $r(n)$-approximate General TSP-NR if we assume that $P \neq NP$
2. Other three versions are *equivalent* from the point of view of approximation algorithms!

Facts about variants

1. *General* TSP *without* repetitions (General TSP-NR)
   - if $P \neq NP$ then there is no poly-time constant-approximation algorithm for General TSP-NR.
   - More generally, if there is any function $r : \mathbb{N} \to \mathbb{N}$ such that $r(n)$ computable in polynomial time, then it is hard to $r(n)$-approximate General TSP-NR if we assume that $P \neq NP$
2. Other three versions are *equivalent* from the point of view of approximation algorithms!

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for Metric TSP-R.

Facts about variants

1. *General* TSP *without* repetitions (General TSP-NR)
    - if $P \neq NP$ then there is no poly-time constant-approximation algorithm for General TSP-NR.
    - More generally, if there is any function $r : \mathbb{N} \to \mathbb{N}$ such that $r(n)$ computable in polynomial time, then it is hard to $r(n)$-approximate General TSP-NR if we assume that $P \neq NP$
2. Other three versions are *equivalent* from the point of view of approximation algorithms!

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for Metric TSP-R.

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for General TSP-R.

Metric TSP-NR equivalent to Metric TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for Metric TSP-R. In particular:

Metric TSP-NR equivalent to Metric TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for Metric TSP-R. In particular:

1. *If (X, d) is an input to Metric TSP, the cost of the optimum is the same whether or not we allow repetitions.*

Metric TSP-NR equivalent to Metric TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for Metric TSP-R. In particular:

1. *If (X, d) is an input to Metric TSP, the cost of the optimum is the same whether or not we allow repetitions.*
2. *Every c-approximation algorithm for Metric TSP-NR is also a c-approximation algorithm for Metric TSP-R.*

m TSP-NR ⇒ m TSP-R

① cost of OPT is same

Metric TSP-NR equivalent to Metric TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for Metric TSP-R. In particular:

1. *If (X, d) is an input to Metric TSP, the cost of the optimum is the same whether or not we allow repetitions.*
2. *Every c-approximation algorithm for Metric TSP-NR is also a c-approximation algorithm for Metric TSP-R.*
3. *Every c-approximation algorithm for Metric TSP-R can be turned into a c-approximate algorithm for Metric TSP-NR, after adding a linear time post-processing.*

Metric TSP-NR equivalent to Metric TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for Metric TSP-R. In particular:

1. *If (X, d) is an input to Metric TSP, the cost of the optimum is the same whether or not we allow repetitions.*
2. *Every c-approximation algorithm for Metric TSP-NR is also a c-approximation algorithm for Metric TSP-R.*
3. *Every c-approximation algorithm for Metric TSP-R can be turned into a c-approximate algorithm for Metric TSP-NR, after adding a linear time post-processing.*

- $OPT_R(X, d)$ be cost of optimal solution for (X, d) in Metric TSP-R

Metric TSP-NR equivalent to Metric TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for Metric TSP-R. In particular:

1. *If (X, d) is an input to Metric TSP, the cost of the optimum is the same whether or not we allow repetitions.*
2. *Every c-approximation algorithm for Metric TSP-NR is also a c-approximation algorithm for Metric TSP-R.*
3. *Every c-approximation algorithm for Metric TSP-R can be turned into a c-approximate algorithm for Metric TSP-NR, after adding a linear time post-processing.*

- $OPT_R(X, d)$ be cost of optimal solution for (X, d) in Metric TSP-R
- $OPT_{NR}(X, d)$ be the cost of optimal solution for (X, d) in Metric TSP-NR.

Metric TSP-NR equivalent to Metric TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for Metric TSP-R. In particular:

1. *If (X, d) is an input to Metric TSP, the cost of the optimum is the same whether or not we allow repetitions.*

Metric TSP-NR equivalent to Metric TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for Metric TSP-R. In particular:

1. *If (X, d) is an input to Metric TSP, the cost of the optimum is the same whether or not we allow repetitions.*

- Solution space of Metric TSP-R is larger than solution space of Metric TSP-NR. Thus

$$OPT_R(X, d) \leq OPT_{NR}(X, d)$$

any solution to mTSP-NR is also a solution to mTSP-R

Metric TSP-NR equivalent to Metric TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for Metric TSP-R. In particular:

1. *If (X, d) is an input to Metric TSP, the cost of the optimum is the same whether or not we allow repetitions.*

removed a ⇒ cost(C′) ≤ cost(C) ⇒ C′ also OPT.

cycle

- Let $\mathcal{C} = p_0 \to p_1 \to p_2 \to \cdots \to p_m = p_0$ be a solution to $OPT_R(X, d)$. Now, create a cycle $\mathcal{C}'$ from C simply by removing the repetitions

$$a \to b \to \cdots c \to a \to d \to \cdots$$

becomes

$$a \to b \to \cdots c \to d \to \cdots$$

by repeating this process end up with C* without rep.

