Linear Programming

Linear programming

Given a set of linear inequalities (constraints) and a linear objective function,

maximize (or minimize) the objective function subject to the given constraints.

Linear programming

Maximize : $c_1\, x_1 + c_2\, x_2 + \ldots + c_d\, x_d$

Subject to :

$a_{1,1}\, x_1 + \ldots + a_{1,d}\, x_d \leq b_1$

$a_{2,1}\, x_1 + \ldots + a_{2,d}\, x_d \leq b_2$

.....

$a_{n,1}\, x_1 + \ldots + a_{n,d}\, x_d \leq b_n$

In matrix notation :

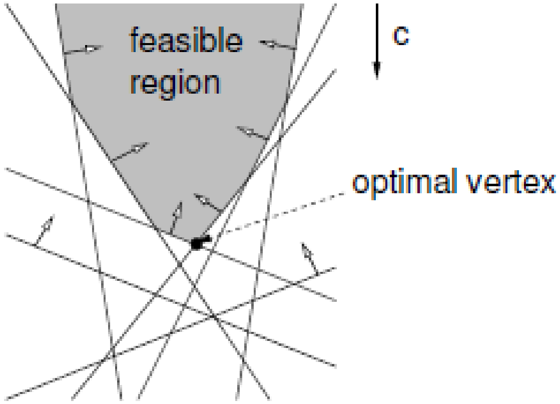
Maximize : $c^T x$,

Subject to : $A\, x \leq b$,

(c ,x : d-vectors, b : n-vector, A : nxd matrix)

Geometric perspective

- Constraints define the feasible region as a (possibly empty or unbounded) polyhedron in d-space.
- Find the point of the feasible region that is furthest in the direction c. (called optimal vertex)
- The magnitude of vector c is irrelevant. Assume that c is pointing straight down. Then, find the lowest point of the feasible region.



Solutions to LP

- Feasible : The extreme point exists (and assuming general position) as a vertex of the feasible region.
- Infeasible : The feasible region may be empty.
- Unbounded : The feasible region is unbounded in the direction of the objective function, so no finite optimal solution exists.

Degenerate case : edge/face of feasible region is perpendicular to the objective function vector. Infinite number of finite optimum solutions.

Linear programming

- Important technique to solve large optimization problems.
- Typically, hundrends to thousands of constraints in very high dimensional space.

High dimensional LP

- Principal methods for high dimensional LP are the simplex algorithm and various interior point methods.
- Simplex algorithm : find a vertex on the feasible polyhedron and walk edge by edge downwards until reaching a local minimum. (may run in exponential time)
- Karmarkar gave a polynomial time algorithm based on moving through the interior of the feasible region. (polynomial in # constraints, dimension, and bits of precision in the numbers)
- Strongly polynomial algorithm that is of combinatorial polynomial complexity (without assumption of bounded precision) is open problem.

Low dimensional LP

- Let's consider when d=2, first. How can we solve it?

Halfspace intersection

- Compute the feasible region by halfspace intersection and find the minimum vertex.
- In d=2, what is the complexity of halfspace intersection?
- How do you compute it?

Divide and conquer

- Divide halfspaces in half (H1, H2).
- Compute $C1 = \cap H1$, $C2 = \cap H2$
- Compute $C1 \cap C2$

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2\,T(n/2) + S(n) & \text{if } n > 1 \end{cases}$$

where S(n) is time to compute the intersection of two convex polyhedra.

What is S(n)?

- Intersection of line segments :
  - In O((n+I) log n) where I is the number of intersection pairs.
  - Since, I = O(n), O(n log n). Not good enough.
- By plane sweep, can be done in O(n) time. Then T(n) = O(n log n).

Can we do better?

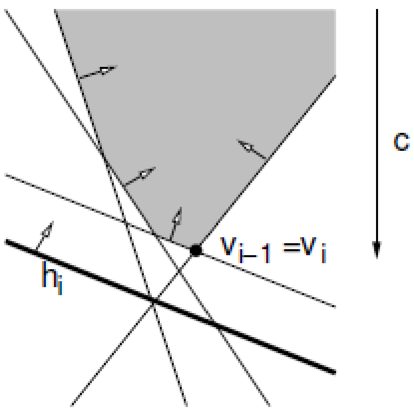
- Actually, we only need the extreme vertex of the feasible region. We don't need to know the feasible region itself.
- Any idea?

