Lecture 14: Linear Programming II

David Woodruff
Carnegie Mellon University

Outline

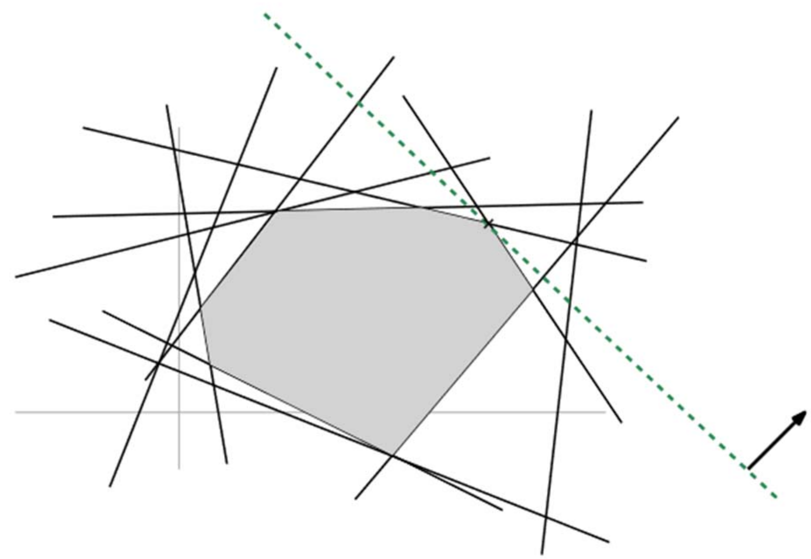
- Another linear programming example – l1 regression
- Seidel's 2-dimensional linear programming algorithm
- Ellipsoid algorithm

L1 Regression

- Input: n x d matrix A with n larger than d, and n x 1 vector b
- Find x with Ax = b
- Unlikely an x exists, so instead compute $\min_{x} \sum_{i=1,\ldots,n} |A_i \cdot x - b_i|$
- Solve with linear programming? How to handle the absolute values?
- Create variables s_i, t_i for i = 1, …, n with $s_i \geq 0$ and $t_i \geq 0$
  - Also have variables $x_1, \ldots, x_d$
- Add constraints $A_i \cdot x - b_i = s_i - t_i$ for i = 1, …, n
- What should the objective function be?
- min $\sum_{i=1,\ldots,n} s_i + t_i$

Seidel's 2-Dimensional Algorithm

- Variables x_1, x_2
- Constraints $a_1 \cdot x \leq b_1, \ldots, a_m \cdot x \leq b_m$
- Maximize $c \cdot x$
- Start by making sure the program has bounded objective function value



What if the LP is unbounded?

- Add constraints $-\mathrm{M} \leq \mathrm{x}_1 \leq \mathrm{M}$ and $-\mathrm{M} \leq \mathrm{x}_2 \leq \mathrm{M}$ for a large value M

- How large should M be?

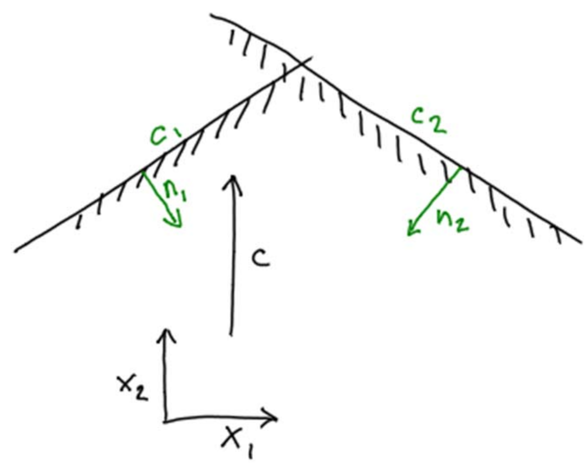
- Maximum, if it were unbounded, occurs at the intersection of two constraints $\mathrm{ax}_1 + \mathrm{bx}_2 = \mathrm{c}$ and $\mathrm{ex}_1 + \mathrm{fx}_2 = \mathrm{d}$

$$\begin{bmatrix} a & b \\ e & f \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} c \\ d \end{bmatrix}$$

