HW 7 (due Wednesday, at noon, October 24, 2018)

CS 473: Algorithms, Fall 2018 Version: **1.4**

Submission guidelines and policies as in homework 1.

1 (100 PTS.) More about coloring.

1.A. (20 PTS.) Prove that a graph G with a chromatic number k (i.e., k is the minimal number of colors needed to color G), must have $\Omega(k^2)$ edges.

Solution:

Consider a coloring of a graph with k colors. It splits the vertices of the graph into k groups $V_1, \ldots, V_k$, where V_i are all the vertices colored by color i. Note, that the vertices of V_i and V_j must have an edge between them if this is a minimal coloring. Otherwise, one could merge V_i and V_j into a single color class and reduce the number of colors.

Thus, we have at least $\binom{k}{2} = \frac{k^2-k}{2}$ edges. That is, $|\mathsf{E}(\mathsf{G})| \geq \frac{k^2-k}{2} \rightarrow |\mathsf{E}(\mathsf{G})| = \Omega(k^2)$.

1.B. (20 PTS.) Consider a graph G with n vertices and m edges. Consider the greedy algorithm, which orders the vertices in arbitrary order $v_1, \ldots, v_n$. Assume the algorithm colored the first $i-1$ vertices. In the ith iteration, the algorithm assign v_i the lowest color that is not assigned to any of its neighbors that are already colored (assume the colors used are the numbers $1, 2, 3, \ldots$). Provide an upper bound, as low as possible, on the number of colors used by your algorithm as a function of m. You can assume $m \geq n \geq 100$.

Solution:

Consider a coloring of the graph with k colors as computed by the specified algorithm. Let $V_1, \ldots, V_k$ be the partition of the $\mathsf{V}(\mathsf{G})$ into the k color classes. Observe that every vertex of V_i must have an edge to at least one vertex of V_j for $j < i$. As such, we can apply the logic from part (A) to conclude that $k^2/4 \leq m$, if $k > 4$. That is, the graph is colored using $k \leq 2\sqrt{m}$ colors.

1.C. (20 PTS.) Prove that if a graph G is k-colorable, then for any vertex v, the graph $\mathsf{H} = \mathsf{G}_{N(v)}$ is $k-1$-colorable, where $N(v)$ is the set of vertices in G adjacent to v, and $\mathsf{G}_{N(v)}$ is the ***induced subgraph*** on $N(v)$. Formally, we have

$$\mathsf{G}_{N(v)} = (N(v), \{uv \in \mathsf{E}(\mathsf{G}) \mid u, v \in N(v)\}).$$

Solution:

The k-coloring χ for G uses k colors, but the colors that is used to color v, can not be used in $N(v)$, since then there would be a monochromatic edge. It follows that χ provides a $k-1$ coloring of H, since a coloring of a graph, is still a valid coloring of any of its subgraphs.

1.D. (20 PTS.) Describe a polynomial time algorithm that given a graph G, which is 3-colorable, it computes a coloring of G using, say, at most $O(\sqrt{n})$ colors, where n is the number of vertices in G. Hint: First color the low degree vertices in the graph. If a vertex has a high degree, then use (C).

Solution:

We remind the reader that a 2-coloring of a graph can be computed in linear time using BFS.

In a 3-colorable graph, the vertices adjacent to any vertex v are 2-colorable.

```
ApproxGraphColor(G) :
  $i \leftarrow 1$
  while there exists $v \in V(G)$ and $d(v) \geq \sqrt{n}$ do
    color the vertices adjacent to $v$ with color $i$ and $i+1$
    color $v$ with color $i+2$
    delete $v$ and all the neighboring vertices
      along with their edges from G
    $i \leftarrow i+2$
  color the rest of the graph using new $O(\sqrt{n})$ via the greedy algorithm.
```

In the while loop, at least $\sqrt{n}+1$ vertices are deleted in each iteration. Thus, there are $\sqrt{n}$ iterations. In each iteration, we use three new colors. For these vertices, the total number of colors used is at most $3\sqrt{n}$. Each of the remaining vertices has a degree strictly less than $\sqrt{n}$. We can therefore color them by at most $1+\sqrt{n}$ colors. The total number of colors we use in the algorithm is at most $4\sqrt{n}$. Implemented carefully the running time of the algorithm is $O(n+m)$.

