# Approximation Algorithms

Lecture 7

September 16, 2014

# Today's Lecture

Don't give up on **NP-Hard** problems:

**(A)** Faster exponential time algorithms: $n^{O(n)}$, $3^n$, $2^n$, etc.

**(B)** Fixed parameter tractable.

**(C)** Find an approximate solution.

# Part I

# Greedy algorithms and approximation algorithms

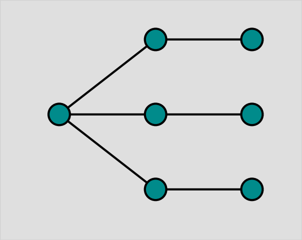
# Greedy algorithms

1. ***greedy algorithms***: do locally the right thing...
2. ...and they suck.

**VertexCoverMin**

**Instance**: A graph **G**.
**Question:** Return the smallest subset $S \subseteq V(\mathbf{G})$, s.t. $S$ touches all the edges of **G**.

3. **GreedyVertexCover**: pick vertex with highest degree, remove, repeat.



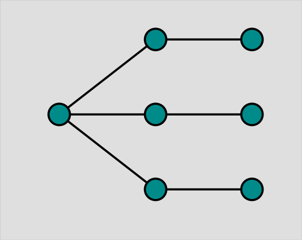
# Greedy algorithms

1. ***greedy algorithms***: do locally the right thing...
2. ...and they suck.

**VertexCoverMin**

> **Instance**: A graph $\mathbf{G}$.
> **Question:** Return the smallest subset $S \subseteq V(\mathbf{G})$, s.t. $S$ touches all the edges of $\mathbf{G}$.

3. **GreedyVertexCover**: pick vertex with highest degree, remove, repeat.



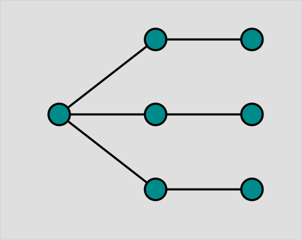
# Greedy algorithms

1. ***greedy algorithms***: do locally the right thing...
2. ...and they suck.

**VertexCoverMin**

> **Instance**: A graph **G**.
> **Question:** Return the smallest subset $S \subseteq V(\mathbf{G})$, s.t. $S$ touches all the edges of **G**.

3. **GreedyVertexCover**: pick vertex with highest degree, remove, repeat.



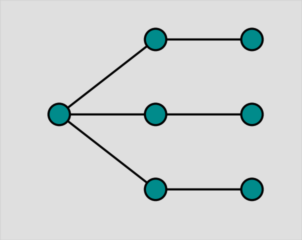
# Greedy algorithms

1. ***greedy algorithms***: do locally the right thing...
2. ...and they suck.

**VertexCoverMin**

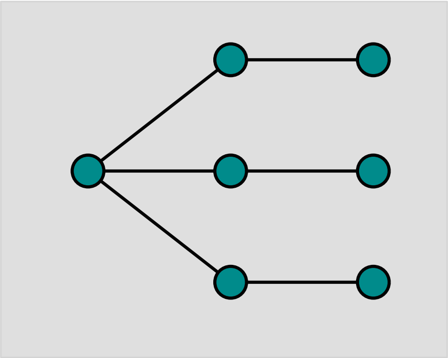
**Instance**: A graph $\mathbf{G}$.
**Question:** Return the smallest subset $S \subseteq V(\mathbf{G})$, s.t. $S$ touches all the edges of $\mathbf{G}$.

3. **GreedyVertexCover**: pick vertex with highest degree, remove, repeat.

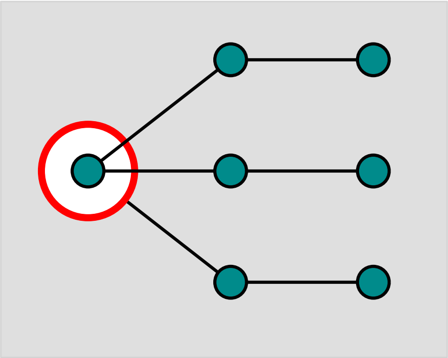


# Greedy algorithms

## **GreedyVertexCover** in action...

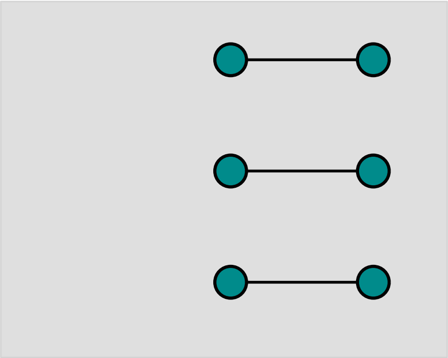


## GreedyVertexCover in action...



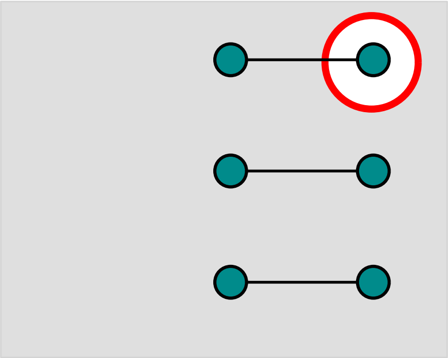
# Greedy algorithms

## **GreedyVertexCover** in action...



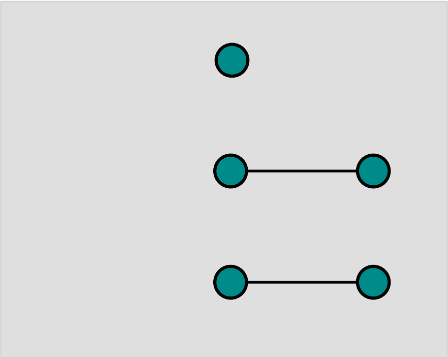
# Greedy algorithms

## GreedyVertexCover in action...



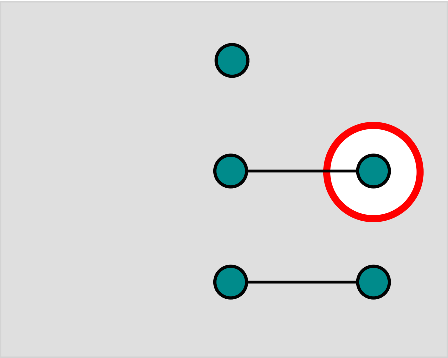
# Greedy algorithms

## GreedyVertexCover in action...



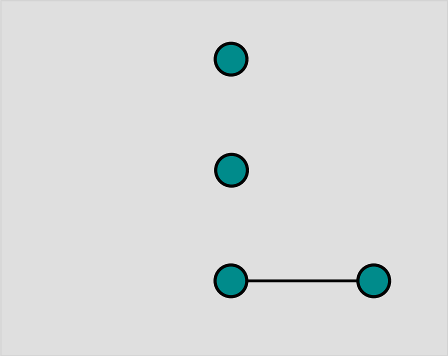
# Greedy algorithms

## GreedyVertexCover in action...



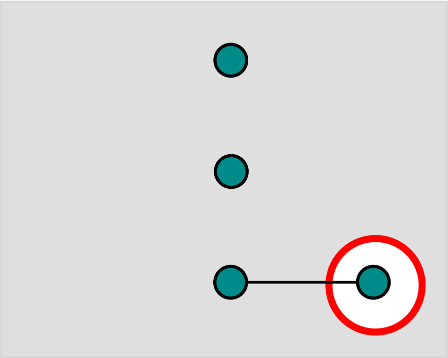
# Greedy algorithms

## **GreedyVertexCover** in action...



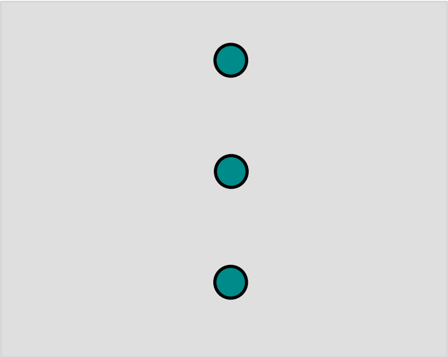
# Greedy algorithms

## **GreedyVertexCover** in action...



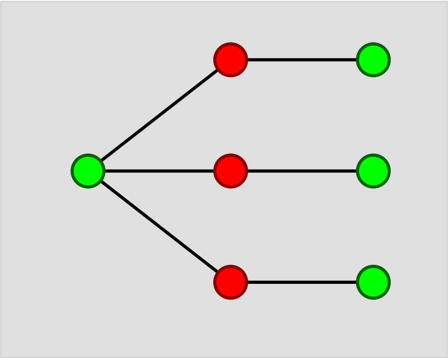
# Greedy algorithms

## **GreedyVertexCover** in action...



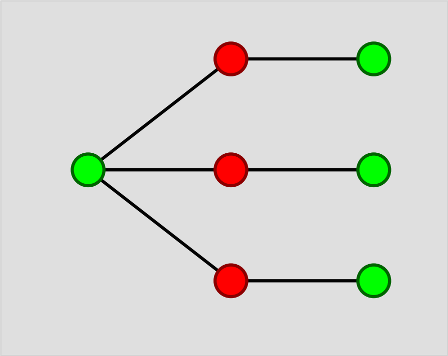
# Greedy algorithms

## **GreedyVertexCover** in action...



# Greedy algorithms

## **GreedyVertexCover** in action...



### Observation

**GreedyVertexCover** *returns 4 vertices, but opt is 3 vertices.*

# Good enough...

## Definition

In a ***minimization*** optimization problem, one looks for a valid solution that minimizes a certain target function.

