

# Ex. 1

Yuval Gitlitz & Oren Roth

May 11, 2019

1. a) Let  $G = (V, E), w$  be our graph and the weight function on the edges respectively. We will create bipartite graph  $G' = (V \times \{0\}, V \times \{1\}, E')$  where,

$$E' = \{((u, 0), (v, 1)) : (u, v) \in E\}$$

With weight function  $w' : E' \rightarrow R$ , s.t.  $w'((u, 0), (v, 1)) = w((u, v))$ . We will run weighted perfect matching and receive  $M$ . The cycle cover is simply created by taking the edges  $C = \{uv | ((u, n), (v, n)) \in E\}$ .

The algorithm is polynomial: we only constructed a graph which is of size  $2n$ , run the polynomial min cost perfect matching algorithm, and constructed the min cost cover in linear time.

Next, we will prove the algorithm works. For every  $v \in V$ , the algorithm returns  $C$  such that  $v$  is incidence exactly twice, for saturating the vertices  $(v, 0), (v, 1)$  in  $G'$ . For every connected part  $p \in (V, C)$ ,  $p$  is a cycle because all its vertices have degree equals to two. Hence the algorithm returns a cycle cover. Additionally, for every cycle cover,  $M$ , in  $G$  we will show there is a perfect matching in  $G'$  of the same weight. Think of  $M$  as directed. Let  $A = \{((u, 0), (v, 1)) | uv \in E(M)\}$ . All the vertices are saturated in  $A$  because for every  $v \in V$ ,  $d_{out}(v) = d_{in}(v) = 1$ . Additionally,  $|A| = |V| = |V'|/2$  hence  $A$  is a perfect matching. By the way the weight function was constructed,  $w'(A) = w(M)$ . Hence  $C$  is bounded by the min cost cycle cover.

We showed that  $C$  is a cycle cover, and it is bounded by the min cost cycle, hence it is a min cost cycle cover.

b) The algorithm:

- Find min cost cycle cover - denoted by  $C = (c_1, \dots, c_k)$ . For every  $i \in [k]$ , define  $e_i = (u_i, v_i)$  as an edge in  $c_i$ .
- $G \leftarrow \{(u_k, v_1)\}$
- for  $i = 1$  to  $k - 1$  do:
  - $G \leftarrow G \cup (c_i \setminus \{e_i\} \cup \{u_i, v_{i+1}\})$
- $G \leftarrow G \cup (c_i \setminus \{e_i\} \cup \{u_i, v_{i+1}\})$

*Proof.* We will show :

I  $G$  is Hamiltonian cycle.

II  $\text{cost } G$  is at most  $\frac{4}{3}OPT$ .

I We will show the edges in  $G$  admit Hamiltonian cycle. We start by  $v_1$  and go through edges of cycle  $c_1$  until the node  $u_1$  then take the edge  $u_1, v_2$  and continue in this fashion until reaching node  $u_k$ , then taking the edge  $\{(u_k, v_1)\}$  and we are done,

II  $\text{cost}(C) \leq OPT$  because the optimal solution is feasible solution for the cycle cover problem. As each cycle is at least of size of 2 we have that  $k \leq \frac{|V|}{2}$ .  $G$  replace  $k$  edges of size at least 1 with  $k$  edges of size at most 2, then:

$$G \leq \text{cost}(C) + k \leq \text{cost}(C) + \frac{|V|}{2} \leq OPT + \frac{|V|}{2} \leq \frac{3}{2}OPT$$

And the last inequality is due to the fact that the optimal solution visits  $|V|$  edges of weight one at least.

