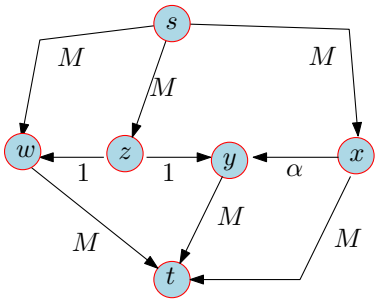
CS 473: Algorithms, Fall 2018
Midterm II: November 13, 2018, 7pm-9pm, DCL 1310

Version: **1.0**

1 FLOW, FLOW, FLOW. (25 PTS.)

[From the discussion section.]

Consider the network flow on the right. Here M is a large positive integer, and $\alpha = (\sqrt{5}-1)/2$. One can verify that (i) $\alpha < 1$, (ii) $\alpha^2 = 1-\alpha$, and (iii) $1-\alpha < \alpha$. We will be interested in running algFordFulkerson on this graph.



1.A. (2 PTS.) What is the value of the maximum flow in this network?

Solution: $2M+1$.

1.B. (3 PTS.) Prove that $\alpha^i - \alpha^{i+1} = \alpha^{i+2}$, for all $i \geq 0$.

Solution:

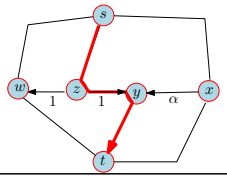
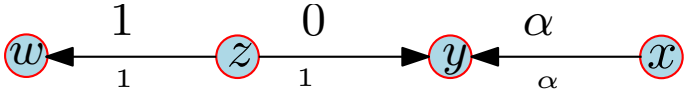
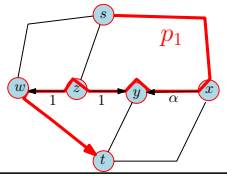
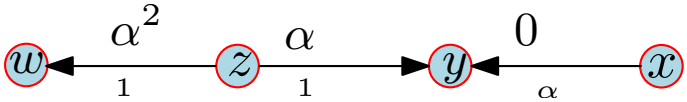
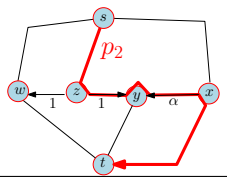
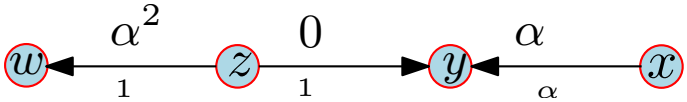
$$\alpha^i - \alpha^{i+1} = \alpha^i(1-\alpha) = \alpha^i\alpha^2 = \alpha^{i+2}.$$

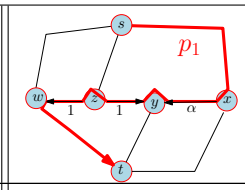
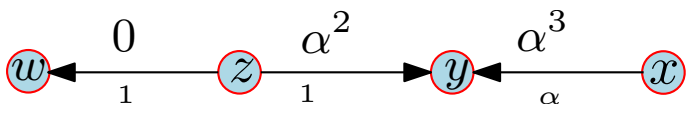
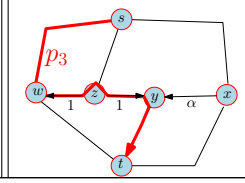
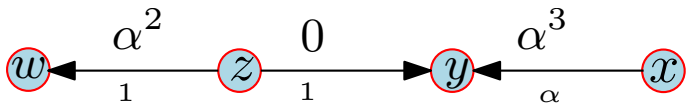
1.C. (10 PTS.) We next specify the sequence of augmenting paths used by the algorithm. In the table below, specify the residual capacities for the designated edges after each step (the residual graph would have also other edges which we do not care about). We start here from the 0 flow.

Please simplify the numbers involved as much as possible, using the relation from (B). Hint: All the numbers you have to write below are either $0, 1$ or α^i, for some i.

