CIS677
Notes for lecture 9: Geometric optimization

Outline

- Convex hulls
  - finishing up analysis of randomized incremental algorithm
  - handling degeneracy
- geometric optimization
  - The minimum-ball problem
  - algorithms:
    * recursive
    * iterative reweighting
    * incremental
  - Linear programming
    * a pivoting algorithm

Analysis of search

Theorem 1 *If the expected number of facets of* $\operatorname{conv}(R_i)$ *is* f_i*, then the expected number of visible base facets for* p_i *is no more than*

$$\sum_{j \leq i} d(d-1) f_j / j(j-1).$$

Proof. Let M be the set of base facets for the sequence

$$p_1 \dots p_{i-1},$$

and let M' be the set of base facets for the sequence

$$p_i, p_1 \dots p_{i-1};$$

that is M' is the set of base facets in the "alternative" history where p_i is added to the random subset first.

The set $M \setminus M'$ is the set of base facets visible to p_i (hence not in M').

The set $M' \setminus M$ is the set of base facets incident to p_i in the alternative history.

We have

$$|M| + |M' \setminus M| = |M'| + |M \setminus M'|,$$

and we want to know $E|M \setminus M'|$, which leads to the problem of bounding

$$E|M| - E|M'| + E|M' \setminus M|.$$

From the analysis of updates, we have

$$E|M| - E|M'| = \sum_{j<i} df_i/i - \sum_{j\leq i} df_i/i = -df_i/i.$$

To bound the expected number of base facets in $M' \setminus M$, we consider the expected number created when p_j is added. Let R'_j denote $R_j \cup p_i$. We want the expected number of facets of R'_j incident to both p_j and to p_i.

Since p_i and p_j are random elements of R'_j, this is

$$d(d-1)f_i/i(i-1),$$

since each facet is incident to $\binom{d}{2}$ pairs of sites, and there are $\binom{i}{2}$ equally likely pairs of sites, and we are averaging over all random subsets R'_j.

Summing up to i gives the theorem. □

From this analysis, the following results are easy:

Theorem 2 *A randomized incremental algorithm can be used to compute the convex hull of a set of n points in d dimensions in $\Theta(n^{\lfloor d/2 \rfloor})$ expected time, and the Delaunay triangulation in $\Theta(n^{\lceil d/2 \rceil})$ expected time. If the point set is such that the expected size of the convex hull of a random subset is $O(i)$ for $i = 1 \ldots n$, then the expected time needed by the algorithm is $O(n \log n)$.*

Degeneracy: all sites in a flat

What if general position fails?

For example, suppose the convex hull of a given planar point set is just a line segment?

More generally, suppose S is a set of sites in E^d but $\dim(S) < d$. How should $\mathrm{conv}(S)$ be represented?

The triangulation can still be used, but with $\dim(S)$-simplices instead of d-simplices.

Here the "triangulation" is a collection of line segments.

For example, if the sites are in E^3 but $\mathrm{aff}(S) = 2$, so the sites all lie in a plane, then the triangulation $TR(S)$ can be a collection of 2-simplices, that is, triangles, in the plane $\mathrm{aff}(S)$.

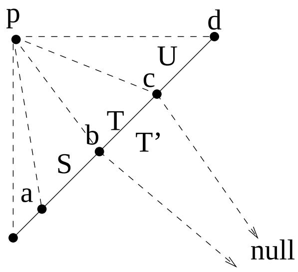
The situation is complicated by the on-line nature of the data structure.

Suppose, in the incremental algorithm,
$\dim(R_i) = 1$, so the sites in R_i are all in a plane, but
$\dim(R_{i+1}) = 2$, so those of R_{i+1} need a full 2 dimensions.

How can the representation be changed?

Here we extend each simplex of $TR(R_i)$, a collection of line segments, to a collection of triangles, each with p_{i+1} as a vertex. Line segments that were neighbors before given neighboring triangles. Line segments that were simplices of the triangulation become facets of the convex hull.

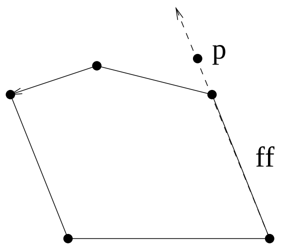
In the figure, the segment T formerly had vertices b and c, and corresponding neighbors U and S. After p is added, T has another vertex, p, and the triangle opposite p is a triangle T', with null peak, representing the convex hull edge $\overline{bd}$.



