

Approximation Algorithms

Lecture 8

September 17, 2015

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.

8.1: Greedy algorithms and approximation algorithms – Vertex Cover

Greedy algorithms

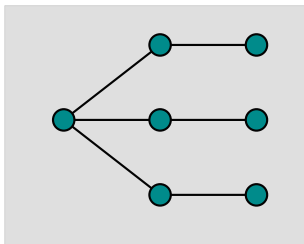
- 1 **greedy algorithms:** do locally the right thing...
- 2 ...and they suck.

VertexCoverMin

Instance: A graph G .

Question: Return the **smallest** subset $S \subseteq V(G)$, s.t. S touches all the edges of G .

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



Greedy algorithms

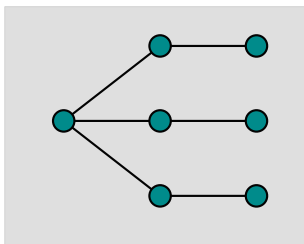
- 1 **greedy algorithms:** do locally the right thing...
- 2 ...and they suck.

VertexCoverMin

Instance: A graph G .

Question: Return the **smallest** subset $S \subseteq V(G)$, s.t. S touches all the edges of G .

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



Greedy algorithms

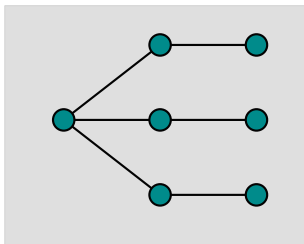
- 1 **greedy algorithms:** do locally the right thing...
- 2 ...and they suck.

VertexCoverMin

Instance: A graph G .

Question: Return the **smallest** subset $S \subseteq V(G)$, s.t. S touches all the edges of G .

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



Greedy algorithms

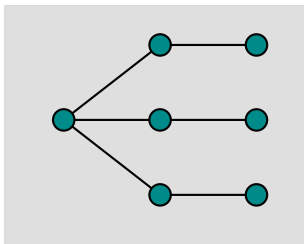
- 1 **greedy algorithms:** do locally the right thing...
- 2 ...and they suck.

VertexCoverMin

Instance: A graph G .

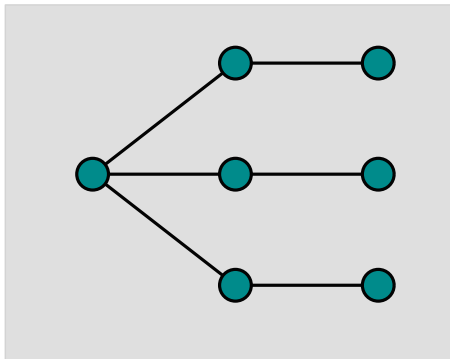
Question: Return the **smallest** subset $S \subseteq V(G)$, s.t. S touches all the edges of G .

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



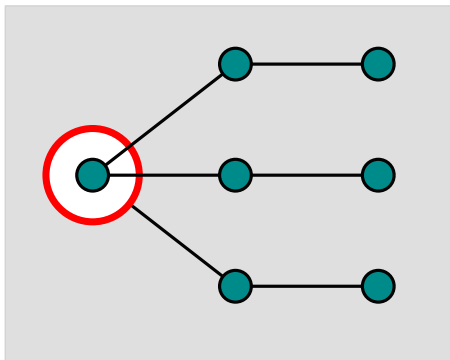
Greedy algorithms

GreedyVertexCover in action...



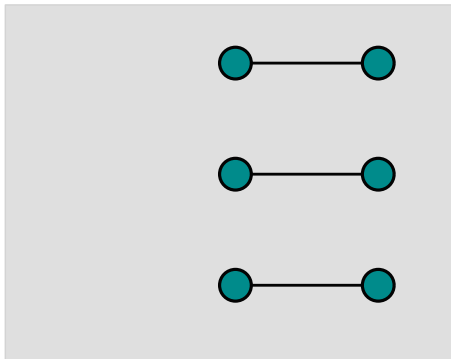
Greedy algorithms

GreedyVertexCover in action...



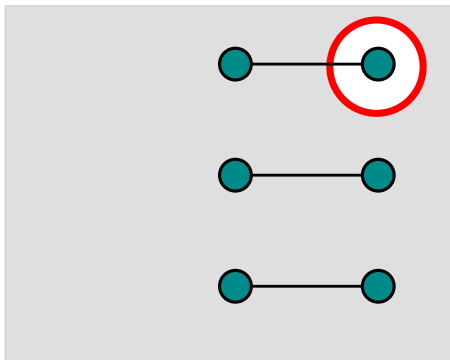
Greedy algorithms

Greedy **VertexCover** in action...



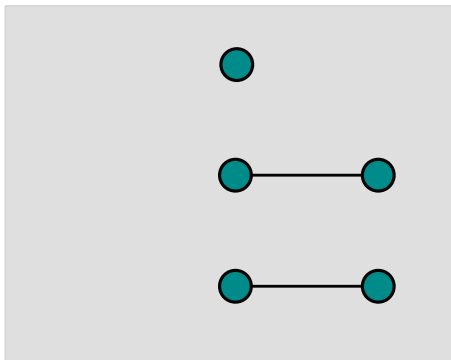
Greedy algorithms

Greedy **VertexCover** in action...



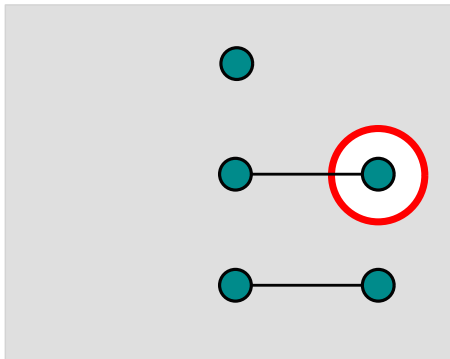
Greedy algorithms

Greedy **VertexCover** in action...



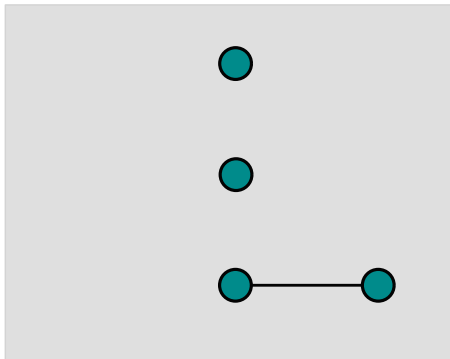
Greedy algorithms

GreedyVertexCover in action...



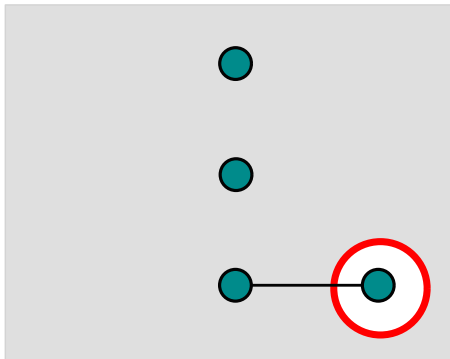
Greedy algorithms

GreedyVertexCover in action...



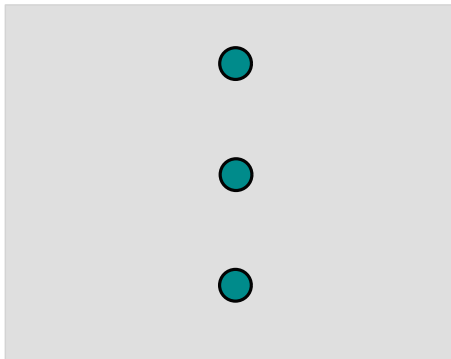
Greedy algorithms

GreedyVertexCover in action...



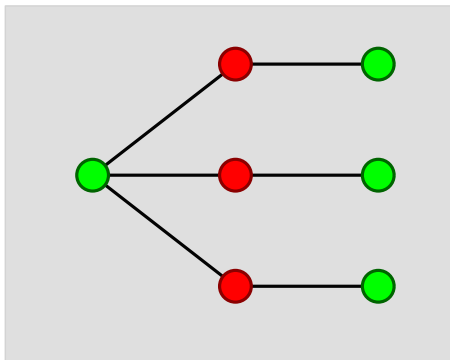
Greedy algorithms

Greedy **VertexCover** in action...



