Linear Programming

The definitions of LP, and other pieces of the material appear in CLRS.
In the diet problem, we will have to compute two values $x$ and $y$.

- $x$ indicates how many **bananas** we plan to consume daily.
- $y$ indicates how many **oranges** we plan to consume daily.

The goal is to find a healthy diet that is as cheap as possible.
Define: (amount consumed per day)

- types of foods: \{x-bananas, y-oranges\}
- \( j \) – types of vitamins \((1 \leq j \leq n)\).
- \( x \) – number of oranges we recommend daily
- \( y \) – number of bananas we recommend daily
  
  // these are the only unknown we have to compute.

- \( a_{i,j} \) – the amount of vitamin \( i \) in a unit of food \( j \).
  
  \((j=1 \text{ for bananas, } j=2 \text{ for oranges})\). These are given constants.

- \(c_1\) – the cost of an banana.
- \(c_2\) – the cost of an orange
- \( b_i \) – minimal daily required amount of vitamin \( j \), for the diet to be healthy, \( i=1..n \)

**Minimize**: minimize the cost of a healthy diet

\[
C((x, y)) = c_1 x + c_2 y
\]
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.

The equation $y = ax + b$ defines a line, which we could also write as $(-a)x + (1)y = b$. Pointed one one side of this line forms a half-plane.

\[
\begin{align*}
a_1x + a_2y &\geq b \\
a_1x + a_2y &\leq b
\end{align*}
\]
A shape $S$ is convex if for every two points $p,q$ inside the shape, the segment connecting these points is also inside the shape.

If $A$ and $B$ are two convex shapes, then their intersection $A \cap B$ (namely, points that belong both to $A$ and to $B$) is also convex.

A half-plane is convex.

Conclusion: Intersection of half-planes is convex.
In our context, a vector $\vec{v}$ in the $d$-dimension space, is an ordered list of $d$ numbers $\vec{v} = (v_1 \ldots v_d)$.

If we have 2 vectors $\vec{v} = (v_1, v_2, v_3, \ldots v_d)$ and $\vec{u} = (u_1, u_2, u_3, \ldots, u_d)$, we define the dot product $\vec{u} \cdot \vec{v}$ between them as follows:

$$\vec{u} \cdot \vec{v} = u_1v_1 + u_2v_2 + \ldots + u_du_d = \sum_{i=1}^{d} v_iu_i$$

Note: $\vec{u} \cdot \vec{u} = \vec{v} \cdot \vec{v}$

We distinguish between a vector and a scalar. A scalar is a single number, while a vector is a list of numbers.

Let $\vec{v} = (a, b)$, we can think about it as an arrow from the point $(0,0)$ to the point $(a, b)$.

Let think about all the points $\vec{p} = (a, b)$ for which $\vec{p} \cdot \vec{v} = a \cdot v_1 + b \cdot v_2 = 0$. They form a line $L$. We can write $\ell := \{p \mid p \cdot \vec{v} = 0\}$, or sometimes abbreviated as $\ell : \vec{p} \cdot \vec{v} = 0$

They are all along the line orthogonal to $\vec{v}$.

In general, if $\vec{q}$ is a point, then the line $\vec{p} \cdot \vec{v} = \vec{v} \cdot \vec{q}$ is passing through $\vec{q}$ and orthogonal to $\vec{v}$.
Define: (amount consumed per day)
- types of foods: {x-bananas, y-oranges} vectors

- $j$ – types of vitamins ($1 \leq j \leq n$).
- $x$ – number of oranges we recommend daily
- $y$ – number of bananas we recommend daily

// these are the only unknown we have to compute
\[ \vec{x} = (x, y) = (\text{#bananas/day, #oranges/day}) \]

- $a_{i,j}$ – the amount of vitamin $i$ in a unit of food $j$. \[ \vec{a}_i = (a_{i,1}, a_{i,2}) \] ($j=1$ for bananas, $j=2$ for oranges). These are given constants.

- $b_i$ – minimal daily required amount of vitamin $j$, for the diet to be healthy, $i=1..n$

- $c_1$ – the cost of a banana.
- $c_2$ – the cost of an orange \[ \vec{c} = (c_1, c_2) \]

Minimize: minimize the cost of a healthy diet
\[ C((x, y)) = c_1 x + c_2 y \]
\[ \vec{c} \cdot \vec{x} \text{ should be as small as possible} \]

\[ a_{11} x + a_{12} y \geq b_1 \]
\[ \vdots \]
\[ a_{n1} x + a_{n2} y \geq b_n \]
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 solution, which is a vertex of the feasible region.

- **Proof:** Convexity …
Degenerate Cases

- The feasible region may be:
  - Empty
  - Unbounded

- The solution may be:
  - Not unique
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 $\Theta(n^{\lfloor d/2 \rfloor})$. 
LP problems - definition and history

Definition: An optimization problem is a **Linear Programming Problem (LP)** if it asks us to find a set of parameters (a vector) that maximizes a linear cost function, which bounded by a set of linear constrains. That is, the solution must be in the intersection of given half space.

The **Simplex Algorithm** is usually used to solve such problems: It has an exponential worst case, but almost always it is extremely fast. So practically, if we could express a problem as an LP problem, we could considered it solved.

**History**
- 1947: George Dantzig  Simplex algorithm. Extremely efficient I’m practice. Exponential in very rare cases.
- Since it is so efficient, if we have a problem and we could phrase it as a linear programming problem (constrains are half-spaces, and linear cost function)
- 1980’s (Khachiyan) ellipsoid algorithm with time complexity poly($n, d$).
- 1980’s (Karmakar) interior-point algorithm with time complexity 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.
Problem: Given a line \( \ell \) and a set of half-planes \( \{ h_1 \ldots h_m \} \) find the lowest point on \( \ell \) which is inside \( \bigcap_{i=1}^{m} h_i = h_1 \cap h_2 \cap \ldots \cap h_m \).