1. VertexCoverMin: $\mathbf{Opt}(\mathbf{G}) = \min_{S \subseteq V(\mathbf{G}), S \text{ cover of } G} |S|$.
2. $\mathbf{VertexCover}(\mathbf{G})$: set realizing sol.
3. $\mathbf{Opt}(\mathbf{G})$: value of the target function for the optimal solution.

# Good enough...

## Definition

In a ***minimization*** optimization problem, one looks for a valid solution that minimizes a certain target function.

1. VertexCoverMin: $\mathbf{Opt}(\mathsf{G}) = \mathbf{min}_{S \subseteq V(\mathsf{G}), S \text{ cover of } G} |S|$.
2. $\mathbf{VertexCover}(\mathsf{G})$: set realizing sol.
3. $\mathbf{Opt}(\mathsf{G})$: value of the target function for the optimal solution.

# Good enough...

## Definition

In a ***minimization*** optimization problem, one looks for a valid solution that minimizes a certain target function.

1. VertexCoverMin: $\mathbf{Opt}(\mathsf{G}) = \min_{S \subseteq V(\mathsf{G}), S \text{ cover of } G} |S|$.
2. $\mathbf{VertexCover}(\mathsf{G})$: set realizing sol.
3. $\mathbf{Opt}(\mathsf{G})$: value of the target function for the optimal solution.

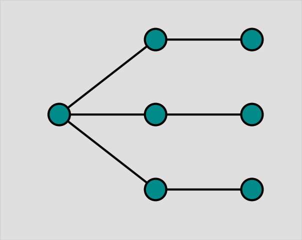
## Definition

**Alg** is $\alpha$-***approximation algorithm*** for problem **Min**, achieving an approximation $\alpha \geq 1$, if for all inputs **G**, we have:

$$\frac{\mathsf{Alg}(\mathsf{G})}{\mathbf{Opt}(\mathsf{G})} \leq \alpha.$$

# Back to GreedyVertexCover

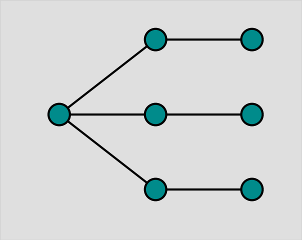
1. **GreedyVertexCover**: pick vertex with highest degree, remove, repeat.
2. Returns 4, but opt is 3!



3. Can **not** be better than a $4/3$-approximation algorithm.
4. Actually it is much worse!

# Back to GreedyVertexCover

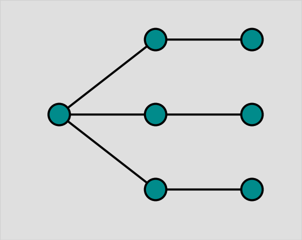
1. **GreedyVertexCover**: pick vertex with highest degree, remove, repeat.
2. Returns 4, but opt is 3!



3. Can **not** be better than a $4/3$-approximation algorithm.
4. Actually it is much worse!

# Back to GreedyVertexCover

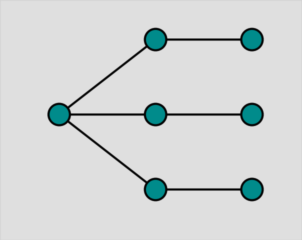
1. **GreedyVertexCover**: pick vertex with highest degree, remove, repeat.
2. Returns 4, but opt is 3!



3. Can **not** be better than a $4/3$-approximation algorithm.
4. Actually it is much worse!

# Back to GreedyVertexCover

1. **GreedyVertexCover**: pick vertex with highest degree, remove, repeat.
2. Returns 4, but opt is 3!



3. Can **not** be better than a $4/3$-approximation algorithm.
4. Actually it is much worse!

# How bad is **GreedyVertexCover**?

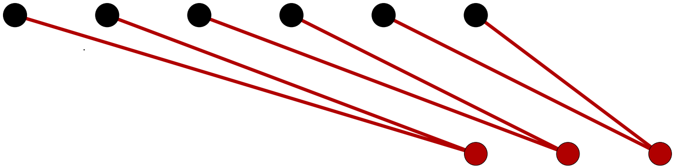
Build a bipartite graph.

Let the top partite set be of size $n$.

# How bad is **GreedyVertexCover**?

Build a bipartite graph.

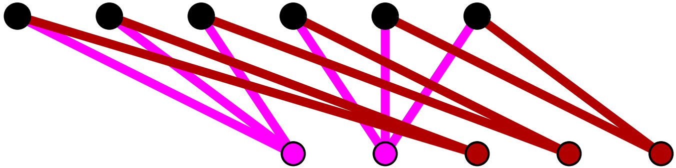
In the bottom set add $\lfloor n/2 \rfloor$ vertices of degree $2$, such that each edge goes to a different vertex above.



# How bad is **GreedyVertexCover**?

Build a bipartite graph.

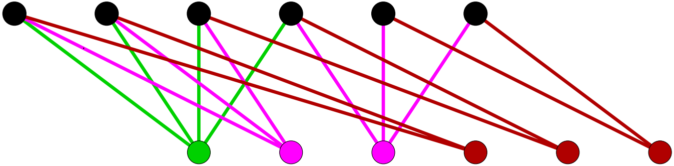
Repeatedly add $\lfloor n/i \rfloor$ bottom vertices of degree $i$, for $i = 2, \ldots, n$.



# How bad is **GreedyVertexCover**?

Build a bipartite graph.

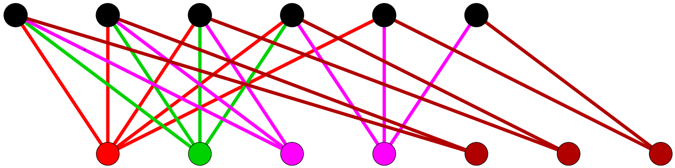
Repeatedly add $\lfloor n/i \rfloor$ bottom vertices of degree $i$, for $i = 2, \ldots, n$.



# How bad is **GreedyVertexCover**?

Build a bipartite graph.

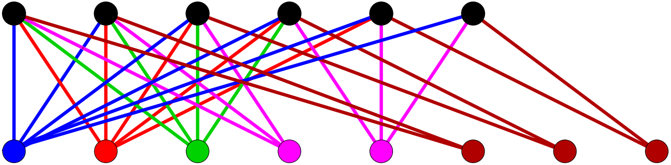
Repeatedly add $\lfloor n/i \rfloor$ bottom vertices of degree $i$, for $i = 2, \ldots, n$.



# How bad is **GreedyVertexCover**?

Build a bipartite graph.

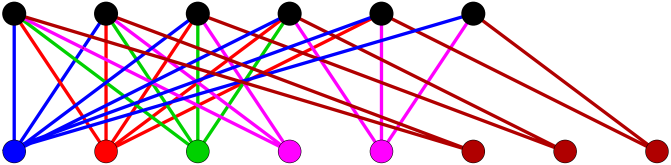
Repeatedly add $\lfloor n/i \rfloor$ bottom vertices of degree $i$, for $i = 2, \ldots, n$.



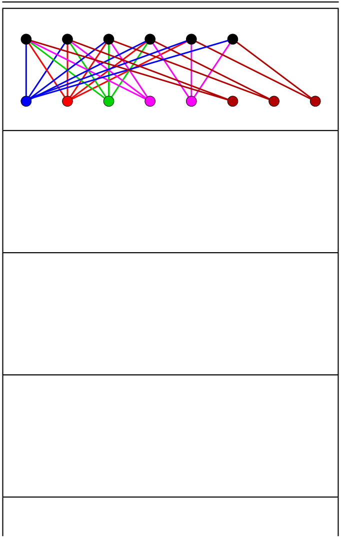
# How bad is **GreedyVertexCover**?

Build a bipartite graph.

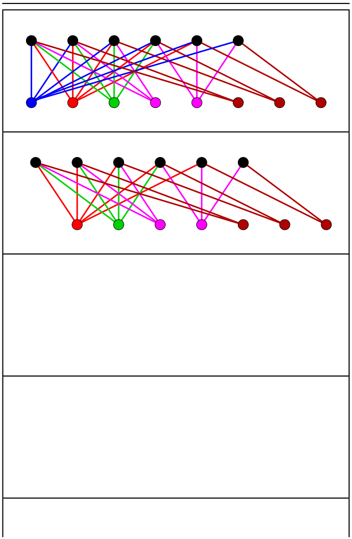
Bottom row has $\sum_{i=2}^{n} \lfloor n/i \rfloor = \Theta(n \log n)$ vertices.



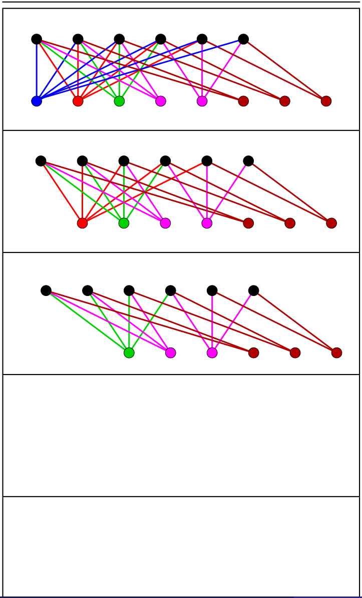
# How bad is **GreedyVertexCover**?



