Computational Geometry and Topology assignment 3: Ishan Agarwal

Ishan Agarwal

March 2017

A Note

- All angles are in degrees
- Unless otherwise stated, when considering figures in a plane we assume that they are all oriented parallely. For eg. in problem 5 we assume that all the rectangles are such that the sides of unit length of each of them form parallel lines.
- in q.1 I am assuming that a solution exists. In any case it is easy to design algorithms to check that some feasible solution exists.

Problem 1(a)

Consider the following greedy algorithm for finding the minimal hitting set. Order the intervals by there right end points in increasing order. Now pick the point lying in the first interval in this ordering with the largest value. Remove all the intervals hit by this point and repeat this process with the next interval left in the ordering. At each step we choose the lowest interval in the ordering and pick out of the points contained in that interval, the rightmost one. At each step we remove atleast one interval from consideration and so if a feasible solution exists this algorithm is bound to terminate in linear time. (Sorting itself will take O(n log n time). Furthermore we claim that the algorithm gives a feasible solution as we remove intervals from consideration only after including some point in our hitting set that hits them.

Consider a similar algorithm for finding the largest discrete independent set. after sorting the intervals as earlier, pick at each stage the lowest interval in this ordering. Then remove from consideration all those intervals that have any point in common with this interval and carry on. This also terminates for obvious reasons and takes O(n log n) time. The solution is also feasible because we never select intervals with any point in common.

Firstly note that if an interval has any point in common with another interval which has it's right end point to the right of the right end point of the first interval then the two intervals must also share the rightmost point of the point set contained in the first interval. Also if am interval shares the rightmost point from the point set with some other interval then by definition both intervals share some point from the point set. This observation tells us that the two algorithms remove precisely the same set of intervals at each step. Thus at each step that algorithm 1 picks a point the second algorithm picks an interval containing that point. In fact it picks the interval which has that point and is the interval with the earliest right end point to do so. Then both algorithms drop precisely the same set of intervals from consideration and proceed. Thus there is a bijective correspondence between points chosen by the first algorithm and intervals chosen by the second. The intervals are precisely the ones that end earliest that contain those points. Thus the solutions are necessarily of the same size.

Problem 1(b)

In part (a) we have given algorithms for both the hitting set problem and the Discrete independent set problem which have the same solution size. Furthermore we have proved that the solutions given by these algorithms are at least correct; i.e they are viable solutions. We now shall prove the optimality of the solutions. Suppose we have a hitting set H of size h and a discrete independent set D of size d. Every interval in D must be hit by some point of H. But no point exists that is in more than a single interval of D. Hence atleast d many points need to be in H just to hit all the intervals of D. Thus the minimal sized hitting set must be at least as large as the largest discrete independent set thus $OPT_{HS} \geq |DIS|$.

Consider again a hitting set H of size h and a discrete independent set D of size d. Note that with h points every interval has been hit. Suppose the the discrete independent set has more than h intervals. Then we would have more than h intervals and no points in common between them. Thus we would require more than h points to hit all these intervals. Contradiction. Hence the maximum discrete independent set has it's size bounded by the size of the minimum hitting set. Thus $OPT_{DIS} \leq |HS|$

Now consider the algorithms in part (a). They give two feasible solutions to the two problems, each of size S. Then : $OPT_{DIS} \leq |S| and OPT_{HS} \geq |S|$ Thus we get that S is a lower bound for HS solutions and an upper bound for DIS solutions. But the algorithms gave solutions of size S. Thus these solutions must be optimal.

Problem 2

We first sort all the points in increasing order and store them in p[] array. Also sort the intervals according to a[] i.e the left end point. Let b[] contain the right end point and w[] the weights.

Now at each state we can have two indices, the first is the index of the interval and the second is the index of the point. We use the variables interval and point for these two.

So at each state what we have two cases:

The first case is we do not select the current interval.

In the second case what we select the current interval if the point lies inside, also note that if we take the current interval than we can just move the index of the point in the new state forward by the number of points we find in the current interval since they are sorted.

Thus the sub problems are of two kinds either they are the original set of points at that state with one less interval or they are the point set with all the points in the interval under consideration removed and the interval set with that interval removed.

Let us denote the problem instance with Q.Thus the recurrence is, at the stage of dealing with the interval i:
Q(I,P)=Q(I-i,P) or Q(I-i,P- {points in i})

Pseudo code for Dynamic Programming solution using recursion. (note the psuedo code below actually outputs the cost but we can keep track of the intervals being used in the process and output them at the end. There are m intervals and n points. table[][] keeps tracks of already solved problem instances.
Initialize all the variables and set the entries of table[][] to -1.

```
algo(interval,point){

    if(interval == m){
if(point == n) return 0
else return some value showing that we ran out of intervals i.e there is no feasible solution }
if(point == n) return 0
if(table[interval][point] != -1) return table[interval][point]

    //we can make 2 choices either to select this interval or not to select this interval
//if we select this interval, then we also take the points that it covers

    //case 1 : do not take current interval
ans = algo(interval + 1, point)

    if(a[interval] $\le$ p[point] and b[interval] $\ge$ p[point]){
//i.e this point is inside current interval
//case  2 : take current interval, and all the points it contains

    index = point
for(i = point + 1; i ⟨ n ; i++){
if(a[interval] $\le$ p[i]  b[i] $\ge$ p[i])
index = i
else break}
ans = min(ans, algo(interval + 1, index + 1) + w[interval]) }
return table[interval][point] = ans }
```
‘

Checking the time complexity of the above pseudo code gives that it is $O(m * n^2)$ but it can be improved to O(mn) by pre computing what the rightmost point in any given interval is.

