Approximation Algorithms

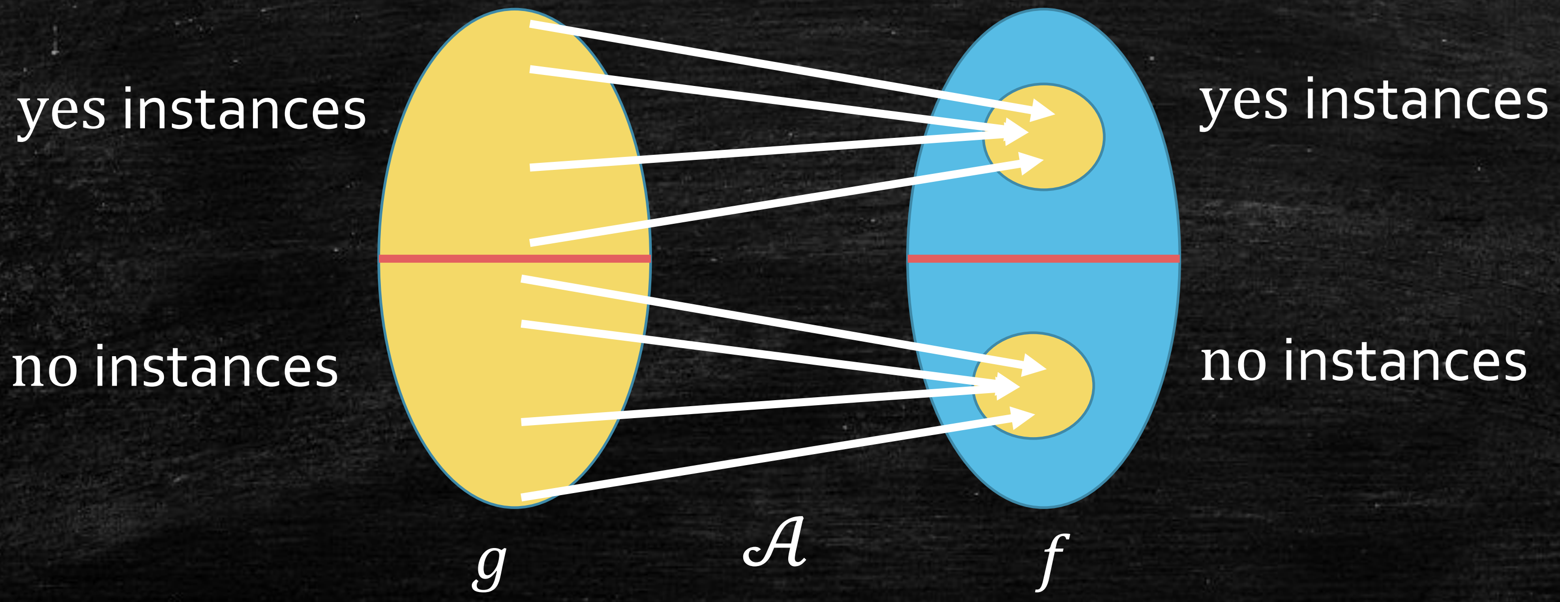
1. One more example of reduction: k-means
2. approximation algorithms

Proving f is NP-complete

- Prove $f \in$ **NP**.
- Find an NP-complete problem g and prove $g \leq_k f$.

Reduction: $\mathcal{A}$ computes $g \leq_k f$

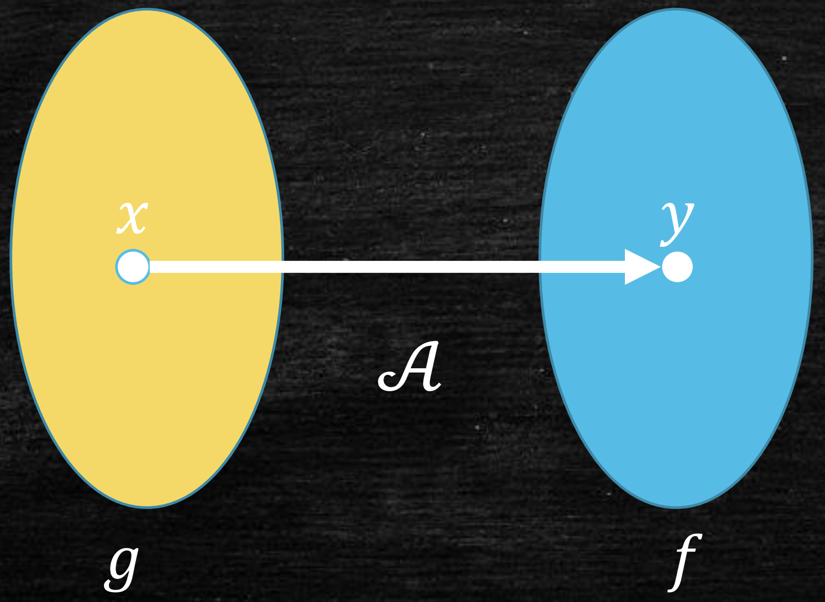
- $x \mapsto y$ under poly-time TM $\mathcal{A}$
- x is yes $\Rightarrow$ y is yes
- x is no $\Rightarrow$ y is no



Reduction: $\mathcal{A}$ computes $g \leq_k f$

- $x \mapsto y$ under poly-time TM $\mathcal{A}$
- x is yes $\Rightarrow$ y is yes
- x is no $\Rightarrow$ y is no

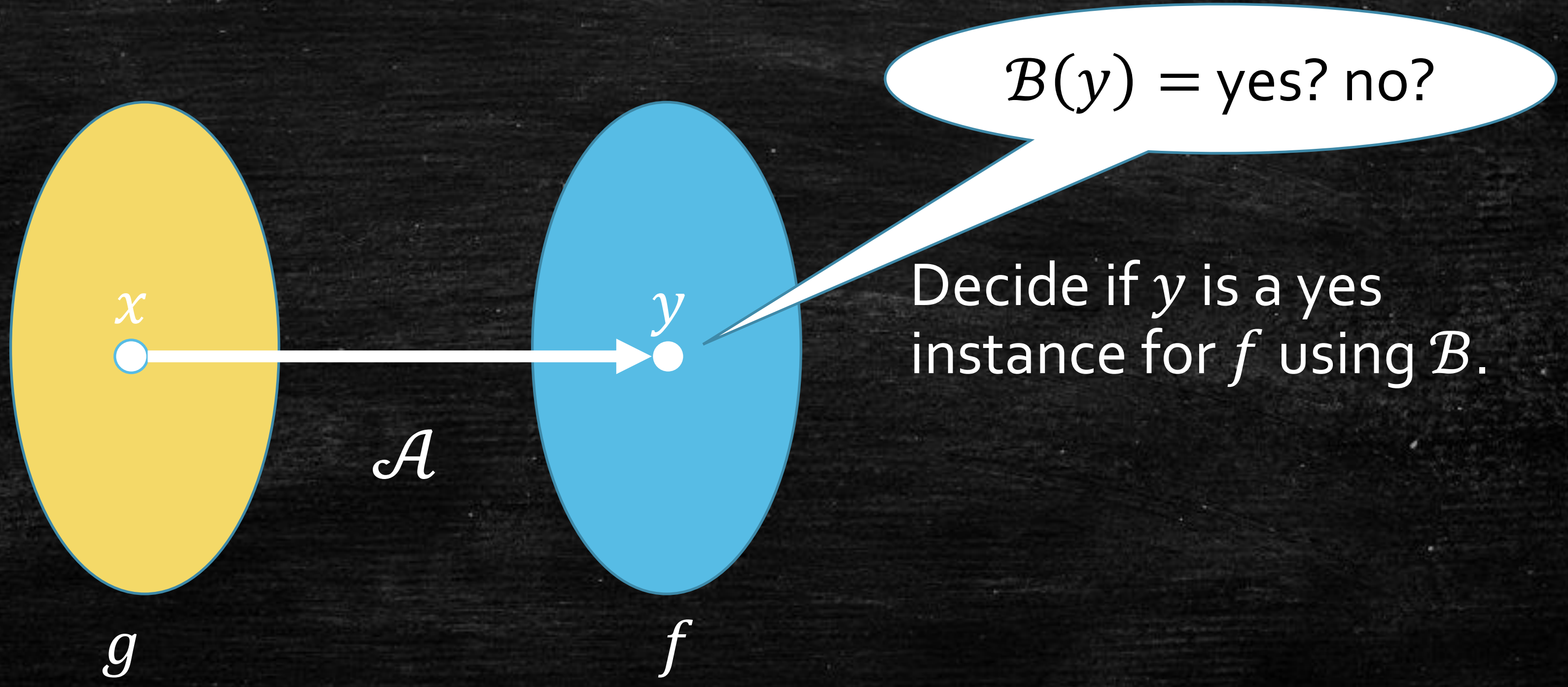
- A poly-time TM $\mathcal{B}$ solving f
- $\Rightarrow$ The TM $\mathcal{B} \circ \mathcal{A}$ solves g



Given any g instance x,
Compute the f instance
$y = \mathcal{A}(x)$.