Let \( \ell_i \) be the line bounding \( h_i \).

Each half-plane either contains the point \((0, +\infty)\) or contains the point \((0, -\infty)\).

Consider first only half-plane containing \((0, +\infty)\).

Compute \( p_i = \ell \cap \ell_i \) and let \( p_{\text{max}} \) the highest such point (\( p_2 \) in the example). Any solution to the LP must be on the portion of \( \ell \) above the \( p_{\text{max}} \).

Similarly, find the half-planes contain \((0, -\infty)\). Compute their intersections with \( \ell \). Let \( q_{\text{min}} \) be the lowest intersection points.

Any solution to the LP which is on \( \ell \) must be between \( p_{\text{max}} \) and \( q_{\text{min}} \).

Running Time: \( T(m) = c \cdot m \)
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.
Incremental Algorithm - Notation

- $h_i$ 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 $C_i$

Cost function to minimize: $c(x,y) = y$. Returns the lowermost point in feasible region.
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 it is sufficient to look for \( v_i \) on \( l_i \) using 1DLP (rather than searching in the whole plane)

Conclusion: If there is no solution on \( l_i \), then there is no solution at all. The feasible region is empty.

Proof:

1. Trivial. Otherwise \( v_i \) would not have been optimum before.
Basic Theorem - case 2.

Assume that \( v_i \) is not on \( l_i \).

\( v_i \) must be in \( C_{i-1} \) By convexity, also the segment \( \overline{v_{i-1}v_i} \) (from \( v_i \) to \( v_{i-1} \)) is in \( C_{i-1} \).

Assume WLOG: Our cost function pushes us downward.

Consider point \( q \): the intersection of the segment \( \overline{v_{i-1}v_i} \) with \( l_i \).

Notice: \( q \) is also in \( h_i \) and in is \( C_{i-1} \). It is lower than \( v_i \).

Contradicting the assumption that \( v_i \) is not on \( l_i \).
Same theorem – in an algorithmic terms

Comute $h_1$, $h_2$ and $v_2$

For $i=3...n$
{
  1. Check if $v_{i-1} \in h_i$. If yes, then $v_i = v_{i-1}$. // $O(1)$,
  `ELSE
  2. // $v_i$ must be on the line $l_i$. call 1D-LP($l_i$, $h_1...h_{i-1}$)
  3. If 1D-LP does not have a solution on $l_i$. - stop. There is no solution anywhere else.
    `ELSE
    set $v_i$ to be the solution that 1D-LP found.
}
Complexity Analysis

- Worst case, each new constrain $h_i$ forces solving a new 1DLP

$$T(n) = \sum_{i=3}^{n} c \cdot i = \Theta(n^2)$$
Incremental Algorithm – $O(n)$ Randomized Version

- Exactly like the deterministic version, only the order of the lines is random.

- Assume that the halfplane are stored in an array $A[1..n]$.

- For $i=3..n$ {
  - Pick a random index $k$ between 1 to $n$.
  - Insert $A[k]$ as the $i$’th half-line $h_i$
  - Swap $A[k] \leftrightarrow A[n]$
  - $n = n-1$
}

- Every permutation is equally likely

- **Theorem:** The expected runtime of the random incremental algorithm (over all $n!$ permutations of the input constraints) is $O(n)$. 
Probability Analysis

Backward analysis

- Recall that if $v_{i-1}$ violates $h_i$ then $v_i \in \ell_i$. In words, the new optimum solution must on the line bounding $h_i$.

- **Question**: What is the probability that at the i’th step of the algorithm, $v_{i-1}$ violates $h_i$? (that is $v_{i-1} \neq v_i$).

- **Answer**: Exactly $\frac{2}{i}$. Here is the reason:
  - $v_i$ is determined by two half-planes. It does not care it which order the half-planes were inserted.
  - The probability that one of them is $h_i$ is $2/i$.
  - The probability that $h_i$ is one of the other halfplanes is $\frac{i-2}{i}$ which is almost 1.

- Conclusion: At the i’th step, the expected work is $1\frac{i-2}{i} \cdot 1 + c \cdot \frac{2}{i} = 1 + 2c = constant$.

- Therefore, the expected work for the algorithm is (a bit hand wave) $n+cn=O(n)$. Linear Algorithm

- YAY.
False Claim:

- The probabilistic analysis is for the average input. Hence there exist bad sets of constraints for which the algorithm’s expected runtime is *more* than $O(n)$, and there exist good sets of constraints for which the algorithm’s expected runtime is *less* than $O(n)$.

True Claim:

- The probabilistic analysis is valid for *all* inputs. The expected complexity is over all *permutations* of this input.
LP in 3D

- Now the input is a collection of **half-spaces** \( \{h_1 \ldots h_n\} \).

  Now \( l_i \) is the plane bounding \( h_i \). (notations are analogous to the 2D case).

  We will define \( v_3 \) as the intersection of the **planes** \( l_1, l_2 \) and \( l_3 \).

  We insert the other halfspaces \( \{h_4 \ldots h_n\} \) at a random order, and update \( v_i \) according to the following 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 the solution (if exists) is on \( l_i \).
     
     run \( v_i = \text{2DLP}( h_1 \cap l_i, h_2 \cap l_i, h_3 \cap l_i, \ldots, h_{i-1} \cap l_i ) \).

     Terminates if there is no solution (that is, \( C_i = \emptyset \) )
LP in 3D and higher dimension

In 3D, the worst case running time is $\Theta(n^3)$ (prove).

However, the expected running time is $O(n)$. In general, the running time in d-dimension is $O(d! \cdot n)$. That is, linear in any fixed (and small) dimension.