- If a, b, e, f, c, d can be specified with L bits, can show $|\mathrm{x}_1|, |\mathrm{x}_2|$ are $2^{\mathrm{O(L)}}$

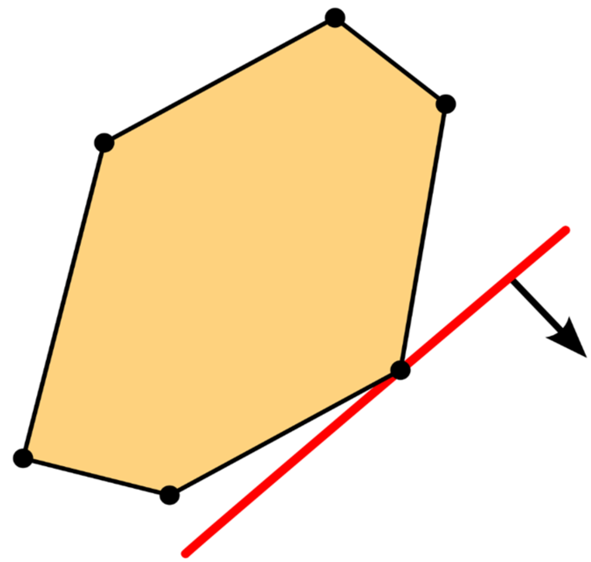
Another Way to Determine Boundedness

- Rotate the problem so that Maximize $\mathrm{c} \cdot \mathrm{x}$ becomes Maximize x_2
  - Rotation doesn't have any affect on boundedness
- Look at constraints and see if there is one whose
  - normal points "down and to the right"
  - normal points "down and to the left"
- System is bounded iff both exist
- Call these two constraints c_1 and c_2



2

What Convexity Tells Us



- Maximizing a linear function over the feasible region finds a tangent point

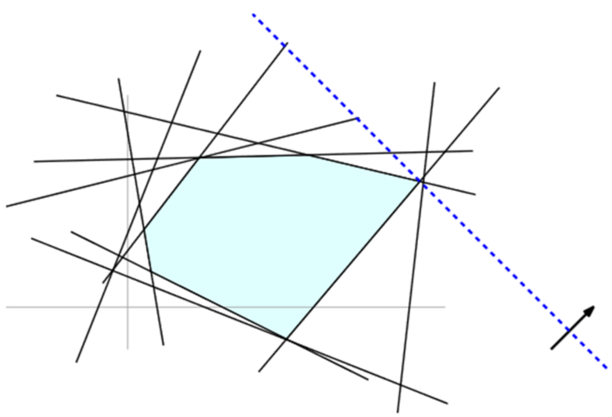
- What's a super naïve $O(m^3)$ time algorithm?

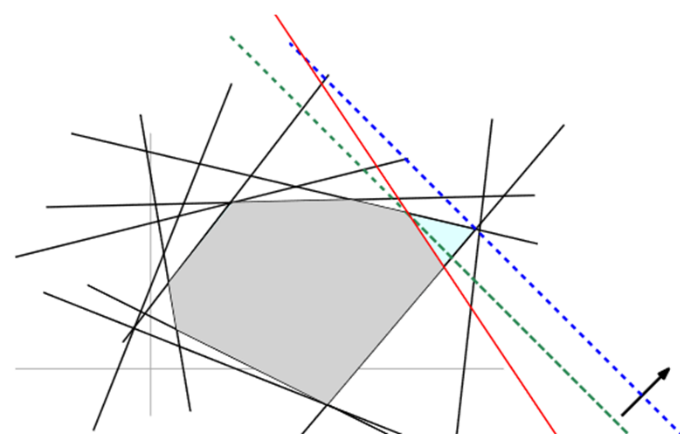
- Find the intersection of each pair of constraints, compute its objective function value, and make sure this point is feasible for all constraints

- What's a less naïve $O(m^2)$ time algorithm?

