CS 445

More LP and ILP. Applications to network flow, graph problems and sensor placements

Alon Efrat

Linear Programming (LP in dimension d with n constrains)

- Linear programming problems are minimization problems where we need to calculate the values of d unknown $(x_1, x_2, x_3 \ldots x_d)$. In addition
- The cost function is a linear combination of these variables. We are given constant $c_1 \ldots c_d$ and the goal is to minimize $\min c_1x_1 + c_2x_2 + \ldots c_dx_d$. It is very easy to use dot product notation - express $\vec{c} = (c_1, c_2 \ldots c_d)$ is a vector (given to us). We need to minimize $\vec{c} \cdot \vec{x} = c_1x_1 + c_2x_2 + \ldots c_dx_d$, where $\vec{x} = (x_1, x_2 \ldots x_d)$ is the vector of unknowns.
- We are also given a set of n vectors $\vec{a}_1, \vec{a}_2 \ldots \vec{a}_n$, and constants $b_1 \ldots b_n$. Each constrains limits the possible locations of $\vec{x}$.
- The constrains are

$$\vec{a}_1 \cdot \vec{x} \le b_1$$
$$\vec{a}_2 \cdot \vec{x} \le b_2$$
$$\vdots$$
$$\vec{a}_n \cdot \vec{x} \le b_n$$

or, if you are familiar with matrix notation, write it as $A \cdot x \le \vec{b}$. A is a matrix whose rows are $\vec{a}_1 \ldots \vec{a}_n$

- Geometrically, Fix some number i. The region of all the points $x \in \mathbb{R}^d$ in the d-dimensional space, satisfies $\vec{a}_i \cdot \vec{x} \le b_i$ is a half-space in $\mathbb{R}^d$. The boundary of this region are all the points $x \in \mathbb{R}^d$ for which $\vec{a}_i \cdot \vec{x} = b_i$.
- The dimension d effects the running time much more than the number of contrails n
- LP in high-dim is solved **simplex** algorithm (available in many libraries - CPLEX is popular)

Integer Linear Programming (ILP in dimension d with n constrains)

- Linear programming problems are minimization problems where we need to calculate the values of d unknown $(x_1, x_2, x_3 \ldots x_d)$. In addition
- The cost function is a linear combination of these variables. We are given constant $c_1 \ldots c_d$ and the goal is to minimize $\min c_1x_1 + c_2x_2 + \ldots c_dx_d$. It is very easy to use dot product notation - express $\vec{c} = (c_1, c_2 \ldots c_d)$ is a vector (given to us). We need to minimize $\vec{c} \cdot \vec{x} = c_1x_1 + c_2x_2 + \ldots c_dx_d$, where $\vec{x} = (x_1, x_2 \ldots x_d)$ is the vector of unknowns.
- We are also given a set of n vectors $\vec{a}_1, \vec{a}_2 \ldots \vec{a}_n$, and constants $b_1 \ldots b_n$. Each constrains limits the possible locations of $\vec{x}$.
- The constrains are

$$\vec{a}_1 \cdot \vec{x} \le b_1$$
$$\vec{a}_2 \cdot \vec{x} \le b_2$$
$$\vdots$$
$$\vec{a}_n \cdot \vec{x} \le b_n$$

or, if you are familiar with matrix notation, write it as $A \cdot x \le \vec{b}$. A is a matrix whose rows are $\vec{a}_1 \ldots \vec{a}_n$

- We can add the constrains that the numbers $x_1 \ldots x_d$ must be integers. Then the problem becomes an Integer Linear Programming (ILP) problems.
- which values of the computed variables must be integers are called Integer Linear Programming (ILP) 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.

In the next slide, we are going to talk about network flow problems. We will visit some properties of max flow

We are not going to describe Ford-Fulkeson algorithm.

The CLRS contains a chapter about Network-Flow. We use only the definitions

Flow networks

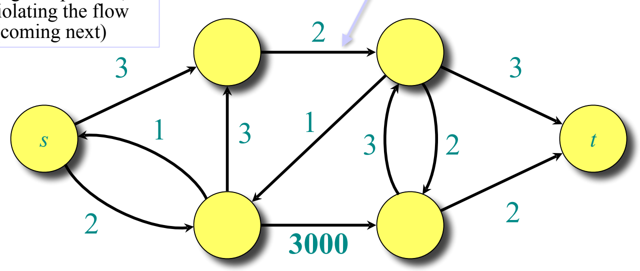
Definition. A ***flow network*** is a directed graph $G = (V, E)$ with two distinguished vertices: a ***source*** s and a ***sink*** t. Each edge $(u, v) \in E$ is given with a nonnegative ***capacity*** $c(u, v)$.

