Chapter 8

Approximation Algorithms

NEW CS 473: Theory II, Fall 2015
September 17, 2015

8.0.0.1 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.

8.1 Greedy algorithms and approximation algorithms – Vertex Cover

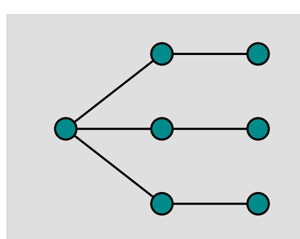
8.1.0.1 Greedy algorithms

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

VertexCoverMin

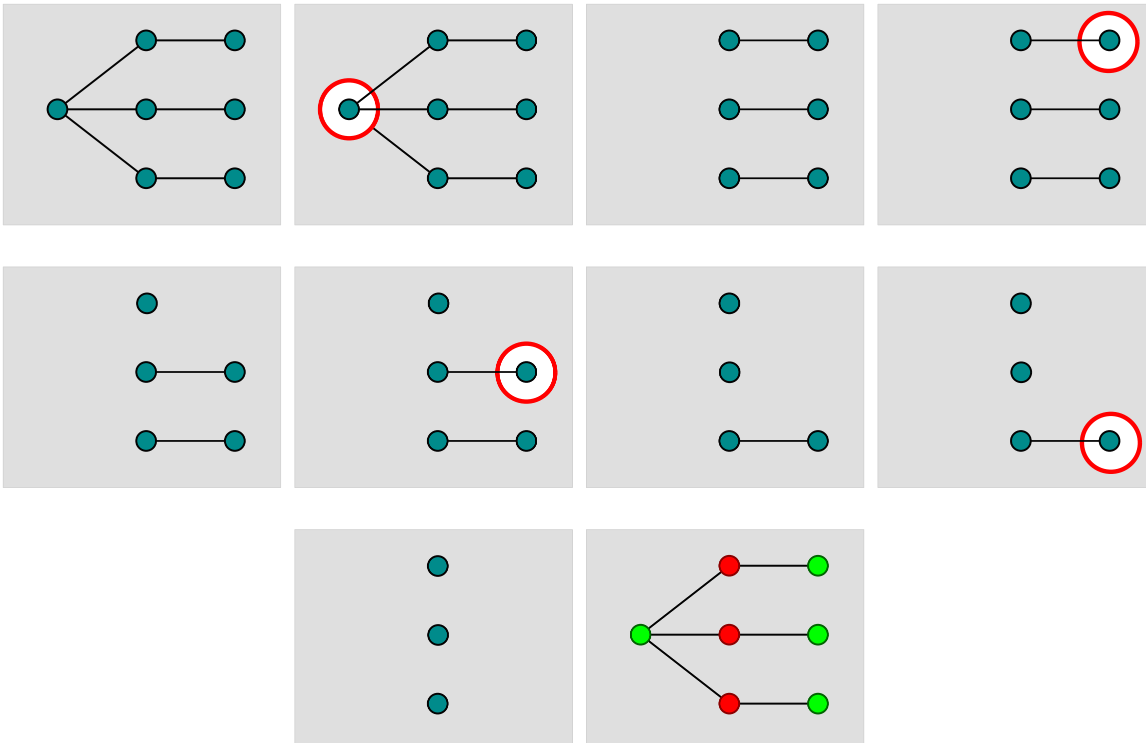
Instance: A graph G.
Question: Return the smallest subset $S \subseteq V(\mathsf{G})$, s.t. S touches all the edges of G.

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



8.1.1 Greedy algorithms

8.1.1.1 GreedyVertexCover in action...



Observation 8.1.1. **GreedyVertexCover** *returns 4 vertices, but opt is 3 vertices.*

8.1.1.2 Good enough...

Definition 8.1.2. In a ***minimization*** optimization problem, one looks for a valid solution that minimizes a certain target function.