Reduction: $\mathcal{A}$ computes $g \leq_k f$

- $x \mapsto y$ under poly-time TM $\mathcal{A}$
- x is yes $\Longrightarrow$ y is yes
- x is no $\Longrightarrow$ y is no
- A poly-time TM $\mathcal{B}$ solving f
- $\Longrightarrow$ The TM $\mathcal{B} \circ \mathcal{A}$ solves g



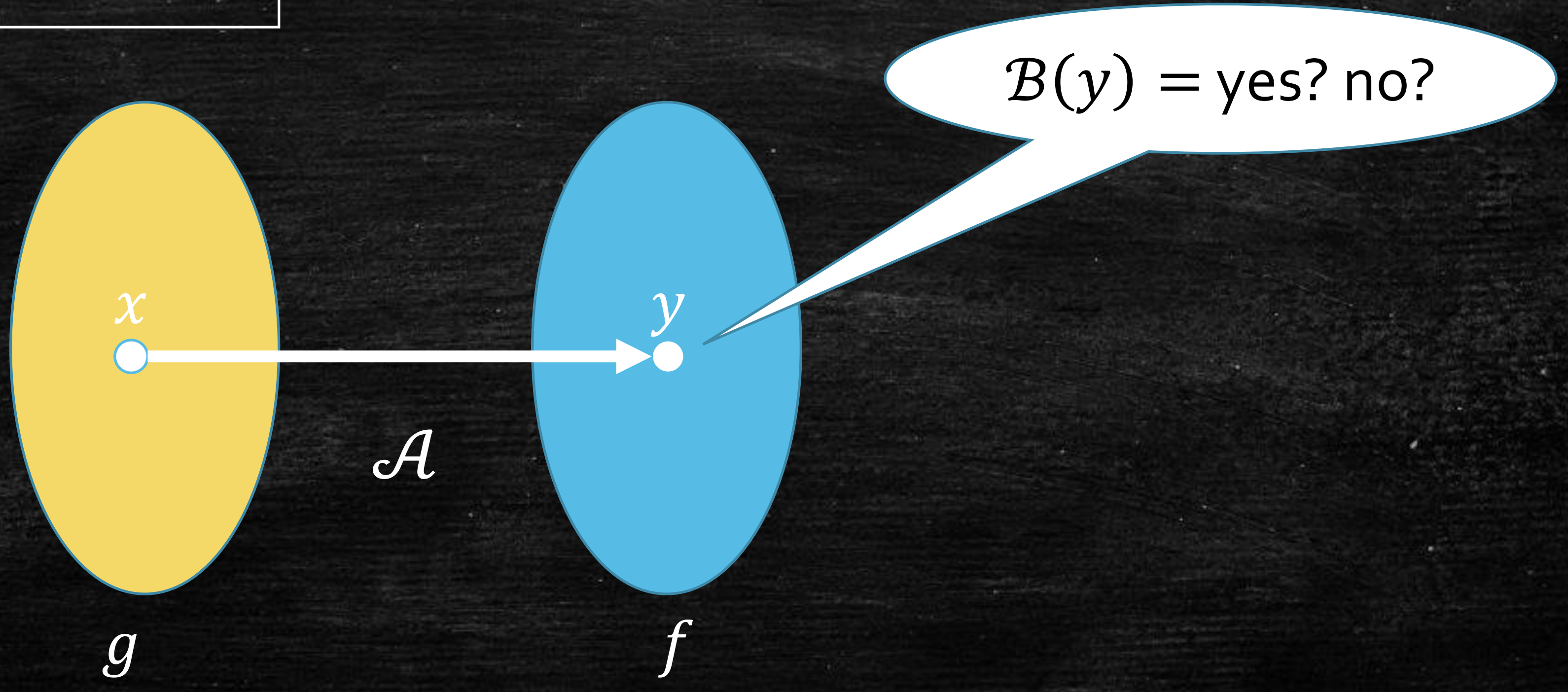
Reduction: $\mathcal{A}$ computes $g \leq_k f$

- $x \mapsto y$ under poly-time TM $\mathcal{A}$
- x is yes $\Longrightarrow$ y is yes
- x is no $\Longrightarrow$ y is no

This is crucial for a reduction to work!

y is yes $\Longrightarrow$ x is yes
y is no $\Longrightarrow$ x is no

- A poly-time TM $\mathcal{B}$ solving f
- $\Longrightarrow$ The TM $\mathcal{B} \circ \mathcal{A}$ solves g



Four Steps for a NP-completeness Proof

1. Prove $f \in \mathbf{NP}$.
2. Construct the reduction $g \leq_k f$.
   - Fix an instance x of g. Describe the corresponding f instance y.
3. [Completeness] x is yes $\Longrightarrow$ y is yes
4. [Soundness] x is no $\Longrightarrow$ y is no
   - Proving the contrapositive "y is yes $\Longrightarrow$ x is yes" is often easier.

NP-hardness for Optimization Problems

Optimization to Decision:

- Maximization $\rightarrow$ decide whether $\mathrm{OPT} \geq k$
- Minimization $\rightarrow$ decide whether $\mathrm{OPT} \leq k$

- A maximization problem is NP-hard if there exists $k \in \mathbb{R}$ such that deciding whether $\mathrm{OPT} \geq k$ is NP-hard.
- A minimization problem is NP-hard if there exists $k \in \mathbb{R}$ such that deciding whether $\mathrm{OPT} \leq k$ is NP-hard.

k-Means

- Input: $S = \{\mathbf{x} : \mathbf{x} \in \mathbb{R}^d\}$ and $k \in \mathbb{Z}^+$
- Output:
  1. Partition of $S = C_1 \cup C_2 \cup \cdots \cup C_k$
  2. A "center" $\mathbf{c}_i \in \mathbb{R}^d$ for each cluster C_i

that minimizes $\sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{c}_i\|^2$

- Only need to specify either output 1 or output 2:
  - Given clusters, optimal centers are easy to compute…
  - Same holds for giving centers.

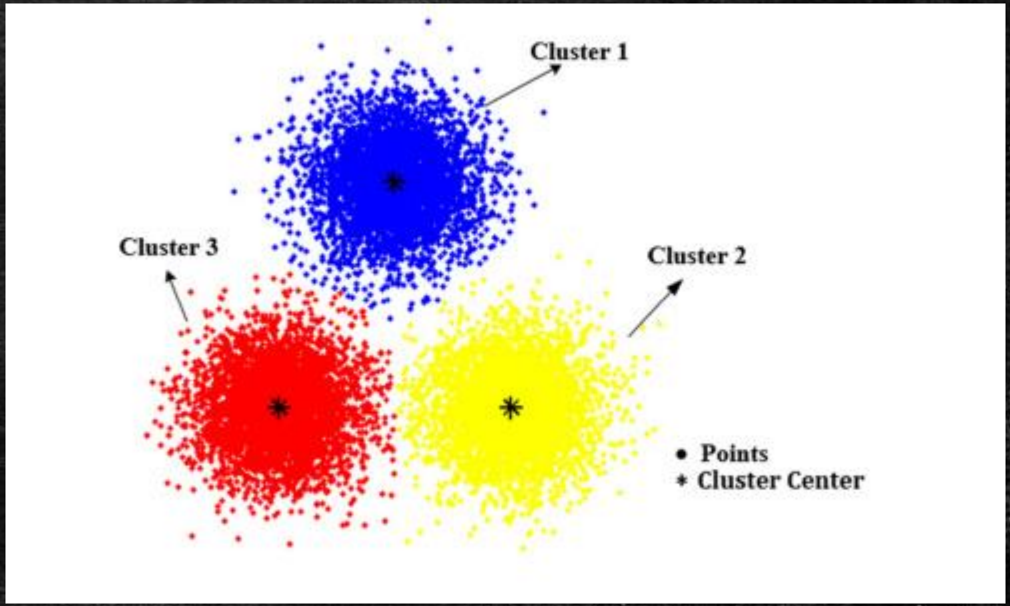


Image from: https://www.osapublishing.org/oe/fulltext.cfm?uri=oe-25-22-27570&id=375887

Proving k-Means Is NP-Hard

Decision version:

- Decide if there exist $C_1, \dots, C_k$ and $\mathbf{c}_1, \dots, \mathbf{c}_k$ s.t.

$$\sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{c}_i\|^2 \leq \theta.$$

- We will show the decision problem is NP-complete.
  - NP-hardness would be suffice, but it is NP-complete anyway...
- We will define the threshold θ later.

Step 1: k-Means ∈ **NP**

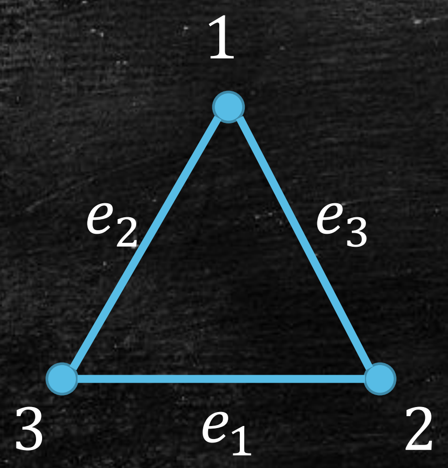
- This is obvious…
- Certificate can be
  - $C_1, \dots, C_k$, or
  - $\mathbf{c}_1, \dots, \mathbf{c}_k$, or
  - both

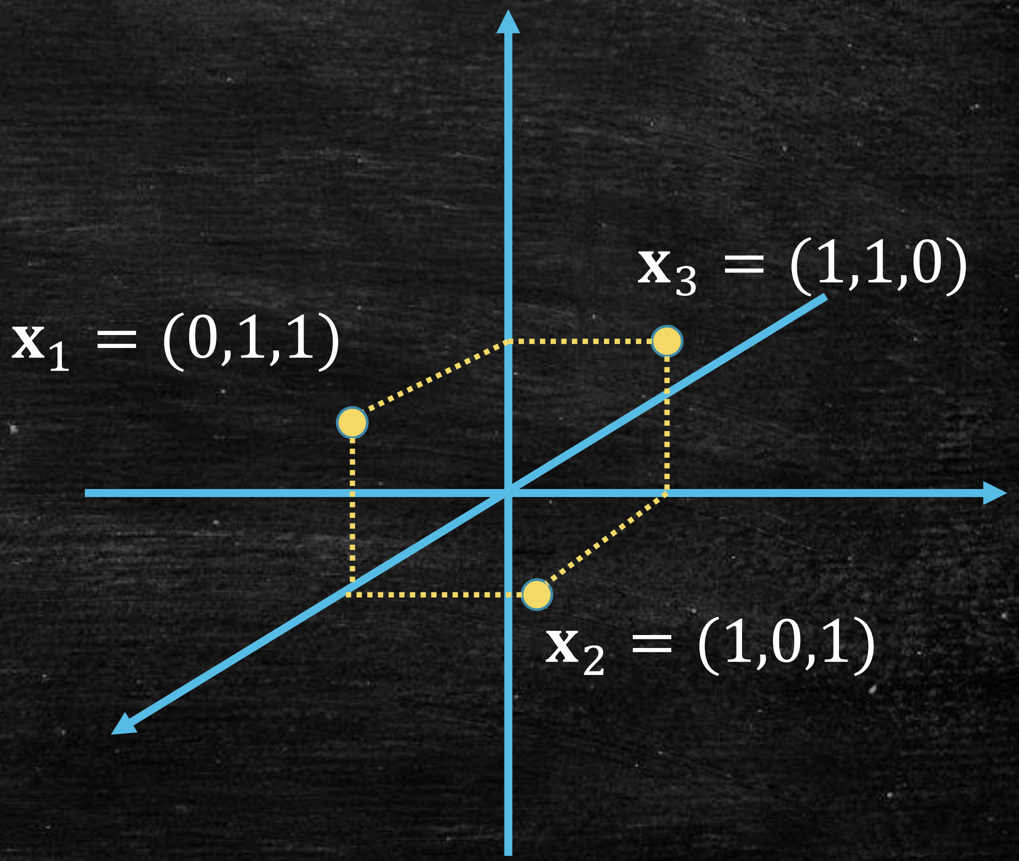
Step 2: Define Construction

- Reduce from VertexCover
- Given any VertexCover instance $(G = (V, E), k)$,
- construct the k-means instance $\left(S = \{\mathbf{x}: \mathbf{x} \in \mathbb{R}^d\}, k, \theta\right)$ as follows:
- Same parameter k in the two instances
- Threshold: $\theta = |E| - k$ (you will see the reason later…)
- Dimension $d = |V|$
- For each $e = (i, j) \in E$, construct a data point

$$\mathbf{x}_e = (0, \dots, 0, \underset{i\text{-th}}{1}, 0, \dots, 0, \underset{j\text{-th}}{1}, 0, \dots, 0)$$

