

# Chapter 7

## Approximation Algorithms

CS 573: Algorithms, Fall 2014

September 16, 2014

### 7.0.0.1 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.

## 7.1 Greedy algorithms and approximation algorithms

### 7.1.0.2 Greedy algorithms

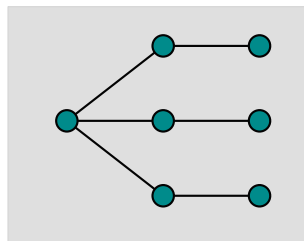
- (A) *greedy algorithms*: do locally the right thing...
- (B) ...and they suck.

#### VertexCoverMin

**Instance:** A graph  $G$ .

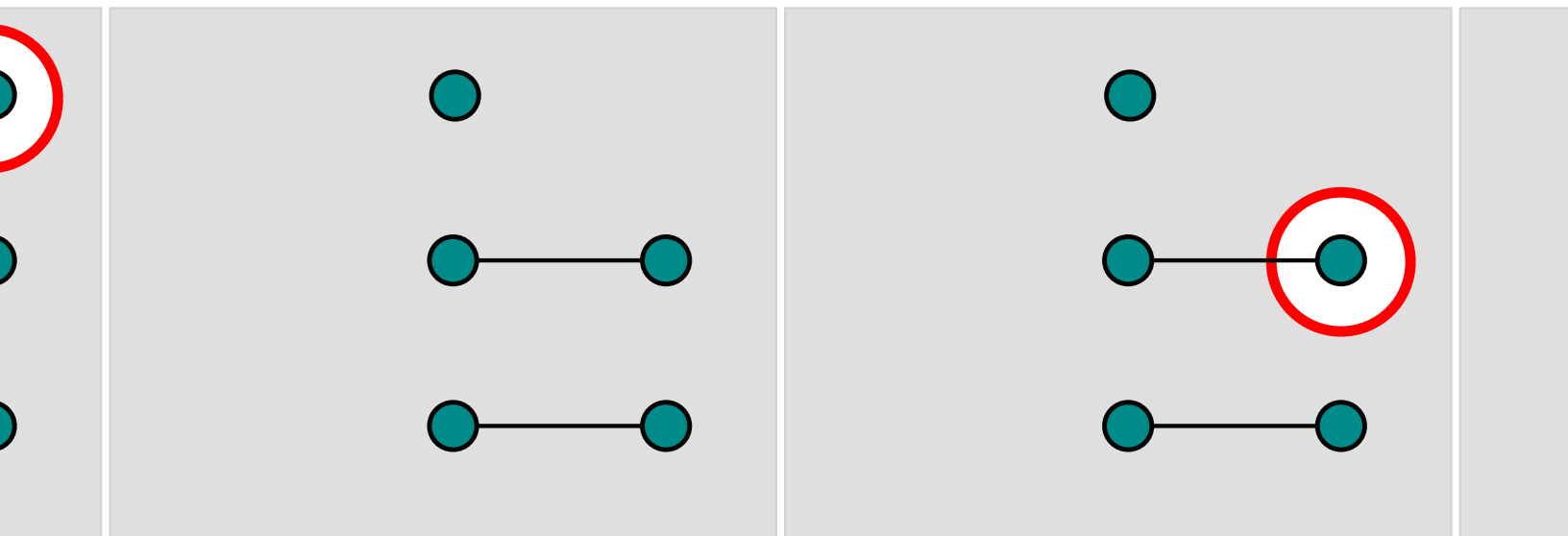
**Question:** Return the **smallest** subset  $S \subseteq V(G)$ , s.t.  $S$  touches all the edges of  $G$ .

- (C) **GreedyVertexCover**: pick vertex with highest degree, remove, repeat.



## 7.1.1 Greedy algorithms

### 7.1.1.1 GreedyVertexCover in action...



**Observation 7.1.1.** GreedyVertexCover returns 4 vertices, but opt is 3 vertices.

### 7.1.1.2 Good enough...

**Definition 7.1.2.** In a *minimization* optimization problem, one looks for a valid solution that minimizes a certain target function.