The values could specify the number of cars per minute on this road, or number of Gbyte on this link

Goal: Send as many cars/bytes/ gallons from s to t, without violating the edges capacities, and without violating the flow conservation (coming next)

Example:

The 2 here mean "only two gallons /minute on this pipe / only 2 cars/second on this road.



Flow in Networks

Def: A solution to the flow network flow problem (or in short, the flow) is on G is a set of values (numbers) $p(u,v)$ specific for every edge $(u,v) \in E$. So for the example below, we need to specify the numbers $\{p(s,d), p(s,b), p(d,c), p(g,b)\ldots\}$
These are the unknown that we need to compute.
p(u,v) is the flow on the edge (u,v).If $(u,v) \notin E$ then p(u,v) is defined by is 0.

To be a legal flow, these values must satisfy two sets of conditions:

- ***Capacity constraint:*** For all $u, v \in V$, $0 \le p(u,v) \le c(u,v)$

$$0 \le p(u, v) \le c(u, v).$$

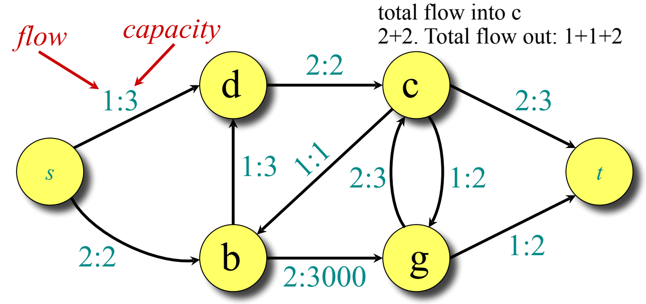
- ***Flow conservation:*** For all $u \in V$, which is not the source nor the sink $\sum_{v_i \in V} p(v_i, u) = \sum_{v_i \in V} p(u, v_i)$ //What comes in must go out.

- That is, every node is a memory-less router. It receives flow, and steer it to destinations.

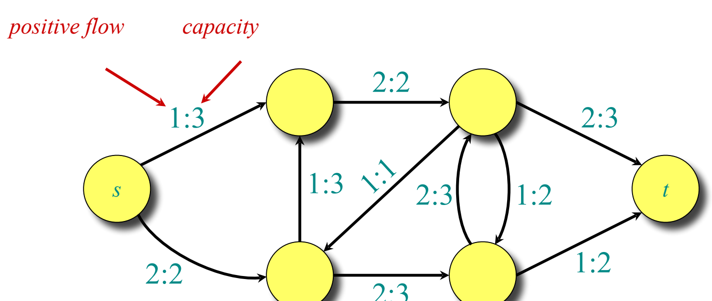
The ***total value*** of a flow is the sum of the flow flows out of the source:

$$\sum_{v_i \in V} p(s, v_i)$$

In the example, the value of the flow equals 1+2=3



Lemma

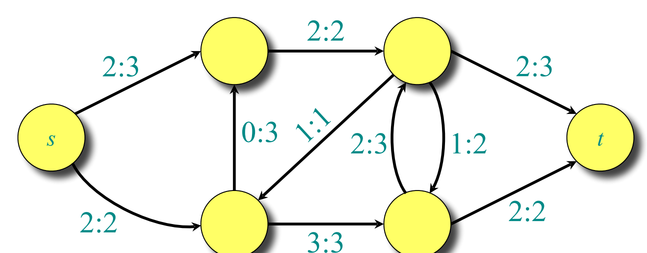


Lemma: The value of the flow equals to the sum of flows entering t

$$\sum_{v \in V} p(s, v) = \sum_{u \in V} p(v, t)$$

The maximum-flow problem

Maximum-flow problem: Given a flow network G, find a flow of maximum value on G.



The value of the maximum flow is 4.