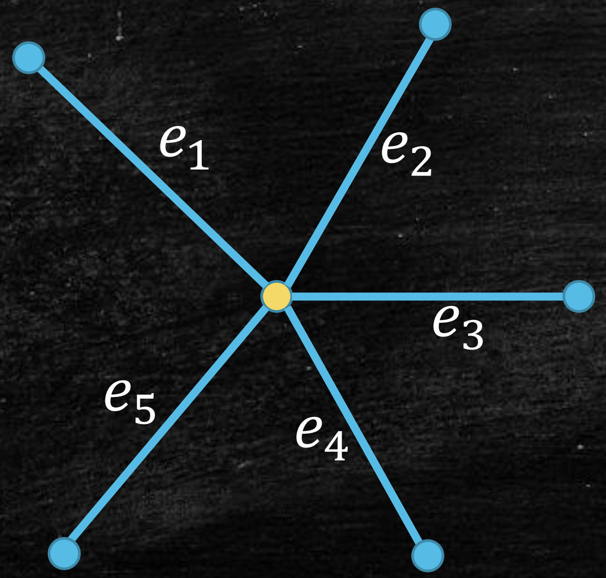
An Example





Intuition for Step 3 & 4

- edges covered by a vertex form a star
- $\Leftrightarrow$ corresponding data points only differ by one entry, they are "very close"



$$
\begin{aligned}
\mathbf{x}_1 &= (1, 1, 0, 0, 0, 0) \\
\mathbf{x}_2 &= (1, 0, 1, 0, 0, 0) \\
\mathbf{x}_3 &= (1, 0, 0, 1, 0, 0) \\
\mathbf{x}_4 &= (1, 0, 0, 0, 1, 0) \\
\mathbf{x}_5 &= (1, 0, 0, 0, 0, 1)
\end{aligned}
$$

Compute the Cost of a Cluster

- For Cluster C, let $G_C = (V, E_C)$ be the subgraph where E_C are the edges whose corresponding data points are in C.
- Let $d_C(i)$ be the degree of i in C.
- **Lemma**. The cost of a cluster C is

$$2|C| - \frac{1}{|C|}\sum_{i=1}^{|V|}\left(d_C(i)\right)^2.$$

Proving $\text{cost}(C) = 2|C| - \frac{1}{|C|}\sum_{i=1}^{|V|}\left(d_C(i)\right)^2$

- Let $\mu = \frac{1}{|C|}\sum_{\mathbf{x}\in C}\mathbf{x}$ be the center of C.
- By thinking about G_C, we have $\mu[i] = \frac{1}{|C|}d_C(i)$.
- $\text{cost}(C) = \sum_{e\in E_C}||\mathbf{x}_e - \mu||^2 = \sum_{e\in E_C}\sum_{i=1}^{|V|}\left(\mathbf{x}_e[i] - \frac{1}{|C|}d_C(i)\right)^2$
- $= \sum_{e\in E_C}\sum_{i=1}^{|V|}\left((\mathbf{x}_e[i])^2 - 2\mathbf{x}_e[i]\frac{1}{|C|}d_C(i) + \left(\frac{1}{|C|}d_C(i)\right)^2\right)$
- $= \sum_{e\in E_C}\sum_{i=1}^{|V|}(\mathbf{x}_e[i])^2 - \sum_{e\in E_C}\sum_{i=1}^{|V|}2\mathbf{x}_e[i]\frac{1}{|C|}d_C(i) + \sum_{e\in E_C}\sum_{i=1}^{|V|}\left(\frac{1}{|C|}d_C(i)\right)^2$

Proving $\mathrm{cost}(C) = 2|C| - \frac{1}{|C|}\sum_{i=1}^{|V|}\left(d_C(i)\right)^2$

- $\mathrm{cost}(C) = \sum_{e\in E_C}\sum_{i=1}^{|V|}(\mathbf{x}_e[i])^2 - \sum_{e\in E_C}\sum_{i=1}^{|V|} 2\mathbf{x}_e[i]\frac{1}{|C|}d_C(i) + \sum_{e\in E_C}\sum_{i=1}^{|V|}\left(\frac{1}{|C|}d_C(i)\right)^2$
- red $= \sum_{e\in E_C} 2 = 2|C|$
- blue $= \sum_{i=1}^{|V|}\sum_{e\in E_C} 2\mathbf{x}_e[i]\frac{1}{|C|}d_C(i) = \frac{2}{|C|}\cdot\sum_{i=1}^{|V|}\left(d_C(i)\right)^2$
- purple $= |C|\cdot\frac{1}{|C|^2}\cdot\sum_{i=1}^{|V|}\left(d_C(i)\right)^2 = \frac{1}{|C|}\cdot\sum_{i=1}^{|V|}\left(d_C(i)\right)^2$
- Putting together:

$$\mathrm{cost}(C) = 2|C| - \frac{1}{|C|}\sum_{i=1}^{|V|}\left(d_C(i)\right)^2$$

Part 3: yes to yes

- Suppose $(G = (V, E), k)$ is a yes instance and S is a vertex cover.
- Let $S = \{1, 2, \dots, k\}$ WLOG.
- Let C_i be those $\mathbf{x}_e$ where e is covered by vertex i
  - If $i, j \in S$ for $e = (i, j)$, include $\mathbf{x}_e$ in any one of C_i, C_j (not both!)
- G_{C_i} is a star:
  - one vertex with degree $|C_i|$, and $|C_i|$ vertices with degree 1
- $\mathrm{cost}(C_i) = 2|C_i| - \frac{1}{|C_i|}\left(|C_i|^2 + 1^2 + \cdots + 1^2\right) = |C_i| - 1$
- Overall cost: $\sum_{i=1}^{k} \mathrm{cost}(C_i) = \left(\sum_{i=1}^{k} |C_i|\right) - k = |E| - k = \theta$
- The k-means instance is yes!

Part 4: no to no (contrapositive)

- Suppose the k-means instance is a yes instance, and the cost of $\{C_1, \dots, C_k\}$ is at most $\theta = |E| - k$.
- **Proposition.** $\mathrm{cost}(C_i) \geq |C_i| - 1$, and $\mathrm{cost}(C_i) = |C_i| - 1$ only if G_{C_i} is a star.
- Suppose G_{C_i} is not a star for some C_i. It's a contradiction:

$$\mathrm{OverallCost} = \sum_{i=1}^{k} \mathrm{cost}(C_i) > \sum_{i=1}^{k} (|C_i| - 1) = |E| - k = \theta.$$

- Thus, each G_{C_i} is a star.
- Those k "central vertex" of the k stars form a vertex cover!

Stronger Hardness Results for k-Means

- k-means is NP-hard even when $k = 2$
  - [Aloise, Deshpande, Hansen & Popat, 2009] [Dasgupta & Freund, 2009]
- k-means is NP-hard even for $\mathbb{R}^2$
  - [Mahajan, Nimbhorkar & Varadarajan, 2009]
- There exists a constant $\varepsilon > 0$ such that k-means is NP-hard to approximate within factor $(1 - \varepsilon)$.
  - [Awasthi, Charikar, Krishnaswamy & Sinop, 2015]

Positive Results for k-Means

- There exists a poly-time $(9 + \varepsilon)$-approximation algorithm.
  - [Kanungo, Mount, Netanyahu, Piatko, Silverman & Wu, 2003]
- Lloyd's heuristic, EM-heuristic
  - No theoretical approximation guarantee

0-1 Integer Programming

$$
\begin{aligned}
\text{maximize} \quad & \mathbf{c}^\top \mathbf{x} \\
\text{subject to} \quad & A\mathbf{x} \le \mathbf{b} \\
& x_i \in \{0, 1\}
\end{aligned}
$$

- 0-1 Integer Programming is NP-hard.
- It can formulate many NP-complete problems, e.g., VertexCover

$$
\begin{aligned}
\text{minimize} \quad & \sum_{v \in V} x_v & \\
\text{subject to} \quad & x_u + x_v \ge 1 & \forall (u, v) \in E \\
& x_v \in \{0, 1\} & \forall v \in V
\end{aligned}
$$

IP (Feasibility)

- Deciding whether the feasible region of an IP is non-empty is NP-complete.

