Problem Set 3

CS 331H

Due Tuesday, March 8

1. Consider a weighted, directed graph where all distances lie in $[1, 2)$.

   (a) Show that a variant of BFS computes single source shortest paths in $O(m)$ time.

   (b) Extend your result to $O(mC)$ time and $O(m + C)$ space for distances in $[1, C)$ for any $C > 1$.

2. You have n power plants which, being fancy modern renewable energy designs, do not work all the time (they depend on sun, wind, tides, etc). Each power plant runs from a start time s_i to a finish time f_i, and costs c_i to run (you either run it the whole time or no time). You would like to find a set of power plants to run such that you have power over the entire interval $[0, T]$. What is the minimum cost achievable?

   You may suppose that the s_i and f_i are integers between 0 and $T = O(n)$.

   (a) Give an $O(n \log n)$ algorithm for this problem.

   (b) Now suppose that you can sell off extra power if you have more than one power plant running at a time. At each time step $t \in [T]$, each power plant beyond the first that you run gives you value $v_t \geq 0$ of benefit. Give an $O(n \log n)$ algorithm for this problem.

   In this part you may assume that the cost c_i to run a power plant is larger than the value of the electricity it produces, $\sum_{t=s_i}^{f_i} v_t$.

   (c) Now solve the previous part without assuming that the cost to run a power plant is larger than the value of the electricity it produces.

3. Given a weighted, undirected graph with nonnegative weights, a source s, and a sink t, find the shortest path from s to t and back to s that uses each edge at most once. Aim for $O(m + n \log n)$ time.

4. In this problem, we're going to examine the results of using A^* with different potential functions. Download the code from the course website. It uses Python and requires numpy (version 1.8 or higher) and

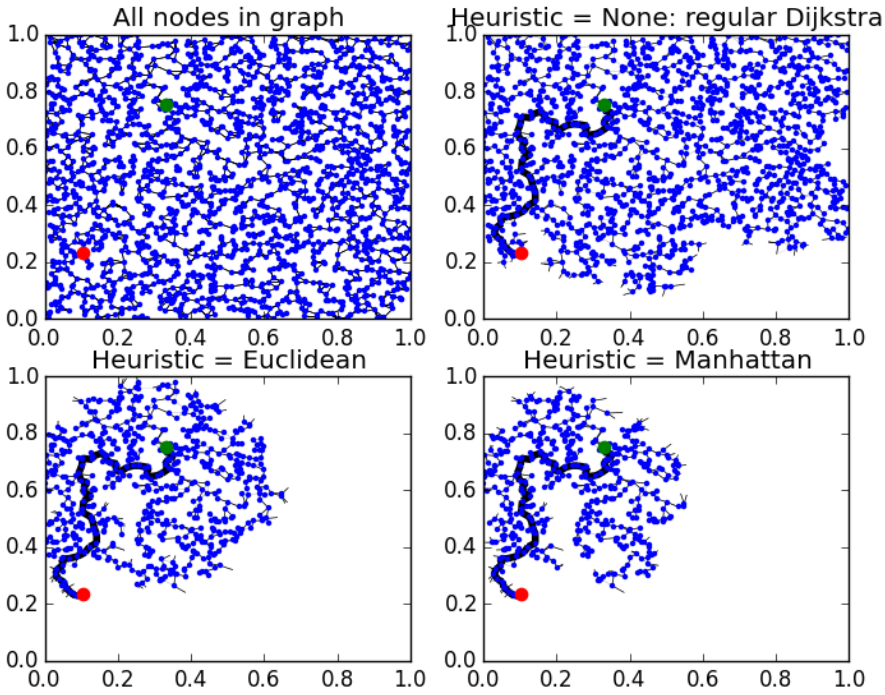

Figure 1: The top left gives the problem instance: shortest paths from the green dot to the red dot. The other plots give the result of running A^* with different potentials. The thick path is the shortest path found; the nodes are the nodes visited; the thin edges are the edges examined, with the edges in the shortest path tree slightly darker.

matplotlib. The program creates a bunch of nodes placed in a square, and links each node to nearby ones, with ℓ_2 distance as the distance. It then picks an s and t and computes the shortest s-t path using A^* and various potential functions. If you run the program, you should see a result like Figure 1, and it should print out the computed distance.

Play around with different potential functions and different size graphs, getting familiar with the program. Then:

(a) If you run the program several times, one of the printed distances will often not match the other two. Why is that?

(b) How do the number of nodes/edges visited compare for the various potentials?

(c) Now, implement the ALT algorithm. The ALT algorithm stands for "A^*, landmarks, and triangle inequality". It's a way of preprocessing the graph to construct a potential function such that fresh queries—new s and t pairs—can be solved very efficiently.

The idea is to choose a small number nodes $l_1, \ldots, l_L$ to be "landmarks" (say, 10-20). We preprocess the graph using full Dijkstra to compute $\text{dist}(u, l_i)$ for all vertices u and landmarks l_i.

On a new query s, t, we observe by the triangle inequality that

$$\text{dist}(u, t) \geq \text{dist}(u, l_i) - \text{dist}(t, l_i)$$

and

$$\text{dist}(u, t) \geq \text{dist}(l_i, t) - \text{dist}(l_i, u)$$

for all landmarks l_i. Hence, if we set

$$\phi(u) := \max_i |\text{dist}(u, l_i) - \text{dist}(t, l_i)|$$

we get an admissible heuristic (where we simplified slightly by using that distances are symmetric).

i. Show that ϕ is in fact consistent (i.e. $\phi(u) - \phi(v) \leq \text{dist}(u, v)$ for all u, v).

ii. Implement the ALT heuristic. With 20 landmarks, it should usually visit less than 1/3 as many nodes as the ℓ_2 heuristic. (You can either choose random landmarks or try to pick them more carefully.)

Print out the resulting graph, comparing the nodes picked by ℓ_2 and ALT. Also include the number of nodes visited by each in several sample runs.