LP could solve flow problems (but values might be non-integers)

Unknown variables: $p(u, v)$, for all $u, v \in V$

Constrains:

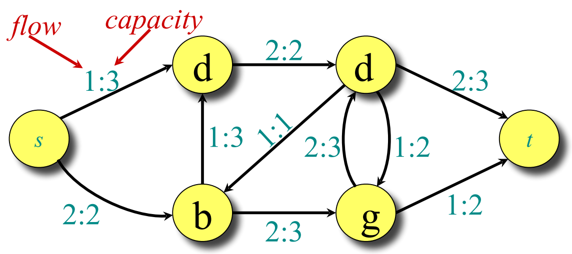
- ***Capacity constraint:*** For all $u, v \in V$,

$$0 \leq p(u, v) \leq c(u, v).$$

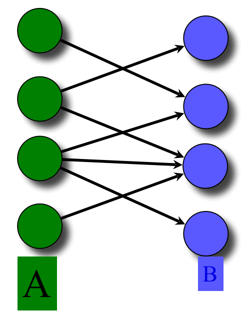
- ***Flow conservation:*** For all $u \in V - \{s, t\}$, $\sum_{v \in V} p(u,v) = \sum_{v \in V} p(v,u)$

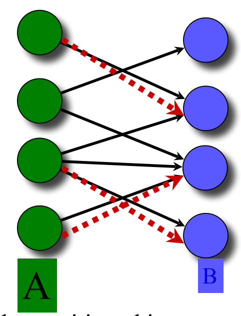
Maximize the ***value*** of the (the net flow out of the source)

$$\max \sum_{v \in V} p(s,v)$$



Application: Bipartite Matching.



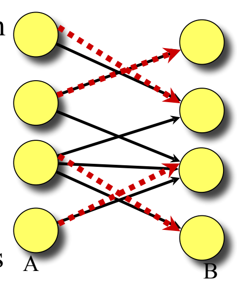


A graph *G(V,E)* is called **bipartite** if *V* can be partitioned into two sets *V=A∪B*, and each edge of *E* connects a vertex of *A* to a vertex of *B*. We sometimes denote these graphs by *G(A∪B,E)*
(we assume that the partition of V to *A* and *B* is given)

A **matching** is a set of edges *M* of *E*, where each vertex of *A* is adjacent to at most one vertex of *B, and vice versa.*

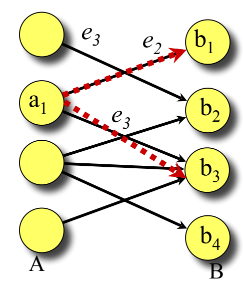
Application: Max-Cardinality Bipartite Matching.

- Max-Cardinality matching Given A bipartite graph *G(A∪B ,E),* find the largest subset M which is a matching.
- A **matching** is a set of edges *M* of *E*, where each vertex of *A* is adjacent to at most one vertex of *B, and vice versa.*
- This problem could be solved with in O(nm) time using Ford-Fulkerson algorithm. Faster algorithms exist as well. However, we will use it as an example to the ease of using ILP.
- This method fits well other variants of matching problems



ILP for Max-Cardinality Bipartite Matching.

- For every edge *e*, define a **Boolean** variable x_e.
- $x_e=1$ if *e* participates in *M, and* $x_e=0$ otherwise.
- The goal is to maximize the number of edges in *M,* while keeping *M* a proper matching.



maximize $\sum_{e \in E} x_e$

subject to

(1) $0 \leq x_x \leq 1 \quad \forall e \in E$

(2) x_i is an integer $\quad \forall e \in E$

(3) $\sum_{\{\forall e \in E \text{ s.t. } e \text{ is incident to } v\}} x_e \leq 1 \quad \forall v \in V$

In the example only one of the edges (a_1, b_1), (a_1, b_3) will be in *M*, since $x_2+x_3 \leq 1$

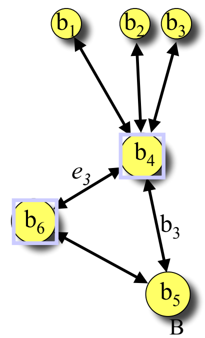
Vertex Cover and ILP

- 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.

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



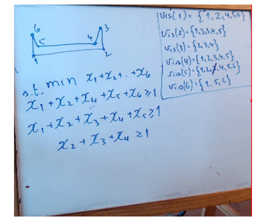
Art Gallery - on the board

- 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.
- For a vertex v_i we set x_i=1 if we place a guard at v_i.
- As usual , x_i are integers between 0 to 1.