- VertexCover:

$$\sum_{v \in V} x_v \leq k$$

$$x_u + x_v \geq 1 \qquad \forall (u,v) \in E$$

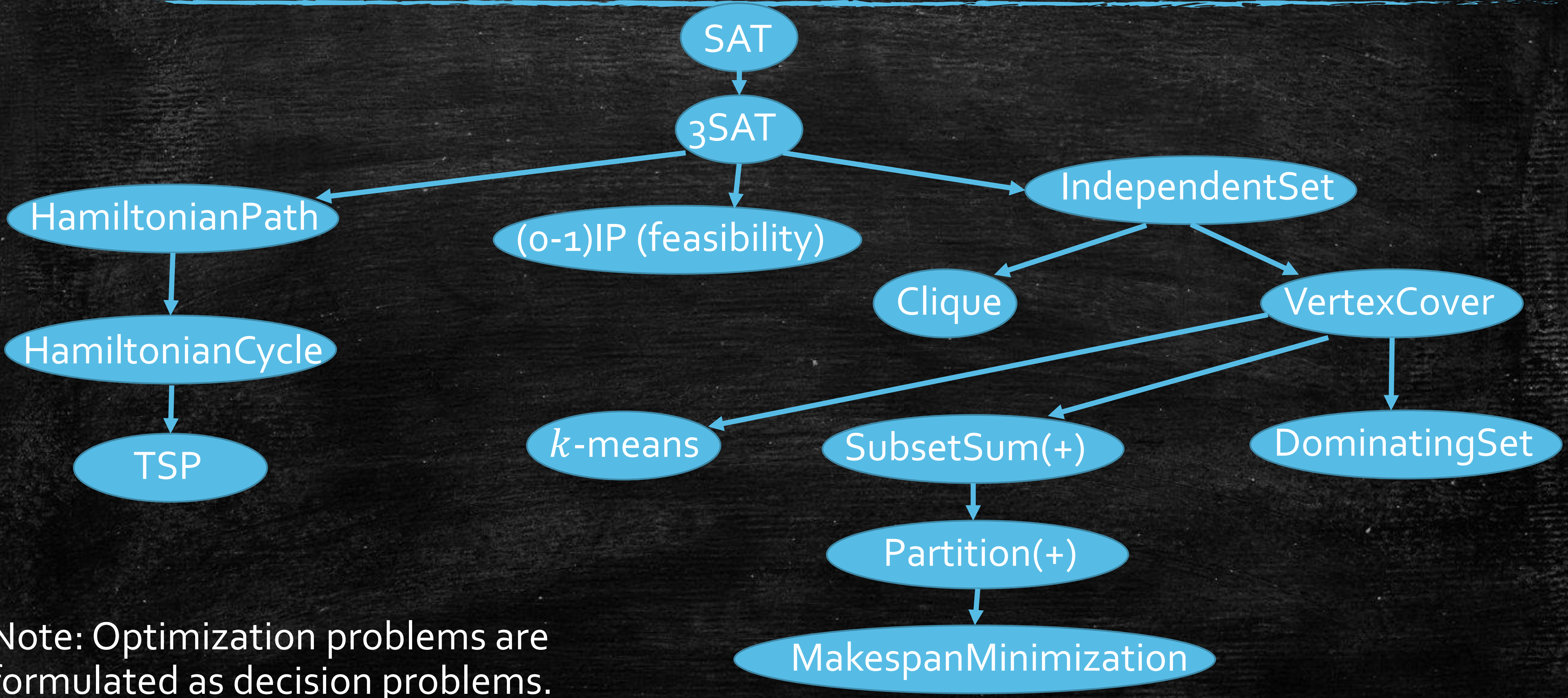
$$x_v \in \{0,1\} \qquad \forall v \in V$$

IP: Hardness of Approximation

- Even if we only allow feasible IP as input, IP is still hard to approximate (just like TSP).

$$
\begin{aligned}
\text{minimize} \quad & 1000000000y + \sum_{v \in V} x_v \\
\text{subject to} \quad & x_u + x_v + y \geq 1 && \forall (u, v) \in E \\
& x_v \in \{0, 1\} && \forall v \in V \\
& y \in \{0, 1\}
\end{aligned}
$$

Web of NP-Complete Problems



Note: Optimization problems are formulated as decision problems.

Deal with NP-hard Optimization Problems

Three approaches to handle NP-hard problems:

1. Approximation algorithms
2. Assumption on inputs
3. Heuristics
   - Heuristics: "algorithms" without theoretical support; their performances are normally justified by experiments/simulations
   - NP-hardness is about worst-case analysis. Heuristics may do well on most of the "practical instances".

Approximation Algorithm for Min-VertexCover

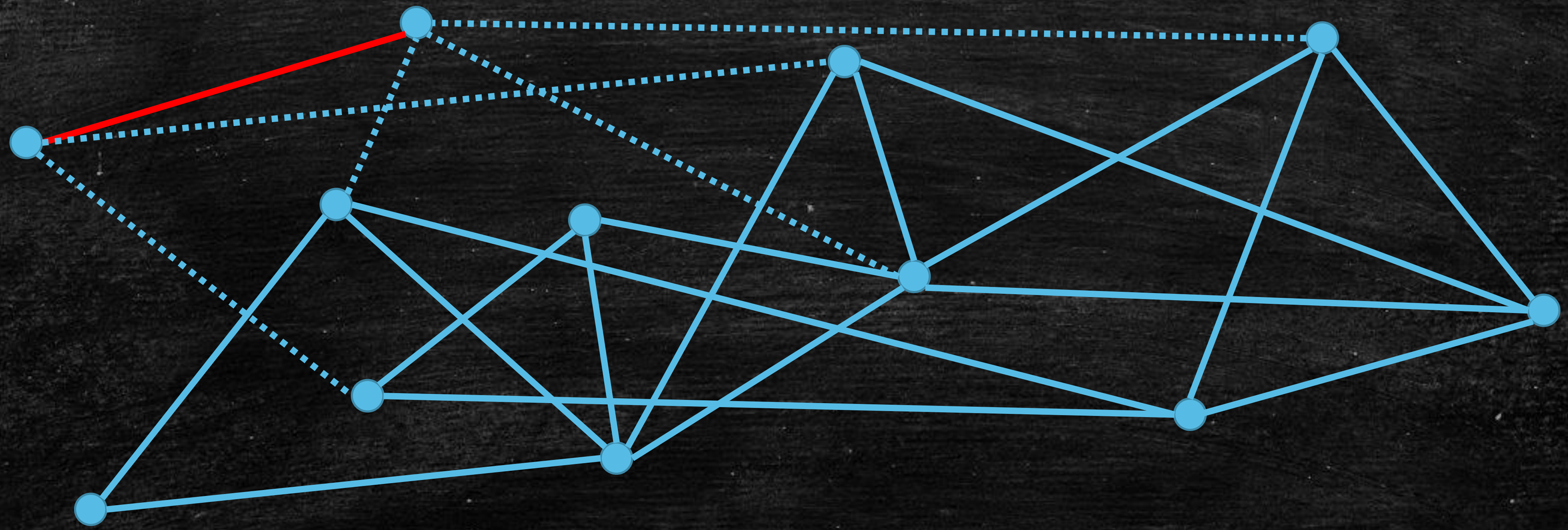
- Input: an undirected graph $G = (V, E)$
- Output: a vertex cover S with minimum $|S|$

Maximal Matching

- A matching M is maximal if no more edge can be added to M while still forming a matching.
- Finding a maximal matching is simple: just iteratively add an edge until no more edges can be added!

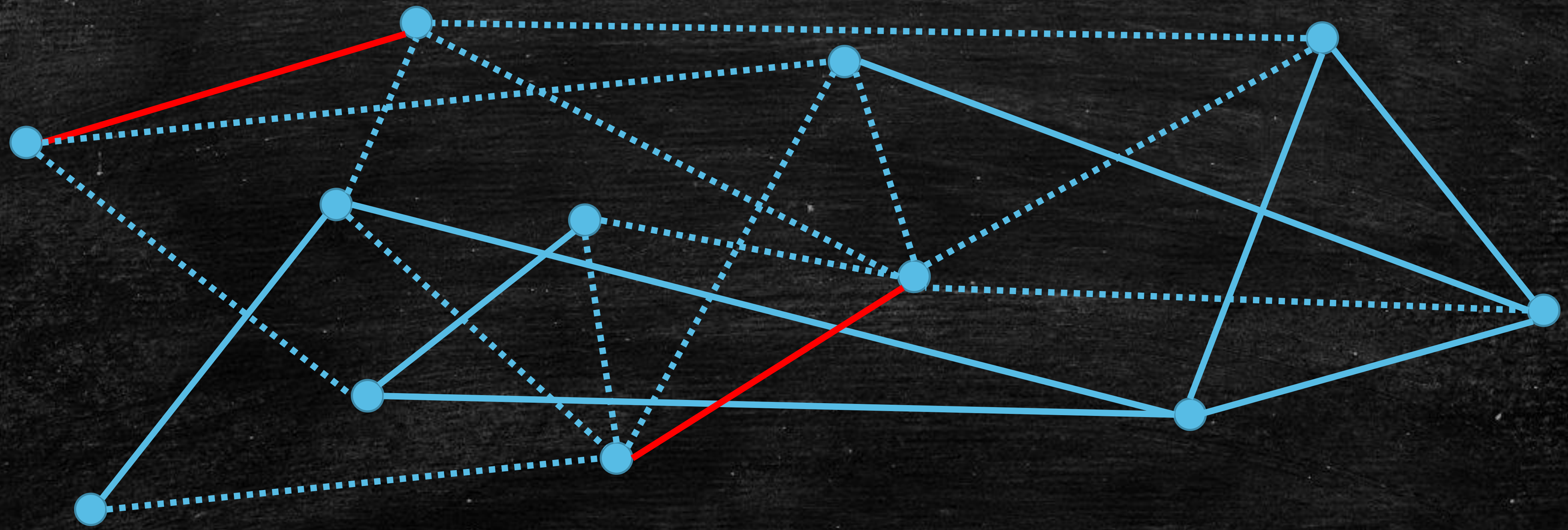
Finding a maximal matching

- Iteratively add an edge until no more edges can be added!



