CS1510 Dynamic Programming Problems 21

Rebecca Negley, Sean Myers

October 7, 2011

21. The input to this problem is a sequence of n points $p_1, ...p_n$ in the Euclidean plane. You are to find the shortest routes for two taxis to service these requests in order. Let us be more specific. The two taxis start at the origin. If a taxi visits a point p_i before p_j then it must be the case that $i < j$. (Stop and think about what this last sentence means.) Each point must be visited by at least one of the two taxis. The cost of a routing is just the total distance traveled by the first taxi plus the total distance traveled by the second taxi. Design an efficient algorithm to find the minimum cost routing.

    Solution: If we were to set this up as a tree, in each node there will be two sets: The current points in the T_1 and the points in the second taxi, T_2. Each level will have the possibilities of if you add p_i to each tree in the level above to T_1 or if you add it to T_2. The leaves of the trees will have the solution.

    Now, if we have a tree, we need to prune some branches. If at any point the added distance of both nodes is greater than that of L (L being the distance of just adding all the distance between points $p_1 + ... + p_n$), then it can't possibly be the tree, so prune it. The other pruning rule is if there are two branches that have the same ending nodes. T_1 will have an ending node n_1, and T_2 will have n_2. Both n_1 and n_2 must be equal for it to be considered. If they are equal, choose the minimum distance value among the two, and prune the other. The reason for this, let's say we have candidate s who has equal nodes n_1 and n_2 and has a value v_s. If it has a lower value than r, which has the same nodes n_1,n_2 but r has value v_r where $v_r \leq v_s$, then any solution that s will have at the bottom level can be equivalently replaced with lesser distance value with that of r. Hence, s can be pruned.

    Now that we have all the rules, let's start creating the pseudocode to run it! We know that either T_1 or T_2 can have up to n points (where the optimal distance happens to be optimal by taking all the points). So we will create a table of Taxi[lvl][n][n]. The first n will be the first taxi, second taxi, second. First, we initialize the entire array to infinity distance. We take our information from the level above us. So at each point: Taxi[lvl+1][t1+1][t2] = min (Taxi[lvl+1][t1+1][t2], Taxi[lvl][t1][t2] + distance(point[t1] to point[lvl+1])). Let's break that down a little; if the current Taxi spot is already lower in value, then accept it. Otherwise, take the current point of taxi 1, find the distance between that and point lvl+1, and add it to the current value of Taxi[lvl][t1][t2], to get the new distance. This will fill the table optimally, and then to find the lowest cost, we look at lvl = n, and find the minimum value of the table where t1 + t2 = n (since there are n points, if they do not add up to n, it is an index we do not care about).

    The code will look something like this:

```
Initialize array Taxi[lvl][n][n] to infinity

for(lvl = 0 to n):
    for(t1 = 0 to n):
        for(t2 = 0 to n):
            Taxi[lvl+1][t1+1][t2] = min (Taxi[lvl+1][t1+1][t2], Taxi[lvl][t1][t2] + distance(point[t1], point[lvl+1])
            Taxi[lvl+1][t1][t2+1] = min (Taxi[lvl+1][t1][t2+1], Taxi[lvl][t1][t2] + distance(point[t2], point[lvl+1])
```

This will find it in polynomial $O(n^3)$ time.