1.E. (20 PTS.) (Harder.) Describe a polynomial time algorithm that given a graph G, which is k-colorable, it computes a coloring of G using, say, at most $O(n^{1-2^{-(k-2)}})$ colors. Here k is a small **constant**. What is roughly the running time of your algorithm.

Solution:

The case $k=3$ is handled above. So assume that $k>3$, and assume inductively that you have a polynomial time algorithm ColorIt(H, $k-1$) that can color a graph H that is $k-1$-colorable using $O(n^{1-2^{-(k-3)}})$ colors.

We now describe ColorIt(H, k). Let $\Delta = n^{1-2^{-(k-2)}}$. If a vertex v has degree larger than Δ, then we color it and its neighbors using $O(n^{1-2^{-(k-3)}})$ new colors, by computing the induced graph $H = G_{N(v)}$, and calling ColorIt(H, $k-1$). The remaining graph has degree at most Δ, and it can be colored using $\Delta+1$ colors. The number of colors used by this algorithm is at most

$$\frac{n}{\Delta+1}O(n^{1-2^{-(k-3)}}) + \Delta + 1 = n^{2^{-(k-2)}}O(n^{1-2^{-(k-3)}}) + n^{1-2^{-(k-2)}} + 1 = O(n^{1-2^{-(k-2)}}),$$

because

$$\frac{1}{2^{k-2}} + 1 - \frac{1}{2^{k-3}} = \frac{1}{2^{k-2}} + 1 - \frac{2}{2^{k-2}} = 1 - \frac{1}{2^{k-2}}.$$

Let $f(k,n)$ bye the running time of ColorIt(H, k). Observe that we have that the running time is

$$f(k,n) = O(n+m) + \frac{n}{\Delta+1}f(k-1,n) = O((n+m)^k).$$

This bound is terrible, and one should be able to prove a better running time, but we don't care.

2 (100 PTS.) Greedy algorithm does not work for TSP with the triangle inequality.

In the greedy Traveling Salesman algorithm, the algorithm starts from a starting vertex $v_1 = s$, and in ith stage, it goes to the closest vertex to v_i that was not visited yet.

2.A. (20 PTS.) Show an example that prove that the greedy traveling salesman does not provide any constant factor approximation to the TSP.

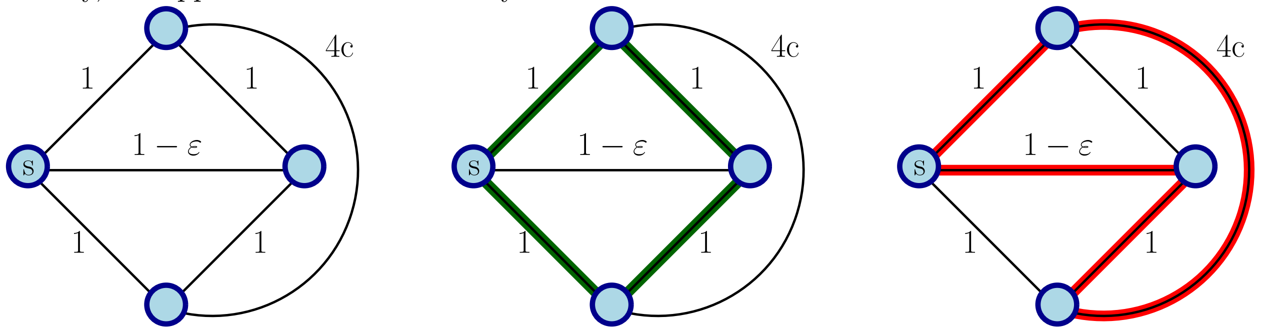
Formally, for any constant $c>0$, provide a complete graph G and positive weights on its edges, such that the length of the greedy TSP is by a factor of (at least) c longer than the length of the shortest TSP of G.