□

c) If every cycle was of size at least 3, then  $k \leq \frac{|V|}{3}$  and we will get

$$G \leq \text{cost}(C) + k \leq \text{cost}(C) + \frac{|V|}{3} \leq OPT + \frac{|V|}{3} \leq \frac{4}{3}OPT$$

2. (a) We build MST  $T = (R, E')$  on the sub graph which includes only nodes in  $R$ . Our algorithm will return  $T$  which is also a feasible solution. We will show  $c(T)$  is at most  $2OPT$ . Let  $\tilde{T} = (\tilde{V}, \tilde{E})$  be the steiner tree which has  $c(\tilde{T}) = OPT$ .  $c(\tilde{T}) = \sum_{v \in \tilde{V}} c(v) + \sum_{e \in \tilde{E}} c(e)$ . In the same way as we showed in class we can have that:

$$2 \cdot \sum_{e \in \tilde{E}} c(e) \geq \sum_{e \in E'} c(e)$$

and since  $\sum_{v \in R} c(v) = 0$  we conclude:

$$c(T) = \sum_{v \in R} c(v) + \sum_{e \in E'} c(e) = \sum_{e \in \tilde{E}} c(e) \leq 2OPT$$

- (b) Assume towards contradiction that there exists a  $(c \cdot \ln |R|)$ -approximation algorithm, we will show how to build a reduction based  $O(\log n)$ -approximation algorithm for set cover and we will arrive to contradiction.

**The reduction algorithm:**

- i. Given  $X = (U, S = \{S_1, \dots, S_m\})$  input for set cover, build the following steiner tree input,  $X' = (G = (V, E), R, w)$  where:

$$\begin{aligned} V &= U \cup S \\ E &= (S \times S) \cup \{(S_i, e_j) : e_j \in S_i, S_i \in S\} \\ R &= U \\ \forall e \in E : \quad w(e) &= 0 \\ \forall S_i \in S : \quad w(S_i) &= w_i \\ \forall e_i \in U : \quad w(e_i) &= 0 \end{aligned}$$

- ii. Run the  $(c \cdot \ln|R|)$ -approximation algorithm on  $G, R, w$  and receive  $T$ .  
iii. Return  $I = \{S_i : S_i \in V(T); S_i \in S\}$ .

We will state two useful lemmas:

**Lemma 1.** *Given  $\tilde{T}$  solution to  $X'$ ,  $\tilde{I} = \{S_i : S_i \in V(\tilde{T}); S_i \in S\}$  is a feasible solution for  $X$ .*

*Proof.* We set  $R = U$ , and because each node in  $R$  is only connected to their sets, hence because  $T$  is connected the only way to saturate all the terminal is by taking sets which include all of them - and we conclude  $I$  is a valid solution to the set cover problem.  $\square$

**Lemma 2.** *Given  $\tilde{T}$  solution to  $X'$ ,  $\tilde{I} = \{S_i : S_i \in V(\tilde{T}); S_i \in S\}$  has the same weight of  $\tilde{T}$  in  $X$ .*

*Proof.* The weight of nodes in  $S$  is the same as the weight of the set cover weights, all the other nodes and edges are of weight zero. Therefore:

$$w(\tilde{T}) = w(\{S_i : S_i \in V(\tilde{T}); S_i \in S\}) = w(\tilde{I})$$

$\square$

**Claim 3.** *The algorithm is  $O(\log n)$ -approximation algorithm for set cover.*

*Proof.* By Lemma 1  $I$  is feasible solution and by Lemma 2 we know  $w(I) = w(T)$ . Denote by  $O_{\text{steiner}}, O_{\text{set-cover}}$  the optimal solutions values of  $X', X$  respectively. We conclude:

$$w(I) = w(T) \stackrel{(i)}{\leq} (c \cdot \ln|R|) \cdot O_{\text{steiner}} = (c \cdot \ln|R|) \cdot O_{\text{set-cover}}$$

(i) is due to our assumption and the last equality is due to Lemma 2.  $\square$

As  $|R| = n$  we found an  $O(\log n)$ -approximation algorithm for set-cover which accordingly to what we learn in class could happen only if  $P = NP$ .

