CS 164 & CS 266: Computational Geometry

Lecture 9

Low-dimensional linear programming

David Eppstein
University of California, Irvine

Fall Quarter, 2023

Definition

Linear programs

Find values for some variables

$$x, y$$

Obey linear inequalities, called "constraints"

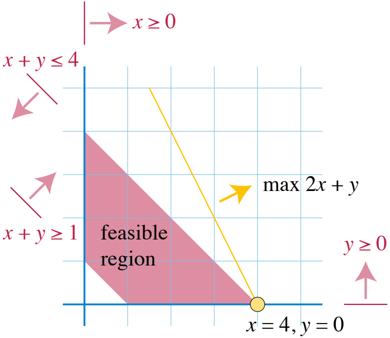
$$\begin{aligned} x &\geq 0 \\ y &\geq 0 \\ x + y &\geq 1 \\ x + y &\leq 4 \end{aligned}$$

Minimize or maximize a linear "objective function"

$$\max 2x + y$$

Think of variables as coordinates

"Feasible region": convex set, points obeying constraints



Min or max is a vertex

Geometric linear programs

For the problems we will be considering:

- Dimension (number of variables) will be $O(1)$
- Size of problem (number of constraints, n) can be large
- Algorithms search among small subsets of constraints and their optimal solution points ("dual simplex method")

  (If you haven't seen this phrase, don't worry about it)
- Time $O_d(n)$: linear in the number of constraints, but with a constant factor that depends badly on the dimension

  Unlike algorithms for high-dimensional LP, time does not depend on numerical precision
- If we just want to test whether there exists a feasible point, can choose objective arbitrarily

Background about more general types of LP

More generally, for linear programs:

- Might have a large number of variables
- Duality: there is an equivalent LP with a variable for each constraint and vice versa
- Can be solved in time polynomial in number of variables, number of constraints, and number of bits needed to represent the numerical coefficients in the linear functions
  - Interior point methods: Follow a curve interior to the feasible region, improving objective, until reaching solution
  - Ellipsoid method: Enclose feasible region by an ellipsoid, bisect it to get a smaller feasible region, and repeat until converging to a solution
- We will give up this added generality in order to obtain linear time and no dependence on number of bits

Examples of geometric LPs

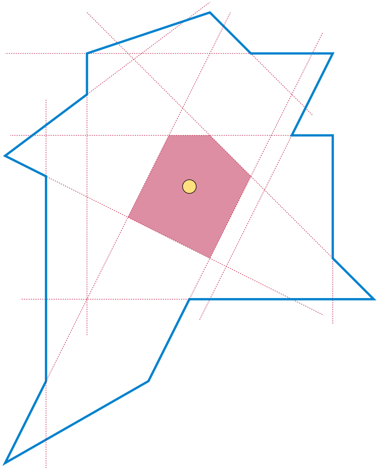
Art gallery with one guard

Input: A polygon without holes

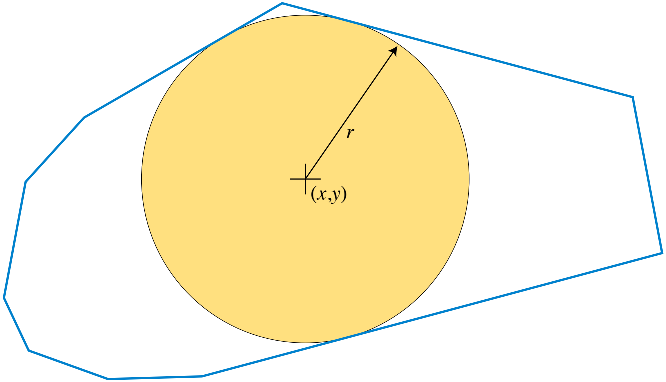
Output: A point inside it from which entire polygon is visible

LP feasibility with a constraint for each polygon side

A polygon that can be guarded by one guard is "star-shaped"; the feasible region of its LP is the "kernel" of the polygon



Biggest circle inside a convex polygon



Variables: x, y, r

Constraint for each polygon edge: x and y are on correct side of the edge, and their distance from the side (a linear function in x and y with coefficients determined from the side) is at least r

Maximize r

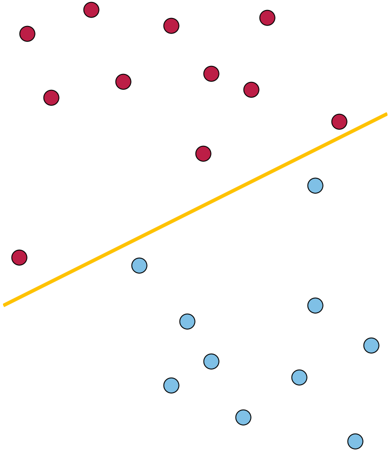
Linear separation

Given red points and blue points with coordinates (x_i, y_i)

Variables: m, b representing the line $y = mx + b$

Constraints:
$y_i \geq mx_i + b$ (for red points)
$y_i \leq mx_i + b$ (for blue points)