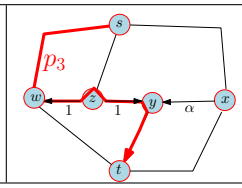
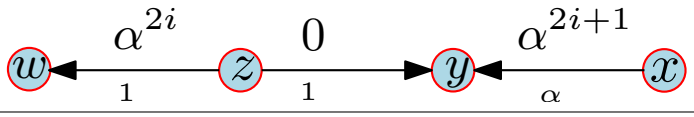
You want to be careful about the calculations - it is easy to make mistakes here.

(See other side of the page for the continuation of this problem).

step	augmenting path	amount pushed	residual capacity
0.		1	
1.		α	
2.		α	

3.		α^2	
4.		α^2	

1.D. (5 PTS.) Imagine that we now repeat steps $1, 2, 3, 4$ above (i.e., we augment along p_1, p_2, p_1, p_3 repeatedly. What would be the residual capacities in the graph after the ith such repetition, starting with the results in the step 4 above?

4i.			

1.E. (5 PTS.) How many iterations would it take this algorithm to terminate, if we repeat the steps in (D)?

Solution:

Infinite – this algorithm never terminates.

2 SPIDERMAN. (25 PTS.)

You are given a directed graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ with n vertices and m edges. There is a set of k base stations $B \subseteq \mathsf{V}$. Each vertex in $\mathsf{V} \setminus B$ has a single message it would like to send to (any) of the base stations. Such a message can be sent along a path in G. In particular, in a *round* of the broadcasting schedule, some of the nodes sends their messages, along some paths that are specified, such that no two paths have a common edge. The problem of course is that one might need several rounds to send all the messages.

Describe an efficient approximation algorithm that minimizes the number of rounds needed till all messages get sent to the base stations.

What is the approximation quality of your algorithm? The lower the better.

What is the running time of your algorithm? (The faster, the better [approximation quality matters more than running time for this question].)

Solution:

The key observation that given a set $U_i \subseteq \mathsf{V} \setminus B$, one can compute the largest subset of vertices that can send simultaneously messages to the base stations. Indeed, set the capacity of every edge in the graph G to 1. Next, add a meta source s and connect it to all the vertices of U_i (again with capacity 1), and connect all the base stations to a meta sink t (with sufficiently large capacity, say, $n+m+1$). Let G_i be the resulting flow network. Compute the maximum flow f_i from s to t in G_i. By the integrality theorem, this is a 0/1-flow, and one can extract $p_i = |f_i|$ edge disjoint paths from s to t in G_i. Clearly, the paths of Π_i can be interpreted as edge disjoint paths from U_i to the vertices of B. Let Π_i be this collection of paths, and let $P_i \subseteq U_i$ be the vertices that were able to send their messages using the paths of Π_i.

Clearly, one can not send more than p_i messages from p_i of the vertices of U_i to B in one round, since if there was such a way, it would correspond to a bigger flow in G_i which is a contradiction.

Now, we use the above procedure in a greedy fashion. We start with $U_1 = \mathsf{V} \setminus B$, and each iteration, we compute Π_i, set it as the set of paths for the ith round, and set $U_{i+1} = U_i \setminus P_i$, and we continue to the next iteration till the set U_{i+1} becomes empty.

Let $u_i = |U_i|$. Let k_{opt} be the number of rounds in the optimal schedule. Observe that $p_i \geq |U_i| / k_{\text{opt}}$. As such, we have that $u_{i+1} \leq (1 - 1/k)u_i \leq n \exp(-i/k)$. In particular, this number is smaller than 1 for $i = \lceil k_{\text{opt}} \ln n \rceil$, which implies that this algorithm would generate a schedule that use at most $O(k_{\text{opt}} \ln n)$, which is a $O(\log n)$ approximation, as desired.

3 Fast clustering. (25 pts.)

[Building on and improving an algorithm seen in class.]

