CIS677
Notes for lecture 11:
Quadtree applications,
Arrangements and k-sets

Outline

- More quadtree applications
- Arrangements
  - and duality
  - complexity
- Levels and k-sets
- high order Voronoi diagrams
- Zones
  - complexity
  - zones and BSP's

More quadtree applications

Quadtrees and k-d trees have many practical applications. I'll next describe a few "theoretical" applications of quadtrees: problems where quadtrees give provably good solutions.

I'll be using the term "quadtrees" as in Lecture 10: the children of a quadtree node correspond to 2^d equal-sized subcubes.

Quadtrees and the all-nearest-neighbors problem

Last week, Jon Bentley spoke about the *all nearest neighbors* problem:

> given a set S of n sites, for each site, find its nearest neighboring site.

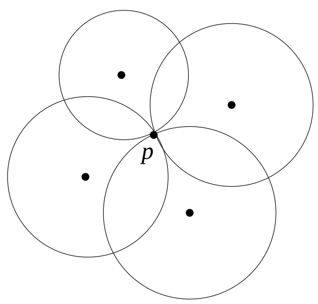
Jon sketched an algorithm needing $O(n \log^{d-1} n)$ time to solve the problem in d dimensions. In Homework 3 one of the problems amounted to showing that the Delaunay edges include all the nearest-neighbor pairs.

Suppose $\rho(S)$ is the *distance ratio* of the sites: the ratio of the distance between the farthest pair of sites to the distance between the closest pair of sites.

Quadtrees can be used to solve the all-nearest-neighbors problem in $O(n \log \rho(S))$ time. The starting point of the algorithm is the following fact.

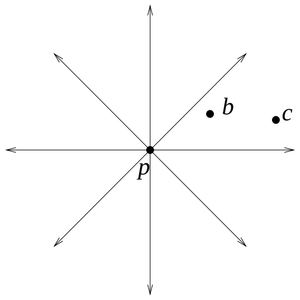
Fact 1 *A site is nearest neighbor to at most $\mathcal{S}_d$ other sites, where $\mathcal{S}_d$ is a constant dependent on the dimension.*

This fact follows from a *sphere-packing* condition; in the figure, each site that has p as nearest neighbor implies that there there is an empty ball touching p.



This fact implies that the total number of nearest-neighbor pairs is $O(n)$, even when there are ties in nearest-neighbor distance.

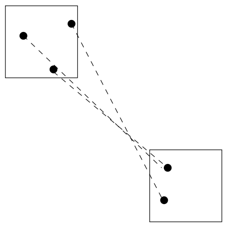
Another way to show this is to use a *fan of narrow cones*: at each site p, place a collection a cones, and consider two other sites in one of the cones.



If the cones are narrow enough, then for any two sites b and c in the same cone, the distance $d(b, c)$ will be less than $\max\{d(p, c), d(p, b)\}$; that is, one of b or c will be farther from p than from the other site.

Such a construction can be used to find a supergraph of the minimum spanning tree as well, and also yields spanner graphs.

Now consider the cells of a quadtree at equal depth in the tree.



The distances between two occupied cells give upper and lower bounds for the nearest neighbor distances for the sites in those cells.

The algorithm will proceed by a series of approximations, considering at each step all the cells of the quadtree of a given size.

Each cell c will keep track of its *approximate nearest neighbor* cells: those cells that might contain nearest neighbors to sites in c.

A crucial

Fact 2 *A given cell is approximate nearest neighbor to at most $\mathcal{S}'_d$ cells, for some value $\mathcal{S}'_d$ dependent only on the dimension.*

This implies that the total number of *approximate* nearest neighbor pairs is $O(n)$.

At a step of the algorithm, the approximate nearest neighbors for the cells are used to find the approximate nearest neighbors of the children of the cells.

This proceeds until all cells contain $O(1)$ sites, where site-by-site processing is done.

The work at each step is $O(n)$, and the depth of the tree is $O(\log \rho(S))$, so the overall work is $O(n \log \rho(S))$.

This idea can be refined to give a worst-case $O(n \log n)$ algorithm.

Related ideas are applicable to approximate matching, minimum spanning trees, and other geometric graph problems.

Quadtrees and the n-body problem

Suppose the sites have mass and gravity, and they move: what paths will they take?