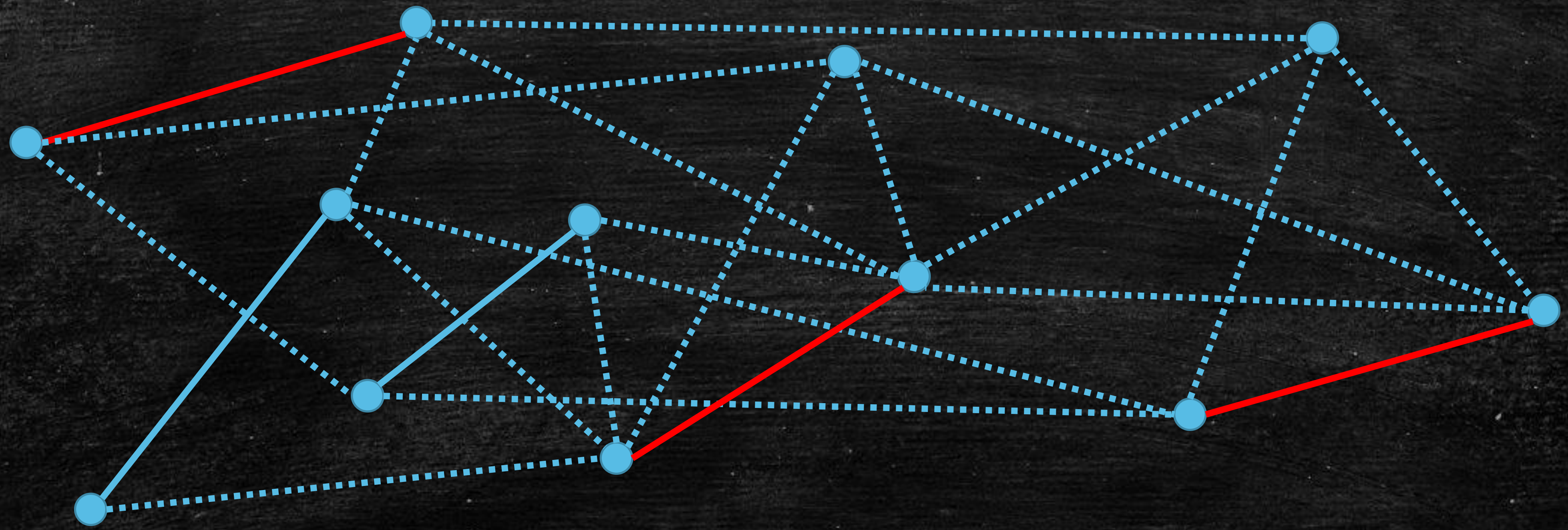
Finding a maximal matching

- Iteratively add an edge until no more edges can be added!



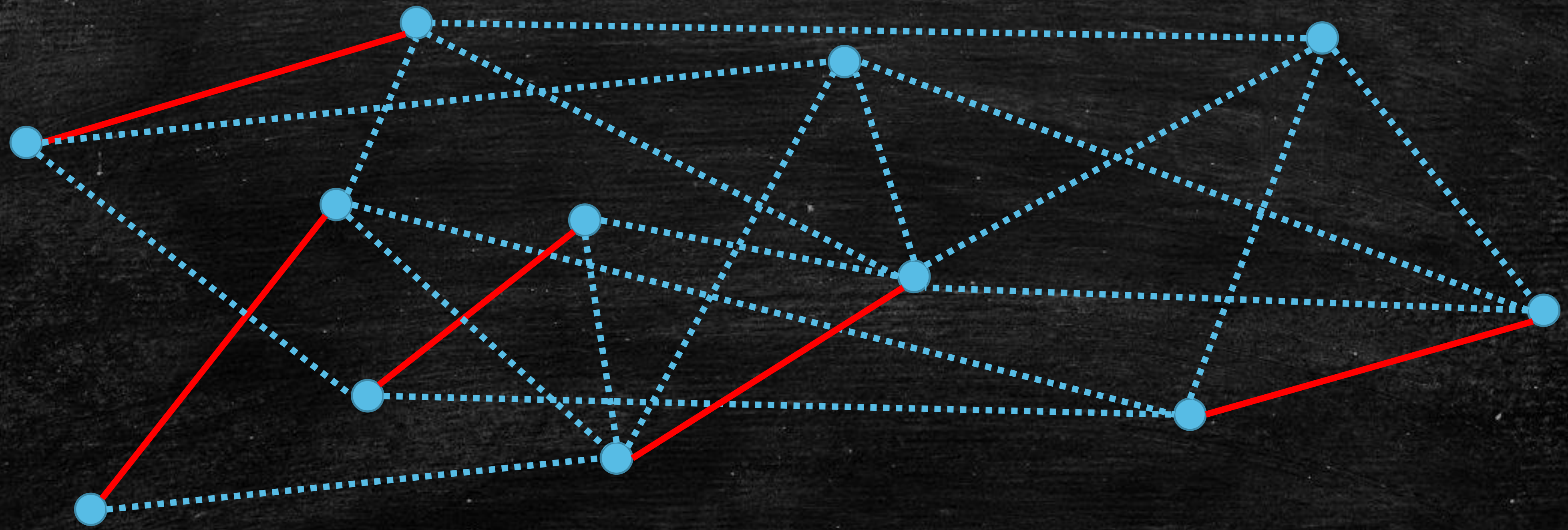
Finding a maximal matching

- Iteratively add an edge until no more edges can be added!



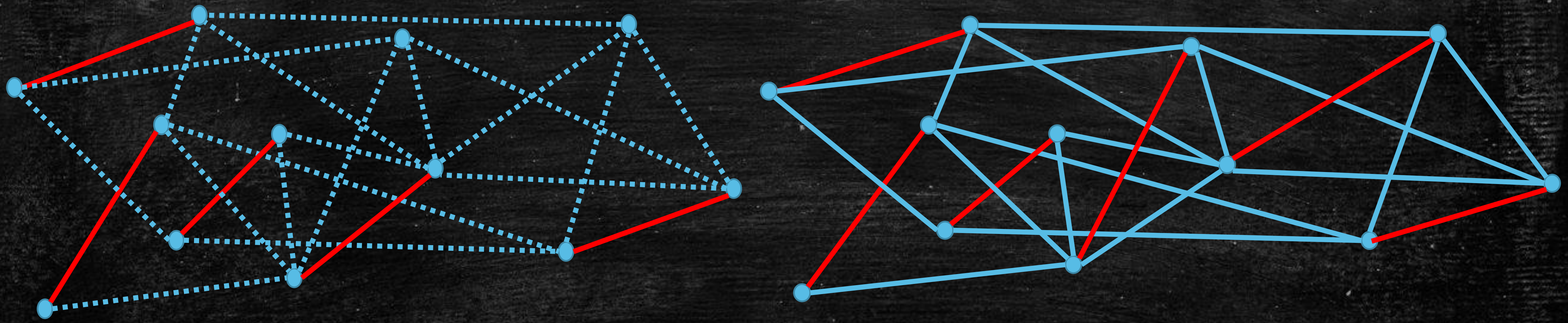
Finding a maximal matching

- Iteratively add an edge until no more edges can be added!

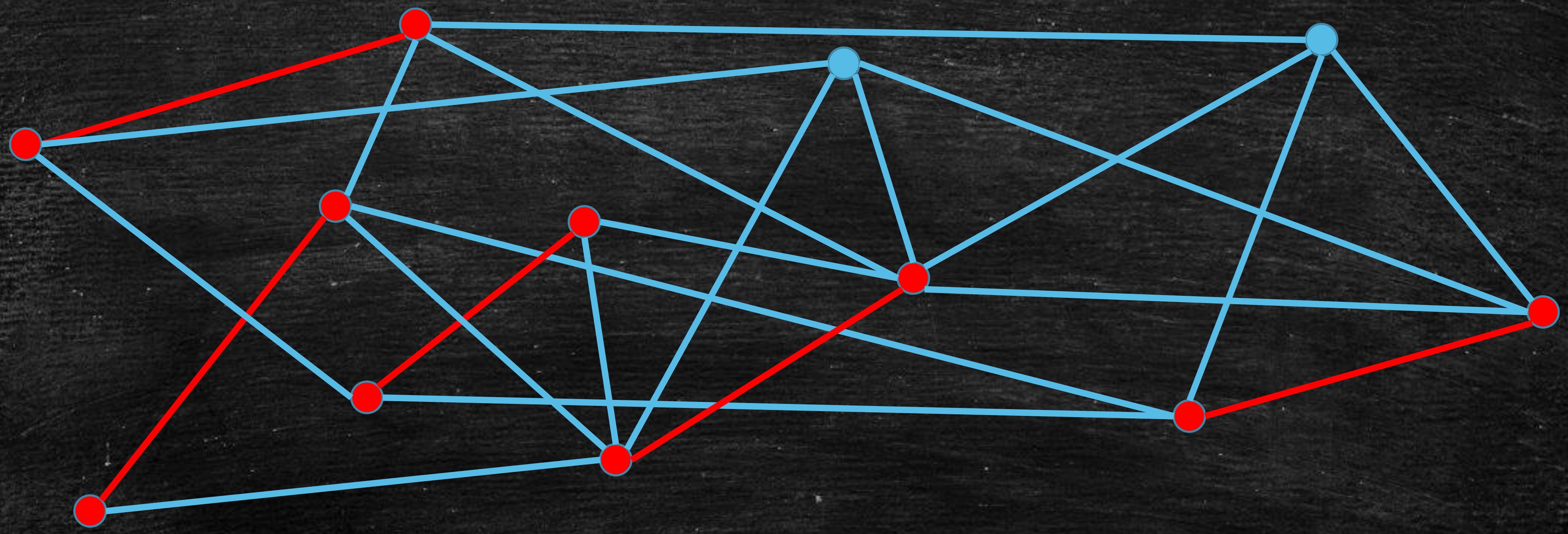


Maximal vs Maximum

- A maximal matching may not be maximum!



