Trapezoidal decomposition:

Motivation:

- manipulate/analayze a collection of *segments*
- e.g. detect segment intersections
- e.g., point location data structure
  - Draw verticals at all points
  - binary search for slab
  - binary search inside slab
  - problem: $O(n^2)$ space

Definition.

- draw altitudes from each intersection till hit a segment.
- trapezoid graph is *planar* (no crossing edges)
- each trapezoid is a *face*
- show a face.
- one face may have many vertices (from altitudes that hit the *outside* of the face)
- max vertex degree is 6 (assuming nondegeneracy)
- so total space $O(n + k)$ for k intersections.
- number of faces also $O(n + k)$ (each face needs one edge)
- (or use Euler's theorem: $n_v - n_e + n_f \geq 2$)
- standard clockwise pointer representation lets you walk around a face

Randomized incremental construction:

- to insert segment, start at left endpoint
- draw altitudes from left end (splits a trapezoid)
- traverse segment to right endpoint, adding altitudes whenever intersect
- traverse again, erasing (half of) altitudes cut by segment

Implementation

- clockwise ordering of neighbors allows traversal of a face in time proportional to number of vertices

- for each face, keep a (bidirectional) pointer to all not-yet-inserted left-endpoints in face
- to insert line, start at face containing left endpoint
- traverse face to see where leave it
- create intersection,
  - update face (new altitude splits in half)
  - update left-end pointers
- segment cuts some altititudes: destroy half
  - removing altitude merges faces
  - update left-end pointers

Analysis:

- Overall, update left-end-pointers in faces neighboring new line
- time to insert s is

  $$\sum_{f \in F(s)} (n(f) + \ell(f))$$

  where
  - $F(s)$ is faces s bounds after insertion
  - $n(f)$ is number of vertices in face f
  - $\ell(f)$ is number of left-ends in f.
- So if S_i is first i segmenets inserted, expected work of insertion i is

  $$\frac{1}{i} \sum_{s \in S_i} \sum_{f \in F(s)} (n(f) + \ell(f))$$

- Note each f appears at most 4 times in sum
- so $O(\frac{1}{i} \sum_f (n(f) + \ell(f)))$.
- Bound endpoint contribution:
  - note $\sum l(f) = n - i$
  - so contributes n/i
  - so total $O(n \log n)$
- Bound intersection contribution
  - $\sum n(f)$ is $O(k_i + i)$ if k_i intersections

- so cost is $E[k_i]$
- intersection present if both segments in first i insertions
- so expected cost is $O((i^2/n^2)k)$
- so cost contribution $(i/n^2)k$
- sum over i, get $O(k)$
- **note:** adding to RIC, assumption that first i items are random.

- Total: $O(n \log n + k)$

Search structure

Goal: apply binary search in slabs, without n^2 space

- Idea: trapezoidal decomp is "important" part of vertical lines
- problem: slab search no longer well defined
- but we show ok

The structure:

- A kind of search tree
- "x nodes" test against an altitude
- "y nodes" test against a segment
- leaves are trapezoids
- each node has two children
- so works like a search tree
- bf But node may have many parents
- sharing descendants saves space.

Inserting an edge contained in a trapezoid

- update trapezoids
- build a 4-node subtree to replace leaf

Inserting an edge that crosses trapezoids

- sequence of traps Δ_i
- if Δ_0 has left endpoint, replace leaf with x-node for left endpoint and y-node for new segment

- Same for last Δ
- middle Δ:
  - cut off pieces form new trapezoids (leaves)
  - replace each cut trapezoid with a y-node for new segment
  - two children of y-node point to appropriate traps
  - note trap can have several incoming nodes

Proof of correctness:

- Claim after each insert, valid search for current segments
- consider last insertion
- search gets to correct place before insertion
- new nodes continue search to correct place

Search time analysis

- depth increases by one for new trapezoids "below" new segment
- RIC argument shows depth $O(\log n)$

Linear programming.

- define
- assumptions:
  - nonempty, bounded polyhedron
  - minimizing x_1
  - unique minimum, at a vertex
  - exactly d constraints per vertex
- definitions:
  - hyperplanes H
  - **basis** $B(H)$ of hyperplanes that define optimum
  - optimum value $O(H)$
- Simplex
  - exhaustive polytope search:
  - walks on vertices
  - runs in $O(n^{\lceil d/2 \rceil})$ time in theory

- often great in practice

- polytime algorithms exist (ellipsoid)
- but bit-dependent (weakly polynomial)!
- OPEN: strongly polynomial LP
- goal today: polynomial algorithms for small d