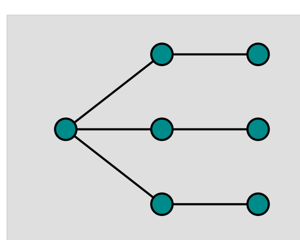
(A) VertexCoverMin: $\mathrm{Opt}(\mathsf{G}) = \min_{S \subseteq V(\mathsf{G}), S \text{ cover of } G} |S|$.
(B) VertexCover(G): set realizing sol.
(C) Opt(G): value of the target function for the optimal solution.

Definition 8.1.3. **Alg** is ***α-approximation algorithm*** for problem **Min**, achieving an approximation $\alpha \geq 1$, if for all inputs G, we have:

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

8.1.1.3 Back to GreedyVertexCover

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



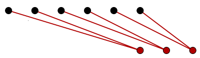
(C) Can **not** be better than a 4/3-approximation algorithm.
(D) Actually it is much worse!

8.1.1.4 How bad is GreedyVertexCover?

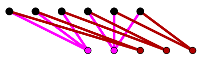
Build a bipartite graph.

Let the top partite set be of size n.

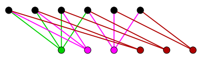
In the bottom set add $\lfloor n/2 \rfloor$ vertices of degree 2, such that each edge goes to a different vertex above.



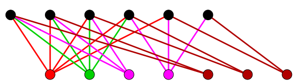
Repeatedly add $\lfloor n/i \rfloor$ bottom vertices echo of degree i, for $i = 2, \ldots, n$.



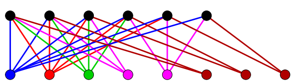
Repeatedly add $\lfloor n/i \rfloor$ bottom vertices of degree i, for $i = 2, \ldots, n$.



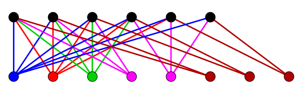
Repeatedly add $\lfloor n/i \rfloor$ bottom vertices of degree i, for $i = 2, \ldots, n$.



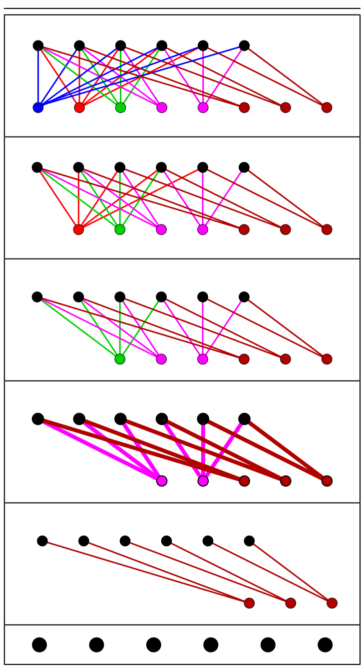
Repeatedly add $\lfloor n/i \rfloor$ bottom vertices of degree i, for $i = 2, \ldots, n$.



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



8.1.1.5 How bad is GreedyVertexCover?



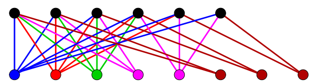
(A) Bottom row taken by Greedy.
(B) Top row was a smaller solution.

Lemma 8.1.4. *The algorithm* **GreedyVertex-Cover** *is $\Omega(\log n)$ approximation to the optimal solution to VertexCoverMin.*

See notes for details!

8.1.2 How bad is GreedyVertexCover?

8.1.2.1 Understanding the graph...



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

8.1.2.2 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 8.1.5. *For $H_n = \sum_{i=1}^{n} 1/i$ we have that $\ln n \leq H_n \leq 1 + \ln n$.*

8.1.2.3 Greedy Vertex Cover

Theorem 8.1.6. *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 $O(mn^2)$.*