Incremental algorithm

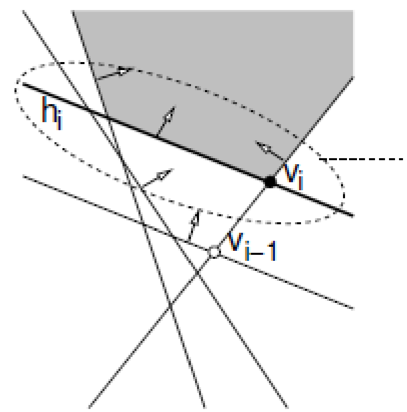
- Insert halfspaces (constraints) one by one and maintain the optimal vertex of the inserted halfspaces.
- Let v_i be the optimal vertex after inserting i halfspaces (h_1,…, h_i)
- Given v_{i-1} and h_i, need to compute v_i.

Update

- Case 1) If v_{i-1} is in h_i, $v_i = v_{i-1}$. Need to do nothing!
- Case 2) Otherwise, v_i lies on the bounding halfplane (line in 2d) e_i of h_i.



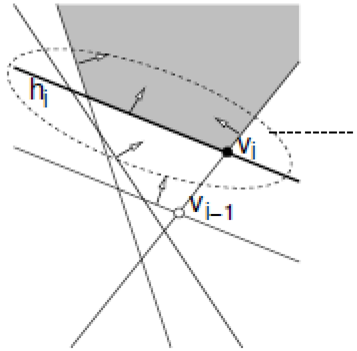
Case 1



Case 2

Update

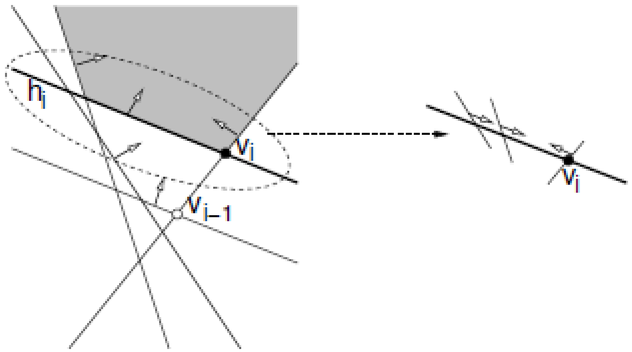
- Case2) If v_{i-1} is not in h_i, v_i lies on the bounding halfplane (line in 2d) e_i of h_i.
  - Suppose new optimal vertex does not lie on e_i. Draw a line segment from v_{i-1} to the new optimum. As you walk along the segment, the value of objective function is decreasing monotonically (by linearity), and this segment must cross e_i (because it goes from infeasible w.r.t h_i to feasible)



Case 2

Update

- How do we find the new optimum vertex on line e_i?
- Turns out to be 1d LP.
- Intersect each halfplane with this line. Each intersection will be a ray on the line. Intersect these intervals and find the point that maximizes the objective function.
- Can be solved in linear time by keeping the smallest upper bound and the largest lower bound.
- 2d LP is solved by a reduction to 1d LP.
- This generalizes to any dimension by repeatedly reducing to LP with next lower dimension.



Case 2

Analysis

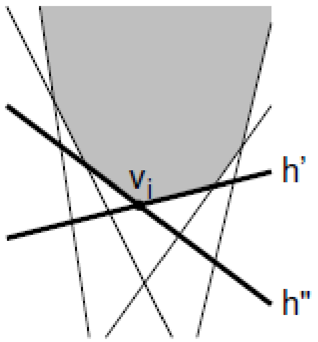
- Worst case $\sum$ O(i) : may run in quadratic time.
- Too pessimistic – assumes that current vertex is almost always infeasible.
- Can we do better?

Randomized incremental LP

- Insert halfplanes at random.
- $\sum O(i) X_i$ where X_i is the random variable which is 1 if v_{i-1} is not in h_i, 0, otherwise.
- The expected running time is $E[\sum O(i) X_i] = \sum O(i) E[X_i]$.
- $E[X_i] = Pr\{v_{i-1}$ is not in $h_i\}$?

Backwards analysis

- Assume the algorithm has already computed v_i and look at it "backwards".
- v_i is defined by two(d) of the halfplanes.
- If $v_{i-1} \neq v_i$, then h_i is one of the halfplanes defining v_i. The probability is 2/i.
- Expected time for i-th stage: O(i) (2/i) = O(1)
- By linearity of expectation, sum up over all n stages, O(n) total expected running time.

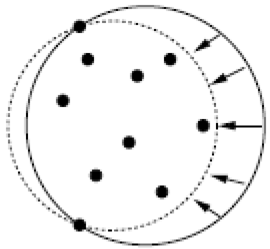


Smallest enclosing disk

- Given n points in the plane, find the closed circular disk of minimum radius that encloses all these points.
- 2-D : circle vs. disk
- 3-D: sphere vs. ball
- d-D : hypersphere vs. ball

- Claim1 : For any finite set of points in general position (no four cocircular), the smallest enclosing disk either has at least three points on its boundary, or it has two points, and these points form the diameter of the circle. If there are three points then they subdivide the circle bounding the disk into arcs of angle at most π.

