**Halfplanes and intersection of halfplanes**

**Linear Programming, healthy diets and ILP**

- 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.

**Example of an LP: The Diet problem**

- In our context, a vector $\vec{v}$ in the $d$-dimensional space is an ordered list of numbers $\vec{v} = (v_1, \ldots, v_d)$.
- For two vectors, $\vec{v} = (v_1, v_2, \ldots, v_d)$ and $\vec{w} = (w_1, w_2, \ldots, w_d)$, we define the dot product $\vec{v} \cdot \vec{w}$ as follows:
  \[ \vec{v} \cdot \vec{w} = v_1w_1 + v_2w_2 + \cdots + v_dw_d = \sum_{i=1}^{d} v_iw_i \]
- Note: $\vec{v} \cdot \vec{w} = \vec{w} \cdot \vec{v}$, and $\vec{v} \cdot (\vec{w} + \vec{z}) = \vec{v} \cdot \vec{w} + \vec{v} \cdot \vec{z}$.
- The length of the vector $\vec{v}$, denoted $|\vec{v}|$, is $\sqrt{\vec{v} \cdot \vec{v}}$ (Pythagoras).
- $\vec{v} \cdot \vec{w} = \vec{w} \cdot \vec{v}$, and $\vec{w} \cdot (\vec{v} + \vec{v}) = \vec{w} \cdot \vec{v} + \vec{w} \cdot \vec{v}$.
- Dot product strongly correlated to the angle between the vectors. If $\vec{v} \cdot \vec{w} = 0$, then they are orthogonal to each other.

**Dot product notation (review from Linear Algebra)**

- In our context, a vector $\vec{v}$ in the $d$-dimensional space is an ordered list of numbers $\vec{v} = (v_1, \ldots, v_d)$.
- For two vectors, $\vec{v} = (v_1, v_2, \ldots, v_d)$ and $\vec{w} = (w_1, w_2, \ldots, w_d)$, we define the dot product $\vec{v} \cdot \vec{w}$ as follows:
  \[ \vec{v} \cdot \vec{w} = v_1w_1 + v_2w_2 + \cdots + v_dw_d = \sum_{i=1}^{d} v_iw_i \]
- Note: $\vec{v} \cdot \vec{w} = \vec{w} \cdot \vec{v}$, and $\vec{v} \cdot (\vec{w} + \vec{z}) = \vec{v} \cdot \vec{w} + \vec{v} \cdot \vec{z}$.
- The length of the vector $\vec{v}$, denoted $|\vec{v}|$, is $\sqrt{\vec{v} \cdot \vec{v}}$ (Pythagoras).
- $\vec{v} \cdot \vec{w} = \vec{w} \cdot \vec{v}$, and $\vec{w} \cdot (\vec{v} + \vec{v}) = \vec{w} \cdot \vec{v} + \vec{w} \cdot \vec{v}$.
- Dot product strongly correlated to the angle between the vectors. If $\vec{v} \cdot \vec{w} = 0$, then they are orthogonal to each other.

- 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 (sometimes) think about it as an arrow from the point $(0,0)$ to the point $(a,b)$.
- Fix $\vec{q} = (a, b)$. Think about all the points $\vec{x} = (x,y)$ for which $\vec{v} \cdot \vec{x} = a \cdot x + b \cdot y = 0$. These points form a line $\ell'$. We can write $\ell' = \{ \vec{x} | \vec{v} \cdot \vec{x} = 0 \}$, or sometimes abbreviated as $\ell' = \{ \vec{x} | \vec{v} = 0 \}$.
- The line $\ell'$ is orthogonal to $\vec{v}$.
- In general, if $\vec{q}$ is a point, then the line $\ell = \{ \vec{x} | \vec{x} = \vec{q} + t \vec{v} \}$ is passing through $\vec{q}$ and orthogonal to $\vec{v}$.
- In higher dimensions, all stay the analogous $\vec{x} = (x,y,z)$. Fix $\vec{v} = (a,b,c)$. The set of points $\ell'' = \{ \vec{x} = (x,y,z) | \vec{v} \cdot \vec{x} = 0 \}$ form a plane in 3D.

- ![Diagram](https://via.placeholder.com/150) In many cases, we can think about a vector as a point and vice versa.
The Diet Problem as an LP problem

- We will denote by \( x \) the number of bananas we consume per day.
- We will denote by \( y \) the number of bananas we consume per day.
- These \( x \) and \( y \) are the only unknown, and what we need to optimize.

\[ \mathbf{x} = (x, y) = (\#\text{bananas/day}, \#\text{oranges/day}) \]

For a diet to be healthy, we need to get a sufficient dose (quantity in grams) of each type vitamins. Assume \( n \) types of vitamins \( 1 \ldots n \)

**Given:**
- \( a_{i,1} \) the amount of vitamin \( i \) in banana.
- \( a_{i,2} \) the amount of vitamin \( i \) in an orange.