Degeneracy: some sites in a flat

Suppose some site p is added and, for some facet f, p is in $\mathrm{aff}(f)$; that is, p is in the hyperplane of f. Should f be visible to p or not?

The tie-breaking can go either way, and the triangulation data structure will still work properly.



It seems to be better to break ties against visibility: that way, no simplices of the triangulation are flat.

Optimization and minimum ball

Suppose we have a set S of (point) sites and we want to find the ball $SB(S)$ that:
contains the sites, and
has the smallest possible radius.

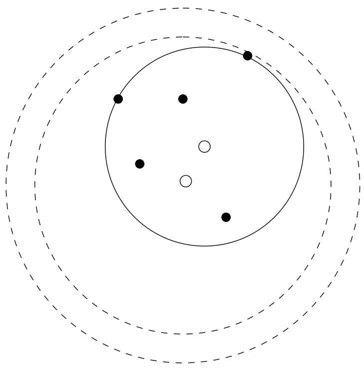
We might do this in order to approximate a collection of objects in a scene by their enclosing sphere, or to find the best target for a bomb...

This is also called the *smallest enclosing sphere* or *minimum spanning sphere* or *1-center* problem.

A minimum ball will have some of the sites on its bounding sphere: otherwise, we can shrink it.

It must have at least two sites on its boundary: otherwise, we can shrink it.

How many sites might be on the bounding sphere? There could be arbitrarily many, but:



Theorem 3 *If ball B contains S, it is the smallest possible iff the center of B is in the convex hull of the sites on the bounding sphere of B.*

Proof. (Omitted; intuitively, suppose the center c is not in the convex hull of the sites S' on the boundary. Then some hyperplane h separates c and S', and moving c toward h would allow a smaller radius ball.) □

From Caratheodory's theorem, we know that if $\operatorname{conv}(S')$ contains c, then some subset $T \subset S'$, of size at most $d+1$, contains c.

Fact 4 *For any set of sites S, there is some subset $T \subset S$ of size at most $d+1$, such that $SB(S) = SB(T)$.*

We also have:

Fact 5 *The minimum ball is unique: there are not two distinct minimum balls.*

Proof omitted; supposing two such distinct balls exist with centers c and c' and radius r, any convex combination of c and c' yields the center of a ball that can have radius less than r.

These facts tell us that when looking for $SB(S)$, there is some set T such that:

- $SB(S) = SB(T)$;
- $|T| \leq d + 1$
- for any $V \subset T$, $SB(V)$ does not contain S.

We'll call such a set a *basis* of S, written basis(S).

We also see a brute force solution: for each subset T' of S of size no more than $d+1$, find the ball inscribed on the sites of T and check if $S \subset SB(T')$.

Suppose we take a subset $R \subset S$, and compute $SB(R)$ somehow. What does $SB(R)$ tell us about $SB(S)$?

Not necessarily much, as far as the center and radius go: these could be very different for the two balls.

However: suppose basis$(S) \subset R$. Then basis$(S) \subset SB(R)$, and the radius of $SB(R)$ is no more than the radius of $SB(T) = SB(S)$, since R is a subset of S. Therefore, $SB(R)$ is unique smallest ball containing basis(S), and therefore is $SB(S)$. We have:

Fact 6 *If $SB(R) \neq SB(S)$, then $S \setminus SB(R)$ contains a member of* basis(S).

If $S \setminus SB(R)$ is large, we haven't really found out much about basis(S).

However, if $|R| = \sqrt{|S|}$, and $R \subset_R S$, then $E|S \setminus SB(R)|$, the expected number of sites outside $SB(R)$, is no more than $d\sqrt{n}$.

That is, we find $SB(R)$ for R a small random subset of S, and we know that one member of basis(S) is in another subset of S that we expect to be small.

To use this fact to solve the problem, suppose we:

> assign $V := S \setminus SB(R)$,
> take another random subset R', and
> compute $SB(R' \cup V)$.

Then $SB(R' \cup V)$ either is $SB(S)$, or $S \setminus SB(R' \cup V)$ contains a member of basis(S), by the same argument as before.