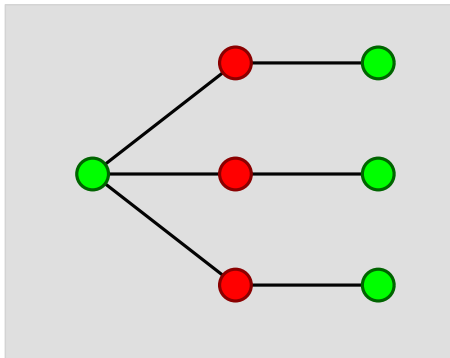
Greedy algorithms

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

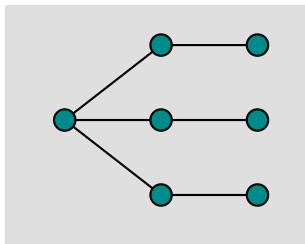
Definition

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

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

Back to GreedyVertexCover

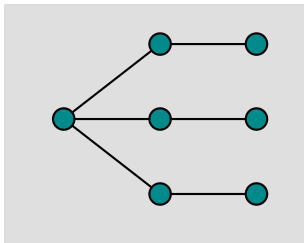
- 1 **GreedyVertexCover:**
pick vertex with highest degree, remove, repeat.



- 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

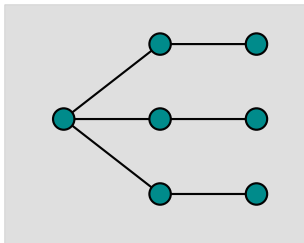
- 1 **GreedyVertexCover:**
pick vertex with highest degree, remove, repeat.
- 2 Returns 4, but opt is 3!



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

Back to GreedyVertexCover

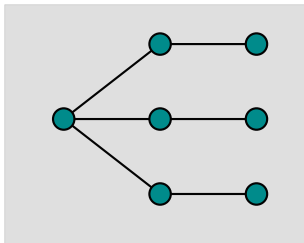
- 1 **GreedyVertexCover:**
pick vertex with highest degree, remove, repeat.
- 2 Returns 4, but opt is 3!



- 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.

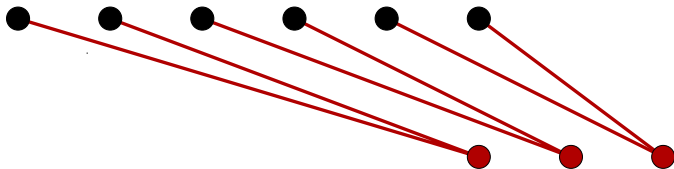
Let the top partite set be of size n .



How bad is GreedyVertexCover?

Build a bipartite graph.

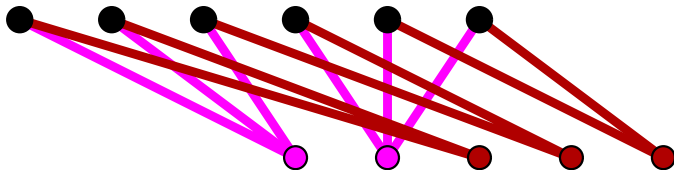
In the bottom set add $\lfloor n/2 \rfloor$ vertices of degree 2, such that each edge goes to a different vertex above.



How bad is GreedyVertexCover?

Build a bipartite graph.

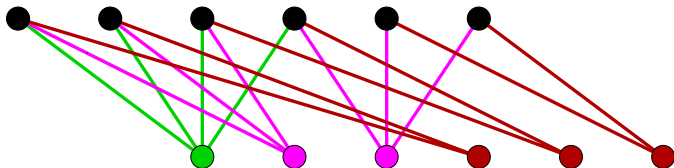
Repeatedly add $\lfloor n/i \rfloor$ bottom vertices echo of degree i , for $i = 2, \dots, n$.



How bad is GreedyVertexCover?

Build a bipartite graph.

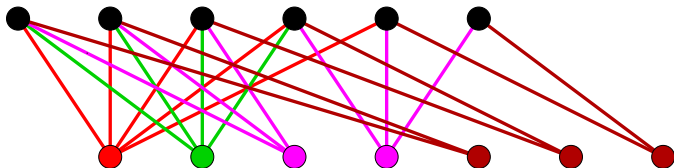
Repeatedly add $\lfloor n/i \rfloor$ bottom vertices of degree i , for
 $i = 2, \dots, n$.



How bad is GreedyVertexCover?

Build a bipartite graph.

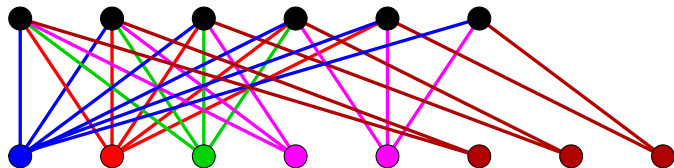
Repeatedly add $\lfloor n/i \rfloor$ bottom vertices of degree i , for
 $i = 2, \dots, n$.



How bad is GreedyVertexCover?

Build a bipartite graph.

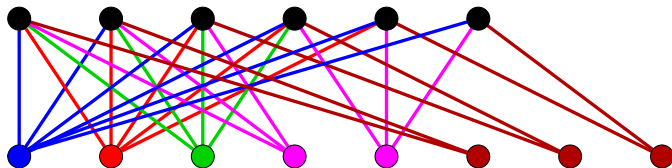
Repeatedly add $\lfloor n/i \rfloor$ bottom vertices of degree i , for
 $i = 2, \dots, n$.



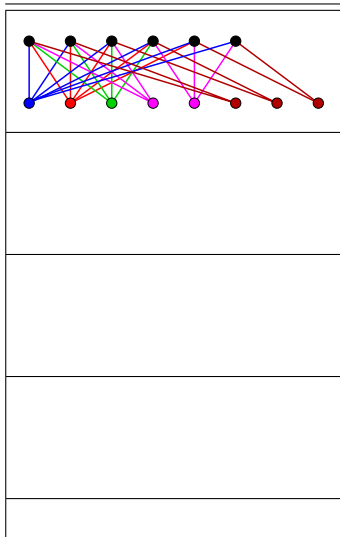
How bad is GreedyVertexCover?

Build a bipartite graph.

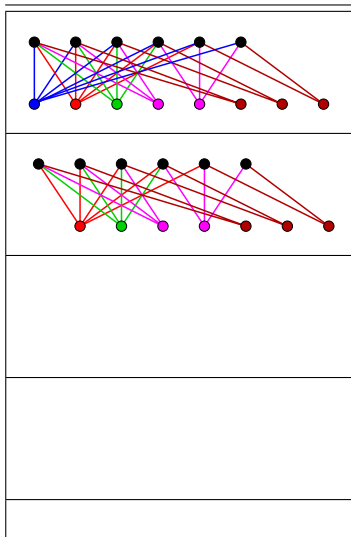
Bottom row has $\sum_{i=2}^n \lfloor n/i \rfloor = \Theta(n \log n)$ vertices.



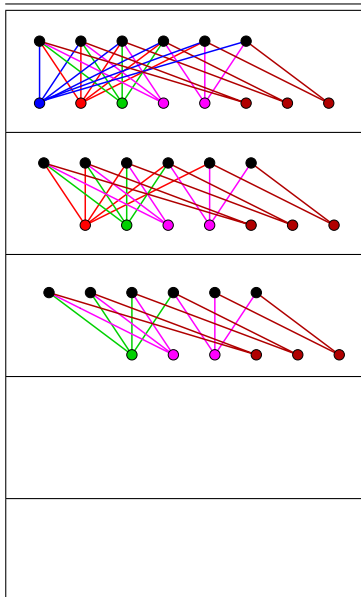
How bad is GreedyVertexCover?



