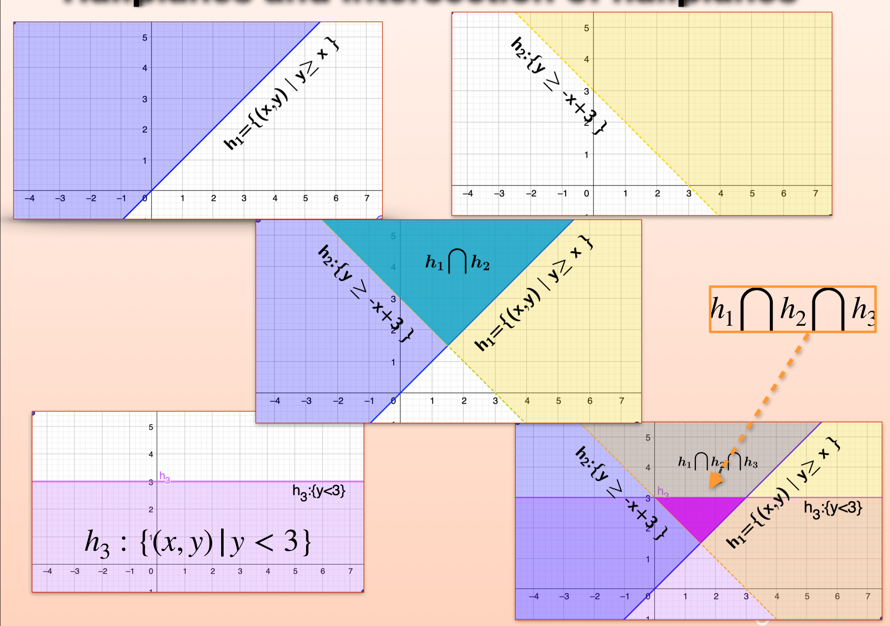
Linear Programming, healthy diets and ILP

Example of an LP: The Diet problem

- 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.

Halfplanes and intersection of halfplanes



Dot product notation (review from Linear Algebra)

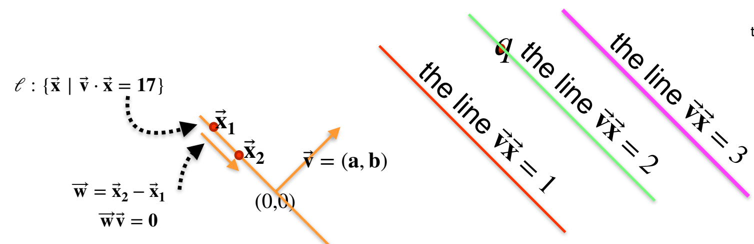
- In out context, a **vector** $\vec{\mathbf{v}}$ in the d-dimension space, is an ordered list of *d* numbers $\vec{\mathbf{v}} = (v_1 \dots v_d)$.
- For two vectors, $\vec{\mathbf{v}} = (v_1, v_2, v_3, \dots v_d)$ and $\vec{\mathbf{u}} = (u_1, u_2 . u_3, \dots, u_d)$, we define the **dot product** $\vec{\mathbf{u}} \cdot \vec{\mathbf{v}}$ as follows:

$$\vec{\mathbf{u}} \cdot \vec{\mathbf{v}} = u_1 v_1 + u_2 v_2 + \dots + u_d u_d = \sum_{i=1}^{d} u_i v_i$$

