CS483 Design and Analysis of Algorithms

Chapter 7 Linear Programming and Reductions

Instructor: Fei Li

lifei@cs.gmu.edu with subject: CS483

Office hours: STII, Room 443, Friday 4:00pm - 6:00pm or by appointments

Course web-site:

http://www.cs.gmu.edu/∼lifei/teaching/cs483_fall08/

Figures unclaimed are from books "Algorithms" and "Introduction to Algorithms"

One of the Top 10 Algorithms in the 20th Century!

1. Formulate a problem using a linear program (Section 7.1)
2. Solve a linear program using the simplex algorithm (Section 7.6)
3. Applications: flows in networks; bipartite matching; zero-sum games (Sections 7.2 - 7.5)

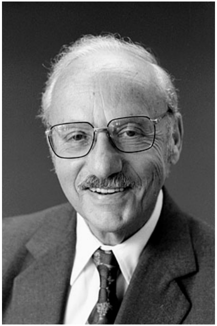


Figure: Father of Linear Programming and Simplex Algorithm: George Dantzig (1914 - 2005)

from `http://en.wikipedia.org/wiki/George_Dantzig`

Warm Up

Definition

Linear programming deals with satisfiability and optimization problems for linear constraints.

Definition

A linear constraint is a relation of the form

$$a_1 \cdot x_1 + \ldots + a_n \cdot x_n = b,$$

or

$$a_1 \cdot x_1 + \ldots + a_n \cdot x_n \leq b \quad \text{or} \quad a_1 \cdot x_1 + \ldots + a_n \cdot x_n \geq b,$$

where the a_i and b are constants and the x_i are the unknown variables.

Definition

Satisfiability: Given a set of linear constraints, is there a value $(x_1, \ldots, x_n)$ that satisfies them all?

Definition

Optimization: Given a set of linear constraints, assuming there is a value $(x_1, \ldots, x_n)$ that satisfies them all, find one which maximizes (or minimizes)

$$c_1 \cdot x_1 + \ldots + c_n \cdot x_n.$$

A Toy Example without Necessity of Calculation – from Eric Schost's Slides

Problem

You are allowed to share your time between two companies

1. *company C_1 pays 1 dollar per hour;*
2. *company C_2 pays 10 dollars per hour.*

Knowing that you can only work up to 8 hours per day, what schedule should you go for?

Of course, work full-time at company C_2.

1. **Linear formulation**:
   x_1 is the time spent at C_1 and x_2 the time spent at C_2.
2. **Constraints**:
   $$x_1 \geq 0, \; x_2 \geq 0, \; x_1 + x_2 \leq 8.$$
3. **Objective function**:
   $$\max \quad x_1 + 10 \cdot x_2.$$
4. **Solution**:
   $$x_1 = 0, \; x_2 = 8.$$

Another Example With Geometrical Solution

Problem

Two products are produced: A and B. Per day, we make x_1 of A with a profit of 1 each, we make x_2 of B with profit 6.
$x_1 \leq 200$ and $x_2 \leq 300$, and the total A and B is no more than 400. What is the best choice of x_1 and x_2 at maximizing the profit?

$$
\begin{array}{rrcl}
\textbf{Objective}: \text{ max} & x_1 & + & 6 \cdot x_2 \\
\textbf{Subject to}: & x_1 & \leq & 200 \\
 & x_2 & \leq & 300 \\
 & x_1 + x_2 & \leq & 400 \\
 & x_1, x_2 & \geq & 0
\end{array}
$$

Definition

The points that satisfy a single inequality are in a half-space.

Definition

The points that satisfy several inequalities are in the intersection of half-spaces. The intersection of (finitely many) half-spaces is a convex polygon (2D) — the feasible region.

Exercise

(7.1 of DPV) Consider the following linear program. Plot the feasible region and identify the optimal solution.

$$
\begin{array}{rcl}
\max & & 5x + 3y \\
5x - 2y & \geq & 0 \\
x + y & \leq & 7 \\
x & \leq & 5 \\
x & \geq & 0 \\
y & \geq & 0
\end{array}
$$

Why Bother Thinking an Algorithm?