We can show that $S \setminus SB(R' \cup V)$ has expected size $O(\sqrt{n})$ for $|R'| = \sqrt{n}$.

Repeating this approach as often as needed, we have the following algorithm:

Algorithm SB1:

```
if |S| is smaller than C_1 d^2,
    then compute SB(S) by brute force, and return it;
V := {}
repeat
    take a random subset R of S of size C_2 sqrt(|S|);
    B := SB1(R ∪ V);
    if B contains S, return B;
    V := V ∪ S \ B;
forever
```

Here C_1 and C_2 are constants we'll pick later.

Analysis: expected size of $S \setminus SB(R)$.

The probability that random $p \in S \setminus R$ is outside $SB(R)$ is equal to the probability that $SB(R \cup p)$ is different from $SB(R)$, which is the event that $SB(R \cup p)$ is determined by p. The probability that p, a random element of $R \cup p$, is one of the at-most $d+1$ sites determining $SB(R \cup p)$ is no more than $(d+1)/(r+1)$. So the expected number of p in $S \setminus R$ that are outside $SB(R)$ is $(d+1)n/(r+1)$.

A *tail estimate*:
Consider any given subset T' of S of size no more than $d+1$. If $|S \setminus SB(T')| = j$, we are interested in the chance that T' determines $SB(R)$, so that $|S \setminus SB(R)| = j$. This will occur when T' is a subset of R, and no member of $S \setminus SB(T')$ is in R.

Let $|T'| = t$. Of the $\binom{n}{r}$ possible subsets R, how many will satisfy these conditions? R must include T'; the number of possible choices of the remaining members of R is $\binom{n-t-j}{n-t}$.

Therefore, the probability that T' determines $SB(R)$ is

$$\binom{n-t-j}{n-t} \Big/ \binom{n}{r},$$

which can be bounded as

$$(t/n)^d \exp(-jr/n).$$

Now consider the set of all T' with $|T'| \leq d+1$, and $|S \setminus SB(T')| > k$ for some k. The number of such T' is no more than $\binom{n}{d+1} < n^{d+1}$, and so the probability that $|S \setminus SB(R)| > k$ is at most

$$t^d \exp(-kr/n) \leq (d+1)^d \exp(-kr/n)$$

which is no more than $1/n$ for a value of k which is $(d \log n)n/r$.

In short, the expected size of $S \setminus SB(R)$ is no more than $(d+1)n/(r+1)$, and with high probability, the size is no more than $O(\log n)$ times that.

Nothing much changes in these arguments when the quantity of interest is $S \setminus SB(R \cup V)$: V is fixed with respect to the random choice of R.

Changing the algorithm to simplify analysis

The algorithm SB1 is a little too tricky to analyze: we end up with a recurrence involving $|V|$, which is itself a random variable, and the square root of that random variable.

To get something easier to analyze, we change the algorithm to *accept* a random subset R only when $S \setminus B$ is small, say, that

$$|S \setminus B| \leq 2E|S \setminus B|,$$

which will be true with probability 1/2 by Markov's inequality.

If we don't accept the subset, we take another and try again, until we get $|S \setminus B| \leq 2E|S \setminus B|$.

Here's the algorithm:

Algorithm SB2:

```
if |S| is smaller than C_1 d^2,
    compute SB(S) by brute force, and return it.
V := {}
repeat
    repeat
        take a random subset R of S of size sqrt(|S|);
        B := SB(R ∪ V);
        if B contains S, return B;
    until |S \ B| ≤ 2(d+1) sqrt(|S|);
    V := V ∪ S \ B;
forever;
```

The expected number of times to succeed in the inner loop is $1 + 1/2 + 1/4 + \ldots < 2$.

Suppose that $f(n)$ bounds the expected work done when $|S| = n$. Then

$$f(n) \leq O(n) + 2(d+2)f(2(d+1)^2\sqrt{n}),$$

since we expect $2(d+2)$ recursive calls, and each call is on a set of size at most $2(d+1)^2\sqrt{n}$.

The result is that the expected work is $O(n)$: if we expand out the recurrence, we get something terms that are reduced much faster than a geometric series.

A different way to tweak the algorithm for ease of analysis is to use a sample size that is based, not on the current size of S, but on the size that we expect S to have at this point in the recursion. We then ground out of the recursion when the expected size of S is small, not when it really is small. The expected work is still linear with this change.