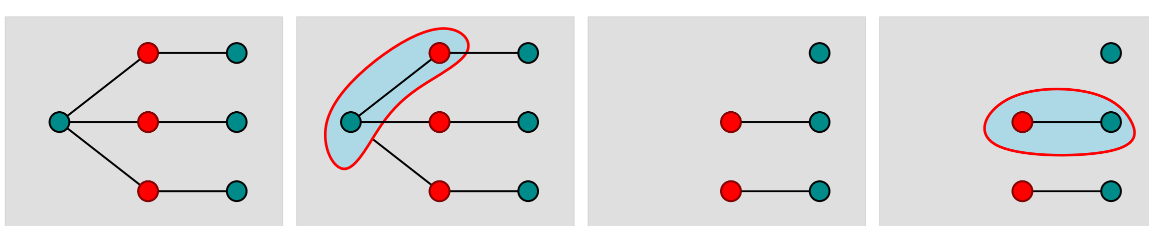
Proof Lower bound follows from lemma.

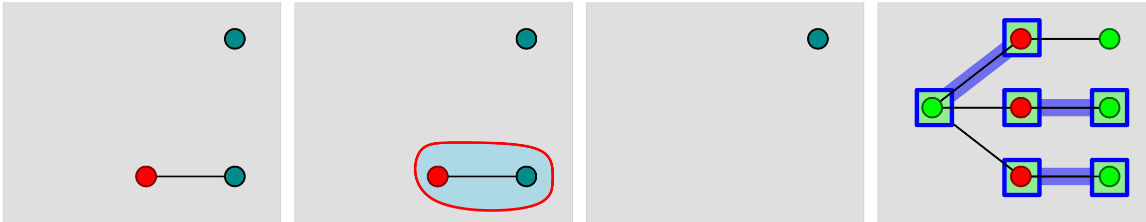
Upper bound follows from analysis of greedy algorithm for **Set Cover**, which will be done shortly.

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

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

8.1.3.1 Two for the price of one - example





8.1.3.2 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 8.1.7. **ApproxVertexCover** *is a* 2*-approximation algorithm for* VertexCoverMin *that runs in* $O(n^2)$ *time.*

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

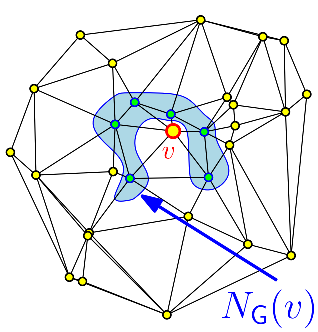
8.2.1 A silly brute force algorithm for vertex cover

8.2.1.1 What if the vertex cover is small?

(A) $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ 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 $O\left(n^{K+2}\right)$ time.

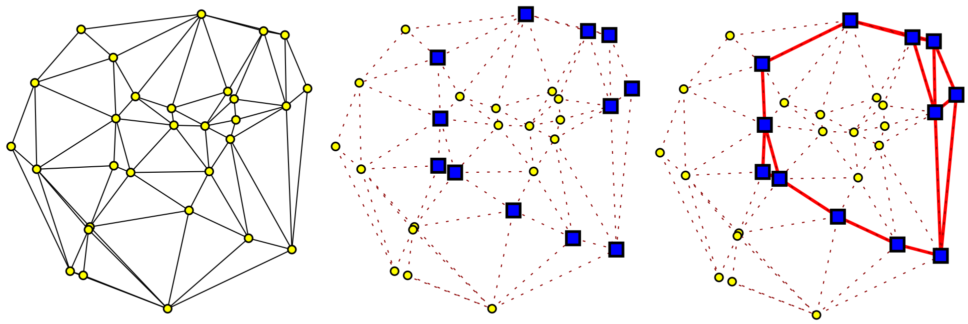
8.2.1.2 Neighborhood of a vertex

Definition 8.2.1. $N_{\mathsf{G}}(v)$: ***Neighborhood*** of v – set of vertices of G adjacent to v.



8.2.1.3 Induced subgraph

Definition 8.2.2. Let $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ be a graph. For a subset $S \subseteq \mathsf{V}$, let G_S be the ***induced subgraph*** over S.