Problem

Suppose we are managing a network containing A, B, and C. Each connection requires at least two units of bandwidth, but can be assigned more. Connection A − B pays $3 per unit of bandwidth, and connection B − C and A − C pay $2 and $4, respectively. Each connection can be routed in two ways, a long path and a short path, or by a combination. How do we route these connections to maximize our network's revenue?

$$
\begin{aligned}
\max \quad & 3x_{AB} + 3x'_{AB} + 2x_{BC} + 2x'_{BC} + 4x_{AC} + 4x'_{AC} & \\
& x_{AB} + x'_{AB} + x_{BC} + x'_{BC} \le 10 & [\text{edge } (b, B)] \\
& x_{AB} + x'_{AB} + x_{AC} + x'_{AC} \le 12 & [\text{edge } (a, A)] \\
& x_{BC} + x'_{BC} + x_{AC} + x'_{AC} \le 8 & [\text{edge } (c, C)] \\
& x_{AB} + x'_{BC} + x'_{AC} \le 6 & [\text{edge } (a, b)] \\
& x'_{AB} + x_{BC} + x'_{AC} \le 13 & [\text{edge } (b, c)] \\
& x'_{AB} + x'_{BC} + x_{AC} \le 11 & [\text{edge } (a, c)] \\
& x_{AB} + x'_{AB} \ge 2 & \\
& x_{BC} + x'_{BC} \ge 2 & \\
& x_{AC} + x'_{AC} \ge 2 & \\
& x_{AB}, x'_{AB}, x_{BC}, x'_{BC}, x_{AC}, x'_{AC} \ge 0 &
\end{aligned}
$$

We need ask computer to do this ⇒ We need to design an algorithm to solve a linear program!

Any Algorithmic Observation?

Definition

An extreme point p is impossible to be expressed as a convex combination of two other distinct points in the convex polygon.

Theorem

The optimal solution, if it exists, is at some extreme point p.

1. A naive algorithm (expensive!):
    1. List all the possible vertices.
    2. Find the optimal vertex (the one with the maximal value of the objective function).
    3. Try to figure out whether it is a global maximum.
2. Our approach (the simplex algorithm):
    1. Start at some extreme point.
    2. Pivot from one extreme point to a neighboring one.
    3. Repeat until optimal.

Standard Form of LP

1. A maximization $\Leftrightarrow$ a minimization problem:
   $\Leftrightarrow$ Multiply the coefficients of the objective function by -1.
2. Equations $\Leftrightarrow$ inequalities:
   $\Rightarrow$ To turn an inequality constraint like $\sum_{i=1}^{n} a_i \cdot x_i \leq b$, introduce a new *slack variable s* and use

$$\begin{aligned} \sum_{i=1}^{n} a_i \cdot x_i + s &= b \\ s &\geq 0 \end{aligned}$$

   $\Leftarrow$ Rewrite $a \cdot x = b$ as the equivalent pair of constraints $a \cdot x \leq b$ and $a \cdot x \geq b$.
3. The variables (say x) can also be unrestricted in sign:
   1. Introduce two non-negative variables $x^+, x^- \geq 0$.
   2. Replace x, wherever it occurs in the constraints or the objective function, by $x^+ - x^-$.

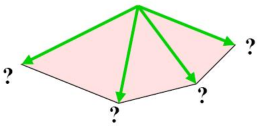
A generic LP in Matrix-vector notation:

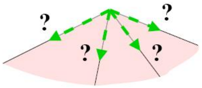
$$\begin{aligned} \max \quad & \vec{c}^{\,T} \vec{x} \\ \mathbf{A}\vec{x} &\leq \vec{b} \\ \vec{x} &\geq 0 \end{aligned}$$

The Simplex Algorithm in Solving a LP

The Simplex Algorithm — Sketch

1. Start at some extreme point v_1.
2. Pivot from one extreme point v_1 to a neighboring one v_2.



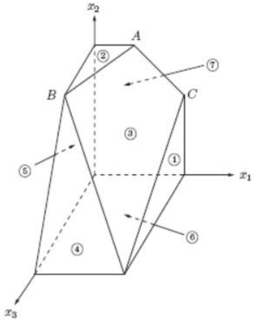


   1. v_2 should increase the value of the objective function.
   2. Several strategies are available to select v_1.
3. Repeat until optimal — reach a vertex where no improvement is possible.