Iterative reweighting

Iterative reweighting is a different approach to using the basic insight that $S \setminus B$ contains a member of basis(S).

The idea is to focus in on the members of basis(S) by reweighting. That is, suppose each member p of S has a *weight*, $w(p)$, which starts out as 1 for all p.

The set $S \setminus SB(R)$ has a member of basis(S), so highlight the importance of $S \setminus SB(R)$ by doubling the weight of each of its members.

Now pick another random subset, but pick R using the weights: pick a random subset, but count p as $w(p)$ points.

The idea is that eventually the members of basis(S) will have so much weight that they will be in R.

```
for all p ∈ S, w(p) := 1;
repeat
    repeat
        pick a random subset R of S, of size C_3 d^2, using the weights
        compute B := SB(R);
    until |S \ B| ≤ 2(d+1)w(S)/r;
    if S \ B is empty, return B;
    for each p ∈ S \ B, w(p) := 2w(p);
forever
```

What do we mean by a random subset "using the weights"? We can take this to mean that we pick a random subset of a set with $w(S)$ elements, where $w(S) := \sum_{p \in S} w(p)$.

To get such a random subset, we mimic the method we've used for picking random permutations: pick a single random element by picking point p with probability $w(p)/w(S)$, where $w(S) = \sum_{p \in S} w(p)$.

Then, decrement $w(p)$, and pick a random element in the same way again, until r choices have been made.

Analysis

At each iteration of the main loop, some member p of basis(S) has $w(p)$ doubled. This increases the chance that $S \setminus B$ will be empty in the future.

On the other hand, all the members of $S \setminus B$ got their weights doubled.

The general idea is that $w(\text{basis}(S))$ grows faster than $w(S)$, but since basis(S) $\subset S$, eventually either we're done or there's a contradiction.

We have

$$w(\text{basis}(S)) = \sum_{p \in \text{basis}(S)} 2^{d_p},$$

where d_p is the number of doublings of p. After K iterations, we know that

$$\sum_{p \in \text{basis}(S)} d_p >= K,$$

and since $|\text{basis}(S)| \leq d+1$, we have

$$w(\text{basis}(S)) >= (d+1)2^{K/(d+1)},$$

spreading the doublings evenly across basis(S), which grows $w(\text{basis}(S))$ the slowest.

On the other hand, $w(S)$ is increased by at most $2(d+1)w(S)/r$ at each step, and so

$$w(S) \leq n(1+2(d+1)/r)^K \leq n\exp(2(d+1)K/r).$$

Suppose r is chosen to be $4(d+1)/\ln(2)$. if the algorithm doesn't finish, then for K larger than about $3d\log(n)$, $w(S)$ will be smaller than $w(\text{basis}(S))$, a contradiction.

Seidel's algorithm

The above algorithms are simple and linear time, but they beg the question of computing the smallest ball enclosing a *small* set of sites, with fewer than d^2 points.

Next we discuss algorithms that need only a primitive for finding the smallest ball containing at most $d+2$ sites.

For a given $p \in S$, either:

$p \in SB(S \setminus p)$, or $p \in \text{basis}(S)$.

Suppose $p \notin \text{basis}(S)$. Then $\text{basis}(S) \subset S \setminus \{p\}$, so $SB(S) = SB(S \setminus p)$, implying $p \in SB(S \setminus p)$.

So, suppose p is in basis(S). How can we use that information?

The members of basis(S) are on the bounding sphere of $SB(S)$, so we can compute the smallest ball containing $S \setminus \{p\}$ and having p on its boundary. Since p is in basis(S), the result is $SB(S)$.

That is, for $T \subset S$, define $SB(S,T)$ as the smallest ball B containing S, and with T on the boundary of B.

We can now apply this recursively: the general version of the problem is,

given S and T a subset S, find SB(S,T).

SeidelSB(S,T) is:

```
if S = T,
   compute SB(S,T) by brute force, return it;
pick p ∈ S \ T at random;
compute B := SeidelSB(S \ p, T);
if p ∈ B then return B;
return SeidelSB(S, T ∪ p); /* p ∈ basis(S) */
```

The first call to the algorithm is SeidelSB$(S, \{\})$.

Analysis

To bound the running time for this algorithm, we first observe that the number of sites determining $SB(S,T)$ is at most $d+1-|T|$.