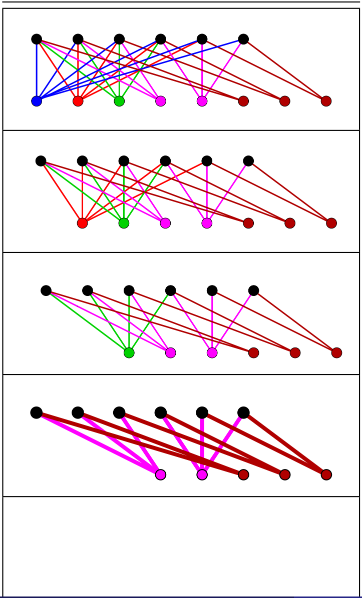
# How bad is **GreedyVertexCover**?



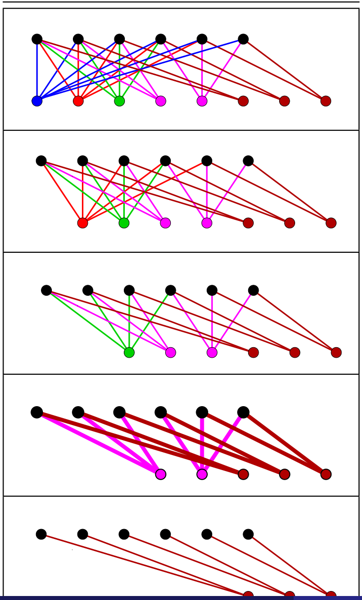
# How bad is **GreedyVertexCover**?



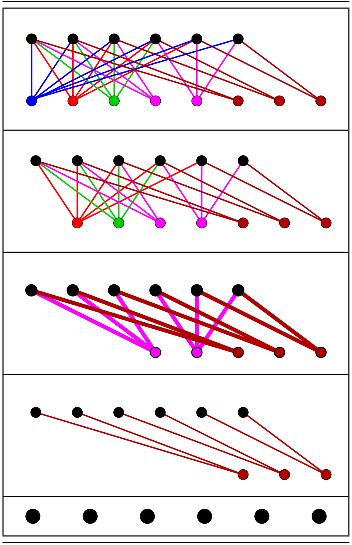
# How bad is **GreedyVertexCover**?



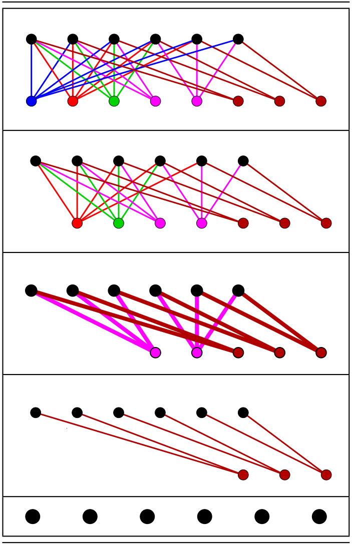
# How bad is **GreedyVertexCover**?



# How bad is **GreedyVertexCover**?

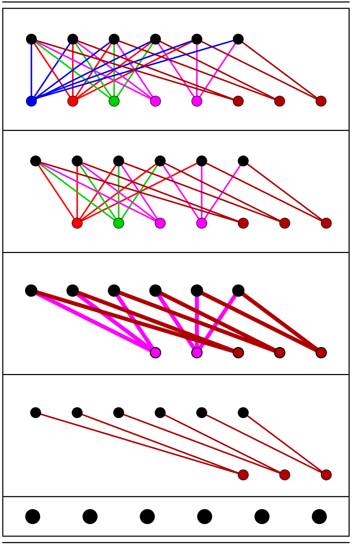


# How bad is **GreedyVertexCover**?



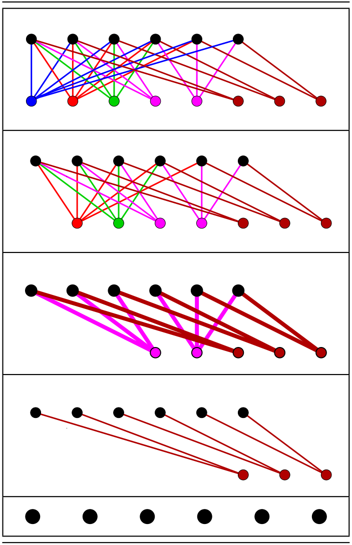
1. Bottom row taken by Greedy.

# How bad is GreedyVertexCover?



1. Bottom row taken by Greedy.
2. Top row was a smaller solution.

# How bad is GreedyVertexCover?



1. Bottom row taken by Greedy.
2. Top row was a smaller solution.

**Lemma**

*The algorithm* **GreedyVertexCover** *is* $\Omega(\log n)$ *approximation to the optimal solution to VertexCoverMin.*

See notes for details!

# Greedy Vertex Cover

## Theorem

*The greedy algorithm for **VertexCover** achieves* $\Theta(\log n)$ *approximation, where* $n$ *(resp.* $m$*) is the number of vertices (resp., edges) in the graph. Running time is* $O(mn^2)$.

## Proof

Lower bound follows from lemma.
Upper bound follows from analysis of greedy algorithm for **Set Cover**, which will be done shortly.
As for the running time, each iteration of the algorithm takes $O(mn)$ time, and there are at most $n$ iterations.

# Two for the price of one

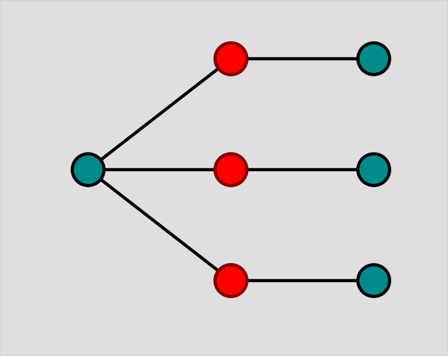
```
ApproxVertexCover(G):
    S ← ∅
    while E(G) ≠ ∅ do
        uv ← any edge of G
        S ← S ∪ {u, v}
        Remove u, v from V(G)
        Remove all edges involving u or v from E(G)

    return S
```

## Theorem

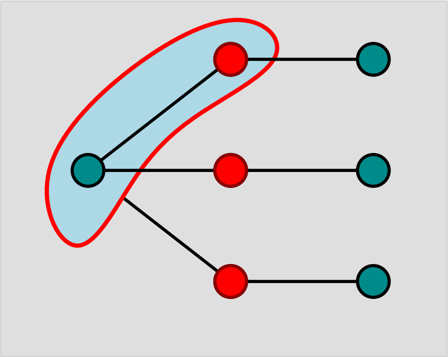
**ApproxVertexCover** *is a* $2$*-approximation algorithm for* *VertexCoverMin* *that runs in* $O(n^2)$ *time.*