Metric TSP-NR equivalent to Metric TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for Metric TSP-R. In particular:

(2) *Every c-approximation algorithm for Metric TSP-NR is also a c-approximation algorithm for Metric TSP-R.*

Metric TSP-NR equivalent to Metric TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for Metric TSP-R. In particular:

(2) *Every c-approximation algorithm for Metric TSP-NR is also a c-approximation algorithm for Metric TSP-R.*

- If we have a c-approximation algorithm for Metric TSP-NR, then we know that our solution (cycle $\mathcal{C}$) satisfies:

$$cost(C) \leq c \cdot OPT_{NR}(X, d)$$

$$= OPT_R(X, d)$$

Metric TSP-NR equivalent to Metric TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for Metric TSP-R. In particular:

(2) *Every c-approximation algorithm for Metric TSP-NR is also a c-approximation algorithm for Metric TSP-R.*

- If we have a c-approximation algorithm for Metric TSP-NR, then we know that our solution (cycle $\mathcal{C}$) satisfies:

$$cost(C) \leq c \cdot OPT_{NR}(X, d)$$

- Since $OPT_{NR}(X, d) = OPT_R(X, d)$ and $\mathcal{C}$ is also a solution to Metric TSP-R, we are done.

Metric TSP-NR equivalent to Metric TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for Metric TSP-R. In particular:

(3) *Every c-approximation algorithm for Metric TSP-R can be turned into a c-approximate algorithm for Metric TSP-NR, after adding a linear time post-processing.*

Metric TSP-NR equivalent to Metric TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-NR if, and only if, there is a polynomial time c-approximation for Metric TSP-R. In particular:

(3) *Every c-approximation algorithm for Metric TSP-R can be turned into a c-approximate algorithm for Metric TSP-NR, after adding a linear time post-processing.*

- Given any solution to Metric TSP-R, simply run the procedure that removes repeated visits to a vertex. This only decreases cost by metric property.

Metric TSP-R equivalent to General TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-R if, and only if, there is a polynomial time c-approximation for General TSP-R. In particular:

1. *Every c-approximation algorithm for General TSP-R is also a c-approximation algorithm for Metric TSP-R.*
2. *Every c-approximation algorithm for Metric TSP-R can be turned into a c-approximate algorithm for General TSP-R, after adding a polynomial time pre and post-processing.*

Metric TSP-R equivalent to General TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-R if, and only if, there is a polynomial time c-approximation for General TSP-R. In particular:

1. *Every c-approximation algorithm for General TSP-R is also a c-approximation algorithm for Metric TSP-R.*
2. *Every c-approximation algorithm for Metric TSP-R can be turned into a c-approximate algorithm for General TSP-R, after adding a polynomial time pre and post-processing.*

- First item follows by the fact that Metric TSP-R is a special case of General TSP-R, when the distance function satisfies the triangle inequality.

Metric TSP-R equivalent to General TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-R if, and only if, there is a polynomial time c-approximation for General TSP-R. In particular:

(2) *Every c-approximation algorithm for Metric TSP-R can be turned into a c-approximate algorithm for General TSP-R, after adding a polynomial time pre and post-processing.*

Metric TSP-R equivalent to General TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-R if, and only if, there is a polynomial time c-approximation for General TSP-R. In particular:

(2) *Every c-approximation algorithm for Metric TSP-R can be turned into a c-approximate algorithm for General TSP-R, after adding a polynomial time pre and post-processing.*

- On input (X, d) to General TSP-R, let $G(X, E, w)$ be the complete weighted graph such that $w(x, y) = d(x, y)$. Now compute new distance $\delta : X \to \mathbb{R}_{\geq 0}$ such that

$$\delta(x, y) \leftarrow \text{length of shortest path from } x \text{ to } y \text{ in G}$$

Metric TSP-R equivalent to General TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-R if, and only if, there is a polynomial time c-approximation for General TSP-R. In particular:

(2) *Every c-approximation algorithm for Metric TSP-R can be turned into a c-approximate algorithm for General TSP-R, after adding a polynomial time pre and post-processing.*

- On input (X, d) to General TSP-R, let $G(X, E, w)$ be the complete weighted graph such that $w(x, y) = d(x, y)$. Now compute new distance $\delta : X \to \mathbb{R}_{\geq 0}$ such that

$$\delta(x, y) \leftarrow \text{ length of shortest path from } x \text{ to } y \text{ in G}$$

- Note that δ satisfies triangle inequality!

$$\delta(x,y) \leq \delta(x,z) + \delta(z,y) \quad \forall z$$

↑ equality iff. z is in a shortest path from x to y.