Let P be a set of $n \geq 2$ points in the plane. A ***partial cover*** D is a set of $k \geq 1$ disks $d_1, \ldots, d_k$. Here a disk d_i has radius r and it is centered at some point of P, where r is the distance between some two points of P. The partial cover D is a ***k-clustering of P*** if all the points of P are covered by the disks of D

In class, we saw an algorithm that computes, in $O(nk)$ time, a clustering of P with k disks of radius r such that $r \leq 2r_{\text{opt}}(P, k)$, where r_{opt} is the minimum radius of any cover of P by k disks of the same radius (in particular, $\forall X \subseteq P$ we have $r_{\text{opt}}(X, k) \leq r_{\text{opt}}(P, k)$).

In the following assume that $k = O(\sqrt{n})$.

3.A. (2 pts.) Give a tight upper bound on the number of partial covers, as a function of n and k.

Solution:

$\binom{n}{k}\left(\binom{n}{2} + 1\right) \leq n^{k+2}$.

3.B. (2 pts.) A partial cover D of P is ***bad*** if there are more than $n/2$ point of P that are not covered by the disks of D. Let $R = \{r_1, \ldots, r_t\}$ be a random sample from P of size $t = ck \lceil \log_2 n \rceil$, where c is a sufficiently large constant. Here, each r_i is chosen uniformly and independently from P.

Give an upper bound (as small as possible) on the probability that a specific bad cover D covers all the points of R (as a function of k, n, c).

Solution:

$1/2^t \leq 1/n^{ck}$.

3.C. (2 pts.) Let $\mathcal{B}$ be the set of all bad partial covers of P. Prove that, with high probability, no cover in $\mathcal{B}$, covers all the points of R.

Solution:

By (B), the probability that any bad cover to cover R is at most $1/n^{ck}$. By (A) there are at most n^{2+k} covers. As such, picking $c = 10$, we have that the probability of any bad cover to cover R i s (by the union bound) at most $n^{2+k}/n^{ck} < 1/n^{6k}$.

3.D. (2 pts.) Let R be the random sample as described above. Let D be any k-clustering of R. Prove that D covers at least half the points of P, with high probability.

Solution:

Follows immediately from (C). The k-clustering D is a partial cover, and as such it is in $\mathcal{B}$. By (C), none of the partial covers of $\mathcal{B}$ covers R, with probability $\geq 1 - 1/n^{6k}$, which implies the claim.

3.E. (17 PTS.) Assume, that given a partial cover D with k disks, one can preprocess it, in $O(k \log k)$ time, such that given a query point p, one can decide, in $O(\log k)$ time, whether or not it is covered by the disks of D.

Describe an algorithm, as fast as possible, using the above, that computes a cover of P by $O(k \log n)$ disks of the same radius r, such that $r \leq 2r_{\text{opt}}(P, k)$. For any credit, the running time of your algorithm has to be $o(kn)$, for $k = \Omega(n^{1/4})$.

Solution:

Let $t = ck \lceil \log_2 n \rceil$. The algorithm starts with $P_0 = P$. In the ith iteration, if $|P_{i-1}| \leq k$ then the algorithm sets $R_i = P_{i-1}$. Otherwise, R_i is a sample from P_{i-1} of size t. Let D_i be a k-clustering of R_i computed by the algorithm seen in class, which has running time $O(tk) = O(k^2 \log n)$.

By the above algorithm, we can now preprocess D_i for point-location queries, such that for every point of P_{i-1} we can decide if it is outside D_i. Let $P_i = P_{i-1} \setminus \mathsf{D}_i$ be the computed set. The algorithm now continues to the next iteration, till P_i is empty. Let u be the index of the last iteration of the algorithm.

