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 unknowns \((x_1, x_2, x_3, \ldots, x_d)\). 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 \(\min c_1 x_1 + c_2 x_2 + \ldots + c_d x_d\). 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 \(\vec{c} \cdot \vec{x} = c_1 x_1 + c_2 x_2 + \ldots + c_d x_d\) 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 or, if you are familiar with matrix notation, write it as
  \[
  \begin{align*}
  \vec{a}_1 \cdot \vec{x} &\leq b_1 \\
  \vec{a}_2 \cdot \vec{x} &\leq b_2 \\
  &\vdots \\
  \vec{a}_n \cdot \vec{x} &\leq b_n
  \end{align*}
  \]
- Geometrically, Fix some number \(i\). The region of all the points \(x \in \mathbb{R}^d\) in the d-dimensional space, satisfies \(\vec{a}_i \cdot \vec{x} \leq b_i\) 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 contrains \(n\)
- 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 \(\min c_1 x_1 + c_2 x_2 + \ldots + c_d x_d\). 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 \(\vec{c} \cdot \vec{x} = c_1 x_1 + c_2 x_2 + \ldots + c_d x_d\) 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 or, if you are familiar with matrix notation, write it as
  \[
  \begin{align*}
  \vec{a}_1 \cdot \vec{x} &\leq b_1 \\
  \vec{a}_2 \cdot \vec{x} &\leq b_2 \\
  &\vdots \\
  \vec{a}_n \cdot \vec{x} &\leq b_n
  \end{align*}
  \]
- Geometrically, Fix some number \(i\). The region of all the points \(x \in \mathbb{R}^d\) in the d-dimensional space, satisfies \(\vec{a}_i \cdot \vec{x} \leq b_i\) 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 contrains \(n\)
- LP in high-dim is solved **simplex** algorithm (available in many libraries - CPLEX is popular)

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-Fulkerson 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.

**Example:**

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

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

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

**Lemma**

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

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

**Flow in Networks**

**Definition**: A solution to the flow network flow problem (or in short, the flow) 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 $p(s, d)$, $p(s, b)$, $p(s, g)$, $p(d, c)$, $p(g, b)$, ... 

These are the unknowns that we need to compute:

- **capacity**: $c(u, v)$ is the flow on the edge $(u, v)$ if $(u, v) \in E$, then $p(u, v)$ is defined by it.
- **flow conservation**: For all $u \in V$, which is not the source nor the sink $\sum_{v \in V} p(v, u) = \sum_{v \in V} p(u, v)$ /What comes in must go out.

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

**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 \( V = A \cup B \), and each edge of \( E \) connects a vertex of \( A \) to a vertex of \( B \). We sometimes denote these graphs by \( G(A \cup B, E) \)
(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 \( G(A \cup B, E) \), 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.

\[
\text{maximize} \sum_{e \in E} x_e \\
\text{subject to}\]

\[
(1) \quad 0 \leq x_e \leq 1 \quad \forall e \in E \\
(2) \quad x_i \text{ is an integer} \quad \forall e \in E \\
(3) \quad \sum_{\{e \in E \text{ s.t. } e \text{ is incident to } v\}} x_e \leq 1 \quad \forall v \in V
\]

In the example only one of the edges \((a_1, b_1), (a_1, b_3)\) will be in \( M \), since \( x_2 + x_3 \leq 1 \)
### Art Gallery - on the board
- Given a polygon, find a subset of the vertices that sees every other vertex.
- Let \( \text{Vis}(i) \) 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.

\[
\text{minimize } \sum_{i=1}^{n} x_i \\
\text{s.t. } \sum_{k \in \text{Vis}(i)} x_k \geq 1 \quad \forall 1 \leq i \leq n
\]

### This is a set cover problem
- Given a polygon domain \( D \), and a set \( P = \{ p_1, ..., p_n \} \) of potential guards.
- Every potential guard \( p_i \) defines a set. This set is \( \text{Vis}(p_i) \). 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…

### Visibility in a polygon. The art Gallery Problem
- Given a polygon domain \( D \), and a set \( P = \{ p_1, ..., p_n \} \) of potential guards.
- Each potential guard \( p_i \) sees some region \( \text{Vis}(p_i) \) of the polygon, but could not see through walls.
- Formally, \( p_i \) sees every point \( q \) for which the segment \( p_i q \) 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)
- \( \mu_i = \text{Area}(\text{Vis}(p_i)) \) 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.

### 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 \( x_i \). All are integers and between 0 and 1.
- \( \forall i \in C \text{ iff } x_i = 1 \) (for \( i = 1, ..., n \))
- \( \sum_{i=1}^{n} x_i \leq 1 \quad \forall (v_p, v_p) \in E \)

\[
\text{minimize } \sum_{i=1}^{n} x_i \\
\text{s.t. } x_i + x_j \geq 1 \quad \forall (v_p, v_p) \in E
\]
Set Cover Problems - terminology

General problem: Given a universe \( X = \{x_1, \ldots, x_n\} \), each \( x_i \) is an atoms. Also given a range space (also called set system). It is a collection of subsets of \( X \). \( R = \{S_1, S_2, \ldots\} \) 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 \( \text{Vis}(p_i) \). \( R = \{ \text{Vis}(p_i) \mid p_i \in P \} \)

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 \( w_i \geq 0 \). 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.
  
  \[ \begin{align*}
  \text{minimize} & \quad \sum_{i=1}^{n} w_i x_i \\
  \text{s.t.} & \quad x_i + x_j \geq 1 \quad \forall (v_i, v_j) \in E
  \end{align*} \]