**Given:**
- \( \mathbf{b} \) – minimum required daily dose of vitamin \( i \) \((i=1..n)\)
- \( \mathbf{c} \) – cost vector

The daily cost of our diet is

Minimize: minimize the cost of a healthy diet

Link

---

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

\[ a_1 x + a_2 y \geq b \]
\[ a_1 x + a_2 y \leq b \]

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 $\Theta(n \frac{d^2}{2})$.

**Linear Programming in d dimension - Example**

- Define: (amount amount consumed per day)
  - $n$ – types of foods (1 banana, 2 oranges, 3 avocado, …) This is the dimension of the LP problem.
  - $a$ – the amount of food $j$ consumed daily $1 \leq j \leq d$.
  (these are the $d$ unknowns that we need to optimize).
- $x$ – a vector of unknowns.
- $b$ – minimal daily dose for vitamin $i$. (1 ≤ i ≤ n)
- $a_i$ – the amount of vitamin $i$ in one unit of food $j$.
- $c$ – the cost of a unit of food $j$ (1 ≤ j ≤ d).

**LP**

minimize $c^T x$.

Such that (s.t.)

for every $1 \leq i \leq n$:

- $a_i \cdot x \geq b_i$.

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

**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 constraints. 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 consider 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 (constraints are half-spaces, and linear cost function)
- 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 $O(C_{\tau} n)$ where $C_{\tau}$ is a constant dependent only on $d$. E.g. $C_{\tau} = 2^{2^{\sqrt{d}}}$.
- The holy grail: An algorithm with complexity independent of $d$.
- In practice the simplex algorithm is used because of its linear expected runtime.

**O(n log n) 2D Linear Programming (details left as hw)**

- Input:
  - $n$ half planes.
  - Cost function that WLOG “points down”.
- Algorithm:
  Partition the $n$ half-planes into two groups.
  $S$ are all halfplanes contain the point $(0, \infty)$
  $S'$ all other halfplanes contain the point $(0, -\infty)$
  Sort them by slopes
  Compute the upper envelop $U(S)$ and the lower envelop $L(S')$
  (using question from hw1)
  Scan simultaneously from left to right, and Computer intersection of two envelopes - they can intersect only at 2 points (why).
  Evaluate cost function at each vertex.
Toward a faster algorithm in small dimensions

- 1-dimensional linear programming
- Given 2n constants (constrains) $\alpha_1, \alpha_2, \ldots, \alpha_n, \beta_1, \beta_2, \ldots, \beta_n$ (not necessarily sorted)
- find in $O(n)$ time the minimum $x$ such that

What is the feasible region? Could it be that the problem has no solution?

Answer

Feasible solution \( \{x \mid \max(\alpha_i) \leq x \leq \min \beta_j\} \)

O(n^2) Incremental Algorithm

- 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.

1D – LP($\ell, h_1, \ldots, h_m$). Solving LP in 2D, but the solution must be on a given line $\ell$

Problem: Given a line $\ell$ and a set of half-planes \{h_1, h_2, \ldots, h_n\}, find the lowest point on $\ell$ which is inside $\bigcap_{i=1}^{n} h_i$.

- Each half-plane either contains the point $(0, +\infty)$ or contains the point $(0, -\infty)$.
- Consider first only half-plane containing $(0, +\infty)$.
- Let $\ell_i$ be the line bounding $h_i$. Compute $p_i = \ell_i \cap \ell$.
- Let $p_{\text{max}}$ be the highest such point ($p_2$ in the example). Any solution to the LP must be on the portion of $\ell$ above $p_{\text{max}}$.
- 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 $q_{\text{min}}$.
- Note that it is possible that $q_{\text{min}}$ is below $p_{\text{max}}$. In this case, we have no solution on $\ell$.

Incremental Algorithm - Notation

- $h_i$ is the $i$'th constrained half-plane
- $\ell_i$ is the line bounding $h_i$
- $C_i = h_1 \cap h_2 \cap \ldots \cap h_i$ is the feasible region of the first $i$ constrains
- $v_i$ is the optimal solution to the first $i$ constrains - it is the lowest point of $C_i$

Cost function to minimize: $c(x, y) = y$.