Metric TSP-R equivalent to General TSP-R

Lemma

For every $c \geq 1$ there is a polynomial time c-approximation for Metric TSP-R if, and only if, there is a polynomial time c-approximation for General TSP-R. In particular:

(2) *Every c-approximation algorithm for Metric TSP-R can be turned into a c-approximate algorithm for General TSP-R, after adding a polynomial time pre and post-processing.*

- On input (X, d) to General TSP-R, let $G(X, E, w)$ be the complete weighted graph such that $w(x, y) = d(x, y)$. Now compute new distance $\delta : X \to \mathbb{R}_{\geq 0}$ such that

$$\delta(x, y) \leftarrow \text{ length of shortest path from } x \text{ to } y \text{ in G}$$

- Note that δ satisfies triangle inequality!
- Give input (X, δ) to our algorithm for Metric TSP-R. Let $\mathcal{C}$ be the cycle it outputs. Thus

proper input to metric TSP

from our approx. algorithm

$$cost_R(\mathcal{C}) \leq c \cdot OPT_R(X, \delta)$$

Metric TSP-R equivalent to General TSP-R

- Give input (X, δ) to our algorithm for Metric TSP-R. Let $\mathcal{C}$ be the cycle it outputs. Thus

$$cost_R(\mathcal{C}) \leq c \cdot opt_R(X, \delta) \leq c \cdot opt_{GR}(X, \delta)$$

previous slide

(X, δ) is a metric TSP

$\Downarrow$

$OPT_R(X, \delta) = OPT_{GR}(X, \delta)$

$\mathcal{C}$

Metric TSP-R equivalent to General TSP-R

- Give input (X, δ) to our algorithm for Metric TSP-R. Let $\mathcal{C}$ be the cycle it outputs. Thus

$$cost_R(\mathcal{C}) \leq c \cdot opt_R(X, \delta) \leq c \cdot opt_{GR}(X, \delta)$$

- For every pair $(x, y) \in X^2$, note that $\delta(x, y) \leq d(x, y)$, so

$$OPT_R(X, \delta) \leq OPT_{GR}(X, d)$$

$\delta(x, y)$ = length shortest path from x to y $\leq d(x, y)$ one path from x, y

$\mathcal{C}$ cycle in X $\quad \mathcal{C} = x_0 \to x_1 \to \cdots \to x_m \to x_{m+1} = x_0$

$$cost_R(\mathcal{C}) = \sum_{i=0}^{m} \delta(x_i, x_{i+1}) \leq \sum_{i=0}^{m} d(x_i, x_{i+1}) = cost_{GR}(\mathcal{C})$$

Metric TSP-R equivalent to General TSP-R

- Give input (X, δ) to our algorithm for Metric TSP-R. Let $\mathcal{C}$ be the cycle it outputs. Thus

$$cost_R(\mathcal{C}) \leq c \cdot opt_R(X, \delta) \leq c \cdot opt_{GR}(X, \delta)$$

- For every pair $(x, y) \in X^2$, note that $\delta(x, y) \leq d(x, y)$, so

$$OPT_R(X, \delta) \leq OPT_{GR}(X, d)$$

- Let Γ be the cycle obtained from $\mathcal{C}$ by simply replacing every $x \to y$ by the shortest path $x \to p_1 \to \cdots \to p_t \to y$ in G.

want cycle Γ s.t. $cost_d(\Gamma) = cost_\delta(\mathcal{C})$

$$cost_d(\Gamma) = cost_\delta(\mathcal{C}) \leq c \cdot opt_{GR}(X, d)$$

Metric TSP-R equivalent to General TSP-R

- Give input (X, δ) to our algorithm for Metric TSP-R. Let $\mathcal{C}$ be the cycle it outputs. Thus

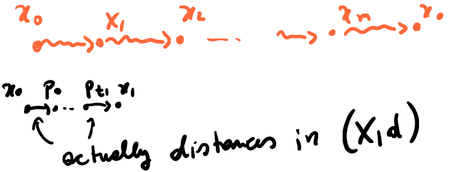
$$cost_R(\mathcal{C}) \leq c \cdot opt_R(X, \delta) \leq c \cdot opt_{GR}(X, \delta)$$

- For every pair $(x, y) \in X^2$, note that $\delta(x, y) \leq d(x, y)$, so

$$OPT_R(X, \delta) \leq OPT_{GR}(X, d)$$

- Let Γ be the cycle obtained from $\mathcal{C}$ by simply replacing every $x \rightarrow y$ by the shortest path $x \rightarrow p_1 \rightarrow \cdots \rightarrow p_t \rightarrow y$ in G.
  1. Note that

$$cost(\mathcal{C}, \delta) = cost(\Gamma, d)$$



Metric TSP-R equivalent to General TSP-R

- Give input (X, δ) to our algorithm for Metric TSP-R. Let $\mathcal{C}$ be the cycle it outputs. Thus

$$cost_R(\mathcal{C}) \leq c \cdot opt_R(X, \delta) \leq c \cdot opt_{GR}(X, \delta)$$