**Proof...**

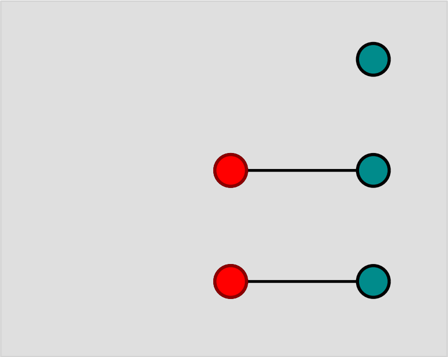
# Two for the price of one - example



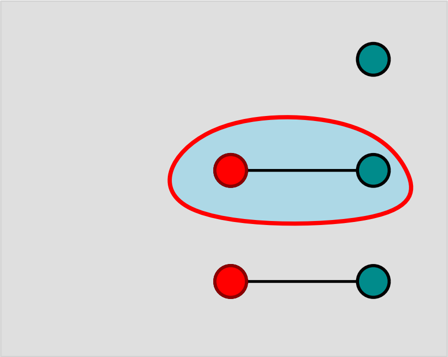
# Two for the price of one - example



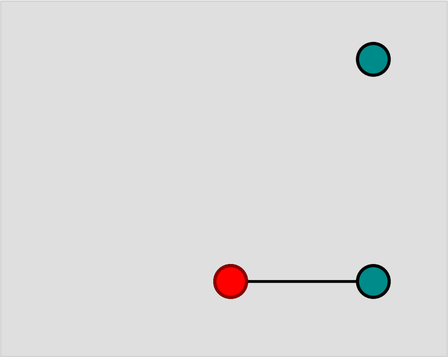
# Two for the price of one - example



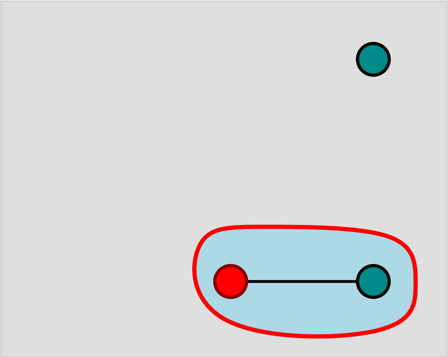
# Two for the price of one - example



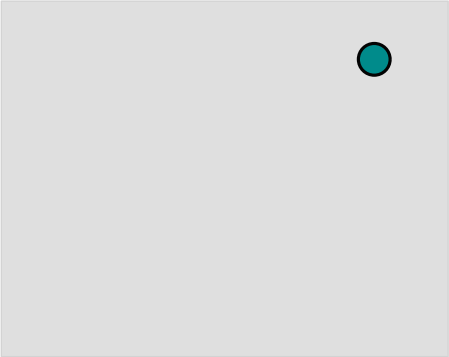
# Two for the price of one - example



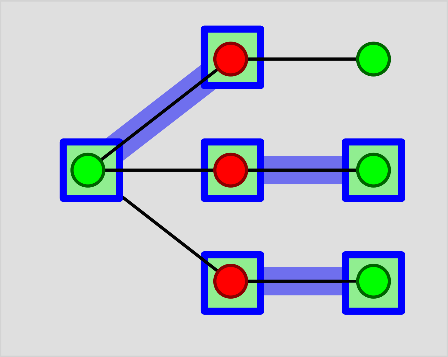
# Two for the price of one - example



# Two for the price of one - example



# Two for the price of one - example



# Part II

## Fixed parameter tractability, approximation, and fast exponential time algorithms (to say nothing of the dog)

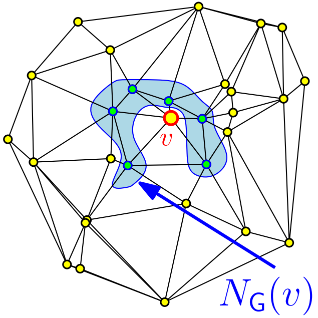
# What if the vertex cover is small?

1. $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ with $n$ vertices
2. $K \leftarrow$ Approximate VertexCoverMin up to a factor of two.
3. Any vertex cover of $G$ is of size $\geq K/2$.
4. Naively compute optimal in $O\left(n^{K+2}\right)$ time.

# Induced subgraph

## Definition

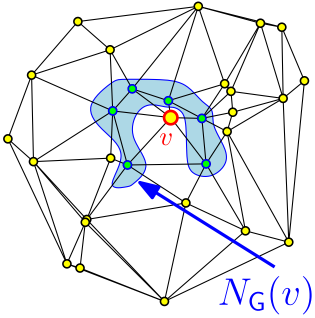
$N_G(v)$: ***Neighborhood*** of $v$
– set of vertices of **G** adjacent to $v$.



# Induced subgraph

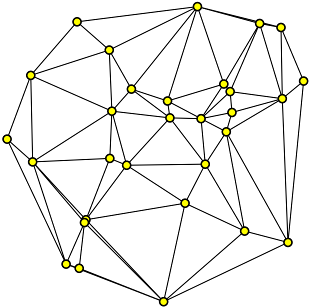
## Definition

$N_{\mathsf{G}}(v)$: ***Neighborhood*** of $v$ – set of vertices of $\mathsf{G}$ adjacent to $v$.




## Definition

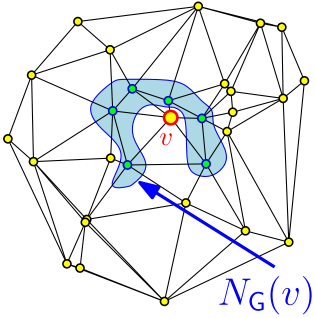
Let $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ be a graph. For a subset $S \subseteq \mathsf{V}$, let $\mathsf{G}_S$ be the ***induced subgraph*** over $S$.



# Induced subgraph

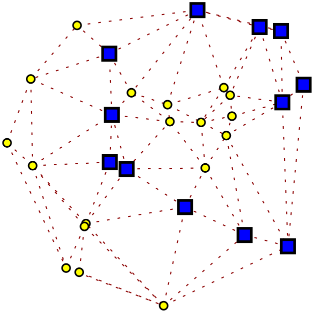
## Definition

$N_{\mathbf{G}}(v)$: ***Neighborhood*** of $v$ – set of vertices of $\mathbf{G}$ adjacent to $v$.




## Definition

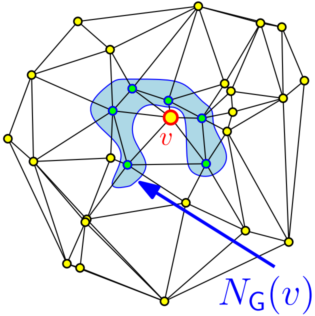
Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a graph. For a subset $S \subseteq \mathbf{V}$, let $\mathbf{G}_S$ be the ***induced subgraph*** over $S$.



# Induced subgraph

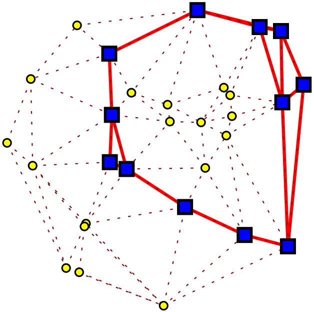
## Definition

$N_{\mathsf{G}}(v)$: ***Neighborhood*** of $v$ – set of vertices of $\mathsf{G}$ adjacent to $v$.




## Definition

Let $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ be a graph. For a subset $S \subseteq \mathsf{V}$, let $\mathsf{G}_S$ be the ***induced subgraph*** over $S$.



# Exact fixed parameter tractable algorithm

Fixed parameter tractable algorithm for **VertexCoverMin**.

Computes minimum vertex cover for the induced graph $\mathsf{G}_X$:

**fpVCI** $(X, \beta)$
    // $\beta$: size of VC computed so far.
    **if** $X = \emptyset$ or $\mathsf{G}_X$ has no edges **then return** $\beta$
    $\mathbf{e} \leftarrow$ any edge $uv$ of $\mathsf{G}_X$.
    $\beta_1 = \textsf{fpVCI}\left(X \setminus \{u, v\}, \beta + 2\right)$
    $\beta_2 = \textsf{fpVCI}\left(X \setminus \left(\{u\} \cup N_{\mathsf{G}_X}(v)\right), \beta + |N_{\mathsf{G}_X}(v)|\right)$
    $\beta_3 = \textsf{fpVCI}\left(X \setminus \left(\{v\} \cup N_{\mathsf{G}_X}(u)\right), \beta + |N_{\mathsf{G}_X}(u)|\right)$
    **return** $\min(\beta_1, \beta_2, \beta_3)$.

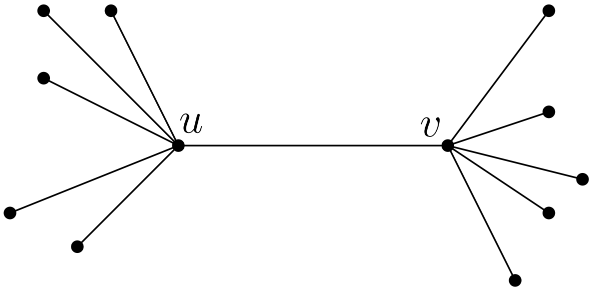
**algFPVertexCover** $(\mathsf{G} = (\mathsf{V}, \mathsf{E}))$
    **return** $\textsf{fpVCI}(\mathsf{V}, 0)$

# Depth of recursion

## Lemma

*The algorithm* **algFPVertexCover** *returns the optimal solution to the given instance of VertexCoverMin.*

**Proof...**



# Depth of recursion II

## Lemma

*The depth of the recursion of* **algFPVertexCover**($\mathbf{G}$) *is at most* $\alpha$*, where* $\alpha$ *is the minimum size vertex cover in* $\mathbf{G}$*.*

## Proof.

1. When the algorithm takes both $u$ and $v$ - one of them in opt. Can happen at most $\alpha$ times.
2. Algorithm picks $N_{\mathsf{G}_X}(v)$ (i.e., $\beta_2$). Conceptually add $v$ to the vertex cover being computed.
3. Do the same thing for the case of $\beta_3$.
4. Every such call add one element of the opt to conceptual set cover. Depth of recursion is $\leq \alpha$.

□

# Vertex Cover

## Exact fixed parameter tractable algorithm

### Theorem

**G**: *graph with* $n$ *vertices. Min vertex cover of size* $\alpha$. *Then,* **algFPVertexCover** *returns opt. vertex cover.*
*Running time is* $O(3^{\alpha} n^2)$.

### Proof:

1. By lemma, recursion tree has depth $\alpha$.
2. Rec-tree contains $\leq 2 \cdot 3^{\alpha}$ nodes.
3. Each node requires $O(n^2)$ work. ∎

Algorithms with running time $O(n^c f(\alpha))$, where $\alpha$ is some parameter that depends on the problem are ***fixed parameter tractable***.

# Vertex Cover

## Exact fixed parameter tractable algorithm

### Theorem

**G**: *graph with* $n$ *vertices. Min vertex cover of size* $\alpha$. *Then,* **algFPVertexCover** *returns opt. vertex cover.*
*Running time is* $O(3^\alpha n^2)$.

### Proof:

1. By lemma, recursion tree has depth $\alpha$.
2. Rec-tree contains $\leq 2 \cdot 3^\alpha$ nodes.
3. Each node requires $O(n^2)$ work. ■

Algorithms with running time $O(n^c f(\alpha))$, where $\alpha$ is some parameter that depends on the problem are ***fixed parameter tractable***.