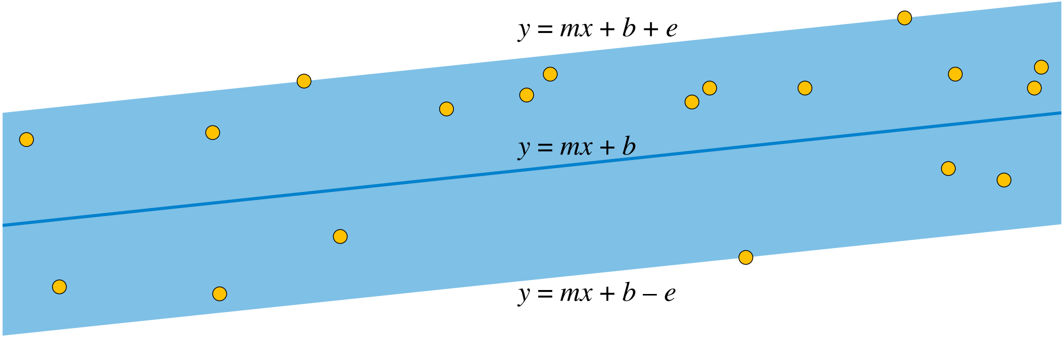
With one more variable, can maximize vertical distance to line $\Rightarrow$ idea behind support vector machine learning



L_∞ linear regression

Regression: Fit a line $y = mx + b$ to a set of data points x_i, y_i minimizing some combination of errors $|(mx_i + b) - y_i|$

L_∞: Minimize max error; variables m, b, e, constraints $-e \leq (mx_i + b) - y_i \leq e$, objective min e



More useful in metrology (how close to flat is this set of measurements of a surface) than statistics, because L_2 regression (least squares) is easier, less sensitive to outliers

Algorithms

How quickly can we solve low-dimensional LP?

Non-random

$O_d(n)$ – Time is function of d times $O(n)$

(Simplifies to $O(n)$ if we assume d is constant)

Originally $O(2^{2^d} n)$, later improved to $O(3^{d^2} n)$

[Megiddo 1984; Clarkson 1986]

Random

$O(d^2 n) + 2^{O(\sqrt{d \log d})}$

[Matoušek et al. 1996]

Today

Simpler randomized algorithm with time $O(d!\, n)$

[Seidel 1991]

Warm-up: Randomized incremental max

Given an array A of n numbers:

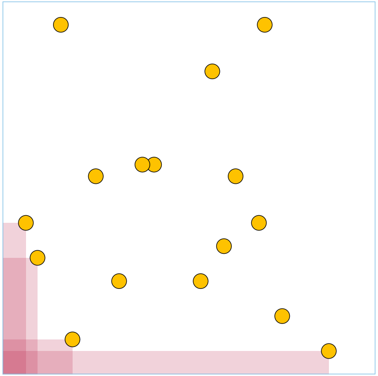
> Randomly permute A
>
> Result $= -\infty$
>
> For $i = 0, \ldots, n-1$:
>
> > If $A[i] >$ result:
> >
> > > result $=$ A[i]

Obviously, this takes $O(n)$ time, and the randomization is completely unnecessary

More interesting question: how many times do we change result?

An equivalent geometric problem in 2d

Given n random points in a unit square
How many have empty quadrant below and to the left of them?



(x-coordinate = order of random permutation,
y-coordinate = values we are finding the minimum among,
empty quadrant = result changes when we get to that point)

Backwards analysis

Suppose we have just looped through the ith value

What is the probability that we just changed the result?

Happens when ith value is minimum among first i values

Random permutation $\Rightarrow$ minimum equally likely to be anywhere

Probability that it is last is exactly $1/i$

To compute expected number of times we changed the result, sum for each step the probability that we changed result in that step

$$\sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1)$$

Seidel's algorithm

To solve a d-dimensional linear program:

Randomly permute the constraints

Choose coordinates $\pm\infty$ for an optimal solution point
(whichever of $+\infty$ or $-\infty$ is better for objective function)

For each constraint $\sum a_i x_i \leq b$, in a random order:

Check whether solution point obeys the constraint

If not, solve recursively a $d-1$-dimensional LP
and replace solution point by the result

The recursive problem works in the $(d-1)$-dimensional subspace of points $\sum a_i x_i = b$, and uses the constraints that have already been added, restricted to that subspace, in a new random order

Backwards analysis of Seidel's algorithm

After processing the ith constraint, what is the probability that you had to make a recursive call for it?

In any d-dimensional LP, some subset of d constraints is exactly satisfied, and determine the solution

- Solution is solution to d linear equations in d variables
- Fewer constraints $\Rightarrow$ can move solution in a linear subspace and get better in some direction
- More constraints $\Rightarrow$ some of them are redundant and not needed to determine solution

If you just made a recursive call, the last constraint you processed was one of these d constraints

Random permutation $\Rightarrow$ Happens with probability $\leq d/i$

(Can be $< d/i$ if $d > i$ or for multiple sets of d right constraints)

Expected time for Seidel's algorithm

Let $T(d, n)$ denote the expected time to solve a d-dimensional LP with n constraints

Expected time for ith constraint: $O(d)$ to check constraint, plus
(probability of making a recursive call) $\times$ (time if we make the call)

Sum this time over all constraints:

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

Prove by induction that $T(d, n) = O(d!n)$

Induction hypothesis $\Rightarrow$ sum becomes $\sum d(d-1)!(i-1)/i < d!n$

References

Kenneth L. Clarkson. Linear programming in $O(n \times 3^{d^2})$ time. *Information Processing Letters*, 22(1):21–24, 1986. doi: 10.1016/0020-0190(86)90037-2.

Jiří Matoušek, Micha Sharir, and Emo Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4–5):498–516, 1996. doi: 10.1007/BF01940877.

Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31(1):114–127, 1984. doi: 10.1145/2422.322418.

Raimund Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry*, 6(5):423–434, 1991. doi: 10.1007/BF02574699.