- Note: $\vec{\mathbf{u}} \cdot \vec{\mathbf{u}} = \vec{\mathbf{v}} \cdot \vec{\mathbf{u}}$, and $\vec{\mathbf{u}} \cdot (\vec{\mathbf{v}} + \vec{\mathbf{w}}) = \vec{\mathbf{u}} \cdot \vec{\mathbf{v}} + \vec{\mathbf{u}} \cdot \vec{\mathbf{w}}$.
- The length of the vector $\vec{\mathbf{v}}$, denoted $|\vec{\mathbf{v}}|$ is $\sqrt{\vec{\mathbf{v}} \cdot \vec{\mathbf{v}}}$ (Pythagoras).
- $\vec{\mathbf{u}} \cdot \vec{\mathbf{v}} = \vec{\mathbf{v}} \cdot \vec{\mathbf{u}}$, and $\vec{\mathbf{u}} \cdot (\vec{\mathbf{v}} + \vec{\mathbf{w}}) = \vec{\mathbf{u}} \cdot \vec{\mathbf{v}} + \vec{\mathbf{u}} \cdot \vec{\mathbf{w}}$.

- Dot product strongly correlated to the angle between the vectors. If $\vec{\mathbf{u}} \cdot \vec{\mathbf{v}} = \mathbf{0}$, 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 $\vec{\mathbf{v}} = (a, b)$. We can (sometimes) think about it as an arrow from the point (0,0) to the point (a, b).
- Fix $\vec{\mathbf{v}} = (a, b)$. Think about all the points $\vec{\mathbf{x}} = (x, y)$ for which $\vec{\mathbf{v}} \cdot \vec{\mathbf{x}} = a \cdot x + b \cdot y = 0$. These points form a line ℓ. We can write $\ell := \{\vec{\mathbf{x}} \mid \vec{\mathbf{v}} \cdot \vec{\mathbf{x}} = \mathbf{0}\}$, or sometimes abbreviated as $\ell : \vec{\mathbf{x}} \cdot \vec{\mathbf{v}} = 0$
- The line ℓ is orthogonal to $\vec{\mathbf{v}}$.
- In general, if **q** is a point, then the line $\vec{\mathbf{v}} \cdot \vec{\mathbf{x}} = \vec{\mathbf{v}} \cdot \vec{\mathbf{q}}$ is passing through q and orthogonal to $\vec{\mathbf{v}}$.
- In higher dimensions, all stay the analogous. $\vec{\mathbf{x}} = (x, y, z)$. Fix $\vec{\mathbf{v}} = (a, b, c)$. The set of points $\ell := \{\vec{\mathbf{x}} \in \mathbb{R}^3 \mid \vec{\mathbf{v}} \cdot \vec{\mathbf{x}} = \mathbf{0}\}$ form a plane in 3D.



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.**

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

- 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…n
- ***Given:*** $a_{i,1}$ – the amount of vitamin *i* in banana. $a_{i,2}$ the amount of vitamin *i* in an orange.

$\vec{\mathbf{a}}_{\mathbf{i}} = (a_{i,1}\ ,\ a_{i,2})$

- **Given:** b_i – minimum required daily dose of vitamin i *(i=1..n)*

$$\vec{\mathbf{a}}_{\mathbf{i}} \cdot \vec{\mathbf{x}} = \underbrace{a_{i,1} \cdot x}_{\text{vitamin i from bananas}} + \underbrace{a_{i,2} \cdot y}_{\text{vitamin i from oranges}} \geq b_i$$

$$a_{11}x + a_{12}y \geq b_1$$
$$\vdots$$
$$a_{n1}x + a_{n2}y \geq b_n$$

- ***Given:*** c_1 – the cost of a banana (dollars/unit). And given: c_2 the cost of an orange.
- $\vec{\mathbf{c}} = (c_1, c_2)$ *is the cost vector*
- *The daily cost of our diet is*

$$\vec{\mathbf{c}} \cdot \vec{\mathbf{x}} = \underbrace{c_1 \cdot x}_{\text{daily cost for bananas}} + \underbrace{a_{i,2} \cdot y}_{\text{cost for oranges}}$$

