lecture 14: How do LP solvers work?

E.g. LP:
maximize \[ 5x_1 + x_2 \]
s.t. \[ x_1 + x_2 \leq 7 \] \( \Rightarrow \)
\[ -2x_1 + 3x_2 \geq -4 \] \( \Rightarrow \)
\[ x_1 \geq 0 \] \( \Rightarrow \)
\[ x_2 \leq 3 \] \( \Rightarrow \)

Geometric viewpoint: plot the "feasible points" \( x = (x_1, \ldots, x_n) \) in \( \mathbb{R}^n \).

[Think first of each inequality as an equality.
These points satisfying \( \Rightarrow \) as equality are "on the boundary" of satisfying inequalities.
In \( n=2 \) case, equality is a line. In \( n=3 \): plane. Generally: "hyperplane".]

[Each inequality says you're on one side of line/plane/hyperplane.
It's a "halfspace". \( \Rightarrow \) "Feasible region" is intersection of halfspaces.

Stupid annoying/annoying possibility 1:
region is empty
"LP is infeasible"
E.g. "\( x_1 \geq 2, x_1 \leq 1 \)."

[Automatically not the case for many real-world problems, e.g.
Max-flow: can always set all flow vals \( f_e = 0 \).]

Stupid/annoying possibility 2:
region is unbounded
E.g. If our example didn't have \( x_1 \geq 0 \)
[Also usually not an issue; don't mind including \( x_i \geq 2^{-1000} \) \& \( x_i \leq 2^{1000} \) for all \( i=1..n \).]

Then everything inside a big box.]

[\( n=2 \) in our example]
Fact: Optimum solution always occurs at a vertex.

If we are so interested in corners?

\[ \frac{x}{y} = \frac{m}{n} \]

But… \( m \) is exponentially large! E.g., if \( m = 2^n \), check if feasible.

Efficiently find intersection by solving linear system.

Given any \( n \) out of \( m \) constraints, can solve one of \( m \) vertices. If \( n \geq m \), then \( \binom{m}{n} \) vertices.

Linear independence

\[ \text{Constrain-equations. } \] 
\[ \text{Intersection/solution of } \]

Observation: Every "vertex" (corner) is the intersection/solution of the segment feasible linearly independent, all points of the segment feasible edges. Also feasible vertices, edges. All faces of a polygon, all polygons, every polyhedron.

Feasible region is a "convex polytope".

I'm not worrying about possibilities, let's not worry about it. There are stupid annoying things to get around these.
Often efficient in practice. Is some simple linear upper bound.

Pros:
- Simple alg.
- To figure out an acceptable path.

This is how the "Simple Algorithm" solves LP.

But for $n > 2$, can have many improving edges.

In ad, only one choice at each

Note: this ends up walking from vertex to vertex, along

always trying to reach up like a helium balloon.

Moving in this direction is

Objective: $x_2 = x_1 + 2x_2$. 

Every local opt is a global opt.

(And of LP: I had local optima.)

Keep going till stuck. 

Any edge will

hanging well, till another will.

Keep trying to move as much in this direction as you can

you hit a wall.

Keep moving in Objective direction (eg, $x_1$)

Suggests a "physical" algorithm:

Start anywhere in feasible region.

Region: last point of contact always a vertex (could also be a

and larger and longer?)

If first time it still intersects feasible

vertex: keep sliding. keep option always occurs at a

Inuition/proof for why
Cons: * How to find a starting vertex?

Often not a big deal; e.g., $x_i = 0 \Rightarrow$ is often a vertex.

This is true for flows: $f_i = 0 \Rightarrow$ is a vertex.

In general, though, have to do an annoying stupidity that
involves setting up and solving another artificial LP. 

* Provably exponential time in worst case.

* Can get stuck doing exponentially small improvements.

First known provably poly-time alg: [but very much not practically efficient!]

"Ellipsoid Alg."

* Can always binary search for optimum, adding "objective $\geq \beta" as a
constraint (for various $\beta$), so suffices to solve "find a feasible point."

---

* Searches for feasible point from outside
1. Start with giant sphere, sure to contain feasible region.
2. Check if center is feasible. If so, done!

Else, get a "violated constraint":

3. Compute [not hard] smallest ellipsoid
   containing cut-off chunk.
4. Go to 2.

// give up if current ellipsoid exponentially small.

Key theorem: after $3n$ slicings, ellipsoid's volume is halved (or smaller)
[So it's like binary search: geometric shrinking of volume]
Why ellipsoids? No deep reason; just a simple kind of shape for which it's easy to compute the smallest enclosing of the previous chunk, and prove the volume decreases geometrically.

Bonus: Alg doesn't need to see/know all constraints. Just needs to solve "separation oracle" problem:

\[
given \ x, \ \ \begin{array}{c}
\rightarrow \ & \text{if feasible, say so} \\
\rightarrow \ & \text{if not, output violated constraint}
\end{array}
\]

Sometimes you can set up an LP for a problem with \( n \) vars & \( \exp(n) \) many constraints, but while there is still an efficient alg for \( \circ \), then these LPs can be solved in \( \text{poly}(n) \) time!

"Interior point methods" - efficient in theory and in practice.

"As the name suggest, these algos walk thru the interior of the polytopes!"

Define \( P_{\beta} = P \cup \{ c^T x \geq \beta \} \). Want to make \( \beta \) as big as possible while still having a point \( x \in P_{\beta} \).

Idea: "hard constraints" \( \rightarrow " \text{soft constraints}" \ ("barrier func")

\[
\alpha \cdot x \geq 6 \ \rightarrow \ \ln(\alpha \cdot x - 6) \text{ is large}
\]

[Objective function is most important, so take \( n \) copies of it.]

\[\text{level sets of } f \]

feas. region \[ \alpha^{(i)} \cdot x \geq b_i \]
Define $f_\beta(x) = m \cdot \ln(c^\top x - \beta) + \sum_{i=1}^\infty \ln(a_i^{(i)} x - \beta)$.

- Inside $P_\beta$, $f_\beta(x)$ is well-defined.
- $f_\beta(x) = -\infty$ on boundary of $P_\beta$.
- $f_\beta(x)$ is smooth & concave. $L = \text{has unique maximizer}$ $\mathbf{w}_\beta$ where $\nabla f_\beta = 0$.

Idea: $t=0$

- Start with very small $\beta_0$, some generic $x_0 \in P_0$.
- Find $W_0$.
  Use gradient ascent to move from $x_0$ towards $\mathbf{w}_\beta$.
  Use Newton's method "...

  A "second order/derivative"-based faster method we can use because $f_\beta$ is strictly concave, no saddle points, and 2nd-deriv (Hessian) of $f_\beta$ can be explicitly given.
  But just a little bit (one step of Newton's method).

- $x_{t+1}$ is new point.
- $\beta_{t+1} = \beta_t + \text{a little bit}$.
- Repeat.

Analysis ideas:
- Newton's method converges rapidly if you basically get very close to $\mathbf{w}_\beta$.
- Since $c^\top x \geq \beta$ so highly weighted (factor $m$), $\mathbf{w}_\beta$ must be at least a little bit far from $c^\top x \geq \beta$.
- Okay to increase $\beta$ by a little $(\approx \frac{1}{\sqrt{m}})$.
- Alg time is $\propto \sqrt{m}$ time to compute matrix inverse of Hessian...