(c)

3. Given graph  $G = (V, E)$  we denote by  $c_{max}^G = \max\{c(e) : e \in E\}$ . Define  $\delta = \frac{\epsilon \cdot c_{max}^G}{|E|}$ . We define two new cost function:

$$\begin{aligned} c' : E &\rightarrow \mathbb{N} \\ c'(e) &= \text{ceil } c(e) \text{ to multiple of } \delta \\ c'' : E &\rightarrow \mathbb{N} \\ c''(e) &= \lceil c(e)/\delta \rceil \end{aligned}$$

We will build a 2-dimensions table  $X = V \times \lceil \frac{|E| \cdot (|V|-1)}{\epsilon} + 1 \rceil$  where  $X[v, j]$  has the min length of a path from  $s$  to  $v$  with cost (by  $c''$ ) is at most  $j$ . We have:

$$X[v, j] = \begin{cases} 0 & \text{if } j = 0, v = s \\ \infty & \text{if } j = 0, v \neq s \\ \infty & \text{if } j < 0 \\ \min_{(u,v) \in E} \{l((u,v)) + X[u, j - c''((u,v))]\} & \text{else} \end{cases}$$

We note that because any min length path is simple (as the lengths are positive) any min length path uses at most  $|V| - 1$  edges and hence the cost (by  $c$ ) of a any min length path is bounded by  $c_{max}^G \cdot (|V| - 1)$ . So we have that the any min length path has a cost, by  $c''$  of at most:

$$\lceil c_{max}^G / \delta \rceil \cdot (|V| - 1) = \frac{|E|}{\epsilon} \cdot (|V| - 1) + 1$$

**Corollary 3.1.** *The min cost (under  $c'$ ) of a path from  $s$  to  $t$  is in one of the cells  $X[t, j]$  for  $1 \leq j \leq t$ . Moreover, it is in the cell of the first  $j$  which holds  $X[t, j] < L$ .*

**Claim 4.** *Let  $O$  be the optimal path with cost  $OPT$  under  $c$ . We have:*

$$\sum_{e \in O} c'(e) \leq OPT + \epsilon \cdot c_{max}^G$$

*Proof.*

$$\begin{aligned} \sum_{e \in O} c'(e) &\leq \sum_{e \in O} (c(e) + \delta) = OPT + \sum_{e \in O} \delta \leq \\ &\leq OPT + |E| \frac{\epsilon \cdot c_{max}^G}{|E|} = OPT + \epsilon \cdot c_{max}^G \end{aligned}$$

□

Let  $c_1, \dots, c_m$  denote the costs of the edges sorted from smallest to largest. We define  $m$  subgraphs of  $G$ :  $G_1, \dots, G_m$  where  $G_i$  has only edges of cost at most  $c_i$ . Now we are ready to present our algorithm-

**FPTAS Algorithm based on linear programming:**

- I for each  $i=1$  to  $m$ :
  - i. build  $G_i$
  - ii. calculate  $c_{max}^{G_i}$
  - iii.  $\delta_i \leftarrow \frac{\epsilon \cdot c_{max}^{G_i}}{|E|}$
  - iv. define  $c'(e) = \lceil c(e)/\delta_i \rceil$
  - v. fill  $X^i[v, j]$  for  $v \in V$ ,  $1 \leq j \leq \frac{|E|}{\epsilon} \cdot (|V| - 1) + 1$  accordingly to  $c''$  and the recurrence relation
  - vi.  $S_i \leftarrow \min_{1 \leq j \leq \frac{|E|}{\epsilon} \cdot (|V| - 1) + 1} \{X^i[t, j]\}$
- II  $A \leftarrow \min_{1 \leq i \leq m} \{S_i\}$
- III  $i \leftarrow \operatorname{argmin}_{1 \leq i \leq m} \{S_i\}$
- IV return Construct\_Solution( $A, i$ )

**Construct\_Solution( $A, i$ )**

- I  $S \leftarrow ()$
- II  $v \leftarrow t$
- III for  $j' = 1$  to  $\frac{|E|}{\epsilon} \cdot (|V| - 1) + 1$  do
  - i. if( $X^i[t, j'] = A$ ):  $j \leftarrow j'$
- IV while  $v \neq s$  do
  - i. for each  $(u, v) \in E$  do
    - A. if  $X^i[v, j] = l((u, v)) + X^i[u, j - c''((u, v))]$ 
      - $j \leftarrow j - c''((u, v))$
      - $v \leftarrow u$
      - $S \leftarrow (u, v) \circ S$
- V return  $S$