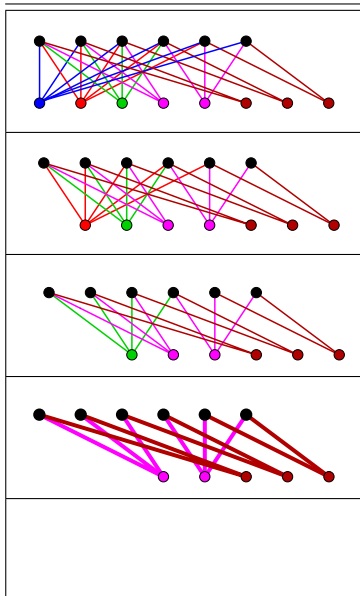
How bad is GreedyVertexCover?



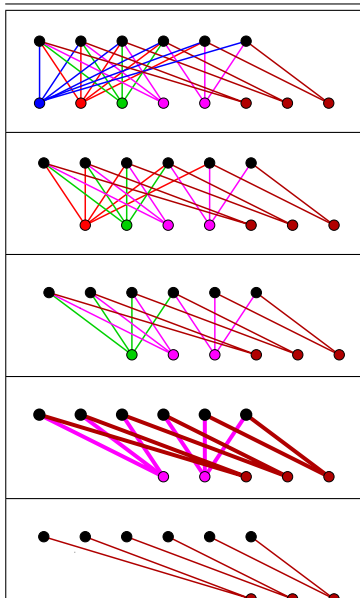
How bad is GreedyVertexCover?



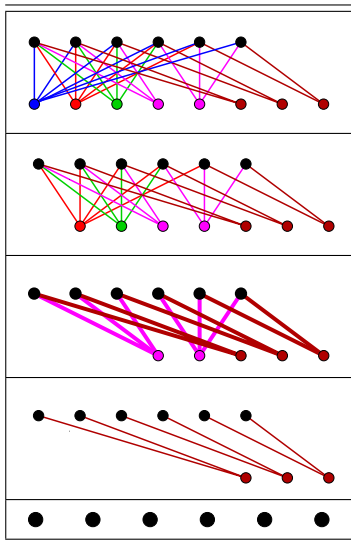
How bad is GreedyVertexCover?



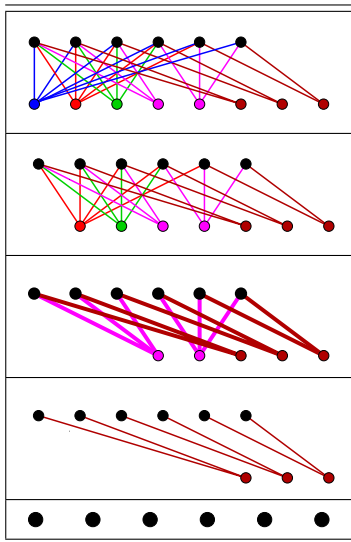
How bad is GreedyVertexCover?



How bad is GreedyVertexCover?

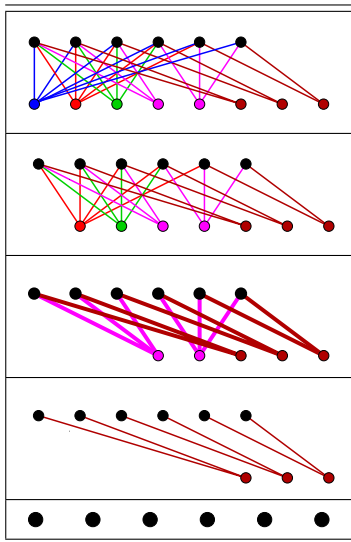


How bad is GreedyVertexCover?



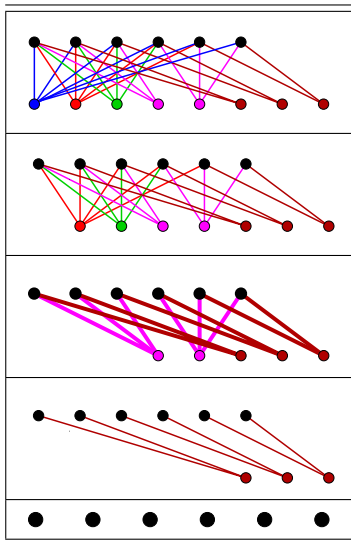
1 Bottom row taken by Greedy.

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.

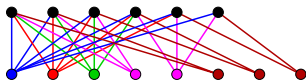
Lemma

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

See notes for details!

How bad is GreedyVertexCover?

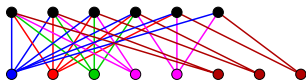
Understanding the graph...



- 1 Top row has n vertices.
- 2 Bottom row has
 - 1 Upper bound:
$$\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \leq \sum_{i=1}^n n/i \leq nH_n = O(n \log n).$$
 - 2 Lower bound: $\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \geq \sum_{i=1}^n n/i - (n-1) - n \geq n(H_n - 2) \geq n(\ln n - 2).$
 - 3 Bottom row has $\Theta(n \log n)$ vertices.
- 3 Greedy algorithm returns bottom row $\Theta(n \log n)$ vertices.
- 4 Optimal solution is top row: n vertices.
- 5 Greedy algorithm is $O(\log n)$ approximation in this case.

How bad is GreedyVertexCover?

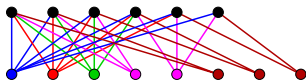
Understanding the graph...



- 1 Top row has n vertices.
- 2 Bottom row has
 - 1 Upper bound:
$$\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \leq \sum_{i=1}^n n/i \leq nH_n = O(n \log n).$$
 - 2 Lower bound: $\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \geq \sum_{i=1}^n n/i - (n-1) - n \geq n(H_n - 2) \geq n(\ln n - 2).$
 - 3 Bottom row has $\Theta(n \log n)$ vertices.
- 3 Greedy algorithm returns bottom row $\Theta(n \log n)$ vertices.
- 4 Optimal solution is top row: n vertices.
- 5 Greedy algorithm is $O(\log n)$ approximation in this case.

How bad is GreedyVertexCover?

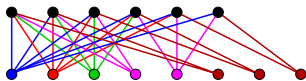
Understanding the graph...



- 1 Top row has n vertices.
- 2 Bottom row has
 - 1 Upper bound:
$$\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \leq \sum_{i=1}^n n/i \leq nH_n = O(n \log n).$$
 - 2 Lower bound: $\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \geq \sum_{i=1}^n n/i - (n-1) - n \geq n(H_n - 2) \geq n(\ln n - 2).$
 - 3 Bottom row has $\Theta(n \log n)$ vertices.
- 3 Greedy algorithm returns bottom row $\Theta(n \log n)$ vertices.
- 4 Optimal solution is top row: n vertices.
- 5 Greedy algorithm is $O(\log n)$ approximation in this case.

How bad is GreedyVertexCover?

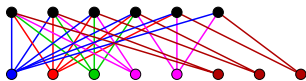
Understanding the graph...



- 1 Top row has n vertices.
- 2 Bottom row has
 - 1 Upper bound:
$$\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \leq \sum_{i=1}^n n/i \leq nH_n = O(n \log n).$$
 - 2 Lower bound: $\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \geq \sum_{i=1}^n n/i - (n-1) - n \geq n(H_n - 2) \geq n(\ln n - 2).$
 - 3 Bottom row has $\Theta(n \log n)$ vertices.
- 3 Greedy algorithm returns bottom row $\Theta(n \log n)$ vertices.
- 4 Optimal solution is top row: n vertices.
- 5 Greedy algorithm is $O(\log n)$ approximation in this case.

How bad is GreedyVertexCover?

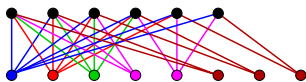
Understanding the graph...



- 1 Top row has n vertices.
- 2 Bottom row has
 - 1 Upper bound:
$$\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \leq \sum_{i=1}^n n/i \leq nH_n = O(n \log n).$$
 - 2 Lower bound: $\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \geq \sum_{i=1}^n n/i - (n-1) - n \geq n(H_n - 2) \geq n(\ln n - 2).$
 - 3 Bottom row has $\Theta(n \log n)$ vertices.
- 3 Greedy algorithm returns bottom row $\Theta(n \log n)$ vertices.
- 4 Optimal solution is top row: n vertices.
- 5 Greedy algorithm is $O(\log n)$ approximation in this case.