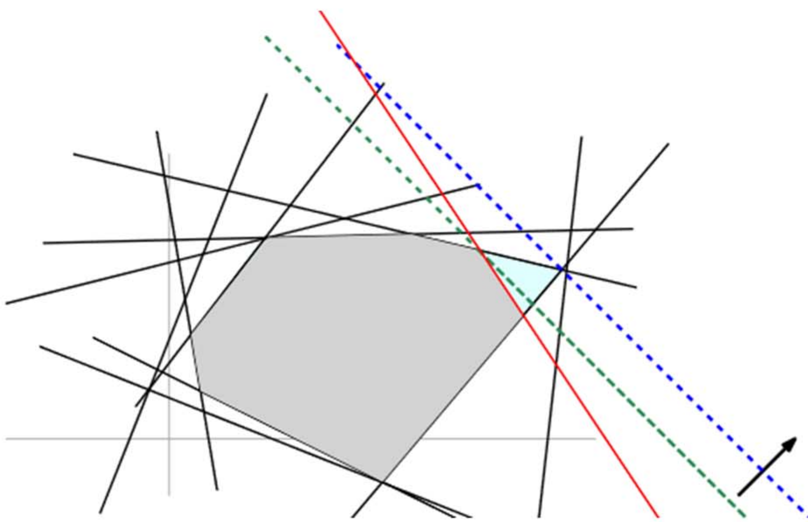
An $O(m^2)$ Time Algorithm

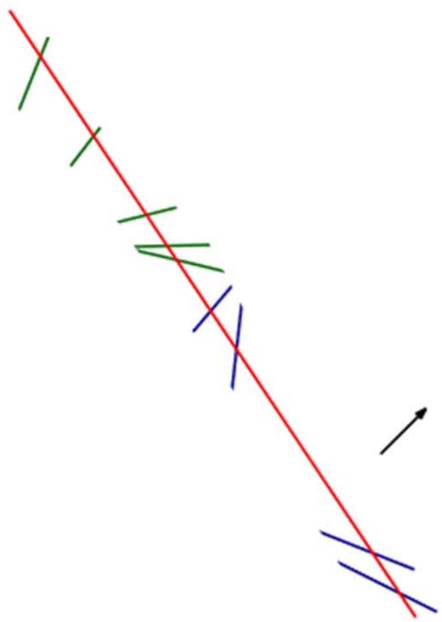
- Order the constraints $a_1 \cdot x \le b_1, \ldots, a_{m-2} \cdot x \le b_{m-1}, c_1, c_2$
- Recursively find optimum point x^* of $a_2 \cdot x \le b_2, \ldots, a_{m-2} \cdot x \le b_{m-2}, c_1, c_2$
- If $a_1 x^* \le b_1$, then x^* is overall optimum
- Otherwise, new optimum intersects the line $a_1 x^* = b_1$
- Need to solve a 1-dimensional problem