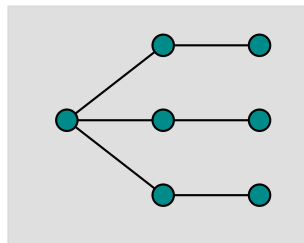
- (A) VertexCoverMin:  $\text{Opt}(G) = \min_{S \subseteq V(G), S \text{ cover of } G} |S|$ .
- (B) VertexCover(G): set realizing sol.
- (C) Opt(G): value of the target function for the optimal solution.

**Definition 7.1.3.** Alg is  $\alpha$ -approximation algorithm for problem Min, achieving an approximation  $\alpha \geq 1$ , if for all inputs G, we have:

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

### 7.1.1.3 Back to GreedyVertexCover

- (A) GreedyVertexCover: pick vertex with highest degree, remove, repeat.
- (B) Returns 4, but opt is 3!




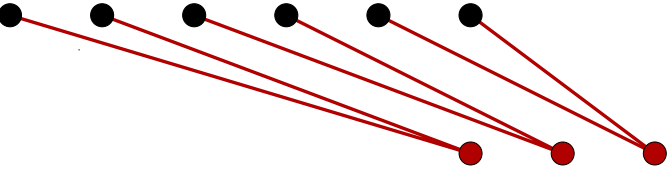
- (C) Can **not** be better than a  $4/3$ -approximation algorithm.
- (D) Actually it is much worse!

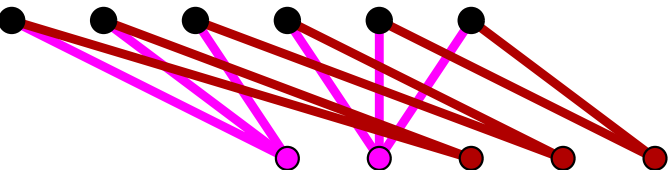
### 7.1.1.4 How bad is GreedyVertexCover?

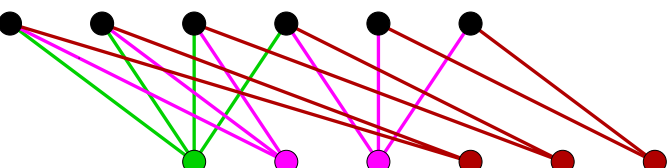
Build a bipartite graph.

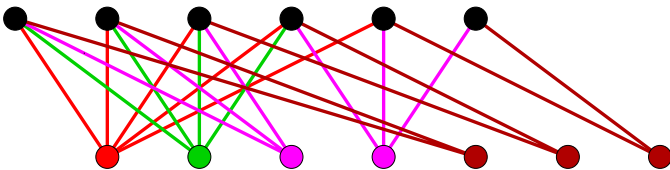
Let the top partite set be of size  $n$ .

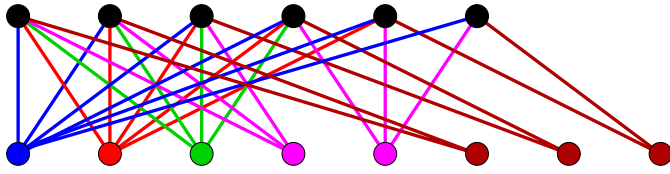
 In the bottom set add  $\lfloor n/2 \rfloor$  vertices of degree 2, such that each edge goes to a different vertex above.

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

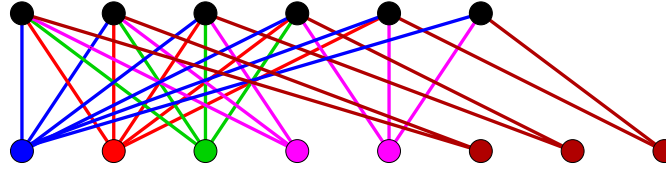
 Repeatedly add  $\lfloor n/i \rfloor$  bottom vertices of degree  $i$ , for  $i = 2, \dots, n$ .

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

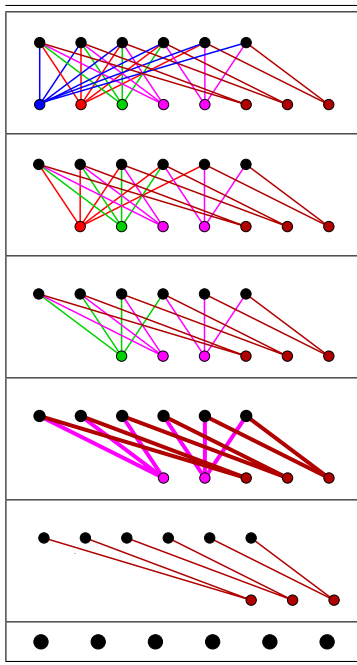
 Repeatedly add  $\lfloor n/i \rfloor$  bottom vertices of degree  $i$ , for  $i = 2, \dots, n$ .