- For every pair $(x, y) \in X^2$, note that $\delta(x, y) \leq d(x, y)$, so

$$OPT_R(X, \delta) \leq OPT_{GR}(X, d)$$

- Let Γ be the cycle obtained from $\mathcal{C}$ by simply replacing every $x \to y$ by the shortest path $x \to p_1 \to \cdots \to p_t \to y$ in G.
  1. Note that

$$cost(\mathcal{C}, \delta) = cost(\Gamma, d)$$

- Combining the inequalities so far, we get:

$$cost(\Gamma, d) = cost(\mathcal{C}, \delta) \leq c \cdot opt_R(X, \delta) \leq c \cdot opt_{GR}(X, d)$$

- Equivalent Versions of Traveling Salesman Problem

- Approximation Algorithms for Traveling Salesman Problem

- Conclusion

- Acknowledgements

A 2-approximation algorithm

The following lemma gives us a way to get a 2-approximation algorithm:

Lemma

Let $T(X, E, d)$ be a weighted tree with vertices X and weights given by the distance function $d : X \times X \to \mathbb{R}_{\geq 0}$. There is a cycle $\mathcal{C}$ that reaches each vertex at least once, and such that

$$cost(\mathcal{C}, d) = 2 \cdot cost(T, d).$$

A 2-approximation algorithm

The following lemma gives us a way to get a 2-approximation algorithm:

Lemma

Let $T(X, E, d)$ be a weighted tree with vertices X and weights given by the distance function $d : X \times X \to \mathbb{R}_{\geq 0}$. There is a cycle $\mathcal{C}$ that reaches each vertex at least once, and such that

$$cost(\mathcal{C}, d) = 2 \cdot cost(T, d).$$

- Consider a DFS visit of the tree.

A 2-approximation algorithm

The following lemma gives us a way to get a 2-approximation algorithm:

Lemma

Let $T(X, E, d)$ be a weighted tree with vertices X and weights given by the distance function $d : X \times X \to \mathbb{R}_{\geq 0}$. There is a cycle $\mathcal{C}$ that reaches each vertex at least once, and such that

$$cost(\mathcal{C}, d) = 2 \cdot cost(T, d).$$

- Consider a DFS visit of the tree.
- Each edge traversed exactly twice

A 2-approximation algorithm

The following lemma gives us a way to get a 2-approximation algorithm:

Lemma

Let $T(X, E, d)$ be a weighted tree with vertices X and weights given by the distance function $d : X \times X \to \mathbb{R}_{\geq 0}$. There is a cycle $\mathcal{C}$ that reaches each vertex at least once, and such that

$$cost(\mathcal{C}, d) = 2 \cdot cost(T, d).$$

- Consider a DFS visit of the tree.
- Each edge traversed exactly twice

Theorem

There is a polynomial-time 2-approximation algorithm for General TSP-R.

A 2-approximation algorithm

The following lemma gives us a way to get a 2-approximation algorithm:

Lemma

Let $T(X, E, d)$ be a weighted tree with vertices X and weights given by the distance function $d : X \times X \to \mathbb{R}_{\geq 0}$. There is a cycle $\mathcal{C}$ that reaches each vertex at least once, and such that

$$cost(\mathcal{C}, d) = 2 \cdot cost(T, d).$$

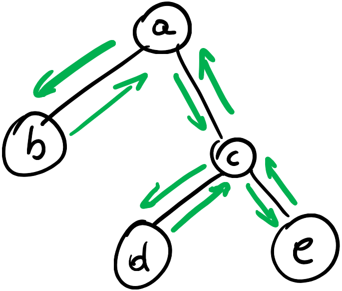
- Consider a DFS visit of the tree.
- Each edge traversed exactly twice

Theorem

There is a polynomial-time 2-approximation algorithm for General TSP-R.

Idea: find a minimum spanning tree on the complete weighted graph $G(X, K_X, d)$.

Example



DFS :

$a \to b \to a \to c \to d$

$\to c \to e \to c \to a$

Proof of Theorem

Theorem

There is a polynomial-time 2-approximation algorithm for General TSP-R.

Proof of Theorem

Theorem

There is a polynomial-time 2-approximation algorithm for General TSP-R.

1. On input (X, d), find minimum spanning tree $T(X, K_X, d)$.

Proof of Theorem

Theorem

There is a polynomial-time 2-approximation algorithm for General TSP-R.

1. On input (X, d), find minimum spanning tree $T(X, K_X, d)$.
2. By our lemma, there is a cycle from T with cost $2 \cdot cost(T, d)$.

Proof of Theorem

Theorem

There is a polynomial-time 2-approximation algorithm for General TSP-R.

1. On input (X, d), find minimum spanning tree $T(X, K_X, d)$.
2. By our lemma, there is a cycle from T with cost $2 \cdot cost(T, d)$.
3. Need to show that this is a 2-approximation.

Proof of Theorem

Theorem

There is a polynomial-time 2-approximation algorithm for General TSP-R.

1. On input (X, d), find minimum spanning tree $T(X, K_X, d)$.
2. By our lemma, there is a cycle from T with cost $2 \cdot cost(T, d)$.
3. Need to show that this is a 2-approximation.
   - To do that, enough to show that $OPT_{GR}(X, d) \geq cost(T, d)$

cost of minimum spanning tree (easy to get)

proxy

is a lower bound on optimum solution

idea: find a proxy of OPT which is easy to construct
then construct a valid solution from it.