1-Dimensional Problem





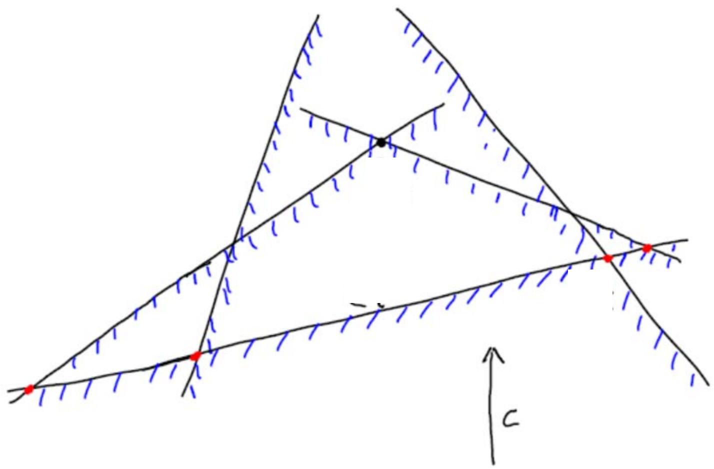
- Takes O(m) time to solve

An $O(m^2)$ Time Algorithm

- Recursively find optimum point x^* of $a_2 \cdot x \leq b_2, \ldots, a_{m-2} \cdot x \leq b_{m-2}, c_1, c_2$
- If $a_1 x^* \leq b_1$, then x^* is still optimal
- Otherwise, new optimum intersects the line $a_1 \cdot x = b_1$
- Solve a 1-dimensional problem in O(m) time
- T(m) = T(m-1) + O(m) = $O(m^2)$ time
- Can we get O(m) time?

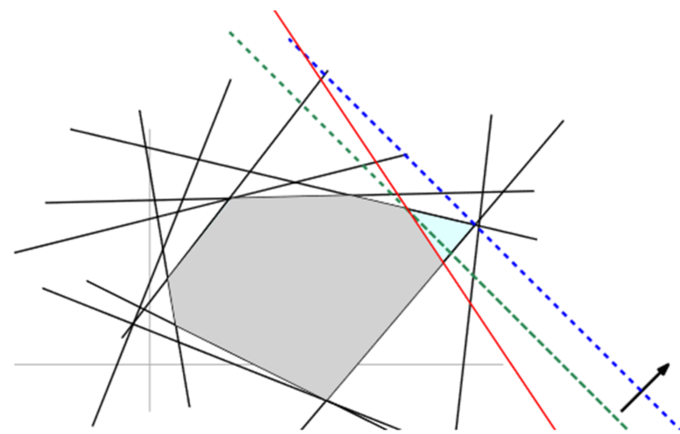
Seidel's O(m) Time Algorithm

- Order constraints randomly: $a_{i_1} \cdot x \leq b_{i_1}, \ldots, a_{i_{m-2}} \cdot x \leq b_{i_{m-2}}, c_1, c_2$
  - Leave c_1, c_2 at the end
- Recursively find the optimum x^* of $a_{i_2} \cdot x \leq b_{i_2}, \ldots, a_{i_{m-2}} \cdot x \leq b_{i_{m-2}}, c_1, c_2$
- Case 1: If $a_{i_1} \cdot x^* \leq b_{i_1}$, then x^* is overall optimum
  - O(1) time
- Case 2: If $a_{i_1} \cdot x^* > b_{i_1}$, then we need to intersect the line $a_{i_1} \cdot x = b_{i_1}$ with each other line $a_{i_j} \cdot x = b_{i_j}$ and solve a 1-dimensional problem in O(m) time



Backwards Analysis

- Let x^* be the optimum point of $a_{i_2} \cdot x \leq b_{i_2}, \ldots, a_{i_{m-2}} \cdot x \leq b_{i_{m-2}}, c_1, c_2$
- What is the chance that $a_{i_1} \cdot x^* > b_{i_1}$?
- Suppose the optimal point x' of the overall LP is the intersection of $a_{i_j} \cdot x = b_{i_j}$ and $a_{i_{j'}} \cdot x = b_{i_j}$
- If we've seen these two constraints, then the new constraint $a_{i_1} \cdot x \quad \leq b_{i_1}$ can't change the optimum. Otherwise, better solution would be on this new line



- T(m) is expected cost for m constraints,

  T(m) $\leq$ (1-2/(m-2)) O(1) + (2/(m-2)) $\cdot$ $O(m)$ + T(m-1)

  = O(1) + T(m-1)

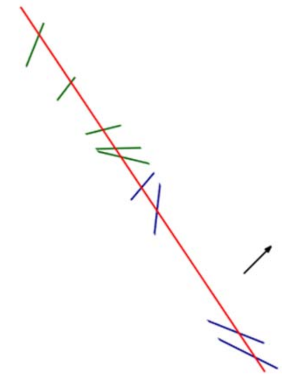
  = O(m). Also add initial O(m) time for finding c_1, c_2

What if the LP is Infeasible?