This process can be simulated; at each instant, the forces on a given site, and its prior inertia, give it a velocity in some direction. By letting the sites move subject to those velocities, and then re-evaluating the velocities after a short time, the motion of the sites can be accurately simulated.

The motion of a given site depends on the sum of the gravitational forces acting on it. The problem is to avoid using $\Omega(n)$ time to figure out the forces on each site.

Quadtrees give a way to approximate these forces in near-linear time: roughly, the forces on a site can be viewed as coming from sites grouped into cells by the quadtree; sites in cells that are far away can be "lumped together" to approximate the effect on the site; the approximation is good when the distance to the cell is large compared with the cell's diameter.

Quadtrees and mesh generation

The *mesh generation* problem is to find a triangulation of a region *conforming* to some input data. For example, if the input is point sites, the triangulation should have the sites as vertices.

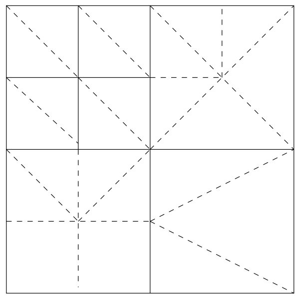
A Delaunay triangulation of the sites might be adequate in this case, but the mesh could have some additional requirement, like having triangles (in the planar case) *well-shaped* or *fat*, with no large or small angles. Such conditions might be needed in numerical analysis applications to give good error bounds.

Suppose the sites have coordinates of the form j/U, where $0 \leq j \leq U$ and U is a power of 2. Then a quadtree for the sites can be built such that the sites are vertices of the leaf cells. Note that $\log U = \Theta(\log \rho(S))$.

Suppose the quadtree for the sites is smoothed, as in Lecture 10. (The text calls this condition *balanced.*) Then a triangulation can be obtained by splitting each leaf cell into a small number of triangles, and each triangle will be fat.

Smoothing, to make leaf cells be near in size to their neighbors, changes the number of nodes of the quadtree by only a constant factor, so the same $O(n \log \rho(S))$ bound holds.

It's not hard to remove the condition on the form of the coordinates of the sites, and have nearly the same bounds.



This idea can be generalized to find triangulations that respect boundaries, where the input data is a collection of line segments, and the conforming triangulation has no triangles that cross the line segments. It can also be generalized to the case of mesh generation in three dimensions.

Arrangements

Arrangement of lines: given a set L of lines, $\mathcal{A}(L)$ is the planar subdivision induced by L, a collection of convex polygons, edges, and vertices.

Arrangements and duality

Suppose we use the duality map taking points

$$p = (p_x, p_y)$$

to lines

$$p^* = \{(x, y) \mid y = p_x x - p_y\},$$

and vice versa, so the slope and intercept of a line determine the x and y coordinates of its dual.

Remember that this correspondence preserves

> incidence: so point p is on line l iff l^* is on p^*,

and

> order: p is above l iff l^* is above p^*.

Consider a line l amidst a set S of points. The line splits the points into those above and below it. The points above have dual lines above l^*, and those below have dual lines below. That is, l^* lies in a cell of the arrangement $\mathcal{A}(S^*)$, and any line that splits the points in the same way lies in the same cell.

Complexity

If every pair of lines meet, the number of vertices is $\binom{n}{2}$; this is clearly an upper bound, met by any *simple* arrangement.

Each line is divided into n edges by the other $n - 1$ lines, so there are n^2 edges.

For each cell of an arrangement, all the points of the cell are either above or below each line: the cell can be described by a sequence of n signs: $(+, +, -, -, -, \ldots)$.

If every possible combination of signs could occur, there would be 2^n cells, but the actual number is much smaller

Lemma 3 *Every bounded cell has a bottom vertex, and no two cells have the same bottom vertex.*

Here we assume that no line is horizontal, or that lexicographic tie-breaking is used.

Proof. Omitted. □

This lemma implies that the number of cells that have a bottom vertex is at most $\binom{n}{2}$.

What about the unbounded cells? Put a horizontal line below all the vertices of the arrangement; each each unbounded cell meets that line in an interval. There are $n+1$ intervals, so there are $n+1$ unbounded cells.

Therefore, the number of cells is at most $\binom{n}{2} + n + 1$.