Returns the lowermost point in feasible region.
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 it is sufficient to look for $v_i$ on $\ell_i$ 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.
  2. - in the next slide

Same theorem – in an algorithmic terms

Compute $C_i = h_1 \cap h_2$, and $v_2$

For $i = 3 \ldots n$

1. Check if $v_{i-1} \in h_i$. If yes, then $v_i = v_{i-1}$. // O(1),
2. ELSE
3. If $v_i$ must be on the line $\ell_i$ call 1D-LP($\ell_i, h_i, \ldots h_{i-1}$)
4. If 1D-LP does not have a solution on $\ell_i$ - stop. There is no solution anywhere.
   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

- $T(n) = \sum_{i=3}^{n} c \cdot i = \Theta(n^2)$
Theorem: The expected time for the randomize version is $O(n)$. 

Backward analysis

- Recall that if $v_{i-1}$ violates $h_i$ then $v_i \notin \ell_i$. 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 which order the half-planes were inserted.
  - The probability that one of them is $h_i$ is $\frac{1}{2i}$.
  - 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 $\frac{i-2}{i} \cdot 1 + c \cdot \frac{2}{i} = 1 + 2c = \text{constant}$.

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

- YAY.

Just to Make Sure ... 

- 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 $v_i = 2\text{DLP}(h_1 \cap l_i, h_2 \cap l_i, h_3 \cap l_i, \ldots, h_{i-1} \cap l_i)$.

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 $O(d! n)$. That is, linear in any fixed (and small) dimension.
Integer Linear Programming (ILP)

- Linear programming problems at which values of the computed variables must be integers are called Integer Linear Programming (ILP) problems.
- If only some of the variables have to integers, we call them Mixed Integer Linear Programming problems.
- There is a huge number of problems that could be phrased as ILP. (include many NP-hard problems, where no polynomial-time algorithms exist)
- A few libraries could handle them, including CPLEX.
- Running time could varies a lot, and could be extremely slow for some instances.
- Yet extremely useful for instances when actual running time is acceptable.
- Also useful for comparing fast heurists to global optimum.

Another useful for comparing fast heurists to global optimum. Yet extremely useful for instances when actual running time is acceptable. Running time could varies a lot, and could be extremely slow for some instances. A few libraries could handle them, including CPLEX.

Linear Programming

If only some of the variables have to integers, we call them Mixed Integer Linear Programming problems.

There is a huge number of problems that could be phrased as ILP (include many NP-hard problems, where no polynomial-time algorithms exist.) A few libraries could handle them, including CPLEX.

Also useful for comparing fast heurists to global optimum.

Vertex Cover and ILP

- Given: A graph G(V,E). A subset C ⊆ V is a vertex cover if every edge(u,v) ∈ E we have either u ∈ C or v ∈ C or both.
- Finding the min-cardinality Vertex Cover is NP-Hard
- ILP for this problem: the variables are x_1…x_n. All are integers and between 0 and 1.
- v_i ∈ C iff x_i = 1 (for i = 1…n)

{\text{minimize}} \sum_{i=1}^{n} x_i

s.t.

x_i + x_j \geq 1 \quad \forall (v_i, v_j) \in E

Min-Weight Vertex Cover and ILP

- Sometimes the LP (instead of the ILP) could help us finding good approximations
- Given: A graph G(V,E). Each vertex v_i is given with a weight w_i > 0. Think about it as the cost of this vertex.
- A subset C ⊆ V is a vertex cover if every edge(u,v) ∈ E we have either u ∈ C or v ∈ C or both.
- The cost of C is the sum of weights of vertices in C.
- Finding the min-cardinality Vertex Cover is NP-Hard
- ILP for this problem: the variables are x_1…x_n. All are integers and between 0 and 1.
- v_i ∈ C iff x_i = 1 (for i = 1…n)

{\text{minimize}} \sum_{i=1}^{n} W_i x_i

s.t.

- 0 \leq x_i \leq 1 and an integer, for every x_i
- x_i + x_j \geq 1 \quad \forall (v_i, v_j) \in E
Art Gallery - on the board

- Given a polygon, find a subset of the vertices that sees every other vertex
- Let $Vis(i)$ be the set of vertices that vertex $i$ sees. $Vis(K) = \{G, D, C, A, K, J, I, H\}$
- For a vertex $v_i$, we set $x_i = 1$ if we place a guard at $v_i$. Otherwise $v_i = 0$
- As usual, $x_i$ are integers between 0 to 1.

$\text{minimize} \sum_{i=1}^{n} x_i$

s.t.

$\sum_{k \in Vis(i)} x_k \geq 1 \quad \forall 1 \leq i \leq n$

$Vis(K) = \{G, D, C, A, K, J, I, H\}$