Proof of Theorem

Theorem

There is a polynomial-time 2-approximation algorithm for General TSP-R.

1. On input (X, d), find minimum spanning tree $T(X, K_X, d)$.
2. By our lemma, there is a cycle from T with cost $2 \cdot cost(T, d)$.
3. Need to show that this is a 2-approximation.
   - To do that, enough to show that $OPT_{GR}(X, d) \geq cost(T, d)$
   - If $\mathcal{C}$ is optimum cycle for (X, d), that is, $cost(\mathcal{C}, d) = OPT_{GR}(X, d)$, take all edges which are used in $\mathcal{C}$. Call this set F.

Proof of Theorem

Theorem

There is a polynomial-time 2-approximation algorithm for General TSP-R.

1. On input (X, d), find minimum spanning tree $T(X, K_X, d)$.
2. By our lemma, there is a cycle from T with cost $2 \cdot cost(T, d)$.
3. Need to show that this is a 2-approximation.
   - To do that, enough to show that $OPT_{GR}(X, d) \geq cost(T, d)$
   - If $\mathcal{C}$ is optimum cycle for (X, d), that is, $cost(\mathcal{C}, d) = OPT_{GR}(X, d)$, take all edges which are used in $\mathcal{C}$. Call this set F.
   - Note that the weighted graph $H(X, F, d)$ is connected. Let T' be a spanning tree of this graph.

$$cost(T', d) \leq cost(\mathcal{C}, d) = OPT_{GR}(X, d)$$

because spanning tree uses a subset of edges.

Proof of Theorem

Theorem

There is a polynomial-time 2-approximation algorithm for General TSP-R.

1. On input (X, d), find minimum spanning tree $T(X, K_X, d)$.
2. By our lemma, there is a cycle from T with cost $2 \cdot cost(T, d)$.
3. Need to show that this is a 2-approximation.
   - To do that, enough to show that $OPT_{GR}(X, d) \geq cost(T, d)$
   - If $\mathcal{C}$ is optimum cycle for (X, d), that is, $cost(\mathcal{C}, d) = OPT_{GR}(X, d)$, take all edges which are used in $\mathcal{C}$. Call this set F.
   - Note that the weighted graph $H(X, F, d)$ is connected. Let T' be a spanning tree of this graph.

$$cost(T', d) \leq cost(\mathcal{C}, d) = OPT_{GR}(X, d)$$

   - Since T' is a spanning tree of X, we have that

$G(X, K_X, d)$

$$cost(T, d) \leq cost(T', d)$$

↑ by minimality of T.

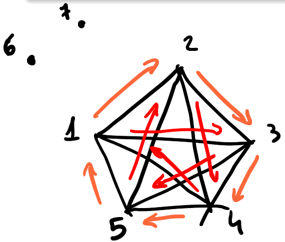
and we are done.

Eulerian Tours

Definition (Eulerian Cycle)

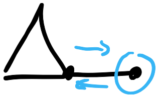
An Eulerian cycle in a multigraph $G(V, E)$ is a cycle $p_0 \to p_1 \to \cdots \to p_m = p_0$ such that the number of edges $\{u, v\} \in E$ is equal to the number of times $\{u, v\}$ is used in the cycle.

In other words, each edge is used *exactly once*.



has Eulerian cycle

1 → 2 → 3 → 4 → 5 → 1 → 3 → 5 → 2 → 4 → 1



does not have Eulerian cycle

Eulerian Tours

Definition (Eulerian Cycle)

An Eulerian cycle in a multigraph $G(V, E)$ is a cycle $p_0 \to p_1 \to \cdots \to p_m = p_0$ such that the number of edges $\{u, v\} \in E$ is equal to the number of times $\{u, v\}$ is used in the cycle.

In other words, each edge is used *exactly once*.

Theorem (Eulerian Cycle Existence and Algorithm)

A multi-graph $G(V, E)$ has an Eulerian cycle if, and only if, every vertex has even degree and the vertices of positive degree are connected.

Moreover, there is a polynomial time algorithm that, on input a connected graph $G(V, E)$ in which every vertex has even degree, outputs an Eulerian cycle.

Proof of Theorem I

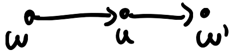
($\Rightarrow$)

$G(V,E)$ has Eulerian cycle $\Rightarrow$ vertices of >0 deg. connected

(trivial) ← formalize it by contrapositive

$u \in V$ need to prove that $\deg(u)$ even

take eulerian cycle $\mathcal{C}$ for each time vertex u appears



pair the edges

$\{w, u\}$ $\{u, w'\}$

all these edges are distinct (by eulerian property)

$\Rightarrow \deg(u)$ even.

Proof of Theorem II

(removing isolated vertices)

($\Leftarrow$) Induction on # edges in graph:

If $G(V,E)$ connected and all vertices have even degree, then G has a cycle.