The probability that p fails to be in $SB(S \setminus p, T)$ is therefore $(d+1-|T|)/n$, since $r+1=n$. Letting $f(|S|,|T|)$ denote the number of primitive computations, we have

$$f(n,m) = f(n-1,m) + (d+1-m)f(n-1,m+1)/n;$$

we see that $(d+1-m)!n$ bounds $f(n,m)$, since

(d+1-m)!(n-1) + (d+1-m)(d-m)!(n-1)/n $\leq (d+1-m)!(n-1)(1+1/n) \leq (d+1-m)!n$.

This algorithm is worse, in its dependence on d, than the previous ones, but some seemingly small changes lead to very interesting results for other problems.

Linear programming

The problem is:

> *Given a set H of halfspaces and a vector c, find the point $x \in P(H)$ minimizing $c \cdot x$.*

Since we can rotate the coordinate system, there's no loss in generality by assuming $c = [0 \ \ldots \ 0 \ 1]$, so that $c \cdot x = x_d$: we want to find the lowest point in $P(H)$.

Example: plane fitting.
Suppose we are given a set S of points, and we want to find a nonvertical plane L that minimizes the maximum vertical distance to each point in S. That is, we want L minimizing

$$\max_{p \in S} \operatorname{dist}(p, L),$$

where $\text{dist}(p, L)$ is the vertical distance between p and L. Write L as

$$L = \{[x\ z] \mid z = b \cdot x\},$$

and each $p \in S$ as $[p_x\ p_z]$, where $p_x \in E^{d-1}$, and $\text{dist}(p, L) = |p_z - b \cdot x|$, so we want a point $[x\ t]$, where x is in E^d, such that t is as small as possible, and

$$\begin{aligned} p_z - b \cdot p_x &\leq t \\ b \cdot p_x - p_z &\leq t, \end{aligned}$$

and so this plane fitting problem is an instance of linear programming.

Example: testing for vertices.
Given a set S and a point $p \in S$, p is a vertex of $\text{conv}(S)$ iff p is not in $\text{conv}(S \setminus p)$, iff there is a plane separating p from $S \setminus p$. So to check if p is a vertex, we can check if the LP

$$\begin{array}{rl} \min & z \\ \text{subject to} & x \cdot q \leq z \text{ for all } q \in S \setminus p \\ \text{and} & x \cdot p \geq z. \end{array}$$

That is, we find the hyperplane "closest" to S.

We might be satisfied with a point $[x\ z]$ satisfying the inequalities, but with z not necessarily minimum. Such a point is called *feasible*.

The general LP problem is reducible to finding a feasible point (of a different LP problem).

The Farkas Lemma says that either there is a feasible point for this problem, or else there is one for

$$\sum_{q \in S} c_q q = p \tag{1}$$

$$\sum_{q \in S} c_q = = 1 \tag{2}$$

$$q \geq 0 \tag{3}$$

That is, here the vector c must satisfy the d equalities involving each coordinate of p and the q's, must sum to 1, and must have all coordinates at least 0.

This is a linear programming problem, the *dual* of the original one.

Minimum ball as quadratic programming

The minimum ball problem, which is to find x so that

$$\max_{p \in S} \|p - x\|$$

is minimized, can also be written as

$$\max_{p \in S} (p - x) \cdot (p - x),$$

which is equal to

$$\max_{p \in S} p^2 - 2p \cdot x + x^2,$$

and we can introduce a new variable r, where we want to minimize r^2, subject to

$$p^2 - 2p \cdot x + x^2 \leq r^2 \text{ for all } p \in S,$$

or putting $h = r^2 - x^2$,

$$\min h + x^2$$

subject to

$$p \cdot x + h/2 >= p^2/2 \text{ for all } p \in S.$$

That is, we have a collection of linear inequalities in $[x\ h]$, giving a polyhedron $P(H)$, and we want to minimize a *quadratic* function, $h + x^2$, within that polyhedron.

...so the minimum ball problem is not so far from linear programming.

(In fact, LP is reducible to the min-ball problem.)

The simplex algorithm

The simplex algorithm walks from vertex to vertex of $P(H)$, always reducing the d-coordinate.

```
get some v ∈ vert(P(H))
repeat
    pick w ∈ vert(P(H)) where vw is an edge and w_d < v_d
    v := w
until no such w exists
```

When the simplex algorithm stops, it has the global optimum point: if the neighbors of v are all above v, then so is $P(H)$.