# Part III

# Traveling Salesperson Problem

# TSP

## TSP-Min

**Instance**: $\mathbf{G} = (V, E)$ a complete graph, and $\omega(e)$ a cost function on edges of $\mathbf{G}$.
**Question:** The cheapest tour that visits all the vertices of $\mathbf{G}$ exactly once.

Solved exactly naively in $\approx n!$ time.
Using DP, solvable in $O(n^2 2^n)$ time.

# TSP

## TSP-Min

**Instance**: $\mathbf{G} = (V, E)$ a complete graph, and $\omega(e)$ a cost function on edges of $\mathbf{G}$.
**Question:** The cheapest tour that visits all the vertices of $\mathbf{G}$ exactly once.

Solved exactly naively in $\approx n!$ time.
Using DP, solvable in $O(n^2 2^n)$ time.

# TSP Hardness

## Theorem

*TSP-Min can not be approximated within **any** factor unless* $\mathbf{NP} = \mathbf{P}$.

## Proof.

1. Reduction from **Hamiltonian Cycle** into **TSP**.
2. $\mathbf{G} = (\mathbf{V}, \mathbf{E})$: instance of Hamiltonian cycle.
3. $\mathbf{H}$: Complete graph over $\mathbf{V}$.

$$\forall u, v \in \mathbf{V} \quad w_{\mathbf{H}}(uv) = \begin{cases} 1 & uv \in \mathbf{E} \\ 2 & \text{otherwise.} \end{cases}$$

4. $\exists$ tour of price $n$ in $\mathbf{H}$ $\iff$ $\exists$ Hamiltonian cycle in $\mathbf{G}$.
5. No Hamiltonian cycle $\implies$ TSP price at least $n+1$.
6. But... replace $2$ by $cn$, for $c$ an arbitrary number

# TSP Hardness

## Theorem

*TSP-Min can not be approximated within **any** factor unless* **NP** $=$ **P**.

## Proof.

1. Reduction from **Hamiltonian Cycle** into **TSP**.
2. $\mathbf{G} = (\mathbf{V}, \mathbf{E})$: instance of Hamiltonian cycle.
3. $\mathbf{H}$: Complete graph over $\mathbf{V}$.

$$\forall u, v \in \mathbf{V} \quad w_{\mathbf{H}}(uv) = \begin{cases} 1 & uv \in \mathbf{E} \\ 2 & \text{otherwise.} \end{cases}$$

4. $\exists$ tour of price $n$ in $\mathbf{H}$ $\iff$ $\exists$ Hamiltonian cycle in $\mathbf{G}$.
5. No Hamiltonian cycle $\implies$ TSP price at least $n+1$.
6. But... replace $2$ by $cn$, for $c$ an arbitrary number

# TSP Hardness

## Theorem

*TSP-Min can not be approximated within **any** factor unless* **NP** $=$ **P**.

## Proof.

1. Reduction from **Hamiltonian Cycle** into **TSP**.
2. $\mathsf{G} = (\mathsf{V}, \mathsf{E})$: instance of Hamiltonian cycle.
3. $\mathsf{H}$: Complete graph over $\mathsf{V}$.

$$\forall u, v \in \mathsf{V} \quad w_{\mathsf{H}}(uv) = \begin{cases} 1 & uv \in \mathsf{E} \\ 2 & \text{otherwise.} \end{cases}$$

4. $\exists$ tour of price $n$ in $\mathsf{H}$ $\iff$ $\exists$ Hamiltonian cycle in $\mathsf{G}$.
5. No Hamiltonian cycle $\implies$ TSP price at least $n + 1$.
6. But... replace $2$ by $cn$, for $c$ an arbitrary number

# TSP Hardness

## Theorem

*TSP-Min can not be approximated within **any** factor unless* **NP** $=$ **P**.

## Proof.

1. Reduction from **Hamiltonian Cycle** into **TSP**.
2. $\mathsf{G} = (\mathsf{V}, \mathsf{E})$: instance of Hamiltonian cycle.
3. $\mathsf{H}$: Complete graph over $\mathsf{V}$.
$$\forall u, v \in \mathsf{V} \quad w_{\mathsf{H}}(uv) = \begin{cases} 1 & uv \in \mathsf{E} \\ 2 & \text{otherwise.} \end{cases}$$
4. $\exists$ tour of price $n$ in $\mathsf{H}$ $\iff$ $\exists$ Hamiltonian cycle in $\mathsf{G}$.
5. No Hamiltonian cycle $\implies$ TSP price at least $n + 1$.
6. But... replace $2$ by $cn$, for $c$ an arbitrary number

# TSP Hardness - proof continued

## Proof.

1. Price of all tours are either:
   (i) $n$ (only if $\exists$ Hamiltonian cycle in **G**),
   (ii) larger than $cn + 1$ (actually, $\geq cn + (n - 1)$).
2. Suppose you had a poly time $c$-approximation to TSP-Min.
3. Run it on **H**:
   (i) If returned value $\geq cn + 1 \implies$ no Ham Cycle since $(cn + 1)/c > n$
   (ii) If returned value $\leq cn \implies$ Ham Cycle since $OPT \leq cn < cn + 1$
4. $c$-approximation algorithm to $\mathrm{TSP} \implies$ poly-time algorithm for **NP-Complete** problem. Possible only if **P** $=$ **NP**.

□

# TSP Hardness - proof continued

## Proof.

1. Price of all tours are either:
   (i) $n$ (only if $\exists$ Hamiltonian cycle in $\mathbf{G}$),
   (ii) larger than $cn + 1$ (actually, $\geq cn + (n - 1)$).
2. Suppose you had a poly time $c$-approximation to TSP-Min.
3. Run it on $\mathbf{H}$:
   (i) If returned value $\geq cn + 1 \implies$ no Ham Cycle since $(cn + 1)/c > n$
   (ii) If returned value $\leq cn \implies$ Ham Cycle since $OPT \leq cn < cn + 1$
4. $c$-approximation algorithm to TSP $\implies$ poly-time algorithm for **NP-Complete** problem. Possible only if $\mathbf{P} = \mathbf{NP}$.

# TSP Hardness - proof continued

## Proof.

1. Price of all tours are either:
   (i) $n$ (only if $\exists$ Hamiltonian cycle in **G**),
   (ii) larger than $cn + 1$ (actually, $\geq cn + (n - 1)$).
2. Suppose you had a poly time $c$-approximation to TSP-Min.
3. Run it on **H**:
   (i) If returned value $\geq cn + 1 \implies$ no Ham Cycle since $(cn + 1)/c > n$
   (ii) If returned value $\leq cn \implies$ Ham Cycle since $OPT \leq cn < cn + 1$
4. $c$-approximation algorithm to $\mathrm{TSP} \implies$ poly-time algorithm for **NP-Complete** problem. Possible only if **P** $=$ **NP**.

# TSP Hardness - proof continued

## Proof.

1. Price of all tours are either:
   (i) $n$ (only if $\exists$ Hamiltonian cycle in **G**),
   (ii) larger than $cn + 1$ (actually, $\geq cn + (n - 1)$).
2. Suppose you had a poly time $c$-approximation to TSP-Min.
3. Run it on **H**:
   (i) If returned value $\geq cn + 1 \implies$ no Ham Cycle since $(cn + 1)/c > n$
   (ii) If returned value $\leq cn \implies$ Ham Cycle since $OPT \leq cn < cn + 1$
4. $c$-approximation algorithm to $\mathrm{TSP} \implies$ poly-time algorithm for **NP-Complete** problem. Possible only if **P** $=$ **NP**.

□

# TSP with the triangle inequality

Because it is not that bad after all.

## $\text{TSP}_{\triangle\neq}$-Min

**Instance**: $\mathbf{G} = (V, E)$ is a complete graph. There is also a cost function $\omega(\cdot)$ defined over the edges of $\mathbf{G}$, that complies with the triangle inequality.
**Question:** The cheapest tour that visits all the vertices of $\mathbf{G}$ exactly once.

***triangle inequality***: $\omega(\cdot)$ if

$$\forall u, v, w \in \mathsf{V}(\mathbf{G}), \qquad \omega(u, v) \leq \omega(u, w) + \omega(w, v).$$

### Shortcutting

$\sigma$: a path from $s$ to $t$ in $\mathbf{G}$ $\implies$ $\omega(st) \leq \omega(\sigma)$.

# TSP with the triangle inequality

Because it is not that bad after all.

## $\mathrm{TSP}_{\triangle\neq}$-Min

**Instance**: $\mathbf{G} = (V, E)$ is a complete graph. There is also a cost function $\omega(\cdot)$ defined over the edges of $\mathbf{G}$, that complies with the triangle inequality.
**Question:** The cheapest tour that visits all the vertices of $\mathbf{G}$ exactly once.

***triangle inequality***: $\omega(\cdot)$ if

$$\forall u, v, w \in \mathsf{V}(\mathbf{G}), \qquad \omega(u, v) \leq \omega(u, w) + \omega(w, v).$$

### Shortcutting

$\sigma$: a path from $s$ to $t$ in $\mathbf{G}$ $\implies$ $\omega(st) \leq \omega(\sigma)$.

# TSP with the triangle inequality

Because it is not that bad after all.