- Let j be the largest index for which $a_{i_j} \cdot x \leq b_{i_j}, \ldots, a_{i_{m-2}} \cdot x \leq b_{i_{m-2}}, c_1, c_2$ is infeasible. That is, $a_{i_{j+1}} \cdot x \leq b_{i_{j+1}}, \ldots, a_{i_{m-2}} \cdot x \leq b_{i_{m-2},} c_1, c_2$ is feasible

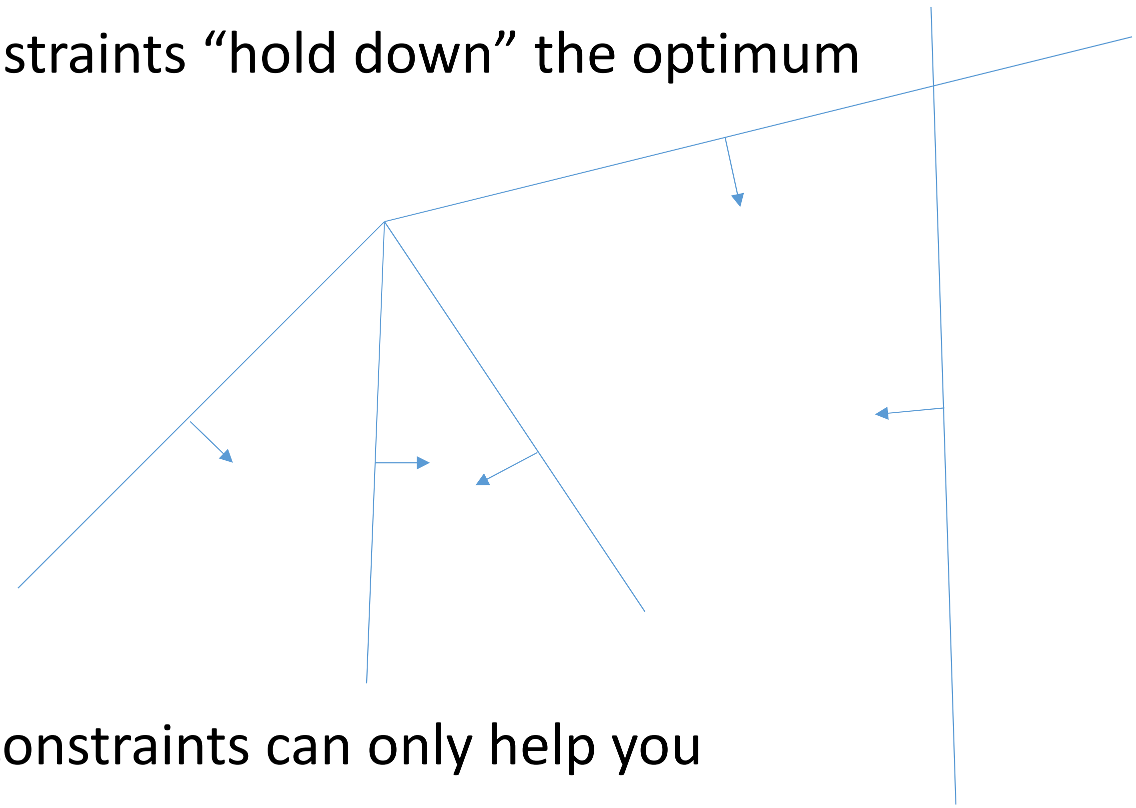
- Since $a_{i_{j+1}} \cdot x \leq b_{i_{j+1}}, \ldots, a_{i_{m-2}} \cdot x \leq b_{i_{m-2},} c_1, c_2$ is randomly ordered, we spend an expected O(m-j) time to process such constraints

- When processing $a_{i_j} \cdot x \leq b_{i_j}$ we will find the constraints are infeasible in O(m) time when solving the 1-dimensional problem



What If More than 2 lines Intersect at a Point?