If every vertex has degree $= 2$, then G must be a cycle (because G is connected) in this case we are done.

Otherwise take cycle without repetitions starting from vertex of degree ≥ 4 (such cycle must exist as G is connected). Removing this cycle and vertices of degree 0 we get smaller connected graph with even degrees. Induction $\Rightarrow$ we get Eulerian cycle.

Procedure gives poly-time algorithm! How to find small cycle? DFS!

Better Approximation Algorithm

- In our previous TSP algorithm, we computed a minimum spanning tree and took our cycle to be a 2-pass over the tree.

Better Approximation Algorithm

- In our previous TSP algorithm, we computed a minimum spanning tree and took our cycle to be a 2-pass over the tree.
- In Eulerian cycle words: we doubled the edges to make sure each vertex in our "double tree" had even degree, then did an Eulerian cycle.

Better Approximation Algorithm

- In our previous TSP algorithm, we computed a minimum spanning tree and took our cycle to be a 2-pass over the tree.
- In Eulerian cycle words: we doubled the edges to make sure each vertex in our "double tree" had even degree, then did an Eulerian cycle.
- This is a bit wasteful.

Better Approximation Algorithm

- In our previous TSP algorithm, we computed a minimum spanning tree and took our cycle to be a 2-pass over the tree.
- In Eulerian cycle words: we doubled the edges to make sure each vertex in our "double tree" had even degree, then did an Eulerian cycle.
- This is a bit wasteful.
  - Doubling every edge works, but what if a node has degree 1001?

Better Approximation Algorithm

- In our previous TSP algorithm, we computed a minimum spanning tree and took our cycle to be a 2-pass over the tree.
- In Eulerian cycle words: we doubled the edges to make sure each vertex in our "double tree" had even degree, then did an Eulerian cycle.
- This is a bit wasteful.
  - Doubling every edge works, but what if a node has degree 1001?
  - Could we just add 1 extra edge, instead of 1001?

Better Approximation Algorithm

- In our previous TSP algorithm, we computed a minimum spanning tree and took our cycle to be a 2-pass over the tree.
- In Eulerian cycle words: we doubled the edges to make sure each vertex in our "double tree" had even degree, then did an Eulerian cycle.
- This is a bit wasteful.
  - Doubling every edge works, but what if a node has degree 1001?
  - Could we just add 1 extra edge, instead of 1001?
- **Idea:** take vertices of odd degree in the tree (there must be an even number of these). Let this set be $O \subseteq X$

$$\underbrace{2|E|}_{\text{even}} = \sum_{v \in V} \deg(v) = \sum_{v \in O} \underbrace{\deg(v)}_{\text{odd}} + \underbrace{\sum_{u \in X \setminus O} \deg(u)}_{\text{even}}$$

$$\Downarrow$$

$$|O| = \text{even}$$

Better Approximation Algorithm

- In our previous TSP algorithm, we computed a minimum spanning tree and took our cycle to be a 2-pass over the tree.
- In Eulerian cycle words: we doubled the edges to make sure each vertex in our "double tree" had even degree, then did an Eulerian cycle.
- This is a bit wasteful.
  - Doubling every edge works, but what if a node has degree 1001?
  - Could we just add 1 extra edge, instead of 1001?
- **Idea:** take vertices of odd degree in the tree (there must be an even number of these). Let this set be $O \subseteq X$
- Find a *minimum cost perfect matching* (in the weighted graph (O, d))!

T + M ← take Eulerian tour

min spanning tree

min matching of O

Better Approximation Algorithm

- In our previous TSP algorithm, we computed a minimum spanning tree and took our cycle to be a 2-pass over the tree.
- In Eulerian cycle words: we doubled the edges to make sure each vertex in our "double tree" had even degree, then did an Eulerian cycle.
- This is a bit wasteful.
  - Doubling every edge works, but what if a node has degree 1001?
  - Could we just add 1 extra edge, instead of 1001?
- **Idea:** take vertices of odd degree in the tree (there must be an even number of these). Let this set be $O \subseteq X$
- Find a *minimum cost perfect matching* (in the weighted graph (O, d))!
- Why would that improve our previous algorithm?

Better Approximation Algorithm

- In our previous TSP algorithm, we computed a minimum spanning tree and took our cycle to be a 2-pass over the tree.
- In Eulerian cycle words: we doubled the edges to make sure each vertex in our "double tree" had even degree, then did an Eulerian cycle.
- This is a bit wasteful.
  - Doubling every edge works, but what if a node has degree 1001?
  - Could we just add 1 extra edge, instead of 1001?
- **Idea:** take vertices of odd degree in the tree (there must be an even number of these). Let this set be $O \subseteq X$
- Find a *minimum cost perfect matching* (in the weighted graph (O, d))!
- Why would that improve our previous algorithm?
  - Min-cost matching will have *half* the total cost of optimum TSP cycle!

