Re-weighting techniques

Shobhit Saxena

November 28, 2006

These notes discuss some problems in computational geometry and demonstrate the utility of reweighting techniques in solving them.

1 The Set Cover Problem

1.1 Problem definition

Given a set of points P in a plane. Determine the optimal set of disks which cover all these points. The radii of disks are bounded by a fixed constant value.

This is essentially a set cover problem. Let $|P| = n$. Then if k is the size of the optimal solution (number of disks), a greedy approach exists which can provide an $O(log\ n)$ approximation, i.e., it outputs $O(klogn)$ disks which fully cover all the points of P.

Theorem 1.1 *In geometric settings, the set cover problem can be solved to an $O(log\ k)$ approximation.*

1.2 The basic algorithm

Let D be the set of all disks (Each relevant disk can be defined by 3 points from P). Let $|D| = m$. Let $w(d)$ denote the weight of a disk $d \in D$. For $p \in P$, define $U(p) = \{d | d \in D, d\ covers\ p\}$.

1. Set $W(d) \leftarrow 1, \forall d \in D$
2. Select a random set of disks R from D, of size r, according to the weights of the disks.
3. If R covers the set P, then report the answer as R and exit.
4. Else if $p \in P$ is a point not covered by R, perform step 5 for each $d \in U(p)$.
5. $w(d) \leftarrow w(d) * 2$
6. Continue to step 2.

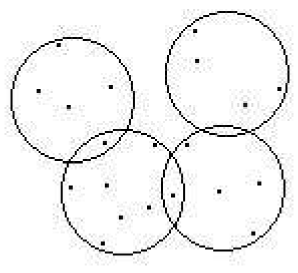

Figure 1: The set cover problem

1.3 Analysis

If the size of the optimal solution is k, let $r = O(k\ log\ k)$. While randomly picking R, we make sure that R is actually an $\epsilon - net$. If R is not an $\epsilon - net$, then the number of iterations go up by a constant factor only. So we assume we always pick an $\epsilon - net$.

Let W_i be the sum of weights of all the disks $d \in D$ after i^{th} iteration. $W_0 = |D| = O(n^3)$. Then,

$$W_{i+1} = W_i + \text{The weight added} \leq W_i + \epsilon W_i = (1+\epsilon)W_i$$

$$\text{Thus, } W_i = (1+\epsilon)^i O(n^3) = O((1+\epsilon)^i n^3)$$

In every iteration, we are increasing the total weight of the optimal solution. This happens in a round-robin fashion for the various disks in the optimal solution.

Let $S_{opt} \subseteq D$ be the optimal solution. We have $|S_{opt}| = k$. If W_i denotes the weight function at the end of i^{th} iteration, then

$$W_i(S_{opt}) = \sum_{d \in S_{opt}} W_i(d)$$

$$= \sum_{d \in S_{opt}} 2^{n_d}, \text{ where } n_d \text{ is the number of times weight of } d \text{ was doubled}$$

$$\geq k\ 2^{i/k}$$

Thus we have the lower and upper bounds on the weight of the optimal solution. We now need a suitable value of ϵ to satisfy the following condition:

$$k 2^{i/k} \leq O((1+\epsilon)^i n^3)$$

With $\epsilon = 1/8k$, this condition is satisfied. The algorithm takes $O(k\ log\ n)$ steps and the size of the solution is $r = O(\frac{1}{\epsilon} log \frac{1}{\epsilon}) = O(k\ log\ k)$.

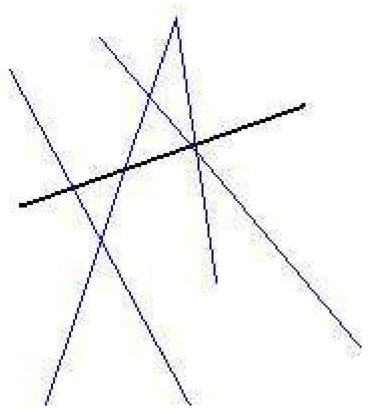

Figure 2: Crossing distance

2 MST Based on Crossing Distance

2.1 Problem definition

Given a set of points P in a plane, $|P| = n$. Define L as the set of $\binom{n}{2}$ lines defined by points in P. The crossing number of an edge is defined as the number of lines from L intersecting this edge. We are looking for a spanning tree with the lowest total crossing number.

2.2 The Algorithm

Let $W(l)$ denote the weight of line $l \in L$. Then the algorithm proceeds as follows:-

1. Initially set $W(l) \leftarrow 1 \; \forall l \in L$
2. Pick two points which have the smallest weighted crossing distance (sum of the weights of the lines that cross the edge formed by these two points)
3. Double the weights of the lines in L which cross this edge.
4. Continue to point 2, until a spanning tree has been obtained.