**Claim 5.** *The algorithm runs in polynomial time.*

*Proof.* The first loop runs  $m = |V|$  times which is linear in the input size. In each iteration, the first three steps are linear in the input size trivially. Filling a cell in  $X^i$  requires sampling a cell in each row in  $X^i$ . There are  $|V|$  rows and  $O(|E| \cdot |V|/\epsilon)$  cols hence filling  $X^i$  takes polynomial time. Steps VI, II, III can be computed in polynomial with a naive implementation.

Construct\_Solution also takes polynomial time. Steps I, II, III, V takes polynomial time trivially. The loop in IV runs at most  $|V| - 1$  iteration because in each iteration, we recover the another step in the path from  $t$  to  $s$ , and as claimed before, it is a simple path. Each iteration runs in  $O(|E|)$  hence the total time complexity of the loop is polynomial.  $\square$

**Claim 6.** *The algorithm return feasible solution.*

*Proof.* By Corollary 3.1 we know in each  $S_i$  (in row  $i$  in the algorithm) we keep a value of min cost path of length at most  $L$  in the graph  $G_i$  by cost function  $c''$ . It is easy to show in induction that the Construct\_Solution algorithm take a cost of solution  $A$  and index of graph and return a path from  $s$  to  $t$  in  $G_i$  of cost  $A$ . We note that any optimal solution under  $c''$  is also optimal solution under  $c'$ , and hence we know we return a solution of a min cost path from  $s$  to  $t$  in some graph  $G_{i^*}$  of length at most  $L$ . Note that as our algorithm didn't change the length of the edges, we return a path from  $s$  to  $t$  which is also feasible solution in the original graph.  $\square$

**Claim 7.** *The algorithm gives  $1 + \epsilon$  approximation.*

*Proof.* By Claim 4 we know that for any  $c'$  defined over any  $G_i$  we have:

$$\sum_{e \in O} c'(e) \leq OPT + \epsilon \cdot c_{max}^{G_i}$$

Let  $c_{j^*}$  be the maximum cost of an edge in  $O$ .  $O$  uses at least one edge of cost  $c_{j^*} = c_{max}^{G_{j^*}}$  so for the  $c'$  defined over  $G_{j^*}$  we have:

$$\sum_{e \in O} c'(e) \leq OPT + \epsilon \cdot c_{max}^{G_{j^*}} \leq (1 + \epsilon)OPT$$

Let  $S$  be the solution our algorithm return. We have:

$$\sum_{e \in S} c(e) \leq \sum_{e \in S} c'(e) \leq \sum_{e \in O} c'(e) \leq (1 + \epsilon)OPT$$

Where the first inequality is due to the fact we ceil the costs and the second inequality is due to the fact we return min-cost solution under  $c'$ .  $\square$

4. (a) Let  $G = (V, E)$  be a graph we will show the claim holds by induction on  $|V|$ . Base:  $|V| = 0$  trivial. Assume that when  $|V| < n$  the claim holds. Let be  $G$  be a graph with maximum degree  $\Delta$ , with  $|V| = n$  and let  $v \in V$ . Let  $G' = G - v$ .  $|V(G')| = n - 1$  and its maximum degree at most  $\Delta$ . We use the induction hypothesis in order to color  $G'$  with  $\Delta + 1$  colors. Use the same coloring used for  $G'$  in  $G$  for all the vertices except  $v$ . For  $v$ , it has at most  $\Delta$  neighbors and it can be colored using a different color than its neighbors. We used at most  $\Delta + 1$  to color the vertices in  $G$  hence the claim holds.