Better Approximation Algorithm

- In our previous TSP algorithm, we computed a minimum spanning tree and took our cycle to be a 2-pass over the tree.
- In Eulerian cycle words: we doubled the edges to make sure each vertex in our "double tree" had even degree, then did an Eulerian cycle.
- This is a bit wasteful.
  - Doubling every edge works, but what if a node has degree 1001?
  - Could we just add 1 extra edge, instead of 1001?
- **Idea:** take vertices of odd degree in the tree (there must be an even number of these). Let this set be $O \subseteq X$
- Find a *minimum cost perfect matching* (in the weighted graph (O, d))!
- Why would that improve our previous algorithm?
  - Min-cost matching will have *half* the total cost of optimum TSP cycle!
  - Thus we get a 3/2-approximation!

Putting Everything Together

1. **Input:** (X, d) instance of *Metric TSP-R*
2. **Output:** Cycle $\mathcal{C}$ over X covering every vertex at least once, with

$$cost(\mathcal{C}, d) \leq 3/2 \cdot OPT_{GR}(X, d)$$

Putting Everything Together

1. **Input:** (X, d) instance of *Metric TSP-R*
2. **Output:** Cycle $\mathcal{C}$ over X covering every vertex at least once, with

$$cost(\mathcal{C}, d) \leq 3/2 \cdot OPT_{GR}(X, d)$$

3. Find minimum cost spanning tree T in (X, K_X, d)

Putting Everything Together

1. **Input:** (X, d) instance of *Metric TSP-R*
2. **Output:** Cycle $\mathcal{C}$ over X covering every vertex at least once, with

$$cost(\mathcal{C}, d) \leq 3/2 \cdot OPT_{GR}(X, d)$$

3. Find minimum cost spanning tree T in (X, K_X, d)
4. Let O be the set of vertices of odd degree in T

Putting Everything Together

1. **Input:** (X, d) instance of *Metric TSP-R*
2. **Output:** Cycle $\mathcal{C}$ over X covering every vertex at least once, with

$$cost(\mathcal{C}, d) \leq 3/2 \cdot OPT_{GR}(X, d)$$

3. Find minimum cost spanning tree T in (X, K_X, d)
4. Let O be the set of vertices of odd degree in T
5. Find minimum cost perfect matching $\mathcal{M}$ in (O, K_O, d)

Putting Everything Together

1. **Input:** (X, d) instance of *Metric TSP-R*
2. **Output:** Cycle $\mathcal{C}$ over X covering every vertex at least once, with

$$cost(\mathcal{C}, d) \leq 3/2 \cdot OPT_{GR}(X, d)$$

3. Find minimum cost spanning tree T in (X, K_X, d)
4. Let O be the set of vertices of odd degree in T
5. Find minimum cost perfect matching $\mathcal{M}$ in (O, K_O, d)
6. Let E be the set of edges of T together with the set of edges of $\mathcal{M}$

$$E = T + M$$

Putting Everything Together

1. **Input:** (X, d) instance of *Metric TSP-R*
2. **Output:** Cycle $\mathcal{C}$ over X covering every vertex at least once, with

$$cost(\mathcal{C}, d) \leq 3/2 \cdot OPT_{GR}(X, d)$$

3. Find minimum cost spanning tree T in (X, K_X, d)
4. Let O be the set of vertices of odd degree in T
5. Find minimum cost perfect matching $\mathcal{M}$ in (O, K_O, d)
6. Let E be the set of edges of T together with the set of edges of $\mathcal{M}$
7. Find Eulerian Cycle $\mathcal{C}$ on E
8. Output $\mathcal{C}$

Analysis

- Note that
$$cost(\mathcal{C}, d) = \underline{cost(T, d)} + \underline{cost(\mathcal{M}, d)}$$
Since we have Eulerian cycle.

Analysis

- Note that
$$cost(\mathcal{C}, d) = cost(T, d) + cost(\mathcal{M}, d)$$
  Since we have Eulerian cycle.
- We already showed that $cost(T, d) \leq OPT_R(X, d)$

Analysis

- Note that
$$cost(\mathcal{C}, d) = cost(T, d) + cost(\mathcal{M}, d)$$
Since we have Eulerian cycle.
- We already showed that $cost(T, d) \leq OPT_R(X, d)$
- Need to show that $cost(\mathcal{M}, d) \leq \frac{1}{2} \cdot OPT_R(X, d)$

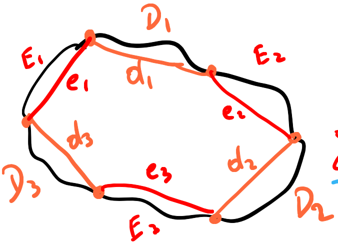
Analysis

- Note that
$$cost(\mathcal{C}, d) = cost(T, d) + cost(\mathcal{M}, d)$$
Since we have Eulerian cycle.
- We already showed that $cost(T, d) \leq OPT_R(X, d)$
- Need to show that $cost(\mathcal{M}, d) \leq \frac{1}{2} \cdot OPT_R(X, d)$
- If Γ is a TSP cycle such that $cost(\Gamma, d) = OPT_R(X, d)$