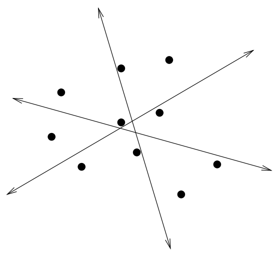
Levels and k-sets

Dually, the lemma says that a set of points can be split up by lines in at most $\binom{n}{2} + n + 1$ different ways, since each cell of the dual arrangement of the points maps to a family of lines cutting up the points in the same way.

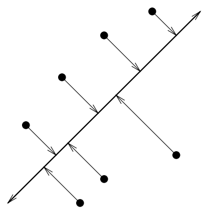
An interesting related combinatorial question is: for given k between 0 and n, how many distinct ways are there for lines to cut a set of points into sets of size k and $n-k$?

Put another way, how many distinct subsets of S are the intersection of S with a halfspace? Such subsets are called k-sets.

The figure shows 3 lines yielding 6 5-sets of a set of 10 points.



Example: Orthogonal regression

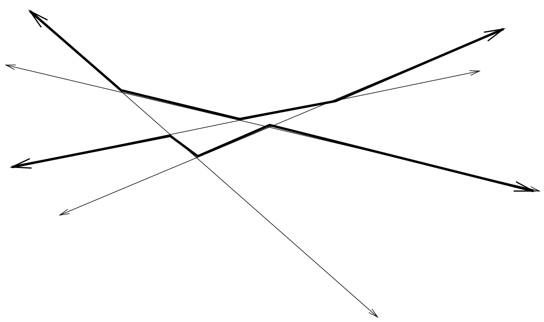


Suppose we want to find a line that minimizes the sum of the *orthogonal* distances to the points of S. Such a line has some nicer statistical properties than least squares.

If a line is translated down by small distance t, then all points above it are t farther way, and all points below it are t closer, so its total orthogonal distance decreases by $(n_- - n_+)t$. Therefore, the best line has $n_- = n_+$, so that the best line splits the set into equal-numbered pieces.

To find the best line, look at all possible *halving* lines, that induce $(n/2)$-sets, and find the best one.

Levels



Consider the set of points with k lines of an arrangement strictly above them. The lower boundary of this set is called a *k-level.*

The 0-level of an arrangement is the boundary of the intersection of the upper halfspaces.

A k-level, in general, is a polygonal line crossing the arrangement, and is the union of a collection of edges of the arrangement.

Higher dimensions

We can define arrangements of hyperplanes in higher dimensions, just as in two dimensions.

The number of vertices is $\binom{n}{d}$, determined by any d hyperplanes. The number of edges is $(n-d+1)\binom{n}{d-1}$, and the number of d-dimensional faces (cells) is

$\sum_{j \le d} \binom{n}{j} = O(n^d)$.

Levels and high order Voronoi diagrams

An *order-k* Voronoi diagram is a partition of the plane induced by a set of sites; all points in a block of the partition have the same set of k sites closest.

A vertex of an order-k Voronoi diagram is the center of a circle with k sites inside and on the circle; this is the analog of a Delaunay disk for an order-1 Voronoi diagram.

Recall that for any set of sites in the plane, there is a set H of halfspaces in 3-space so that the Voronoi diagram of the sites is the projection of the intersection of those halfspaces.

Consider the arrangement of the bounding planes of H. Suppose we project the k-level of that arrangement to the plane. The result is the order-k Voronoi diagram of the sites.

Complexity of levels

The number of k-sets of a set of n points in the plane was proven to be at most $O(n\sqrt{k})$ in 1971, and this bound stood until last year, when Tamal Dey showed that a bound of $O(nk^{1/3})$ holds. Most people think that the current lower bound of $\Omega(n \log k)$ is tight.

There are some related quantities of interest: a j-set is an $(\le k)$-set if $j \le k$; the number of $(\le k)$-sets has a known tight bound.

The bound for $(\le k)$-sets comes by way of a bound on $(\le k)$-facets: a *j-facet* of a set of sites is a hyperplane through d of the sites, such that j of the sites are above the hyperplane. A j-facet with $j \le k$ is a $(\le k)$-facet.

While k-sets correspond, dually, to cells of an arrangement, k-facets correspond to vertices of the k-level.