The algorithm for finding  $(\Delta + 1)$ -coloring will work in a greedy fashion each time choose an uncolored node and color it with an available color. As the maximum degree is  $\Delta$  we know we can do it with  $\Delta + 1$  colors.

Next we will show that a bipartite graph is two colorable in polynomial time. A bipartite graph is two colorable because we can color each disjoint group  $A, B$  in the first and second color respectively. Let  $G = (V, E)$  be a bipartite graph. Using BFS, color each vertex  $v \in V$  using the first color if it is not connected to a vertex with the first color. Otherwise, color it using the second color. The coloring is valid because all for each connected part, all the coloring (except the first) were mandatory for a valid coloring. Since we know a valid coloring exists, the resulted coloring is also valid.

(b) **Algorithm for finding  $O(\sqrt{n})$  – coloring:**

- i. While there exist  $v \in V(G)$  such that  $\deg(v) \geq \sqrt{n}$ 
  - A. Color its neighbors using two colors
  - B.  $G \leftarrow G - N(v)$
- ii. color  $G$  using  $\sqrt{n} + 1$  colors

**Claim 8.** *The algorithm run in polynomial time*

*Proof.* First, let us show that step A can be done in polynomial time. The neighborhood of any vertex  $v$  in the graph can be two colored because each subgraph is three colored, and if we used one color for  $v$ , the neighborhood can be two colored. Two colored subgraph is also a two bipartite graph, hence we can use the previous question to color it using 2 colors.

The loop in step  $i$  runs at most  $\sqrt{n}$  times because each iteration, we remove at least  $\sqrt{n}$  vertices from  $G$ . Additionally, each iteration run polynomial time. Hence, the total run time of step  $i$  is polynomial. Step  $ii$  runs in polynomial time using the algorithm from the previous section.  $\square$

**Claim 9.** *The algorithm is using  $O(\sqrt{n})$  colors*

*Proof.* In each iteration of loop  $i$ , we use two colors. There are at most  $\sqrt{n}$  iteration, hence for step  $i$  we use  $2\sqrt{n} = O(\sqrt{n})$  colors. For step  $ii$  we used  $\sqrt{n} + 1$  colors. For that reason, the total number of colors used by the algorithm is  $O(\sqrt{n})$ .  $\square$

(c) **Algorithm for finding  $O(n^{\frac{2}{3}})$  – coloring:**

- i. While there exist  $v \in V(G)$  such that  $\deg(v) \geq n^{\frac{2}{3}}$ 
  - A. Color  $n^{\frac{2}{3}}$  of his neighbors using  $O(n^{\frac{1}{3}})$  colors.
  - B.  $G \leftarrow G - N(v)$
- ii. color  $G$  using  $n^{\frac{2}{3}} + 1$  colors

**Claim 10.** *The algorithm run in polynomial time*

*Proof.* First, let us show that step *A* can be done in polynomial time. The neighborhood of any vertex  $v$  in the graph can be three colored because each subgraph is 4-colored, and if we used one color for  $v$ , the neighborhood can be three colored. Three colored subgraph of size  $n^{\frac{2}{3}}$  can be colored with using the previous algorithm with  $n^{\frac{1}{3}}$ .

The loop in step  $i$  runs at most  $n^{\frac{1}{3}}$  times because each iteration, we remove at least  $n^{\frac{2}{3}}$  vertices from  $G$ . Additionally, each iteration run polynomial time. Hence, the total run time of step  $i$  is polynomial.

Step  $ii$  runs in polynomial time using the algorithm from section (a).  $\square$

**Claim 11.** *The algorithm is using  $O(n^{\frac{2}{3}})$  colors*

*Proof.* In each iteration of loop  $i$ , we use  $n^{\frac{1}{3}}$  colors. There are at most  $n^{\frac{1}{3}}$  iteration, hence for step  $i$  we use  $n^{\frac{2}{3}}$  colors. For step  $i$  we used  $n^{\frac{2}{3}} + 1$  colors. For that reason, the total number of colors used by the algorithm is  $O(n^{\frac{2}{3}})$ .  $\square$