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



#### 7.1.1.5 How bad is GreedyVertexCover?



- (A) Bottom row taken by Greedy.
- (B) Top row was a smaller solution.

**Lemma 7.1.4.** *The algorithm **GreedyVertexCover** is  $\Omega(\log n)$  approximation to the optimal solution to **VertexCoverMin**.*

See notes for details!

#### 7.1.1.6 Greedy Vertex Cover

**Theorem 7.1.5.** *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.

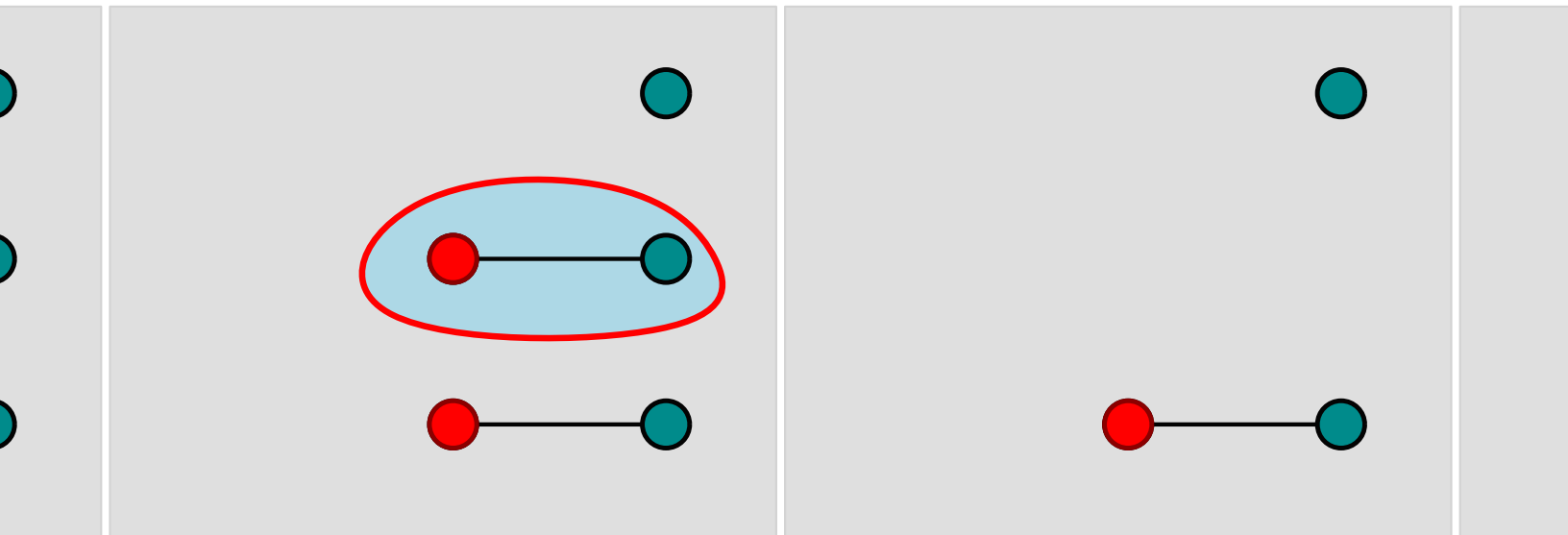
### 7.1.1.7 Two for the price of one

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

Proof...