## $\mathsf{TSP}_{\triangle\neq}$-Min

**Instance**: $\mathbf{G} = (V, E)$ is a complete graph. There is also a cost function $\omega(\cdot)$ defined over the edges of $\mathbf{G}$, that complies with the triangle inequality.
**Question:** The cheapest tour that visits all the vertices of $\mathbf{G}$ exactly once.

***triangle inequality***: $\omega(\cdot)$ if

$$\forall u, v, w \in \mathsf{V}(\mathbf{G}), \qquad \omega(u, v) \leq \omega(u, w) + \omega(w, v).$$

### Shortcutting

$\sigma$: a path from $s$ to $t$ in $\mathbf{G}$ $\implies$ $\omega(st) \leq \omega(\sigma)$.

# **TSP** with the triangle inequality

Continued...

## Definition

Cycle in **G** is ***Eulerian*** if it visits every **edge** of **G** exactly once.

Assume you already seen the following:

## Lemma

*A graph* **G** *has a cycle that visits every edge of* **G** *exactly once (i.e., an Eulerian cycle) if and only if* **G** *is connected, and all the vertices have even degree. Such a cycle can be computed in* $O(n + m)$ *time, where* $n$ *and* $m$ *are the number of vertices and edges of* **G***, respectively.*

# TSP with the triangle inequality

Continued...

1. $C_{\text{opt}}$ optimal **TSP** tour in **G**.
2. **Observation**:
   $\omega(C_{\text{opt}}) \geq \text{weight}\Big(\text{cheapest spanning graph of } \mathbf{G}\Big)$.
3. MST: cheapest spanning graph of **G**.
   $\omega(C_{\text{opt}}) \geq \omega(\text{MST}(\mathbf{G}))$
4. $O(n \log n + m) = O(n^2)$: time to compute MST.
   $n = |\mathsf{V}(\mathbf{G})|$, $m = \binom{n}{2}$.

# TSP with the triangle inequality

Continued...

1. $C_{\text{opt}}$ optimal **TSP** tour in **G**.
2. **Observation**:
   $\omega(C_{\text{opt}}) \geq \textbf{weight}\Big(\text{cheapest spanning graph of } \mathbf{G}\Big)$.
3. MST: cheapest spanning graph of **G**.
   $\omega(C_{\text{opt}}) \geq \omega(\text{MST}(\mathbf{G}))$
4. $O(n \log n + m) = O(n^2)$: time to compute MST.
   $n = |\mathbf{V}(\mathbf{G})|$, $m = \binom{n}{2}$.

# TSP with the triangle inequality

## Continued...

1. $C_{\text{opt}}$ optimal **TSP** tour in **G**.
2. **Observation**:
   $\omega(C_{\text{opt}}) \geq \text{weight}\Big(\text{cheapest spanning graph of } \mathbf{G}\Big)$.
3. $\text{MST}$: cheapest spanning graph of **G**.
   $\omega(C_{\text{opt}}) \geq \omega(\text{MST}(\mathbf{G}))$
4. $O(n \log n + m) = O(n^2)$: time to compute $\text{MST}$.
   $n = |\mathsf{V}(\mathbf{G})|$, $m = \binom{n}{2}$.

# TSP with the triangle inequality

Continued...

1. $C_{\text{opt}}$ optimal **TSP** tour in **G**.
2. **Observation**:
   $\omega(C_{\text{opt}}) \geq \text{weight}\Big(\text{cheapest spanning graph of } \mathbf{G}\Big)$.
3. $\mathrm{MST}$: cheapest spanning graph of **G**.
   $\omega(C_{\text{opt}}) \geq \omega(\mathrm{MST}(\mathbf{G}))$
4. $O(n \log n + m) = O(n^2)$: time to compute $\mathrm{MST}$.
   $n = |\mathbf{V}(\mathbf{G})|$, $m = \binom{n}{2}$.

# TSP with the triangle inequality

## 2-approximation

1. $T \leftarrow \mathrm{MST}(\mathbf{G})$
2. $\mathbf{H} \leftarrow$ duplicate very edge of $T$.
3. $H$ has an Eulerian tour.
4. $\mathbf{C}$: Eulerian cycle in $H$.
5. $\omega(\mathbf{C}) = \omega(H) = 2\omega(T) = 2\omega(MST(\mathbf{G})) \leq 2\omega(C_{\mathrm{opt}})$.
6. $\pi$: Shortcut $\mathbf{C}$ so visit every vertex once.
7. $\omega(\pi) \leq \omega(\mathbf{C})$

# TSP with the triangle inequality

## 2-approximation

1. $T \leftarrow \mathrm{MST}(\mathbf{G})$
2. $\mathbf{H} \leftarrow$ duplicate very edge of $T$.
3. $H$ has an Eulerian tour.
4. $\mathbf{C}$: Eulerian cycle in $H$.
5. $\omega(\mathbf{C}) = \omega(H) = 2\omega(T) = 2\omega(MST(\mathbf{G})) \leq 2\omega(C_{\mathrm{opt}})$.
6. $\pi$: Shortcut $\mathbf{C}$ so visit every vertex once.
7. $\omega(\pi) \leq \omega(\mathbf{C})$

# TSP with the triangle inequality

## $2$-approximation

1. $T \leftarrow \mathrm{MST}(\mathbf{G})$
2. **H** $\leftarrow$ duplicate very edge of $T$.
3. $H$ has an Eulerian tour.
4. **C**: Eulerian cycle in $H$.
5. $\omega(\mathbf{C}) = \omega(H) = 2\omega(T) = 2\omega(MST(\mathbf{G})) \leq 2\omega(C_{\mathrm{opt}})$.
6. $\pi$: Shortcut **C** so visit every vertex once.
7. $\omega(\pi) \leq \omega(\mathbf{C})$

# TSP with the triangle inequality

## $2$-approximation

1. $T \leftarrow \mathrm{MST}(\mathbf{G})$
2. $\mathbf{H} \leftarrow$ duplicate very edge of $T$.
3. $H$ has an Eulerian tour.
4. $\mathbf{C}$: Eulerian cycle in $H$.
5. $\omega(\mathbf{C}) = \omega(H) = 2\omega(T) = 2\omega(MST(\mathbf{G})) \leq 2\omega(C_{\text{opt}})$.
6. $\pi$: Shortcut $\mathbf{C}$ so visit every vertex once.
7. $\omega(\pi) \leq \omega(\mathbf{C})$

# TSP with the triangle inequality

## $2$-approximation

1. $T \leftarrow \mathrm{MST}(\mathsf{G})$
2. $\mathsf{H} \leftarrow$ duplicate very edge of $T$.
3. $H$ has an Eulerian tour.
4. $\mathsf{C}$: Eulerian cycle in $H$.
5. $\omega(\mathsf{C}) = \omega(H) = 2\omega(T) = 2\omega(MST(\mathsf{G})) \leq 2\omega(C_{\mathrm{opt}})$.
6. $\pi$: Shortcut $\mathsf{C}$ so visit every vertex once.
7. $\omega(\pi) \leq \omega(\mathsf{C})$

# TSP with the triangle inequality

## $2$-approximation

1. $T \leftarrow \mathrm{MST}(\mathbf{G})$
2. $\mathbf{H} \leftarrow$ duplicate very edge of $T$.
3. $H$ has an Eulerian tour.
4. $\mathbf{C}$: Eulerian cycle in $H$.
5. $\omega(\mathbf{C}) = \omega(H) = 2\omega(T) = 2\omega(MST(\mathbf{G})) \leq 2\omega(C_{\mathrm{opt}})$.
6. $\pi$: Shortcut $\mathbf{C}$ so visit every vertex once.
7. $\omega(\pi) \leq \omega(\mathbf{C})$

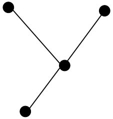
# TSP with the triangle inequality

## $2$-approximation

1. $T \leftarrow \mathrm{MST}(\mathbf{G})$
2. $\mathbf{H} \leftarrow$ duplicate very edge of $T$.
3. $H$ has an Eulerian tour.
4. $\mathbf{C}$: Eulerian cycle in $H$.
5. $\omega(\mathbf{C}) = \omega(H) = 2\omega(T) = 2\omega(MST(\mathbf{G})) \leq 2\omega(C_{\mathrm{opt}})$.
6. $\pi$: Shortcut $\mathbf{C}$ so visit every vertex once.
7. $\omega(\pi) \leq \omega(\mathbf{C})$

# TSP with the triangle inequality

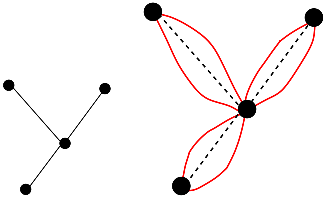
## 2-approximation algorithm in figures



(a) (b) (c) (d)

# TSP with the triangle inequality

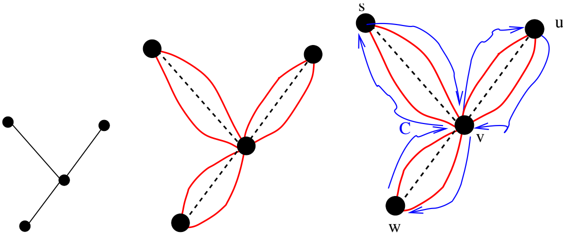
## 2-approximation algorithm in figures



(a) (b) (c) (d)

# TSP with the triangle inequality

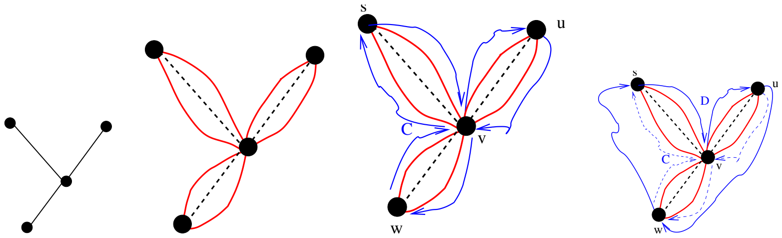
## 2-approximation algorithm in figures




(a) (b) (c) (d)

# TSP with the triangle inequality

## 2-approximation algorithm in figures




(a) (b) (c) (d)

Euler Tour: **vuvwvsv**

First occurrences: **vuvwvsv**

Shortcut String: **vuwsv**

# TSP with the triangle inequality

## $\mathbf{2}$-approximation - result

### Theorem

$\mathbf{G}$: *Instance of TSP$_{\triangle\neq}$-Min.*
$C_{\mathrm{opt}}$: *min cost TSP tour of* $\mathbf{G}$.
$\implies$ *Compute a tour of* $\mathbf{G}$ *of length* $\leq 2\omega(C_{\mathrm{opt}})$.
*Running time of the algorithm is* $O(n^2)$.