Theorem 4 *The number of ($\leq k$)-facets of a set of n points in d dimensions is $O(n^{\lfloor d/2\rfloor}k^{\lceil d/2\rceil})$, for $k > 0$.*

In particular, in two dimensions the bound is $O(nk)$, and in three dimensions $O(nk^2)$. This agrees with the $O(n)$ bound for convex hull facets in 2 and 3 dimensions, and with the $O(n^d)$ bound for the number of vertices of the dual arrangement. This agreement is more than coincidence.

Proof. Suppose R is a random subset of S of size $r = n/k$.

A 0-facet of R is a facet of conv(R), and the vertices of that 0-facet induce a j-facet of S for some j. Every 0-facet has that form. We use the bound $O(r^{\lfloor d/2\rfloor})$ on the number of facets of conv(R) to get a bound on the number of ($\leq j$)-facets. We'll show that a ($\leq k$)-facet has a relatively high probability of resulting in a facet of conv(R), and so there can't be too many ($\leq k$)-facet.

The expected number of 0-facets of conv(R) is equal to

$$\sum_j \sum_{T \text{ a } j-\text{facet}} \text{Prob}\{T \text{ a } 0-\text{facet of } R\},$$

as can be shown using indicator functions.

A given j-facet T of S yields a 0-facet of R if and only if:

- the d sites of T are chosen for R;
- the j sites above T are *not* chosen for R.

The probability of the first condition holding is about r^d/n^d, since each site of T has probability r/n of being in R. The probability of the second condition is about

$$(1 - j/n)^{r-d} \approx e^{-rj/n} \geq 1/e,$$

since each of the $r-d$ remaining choices for R have probability about $1-j/n$ of not being among the j of T.

Hence a ($\leq k$)-facet of S yields a 0-facet of R with probability at least r^d/en^d; therefore if M is the number of ($\leq k$)-facets, then

$$Mr^d/en^d = O(r^{\lfloor d/2\rfloor}),$$

so

$$M \leq O(n^d / r^{\lceil d/2 \rceil}) = O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil}).$$

□

A similar argument can be used in a wide variety of contexts: for example, for a set of line segments, let T be a 'j-trapezoid" if it is a trapezoid determined by four or fewer line segments, and meets j segments. Then the number of $(\leq k)$-trapezoids is $O(nk^3)$.

As another example, given a set H of halfspaces, let x be a j-minimum of H if there is some $H' \subset H$ with $|H'| = j$ and with

$$x = \min\{x_d \mid x \in P(H \setminus H')\}.$$

Then the number of $(\leq k)$-minima is $O(k^d)$.

Exact bounds for k-sets

In a few cases, these kinds of arguments can be used to find exact expressions for k-set-like quantities. For example, it can be shown that the number of j-minima, in the above LP example, is j^{d-1}.

Another special case is that of j-facets of points in convex position, that is, points that are vertices of the their convex hull. In 2 dimensions, the number of j-facets of such a set is n: for any point, skip ahead $j+1$ around the hull, and use that other point to define a j-facet.

Bounds for high-order Voronoi diagrams

For point sets in convex position in 3 dimensions, the number of 0-facets is exactly $2n-4$. (General position is also required here.) The probability that a j-facet of S yields a 0-facet of a random subset of size r is exactly $\binom{n-j-3}{r-3} / \binom{n}{r}$; if y_j is the number of j-facets, then for $r = 3 \ldots n$,

$$\sum_j y_j \binom{n-j-3}{r-3} \Big/ \binom{n}{r} = 2r - 4.$$

This is a system of $n-3$ equations in $n-3$ unknowns that can be solved, yielding $y_j = 2(j+1)(n-j-2)$.

Since the Delaunay lifting map of a set of points in the plane yields a collection of points in 3 dimensions in convex position, and the j-facets of that set correspond to vertices of the order j Voronoi diagram, we have a bound on the complexity of these diagrams of $O(nj)$.

Zones of arrangements

The cells in an arrangement can have a wide range of numbers of vertices; in the plane, a cell can have n vertices. On the other hand, since there are $\binom{n}{2}$ vertices and $\binom{n}{2}+n+1$ cells, an average cell has a constant number of vertices.

The region of an arrangement above the k-level is collection of cells; the above bounds show that the total number of vertices of that collection is $O(nk)$ in the plane.