A sketchy proof:
If v is the vertex at the stopping point, the condition that v is the global optimum is equivalent to the condition that the point $[0\ \ -1/v_d]$ is in the polar set of $P(H)$. Since $[0\ \ -1/v_d] \cdot v = 1$, and $[0\ \ -1/v_d] \cdot v' \leq 1$ for v' a neighboring vertex of v, it follows that $[0\ \ -1/v_d]$ is a point in the bounding plane of v^*, and is in v'^* for v' a neighbor of v, and so is in v°.

Running time:

It takes $O(n + d^3)$ to move from vertex to vertex.

The simplex algorithm visits each vertex of $P(H)$ at most once, so runs in at most $O(d^3 n^{\lfloor d/2 \rfloor + 1})$ time; it is always much faster than this is practice, usually $O(n + d)$.

There are examples where the worst-case running time is $2^{\Omega(d)}$; a variation of Seidel's algorithm gives a randomized version of a variation of simplex algorithm that takes subexponential time.

LP algorithms in small dimension

In small dimension, the same ideas as for minimum ball also work for linear programming: rather than sample sites, we sample halfspaces. The optimum vertex opt(R) for $P(R)$, where R is a subset of H, has similar properties as the minimum ball of a random subset of sites:

> opt(R) violates an expected small number of constraints (halfspaces);
> one of the constraint halfspaces violated by opt(R) is in basis(H).

In this context basis(H) is the set of d halfspaces that determine opt(H), so the same algorithms can be used.

Some complications for LP:

- $P(H)$ may be empty;
- opt(R) may not be unique;
- opt(R) may not exist because $P(H)$ is unbounded and there is no lowest point.

If $P(H)$ is empty, the problem is said to be *infeasible*.
If $P(H)$ is empty, it turns out that there must be $H' \subset H$ with $P(H')$ empty. Since any point violates (is not in) one of the halfspaces in H', the algorithms will find H' for the same reasons they find the halfspaces determining the optimum vertex.

To enforce uniqueness of opt(R), we use as optimum the lexicographically minimum point of $P(H)$ that achieves the smallest d'th coordinate; that is, among all points of $P(H)$ with smallest x_d, we consider only those with smallest x_{d-1}, and so on.

To handle the unbounded case, we conceptually put a big box around $P(H)$: a set $H^{@}$ of $2d$ halfspaces $x_i > -\delta$ and $x_i < \delta$, for $i = 1 \ldots d$, and for δ some big value; we make δ act like infinity: for any given number we calculate using the real data, δ is bigger.

Seidel's algorithm, for LP

To return to Seidel's algorithm: given sets of halfspaces G and T, let

opt(G, T) be the optimum point in $P(G)$, subject to the condition that the point in on the bounding hyperplanes of all halfspaces in T.

basis(G, T) be the halfspaces whose bounding hyperplanes contain opt(G, T), so $T \subset$ basis(G, T).

$v(G, T)$ be the x_d value of opt(G, T).

So SeidelLP(H, T) is:

> `if` $H = T$,
>   `return` opt(H, T);
> `pick` h `from` $H \setminus T$ `at random;`
> `compute` $x :=$ SeidelLP$(H \setminus h, T)$;
> `if` $x \in P(H)$, `return` x;
> `return` SeidelLP$(H, T \cup h)$;

The above algorithm is wasteful, in a way: the work done in finding basis$(H \setminus h, T)$ in not used when finding basis$(H, T \cup h)$.

Consider using $T' :=$ basis$(H \setminus h, T)$ instead of T in the last line of Seidel's algorithm: use h together with the basis obtained when computing x.

The trouble with this idea is that the set $T' \cup h$ will have $d + 1$ halfspaces: there may be no point in the intersection of the bounding planes.

Instead, use the basis of this set of $d + 1$ elements, found by brute force: solve a linear programming problem with $d + 1$ halfspaces, which means that one halfspace of the $d + 1$ will not determine the optimum. So the changed algorithm returns opt$(H,$ basis$(T' \cup h))$.

This algorithm, due to Sharir and Welzl, turns out to run in expected time *subexponential* in d.