How bad is GreedyVertexCover?

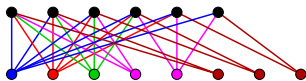
Understanding the graph...



- 1 Top row has n vertices.
- 2 Bottom row has
 - 1 Upper bound:
$$\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \leq \sum_{i=1}^n n/i \leq nH_n = O(n \log n).$$
 - 2 Lower bound: $\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \geq \sum_{i=1}^n n/i - (n-1) - n \geq n(H_n - 2) \geq n(\ln n - 2).$
 - 3 Bottom row has $\Theta(n \log n)$ vertices.
- 3 Greedy algorithm returns bottom row $\Theta(n \log n)$ vertices.
- 4 Optimal solution is top row: n vertices.
- 5 Greedy algorithm is $O(\log n)$ approximation in this case.

How bad is GreedyVertexCover?

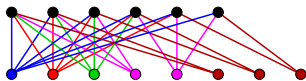
Understanding the graph...



- 1 Top row has n vertices.
- 2 Bottom row has
 - 1 Upper bound:
$$\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \leq \sum_{i=1}^n n/i \leq nH_n = O(n \log n).$$
 - 2 Lower bound: $\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \geq \sum_{i=1}^n n/i - (n-1) - n \geq n(H_n - 2) \geq n(\ln n - 2).$
 - 3 Bottom row has $\Theta(n \log n)$ vertices.
- 3 Greedy algorithm returns bottom row $\Theta(n \log n)$ vertices.
- 4 Optimal solution is top row: n vertices.
- 5 Greedy algorithm is $O(\log n)$ approximation in this case.

How bad is GreedyVertexCover?

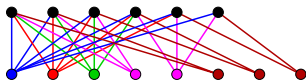
Understanding the graph...



- 1 Top row has n vertices.
- 2 Bottom row has
 - 1 Upper bound:
$$\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \leq \sum_{i=1}^n n/i \leq nH_n = O(n \log n).$$
 - 2 Lower bound: $\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \geq \sum_{i=1}^n n/i - (n-1) - n \geq n(H_n - 2) \geq n(\ln n - 2).$
 - 3 Bottom row has $\Theta(n \log n)$ vertices.
- 3 Greedy algorithm returns bottom row $\Theta(n \log n)$ vertices.
- 4 Optimal solution is top row: n vertices.
- 5 Greedy algorithm is $O(\log n)$ approximation in this case.

How bad is GreedyVertexCover?

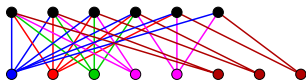
Understanding the graph...



- 1 Top row has n vertices.
- 2 Bottom row has
 - 1 Upper bound:
$$\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \leq \sum_{i=1}^n n/i \leq nH_n = O(n \log n).$$
 - 2 Lower bound: $\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \geq \sum_{i=1}^n n/i - (n-1) - n \geq n(H_n - 2) \geq n(\ln n - 2).$
 - 3 Bottom row has $\Theta(n \log n)$ vertices.
- 3 Greedy algorithm returns bottom row $\Theta(n \log n)$ vertices.
- 4 Optimal solution is top row: n vertices.
- 5 Greedy algorithm is $O(\log n)$ approximation in this case.

How bad is GreedyVertexCover?

Understanding the graph...



- 1 Top row has n vertices.
- 2 Bottom row has
 - 1 Upper bound:
$$\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \leq \sum_{i=1}^n n/i \leq nH_n = O(n \log n).$$
 - 2 Lower bound: $\alpha = \sum_{i=2}^n \lfloor n/i \rfloor \geq \sum_{i=1}^n n/i - (n-1) - n \geq n(H_n - 2) \geq n(\ln n - 2).$
 - 3 Bottom row has $\Theta(n \log n)$ vertices.
- 3 Greedy algorithm returns bottom row $\Theta(n \log n)$ vertices.
- 4 Optimal solution is top row: n vertices.
- 5 Greedy algorithm is $O(\log n)$ approximation in this case.

Some math required

$$H_n = \sum_{i=1}^n \frac{1}{i} \leq \underbrace{1 + \int_{x=1}^n \frac{1}{x} dx}_{1/(i+1) \leq \int_{x=i}^{i+1} (1/x) dx} = 1 + \ln n - \ln 1 = 1 + \ln n.$$

$$\begin{aligned} H_n &= \sum_{i=1}^n \frac{1}{i} \geq \underbrace{\int_{x=1}^{n+1} \frac{1}{x} dx}_{\frac{1}{i} \geq \int_{x=i}^{i+1} \frac{1}{x} dx} = \ln(n+1) - \ln 1 = \ln(n+1) \\ &\geq \ln n. \end{aligned}$$

Lemma

For $H_n = \sum_{i=1}^n 1/i$ we have that $\ln n \leq H_n \leq 1 + \ln n$.

Some math required

$$H_n = \sum_{i=1}^n \frac{1}{i} \leq \underbrace{1 + \int_{x=1}^n \frac{1}{x} dx}_{1/(i+1) \leq \int_{x=i}^{i+1} (1/x) dx} = 1 + \ln n - \ln 1 = 1 + \ln n.$$

$$H_n = \sum_{i=1}^n \frac{1}{i} \geq \underbrace{\int_{x=1}^{n+1} \frac{1}{x} dx}_{\frac{1}{i} \geq \int_{x=i}^{i+1} \frac{1}{x} dx} = \ln(n+1) - \ln 1 = \ln(n+1) \\ \geq \ln n.$$

Lemma

For $H_n = \sum_{i=1}^n 1/i$ we have that $\ln n \leq H_n \leq 1 + \ln n$.

Some math required

$$H_n = \sum_{i=1}^n \frac{1}{i} \leq \underbrace{1 + \int_{x=1}^n \frac{1}{x} dx}_{1/(i+1) \leq \int_{x=i}^{i+1} (1/x) dx} = 1 + \ln n - \ln 1 = 1 + \ln n.$$

$$H_n = \sum_{i=1}^n \frac{1}{i} \geq \underbrace{\int_{x=1}^{n+1} \frac{1}{x} dx}_{\frac{1}{i} \geq \int_{x=i}^{i+1} \frac{1}{x} dx} = \ln(n+1) - \ln 1 = \ln(n+1) \\ \geq \ln n.$$

Lemma

For $H_n = \sum_{i=1}^n 1/i$ we have that $\ln n \leq H_n \leq 1 + \ln n$.

Some math required

$$H_n = \sum_{i=1}^n \frac{1}{i} \leq \underbrace{1 + \int_{x=1}^n \frac{1}{x} dx}_{1/(i+1) \leq \int_{x=i}^{i+1} (1/x) dx} = 1 + \ln n - \ln 1 = 1 + \ln n.$$

$$H_n = \sum_{i=1}^n \frac{1}{i} \geq \underbrace{\int_{x=1}^{n+1} \frac{1}{x} dx}_{\frac{1}{i} \geq \int_{x=i}^{i+1} \frac{1}{x} dx} = \ln(n+1) - \ln 1 = \ln(n+1) \\ \geq \ln n.$$

Lemma

For $H_n = \sum_{i=1}^n 1/i$ we have that $\ln n \leq H_n \leq 1 + \ln n$.

Some math required

$$H_n = \sum_{i=1}^n \frac{1}{i} \leq \underbrace{1 + \int_{x=1}^n \frac{1}{x} dx}_{1/(i+1) \leq \int_{x=i}^{i+1} (1/x) dx} = 1 + \ln n - \ln 1 = 1 + \ln n.$$

$$H_n = \sum_{i=1}^n \frac{1}{i} \geq \underbrace{\int_{x=1}^{n+1} \frac{1}{x} dx}_{\frac{1}{i} \geq \int_{x=i}^{i+1} \frac{1}{x} dx} = \ln(n+1) - \ln 1 = \ln(n+1) \\ \geq \ln n.$$

Lemma

For $H_n = \sum_{i=1}^n 1/i$ we have that $\ln n \leq H_n \leq 1 + \ln n$.