- Pf) If there are less than two points on the boundary the disk's radius could be decreased. If there are two points on the boundary, and they are separated by an arc < π, then we can find a disk that passes through both points and has a slightly smaller radius. (By considering a disk whose center point is only the perpendicular bisector of the two points and lies a small distance closer to the line segment joining the points.)
- If there are three points that define the smallest enclosing disk they subdivide the circle into three arcs each of angle at most π (for otherwise we could apply the same operation above).



Randomized incremental algorithm

- Generate a random permutation $p_1, \ldots, p_n$.
- Let $P_i = \{ p_1, \ldots, p_i \}$, D_i : the smallest enclosing disk of P_i.
- If $p_i \in D_{i-1}$, $D_i = D_{i-1}$. Otherwise, p_i lies on the boundary of D_i.

Claim 2: If $p_i \notin D_{i-1}$ then, p_i lies on the boundary of D_i.

- Pf) Given a disk of radius r_1 and a circle of radius r_2 ($r_1 < r_2$), the intersection of the disk with the circle is an arc of angle $< \pi$. This is because an arc of angle $\geq \pi$ contains two (diametrically opposite) points whose distance from each other is $2r_2$, but the disk of radius r_1 has diameter only $2r_1$.
- Now, suppose to the contrary that p_i is not on the boundary of D_i. Because D_i covers a point not covered by D_{i-1}, D_i must have larger radius than D_{i-1}.
- Let r_1 and r_2 denote the radius of D_{i-1} and D_i, respectively.
- By the above argument, D_{i-1} intersects the circle bounding D_i in an arc of angle less than π.
- Since p_i is not on the boundary of D_i, the points defining D_i must be chosen from among the first i-1 points, from which it follows that they all lie within this arc. However, this would imply that between two of the points is an arc of angle $> \pi$.
- By claim1, D_i could not be a minimum enclosing disk.

Update

- If $p_i \notin D_{i-1}$, find a minimum enclosing disk of P_{i-1} with p_i on the boundary.
- D_i = MinDiskWith1Pt(P_{i-1}, p_i)
- MinDiskWith1Pt(P,q) : similarly, using randomized incremental algorithm
  - If $p_i \notin D_{i-1}$, D_i = MinDiskWith2Pts(P_{i-1},q,p_i)
- MinDiskWith2Pts(P,q1,q2) : similarly…
  - If $p_i \notin D_{i-1}$, D_i = disk(q1,q2,p_i)

Algorithm

MinDisk(P) :

(1) If $|P| \leq 3$, then return the disk passing through these points. Otherwise, randomly permute the points in P yielding the sequence $\langle p_1, p_2, \ldots, p_n \rangle$.

(2) Let D_2 be the minimum disk enclosing $\{p_1, p_2\}$.

(3) for $i = 3$ to $|P|$ do

(a) if $p_i \in D_{i-1}$ then $D_i = D_{i-1}$.

(a) else $D_i = \text{MinDiskWith1Pt}(P[1..i-1], p_i)$.

MinDiskWith1Pt(P, q) :

(1) Randomly permute the points in P. Let D_1 be the minimum disk enclosing $\{q, p_1\}$.

(2) for $i = 2$ to $|P|$ do

(a) if $p_i \in D_{i-1}$ then $D_i = D_{i-1}$.

(a) else $D_i = \text{MinDiskWith2Pts}(P[1..i-1], q, p_i)$.

MinDiskWith2Pts(P, q_1, q_2) :

(1) Randomly permute the points in P. Let D_0 be the minimum disk enclosing $\{q_1, q_2\}$.

(2) for $i = 1$ to $|P|$ do

(a) if $p_i \in D_{i-1}$ then $D_i = D_{i-1}$.

(a) else $D_i = \text{Disk}(q_1, q_2, p_i)$.

Analysis

- MinDiskWith2Pts : O(n) time, space.
- MinDiskWith1Pt is O(n) + time spent in calls MinDiskWith2Pts.
- What is the probability to make such a call?

Backwards analysis

- Fix a subset $\{p_i, \ldots, p_i\}$ and let D_i be the smallest disk enclosing $\{p_i, \ldots, p_i\}$ and having q on its boundary.
- When does the smallest enclosing circle change? when we remove one of the three points on the boundary.
- The probability that p_i is one of those points is 2/i.
- So expected running time of MinDiskWith1Pt is $O(n) + \sum O(i)\, 2/i = O(n)$
- Similarly, expected running time of MinDisk is O(n).

LP-type problems

- Generally applicable to the optimization problem where the solution either does not change when a new constraint is added, or the solution is partially defined by the new constraint so that the dimension of the problem is reduced.