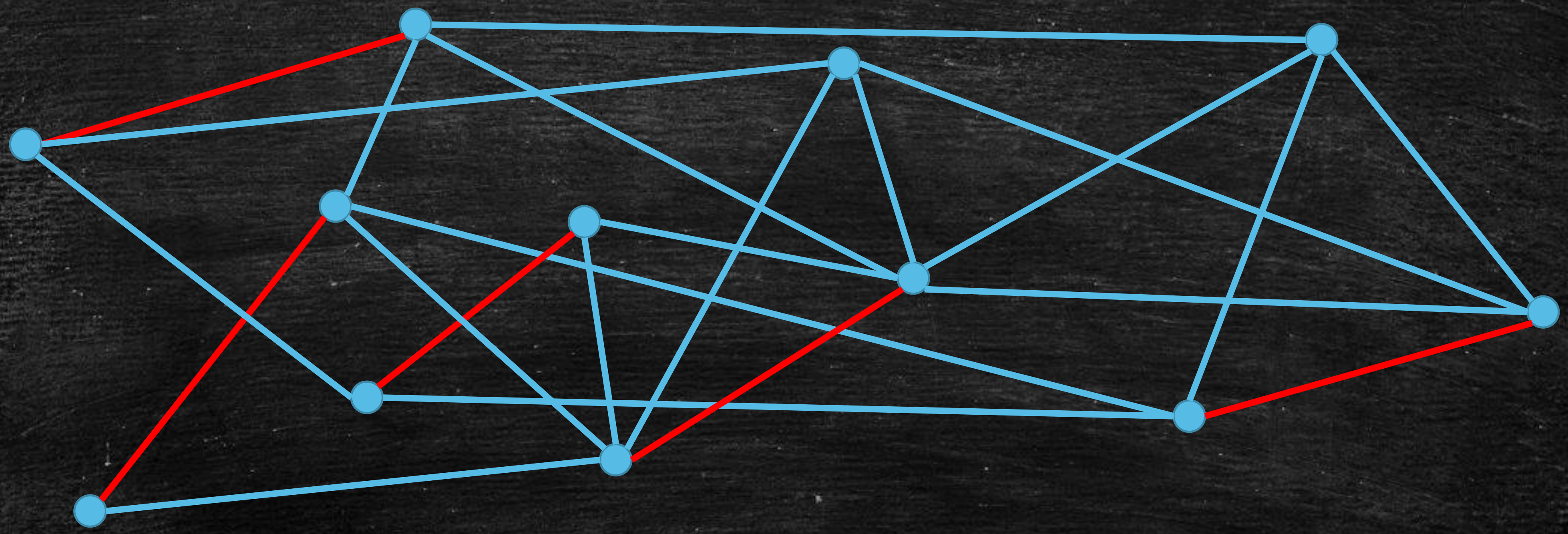
Lemma 1. The set of endpoints for all edges in a maximal matching is a vertex cover.



Proof. Let $M \subseteq E$ be a maximal matching.

- For any edge $e = (u, v)$, one or both of u, v must be an endpoint of an edge in M. (Otherwise, $M \cup \{e\}$ is still a matching, and M is not maximal.)
- This already implies endpoints of M is a vertex cover!

Lemma 2. *For any maximal matching M, the size of any vertex cover is at least $|M|$.*



Proof.

- Edges in M must be covered
- A vertex cannot cover two edges in M
- We need $|M|$ vertices to at least cover edges in M

A 2-approximation algorithm

Algorithm 1:

- Find a maximal matching M
- Let S be the endpoints of all edges in M
- Output S

Given an undirected graph $G = (V, E)$, let
- $OPT(G)$ be the size of a minimum vertex cover
- $S(G)$ be the vertex set output by Algorithm 1

Theorem: *For any undirected graph G, we have* $|S(G)| \leq 2 \cdot OPT(G)$

$\forall G: |S(G)| \leq 2 \cdot OPT(G)$

- Lemma 1. *The set of endpoints for all edges in a maximal matching is a vertex cover.*
- $\Rightarrow S(G)$ is a vertex cover
- $|S(G)| = 2|M|$
- Lemma 2: *For any maximal matching* M, *the size of any vertex cover is at least* $|M|$.
- $\Rightarrow OPT(G) \geq |M|$

Approximation Algorithm

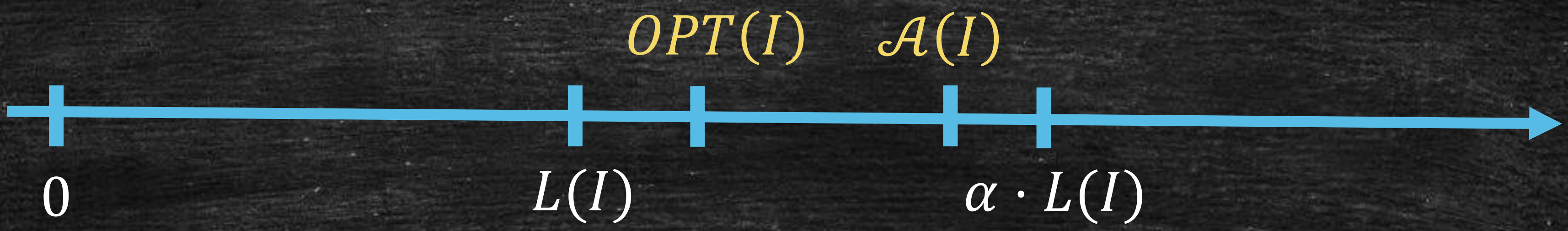
- Definition. Consider a minimization problem and an algorithm $\mathcal{A}$ for it. Given a instance I, let $\mathcal{A}(I)$ be the value output by $\mathcal{A}$ for input I, let $OPT(I)$ be the optimal solution for I. $\mathcal{A}$ is an α-approximation algorithm if

$$\forall I: \quad \frac{\mathcal{A}(I)}{OPT(I)} \leq \alpha$$

- Definition. For maximization problem, $\mathcal{A}$ is an α-approximation algorithm if

$$\forall I: \quad \frac{\mathcal{A}(I)}{OPT(I)} \geq \alpha$$

General Framework for Designing Approximation Algorithms



- Find a lower bound $L(I)$ for $OPT(I)$ (that is easy to calculate)
- Design algorithm $\mathcal{A}$ and find some α such that $\forall I$: $\mathcal{A}(I) \leq \alpha \cdot L(I)$

Revisiting our 2-approximation algorithm

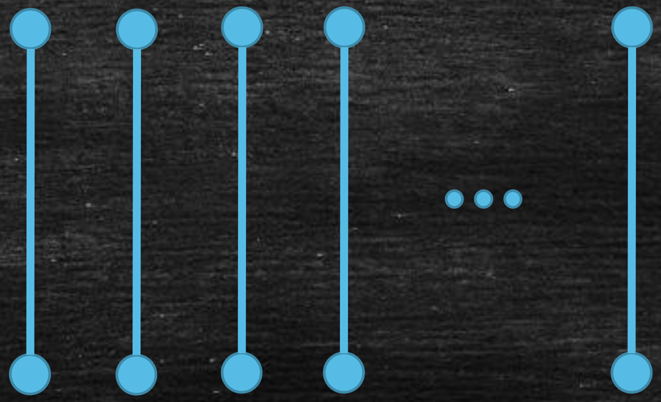
Algorithm 1:

- Find a maximal matching M
- Let S be the endpoints of all edges in M
- Output S

Question: Can we do better than 2-approximation?

- Idea 1: same algorithm with a more careful analysis?
- Idea 2: another more clever algorithm?

Idea 1 doesn't work

$G =$ 

- Suppose G has $2n$ vertices and n edges as above.
- $OPT(G) = n$
- $\mathcal{A}(G) = 2n$

Idea 2 is unlikely to work

- [Khot & Regev, 2008] Assuming Unique Game Conjecture, if minimum vertex cover has a polynomial time $(2-\epsilon)$-approximation algorithm for some $\epsilon > 0$, then $\mathbf{P} = \mathbf{NP}$.
- [Khot, Minzer & Safra, 2017] If minimum vertex cover has a polynomial time $(\sqrt{2}-\epsilon)$-approximation algorithm for some $\epsilon > 0$, then $\mathbf{P} = \mathbf{NP}$.

Once we have an α-approximation algorithm…

Two natural directions for improving α:

- A more careful analysis
- A new approximation algorithm

Approximation Algorithms Based on LP-Relaxation

- Integer Programming is NP-complete, even for 0-1 case $\forall i\colon\ x_i \in \{0, 1\}$.
- Use the fact that LP is polynomial-time solvable to design approximation algorithm.
- Relax $x_i \in \{0,1\}$ to $0 \le x_i \le 1$.
- Then "round" the fractional solution to integral one:
  - E.g., $x_i = 0.7$ is rounded to $x_i = 1$, $x_i = 0.2$ is rounded to $x_i = 0$.
- and show that the rounded solution is feasible and achieves good approximation guarantee.

LP-Relaxation Example: Vertex Cover

- Minimum Vertex Cover Formulation by integer program:
  - $x_u = 1$ represents $u \in V$ is selected in the cover; $x_u = 0$ otherwise.

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{v \in V} x_v & \\
\text{subject to} & x_u + x_v \geq 1 & \forall (u, v) \in E \\
 & x_v \in \{0, 1\} & \forall v \in V
\end{array}
$$

LP-Relaxation Example: Vertex Cover

- Relax it to a linear program below:

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in V} x_v & \\ \text{subject to} \quad & x_u + x_v \geq 1 & \forall (u, v) \in E \\ & 0 \leq x_v \leq 1 & \forall v \in V \end{aligned}$$

LP-Relaxation Example: Vertex Cover

- OPT(IP) – optimal objective value $\sum_{v \in V} x_v$ for IP
  - This is the objective we want for vertex cover
- OPT(LP) – optimal objective value $\sum_{v \in V} x_v$ for LP
- OPT(IP) $\geq$ OPT(LP): because LP has a larger feasible region.