Problem 3

We will reduce the minimum dominating set problem to a shortest path problem on a directed graph. Note that Dijkstra's algorithm finds the shortest path between two nodes of a directed graph with the edges having positive weights in O(v^2) time at worst (where v is the number of vertices). Infact for an acyclic directed graphs the shortest path problem can be solved in linear time.
Consider for each interval i, a node n_i. Augment the set of intervals labelled $(1, 2, .., n)$ with two more intervals: 0 and n+1. Let a_i and b_i be the starting and ending points of the i^{th} interval. We assume further that the intervals are indexed by increasing order of a_i. We introduce the intervals 0 and n+1 such that: $b_0 \leq a_1$ and $b_n \leq a_{n+1}$. let us assign weights w_i such that w_i is 1 for all i except 0 and n+1 for which it is 0. Now we will introduce an edge of weight w_i from i to j if and only if 1 of the following conditions is satisfied:

$$a_i \leq a_j \leq b_i \leq b_j$$

$$b_i \leq a_j and \nexists k : b_i \leq a_k \leq b_k \leq a_j$$

note that this forms an acyclic graph as edges can only be from intervals that start earlier to intervals that start later. We will now prove that the shortest path from interval 0 to interval n+1 in such a graph corresponds to a minimum dominating set of the set of augmented intervals. If this is the case we can simply drop intervals 0 and n+1 and get a minimum dominating set for the original set of intervals. Also note that this would take only O(n log n) time as the shortest path problem can be done in linear time and the sorting of intervals by a_i values takes n log n time.

We define a set of intervals corresponding to a path by taking the nodes in the path in the graph as the intervals in our set. If we prove that every path corresponds to a dominating set and that every dominating set corresponds to a path then clearly the shortest path corresponds to the dominating set with fewest intervals.

Firstly consider any dominating set. List the intervals of the set in increasing order of a_i. This listing specifies the path because say there is no edge from a_p to a_{p+1}: then p and p+1 must violate both of the above conditions. Say $a_{p+1} \geq b_p$, then there must be some interval lying between p and p+1 by the second condition. Contradiction. So consider now condition 1: if it is to be violated a_{p+1} must be a subset of a_p. In that case drop a_{p+1}(infact this situation can never arise for a minimum dominating set). The set still remains a dominating set. Proceed and note by the same argument there is a path from a_p and a_{p+2} or yet again a_{p+2} can be dropped. As the number of intervals is finite we see that either the dominating set corresponds to a path or some subset of it does. Note that as the subset has less intervals than the original this is good enough for the purposes of our proof.

We now prove the other direction: We want to show that every path corresponds to a dominating set. Consider some path with intervals in order of increasing a_i. Say that there is some interval k that is not intersected by any intervals in the path. This interval must intersect some interval in the path because if it does not it must lie strictly in between two intervals in the path which is not possible. Contradiction.

Problem 4

Consider a graph with nodes for each rectangle and edges between intersecting rectangles. Consider indicator variables x_i for each rectangle. The variable is set to value 1 if the rectangle numbered i is included in $R^{'}$ and 0 otherwise. Then we have the ILP as follows:

$$minimize \sum x_i$$

$$\forall i (x_i \neq 1) \sum x_j \geq 1$$

where j ranges over all neighbours of the vertex i. The permitted values of x_i are 0 and 1. In fact this ILP corresponds to the problem of choosing a set of vertices in a graph such that any vertex not in this set has some vertex from the set as it's neighbour.

Problem 5

Consider a set of lines L such that each line is perpendicular to the direction in which the heights of all the rectangles (which is a common direction) are. Furthermore the lines in L are such that there are only lines one unit above and one unit below any given line.

We will now assign rectangles to lines. Note that because of he construction of L, every rectangle can be pierced by atmost 1 line because if two lines pierce the same rectangle they must be less than unit distance apart. It is possible for two lines to be coincident with two edges of a rectangle but this can happen only finitely many times and there are infinitely many shifts we can give to the lines to avoid this. Furthermore some line does pierce every rectangle (or atleast co-incides with an edge of it) as the rectangles have unit heights and there are parallel lines every unit distance. Thus we can assign to each rectangle the unique line that pierces through it.

Consider now a partition of L into two disjoint sets of lines, L_1 and L_2. If we give the lines labels from natural numbers, in order, we can specify these sets as the even and odd labeled lines respectively.

For any given line, the independent set problem for rectangles pierced by the line is the same as the independent set problem for intervals on the line. This reduction is easy to see by projecting the rectangles onto the line. However we have seen in class that this problem can be solved in O(n log n) time. Also note that rectangles pierced by even numbered lines cannot intersect so we can solve the independent set problem for R_1 which is the set of rectangles pierced by lines in L_1 exactly in O(n log n) time. Similarly we can solve for R_2 now consider any optimal solution for independent set problem on the set of all rectangles. Clearly greater than or equal to half of the sets in this optimal solution lie in either R_1 or in R_2. Our algorithm solves the problem exactly for R_1 and R_2 and outputs the larger set. Thus our solution must be atleast half the size of the optimal. It runs in time O(n log n), as solving the problem for n intervals on a line takes O(n log n) time.

(as $\sum x_i log(x_i)$is maximized for constant sum of x_i if all x_i are equal.)