CS483 Analysis of Algorithms
Lecture 09 – Linear Programming 01 *

Jyh-Ming Lien

April 09, 2009

Introduction
Linear Programming

- Similar to dynamic programming, “programming” here means *optimization*

- Linear programming (LP) problems are optimization problems whose **objective** and **constraints** are all *linear* (i.e., exponents of all variables are 1)

- Many real-life problems can be expressed as LP problems
  - **Example: Profit maximization**
    - You are selling two kinds of chocolates: Pyramide and Pyramide Nuit
    - You make $1 profit by selling one box of Pyramide and $6 profit by selling one box of Pyramide Nuit
    - Your factory can only make 200 and 300 boxes of Pyramide and Nuit, resp., per day
    - Your worker can only produce 400 boxes per day.
    - You want to maximize your profit
  - How many boxes of Pyramide and Pyramide Nuit do you make to maximize your profit?
Example: Profit maximization

- Let \( x_1 \) and \( x_2 \) be the number of boxes we want to produce for Pyramide and Pyramide Nuit.

- **Objective Function:**

- **Constraints:**
  1. 
  2. 
  3. 
  4. 

- A LP problem can have **zero**, **one**, or **infinity** optimal solutions
  1. \( x > 5, x \leq 3 \)
  2. \( \max\{x_1 + x_2\}, x_1, x_2 > 0 \)
Geometric Interpretations of LP problems

- Each linear constraint can be represented as a **halfspace**

- A set of feasible solutions of a LP problem forms a **convex** set

- The objective function can be represented as a **hyperplane**

- When there is a unique solution, this solution must be a vertex of the convex set formed by the constraints

- **Example:** maximize $x_1 + 6x_2$

  \[
  \begin{align*}
  x_1 & \leq 200 \\
  x_2 & \leq 300 \\
  x_1 + x_2 & \leq 400 \\
  x_1 & \geq 0 \\
  x_2 & \geq 0
  \end{align*}
  \]
□ LPs can be solved by the *simplex method* (named one of the top ten best algorithms in 20th century)

□ Closely related to *hill-climbing* by jumping from one vertex to an adjacent vertex

□ Simplex is a type of “iterative improvement” method

□ We will cover simplex in the next lecture (for now we assume we have a simplex package that solves our problems).
Example: Production Planning

- We have a company making hand-made carpets and today is Jan/1st.
  - We now have 30 employees and each of them makes 20 carpets and get $2000 per month.
  - Each employee gets paid 80% more by working overtime but can only put in at most 30% overtime.
  - We can hire and fire employee. Hiring costs $320 and firing costs $400 per worker.
  - Storing surplus will cost $8 per carpet per month.
  - We do not have surplus now and we must end the year without surplus.
  - The demand for all months are $d_1, d_2, \ldots, d_{12}$

- How do we minimize our total cost?
Example: Production Planning

Introduction
Linear Programming
Example: Profit maximization
Geometric Interpretations of LP problems
Solving LP problems (Simplex)
Example: Production Planning
Example: Production Planning
Example: Bandwidth Allocation
Example: Bandwidth Allocation
LP variants and Standard form

Flows in networks

Simplex
Our company now is a network services provider

- The network has 3 nodes: A, B, C
- Connection A – B pays $3 per unit of bandwidth
- Connection B – C pays $2 per unit of bandwidth
- Connection A – C pays $4 per unit of bandwidth
- Each connection requires at least two units of bandwidth
- Each connection can be routed in two ways: long and short routes
- Bandwidths of the network are shown below

How do we route these connections to maximize our network’s revenue?
Example: Bandwidth Allocation

Introduction
Linear Programming
Example: Profit maximization
Geometric Interpretations of LP problems
Solving LP problems (Simplex)
Example: Production Planning
Example: Production Planning
Example: Bandwidth Allocation

LP variants and Standard form

Flows in networks

Simplex

Analysis of Algorithms
LP variants and Standard form

**Variants**

1. Objective functions: maximization and minimization
2. Constraints: equation or/and inequalities
3. Restrictions: variables are often restricted to be non-negative

**Standard form**

1. Objective functions: minimization
2. Constraints: equation
3. Restrictions: variables are all non-negative

**Reduction to standard form**

$maximize \ x_1 + 6x_2$

\[
\begin{align*}
    x_1 & \leq 200 \\
    x_2 & \leq 300 \\
    x_1 + x_2 & \leq 400 \\
    x_1 & \geq 0
\end{align*}
\]
Flows in networks

Introduction

- Flows in networks
- Maximum-flow problem
- LP and Maximum-flow problem
- Maximum-flow problem
- Residual graph
- Example
- Example
- Minimum Cut
- Maximum Bipartite Matching
- Maximum Bipartite Matching
- Stable Matching
- Stable Matching
- Stable Matching
- Simplex
Assuming that you are working for an oil company and the company owns a network of pipe lines along which oil can be sent, you are asked to find out the maximum capacity of oil can be sent from a city $s$ to another city $t$ over the network.

