CS787: Advanced Algorithms **Homework 1 Solutions**

The solutions to homework 1 have two parts: the second part which is contributed by Kexin Li and Xiating Ouyang has solutions for every question in hw1; the first part is an alternative solution for Question 4.

Q4. Linear Programming

Similar to the problem of "Routing to Minimize Congestion" we discussed in class, we can write down the "path LP" and the "flow LP" for this problem respectively as follows.

$$\begin{aligned}
\text{minimize} \quad & \sum_{p \in \mathcal{P}} l_p x_p \\
\text{subject to} \quad & \sum_{p \in \mathcal{P}} x_p = 1 \\
& x_p \in [0,1], \ \forall p \in \mathcal{P}
\end{aligned}$$

In the above path LP formulation, we use $l_p := \sum_{e \in p} l_e$ to denote the length of a s-t-path p, and use $\mathcal{P}$ to denote the set of all s-t-paths. In the ILP version, $x_p \in \{0,1\}$ indicates whether p is adopted; however, in the LP version, $x_p \in [0,1]$ indicates the proportion of p used.

$$\begin{aligned}
\text{minimize} \quad & \sum_{e \in E} l_e y_e \\
\text{subject to} \quad & \sum_{e \in \delta^-(s)} y_e = \sum_{e \in \delta^+(t)} y_e = 1 \\
& \sum_{e \in \delta^-(v)} y_e = \sum_{e \in \delta^+(v)} y_e, \ \forall v \in V \setminus \{s,t\} \\
& y_e \in [0,1], \ \forall e \in E
\end{aligned}$$

In the ILP version, $y_e \in \{0,1\}$ indicates whether e is adopted; however, in the LP version, $y_e \in [0,1]$ indicates the proportion of e used.

By the class discussion, we know that the two LP formulations are equivalent in the sense that both versions will produce the same objective. (But note that only the second flow LP has only polynomial size in the input.) We know that for the solution to their ILP versions gives us the correct result. Next we show that the LP solution also solves the problem exactly. In particular, we will find an integral solution given a fractional solution.

Given a fractional solution S_{flow} to the flow LP, we can easily find an equivalent fractional solution S_{path} to the path LP using a standard flow decomposition (discussed in class). Consider the set of paths $\mathcal{P}' = \{p \in P \mid x_p > 0\}$ in S_{path}, i.e. the set of all the paths that have nonnegative weights. We argue in the following lemma that all paths in $\mathcal{P}'$ have the same length, which means $\sum_{p\in\mathcal{P}} l_p x_p = l_{p^*}$ for any $p^* \in \mathcal{P}'$. Then it immediately follows that the flow LP gives an exact solution to the problem.

Lemma 1 $l_{p_1} = l_{p_2}$ *for any* $p_1, p_2 \in \mathcal{P}'$.

Proof: We prove by contradiction. Suppose, on the contrary and without loss of generality, $l_{p_1} < l_{p_2}$. Then consider another solution S'_{path} where $x'_{p_1} = x_{p_1} + x_{p_2}$, $x'_{p_2} = 0$, and $x'_p = x_p$ for any $p \in P \setminus \{p_1, p_2\}$. It is easy to check that S'_{path} satisfies all the constraints for the path LP, but has a smaller objective as

$$\sum_{p\in\mathcal{P}} l_p x'_p - \sum_{p\in\mathcal{P}} l_p x_p = l_{p_1} x_{p_2} - l_{p_2} x_{p_2} < 0.$$

This is a contradiction to the fact that S_{path} is an optimal solution for the path LP. ■

CS 787 Homework 1

Kexin Li Xiating Ouyang

Problem 1

(a)

Step 4: Find the node N in tree T that contains $N(v)$, the neighbors of v. Add a new node $N' = \{v\} \cup N(v)$ and connect N' to N to obtain T. If $N \subseteq N'$, contract N in T.

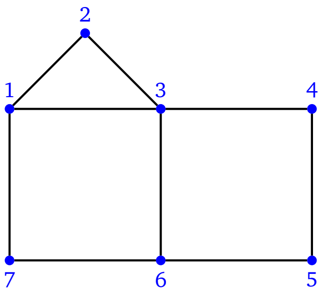
We can always find such an N since in G', the vertices $N(v)$ are connected as a clique, and thus must appear in some node N. It can be found in linear time by iterating all the tree nodes in T.