Some math required

$$H_n = \sum_{i=1}^n \frac{1}{i} \leq \underbrace{1 + \int_{x=1}^n \frac{1}{x} dx}_{1/(i+1) \leq \int_{x=i}^{i+1} (1/x) dx} = 1 + \ln n - \ln 1 = 1 + \ln n.$$

$$\begin{aligned} H_n &= \sum_{i=1}^n \frac{1}{i} \geq \underbrace{\int_{x=1}^{n+1} \frac{1}{x} dx}_{\frac{1}{i} \geq \int_{x=i}^{i+1} \frac{1}{x} dx} = \ln(n+1) - \ln 1 = \ln(n+1) \\ &\geq \ln n. \end{aligned}$$

Lemma

For $H_n = \sum_{i=1}^n 1/i$ we have that $\ln n \leq H_n \leq 1 + \ln n$.

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

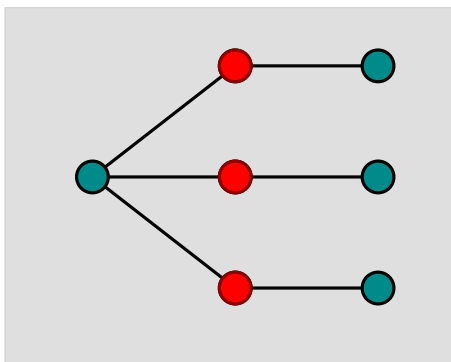
Lower bound follows from lemma.

Upper bound follows from analysis of greedy algorithm for **Set Cover**, which will be done shortly.

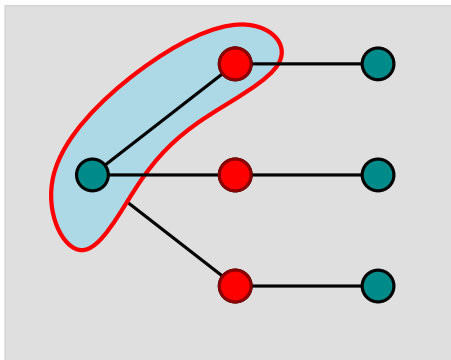
As for the running time, each iteration of the algorithm takes $O(mn)$ time, and there are at most n iterations.

8.1.1: A better greedy algorithm: Two for the price of one

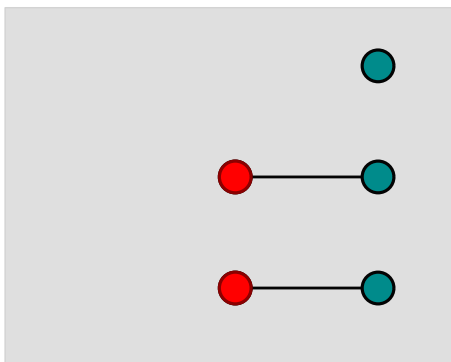
Two for the price of one - example



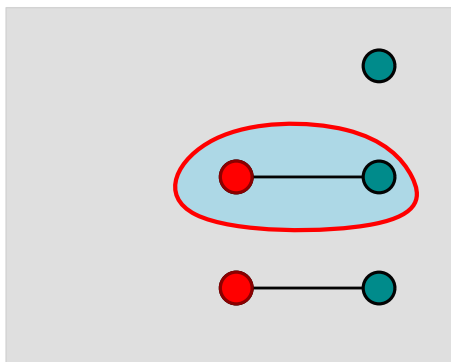
Two for the price of one - example



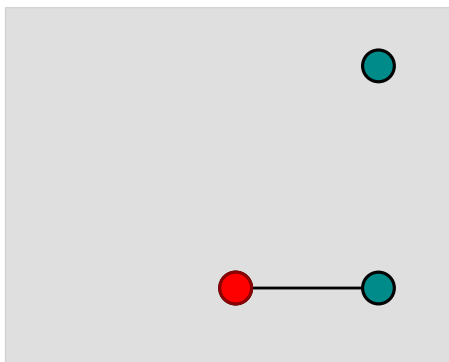
Two for the price of one - example



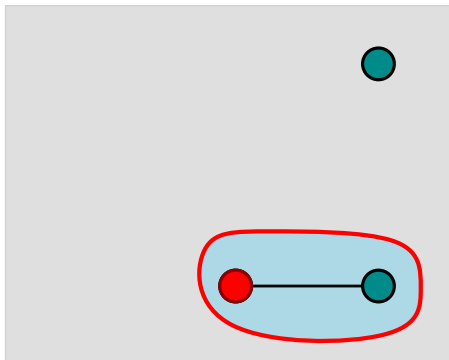
Two for the price of one - example



Two for the price of one - example



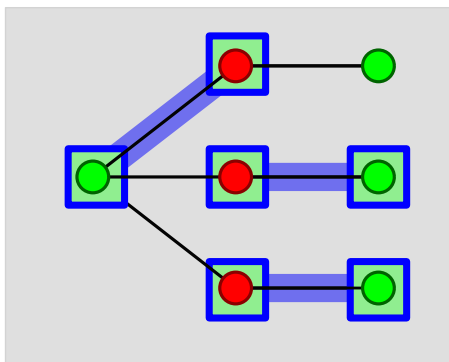
Two for the price of one - example



Two for the price of one - example



Two for the price of one - example



Two for the price of one

ApproxVertexCover(G):

$S \leftarrow \emptyset$

while $E(G) \neq \emptyset$ **do**

$uv \leftarrow$ any edge of G

$S \leftarrow S \cup \{u, v\}$

 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...

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

8.2.1: A silly brute force algorithm for vertex cover

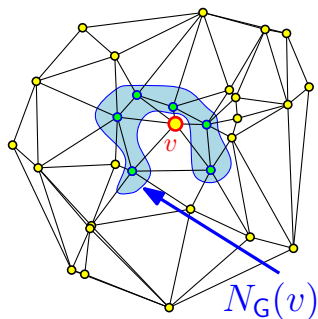
What if the vertex cover is small?

- ① $G = (V, E)$ 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 $O(n^{K+2})$ time.

Neighborhood of a vertex

Definition

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



Induced subgraph

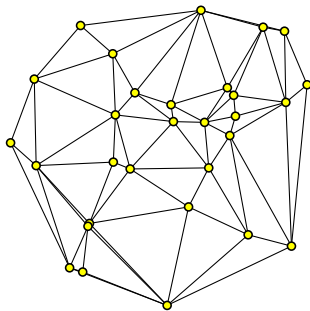
Definition

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

Induced subgraph

Definition

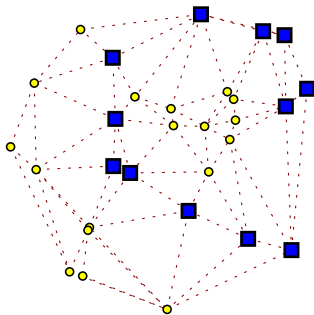
Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a graph. For a subset $S \subseteq \mathbf{V}$, let \mathbf{G}_S be the **induced subgraph** over S .



Induced subgraph

Definition

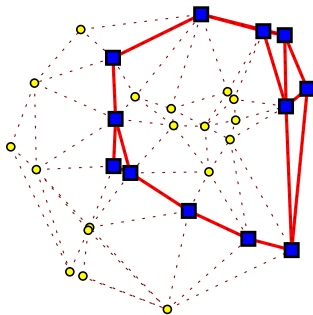
Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a graph. For a subset $S \subseteq \mathbf{V}$, let \mathbf{G}_S be the **induced subgraph** over S .



Induced subgraph

Definition

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 .



8.2.2: Fixed parameter algorithm

Exact algorithm for Set Cover

- 1 **G**: Input graph.
- 2 $\text{Opt} = \text{min size vertex cover for } \mathbf{G}$. $\text{opt} = |\text{Opt}|$.
- 3 Compute a set $S \subseteq \mathbf{V}(\mathbf{G})$ s.t. $|S| \leq 2\text{opt}$.
Takes: $O(n + m)$ time.
- 4 Enumerate over all possible $X \subseteq S$:
 - 1 Check if X is vertex cover in \mathbf{G} .
Takes $O(n + m)$ time.
- 5 Return smallest VC encountered.
- 6 Running time: $O(2^{2\text{opt}}(n + m) + n + m) = O(2^{2\text{opt}}m)$.