2.3 Analysis

Lemma 2.1 *Let L be a set of W lines and P be a set of n points. Then there are always two points $p, q \in P$ such that they cross at most $c\frac{W}{\sqrt{n}}$ lines, where c is a constant.*

Proof:
Define $b(x, r)$ as the set of all vertices at a crossing distance at most r from x. We first claim that $|b(x, r)| \geq O(r^2)$. This is because there is always a direction

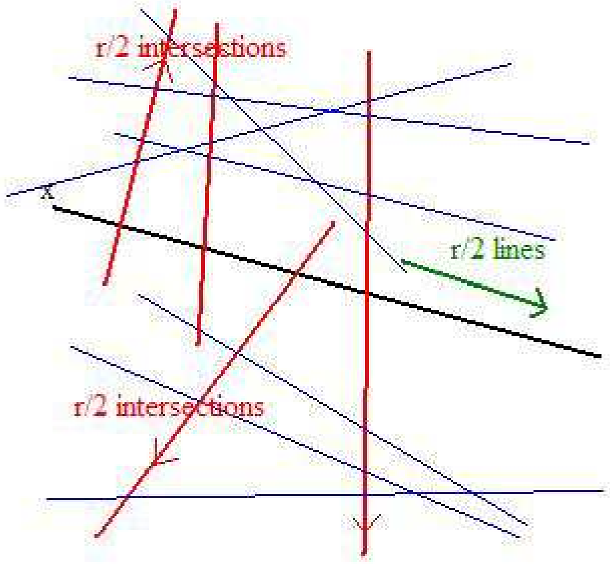

Figure 3: Getting $b(x, r)$

from x which will cross at least $W/2$ lines. We move along this direction and cross $r/2$ lines. For each of these lines, we now move along these lines and expect to cross at least $r/2$ vertices in one of the directions, thus giving $r^2/4$ vertices. A careful observation about picking duplicate vertices puts this number at a value of $r^2/8$.

With W unweighted lines, we have $\binom{W}{2}$ vertices. When $n\frac{r^2}{8} > \frac{W^2}{2}$, then there exist two vertices $p, q \in P$, such that they each have a common vertex x at a crossing distance $< r$ from each of them. This implies that the crossing distance between p and q is then $< 2r$. Clearly then, we have $r = \frac{2W}{\sqrt{n}}$ and so we have two vertices at a distance $< \frac{4W}{\sqrt{n}}$ from each other.

This time we have $W_0 = |L| = O(n^2)$. Now,

$$W_{i+1} = W_i + \text{The weight added} = (1 + \frac{4}{\sqrt{n-i+1}})W_i$$

The $n-i+1$ in the denominator is due to the reason that the number of points keep on reducing by 1 in every iteration.

$$\Rightarrow W_{i+1} \leq exp(\frac{4}{\sqrt{n-i+1}})W_i$$

$$\Rightarrow W_n \leq n^2 exp(\frac{4}{\sqrt{n}} + \frac{4}{\sqrt{n-1}} + \frac{4}{\sqrt{n-2}} + \cdots + \frac{4}{\sqrt{1}}) \approx n^2 exp(O(\sqrt{n}))$$

We now show a property that if we pick any line, it cannot cross a very large number of spanning tree edges. For a line $l \in L$, let $k(l)$ = the number of edges

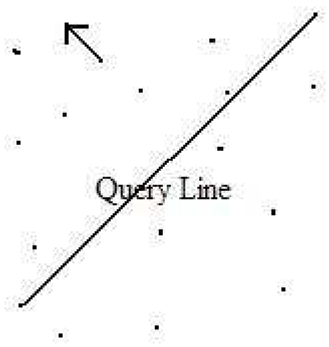

Figure 4: Range searching

it crosses. Then, $W(l) = 2^{k(l)} \leq n^2 exp(O(\sqrt{n})) \Rightarrow k(l) \leq O(log\ n + \sqrt{n}) = O(\sqrt{n})$. Thus a line l cannot cross more than $O(\sqrt{n})$ lines.

3 Range searching

In this section we illustrate an application of minimum crossing number spanning trees, called range searching.

3.1 Problem statement

Given a set of points P in a plane. Given a query line, report the number of points on one size of this line.

3.2 The approach

Define a path Π which spans all the points of P, such that every line crosses it $O(\sqrt{n})$ times. Such a path can be obtained using the minimum crossing number spanning tree, by using shortcut edges, as shown in the figure.

We create a data structure using this path. We arrange the points in Π in a line, in the order of their appearance in Π. We subdivide these points into two equal subsets based on their position in the path (former and latter halves). For each half, we store its convex hull and the number of points the half contains. We do the same process recursively for these halves, to generate a binary tree structure.

When answering a query, we check if the query line intersects the convex hull at the root node. If it doesn't, we determine the side at which all these points lie and accordingly include or exclude their count in the final answer. If, however, the convex hull is intersected by this query line, we go down this node and recursively follow the same procedure with its children nodes.

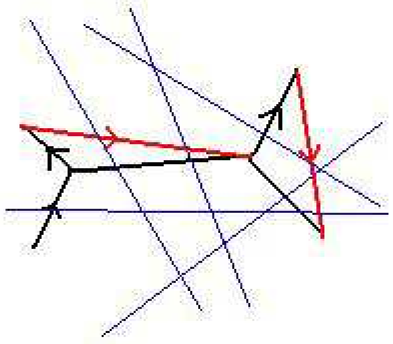

Figure 5: Getting the path Π

An alternative mechanism to do the same is to mark the points making up those edges of Π that are intersected by the query line. Now if a node contains two or more marked points, only then do we go down this node. Otherwise we can directly decide whether to include the count of the number of points at the node into our answer or not.

The total query time is $O(\sqrt{n} log\ n)$, since it takes $O(log\ n)$ time to check a line's intersection with a convex hull and we have to do this $O(\sqrt{n})$ times. The total space requirement is $O(n\ log\ n)$ and the pre-processing time is $O(n\ log^2\ n)$.