8.2.2 Fixed parameter algorithm

8.2.2.1 Exact algorithm for Set Cover

(A) G: Input graph.
(B) Opt = min size vertex cover for G. $\mathsf{opt} = |\text{Opt}|$.
(C) Compute a set $S \subseteq \mathsf{V}(\mathsf{G})$ s.t. $|S| \leq 2\mathsf{opt}$.
    Takes: $O(n+m)$ time.
(D) Enumerate over all possible $X \subseteq S$:
    (A) Check if X is vertex cover in G.
        Takes $O(n+m)$ time.
(E) Return smallest VC encountered.
(F) Running time: $O(2^{2\mathsf{opt}}(n+m) + n + m) = O(2^{2\mathsf{opt}} m)$.

8.2.2.2 Summary of result

Theorem 8.2.3. *Given a graph* 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* G *in* $O(2^{2k} m)$ *time.*

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

8.3 Traveling Salesperson Problem

8.3.0.1 TSP

TSP-Min

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

8.3.0.2 TSP Hardness

Theorem 8.3.1. *TSP-Min can not be approximated within* ***any*** *factor unless* **NP** $=$ **P**.

Proof.

(A) Reduction from **Hamiltonian Cycle** into **TSP**.
(B) $\mathsf{G} = (\mathsf{V}, \mathsf{E})$: instance of Hamiltonian cycle.
(C) J: Complete graph over V.

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

(D) $\exists$ tour of price n in J $\iff$ $\exists$ 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

8.3.0.3 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 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 $OPT \leq cn < cn + 1$
(D) c-approximation algorithm to TSP $\implies$ poly-time algorithm for **NP-Complete** problem. Possible only if **P** $=$ **NP**.

8.3.1 TSP with the triangle inequality

8.3.1.1 Because it is not that bad after all.

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

> **Instance**: $\mathsf{G} = (V, E)$ 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

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

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

8.3.2 TSP with the triangle inequality

8.3.2.1 Continued...

Definition 8.3.2. Cycle in G is ***Eulerian*** if it visits every **edge** of G exactly once.

Assume you already seen the following:

Lemma 8.3.3. *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.*

8.3.3 TSP with the triangle inequality

8.3.3.1 Continued...

(A) C_{opt} optimal **TSP** tour in G.

(B) **Observation**: $\omega(C_{\text{opt}}) \geq \text{weight}\Big(\text{cheapest spanning graph of } \mathsf{G}\Big)$.

(C) MST: cheapest spanning graph of G.
$\omega(C_{\text{opt}}) \geq \omega(\text{MST}(\mathsf{G}))$

(D) $O(n \log n + m) = O(n^2)$: time to compute MST. $n = |\mathsf{V}(\mathsf{G})|$, $m = \binom{n}{2}$.

8.3.4 TSP with the triangle inequality

8.3.4.1 2-approximation

(A) $T \leftarrow \text{MST}(\mathsf{G})$

(B) $\mathsf{J} \leftarrow$ duplicate very edge of T.

(C) H has an Eulerian tour.

(D) C: Eulerian cycle in H.

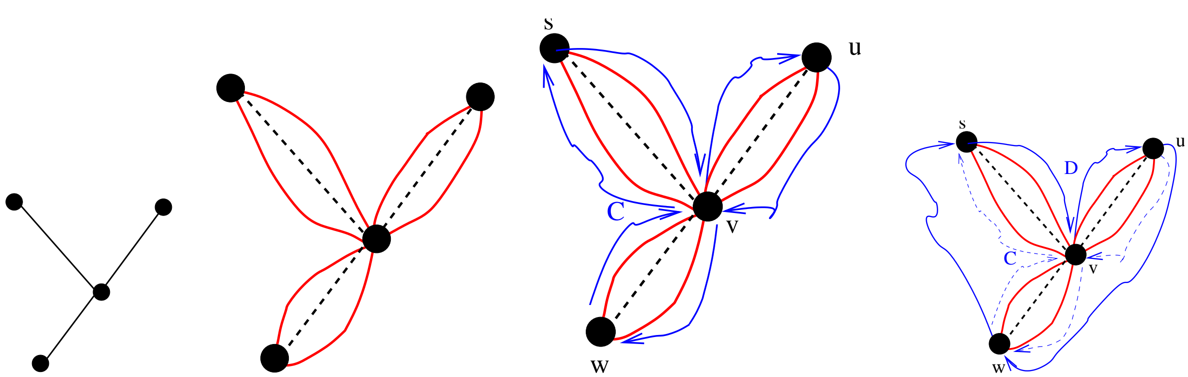
(E) $\omega(\mathsf{C}) = \omega(H) = 2\omega(T) = 2\omega(MST(\mathsf{G})) \leq 2\omega(C_{\text{opt}})$.