- 2 of the constraints “hold down” the optimum



- Additional constraints can only help you

Higher Dimensions?

- The probability that our optimum changes is now at most d/(m-d) instead of 2/(m-2)

- When we find a violated constraint, we need to find a new optimum

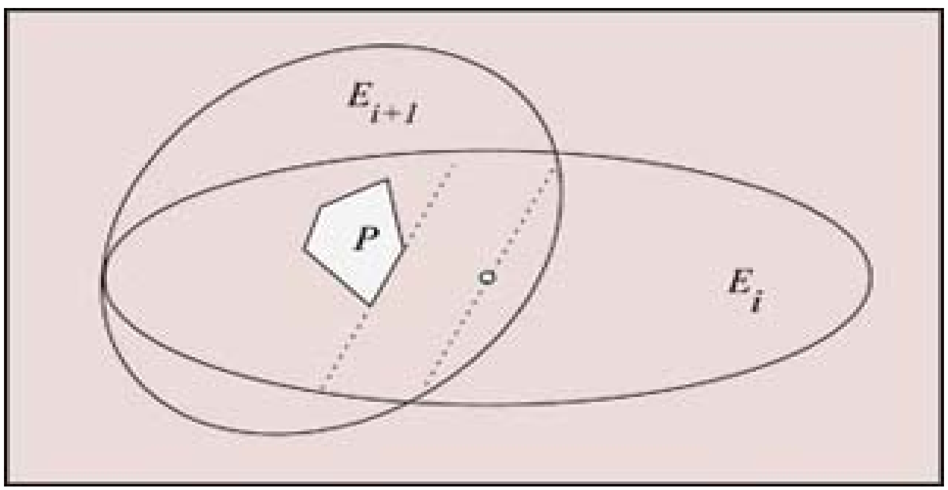
- New optimum inside this hyperplane
  - Project each constraint into this hyperplane
  - Solve a (d-1)-dimensional linear program on m-1 constraints to find optimum
  - $\mathrm{T(d,m)} \leq \mathrm{T(d,m-1)} + \mathrm{O(d)} + \frac{\mathrm{d}}{\mathrm{m-d}}[\mathrm{O(dm)} + \mathrm{T(d-1,m-1)}]$
  - T(d,m) = O(d! m)

Ellipsoid Algorithm

Solves feasibility problem

Replace objective function with constraint, do binary search

Replace “minimize $x_1 + x_2$" with $x_1 + x_2 \leq \lambda$



Can handle exponential number of constraints if there’s a separation oracle

Ellipsoid Algorithm in d dimensions

- Start with a big ellipsoid which contains the feasible region
- Check each constraint to see if the ellipsoid center is feasible
- If so, done
- Otherwise, find a violated constraint cutting the ellipsoid in half
- In poly(d) time find a new ellipsoid containing the half of the old ellipsoid containing the feasible region

Volume Argument

- Volume of new ellipsoid at most (1-1/d)*volume of old ellipsoid

- After d iterations, what is volume of new ellipsoid?
- After d^2L iterations, what is volume of new ellipsoid?

- Starting volume is $2^{\Theta(Ld^2)}$
  - Use Cramer's rule to show optimal solution has entries at most $d! \cdot 2^{\Theta(Ld)}$

- End volume is $2^{-\Theta(Ld^2)}$
  - Add $2^{-\Theta(Ld^2)}$ to right hand side of each inequality $A_i \cdot x \leq b_i$
  - Feasible region could be a point, but after adding this, it has volume at least $2^{-\Theta(Ld^2)}$
  - If infeasible, then because of bit complexity L, after adding this, still infeasible

Time Complexity

- Ld^2 iterations, in each use O(mdL) time to find a violated constraint (if operations on L bit numbers are O(L) time)

- Find description of new ellipsoid in poly(dL) time
  - Do some linear algebra

- Overall poly(mdL) time