### 7.1.1.8 Two for the price of one - example



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

### 7.2.1 A silly brute force algorithm for vertex cover

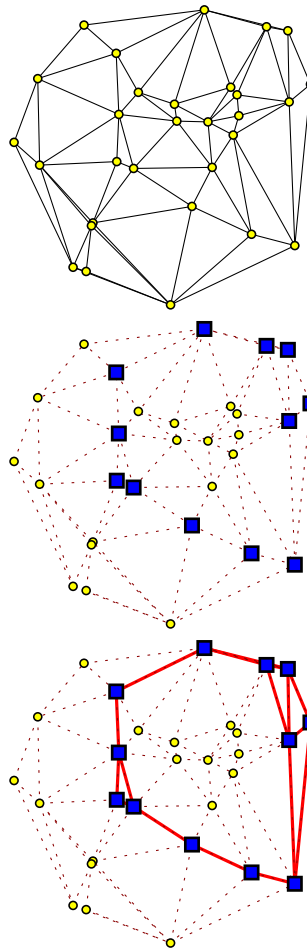
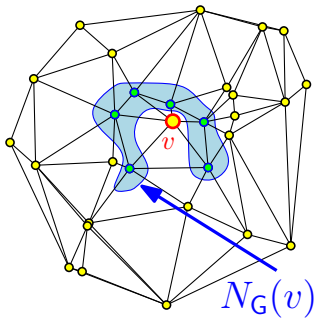
#### 7.2.1.1 What if the vertex cover is small?

- (A)  $G = (V, E)$  with  $n$  vertices
- (B)  $K \leftarrow$  Approximate **VertexCoverMin** up to a factor of two.
- (C) Any vertex cover of  $G$  is of size  $\geq K/2$ .
- (D) Naively compute optimal in  $O(n^{K+2})$  time.

### 7.2.1.2 Induced subgraph

Definition 7.2.2. Let  $G = (V, E)$  be a graph. For a subset  $S \subseteq V$ , let  $G_S$  be the *induced subgraph* over  $S$ .

Definition 7.2.1.  $N_G(v)$ : *Neighborhood* of  $v$  – set of vertices of  $G$  adjacent to  $v$ .



## 7.2.2 Exact fixed parameter tractable algorithm

### 7.2.2.1 Fixed parameter tractable algorithm for VertexCoverMin.

Computes minimum vertex cover for the induced graph  $G_X$ :

```

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

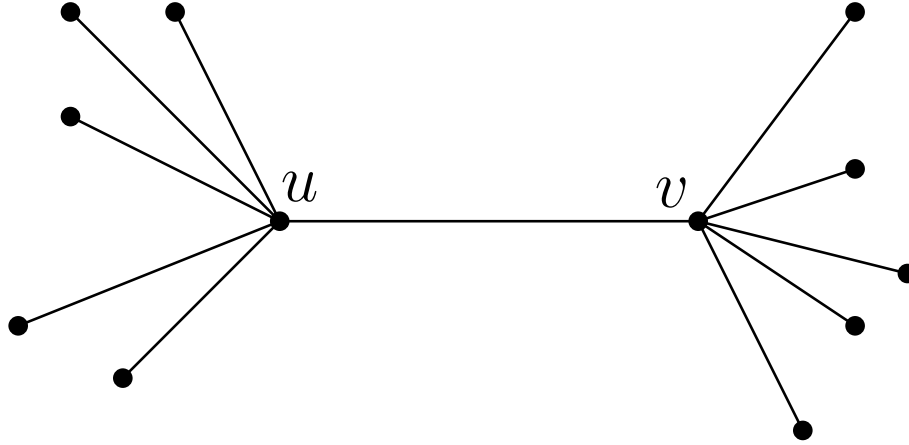
algFPVertexCover( $G = (V, E)$ )
    return fpVCI( $V, 0$ )

```

### 7.2.2.2 Depth of recursion

**Lemma 7.2.3.** *The algorithm **algFPVertexCover** returns the optimal solution to the given instance of **VertexCoverMin**.*

Proof...



### 7.2.2.3 Depth of recursion II

**Lemma 7.2.4.** *The depth of the recursion of **algFPVertexCover**( $G$ ) is at most  $\alpha$ , where  $\alpha$  is the minimum size vertex cover in  $G$ .*

*Proof:* (A) When the algorithm takes both  $u$  and  $v$  - one of them in opt. Can happen at most  $\alpha$  times.  
 (B) Algorithm picks  $N_{G_X}(v)$  (i.e.,  $\beta_2$ ). Conceptually add  $v$  to the vertex cover being computed.  
 (C) Do the same thing for the case of  $\beta_3$ .  
 (D) Every such call add one element of the opt to conceptual set cover. Depth of recursion is  $\leq \alpha$ .