Solution:

Consider the graph in Figure 1. The optimal path has length 4, where as the path found by the greedy algorithm has length $(3 - \varepsilon) + 4c$. As such, the approximation ratio is

$$\frac{(3-\varepsilon)+4c}{4} \geq c.$$

Namely, the approximation is arbitrarily bad.



(A) Input (B) Optimal. (C) Greedy.

Figure 1: Greedy fails for TSP.

2.B. (80 PTS.) (Harder.) Show an example, that prove that the greedy traveling salesman does not provide any constant factor approximation to the TSP with *triangle inequality*.

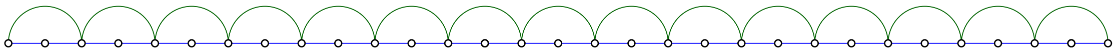
Formally, for any constant $c > 0$, provide a complete graph G, and positive weights on its edges, such that the weights obey the triangle inequality, and the length of the greedy TSP is by a factor of (at least) c longer than the length of the shortest TSP of G. (In particular, *prove* that the triangle inequality holds for the weights you assign to the edges of G.)

Solution:

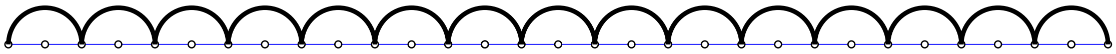
This is a hard problem. Let start with a solution that conveys the idea and is less formal. The next solution would be a more formal restatement of the same solution.

Let ε be a tiny number between 0 and 1 (you can think about $\varepsilon \approx 1/n^3$). Let $v_0, \ldots, v_{n-1}$ be vertices forming a path, where all the consecutive edges are of distance 1. Let this path be π_1. We are no going to create $O(\log n)$ paths, that have roughly the same length as π, such that the greedy TSP would follow all of them. To this end, let π_2 be the path $v_0, v_2, v_4, \ldots, v_{2\lfloor n/2 \rfloor}$. We add the edges of the π_2 to the resulting graph, with weight $1-\varepsilon$ on these edges. Let $\mathsf{G}_2 = \pi_1 \cup \pi_2$, and consider the shortest path distance as the weight for all the edges that are not specified in the graph. Clearly, the greedy TSP starting at v_0, would first traverse π_2, and then it would traverse the remaining vertices of π_1, since the vertices of π_2 are no longer available, it would have to traverse back on the odd vertices, paying distance 2 between each consecutive pair. Thus, we get a greedy walk that visits this path twice.

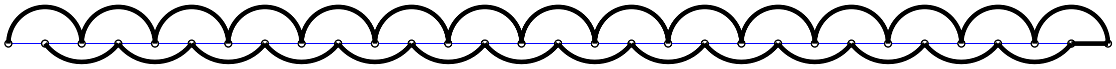
The basic graph:



The greedy walk after we use the even vertices (starting at v_0):



The whole greedy walk after we use the even vertices:



Note, that the edges between consecutive odd vertices do not exist in this graph, but in the complete graph, they exist with distance 2.

The idea is now to generalize this. We start again with a set of vertices $V = \{v_1, \ldots, v_n\}$ which form a path π_1. We now form π_2 as before, as the even vertices. Let $V_2 = V \setminus V(\pi_1)$, and let π_2 be the path visiting the even vertices of V_2 when we order them from left to right. We repeat this process. At the ith iteration, we have a set of remaining vertices V_{i-1} ordered along the path π_1. Let π_i be the path visiting the even vertices of V_{i-1}. We define the length of the edges between two consecutive vertices of π_i to be $2^{i-1} - \varepsilon$. Clearly, we can extract $\log n$ such paths, till all vertices are used. We now connect these paths in their endpoints in the natural way. That is, we connect the right endpoint of π_{2i+1} to the right endpoint of π_{2i+2} using an edge of the same price as the one used by the edges of π_{2i+1}. Similarly, we connect the left endpoints of π_{2i+2} and π_{2i+3} using an edge of the right price. The paths $\pi_1, \pi_2, \pi_3, \ldots$ are $O(\log n)$ paths, all of total length roughly half the length of π_1. The greedy TSP is going to traverse each one of them one by one. As such, overall, the prices of the resulting greedy TSP tour is going to be $\Theta(\text{length}(\pi_1) \log n)$, as desired.