Exact algorithm for Set Cover

- 1 **G**: Input graph.
- 2 **Opt** = min size vertex cover for **G**. **opt** = |**Opt**|.
- 3 Compute a set $S \subseteq V(G)$ s.t. $|S| \leq 2\text{opt}$.
Takes: $O(n + m)$ time.
- 4 Enumerate over all possible $X \subseteq S$:
 - 1 Check if X is vertex cover in **G**.
Takes $O(n + m)$ time.
- 5 Return smallest VC encountered.
- 6 Running time: $O(2^{2\text{opt}}(n + m) + n + m) = O(2^{2\text{opt}}m)$.

Exact algorithm for Set Cover

- 1 **G**: Input graph.
- 2 **Opt** = min size vertex cover for **G**. **opt** = |**Opt**|.
- 3 Compute a set $S \subseteq V(G)$ s.t. $|S| \leq 2\text{opt}$.
Takes: $O(n + m)$ time.
- 4 Enumerate over all possible $X \subseteq S$:
 - 1 Check if X is vertex cover in **G**.
Takes $O(n + m)$ time.
- 5 Return smallest VC encountered.
- 6 Running time: $O(2^{2\text{opt}}(n + m) + n + m) = O(2^{2\text{opt}}m)$.

Exact algorithm for Set Cover

- 1 **G**: Input graph.
- 2 **Opt** = min size vertex cover for **G**. **opt** = |**Opt**|.
- 3 Compute a set $S \subseteq V(G)$ s.t. $|S| \leq 2\text{opt}$.
Takes: $O(n + m)$ time.
- 4 Enumerate over all possible $X \subseteq S$:
 - 1 Check if X is vertex cover in **G**.
Takes $O(n + m)$ time.
- 5 Return smallest VC encountered.
- 6 Running time: $O(2^{2\text{opt}}(n + m) + n + m) = O(2^{2\text{opt}}m)$.

Exact algorithm for Set Cover

- 1 **G**: Input graph.
- 2 **Opt** = min size vertex cover for **G**. **opt** = |**Opt**|.
- 3 Compute a set $S \subseteq V(G)$ s.t. $|S| \leq 2\text{opt}$.
Takes: $O(n + m)$ time.
- 4 Enumerate over all possible $X \subseteq S$:
 - 1 Check if X is vertex cover in **G**.
Takes $O(n + m)$ time.
- 5 Return smallest VC encountered.
- 6 Running time: $O(2^{2\text{opt}}(n + m) + n + m) = O(2^{2\text{opt}}m)$.

Exact algorithm for Set Cover

- 1 **G**: Input graph.
- 2 **Opt** = min size vertex cover for **G**. **opt** = |**Opt**|.
- 3 Compute a set $S \subseteq V(G)$ s.t. $|S| \leq 2\text{opt}$.
Takes: $O(n + m)$ time.
- 4 Enumerate over all possible $X \subseteq S$:
 - 1 Check if X is vertex cover in **G**.
Takes $O(n + m)$ time.
- 5 Return smallest VC encountered.
- 6 Running time: $O(2^{2\text{opt}}(n + m) + n + m) = O(2^{2\text{opt}}m)$.

Exact algorithm for Set Cover

- 1 **G**: Input graph.
- 2 **Opt** = min size vertex cover for **G**. **opt** = |**Opt**|.
- 3 Compute a set $S \subseteq V(G)$ s.t. $|S| \leq 2\text{opt}$.
Takes: $O(n + m)$ time.
- 4 Enumerate over all possible $X \subseteq S$:
 - 1 Check if X is vertex cover in **G**.
Takes $O(n + m)$ time.
- 5 Return smallest VC encountered.
- 6 Running time: $O(2^{2\text{opt}}(n + m) + n + m) = O(2^{2\text{opt}}m)$.

Summary of result

Theorem

Given a graph \mathbf{G} with n vertices and m ($\geq n$) edges, and with a vertex cover of size k . Then, one can compute the optimal vertex cover in \mathbf{G} in $O(2^{2k}m)$ time.

Note, that running time is **Fixed Parameter Tractable**.

8.3: Traveling Salesperson Problem

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-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 **NP = P**.

Proof.

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

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

- 4 \exists tour of price n in $\mathbf{J} \iff \exists$ 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{J} : Complete graph over \mathbf{V} .

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

④ \exists tour of price n in $\mathbf{J} \iff \exists$ 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{J} : Complete graph over \mathbf{V} .

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

④ \exists tour of price n in $\mathbf{J} \iff \exists$ 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{J} : Complete graph over \mathbf{V} .

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

④ \exists tour of price n in $\mathbf{J} \iff \exists$ 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 - 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{J} :
 - (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 \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{J} :
 - (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 \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{J} :
 - (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 \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{J} :
 - (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 with the triangle inequality

Because it is not that bad after all.

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

Instance: $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ 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

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

Shortcutting

σ : 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.

TSP_{△≠}-Min

Instance: $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ 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

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

Shortcutting

σ : 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.

TSP_{Δ≠}-Min

Instance: $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ 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

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

Shortcutting

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

TSP with the triangle inequality

Continued...

Definition

Cycle in \mathbf{G} is **Eulerian** if it visits every **edge** of \mathbf{G} exactly once.

Assume you already seen the following:

Lemma

A graph \mathbf{G} has a cycle that visits every edge of \mathbf{G} exactly once (i.e., an Eulerian cycle) if and only if \mathbf{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 \mathbf{G} , respectively.

TSP with the triangle inequality

Continued...

- 1 C_{opt} optimal **TSP** tour in \mathbf{G} .
- 2 **Observation:**
 $\omega(C_{\text{opt}}) \geq \text{weight}(\text{cheapest spanning graph of } \mathbf{G})$.
- 3 **MST:** cheapest spanning graph of \mathbf{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_{opt} optimal **TSP** tour in \mathbf{G} .
- 2 **Observation:**
 $\omega(C_{\text{opt}}) \geq \text{weight}(\text{cheapest spanning graph of } \mathbf{G})$.
- 3 **MST:** cheapest spanning graph of \mathbf{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_{opt} optimal **TSP** tour in \mathbf{G} .
- 2 **Observation:**
 $\omega(C_{\text{opt}}) \geq \text{weight}(\text{cheapest spanning graph of } \mathbf{G})$.
- 3 **MST:** cheapest spanning graph of \mathbf{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_{opt} optimal **TSP** tour in \mathbf{G} .
- 2 **Observation:**
 $\omega(C_{\text{opt}}) \geq \text{weight}(\text{cheapest spanning graph of } \mathbf{G})$.
- 3 **MST:** cheapest spanning graph of \mathbf{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

2-approximation

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

TSP with the triangle inequality

2-approximation

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

TSP with the triangle inequality

2-approximation

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

TSP with the triangle inequality

2-approximation

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

TSP with the triangle inequality

2-approximation

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

TSP with the triangle inequality

2-approximation

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

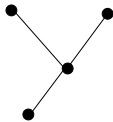
TSP with the triangle inequality

2-approximation

- 1 $T \leftarrow \text{MST}(\mathbf{G})$
- 2 $\mathbf{J} \leftarrow$ duplicate every edge of T .
- 3 H has an Eulerian tour.
- 4 \mathbf{C} : Eulerian cycle in H .
- 5 $\omega(\mathbf{C}) = \omega(H) = 2\omega(T) = 2\omega(\text{MST}(\mathbf{G})) \leq 2\omega(C_{\text{opt}})$.
- 6 π : 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)

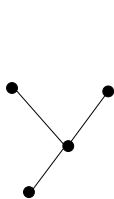
(b)

(c)

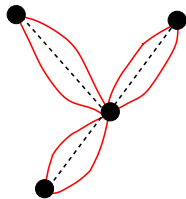
(d)

TSP with the triangle inequality

2-approximation algorithm in figures



(a)