The i iteration of the algorithm takes $O(k^2 \log n + |P_{i-1}| \log n)$ time. Observe that, with high probability $|P_i| \leq |P_{i-1}|/2$, by the above. As such, we are just going to assume that this holds for all i. This implies, that the overall running time of the algorithm is

$$\sum_{i=1}^{u} O(k^2 \log n + |P_{i-1}| \log n) = O(k^2 \log^2 n + n \log n) = O(n \log n),$$

since $k = O(\sqrt{n})$.

As for correctness, let r_i be the radius of the disks of D_i, and observe that $r_i \leq 2r_{\text{opt}}(R_i, k) \leq 2r_{\text{opt}}(P, k)$. As such, the maximum radius of the disks of $\mathsf{D}_1, \ldots, \mathsf{D}_u$ is at most $r' = \max_i r_i \leq 2r_{\text{opt}}(P, k)$. Clearly, the set of disks of $\mathsf{D} = \cup_i \mathsf{D}_i$ covers all the points of P (otherwise, the algorithm would not terminate).

If we insist that all the disks of D would be of the same radius, we can resize all the disks to have radius r'.

As for the size of D. Since $u \leq \log_2(n/t) \leq \log_2(n/t)$, it follows that $|\mathsf{D}| \leq ku = O(k \log n)$, as desired.

Solution:

More information: One can push the above ideas further. The point-location step can be done, in this case, in linear time using a grid. Furthermore, one can cluster the clustering. This results in linear time algorithm which provides a constant approximation in linear time. Some more work leads to $O(n)$ time algorithm for this problem. See `https://sarielhp.org/p/01/cluster/cluster.pdf` for more details.

4 Hitting set. (25 pts.)

[Similar stuff seen in class.]

Let $(P, \mathcal{F})$ be a given set system – that is, P is a set of n elements, and $\mathcal{F}$ is a collection of m subsets of P. Describe an *efficient* approximation algorithm that computes a subset $X \subseteq P$, as small as possible, such that for any set $S \in \mathcal{F}$, we have that $X \cap S \neq \emptyset$.

Prove the quality of approximation of your algorithm (the quality of approximation has to be as good as possible). What is the running time of your algorithm.

Solution:

Let $\mathcal{F}_0 = \mathcal{F}$, and let $X_0 = \emptyset$. In the ith iteration, the algorithm would find the element in P that is contained in the largest number of sets of $\mathcal{F}_{i-1}$, and let x_i be this element. Clearly, this can be done in $O(nm)$ time. We set $X_i = X_{i-1} \cup \{x_i\}$, and $\mathcal{F}_i = \{S \in \mathcal{F}_{i-1} \mid x_i \notin S\}$. If $\mathcal{F}_i$ is empty, then the algorithm stops and output X_i as the desired hitting set.

Clearly, each iteration can be implemented in $O(nm)$ time, and as such the running time is $O(n^2 m)$, since X_i grows by one in each iteration, and it can be all of P in the worst case. That is, the number of iterations is bounded by n.

As for the quality of approximation. Assume, that there is an optimal hitting set $\{_1, \ldots, _k\} \subseteq P$ of size k. Let ν_i be the number of sets of $\mathcal{F}_{i-1}$ that x_i intersects. Similarly, let $m_i = |\mathcal{F}_i|$. Clearly, there must be an element of the optimal solution that intersects at least m_{i-1}/k sets of $\mathcal{F}_{i-1}$. As such, we have that

$$\nu_i \geq m_{i-1}/k \qquad \text{and} \qquad m_i = m_{i-1} - \nu_i \leq (1 - 1/k) m_{i-1}.$$

Easy calculation shows that

$$m_i \leq (1 - 1/k)^i m \leq \exp(-i/k) m.$$

In particular, for $i = \lceil 1 + k \ln m \rceil$, we have that $m_i < 1$, which implies that the algorithm stopped. That is, the algorithm output a hitting set of size $O(k \ln m)$.

Namely, this algorithm is $O(\log m)$ approximation algorithm.