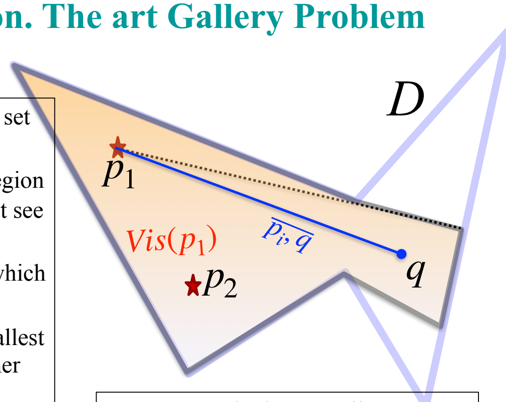
$$minimize \sum_{i=1}^{n} x_i$$

s.t.

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



Visibility in a polygon. The art Gallery Problem

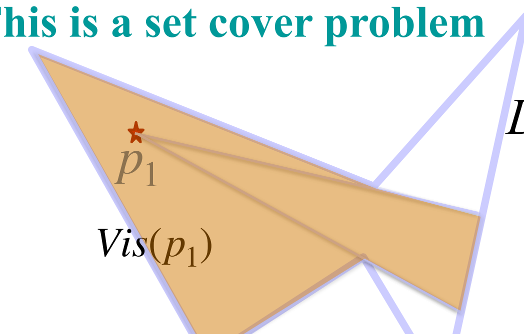


- Given - a polygon domain D, and a set $P = \{p_1 \ldots p_n\}$ of potential guards.
- Each potential guard p_i sees some region $Vis(p_i)$ of the polygon, but could not see through walls.
- Formally, p_i sees every point q for which the segment $\overline{p_i\, q}$ is fully in D.
- **Art Gallery Problem** - find the smallest set of guards (all from P) that together see the whole D.
- NP-hard (and extremely practical)
- $\mu_i = Area(Vis(p_i))$ the area (in meters^2) that it sees.
- Budget Art-Gallery Problem: Given a number k (`budget'), find a set G of $\leq k$ guards from P, that sees together the maximum area.

"Standard" Art Gallery:
Find the **smallest** set $\{g_1, g_2 \ldots g_r\} \subseteq P$
s.t
$D = Vis(g_1) \cup Vis(g_i) \cup .. Vis(g_r)$

Budget Art Galley:
Given k, find $\{g_1, g_2 \ldots g_k\} \subseteq P$
Maximize
$Area(\ Vis(g_1) \cup Vis(g_2) \cup .. Vis(g_k))$

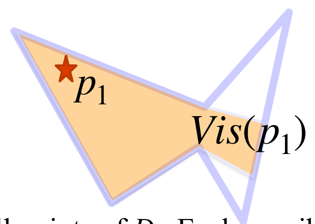
This is a set cover problem



- Given - a polygon domain D, and a set $P = \{p_1 \ldots p_n\}$ of potential guards.
- Every potential guard p_i defines a **set**. This set is $Vis(p_i)$. A set cover problem is to find a collection of sets that together covers the whole domain.
- Greedy Approach. The first guard is the point that sees maximum area $g_1 = \arg\max_{p \in P} \mu(p)$
- The second guard g_2 sees the maximum area that g_1 does not see
- g_3 sees the max area not seen by neither g_1 nor g_2, etc…

Set Cover Problems - terminology

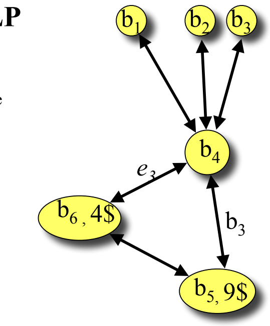
General problem: Given a **universe** $X = \{x_1 \ldots x_m\}$, each x_i is an **atoms**.
Also given a range space (also called set system). It is a collection of subsets of X. $\mathbf{R} = \{S_1, S_2 \ldots\}$ a collection of subsets of X. $(S_i \subseteq X)$



Examples:

1. In a polygon D, the atoms are all points of D. Each possible guard p_i defines $Vis(p_i)$. $\mathbf{R} = \{Vis(p_i) \mid p_i \in P\}$
2. Given a graph $G(V, E)$, we could treat V as the universe. Each edge is a set of two atoms. (edge-cover)
3. In a graph $G(V, E)$, the atoms are the **edges**. Each vertex $v_i \in V$ defines the set S_i of all the edges that v_i is adjacent to. (vertex cover)

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.

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