Analysis

- Note that
$$cost(\mathcal{C}, d) = cost(T, d) + cost(\mathcal{M}, d)$$
  Since we have Eulerian cycle.
- We already showed that $cost(T, d) \leq OPT_R(X, d)$
- Need to show that $cost(\mathcal{M}, d) \leq \frac{1}{2} \cdot OPT_R(X, d)$
- If Γ is a TSP cycle such that $cost(\Gamma, d) = OPT_R(X, d)$
  - Let C be the cycle we obtain from Γ by skipping elements of $X \setminus O$ and removing duplicate vertices from O

orange vertices in O



$d_i \leq D_i$

$e_i \leq E_i$

by Δ-inequality.

$\sum E_i + \sum D_i = cost(\Gamma, d)$

$\sum E_i \geq \sum e_i$, $\sum D_i \geq \sum d_i$

Analysis

- Note that
$$cost(\mathcal{C}, d) = cost(T, d) + cost(\mathcal{M}, d)$$
Since we have Eulerian cycle.
- We already showed that $cost(T, d) \leq OPT_R(X, d)$
- Need to show that $cost(\mathcal{M}, d) \leq \frac{1}{2} \cdot OPT_R(X, d)$
- If Γ is a TSP cycle such that $cost(\Gamma, d) = OPT_R(X, d)$
  - Let C be the cycle we obtain from Γ by skipping elements of $X \setminus O$ and removing duplicate vertices from O
  - Triangle inequality $\Rightarrow$ $cost(C, d) \leq cost(\Gamma, d)$

Analysis

- Note that
$$cost(\mathcal{C}, d) = cost(T, d) + cost(\mathcal{M}, d)$$
Since we have Eulerian cycle.
- We already showed that $cost(T, d) \leq OPT_R(X, d)$
- Need to show that $cost(\mathcal{M}, d) \leq \frac{1}{2} \cdot OPT_R(X, d)$
- If Γ is a TSP cycle such that $cost(\Gamma, d) = OPT_R(X, d)$
  - Let C be the cycle we obtain from Γ by skipping elements of $X \setminus O$ and removing duplicate vertices from O
  - Triangle inequality $\Rightarrow cost(C, d) \leq cost(\Gamma, d)$
  - Cycle C induces two matchings of O. One of them has weight $\leq \frac{1}{2} \cdot cost(C, d)$.

suppose $\sum D_i \leq \frac{1}{2} cost(C, d) = \frac{1}{2} OPT_R$

then Cost of min matching $\leq \sum d_i \leq \sum D_i \leq \frac{1}{2} OPT_R$ (matching)

Analysis

- Note that
$$cost(\mathcal{C}, d) = cost(T, d) + cost(\mathcal{M}, d)$$
Since we have Eulerian cycle.
- We already showed that $cost(T, d) \leq OPT_R(X, d)$
- Need to show that $cost(\mathcal{M}, d) \leq \frac{1}{2} \cdot OPT_R(X, d)$
- If Γ is a TSP cycle such that $cost(\Gamma, d) = OPT_R(X, d)$
  - Let C be the cycle we obtain from Γ by skipping elements of $X \setminus O$ and removing duplicate vertices from O
  - Triangle inequality $\Rightarrow cost(C, d) \leq cost(\Gamma, d)$
  - Cycle C induces two matchings of O. One of them has weight $\leq \frac{1}{2} \cdot cost(C, d)$.
  - Thus:

$$cost(\mathcal{M}, d) \leq \frac{1}{2} \cdot cost(C, d) \leq \frac{1}{2} \cdot cost(\Gamma, d) = \frac{1}{2} \cdot OPT_R(X, d).$$

Δ-inequality

Conclusion

- Traveling Salesman Problem - important, but NP-hard
- Equivalent variants of TSP

Conclusion

- Traveling Salesman Problem - important, but NP-hard
- Equivalent variants of TSP
- Combinatorial Approximation Algorithms for TSP

Conclusion

- Traveling Salesman Problem - important, but NP-hard
- Equivalent variants of TSP
- Combinatorial Approximation Algorithms for TSP
- Achieve approximation algorithm by looking at an object (minimum spanning tree) which is a *lower bound* on the cost of the optimum

Conclusion

- Traveling Salesman Problem - important, but NP-hard
- Equivalent variants of TSP
- Combinatorial Approximation Algorithms for TSP
- Achieve approximation algorithm by looking at an object (minimum spanning tree) which is a *lower bound* on the cost of the optimum
- This object (minimum spanning tree) is also easy to find, so exploit that to our advantage to get approximation algorithm.

Acknowledgement

- Lecture based largely on:
  - Lectures 2-4 of Luca's Optimization class
- See Luca's Lecture 3 notes at https://lucatrevisan.github.io/teaching/cs261-11/lecture03.pdf
- See Luca's Lecture 4 notes at https://lucatrevisan.github.io/teaching/cs261-11/lecture04.pdf