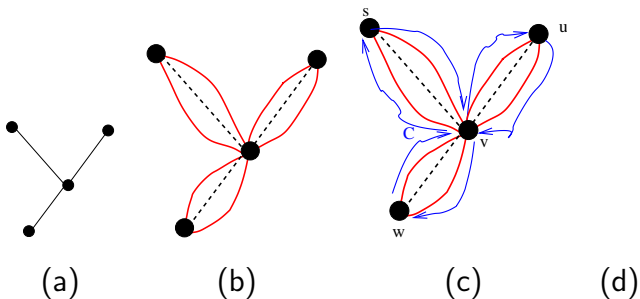
(b)

(c)

(d)

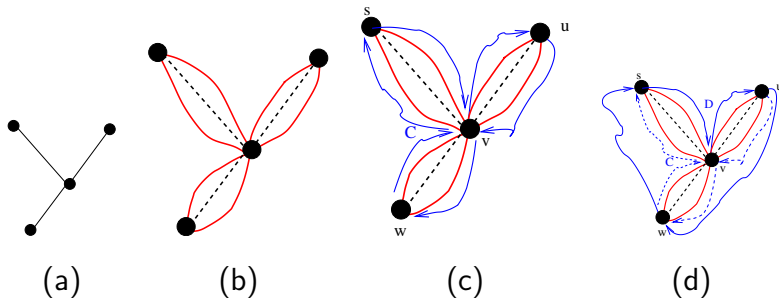
TSP with the triangle inequality

2-approximation algorithm in figures



TSP with the triangle inequality

2-approximation algorithm in figures



Euler Tour: $VUVWVSV$

First occurrences: $VUVWVSV$

Shortcut String: $VUWSV$

TSP with the triangle inequality

2-approximation - result

Theorem

G: Instance of $TSP_{\Delta \neq}$ -Min.

C_{opt} : min cost TSP tour of **G**.

\implies Compute a tour of **G** of length $\leq 2\omega(C_{opt})$.

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* _{$\Delta \neq$} -Min.

C_{opt} : min cost TSP tour of **G**.

\implies Compute a tour of **G** of length $\leq 2\omega(C_{\text{opt}})$.

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_{opt} : min cost TSP tour of **G**.

\implies Compute a tour of **G** of length $\leq 2\omega(C_{\text{opt}})$.

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_{opt} : min cost TSP tour of **G**.

\implies Compute a tour of **G** of length $\leq 2\omega(C_{\text{opt}})$.

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

3/2-approximation

Definition

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

A **perfect matching** is a matching that covers all the vertices of \mathbf{G} .
 ω : weight function on the edges. **Min-weight perfect matching**, is the minimum weight matching among all perfect matching, where

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

TSP with the triangle inequality

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 $\leq \omega(\text{TSP}(\mathbf{G}))/2$.

Min weight perfect matching vs. TSP

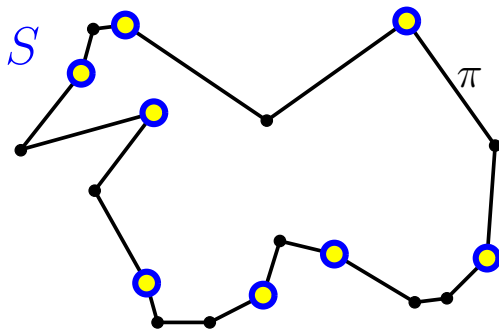
Lemma

$\mathbf{G} = (\mathbf{V}, \mathbf{E})$: 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 $\leq \omega(\text{TSP}(\mathbf{G}))/2$.



Min weight perfect matching vs. TSP

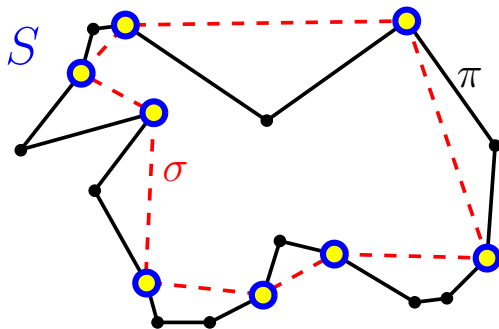
Lemma

$\mathbf{G} = (\mathbf{V}, \mathbf{E})$: 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 $\leq \omega(\text{TSP}(\mathbf{G}))/2$.



Min weight perfect matching vs. TSP

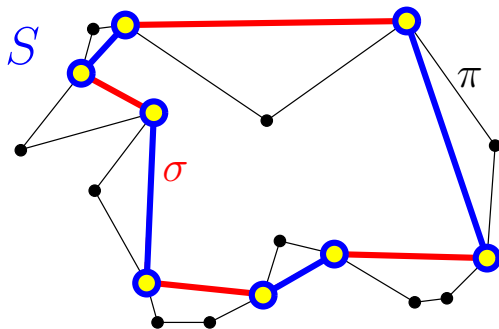
Lemma

$\mathbf{G} = (\mathbf{V}, \mathbf{E})$: complete graph.

$S \subseteq \mathbf{V}$: even size.

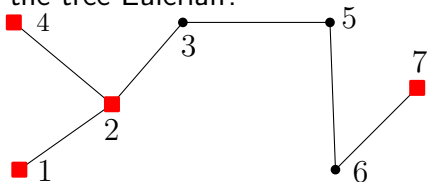
$\omega(\cdot)$: a weight function over \mathbf{E} .

\implies min-weight perfect matching in \mathbf{G}_S is $\leq \omega(\text{TSP}(\mathbf{G}))/2$.



A more perfect tree?

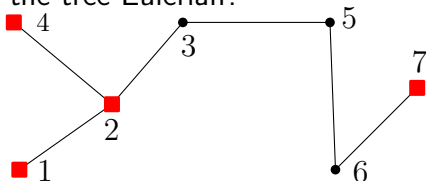
- 1 How to make the tree Eulerian?



- 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 $J = \text{MST} + (\text{min-weight-matching})$ is Eulerian.
- 6 Weight of resulting cycle in $J \leq (3/2)\omega(\text{TSP})$.

A more perfect tree?

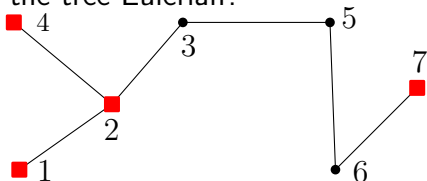
- ① How to make the tree Eulerian?



- 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 $J = \text{MST} + (\text{min-weight-matching})$ is Eulerian.
- ⑥ Weight of resulting cycle in $J \leq (3/2)\omega(\text{TSP})$.

A more perfect tree?

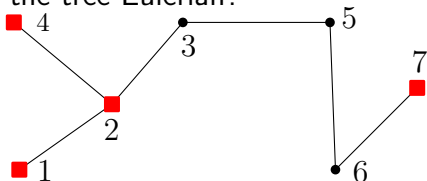
- 1 How to make the tree Eulerian?



- 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 $J = \text{MST} + (\text{min-weight-matching})$ is Eulerian.
- 6 Weight of resulting cycle in $J \leq (3/2)\omega(\text{TSP})$.

A more perfect tree?

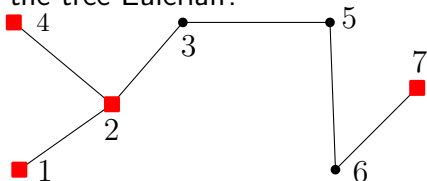
- 1 How to make the tree Eulerian?



- 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 $J = \text{MST} + (\text{min-weight-matching})$ is Eulerian.
- 6 Weight of resulting cycle in $J \leq (3/2)\omega(\text{TSP})$.

A more perfect tree?

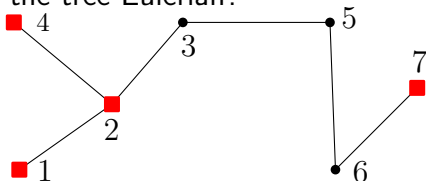
- 1 How to make the tree Eulerian?



- 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 **J = MST + (min-weight-matching)** is Eulerian.
- 6 Weight of resulting cycle in **J** $\leq (3/2)\omega(\text{TSP})$.