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{v \in V} x_v & \\
\text{subject to} & x_u + x_v \geq 1 & \forall (u, v) \in E \\
& x_v \in \{0, 1\} & \forall v \in V
\end{array}
$$

Integer Program (IP)

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{v \in V} x_v & \\
\text{subject to} & x_u + x_v \geq 1 & \forall (u, v) \in E \\
& 0 \leq x_v \leq 1 & \forall v \in V
\end{array}
$$

Linear Program (LP)

LP-Relaxation Example: Vertex Cover

An approximation algorithm for vertex cover:

- Formulate the problem as an integer program and obtain its LP-relaxation.
- Solve the linear program and obtain its optimal solution $\{x_v^*\}_{v \in V}$.
- Return $S = \{v \mid x_v^* \geq \frac{1}{2}\}$

Correctness

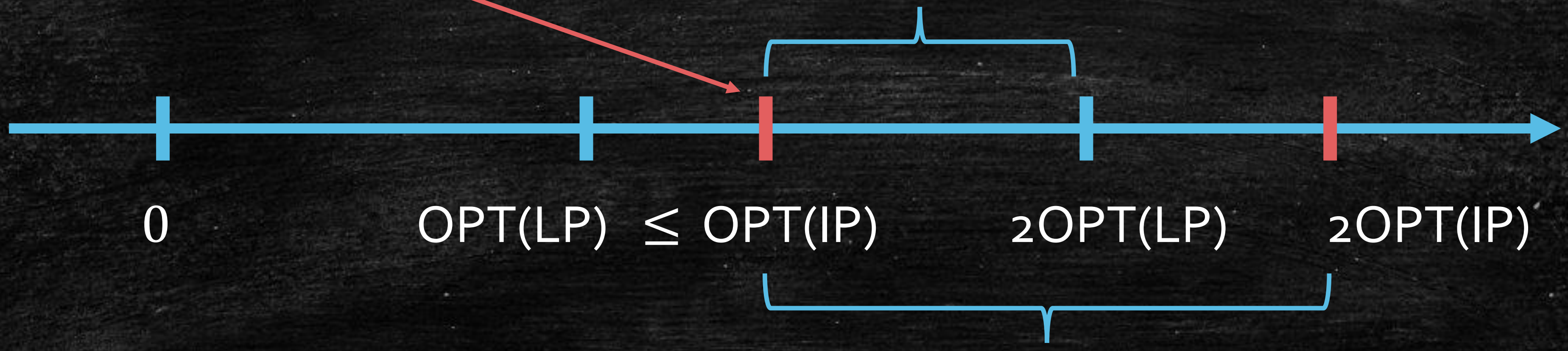
S returned by the algorithm is a vertex cover.

- Proof. Consider an arbitrary edge $(u, v) \in E$.
- We have $x_u^* + x_v^* \geq 1$ by feasibility, which implies we have either $x_u^* \geq \frac{1}{2}$ or $x_v^* \geq \frac{1}{2}$, or both.
- By our algorithm, we have either $u \in S$ or $v \in S$, or both.

The algorithm is a 2-approximation.

The algorithm is a 2-approximation algorithm: $|S| \leq 2 \cdot \mathrm{OPT(IP)}$.

- Proof. Since we have $\mathrm{OPT(IP)} \geq \mathrm{OPT(LP)}$, it suffices to prove $|S| \leq 2 \cdot \mathrm{OPT(LP)}$.



The algorithm is a 2-approximation.

The algorithm is a 2-approximation algorithm: $|S| \le 2 \cdot \mathrm{OPT(IP)}$.

- Proof. Since we have $\mathrm{OPT(IP)} \ge \mathrm{OPT(LP)}$, it suffices to prove $|S| \le 2 \cdot \mathrm{OPT(LP)}$.
- $\mathrm{OPT(LP)} = \sum_{v \in V} x_v^* = \sum_{v : x_v^* < \frac{1}{2}} x_v^* + \sum_{v : x_v^* \ge \frac{1}{2}} x_v^*$
- $\ge \sum_{v : x_v^* < \frac{1}{2}} 0 + \sum_{v : x_v^* \ge \frac{1}{2}} \frac{1}{2} = \frac{1}{2} \cdot |S|$
- which implies $|S| \le 2 \cdot \mathrm{OPT(LP)}$.

Let’s Come Back to our two questions

Question: Can we do better than 2-approximation?

- Idea 1: same algorithm with a more careful analysis?
- Idea 2: another more clever algorithm?

- We know the answer to 2 is probably no…
- Let’s forget about this for a moment…
- LP-Relaxation: how to analyze “it more carefully”?

Integrality Gap

- $\mathrm{IntegralityGap} = \frac{\mathrm{OPT(IP)}}{\mathrm{OPT(LP)}}$
- If you analyze your approximation algorithm based on $\mathrm{OPT(LP)}$...
- the best approximation ratio you can ever get is the integrality gap!

Integrality Gap for Vertex Cover

- Consider a complete graph with n vertices.
- $\mathrm{OPT(IP)} = n - 1$: you need $n - 1$ vertices to cover all edges
- $\mathrm{OPT(LP)} = \frac{n}{2}$: just assign $x_v = \frac{1}{2}$ for all $v \in V$.
- Integrality gap is 2.

Metric TSP

[TSP]

- Input: a complete weighted graph $G = (V, E = V \times V, w)$
- Output: a Hamiltonian cycle with minimum weight

[Metric TSP]

- Input: a complete weighted graph $G = (V, E = V \times V, w)$ such that $w(u, v) + w(v, w) \geq w(u, w)$ for any $u, v, w \in V$
- Output: a Hamiltonian cycle with minimum weight

Metric TSP is NP-hard

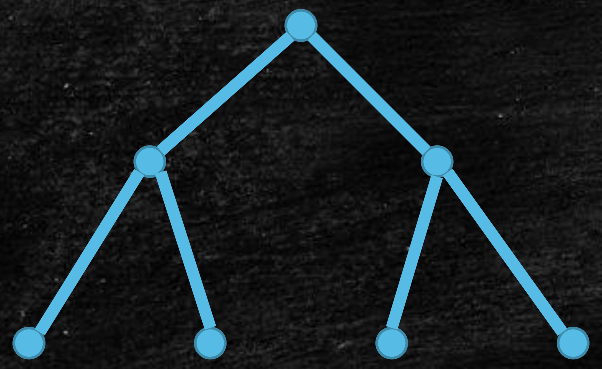
- HamiltonianCycle instance $G' = (V, E')$
- TSP instance $G = (V, E = V \times V, w)$ with

$$w(u, v) = f(x) = \begin{cases} 1, & (u, v) \in E \\ 2, & (u, v) \notin E \end{cases}$$

- Yes HamiltonianCycle instance $\Longrightarrow\ \mathrm{OPT}_{\mathrm{TSP}} = |V|$
- No HamiltonianCycle instance $\Longrightarrow\ \mathrm{OPT}_{\mathrm{TSP}} \geq |V| + 1$

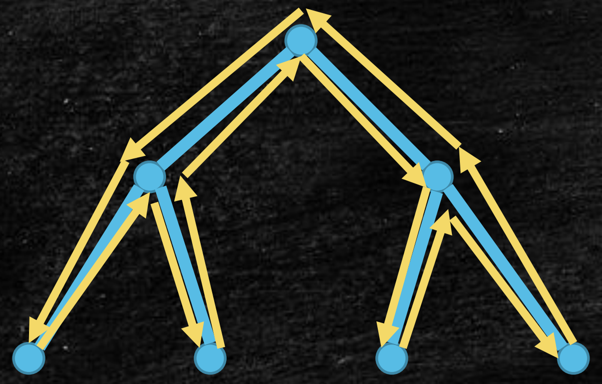
Approximation Algorithm for TSP

1. Find a minimum weight spanning tree T.



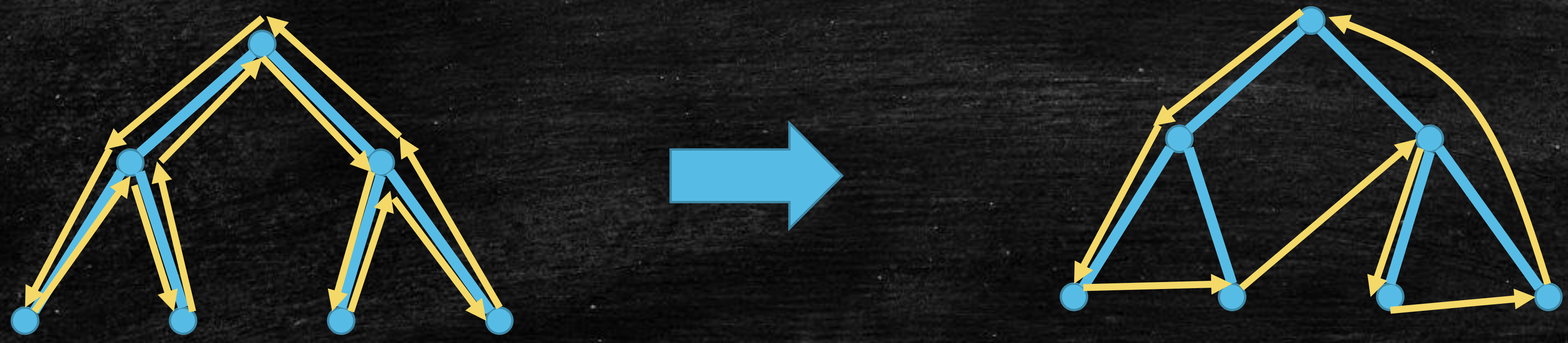
Approximation Algorithm for TSP

1. Find a minimum weight spanning tree T.
2. Find a tour C in T that visit each edge exactly twice.



Approximation Algorithm for TSP

1. Find a minimum weight spanning tree T.
2. Find a tour C' in T that visit each edge exactly twice.
3. Shortcut C' to get C by skipping visited vertices.
   – So we get a valid Hamiltonian cycle…
4. Return C.

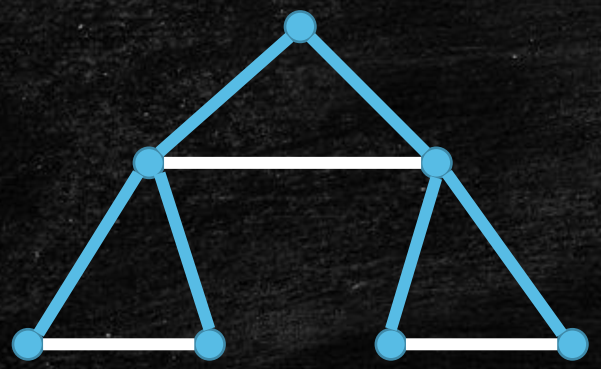


2-Approximation

- $\mathrm{OPT_{TSP}} \geq w(T)$:
  - A Hamiltonian path is a spanning tree.
  - Min spanning tree ≤ min Hamiltonian Path ≤ min Hamiltonian Cycle
- $w(C') = 2w(T) \leq 2\mathrm{OPT_{TSP}}$
- $w(C) \leq w(C')$
  - Triangle inequality
- Putting together: $w(C) \leq 2\mathrm{OPT_{TSP}}$

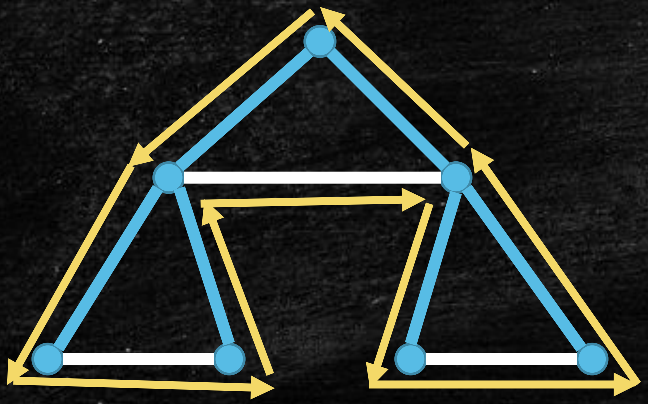
Christofides algorithm

1. Find a minimum weight spanning tree T.
2. Find a minimum weight perfect matching M on $U \subseteq V$, where U are odd-degree vertices in T.