We stop the recursion when we have a complete graph. This will always happen at $n - 2$ steps (n denotes the number of vertices in G) because the elimination does not change the connected components of G. When we have 2 vertices left. Then it's trivially complete graph K_2.

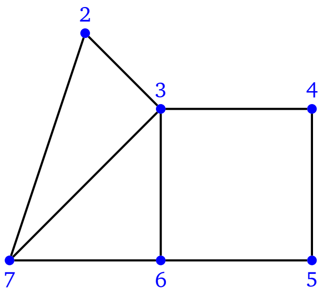
We can always obtain a tree decomposition because a) all the edges in G are covered by using one single node in T to cover all the edges incident on each v for all v in G. b) all the nodes containing v are connected, since N is connected with N', and N' is connected with any other nodes containing v because T' is a valid tree decomposition returned in the recursive step.

(b)

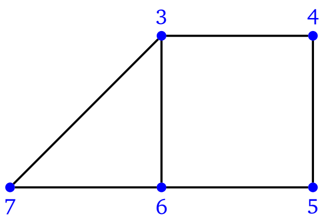
The elimination scheme and the tree decomposition construction procedure are depicted in Figure 1.



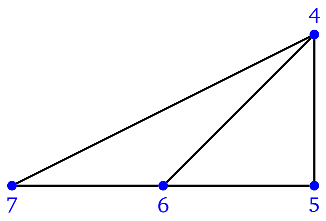
(a) Original graph



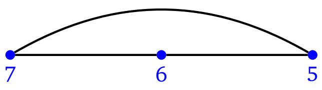
(b) Remove v_1



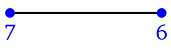
(c) Remove v_2



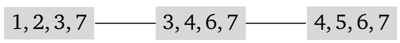
(d) Remove v_3



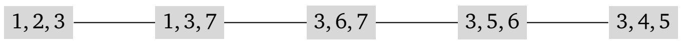
(e) Remove v_4



(f) Remove v_5



(g) Tree decomposition T obtained from vertex elimination with tree width 3



(h) Tree decomposition with minimum tree width 2

Figure 1

(c)

The following lemma is useful.

Lemma 1. *Let* T *be a tree decomposition of a graph* G *where no node whose vertices are properly contained in another node. Then there exists a vertex* v *that appears in exactly one node* N *of* T *and* $\deg(v) \leqslant tw(\mathrm{G})$*. Moreover,* N *is a leaf node in* T*.*

Proof. Suppose for contradiction that every vertex of G appears at least twice. Consider a leaf node N of T and denote its neighbor as N'. Then all vertices $v \in \mathrm{N}$ appears at least twice, and since T is a tree decomposition, $v \in \mathrm{N}'$. Hence $\mathrm{N} \subseteq \mathrm{N}'$, contradicting that the tree decomposition does not contain any node whose vertices are properly contained in another node. Then we have $\deg(v) \leqslant |\mathrm{N}| - 1 \leqslant tw(\mathrm{G})$. ☐

If we have a vertex $v \in V(\mathrm{G})$ with $\deg(v) \leqslant k$ where $tw(\mathrm{G}) = k$, denote the graph obtained from eliminating v and connecting $\mathrm{N}(v)$ as a clique as G'. Then $tw(\mathrm{G}') \leqslant tw(\mathrm{G})$, since we only introduce a clique of size at most $k+1$ and v only appears in one node, so it does not increase the tree width of G.

We prove the following lemma which implies the proposition concerned in problem.

Lemma 2. *For any graph* G *with* n *vertices, there exists an elimination ordering of* $V(\mathrm{G})$ *which results in a tree decomposition of* G *with minimum tree width.*

Proof. When $n = 1$, the graph contains only one vertex v and the elimination ordering returns a tree T of only one tree node $\{v\}$, which is has tree width 0. Hence the lemma holds when $n = 1$. Assume that the lemma holds for all graphs with less than n vertices, consider a graph G with n vertices.

Let T^* be the tree decomposition of G with minimum tree width k. Lemma 1 implies that there exists a vertex v that appears only once in T^*. We eliminate v, and consider the graph G' with $n-1$ vertices after the elimination scheme. Then by induction hypothesis, there exists an elimination ordering σ over $V(\mathrm{G}')$ that produces the tree decomposition T' of G' with minimum tree width. The algorithm hence finds a tree node N' in T' containing $\mathrm{N}(v)$, adds $\mathrm{N} = \{v\} \cup \mathrm{N}'$ and connect it with N', resulting in T. The tree T can be generated by the elimination ordering $\sigma' = v + \sigma$.