$\mathbf{G}$: $n$ vertices, cost function $\omega(\cdot)$ on the edges that comply with the triangle inequality.

# TSP with the triangle inequality

## $2$-approximation - result

**Theorem**

**G**: *Instance of TSP$_{\triangle\neq}$-Min.*
$C_{\mathrm{opt}}$: *min cost TSP tour of* **G**.
$\implies$ *Compute a tour of* **G** *of length* $\leq 2\omega(C_{\mathrm{opt}})$.
*Running time of the algorithm is* $O(n^2)$.

**G**: $n$ vertices, cost function $\omega(\cdot)$ on the edges that comply with the triangle inequality.

# TSP with the triangle inequality

## $2$-approximation - result

### Theorem

**G**: *Instance of* *$TSP_{\triangle\neq}$-Min*.
$C_{\mathrm{opt}}$: *min cost TSP tour of* **G**.
$\implies$ *Compute a tour of* **G** *of length* $\leq 2\omega(C_{\mathrm{opt}})$.
*Running time of the algorithm is* $O(n^2)$.

**G**: $n$ vertices, cost function $\omega(\cdot)$ on the edges that comply with the triangle inequality.

# TSP with the triangle inequality

## 2-approximation - result

**Theorem**

*$\mathbf{G}$: Instance of TSP$_{\triangle\neq}$-Min.*
*$C_{\mathrm{opt}}$: min cost TSP tour of $\mathbf{G}$.*
*$\implies$ Compute a tour of $\mathbf{G}$ of length $\leq 2\omega(C_{\mathrm{opt}})$.*
*Running time of the algorithm is $O(n^2)$.*

$\mathbf{G}$: $n$ vertices, cost function $\omega(\cdot)$ on the edges that comply with the triangle inequality.

## Definition

$\mathbf{G} = (V, E)$, a subset $M \subseteq E$ is a ***matching*** if no pair of edges of $M$ share endpoints.

A ***perfect matching*** is a matching that covers all the vertices of $\mathbf{G}$.

$w$: weight function on the edges. ***Min-weight perfect matching***, is the minimum weight matching among all perfect matching, where

$$\omega(M) = \sum_{e \in M} \omega(e) .$$

# **TSP** with the triangle inequality

## $\mathbf{3/2}$-approximation

The following is known:

**Theorem**

*Given a graph* **G** *and weights on the edges, one can compute the min-weight perfect matching of* **G** *in polynomial time.*

# Min weight perfect matching vs. TSP

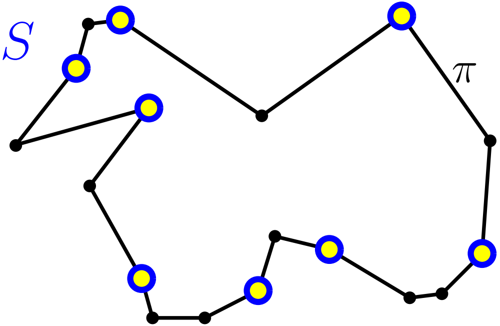
## Lemma

$\mathbf{G} = (\mathbf{V}, \mathbf{E})$: *complete graph.*
$S \subseteq \mathbf{V}$: *even size.*
$\omega(\cdot)$: *a weight function over* $\mathbf{E}$.
$\implies$ *min-weight perfect matching in* $\mathbf{G}_S$ *is* $\leq \omega(\mathrm{TSP}(\mathbf{G}))/2$.

# Min weight perfect matching vs. TSP

## Lemma

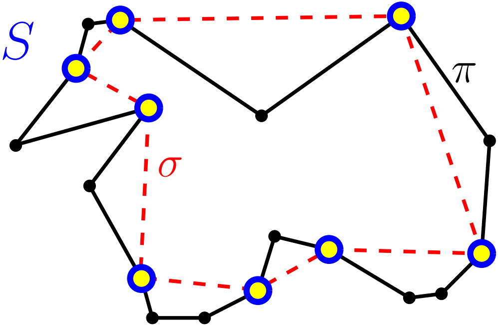
$\mathbf{G} = (\mathbf{V}, \mathbf{E})$: *complete graph.*
$S \subseteq \mathbf{V}$: *even size.*
$\omega(\cdot)$: *a weight function over* $\mathbf{E}$.
$\implies$ *min-weight perfect matching in* $\mathbf{G}_S$ *is* $\leq \omega(\mathrm{TSP}(\mathbf{G}))/2$.



# Min weight perfect matching vs. TSP

## Lemma

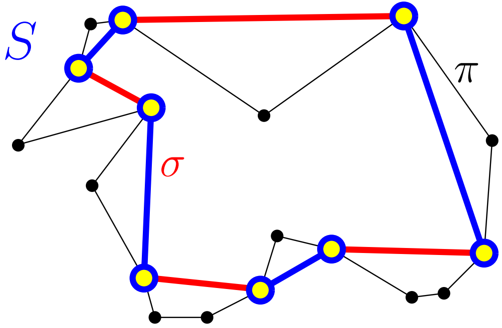
$\mathbf{G} = (\mathbf{V}, \mathbf{E})$: *complete graph.*
$S \subseteq \mathbf{V}$: *even size.*
$\omega(\cdot)$: *a weight function over* $\mathbf{E}$.
$\implies$ *min-weight perfect matching in* $\mathbf{G}_S$ *is* $\leq \omega(\mathrm{TSP}(\mathbf{G}))/2$.



# Min weight perfect matching vs. TSP

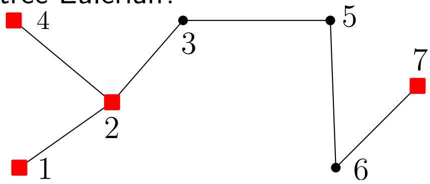
## Lemma

$\mathbf{G} = (\mathbf{V}, \mathbf{E})$: *complete graph.*
$S \subseteq \mathbf{V}$: *even size.*
$\omega(\cdot)$: *a weight function over* $\mathbf{E}$.
$\implies$ *min-weight perfect matching in* $\mathbf{G}_S$ *is* $\leq \omega(\mathbf{TSP}(\mathbf{G}))/2$.



# A more perfect tree?

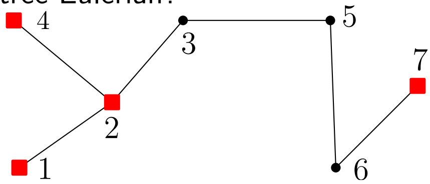
1. How to make the tree Eulerian?




2. Pesky odd degree vertices must die!
3. Number of odd degree vertices in a graph is even!
4. Compute min-weight matching on odd vertices, and add to MST.
5. $\mathsf{H} = \mathrm{MST} + \mathbf{min-weight-matching}$ is Eulerian.
6. Weight of resulting cycle in $\mathsf{H} \leq (3/2)\omega(\mathrm{TSP})$.

# A more perfect tree?

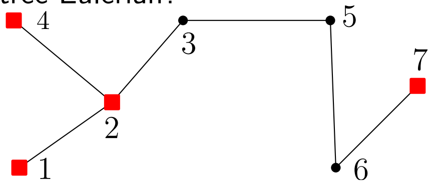
1. How to make the tree Eulerian?




2. Pesky odd degree vertices must die!
3. Number of odd degree vertices in a graph is even!
4. Compute min-weight matching on odd vertices, and add to MST.
5. $\mathbf{H} = \mathrm{MST} + \mathbf{min-weight-matching}$ is Eulerian.
6. Weight of resulting cycle in $\mathbf{H} \leq (3/2)\omega(\mathrm{TSP})$.

# A more perfect tree?

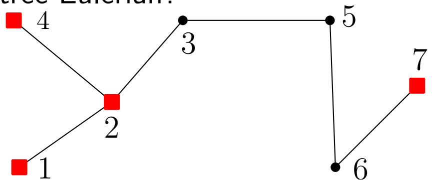
1. How to make the tree Eulerian?