**Maximum-flow problem:** Given a weighted direct graph $G = \{V, E\}$, whose edge weight indicates the maximum capacity of an edge, find the maximum flow from a vertex $s$ (source) and to another vertex $t$ (sink) so that the following requirements are satisfied.

- The flow $f_e$ on edge $e$ must be $0 \leq f_e \leq c_e$
- Flow is conserved, i.e., $\sum_{(u,v) \in E} f_{uv} = \sum_{(v,w) \in E} f_{vw}$
<table>
<thead>
<tr>
<th>Topics</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td></td>
</tr>
<tr>
<td>Flows in networks</td>
<td></td>
</tr>
<tr>
<td>Maximum-flow</td>
<td></td>
</tr>
<tr>
<td>LP and Maximum-flow problem</td>
<td></td>
</tr>
<tr>
<td>Maximum-flow problem</td>
<td></td>
</tr>
<tr>
<td>Residual graph</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>Minimum Cut</td>
<td></td>
</tr>
<tr>
<td>Maximum Bipartite Matching</td>
<td></td>
</tr>
<tr>
<td>Maximum Bipartite Matching</td>
<td></td>
</tr>
<tr>
<td>Stable Matching</td>
<td></td>
</tr>
<tr>
<td>Stable Matching</td>
<td></td>
</tr>
<tr>
<td>Stable Matching</td>
<td></td>
</tr>
<tr>
<td>Stable Matching</td>
<td></td>
</tr>
<tr>
<td>Simplex</td>
<td></td>
</tr>
</tbody>
</table>

**Variables:**

**Objective:**

**Constraints:**
Iterative improvement

- Start with 0 capacity
- **Repeat**: Find a path from $s$ to $t$, and increase the flow along this path as much as possible

**Example:**

![Diagram of a network with nodes s, a, b, t and edges with capacities 1]
To make the algorithm work: We allow path to **cancel existing flow**

Residual graph $G^f$, whose edge weight indicate the remaining capacity of an edge. Two types of edge weights are available in $G^f$:

1. $c_{uv} - f_{uv}$, if $(u, v)$ is an edge of $G$ and $f_{uv} < c_{uv}$
2. $f_{vu}$, if $(u, v)$ is an edge of $G$ and $f_{uv} > 0$

Example:

```
□
```

```
□
```
Example

Example

Flow

Residual graph $G^f$

Diagram of a network flow graph with nodes s, b, c, d, e, and t, and edges with capacities as shown.
Flow

Residual graph $G^f$

- Time complexity:
Graph cut: \((s, t)\)-cut is the removal of a set of edges so that a connected component splits \(s\) and \(t\) into two connected components

The total capacity (edge weights) of a cut is an upper-bound of the capacity flow from one component to the other component

Theorem: **Maximum-flow Minimum cut**: The maximum flow of a graph from \(s\) to \(t\) equals to the capacity of the smallest \((s, t)\)-cut

Question: How to compute the minimum cut of a given graph?
Maximum Bipartite Matching

Given $n$ men and $n$ women, we add an edge between a man and a woman if they like each other. Can you find a perfect matching?

A graph is **bipartite** if you can split the vertices to two groups such that there is no edge connecting vertices in the same group.

| A bipartite graph | Not a bipartite graph |
Solving maximum bipartite matching problem:
Let’s make the problem more realistic: Given \( n \) men and \( n \) women, every man (woman) will rank all women (men).

We say a set of marriages (matching) is unstable if there are two pairs \((m, w)\) and \((m', w')\) with the following properties:

1. \( m \) prefers \( w' \) to \( w \)
2. \( w' \) prefers \( m \) to \( m' \)

Example 1 \((m, m', w, w')\):

1. \( m \) prefers \( w \) to \( w' \)
2. \( m' \) prefers \( w \) to \( w' \)
3. \( w \) prefers \( m \) to \( m' \)
4. \( w' \) prefers \( m \) to \( m' \)

Example 2 \((m, m', w, w')\):

1. \( m \) prefers \( w \) to \( w' \)
2. \( m' \) prefers \( w' \) to \( w \)
3. \( w \) prefers \( m' \) to \( m \)
4. \( w' \) prefers \( m' \) to \( m \)

Given \( n \) men and \( n \) women and a list of preferences, can you find a stable marriage for them?
Ideas:

- The idea is to have the pair \((m, w)\) enter a state called “engagement” before marriage
- A free (not engaged) man \(m\) can propose to a woman \(w\), there will be two possibilities:
  1. \(w\) rejects \(m\) (when \(w\) prefers her fiancee)
  2. \(w\) and \(m\) are engaged (when \(w\) is free or \(w\) prefers \(m\))
- A man can only propose to a woman once
Algorithm

**Algorithm 0.1: StableMatching(n)**

```plaintext
while there are free men
    pick a free man \( m \)
    Let \( w \) be the woman with the highest ranking, to whom
    \( m \) has not yet proposed
    if \( w \) is free
        then \( w \) and \( m \) are engaged
        do 
            if \( w \) prefers \( m' \)
                then \( m \) is still free
            else 
                else 
                    \( w \) and \( m \) are engaged
                    \( m' \) is now free
        Each engaged couple are now married
```

What is the time complexity?
Properties

- A woman remain engaged after she was proposed first time. Her fiancee gets better and better.
- A man can become free after engagement (his fiancee left him). His fiancee get worse and worse.
- This algorithm is biased to man: the matching is always a man-optimal matching

Is the algorithm correct?
Simplex
TSPortrait of Dantzig by Robert Bosch. George Dantzig (1914-2005) was the father of linear programming and the inventor of the Simplex Method.
Simplex Algorithm

- Simplex algorithm is an iterative improvement method
  - starting with a vertex $v$ of the convex set (of feasible solutions)
  - find another vertex $v'$ adjacent to $v$ with a higher objective value
  - $v = v'$, until no better adjacent vertex

- Example: maximize $x_1 + 6x_2$

\[
\begin{align*}
x_1 &\leq 200 \\
x_2 &\leq 300 \\
x_1 + x_2 &\leq 400 \\
x_1 &\geq 0 \\
x_2 &\geq 0
\end{align*}
\]

- Some more geometry
  - A vertex is formed by intersecting $n$ constraints (for a problem with $n$ variables)
  - Two adjacent vertices will share $n - 1$ constraints (and one different constraint)
Simplex Algorithm

- For a the simplex algorithm, we need to:
  - find an initial solution
  - update the current solution

- In some cases, our initial point is simple, i.e., \((0, 0, \cdots, 0)\), which gives us many advantages:
  1. This vertex is the intersection of \(x_i \geq 0\) constraints
  2. When all coefficients in the objective function are negative, our initial solution is optimal
  3. To pick an adjacent vertex, we simply pick a variable \(x_i\) whose coefficient in the objective function is positive and try to maximize \(x_i\)

- Example: maximize \(x_1 + 6x_2\)

\[
\begin{align*}
  x_1 & \leq 200 \\
  x_2 & \leq 300 \\
  x_1 + x_2 & \leq 400 \\
  x_1 & \geq 0 \\
  x_2 & \geq 0
\end{align*}
\]
Now, what do we do when our current solution is not at \((0, 0, \cdots, 0)\) anymore?

Well, we transform our problem so the current solution is at \((0, 0, \cdots, 0)\)

Transform coordinate system:

- Note that coordinates are defined as distances to the constraints
- After we move to an adjacent vertex, **one** constraint is changed
- Therefore, the coordinate defined by the new constraint needs to be updated
- The distance from a point to a hyper-plane \(a_i x = b_i\) is simply \(b_i - a_i x\)

Example: \textbf{maximize} \(x_1 + 6x_2\)

\[
\begin{align*}
    x_1 & \leq 200 \\
    x_2 & \leq 300 \\
    x_1 + x_2 & \leq 400 \\
    x_1 & \geq 0 \\
    x_2 & \geq 0
\end{align*}
\]
Let’s finish the example
What if \((0, 0, \cdots, 0)\) is not a feasible vertex? How do we start the process?

We can modify the original LP problem by adding \(m\) artificial variables \(z_i\), where \(m\) is the number of constraints. Now our new LP problem becomes:

- \(z_0 \geq 0, z_1 \geq 0, \ldots, z_{m-1} \geq 0\)
- Add \(z_i\) to the left size of the \(i\)-th constraint
- Minimize \(z_0 + z_1 + \cdots + z_{m-1}\)

First the initial vertex of the modified LP is easy to obtain:

\[
(x_1 = 0, x_2 = 0, \cdots, x_{n-1} = 0, z_0 = b_0, z_1 = b_1, \cdots, z_{m-1} = b_{m-1})
\]

Once we have the initial vertex, we can use the Simplex algorithm to solve the modified LP problem.

Now, if we have \(z_0 + z_1 + \cdots + z_{m-1} = 0\), we have an initial solution to solve the original LP problem.

If \(z_0 + z_1 + \cdots + z_{m-1} \neq 0\), the original LP will not have a feasible solution.