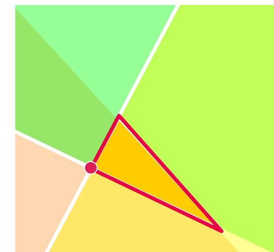
link

Minimize: minimize the cost of a healthy diet

Linear Programming – The Geometry

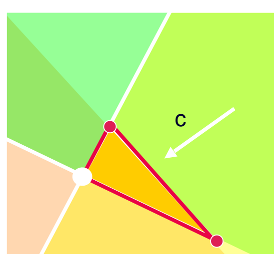
- Each constraint defines defines a half-space region in *d*-dimensional space.
- The *feasible region* is the (convex) intersection of these half-spaces.
- We will treat the case *d* = 2, where each constraint defines a *half-plane*.
- The equation y=ax+b defines a line, which we could also write as (-a)x+(1)y=b. Pointed one one side of this line forms a half-plane.

$$a_1x + a_2y \geq b$$
$$a_1x + a_2y \leq b$$



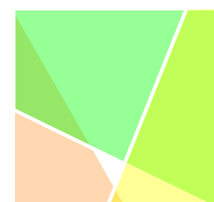
More Geometry

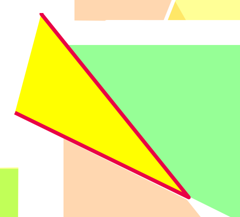
- The solution to the linear program is a point in the feasible region that is extreme in the direction of the target function.
- **Theorem:** Any bounded linear program that is feasible has **a** solution, which is a *vertex* of the feasible region.
- **Proof:** Convexity …

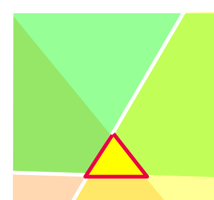


Degenerate Cases

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



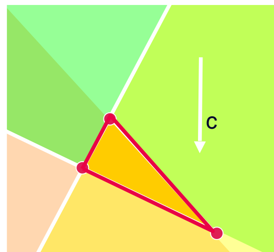




The Simplex Algorithm

- 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 R^d, the number of vertices might be $\Theta(n^{\lfloor d/2 \rfloor})$.



Linear Programming in d dimension - Example

- Define: (amount amount consumed per day)
  - j – types of foods ($1 \le i \le \mathbf{d}$). *($i = 1 \to$ banana, $i = 2 \to$ oranges, $i = 3 \to$ avocado…) This is the dimension of the LP problem.*
  - x_j - the amount of food j consumed daily $1 \le j \le d$)
    (these are the d unknowns that we need to optimize)
    $\vec{\mathbf{x}} = (x_1, x_2 \dots x_d)$
  - i – types of vitamins ($1 \le i \le n$).
  - a_{ij} – the amount of vitamin j in one unit of food i.
    $\vec{\mathbf{a}}_\mathbf{i} = (a_{i,1}, a_{i,2} \dots a_{i,d})$
  - b_j – minimal daily dose for vitamin i. $(1 \le i \le n)$
    - Constrains : $\forall i : a_{i,1}x_1 + a_{i,2}x_2 + \dots a_{i,d}x_d \ge b_i$ for every i
    - same using vector notation: $\vec{\mathbf{a}}_\mathbf{i} \cdot \vec{\mathbf{x}} \ge b_i$
  - c_i – the cost of a unit of food j $(1 \le j \le d)$
    $\vec{\mathbf{c}} = (c_1, \dots c_d)$

$\vec{a_i} \cdot \vec{x} \ge b_1$

- **LP problem**
  minimize the cost $\vec{\mathbf{c}}\vec{\mathbf{x}} = \sum c_j x_j$
  Such that (s.t.)
  for every $1 \le i \le n$
  $\vec{\mathbf{a}}_\mathbf{i} \cdot \vec{\mathbf{x}} \ge b_i$

$$Minimize: \; c^T x$$
$$Subject\,to: \; Ax \ge b$$

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 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.

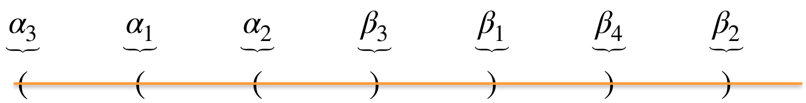
- In practice the simplex algorithm is used because of its linear *expected* runtime.