It suffices to show that $tw(\mathrm{T}) = k$ since contracting the nodes in T does not increase the tree width. Note that N cannot be contracted, since it contains the unique vertex v. By construction, we have $tw(\mathrm{T}) \geqslant tw(\mathrm{T}')$ since we only add one more node to T'.

Case 1: $tw(\mathrm{T}) = tw(\mathrm{T}')$. Then we have

$$k = tw(\mathrm{G}) \leqslant tw(\mathrm{T}) = tw(\mathrm{T}') = tw(\mathrm{G}') \leqslant tw(\mathrm{G}).$$

Therefore $tw(\mathrm{T}) = k$.

Case 2: $tw(\mathrm{T}) > tw(\mathrm{T}')$. In this case, the tree width of T is precisely $|\mathrm{N}| - 1$. Since $\mathrm{N} = \{v\} \cup \mathrm{N}(v)$ and $\deg(v) \leqslant k$, we have $tw(\mathrm{T}) = |\mathrm{N}| - 1 \leqslant k + 1 - 1 = k$. Therefore

$$k = tw(\mathrm{G}) \leqslant tw(\mathrm{T}) \leqslant k.$$

That is, T also has the minimum tree width.

Therefore T has the minimum tree width k, and can be obtained by the elimination sequence $\sigma' = v + \sigma$. By induction, the lemma holds. ☐

Problem 2

(a)

Consider a wheel graph with $2n+1$ vertices centered at c, where all edges have weight 1, depicted in Figure 3a.

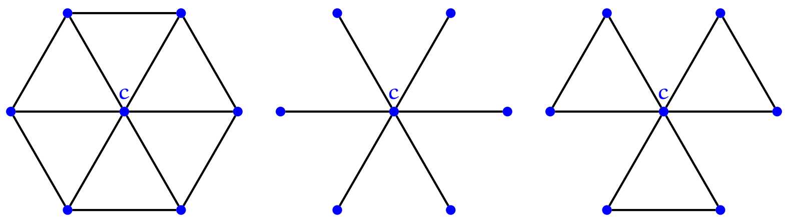
The algorithm would first produce a minimum spanning tree T of G_1, and let T be the star graph centered at c (Figure 3b). The perfect matching on all odd degree vertices then contains three nonadjacent edges, and the total tour cost, as depicted in Figure 3c, is hence $2n + n = 3n$.

However, the minimum tour is to walk along the edge of the wheel and then visit the center, yielding a minimum tour of $2n+1$.

Hence the ratio is

$$\frac{3n}{2n+1} = \frac{3}{2} - \frac{3}{2(2n+1)} \to \frac{3}{2}$$

as $n \to \infty$.



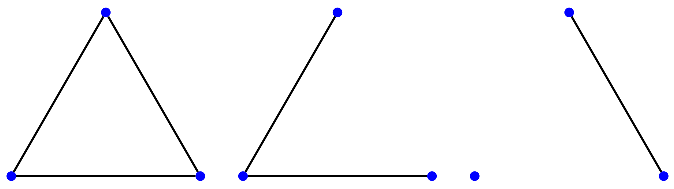
(a) A wheel graph G_1. (b) A minimum spanning tree T of G_1. (c) The final tour on G_1 produced by the algorithm.

Figure 2

(b)

Consider a triangle in which all edges have weight 1. Then the optimal tour has length 3. Whereas the lower bound, the weight of the minimum spanning tree is 2, and twice the weight of the maximum perfect matching is also 2, yielding a lower bound of 2. Hence this example achieves

$$\mathrm{LB} = \frac{2}{3}\mathrm{TSP}.$$



(a) A triangle G_2. (b) A minimum spanning tree T of G_2. (c) The maximum perfect matching of G_2.

Figure 3

Problem 3

(a)

Consider a star graph with $n+1$ vertices centered at c, depicted in Figure 4. The algorithm may pick each edge $\{c, v\}$, and choose to add v to S each time, yielding a vertex cover of size n, whereas the minimum vertex cover is $\{c\}$. Hence the algorithm yields an $\Omega(n)$ ratio on this graph.

