

Linear Programming

The definitions of LP, and other pieces of the material appear in CLRS Chapter 29

The linear-time algorithm for LP in 2D from MMOM

Slides courtesy of Craig Gotsman

14.

Linear Programming – Example: designing non-fat multi-vitamins diet.

- Define:
 - i – types of foods ($1 \leq i \leq d$).
 - j – types of vitamins ($1 \leq j \leq n$).
 - x_i – **variables** (what we want to find). The amount of food of type i we need to consume
 - a_{ij} – the amount of vitamin j in one unit of food i .
 - c_i – the number of calories in one unit of food i .
 - b_j – minimal required amount of vitamin j .

- Constraints (we need to consume some minimal amount of each vitamin):

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1d}x_d &\geq b_1 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nd}x_d &\geq b_n \end{aligned}$$

Minimize: $c^T x$
Subject to: $Ax \geq b$

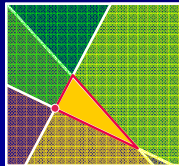
- Minimize: the total number of calories consumed:

$$C(x) = c_1x_1 + c_2x_2 + \dots + c_dx_d$$

24.

Linear Programming – The Geometry

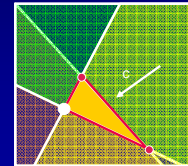
- Each constraint defines a half-space region in d -dimensional space.
- The *feasible region* is the (convex) intersection of these half-spaces.
- We will treat the case $d = 2$, where each constraint defines a *half-plane*.



34.

More Geometry

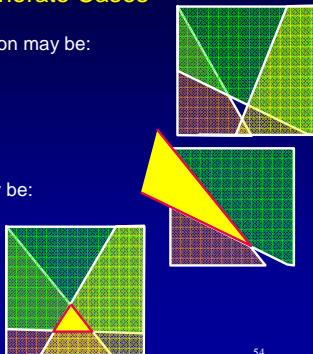
- The solution to the linear program is a point in the feasible region that is extreme in the direction of the target function.
- **Theorem:** Any bounded linear program that is feasible has a unique solution, which is a *vertex* of the feasible region.
- **Proof:** Convexity ...



44.

Degenerate Cases

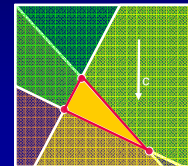
- The feasible region may be:
 - Empty
 - Unbounded
- The solution may be:
 - Not unique



54.

The Simplex Algorithm

- Assume WLOG that the cost function points "downwards".
- Construct (some of) the vertices of the feasible region.
- Walk edge by edge downwards until reaching a local minimum (which is also a global minimum).
- In \mathbb{R}^d , the number of vertices might be $\Theta(n^{\lfloor d/2 \rfloor})$.



64.

LP History

- ❑ Mid 20th century: Simplex algorithm, time complexity $\Theta(n \cdot d^2)$ in the **worst** case.
- ❑ 1980's (Khachiyan) ellipsoid algorithm with time complexity $\text{poly}(n, d)$.
- ❑ 1980's (Karmakar) interior-point algorithm with time complexity $\text{poly}(n, d)$.
- ❑ 1984 (Megiddo) – parametric search algorithm with time complexity $O(C_d n)$ where C_d is a constant dependent only on d . E.g. $C_d = 2^{d^2}$.
- ❑ The holy grail: An algorithm with complexity independent of d .
- ❑ In practice the **simplex** algorithm is used because of its linear *expected* runtime. Very very fast in practice. Available on many free and commercial libraries.⁷⁴

$O(n)$ Algorithm for LP in 1D

- ❑ Problem: Find the most extreme point on a line, which is in the intersections of half-lines.
- ❑ LP: find max x such that:
 - $X > a_i$ ($i=1,2,\dots,n$) and $X < b_i$
- ❑ Easily solved by checking if $\max\{a_i\} < \min\{b_i\}$
 - Return $\min\{b_i\}$ if yes. Otherwise there is no solution.
- ❑ Same idea if the problem is to find a point (x, y) on a given line l , and (x, y) maximizes $c_1x + c_2y$ s.t. $a_i x + b_i y < K_i$ for $i=1,\dots,n$

84.