2. Pesky odd degree vertices must die!
3. Number of odd degree vertices in a graph is even!
4. Compute min-weight matching on odd vertices, and add to MST.
5. $\mathbf{H} = \mathrm{MST} + \mathbf{min-weight-matching}$ is Eulerian.
6. Weight of resulting cycle in $\mathbf{H} \leq (3/2)\omega(\mathrm{TSP})$.

# A more perfect tree?

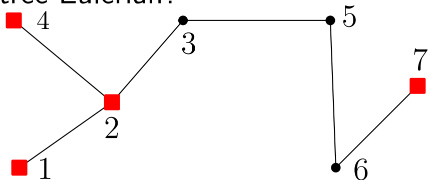
1. How to make the tree Eulerian?




2. Pesky odd degree vertices must die!
3. Number of odd degree vertices in a graph is even!
4. Compute min-weight matching on odd vertices, and add to $\mathrm{MST}$.
5. $\mathbf{H} = \mathrm{MST} + \mathbf{min-weight-matching}$ is Eulerian.
6. Weight of resulting cycle in $\mathbf{H} \leq (3/2)\omega(\mathrm{TSP})$.

# A more perfect tree?

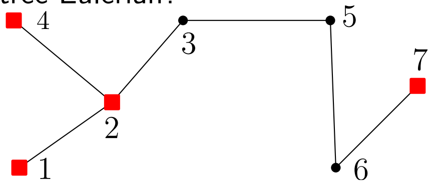
1. How to make the tree Eulerian?




2. Pesky odd degree vertices must die!
3. Number of odd degree vertices in a graph is even!
4. Compute min-weight matching on odd vertices, and add to $\mathrm{MST}$.
5. $\mathbf{H} = \mathrm{MST} + \mathbf{min} - \mathbf{weight} - \mathbf{matching}$ is Eulerian.
6. Weight of resulting cycle in $\mathbf{H} \leq (3/2)\omega(\mathrm{TSP})$.

# A more perfect tree?

1. How to make the tree Eulerian?




2. Pesky odd degree vertices must die!
3. Number of odd degree vertices in a graph is even!
4. Compute min-weight matching on odd vertices, and add to $\mathrm{MST}$.
5. $\mathbf{H} = \mathrm{MST} + \mathbf{min-weight-matching}$ is Eulerian.
6. Weight of resulting cycle in $\mathbf{H} \leq (3/2)\omega(\mathrm{TSP})$.

# Even number of odd degree vertices

## Lemma

*The number of odd degree vertices in any graph $G'$ is even.*

## Proof:

$\mu = \sum_{v \in V(G')} d(v) = 2|E(G')|$ and thus even.

$U = \sum_{v \in V(G'), d(v) \text{ is even}} d(v)$ even too.

Thus,

$$\alpha = \sum_{v \in V, d(v) \text{ is odd}} d(v) = \mu - U = \text{even number},$$

since $\mu$ and $U$ are both even.

Number of elements in sum of all odd numbers must be even, since the total sum is even.

# Even number of odd degree vertices

## Lemma

*The number of odd degree vertices in any graph $G'$ is even.*

## Proof:

$\mu = \sum_{v \in V(G')} d(v) = 2|E(G')|$ and thus even.
$U = \sum_{v \in V(G'), d(v) \text{ is even}} d(v)$ even too.
Thus,

$$\alpha = \sum_{v \in V, d(v) \text{ is odd}} d(v) = \mu - U = \text{even number},$$

since $\mu$ and $U$ are both even.
Number of elements in sum of all odd numbers must be even, since the total sum is even.

# Even number of odd degree vertices

## Lemma

*The number of odd degree vertices in any graph $G'$ is even.*

## Proof:

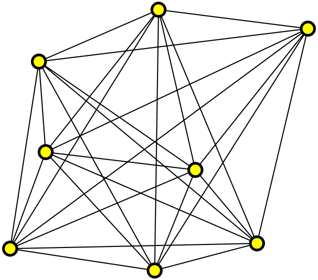
$\mu = \sum_{v \in V(G')} d(v) = 2|E(G')|$ and thus even.
$U = \sum_{v \in V(G'), d(v) \text{ is even}} d(v)$ even too.
Thus,

$$\alpha = \sum_{v \in V, d(v) \text{ is odd}} d(v) = \mu - U = \text{even number},$$

since $\mu$ and $U$ are both even.
Number of elements in sum of all odd numbers must be even, since the total sum is even.

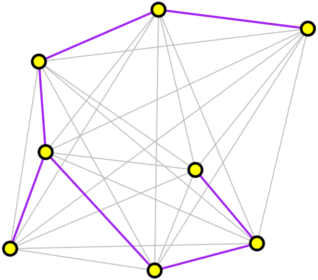
# $3/2$-approximation algorithm for TSP

Animated!



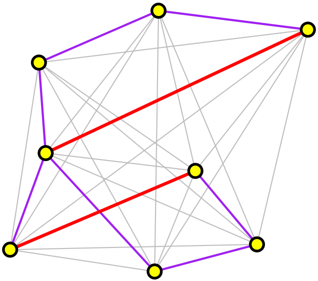
# $\mathbf{3/2}$-approximation algorithm for TSP

Animated!



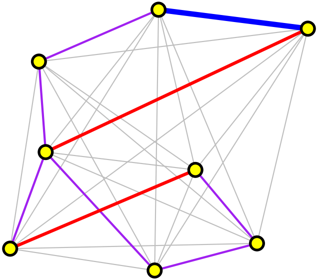
# $3/2$-approximation algorithm for TSP

## Animated!



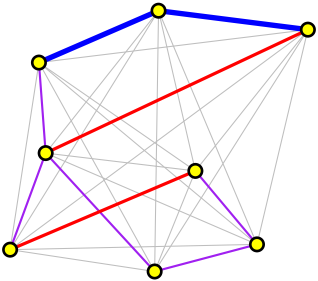
# $3/2$-approximation algorithm for TSP

## Animated!



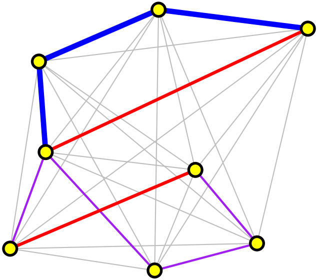
# $3/2$-approximation algorithm for TSP

Animated!



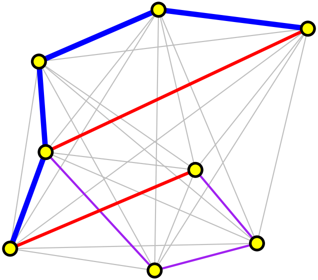
# $3/2$-approximation algorithm for TSP

## Animated!



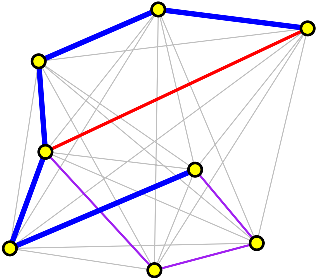
# $3/2$-approximation algorithm for TSP

Animated!



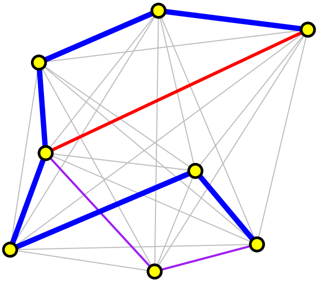
# $\mathbf{3/2}$-approximation algorithm for TSP

Animated!



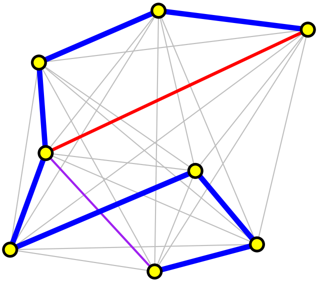
# $\mathbf{3/2}$-approximation algorithm for TSP

## Animated!



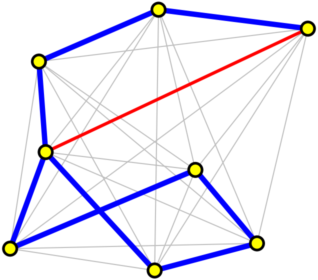
# $3/2$-approximation algorithm for TSP

Animated!

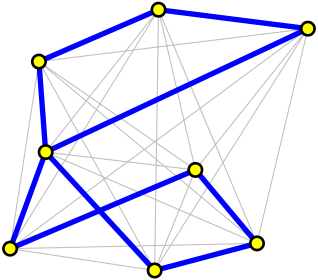


# $3/2$-approximation algorithm for TSP

## Animated!

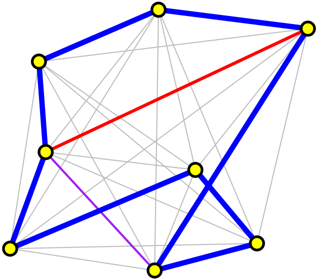


# $\mathbf{3/2}$-approximation algorithm for TSP
Animated!



# $\mathbf{3/2}$-approximation algorithm for TSP

Animated!



# $3/2$-approximation algorithm for TSP

## The result

**Theorem**

*Given an instance of TSP with the triangle inequality, one can compute in polynomial time, a $(3/2)$-approximation to the optimal TSP.*

# Biographical Notes

The $\mathbf{3/2}$-approximation for TSP with the triangle inequality is due to **?**.

# Notes

# Notes

# Notes

# Notes