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 $\mathbf{J} = \mathbf{MST} + (\text{min-weight-matching})$ is Eulerian.
- ⑥ Weight of resulting cycle in $\mathbf{J} \leq (3/2)\omega(\mathbf{TSP})$.

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 μ 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 μ 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,

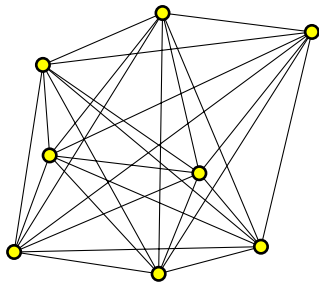
$$\alpha = \sum_{v \in V, d(v) \text{ is odd}} d(v) = \mu - U = \text{even number,}$$

since μ and U are both even.

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

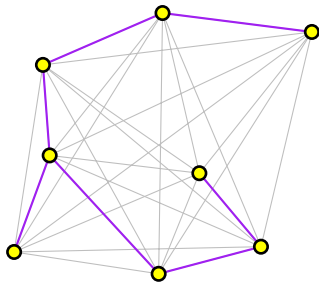
3/2-approximation algorithm for TSP

Animated!



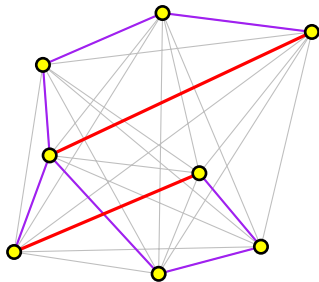
$3/2$ -approximation algorithm for TSP

Animated!



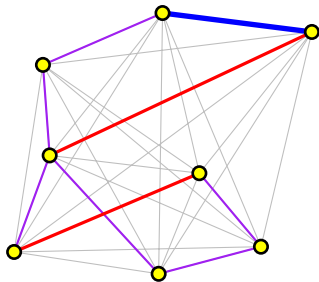
$3/2$ -approximation algorithm for TSP

Animated!



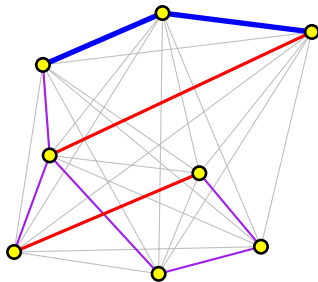
$3/2$ -approximation algorithm for TSP

Animated!



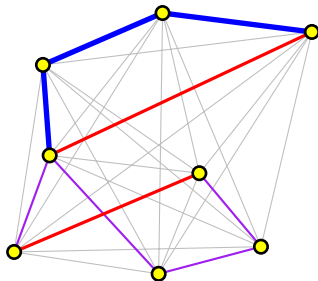
3/2-approximation algorithm for TSP

Animated!



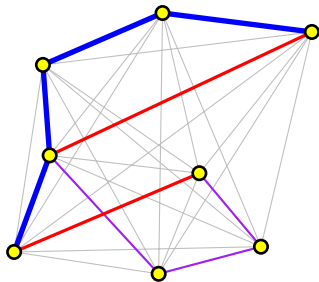
$3/2$ -approximation algorithm for TSP

Animated!



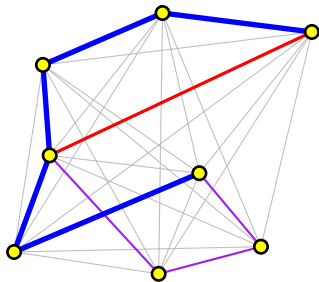
3/2-approximation algorithm for TSP

Animated!



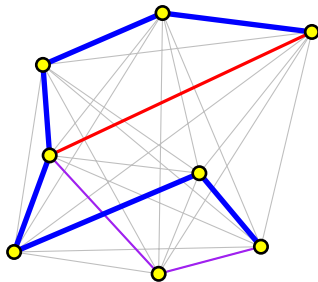
3/2-approximation algorithm for TSP

Animated!



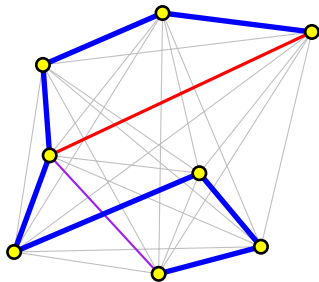
$3/2$ -approximation algorithm for TSP

Animated!



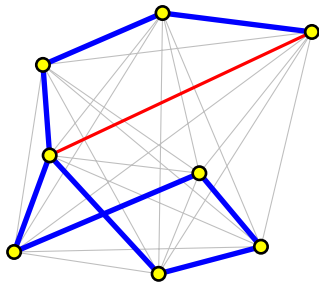
3/2-approximation algorithm for TSP

Animated!



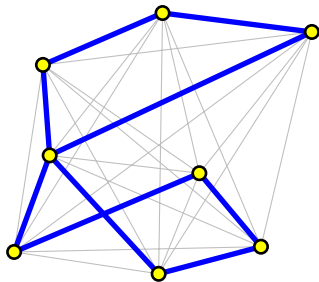
$3/2$ -approximation algorithm for TSP

Animated!



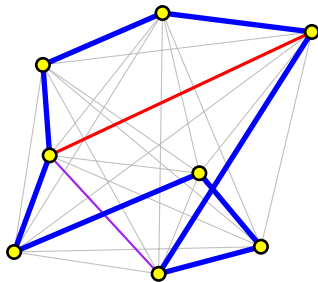
$3/2$ -approximation algorithm for TSP

Animated!



$3/2$ -approximation algorithm for TSP

Animated!



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 $3/2$ -approximation for TSP with the triangle inequality is due to **Christofides [1976]**.

8.4: Alternative FPT algorithm for Vertex Cover (not for lecture)

Exact fixed parameter tractable algorithm

Fixed parameter tractable algorithm for **VertexCoverMin**.

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

fpVCI (X, β)

// β : size of VC computed so far.

if $X = \emptyset$ or \mathbf{G}_X has no edges then return β

$e \leftarrow$ any edge uv of \mathbf{G}_X .

$\beta_1 = \text{fpVCI}(X \setminus \{u, v\}, \beta + 2)$

$\beta_2 = \text{fpVCI}(X \setminus (\{u\} \cup N_{\mathbf{G}_X}(v)), \beta + |N_{\mathbf{G}_X}(v)|)$

$\beta_3 = \text{fpVCI}(X \setminus (\{v\} \cup N_{\mathbf{G}_X}(u)), \beta + |N_{\mathbf{G}_X}(u)|)$

return $\min(\beta_1, \beta_2, \beta_3)$.

algFPVertexCover ($\mathbf{G} = (\mathbf{V}, \mathbf{E})$)

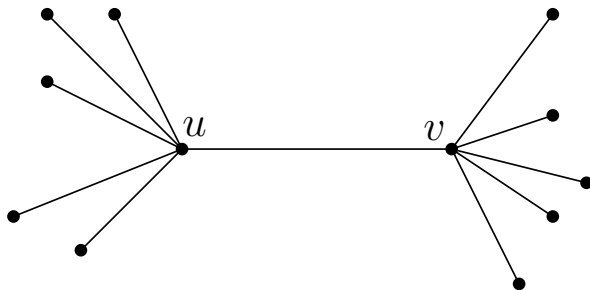
return **fpVCI**($\mathbf{V}, 0$)

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 α , where α 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 α times.
- 2 Algorithm picks $N_{\mathbf{G}_x}(v)$ (i.e., β_2). Conceptually add v to the vertex cover being computed.
- 3 Do the same thing for the case of β_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 α . Then, **algFPVertexCover** returns opt. vertex cover. Running time is $O(3^\alpha n^2)$.

Proof:

- 1 By lemma, recursion tree has depth α .
- 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 α 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 α . Then, **algFPVertexCover** returns opt. vertex cover. Running time is $O(3^\alpha n^2)$.

Proof:

- 1 By lemma, recursion tree has depth α .
- 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 α is some parameter that depends on the problem are **fixed parameter tractable**.

Notes

N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.