## 7.2.3 Vertex Cover

### 7.2.3.1 Exact fixed parameter tractable algorithm

**Theorem 7.2.5.**  *$G$ : 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:

- (A) By lemma, recursion tree has depth  $\alpha$ .
- (B) Rec-tree contains  $\leq 2 \cdot 3^\alpha$  nodes.
- (C) 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**.

## 7.3 Traveling Salesperson Problem

### 7.3.0.2 TSP

#### TSP-Min

**Instance:**  $G = (V, E)$  a complete graph, and  $\omega(e)$  a cost function on edges of  $G$ .

**Question:** The cheapest tour that visits all the vertices of  $G$  exactly once.

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

### 7.3.0.3 TSP Hardness

**Theorem 7.3.1.** **TSP-Min** can not be approximated within **any** factor unless **NP = P**.

Proof.

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

### 7.3.0.4 TSP Hardness - proof continued

*Proof:* (A) Price of all tours are either:

- (i)  $n$  (only if  $\exists$  Hamiltonian cycle in  $G$ ),
- (ii) larger than  $cn + 1$  (actually,  $\geq cn + (n - 1)$ ).
- (B) Suppose you had a poly time  $c$ -approximation to **TSP-Min**.
- (C) 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  $OPT \leq cn < cn + 1$
- (D)  $c$ -approximation algorithm to **TSP**  $\implies$  poly-time algorithm for **NP-Complete** problem.  
Possible only if **P = NP**.



### 7.3.1 TSP with the triangle inequality

#### 7.3.1.1 Because it is not that bad after all.

##### TSP<sub>Δ≠</sub>-Min

**Instance:**  $G = (V, E)$  is a complete graph. There is also a cost function  $\omega(\cdot)$  defined over the edges of  $G$ , that complies with the triangle inequality.

**Question:** The cheapest tour that visits all the vertices of  $G$  exactly once.

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

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

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

### 7.3.2 TSP with the triangle inequality

#### 7.3.2.1 Continued...

Definition 7.3.2. Cycle in  $G$  is **Eulerian** if it visits every **edge** of  $G$  exactly once.

Assume you already seen the following:

**Lemma 7.3.3.** *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.*

### 7.3.3 TSP with the triangle inequality

#### 7.3.3.1 Continued...

- (A)  $C_{\text{opt}}$  optimal **TSP** tour in  $G$ .
- (B) **Observation:**  $\omega(C_{\text{opt}}) \geq \text{weight}(\text{cheapest spanning graph of } G)$ .
- (C) **MST**: cheapest spanning graph of  $G$ .  
 $\omega(C_{\text{opt}}) \geq \omega(\text{MST}(G))$
- (D)  $O(n \log n + m) = O(n^2)$ : time to compute **MST**.  $n = |V(G)|$ ,  $m = \binom{n}{2}$ .

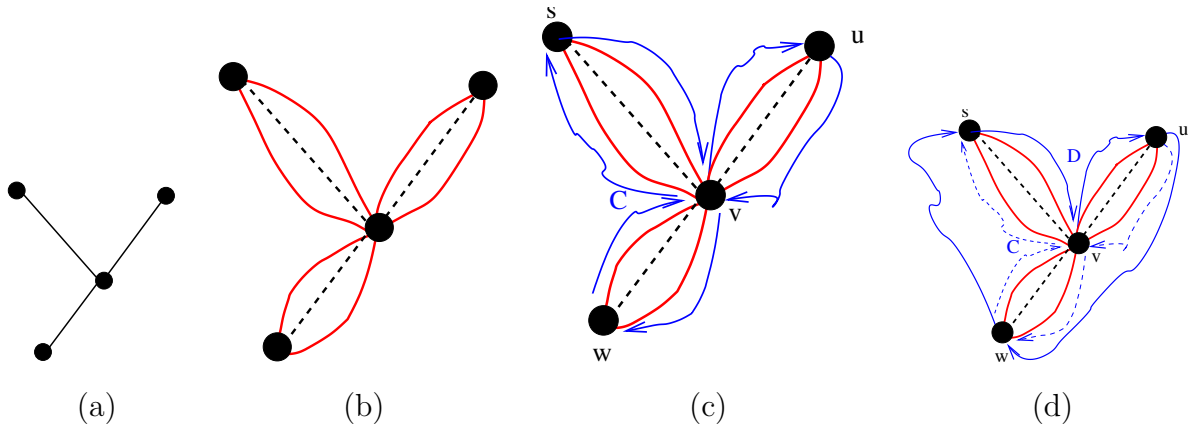
### 7.3.4 TSP with the triangle inequality

