# 2 May 2018 Linear Programming

Linear programming (LP) is the problem of maximizing or minimizing a linear function of $\vec{x} \in \mathbb{R}^n$ under a set of linear constraints.

$$\begin{aligned} &\max \text{ (or min)} && c \cdot x && \text{[obj function]} \\ &\text{subject to} && a_1 \cdot x \le b_1 \\ & && a_2 \cdot x \le b_2 && \text{[constraints]} \\ & && \vdots \\ & && a_n \cdot x \le b_n \end{aligned} \quad \Longleftrightarrow \quad Ax \le b$$

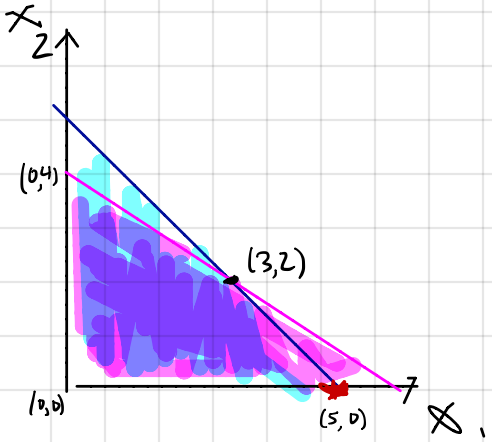
$A$ = constraint matrix $= \begin{bmatrix} — & a_1 & — \\ — & a_2 & — \\ & \vdots & \\ — & a_n & — \end{bmatrix}$

$b$ = right-hand side vector $= \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$

**Ex.** maximize $3x_1 + 2x_2$

subject to

$$\begin{aligned} x_1 + x_2 &\le 5 \\ 2x_1 + 3x_2 &\le 12 \\ x_1 &\ge 0 \\ x_2 &\ge 0 \end{aligned}$$




$$\max \begin{bmatrix} 3 \\ 2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\text{s.t.} \quad \begin{bmatrix} 1 & 1 \\ 2 & 3 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \le \begin{bmatrix} 5 \\ 12 \\ 0 \\ 0 \end{bmatrix}$$

The set of vectors $\vec{x}$ that satisfy the constraints is an intersection of halfspaces (one per constraint) so it is a convex polyhedron.

The optimum is always achieved at a vertex of the polyhedron. (Unless opt $= \infty$.)

The polyhedron may have exponentially many vertices.

E.g. $-1 \le x_1 \le 1$, $-1 \le x_2 \le 1$, ..., $-1 \le x_n \le 1$.

$2n$ inequalities, solution set is $[-1,1]^n$ with $2^n$ vertices.

How to solve LP efficiently? (Take 6820 if you want to know.)

(1) "Simplex algorithm"

A local search algorithm that walks along vertices, always moving to a neighbor so as to improve the objective function value.

Every known deterministic implementation has exponential worst-case running time.

Open question: poly-time deterministic simplex algorithm?

In practice simplex is a very fast method to solve LP.

(2) "Ellipsoid algorithm" poly-time in theory, slow in practice.

higher-dimensional generalization of binary search.

Draw an ellipsoid around the optimum, iteratively shrink it until only one vertex remains inside.

(3) "Interior-point methods" (Take OR courses on optimization if you want to know.) Fast in both theory and practice.

For 4820 purposes: LP can be solved in polynomial time.

Network flow as a LP:

$$\text{maximize} \quad \sum_{e \text{ out of } s} f(e)$$

$$\text{subject to} \quad 0 \le f(e) \le c(e) \qquad \forall e \in E$$

$$\boxed{\sum_{e \text{ out of } v} f(e) = \sum_{e \text{ into } v} f(v)} \qquad \forall v \neq s,t$$

$$\Downarrow$$

$$\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ into } v} f(e) \le 0$$

$$\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ into } v} f(e) \ge 0$$

Shortest path as an LP:

$$\text{minimize} \quad \sum_{e \in E} \text{length}(e) \cdot x_e$$

An extreme point solution is a shortest path. $\Longrightarrow$

$$\text{subject to} \quad \sum_{e \text{ out of } s} x_e = 1 \qquad \sum_{e \text{ into } t} x_e = 1$$

$$\sum_{e \text{ out of } v} x_e = \sum_{e \text{ into } v} x_e \qquad \forall v \neq s,t$$

# MIN VTX COVER as a LP with integrality constraints.

Given $G$, find a vertex cover with min # of vertices.

Use variables $x_v$ whose "intended interpretation" is that $x_v = 1$ if $v \in$ {vertex cover set}, 0 if not.

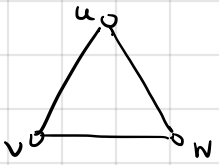
$$\min \sum_{v \in V} x_v$$

$$\text{s.t.} \quad x_u + x_v \geq 1 \qquad \forall (u,v) \in E$$
$$0 \leq x_v \leq 1 \qquad \forall v \in V$$

} LP relaxation of Vertex Cover

The solution set of the LP relaxation includes vectors in $\{0,1\}^n$ — these are in 1:1 correspondence with vertex covers of $G$.

It also includes fractional vectors that don't correspond to vertex covers.




Vertex Covers

$$\begin{bmatrix} x_u \\ x_v \\ x_w \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Other LP solutions

$$\begin{bmatrix} x_u \\ x_v \\ x_w \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \end{bmatrix}.$$

Obj func: 2 2 2 3 1.5

Approximation Algorithm for Vertex Cover.