Correctness?

Complexity analysis?

The Simplex Algorithm — Visualization and Intuition



$$
\begin{aligned}
\max \quad & x_1 + 6x_2 + 13x_3 & \\
& x_1 \le 200 & ① \\
& x_2 \le 300 & ② \\
& x_1 + x_2 + x_3 \le 400 & ③ \\
& x_2 + 3x_3 \le 600 & ④ \\
& x_1 \ge 0 & ⑤ \\
& x_2 \ge 0 & ⑥ \\
& x_3 \ge 0 & ⑦
\end{aligned}
$$

Definition

Each *vertex* is the unique point at which some subset of hyper-planes meet $\Rightarrow$ (a) Pick a subset of the inequalities. (b) If there is a unique point that satisfies them with equality, and this point happens to be feasible, then it is a *vertex*. (c) Each vertex is specified by a set of n inequalities.

Definition

Two vertices are *neighbors* if they have $n-1$ defining inequalities in common.

The Simplex Algorithm

Consider a generic LP

$$
\begin{array}{rcl}
\max & & \vec{c}^{\,T}\vec{x} \\
\mathbf{A}\vec{x} & \leq & \vec{b} \\
\vec{x} & \geq & 0
\end{array}
$$

One each iteration, simplex has two tasks:

1. Check whether the current vertex is optimal (and if so, halt).
2. Determine where to move next.

    1. Move from the origin by increasing some x_i for which $c_i > 0$. *Until we hit some other constraint*.

    That is, we release the tight constraint $x_i \geq 0$ and increase x_i until some other inequality, previously loose, now become tight. At that point, we are at a new vertex.

Remark

Both tasks are easy if the vertex happens to be at the origin. That is, if the vertex is elsewhere, we will transform the coordinate system to move it to the origin.

Theorem

The objective is optimal when the coordinates of the local cost vector are all zero or negatives.

Simplex in Action

Initial LP:

$$\max\ 2x_1 + 5x_2$$

$$\begin{aligned} 2x_1 - x_2 &\le 4 && \text{①} \\ x_1 + 2x_2 &\le 9 && \text{②} \\ -x_1 + x_2 &\le 3 && \text{③} \\ x_1 &\ge 0 && \text{④} \\ x_2 &\ge 0 && \text{⑤} \end{aligned}$$

Current vertex: {④, ⑤} (origin).
Objective value: 0.

Move: increase x_2.
⑤ is released, ③ becomes tight. Stop at $x_2 = 3$.

New vertex {④, ③} has local coordinates (y_1, y_2):

$$y_1 = x_1, \quad y_2 = 3 + x_1 - x_2$$

Rewritten LP:

$$\max\ 15 + 7y_1 - 5y_2$$

$$\begin{aligned} y_1 + y_2 &\le 7 && \text{①} \\ 3y_1 - 2y_2 &\le 3 && \text{②} \\ y_2 &\ge 0 && \text{③} \\ y_1 &\ge 0 && \text{④} \\ -y_1 + y_2 &\le 3 && \text{⑤} \end{aligned}$$

Current vertex: {④, ③}.
Objective value: 15.

Move: increase y_1.
④ is released, ② becomes tight. Stop at $y_1 = 1$.

New vertex {②, ③} has local coordinates (z_1, z_2):

$$z_1 = 3 - 3y_1 + 2y_2, \quad z_2 = y_2$$

Rewritten LP:

$$\begin{array}{rcll} \max\ 15 + 7y_1 - 5y_2 & & & \\ y_1 + y_2 & \leq & 7 & ① \\ 3y_1 - 2y_2 & \leq & 3 & ② \\ y_2 & \geq & 0 & ③ \\ y_1 & \geq & 0 & ④ \\ -y_1 + y_2 & \leq & 3 & ⑤ \end{array}$$

Current vertex: {④, ③}.
Objective value: 15.