(F) π: Shortcut C so visit every vertex once.

(G) $\omega(\pi) \leq \omega(\mathsf{C})$

8.3.5 TSP with the triangle inequality

8.3.5.1 2-approximation algorithm in figures



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

Euler Tour: VUVWVSV
First occurrences: VUVWVSV
Shortcut String: VUWSV

8.3.6 TSP with the triangle inequality

8.3.6.1 2-approximation - result

Theorem 8.3.4. G*: Instance of* $\text{TSP}_{\triangle\neq}$*-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.

8.3.7 TSP with the triangle inequality

8.3.7.1 3/2-approximation

Definition 8.3.5. $\mathsf{G} = (V, E)$, 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

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

8.3.8 TSP with the triangle inequality

8.3.8.1 3/2-approximation

The following is known:

Theorem 8.3.6. *Given a graph G and weights on the edges, one can compute the min-weight perfect matching of G in polynomial time.*

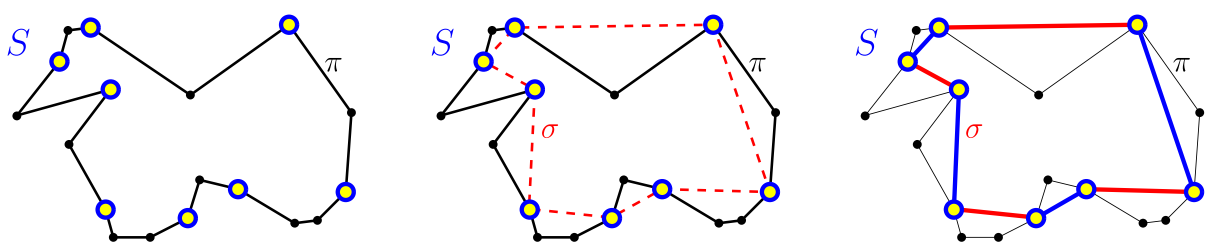
8.3.8.2 Min weight perfect matching vs. TSP

Lemma 8.3.7. $\mathsf{G} = (\mathsf{V}, \mathsf{E})$*: complete graph.*

$S \subseteq \mathsf{V}$*: even size.*

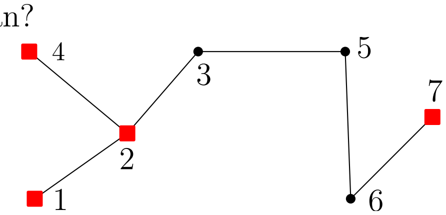
$\omega(\cdot)$*: a weight function over* E*.*

$\implies$ *min-weight perfect matching in* G_S *is* $\leq \omega(\mathrm{TSP}(\mathsf{G}))/2$*.*



8.3.8.3 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) $\mathsf{J} = \mathrm{MST} + $ (min-weight-matching) is Eulerian.

(F) Weight of resulting cycle in $\mathsf{J} \leq (3/2)\omega(\mathrm{TSP})$.