Solution:

Let $v_1, \ldots, v_n$ be the vertices of the graph, and for any two vertices v_i, v_j define an edge between them if $f(i, j) = 1$, where

$$f(i,j) = \begin{cases} 0 & i = j \\ 1 & i + 1 = j \\ 1 & i \text{ is odd, } j \text{ is odd, and } i = j + 2 \\ f(i/2, j/2) & i \text{ is even, and } j \text{ is even} \\ 0 & \text{otherwise.} \end{cases}$$

This graph looks like a skip-list - there are roughly $O(n/2^i)$ edges connecting vertices that their index is divisible by 2^i.

Let $G = (V, E)$ be the resulting graph.

Assign the edges of G the weights according to the following function:

$$d(v_i, v_j) = \begin{cases} 0 & i = j \\ 1 & i + 1 = j \\ (1 - \varepsilon) & i \text{ is odd, } j \text{ is odd, and } i = j + 2 \\ 2d(v_{i/2}, v_{j/2}) & i \text{ is even, and } j \text{ is even} \end{cases}$$

where $\varepsilon > 0$ is an arbitrarily small constant. Note, that this defines the distances only between the vertices of G that are connected by edges. For all other vertices, v_i, v_j, we define their distance to the length of shortest path in this graph that this distance function defines.

Clearly, the shortest path in this graph, is the path that starts from v_1 and goes $v_2, v_3, \ldots, v_n$. This path would cost $n - 1$.

On the other hand, the greedy algorithm, starting in v_1 would first visit all the odd vertices, paying "only" $(n - 1)(1 - \varepsilon)/2$. At this point, the greedy algorithm would go to the closest even vertex and visit all the *alternate* even vertices (because the distance function is defined recursively, and the even vertices are being partitioned to the ones that their index is divisible by 4, and the one which are not). And so on. The greedy algorithm would essentially have to travel from left to right $\log n$ times, to visit all vertices, in each trip, paying "only" $(n - 1)(1 - \varepsilon)/2$ for the pleasure.

Since it is doing it $\Omega(\log n)$ times, it follows that the greedy algorithm, would yield a $\Omega(\log n)$ factor approximation in this case.

Proving that the triangle inequality holds for this distance function, is a tedious but straightforward exercise in induction, and it is omitted.

3 (100 PTS.) Maximum Clique

The max-clique problem has the property that given a low quality approximation algorithm, one can get a better quality approximation algorithm – this question describe this amplification behavior.

Let $\mathsf{G} = (V, E)$ be an undirected graph. For any $k \geq 1$, define $\mathsf{G}^{(k)}$ to be the undirected graph $(V^{(k)}, E^{(k)})$, where $V^{(k)}$ is the set of all ordered k-tuples of vertices from V and $E^{(k)}$ is defined so that $(v_1, v_2, ..., v_k)$ is adjacent to $(w_1, w_2, ..., w_k)$ if and only if for each i $(1 \leq i \leq k)$ either vertex v_i is adjacent to w_i in G, or else $v_i = w_i$.

3.A. (50 PTS.) Prove that the size of the maximum clique in $G^{(k)}$ is equal to the kth power of the size of the maximum clique in G. That is, if the largest clique in G has size α, then the largest clique in $\mathsf{G}^{(k)}$ is α^k, and vice versa.

Solution:

We first prove that a set whose size is the kth power of the size of the maximum clique in G is a clique in $\mathsf{G}^{(k)}$. Then we prove it is the maximum clique in $\mathsf{G}^{(k)}$.