One other case where the complexity of a family of cells can be bounded is the case of a set of cells that meet a given line. Such a set is called a *zone*.

Theorem 5 *In the plane, the total number of vertices of any zone is $O(n)$.*

The number of cells meeting the zone is n, so the average number of vertices of a cell in a zone is $O(1)$.

This *zone theorem* implies that an incremental algorithm can be used to build an arrangement: add lines one by one, walking along the zone of a new line to find the cells that it meets and the new edges and vertices that are incident to it.

Proof. For each line of the arrangement, we count the number of edges of the zone that are *not* contained in that line. Such edges are either not incident to the line, or incident to it. The number of edges that are not incident to the line is at most T_{n-1}, where T_{n-1} is a bound on the zone complexity of an arrangement of $n-1$ lines. The number of edges incident to the line is 4. Hence the number of edges of the zone that are not incident to the line is $T_{n-1}+4$, and the total of these over all lines is $nT_{n-1}+4n$, which we inductively bound by $n(n-1)^2+4n$. In this count, a given edge of the zone is counted $n-1$ times, so the number of edges of the zone is

$$(n(n-1)^2+4n)/(n-1) \leq n(n-1)+4 < n^2.$$

□

The book gives a different proof; this one generalizes to higher dimensions.

Zones and BSP's

As an application of the zone theorem, we can analyze an algorithm for building Binary Space Partition trees in three dimensions.

The input to the algorithm is a set S of non-intersecting triangles in 3 dimensions.

The output is a BSP tree, whose leaves correspond to convex polyhedra, which I'll call *leaf cells.*

The algorithm will use *free splits*: suppose a triangle (by itself, not the plane through it) splits a leaf cell completely in two. Then cutting the leaf cell using that triangle will not split up other triangles: its splitting is for "free."

The algorithm is:

```
If there are any free cuts, do them.
pick triangles one by one at random;
 for each triangle t chosen,
 split all leaf cells that meet t, using aff(t).
```

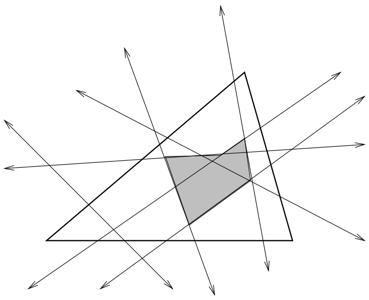
This isn't the most natural algorithm: it's not simply recursive within each leaf cell. However, we can analyze it.

Analysis. We look at the algorithm from the point of view of triangle t_k, to be added at step k.

Each previous triangle t_i may cut t_k by a plane; the intersection of $\text{aff}(t_i)$ with $\text{aff}(t_k)$ is some line ℓ_i; we have a set of the lines $\ell_1 \ldots \ell_{k-1}$ on $\text{aff}(t_k)$.

The arrangement of those lines cuts t_k into a number of pieces, but not so fine a split as the arrangement of the lines.

Consider the effect of triangle t_i on the subdivision induced on t_k up to step i. A piece of that subdivision that is entirely inside t_k corresponds to a leaf cell that is split by t_k: a free split. Such a region of t_k is not cut by ℓ_i, because of the free split done with t_k.



The regions corresponding to free splits are shaded in the figure.
That is, when a cut using t_i is done, the cutting occurs only in the zone of the edges of t_k in $\mathcal{A}(L_i)$), where $L_i \equiv \{\ell_1 \ldots \ell_i\}$. Since ℓ_i is a random member of of L_i, and by the zone theorem, the number of edges of the zone is $O(i)$, it follows that the expected number of edges of the subdivision caused by ℓ_i is $O(1)$.

Therefore, the total number of edges of the subdivision of t_k is $O(k)$, and summing over all triangles, the total number of pieces is $O(n^2)$.

BSP tree lower bounds

While is $O(n^2)$ bound is large compared to the complexity of BSP trees seen in practice, it is tight: there are $\Omega(n^2)$ lower bounds.

Recent work has tried "explain" this by considering restrictions on the input with two properties:

1. The restrictions are "natural," and plausible for real input data.
2. The restrictions allow better bounds to be proven.

For example, if the triangles are "fat," and aren't long and skinny, then near-linear bounds can be shown.