#### 7.3.4.1 2-approximation

- (A)  $T \leftarrow \text{MST}(G)$
- (B)  $H \leftarrow$  duplicate every edge of  $T$ .
- (C)  $H$  has an Eulerian tour.
- (D)  $C$ : Eulerian cycle in  $H$ .
- (E)  $\omega(C) = \omega(H) = 2\omega(T) = 2\omega(\text{MST}(G)) \leq 2\omega(C_{\text{opt}})$ .
- (F)  $\pi$ : Shortcut  $C$  so visit every vertex once.
- (G)  $\omega(\pi) \leq \omega(C)$

### 7.3.5 TSP with the triangle inequality

#### 7.3.5.1 2-approximation algorithm in figures



Euler  
First  
Shortcut String: VUWSV

Tour:  
occurrences:

VUUVWSV  
VUUVWSV

### 7.3.6 TSP with the triangle inequality

#### 7.3.6.1 2-approximation - result

**Theorem 7.3.4.**  $G$ : 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.

### 7.3.7 TSP with the triangle inequality

#### 7.3.7.1 3/2-approximation

**Definition 7.3.5.**  $G = (V, E)$ , 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  $G$ .

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

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

### 7.3.8 TSP with the triangle inequality

#### 7.3.8.1 3/2-approximation

The following is known:

**Theorem 7.3.6.** Given a graph  $G$  and weights on the edges, one can compute the min-weight perfect matching of  $G$  in polynomial time.

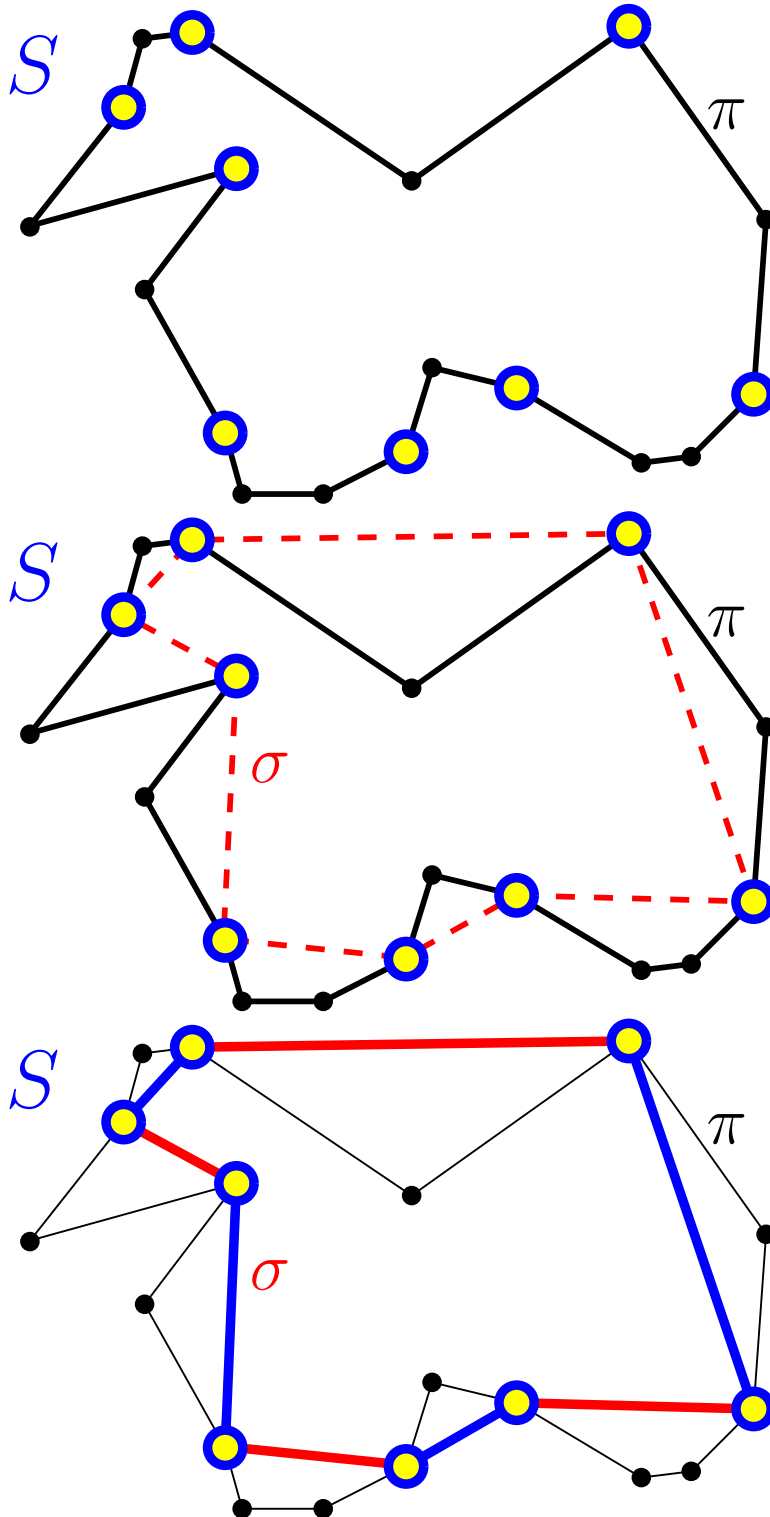
### 7.3.8.2 Min weight perfect matching vs. TSP

