

1 Administrivia

I'll be out of town on Feb 27, so I'm deferring the reading for the 27th to Mar 4.

2 Linear Programming, Concluded

- Last time, we saw the beginnings of LP. We started with the “diet” problem.
- Our (trivial) example problem was:

$$\begin{aligned} \text{Minimize} \quad & \text{cost} = 2x_{pb} + 3x_h \\ \text{Subject to} \quad & x_{pb} + 2x_h \geq 4 \\ & x_{pb} \geq 0 \\ & x_h \geq 0 \end{aligned}$$

- For illustration purposes, I'm going to change the cost function to

$$\text{Minimize} \quad \text{cost} = 2x_{pb} + x_h$$

(Someone's got a great sale on hamburger. Possibly pretty scary hamburger, but if you're only in it for the protein. . .)

- Recall that this simply says “find the minimal cost diet of peanut butter and protein s.t. you get ≥ 4 units of protein (and you can't eat a negative amount of anything)”
- Now we played a little bit with the geometry of this problem last time.
- **Diagram 2-d geometry**
- In this space, we plot only the two variables that we can control (x_{pb} and x_h) and the constraints that affect them.
- The area in the upper right is the “feasible region” in which solutions are possible.
- Now think – in general, you can construct as many linear constraints as you like. Each forms an edge in X space. In \mathbb{R}^2 each constraint is a line, in 3-space, a plane, in n -space, a hyperplane.
- The region on one side of a hyperplane is called a **halfspace**.
- The general structure you're building is, in 2-d, a polygon, in 3-d, a polyhedron, and, in n -d a polytope.
- There are 3 things that could happen to the feasible region:
 1. Could be empty
 2. Could be unbounded in one direction
 3. Could have some bounded region.

- Various people suggested plotting the objective function on this plot as well.
- Note that the objective function relates our two variables of interest to a third quantity *cost* – the expense of a particular diet.
- Cost doesn't appear anywhere on this plot yet. The point is that you can draw a line on this plot *only* after fixing a cost.
- So, for example, for $\text{cost}=0$, you get the line

$$2x_{pb} + x_h = 0$$

- In general, you have a whole family of such lines – one for each possible cost.
- Our question, then, is find the line with the lowest value that still falls within the feasible region.
- Note that everything is linear here. The point of interest *must* be at a vertex of the FR. (Or along an edge, but this is equivalent – why?)
- So the simplest thing we could do is to simply enumerate all the vertices of the polytope, evaluate the cost function there, and pick the smallest.
- This doesn't work in practice – why?
- How many vertices could there be in an n -d polytope? Well, suppose there are k constraints (hyperplanes), plus n “ $x \geq 0$ ” constraints for a total of $n + k$ hyperplanes. Each vertex is formed by the intersection of n hyperplanes, therefore there are $\binom{k+n}{n} = \frac{(k+n)!}{n!k!}$ vertices. For large k and n this is totally intractable. We need something smarter.
- Observation: if we know *one* vertex in the feasible region, we can evaluate its cost, as well as the cost of all of its neighbors.
- Effectively, we can trace down one of the edges that leads to the vertex to find a neighbor. Q: how many edges does a vertex have in n -space? A: just n , so this is a tractable operation.
- So of the n possible successor vertices, choose the one that decreases the cost function the most. Move there and repeat. So long as cost keeps decreasing, you're always improving your solution. Because of the linear cost function, *every vertex that is not the solution must have at least one cost-decreasing edge leading out of it.*
- If you ever find a vertex with no cost-decreasing edges, halt with the solution.
- In practice, a lot of linear algebra is involved in making this work. I won't drag you through it, for which I'm sure you're thankful, but there are some good resources on the web if you're interested.
- It *is* useful to know that *in the worst case*, the simplex method *still* has to examine all the vertices. I.e., all exponentially many of them. In practice, this *very* rarely happens, and people use simplex in real applications all the time.

- For a long time, it was unknown whether LP could be solved in poly time in worst case. In the mid-1980s, Karmakar showed an entirely *different* class of methods that weren't based on looking at vertices. His algorithm is provably polytime, though it's hard to make it run as well as simplex in practice.
- Thus, LP is known to be in P now.
- Interestingly, you can use a very closely related formulation to solve the TSP.
- Consider a TSP instance, with distances between cities i and j of d_{ij} . Let us use **indicator variables** (a.k.a. **decision variables**) $x_{ij} = 1$ iff we go between cities i and j in the tour and 0 otherwise. Then we can write a TSP instance as

$$\begin{array}{ll}
 \text{Minimize} & \sum_{i,j} d_{ij}x_{ij} \\
 \text{Subject to} & \sum_i x_{ij} = 1 \\
 & \sum_j x_{ij} = 1 \\
 & \sum_{i,j \in S} x_{ij} \leq |S| - 1 \quad \forall S \subset E
 \end{array}$$

- This says that we should minimize the sum of all edges included in the tour, that every city should have exactly one edge going into it and one edge going out of it. The final set of constraints says that no subset of the cities should have enough edges in it to form a subtour (i.e., there can only be one tour and it has to be a Hamiltonian tour).