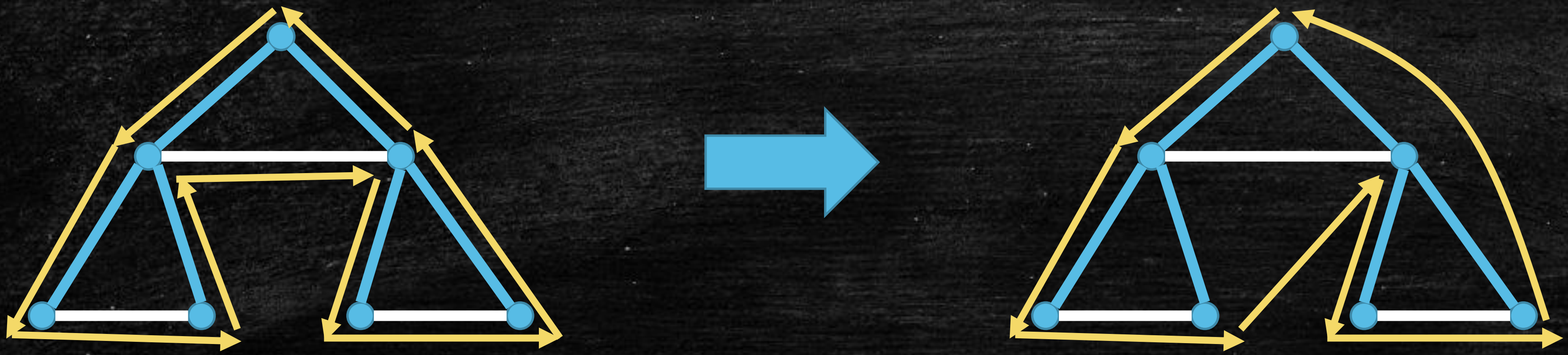
Christofides algorithm

1. Find a minimum weight spanning tree T.
2. Find a <u>minimum weight perfect matching</u> M on $U \subseteq V$, where U are odd-degree vertices in T.
3. Find a Eulerian tour C' on $T \cup M$.



Christofides algorithm

1. Find a minimum weight spanning tree T.
2. Find a minimum weight perfect matching M on $U \subseteq V$, where U are odd-degree vertices in T.
3. Find a Eulerian tour C' on $T \cup M$.
4. Shortcut C' to C by skipping visited vertices.

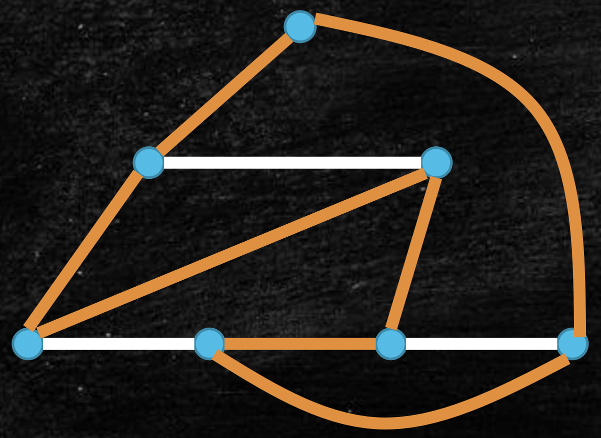


1.5-Approximation

- Same as before: $\mathrm{OPT}_{\mathrm{TSP}} \geq w(T)$
- $w(C) \leq w(C') = w(T) + w(M)$
- We aim to show $w(M) \leq 0.5\ \mathrm{OPT}_{\mathrm{TSP}}$

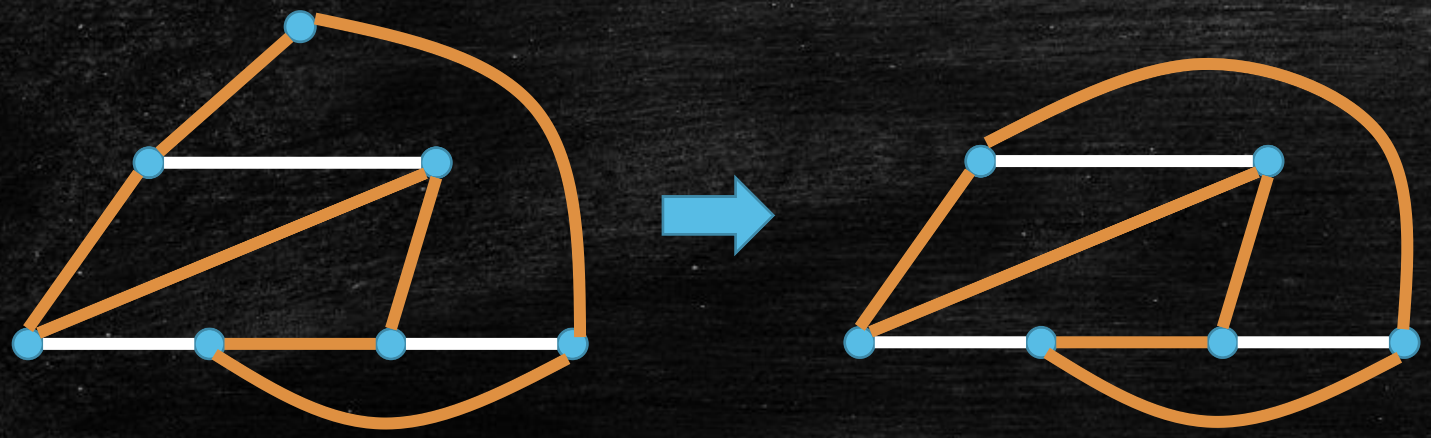
$w(M) \leq 0.5\ \mathrm{OPT}_{\mathrm{TSP}}$

- Let O be the optimal cycle.



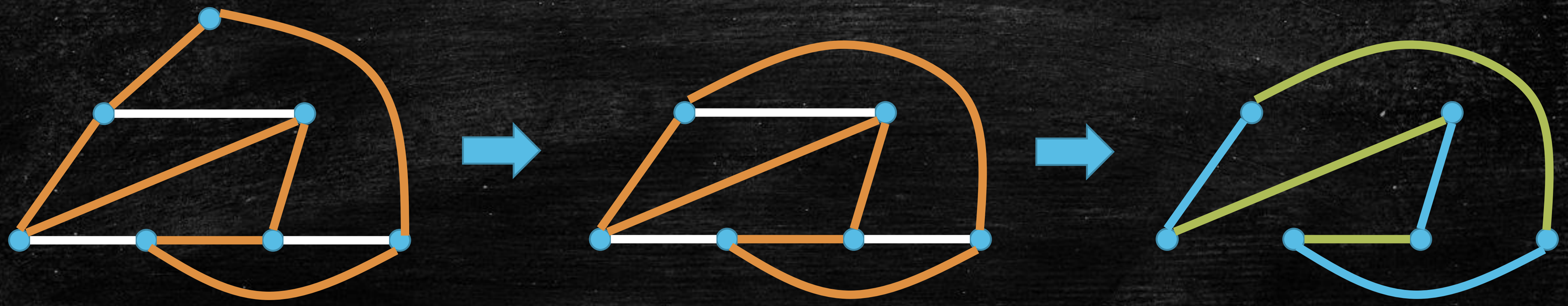
$w(M) \leq 0.5\ \mathrm{OPT}_{\mathrm{TSP}}$

- Let O be the optimal cycle.
- Let O' shortcut those vertices not in the matching.



$w(M) \leq 0.5\ \mathrm{OPT}_{\mathrm{TSP}}$

- Let O be the optimal cycle.
- Let O' shortcut those vertices not in the matching.
- Two disjoint matchings M_1, M_2 in O'
- $w(M_1) + w(M_2) = w(O') \leq w(O)$ (triangle inequality)
- One of M_1 or M_2 has weight at most $0.5 w(O)$



$w(M) \leq 0.5\ \mathrm{OPT}_{\mathrm{TSP}}$

- Let O be the optimal cycle.
- Let O' shortcut those vertices not in the matching.
- Two disjoint matchings M_1, M_2 in O'
- $w(M_1) + w(M_2) = w(O') \leq w(O)$ (triangle inequality)
- One of M_1 or M_2 has weight at most $0.5w(O)$
- Since M has minimum weight…
- $w(M) \leq 0.5w(O) = 0.5\mathrm{OPT}_{\mathrm{TSP}}$

Metric TSP Results

- A $\left(1.5 - 10^{-36}\right)$-approximation algorithm
  - [Karlin, Klein, Gharan, 2020]
- NP-hard to approximate with factor $\frac{123}{122}$.
  - [Karpinski, Lampis & Schmied, 2015]

This Lecture

NP-Hardness:

- One more reduction: NP-hardness of k-means

Approximation Algorithms:

- Example:
  - VertexCover (2-approximation)
  - TSP (1.5-approximation)
- Framework:
  - find an approachable lower bound L (or upper bound in the maximization case) of OPT;
  - Show that $\mathrm{ALG} \leq \alpha \cdot L$
- Two techniques for designing approximation algorithms:
  - Combinatorial
  - LP-relaxation (Integrality Gap to analyze approximation ratio)