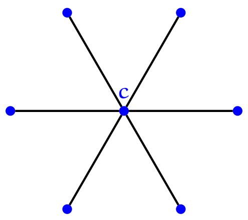


Figure 4

(b)

Consider any vertex v and its closed neighborhood $N[v]$. Let d be the degree of v. Then

$$1 \leqslant |N[v] \cap S| \leqslant d.$$

For any $1 \leqslant k < d$, we have

$$\Pr[|N[v] \cap S| = k] = (1 - \frac{1}{2})^{k-1} \cdot \frac{1}{2} = \frac{1}{2^k},$$

since for the first $k-1$ edges incident to v we picked the other endpoint, and we pick v at the k-th edge. And thus

$$\Pr[|N[v] \cap S| = d] = 1 - \sum_{k=1}^{d-1} \Pr[|N[v] \cap S| = k] = \frac{1}{2^{d-1}}.$$

Thus

$$\begin{aligned} E[|N[v] \cap S|] &= \sum_{k=1}^{d} k \cdot \Pr[|N[v] \cap S| = k] \\ &= \sum_{k=1}^{d-1} \frac{k}{2^k} + \frac{d}{2^{d-1}} \\ &= 2 - \frac{1}{2^{d-1}} \\ &< 2. \end{aligned}$$

Let S^* be the optimal vertex cover on graph G. We show that $S \subseteq \bigcup_{v \in S^*} N[v] \cap S$. Suppose for contradiction that there exists a vertex $u \in S$ such that for all $v \in S^*$, $u \notin N[v]$. Since the algorithm picks u, then the vertex u has at least one neighbor w, and $w \notin S^*$ because $u \notin N[v]$. Then the edge $\{u, w\}$ is not covered by S^*, contradicting that S^* is a vertex cover. Thus $S \subseteq \bigcup_{v \in S^*} N[v] \cap S$, and further,

$$|S| \leqslant \sum_{v \in S^*} |N[v] \cap S|.$$

Therefore,

$$\begin{aligned} E[|S|] &\leqslant E[\sum_{v \in S^*} |N[v] \cap S|] \\ &= \sum_{v \in S^*} E[|N[v] \cap S|] \\ &< \sum_{v \in S^*} 2 \\ &= 2|S^*|. \end{aligned}$$

(c)

- The algorithm in class

  Consider the graph in Figure 5a. The algorithm in class returns all four vertices in the solution achieving a total weight of $2w + 2$, where as the optimal solution has weight 2. Thus the ratio is $w + 1$, which can be arbitrarily large when w is arbitrarily large.

- The first variant of the algorithm

  Consider the graph in Figure 5b with $n + 1$ vertices where w is a parameter. The algorithm could choose S as all the leaf vertices, yielding a total weight of nw, whereas the optimal solution has weight 1.

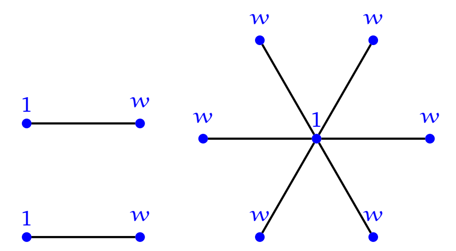
- The randomized algorithm

  Still consider the graph in Figure 5b. The optimal solution is picking the weight 1 vertex.

  The algorithm could pick 1 to n vertices. Suppose the algorithm picks k vertices where $1 \leqslant k < n$, then
  $$\Pr[|S| = k] = (1 - \frac{1}{2})^{k-1} \cdot \frac{1}{2} = \frac{1}{2^k}$$
  with total weight $(k-1)w + 1$. When the algorithm picks n vertices, it either picks n leaf vertices, or picks $n-1$ leaves and the center. For either case, the probability is $\Pr[|S| = n] = \frac{1}{2^n}$, but the total weights are nw and $(n-1)w + 1$ respectively. Therefore,

  $$\begin{aligned} E[w(S)] &= \sum_{k=1}^{n-1} ((k-1)w + 1) \cdot \frac{1}{2^k} + \frac{(n-1)w + 1}{2^n} + \frac{nw}{2^n} \\ &= (1 + w)(1 - \frac{1}{2^n}). \end{aligned}$$

By setting w to be arbitrarily large, we could obtain an arbitrarily large solution, and hence the algorithm does not produce a satisfactory output on this instance.