8.3.8.4 Even number of odd degree vertices

Lemma 8.3.8. *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,

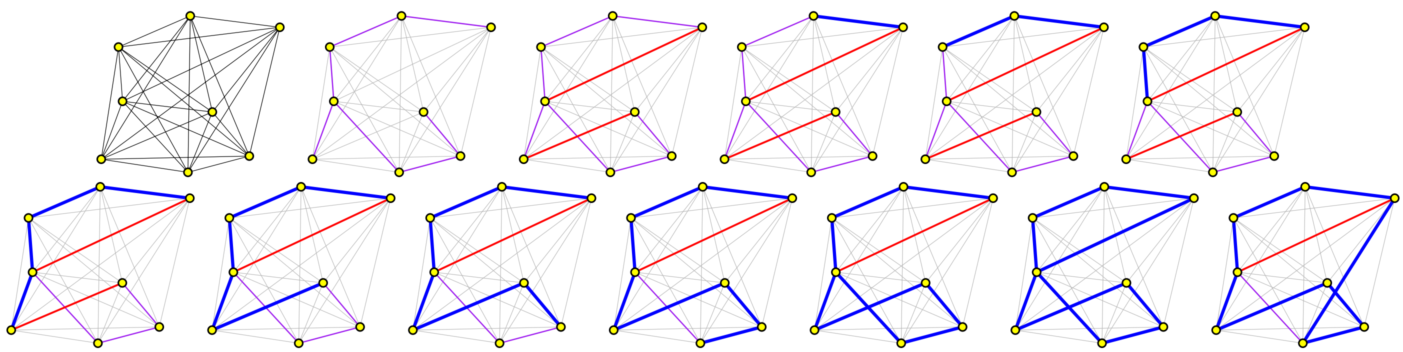
$$\alpha = \sum_{v \in V, d(v) \text{ is odd}} d(v) = \mu - U = \text{even number},$$

since μ and U are both even.

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

8.3.9 3/2-approximation algorithm for TSP

8.3.9.1 Animated!



8.3.10 3/2-approximation algorithm for TSP

8.3.10.1 The result

Theorem 8.3.9. *Given an instance of TSP with the triangle inequality, one can compute in polynomial time, a (3/2)-approximation to the optimal TSP.*

8.3.10.2 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)

8.4.1 Exact fixed parameter tractable algorithm

8.4.1.1 Fixed parameter tractable algorithm for VertexCoverMin.

Computes minimum vertex cover for the induced graph G_X:

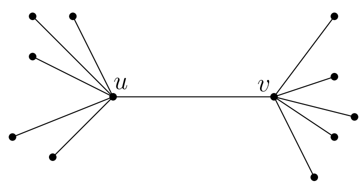
fpVCI (X, β)
    // β: size of VC computed so far.
    if $X = \emptyset$ or G_X has no edges **then return** β
    $\mathsf{e} \leftarrow$ any edge uv of G_X.
    $\beta_1 = \textbf{fpVCI}\Big(X \setminus \{u, v\}, \beta + 2\Big)$
    $\beta_2 = \textbf{fpVCI}\Big(X \setminus \Big(\{u\} \cup N_{\mathsf{G}_X}(v)\Big),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

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

8.4.1.2 Depth of recursion

Lemma 8.4.1. *The algorithm* **algFPVertexCover** *returns the optimal solution to the given instance of* *VertexCoverMin*.

Proof...



8.4.1.3 Depth of recursion II

Lemma 8.4.2. *The depth of the recursion of* **algFPVertexCover**(G) *is at most* α*, where* α *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 α times.
(B) Algorithm picks $N_{\mathsf{G}_X}(v)$ (i.e., β_2). Conceptually add v to the vertex cover being computed.
(C) Do the same thing for the case of β_3.
(D) Every such call add one element of the opt to conceptual set cover. Depth of recursion is $\leq \alpha$.

8.4.2 Vertex Cover

8.4.2.1 Exact fixed parameter tractable algorithm

Theorem 8.4.3. 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:
(A) By lemma, recursion tree has depth α.
(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 α is some parameter that depends on the problem are ***fixed parameter tractable***.

Bibliography

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.