CS268: Geometric Algorithms Handout #14
Design and Analysis Original Handout #14
Stanford University Thursday, 24 May 1994

Original Lecture #14: Thursday, May 26, 1994
Topics: Range Searching

1 Introduction

This lecture discusses a smattering of range searching topics which we do not have time to cover in depth. In section 2, we discuss high space solutions to counting queries for halfspace ranges. Section 3 gives a surprising fact about the achievable query time for reporting queries for halfspace ranges. In section 4, we cover the use of multilevel partition structures. Such structures are useful in solving problems which may be decomposed into the conjunction of simple range searching problems — multilevel partition structures for box and simplex queries in E^d are examples. Section 5 discusses the technique of *linearization* as a way to handle non-linear queries. Finally, we conclude in section 6 with the statement of an open problem concerning triangulation of arrangements of hypersurfaces.

2 Counting Queries For Halfspace Ranges Using $O(n^d)$ Space

In this section we consider the counting version of the halfspace range search problem under the assumption that we can afford a search structure of size $O(n^d)$. We are given a set $P = \{p_1, p_2, \ldots, p_n\}$ of n points in E^d and we must answer queries of the form: How many points in P lie above a given query plane π? In dual space, we have a set of n hyperplanes $D(P) = \{\pi_1, \pi_2, \ldots, \pi_n\}$ and a query point $q = D(\pi)$. If we use a duality transform D which preserves the relation of aboveness, then the dual query is of the form: How many hyperplanes $\pi_i \in D(P)$ lie above the point q? Note that the answer to this question is the same for all points q in the same cell of the arrangement $\mathcal{A} = \mathcal{A}(D(P))$. Thus our halfspace range search problem reduces to the problem of point location in $\mathcal{A}$.

In the previous lecture we defined a $\frac{1}{r}$-cutting with respect to a set of n hyperplanes in E^d to be a triangulation of E^d in which each cell of the triangulation is cut by no more than n/r of the hyperplanes. Cuttings will be used to build a search structure for point location in $\mathcal{A}$ which we can query in time $O(\log n)$. Our search structure will be a tree with nodes corresponding to cells of a cutting. Let r be a constant (with respect to n). The root of the tree is the entire space E^d. The children of the root are the cells of a $\frac{1}{r}$-cutting with respect to $D(P)$. Each of these level 1 nodes are cut by no more than n/r planes of $D(P)$. For each level 1 cell C we compute a $\frac{1}{r}$-cutting with respect to the set of planes that intersect C. The cells of such a cutting are the children of C and are at level 2 in our search tree. Note that some of the children of C may not intersect C. These nodes can be pruned, but this is not necessary. The construction of new levels of nodes proceeds in the obvious manner. A cell L is a leaf of the search tree if it is intersected by fewer than some constant threshold of the planes that define the cutting to which L belongs. The height of the search tree is $O(\log_r n)$ since the number of hyperplanes under consideration is decreased by a factor of $1/r$ each time we descend a level in the tree.

How much space does the tree require? Let $S(k)$ denote the size of the search tree built with respect to a set of k hyperplanes. The size $S(n)$ of the whole search tree satisfies the recurrence

$$S(n) = O(r^d) + O(r^d)S(n/r).$$

The first term is due to the fact that a $\frac{1}{r}$-cutting of E^d has $O(r^d)$ cells. The second term reflects the construction of a search tree with respect to $\leqslant n/r$ hyperplanes for each of the $O(r^d)$ cells. A little algebra shows that $S(n) = O(n^{d+\varepsilon})$, where ε is proportional to $\frac{1}{\log r}$. We can make ε as small as we wish by taking r large enough (but still constant with respect to n). The constant hidden in the $O(n^{d+\varepsilon})$ depends on r, however, so we shouldn't take r to be too large. Using other methods, we can get rid of the ε and still obtain $O(\log n)$ query time.

Our search tree can be used to answer queries in $O(\log n)$ time because there are $O(\log_r n)$ levels and we can move from level to level in constant time. We get constant time decent from level l to level $(l+1)$ because each node has $O(r^d)$ children, where r and d are constant with respect to n, and the constant complexity of the cells of a cutting allows us to check in constant time if a point is in a cell. If at each node C of the tree we store the number of hyperplanes which do not intersect C and lie above C, then we can incrementally compute our answer as we walk down the tree. Alternatively, we can store query answers only at the leaves of the tree (assuming each leaf of the tree is contained entirely in one cell of the arrangement $\mathcal{A}$).

3 Reporting Queries For Halfspace Ranges

In a previous lecture we learned that a method which uses space $m \in [n..n^d]$ can achieve a query time which is (up to polylog factors) $O(\frac{n}{m^{1/d}})$. The obvious generalization of this bound to the reporting version of our problem is $O(\frac{n}{m^{1/d}} + k)$, where k is the size of the output. Can we do better? The answer is yes — we can actually get a query time which is (again ignoring polylog factors) $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} + k)$. Where does the $\lfloor d/2 \rfloor$ come from? Strangely enough, the reporting problem is closely linked to the emptiness problem. In the language of the previous section, the upper halfspace of π is empty if and only if $q = D(\pi)$ lies in the upper convex polytope U of $\mathcal{A}$ (since a point in U does not have any planes in $D(P)$ above it). The convex polytope U in E^d has no more than n facets, and thus by the Upper Bound Theorem, it has overall complexity $O(n^{\lfloor d/2 \rfloor})$. The $\lfloor d/2 \rfloor$ in the complexity of the upper polytope U is related to the $\lfloor d/2 \rfloor$ in the achievable query time for the reporting problem.