(a) Instance for original algorithm (b) Instance for second variant and the randomized algorithm

Figure 5

(d)

Let v be any vertex in S^* with $N(v) = \{v_i \mid 1 \leqslant i \leqslant d\}$. Suppose without loss of generality that the algorithm iterates vertices incidental to v in the sequence $v_1, v_2, \dots, v_d$ and produces solution S for input graph G. When $|S \cap N[v]| = k < d$, we have $w(S \cap N[v]) = w(v) + \sum_{i=1}^{k-1} w(v_i)$ with probability

$$\Pr[|S \cap N[v]| = k] = (\prod_{i=1}^{k-1} \frac{w(v)}{w(v) + w(v_i)}) \cdot \frac{w(v_k)}{w(v) + w(v_k)}.$$

When $|S \cap N[v]| = d$, we have either (1) $w(S \cap N[v]) = w(v) + \sum_{i=1}^{d-1} w(v_i)$ with probability

$$P_1 = (\prod_{i=1}^{d-1} \frac{w(v)}{w(v) + w(v_i)}) \cdot \frac{w(v_d)}{w(v) + w(v_d)}$$

or (2) $w(S \cap N[v]) = \sum_{i=1}^{d} w(v_i)$ with probability

$$P_2 = \prod_{i=1}^{d} \frac{w(v)}{w(v) + w(v_i)}.$$

Therefore, the expected weight of $S \cap N[v]$

$$\begin{aligned}
E[w(S \cap N[v])] &= \sum_{k=1}^{d} w(S \cap N[v]) \Pr[|S \cap N[v]| = k] \\
&= \sum_{k=1}^{d} [(w(v) + \sum_{i=1}^{k-1} w(v_i)) \cdot (\prod_{i=1}^{k-1} \frac{w(v)}{w(v) + w(v_i)}) \cdot \frac{w(v_k)}{w(v) + w(v_k)}] + (\sum_{i=1}^{d} w(v_i)) \cdot \prod_{i=1}^{d} \frac{w(v)}{w(v) + w(v_i)} \\
&= 2w(v) \cdot (1 - \frac{w(v)^d}{\prod_{i=1}^{d} (w(v) + w(v_i))}) \\
&< 2w(v).
\end{aligned}$$

By (b), we have $S \subseteq \bigcup_{v \in S^*} N[v] \cap S$, and it follows that

$$\begin{aligned}
E[w(S)] &\leqslant E[\sum_{v \in S^*} w(N[v] \cap S)] \\
&= \sum_{v \in S^*} E[w(N[v] \cap S)] \\
&< \sum_{v \in S^*} 2w(v) \\
&= 2W.
\end{aligned}$$

Thus this algorithm achieves an approximation ratio of 2.

Problem 4

Given a directed graph G, introduce a variable x_v for each vertex $v \in V(G)$, and consider the following linear program:

$$\begin{array}{lll} \text{maximize} & x_t & \\ \text{subject to} & x_u + w(u,v) \geqslant x_v & \forall (u,v) \in E(G) \\ & x_v \geqslant 0 & \forall v \in V(G) \\ & x_s = 0. & \end{array}$$

This linear program has n variables and $m + n + 1$ constraints, where $n = |V(G)|$ and $m = |E(G)|$. Let x_u^* be the length of the shortest path from s to u for each vertex $u \in V(G)$. Since all weights are nonnegative and x_u^* is defined to be the length shortest path, we have that $\{x_u^*\}_{u \in V(G)}$ is a feasible solution, and therefore $x_t = x_t^*$ is feasible in the linear program.

Now consider any path $P = (s, v_1, v_2, \dots, v_k, t)$ connecting vertices s and t with $k \geqslant 0$. When $k = 0$, the path is simply the directed edge (s,t). Denote $\ell(P)$ as the length of the path P, then we have

$$\begin{aligned} x_t &= x_t - x_s \\ &= x_t - x_{v_k} + \sum_{i=1}^{k-1} (x_{v_{i+1}} - x_{v_i}) + x_{v_1} - x_s \\ &\leqslant w(v_k, t) + \sum_{i=1}^{k-1} w(v_i, v_{i+1}) + w(s, v_1) \\ &= \ell(P). \end{aligned}$$

Since the path P is arbitrary, then in particular for the shortest path P^*, we have that

$$x_t \leqslant \ell(P^*) = x_t^*.$$

Our previous argument shows that $x_t = x_t^*$ is also feasible. Hence the solution of the linear program is $x_t = x_t^*$, and it solves the shortest path exactly.