**Lemma 7.3.7.**  $G = (V, E)$ : complete graph.

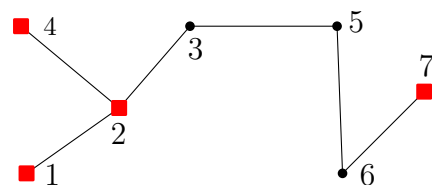
$S \subseteq V$ : even size.

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

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



### 7.3.8.3 A more perfect tree?



- (A) How to make the tree Eulerian?
- (B) Pesky odd degree vertices must die!
- (C) Number of odd degree vertices in a graph is even!
- (D) Compute min-weight matching on odd vertices, and add to **MST**.
- (E)  $H = \text{MST} + \text{min-weight-matching}$  is Eulerian.
- (F) Weight of resulting cycle in  $H \leq (3/2)\omega(\text{TSP})$ .

### 7.3.8.4 Even number of odd degree vertices

**Lemma 7.3.8.** *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,

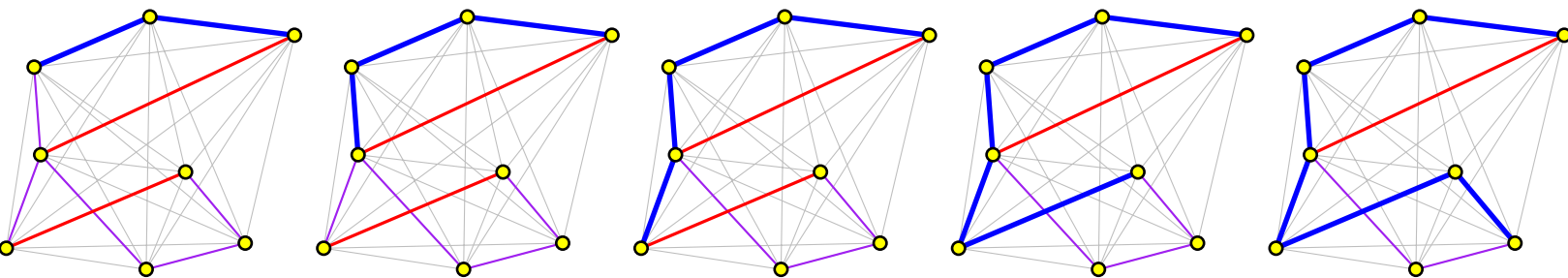
$$\alpha = \sum_{v \in V, d(v) \text{ is odd}} d(v) = \mu - U = \text{even number},$$

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.

### 7.3.9 3/2-approximation algorithm for TSP

#### 7.3.9.1 Animated!



### 7.3.10 3/2-approximation algorithm for TSP

#### 7.3.10.1 The result

**Theorem 7.3.9.** *Given an instance of TSP with the triangle inequality, one can compute in polynomial time, a  $(3/2)$ -approximation to the optimal TSP.*

#### 7.3.10.2 Biographical Notes

The  $3/2$ -approximation for TSP with the triangle inequality is due to ?.