4 Multilevel Partition Structures

4.1 A Problem in the Plane

Consider the following problem: Given n segments in the plane, count the number of segments cut by a query line. Note that a segment s is cut by a line l if and only if one endpoint of s is in the upper halfspace of l and the other endpoint of s is in the lower halfspace of l. This observation suggests that we build a two level partition structure in which the first level handles upper halfspace queries and the second level handles lower halfspace queries. More precisely, we first build a partition structure to answer upper halfspace queries for the $2n$ segment endpoints. This partition structure responds to a query by returning a set of canonical pieces which contain the endpoints in the upper halfspace of the query line l. Each level one canonical piece Δ contains some subset E_Δ of the $2n$ endpoints. Let F_Δ be the set of endpoints a such that $b \in E_\Delta$ if ab is a segment. For each level one Δ, build a partition structure to answer lower halfspace queries for the points in F_Δ. To answer a query line l, first query the level one structure to get $O(\sqrt{n})$ canonical pieces which contain the endpoints above l. For each level one piece returned, query the corresponding level two structure to determine if the "other" endpoints lie below l.

4.2 Box and Simplex Queries in E^d

A multilevel partition structure can be used to handle box and simplex range queries in E^d. A $d-$dimensional box is the product of d one-dimensional intervals. This suggests the following preprocessing strategy to answer box queries. Build a partition structure to handle queries of the form: Which of the given n points $(x_1,\ldots,x_d)$ have $x_1 \in J_1$ (some interval)? For each canonical piece Δ_1 in this level one structure, build a partition structure to answer the query: Which of the points $(x_1,\ldots,x_d) \in \Delta_1$ have $x_2 \in J_2$? This construction continues in the obvious way until all d levels have been built. To handle a query box $B = I_1 \times I_2 \times \cdots \times I_d$, we successively query the levels of the partition structure. The kth level queries result in kth level canonical pieces which collectively contain the input points which lie in $I_1 \times \cdots \times I_k \times \mathbf{R}^{d-k}$. After querying the dth level, we have the dth level canonical pieces which collectively contain all input points which lie in B. Simplex queries can be handled similarly to box queries since a $d-$dimensional simplex is the intersection of $d+1$ halfspaces. In this case, the multilevel partition structure has $d+1$ levels and the basic query is for halfspaces instead of intervals.

The amazing fact about multilevel partition structures is that they do not blow up too much the space and time bounds of one-level structures in order to answer queries.

5 Non-linear Queries

So far in class we've only consider rectangular, halfspace, and simplex ranges. A range of a more curved nature is a ball in E^d — or we can imagine other ranges bounded by algebraic surfaces. How can we count or report input points which lie in a query ball? One idea is to try to reduce the problem to one of the problems we already know how to solve. The technique of *linearization* attempts to do this. Consider the case $d = 2$ for which queries are discs. A query disc is described by the equation $(x-a)^2 + (y-b)^2 \leqslant r^2$. We can write this out as $x^2 + y^2 - 2ax + a^2 - 2by + b^2 \leqslant r^2$. We linearize this equation by replacing the quadratic term $x^2 + y^2$ by a new variable u. The constraint $u - 2ax + a^2 - 2by + b^2 \leqslant r^2$ is linear in (x, y, u) space and describes the set of points which lie below the plane $(-2a)x + (-2b)y + u = r^2 - a^2 - b^2$. Thus a disc query in the plane can be reduced to a halfspace query in three dimensions. By the preceeding discussion, the counting problem for disc ranges can be solved in $O(n^3)$ space and $O(\log n)$ query time. From the previous lecture, we have a low space (i.e. $O(n)$) solution with query time $O(n^{1-\frac{1}{3}}) = O(n^{\frac{2}{3}})$. Notice, however, that we are ignoring some structure of the linearized problem in E^3. The given points in P are points in the plane E^2. These points become points on our favorite paraboloid $u = x^2 + y^2$ in the three-dimensional problem. We might be able to specialize a halfspace algorithm in E^3 to take advantage of the fact that all the input points lie on a particular $2-$dimensional manifold. We can, for example, obtain a low space algorithm for counting queries for disc ranges which takes only $O(\sqrt{n})$ query time.

6 An Open Triangulation Problem

In obtaining efficient high space solutions of algebraic range queries, a key ingredient is an efficient triangulation of an arrangement of algebraic hypersurfaces of the kind used in the queries. Unfortunately, an optimal solution to this problem is not known. An arrangement of r hypersurfaces in E^d has complexity $O(r^d)$, but the best known triangulation of it is the vertical decomposition which has complexity $O(r^{2d-3}\beta(r))$. In this bound, $\beta(r)$ is a very slowly growing function.