Random sampling algorithm

- Goal: find $B(H)$
- Plan: random sample
  - solve random subproblem
  - keep only violating constraints V
  - recurse on leftover
- problem: violators may not contain all of $B(H)$
- bf BUT, contain **some** of $B(H)$
  - opt of sample better than opt of whole
  - but any point feasible for $B(H)$ no better than $O(H)$
  - so current opt not feasible for $B(H)$
  - so some $B(H)$ violated
- revised plan:
  - random sample
  - discard useless planes, add violators to "active set"
  - repeat sample on whole problem while keeping active set
  - claim: add one $B(H)$ per iteration
- Algorithm **SampLP**:
  - set S of "active" hyperplanes.
  - if $n < 9d^2$ do simplex ($d^{d/2+O(1)}$)
  - pick $R \subseteq H - S$ of size $d\sqrt{n}$
  - $x \leftarrow \textbf{SampLP}(R \cup S)$
  - $V \leftarrow$ hyperplanes of H that violate x
  - if $V \leq 2\sqrt{n}$, add to S
- Runtime analysis:

- mean size of V at most $\sqrt{n}$
- each iteration adds to S with prob. 1/2.
- each successful iteration adds a $B(H)$ to S
- deduce expect $2d$ iterations.
- $O(dn)$ per phase needed to check violating constraints: $O(d^2 n)$ total
- recursion size at most $2d\sqrt{n}$

$$T(n) \leq 2dT(2d\sqrt{n}) + O(d^2 n) = O(d^2 n \log n) + (\log n)^{O(\log d)}$$

(Note valid use of linearity of expectation)

Must prove claim, that mean $V \leq \sqrt{n}$.

- Lemma:
  - suppose $|H - S| = m$.
  - sample R of size r from $H - S$
  - then expected violators $d(m - r - 1)/(r - d)$
- **book broken: only works for empty S**
- Let C_H be set of optima of subsets $T \cup S$, $T \subseteq H$
- Let C_R be set of optima of subsets $T \cup S$, $T \subseteq R$
- note $C_R \subseteq C_H$, and $O(R \cup S)$ is only point violating no constraints of R
- Let v_x be number of constraints in H violated by $x \in C_H$,
- Let i_x indicate $x = OPT(R \cup S)$

$$\begin{aligned} E[|V|] &= E[\sum v_x i_x] \\ &= \sum v_x \Pr[i_x] \end{aligned}$$

- decide $\Pr[v_x]$
  - $\binom{m}{r}$ equally likely subsets.
  - how many have optimum x?
  - let q_x be number of planes defining x **not** already in S
  - must choose q_x planes to define x
  - all others choices must avoid planes violating x. prob.

$$\begin{aligned} \binom{m - v_x - q_x}{r - q_x} \Big/ \binom{m}{r} &= \frac{(m - v_x - q_x) - (r - q_x) + 1}{r - q_x} \binom{m - v_x - q_x}{r - q_x - 1} \Big/ \binom{m}{r} \\ &\leq \frac{(m - r + 1)}{r - d} \binom{m - v_x - q_x}{r - q_x - 1} \Big/ \binom{m}{r} \end{aligned}$$

– deduce

$$E[V] \leq \frac{m-r+1}{r-d} \sum v_x \binom{m-v_x-q_x}{r-q_x-1} / \binom{m}{r}$$

– summand is prob that x is a point that violates exactly one constraint in r.

  * must pick q_x constraints defining x
  * must pick $r - q_x - 1$ constraints from $m - v_x - q_x$ nonviolators
  * must pick one of v_x violators

– therefore, sum is expected number of points that violate exactly one constraint in R.

– but this is only d (one for each constraint in basis of R)

Result:

- saw sampling LP that ran in time $O((\log n)^{O(\log d)} + d^2 n \log n + d^{O(d)}$
- key idea: if pick r random hyperplanes and solve, expect only dm/r violating hyperplanes.

Iterative Reweighting

Get rid of recursion and highest order term.

- idea: be "softer" regarding mistakes
- plane in V gives "evidence" it's in $B(H)$
- Algorithm:
  - give each plane weight one
  - pick $9d^2$ planes with prob. proportional to weights
  - find optimum of R
  - find violators of R
  - if

    $$\sum_{h \in V} w_h \leq (2 \sum_{h \in H} w_h)/(9d-1)$$

    then double violator weights
  - repeat till no violators
- Analysis
  - show weight of basis grows till rest is negligible.
  - claim $O(d \log n)$ iterations suffice.
  - claim iter successful with prob. 1/2

  - deduce runtime $O(d^2 n \log n) + d^{d/2+O(1)} \log n$.
  - proof of claim:
    * after each iter, double weight of some basis element
    * after kd iterations, basis weight at least $d2^k$
    * total weight increase at most $(1 + 2/(9d-1))^{kd} \leq n \exp(2kd/(9d-1))$
  - after $d \log n$ iterations, done.

- so runtime $O(d^2 n \log n) + d^{O(d)} \log n$
- Can improve to linear in n

Randomized incremental algorithm

$$T(n) \leq T(n-1, d) + \frac{d}{n}(O(dn) + T(n-1, d-1)) = O(d!n)$$

Incomparable to prior bound.
Can improve to $O(d^4 2^d N)$ (see book)
Can improve to $O(d^2 n + b^{\sqrt{d \log d \log n}})$