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, ∞)
  S' all other halfplanes contain the point (0, -∞)
  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.

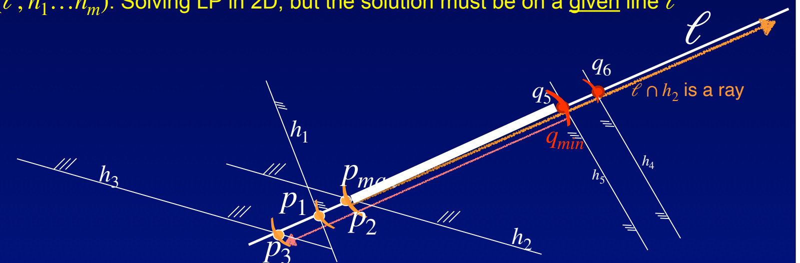
Toward a faster algorithm in small dimensions

- 1-dimensional linear programming
  - Given 2n constants (constrains) $\alpha_1, \alpha_2 \ldots \alpha_n, \quad \beta_1, \beta_2 \ldots \beta_n$ (not necessarily sorted)
  - find in O(n) time the minimum x such that
- $x \geq \alpha_i$ (for every $1 \leq i \leq n$) and $x \leq \beta_i$ (for every $1 \leq i \leq n$)
- What is the feasible region ? Could it be that the problem has no solution ?
- Answer



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

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



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

- Each half-plane either contains the point $(0, +\infty)$ or contains the point $(0, -\infty)$.
- Consider first only half-plane containing $(0, +\infty)$.
- Let ℓ_i be the line bounding h_i. Compute $p_i = \ell \bigcap \ell_i$
- Let p_{max} be the highest such point (p_2 in the example). Any solution to the LP must be on the portion of ℓ above p_{max}

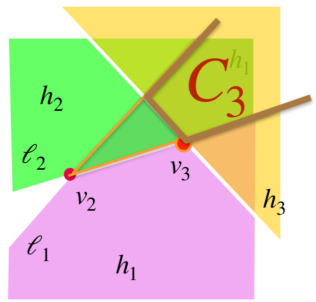
- Similarly, find the half-planes contain $(0, -\infty)$. Compute their intersections with ℓ. Let q_{min} be the lowest intersection points.
- Any solution to the LP which is on ℓ must be between p_{max} and q_{min}.
- Note that it is possible that qmin is below p_{max}. In this case, we have no solution on ℓ

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*.

Incremental Algorithm - Notation

- h_i is the i'th constrained half-plane
- ℓ_i is the line bounding h_i
- $C_i = h_1 \cap h_2 \cap \ldots h_i$ 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 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.
  2. - in the next slide

Basic Theorem - case 2.

Recall v_i is the lowest point at $C_i = h_1 \cap h_2 \cap \ldots \cap h_i$

Assume that v_i is not on ℓ_i

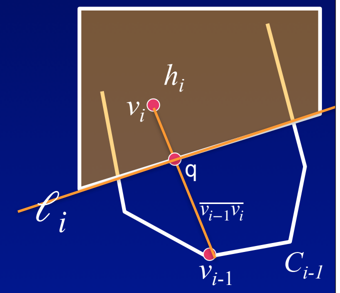
v_i must be in C_{i-1} By convexity, also the segment $\overline{v_{i-1}v_i}$ (from v_i to v_{i-1}) is in C_{i-1}.

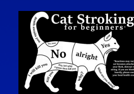
Assume WLOG: Our cost function pushes us downward.

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

Compute $C_i = h_1 \cap h_2$, 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, $h_1 \ldots h_{i-1}$)
3. If 1D-LP does not have a solution on ℓ_i - stop. There is no solution anywhere.
   set v_i to be the solution that 1D-LP found.