1. Let $C = \{v_1, v_2, \ldots, v_m\}$ be the maximum clique in G. We create a set $C^{(k)} = \left\{(u_1, u_2, \ldots, u_k) \,\middle|\, u_i \in C\right\}$. It's obvious that there is an edge between any two vertices from $C^{(k)}$ in $\mathsf{G}^{(k)}$ since either there is an edge between u_i and u_j in G or $u_i = u_j$, for $1 \leq i, j \leq k$. The size of $C^{(k)}$ is m^k because each u_i can be any vertex chosen from C.
2. Suppose there is a clique $V'^{(k)}$ in $\mathsf{G}^{(k)}$ such that $\left|V'^{(k)}\right| > m^k = \left|C^{(k)}\right|$. There exists an i such that $1 \leq i \leq k$, and set $D_i = \left\{u'_i \,\middle|\, u'_i \text{ is the } i\text{th element in some k-tuple in } V'^{(k)}\right\}$ and $cardin D_i > m$. In other words, there are more than m different vertices appearing at some index i of all the k-tuples in $V'^{(k)}$. Otherwise, $\left|V'^{(k)}\right| \leq m^k$. That means that D_i forms a clique in G and its size is greater than m. This contradicts the assumption that $C = \{v_1, v_2, \ldots, v_m\}$ is the maximum clique in G. Therefore $V^{(k)}$ is the maximum clique in $\mathsf{G}^{(k)}$.

3.B. (50 PTS.) Argue that if there is a c-approximation algorithm for finding a maximum-size clique in a graph, for some constant $c > 1$, then there is a polynomial time γ-approximation algorithm for max-clique, where $\gamma = (c + 1)/2$.

(Hint: Use the first part. Specifically, show an algorithm that is given a clique of size β in $\mathsf{G}^{(k)}$ and outputs a clique of size $\lceil \beta^{1/k} \rceil$ in G, and use it to get a good approximation to the original instance.)

Solution:

Claim 0.1. *Given a clique C of size β in $\mathsf{G}^{(k)}$, the in polynomial time one can output a clique of size $\lceil \beta^{1/k} \rceil$ in G.*

Proof: Given a clique C in $\mathsf{G}^{(k)}$, compute the sets $D_1, \ldots, D_k$, as suggested in the above; that is, D_i is the set of all vertices that appear in the ith coordinate of the vertices of C. Let $d_i = |D_i|$, for $i = 1, \ldots, k$. Clearly, $\beta = |C| \leq \prod_{i=1}^{k} d_i$. In particular, let $j = \arg\max_i d_i$, and observe that $d_j \geq \beta^{1/k}$, which implies $d_j \geq \lceil \beta^{1/k} \rceil$. Now, all the vertices of D_j are connected in G, as can be easily verified, implying the claim.

Assumption. Assume you are given a polynomial time c-approximation Approx(G) that can output a clique C_{G} such that $\frac{|C^*_{\mathsf{G}}|}{|C_{\mathsf{G}}|} \leq c$, where C^*_{G} is the maximum clique in G.

Algorithm. Run Approx$(\mathsf{G}^{(k)})$ – this takes $O\bigl(T\bigl(n^k\bigr)\bigr)$ time, as $\mathsf{G}^{(k)}$ has n^k vertices, and let $C^{(k)}$ be the computed clique. Next, using the algorithm in the above claim, compute a clique in G, and output it.

Analysis. Let $M = \left|C^k\right|$ be the size of the clique computed in $\mathsf{G}^{(k)}$.
Let m_{opt} be the size of the maximum clique in G, and as suggested by the claim extract from the computed clique of $\mathsf{G}^{(k)}$ a clique in G. Let m_{alg} be the size of the computed clique in G. We now have that $\frac{m^k}{M} \leq c$, $m_{\text{alg}} \geq M^{1/k}$. As such,

$$m_{\text{alg}} \geq M^{1/k} \geq \left(\frac{m^k_{\text{opt}}}{c}\right)^{1/k} = \frac{m_{\text{opt}}}{c^{1/k}}.$$

In particular, for k sufficiently large constant, we have that $(1+c)/2 > c^{1/k}$, as desired.