$O(n^2)$ Incremental Algorithm for LP in 2D

- ❑ The idea:
 - Start by intersecting two halfplanes.
 - Add halfplanes one by one and update optimal vertex by solving one-dimensional LP problem on new line *if needed*.

94.

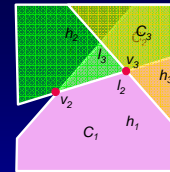
Incremental Algorithm - Symbols

h_i the i^{th} half plane

l_i the line that defines h_i

C_i the feasible region after i constraints

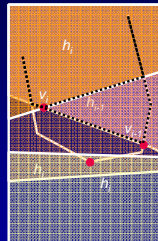
v_i the optimal vertex of C_i



104.

Incremental Algorithm Basic Theorem

- ❑ Theorem:
 1. if $v_{i-1} \in h_i$, then $v_i = v_{i-1}$. // $O(1)$ check, nothing to do
 2. if $v_{i-1} \notin h_i$, then either $C_i = \emptyset$ // terminate or $C_i = C_{i-1} \cap h_i$ and v_i lies on l_i // run 1D LP
- ❑ Proof:
 1. Trivial. Otherwise v_i would not have been optimum before.



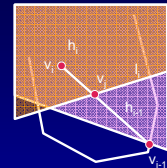
114.

Basic Theorem - Cont.

2. Assume that v_i is not on l_i . v_i must be in C_{i-1} . By convexity, also the segment $v_i v_{i-1}$ is in C_{i-1} .

Consider point v_i - the intersection of $v_i v_{i-1}$ with l_i . v_i is in both C_{i-1} and C_i , and is better than v_i .

Contradiction.



124.

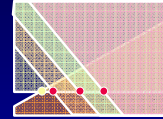
Finding v_i given l_i (one-dimensional LP)

- ❑ Call the 1D algorithm – (repeated with the new notation):
- ❑ Intersect each h_j ($j < i$) with l_i , generating $i-1$ rays representing (unbounded) intervals.
- ❑ Intersect the $i-1$ intervals in $O(i)$ time.
- ❑ If the intersection is empty then report no solution, else report the lowest point.

134.

Complexity Analysis

$$T(n) = \sum_{i=3}^n O(i) = O(n^2)$$



144.

Incremental Algorithm – $O(n)$ Randomized Version

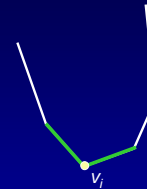
- ❑ Exactly like the deterministic version, only the order of the lines is random.
- ❑ **Theorem:** The expected runtime of the random incremental algorithm (over all $n!$ permutations of the input constraints) is $O(n)$.
- ❑ (idea of proof – very similar to the closest pair example).
 - When adding the i 'th plane, the probability that we need to solve a 1D LP is small $2/i$,
 - With probability $(i-2)/i$, we need to do almost nothing, since $v_i = v_{i-1}$

154.

Probability Analysis

Backward analysis

- ❑ **Question:** When given a solution after i half-planes, what is the probability that the *last* half-plane affected the solution?
- ❑ **Answer:** Exactly $2/i$, because a change can occur only if the last half-plane inserted is one of the two half-planes thru v_i .
(note that v_i depends on the i half-planes, but not on their order)



164.

Finishing the analysis

So in the i 'th stage we are spending $O(1)$ time with probability $(i-2)/i$, and $O(i)$ time with probability $2/i$, so the expected work in this stage is $O(i) \cdot 2/(i+1) = O(1)$.

Hence the total expected time is $O(n)$.

This argument can be done more formal using random variables.

174.

Expected time

Let T_i denote the expected time spent at stage i . Then $T_i=1$ with probability $(i-2)/i$ and $T_i=i$ with probability $2/i$

$$E(T_i) = \sum_{j=1}^{\infty} j \Pr(T_i = j) = 1 \Pr(T_i = 1) + i \Pr(T_i = i) = 3$$

Hence the expected total time is

$$E\left(\sum_{i=3}^n T_i\right) = \sum_{i=3}^n E(T_i) = \sum_{i=3}^n 3 = O(n)$$

184.