}

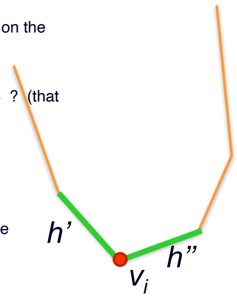
Complexity Analysis

- Worst case, each new constrain h_h 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 $v_i \in \ell_i$. 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 $\frac{\mathbf{2}}{\mathbf{i}}$. Here is the reason:
- v_i is determined by two half-planes. It does not care it which order the halfplanes 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 $1\frac{i-2}{i} \cdot 1 + c \cdot i\frac{2}{i} = 1 + 2c =$ constant.
- Therefor, 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** $\{h_1 \dots h_n\}$.

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 $\{h_4 \dots h_n\}$ 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 = 2DLP($h_1 \cap l_i, h_2 \cap l_i$, $h_3 \cap l_i, \dots, h_{i-1} \cap l_i$).

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 ***Θ(n³) (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.

Integer Linear Programming (ILP)
Example in Two Dimensions

- Define: (amount consumed per day)
  - types of foods : {oranges, bananas}
  - j – types of vitamins ($1 \le j \le n$).
  - x – number of pounds of oranges we recommend daily
  - y – number of pounds of bananas we recommend daily

    // these are the only unknown we have to compute.
  - a_{ji} – the amount of vitamin j in a unit of food i
    - ($i=1$ for oranges, $i=2$ for bananas)
  - c_1 – the number of calories in an orange.
  - c_2 – the number of calories in a banana.
  - b_j – minimal daily required amount of vitamin j.
- Constraints (we need to consume some minimal amount of each vitamin):

$$a_{11}x + a_{12}y \ge b_1$$
$$\vdots$$
$$a_{n1}x + a_{n2}y \ge b_n$$

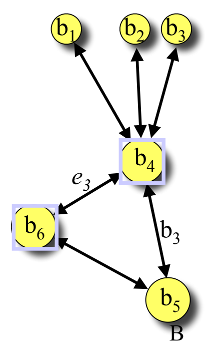
Another constrain: both *x,y* in the solution are integers.

Now we have ILP problem.

Minimize: the total number of calories consumed:

$$C((x,y)) = c_1 x + c_2 y$$

Vertex Cover and ILP



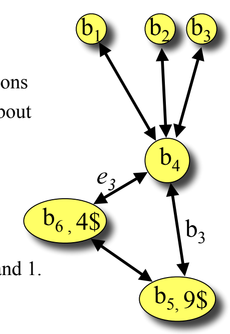
- Given: A graph G(V,E). A subset $C \subseteq V$ is a *vertex cover* if every edge$(u, v) \in E$ we have either $u \in C$ or $v \in C$ or both
- Finding the **min-cardinality** Vertex Cover is NP-Hard
- ILP for this problem: the variables are $x_1 \ldots x_n$. All are integers and between 0 and 1.
-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

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

s.t.

$$x_i + x_j \ge 1 \qquad \forall (v_i, v_j) \in E$$

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 $C \subseteq V$ is a *vertex cover* if every edge$(u, v) \in E$ we have either $u \in C$ or $v \in C$ 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 $x_1 \ldots x_n$. All are integers and between 0 and 1.
- $v_i \in C$ iff $x_i = 1$ (for $i = 1 \ldots n$)

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

s.t.

- $0 \le x_i \le 1$ and an integet, for every x_i
- $x_i + x_j \ge 1 \qquad \forall (v_i, v_j) \in E$

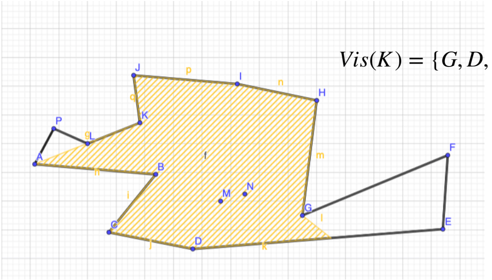
Art Gallery - on the board

- 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.

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

s.t.

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



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