Move: increase y_1.
④ is released, ② becomes tight. Stop at $y_1 = 1$.

New vertex {②, ③} has local coordinates (z_1, z_2):

$$z_1 = 3 - 3y_1 + 2y_2, \quad z_2 = y_2$$

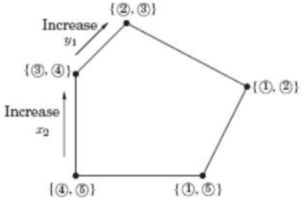
Rewritten LP:

$$\begin{array}{rcll} \max\ 22 - \frac{7}{3}z_1 - \frac{1}{3}z_2 & & & \\ -\frac{1}{3}z_1 + \frac{5}{3}z_2 & \leq & 6 & ① \\ z_1 & \geq & 0 & ② \\ z_2 & \geq & 0 & ③ \\ \frac{1}{3}z_1 - \frac{2}{3}z_2 & \leq & 1 & ④ \\ \frac{1}{3}z_1 + \frac{1}{3}z_2 & \leq & 4 & ⑤ \end{array}$$

Current vertex: {②, ③}.
Objective value: 22.

Optimal: all $c_i < 0$.

Solve ②, ③ (in original LP) to get optimal solution $(x_1, x_2) = (1, 4)$.

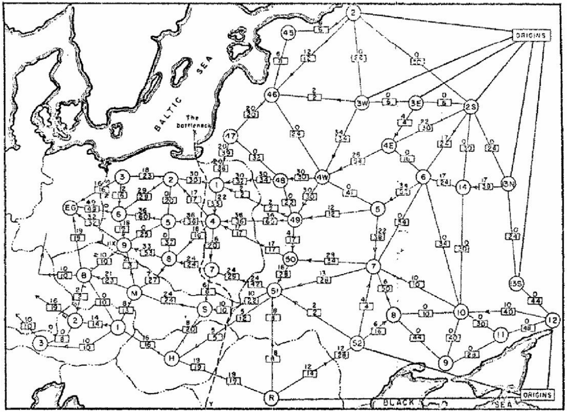


Complexity of the Simplex

1. **Worst case**.
   One can construct examples where the simplex algorithm visits all vertices (which can be exponential in the dimension and the number of constraints).
2. **Most cases**.
   The simplex algorithm works very well.

Flows in Networks

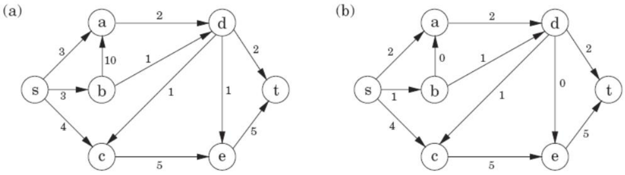
Soviet Rail Network, 1955



Reference: *On the history of the transportation and maximum flow problems.*
Alexander Schrijver in Math Programming, 91: 3, 2002.

Flows in Networks

Figure 7.4 (a) A network with edge capacities. (b) A *flow* in the network.



Definition

Consider a directed graph $G = (V, E)$; two specific nodes $s, t \in V$. s is the *source* and t is the *sink*. The *capacity* $c_e > 0$ of an edge e.

Definition

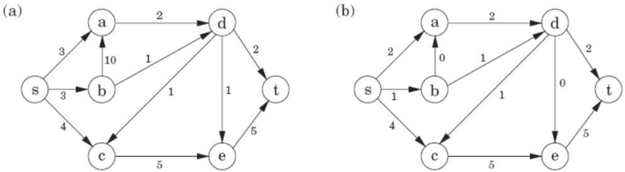
Flow. A particular shipping scheme consisting a variable f_e for each edge e of the network, satisfying the following two properties:

1. $0 \leq f_e \leq c_e$, $\forall e \in E$.
2. For all nodes $u \neq s, t$, the amount of flow entering u equals the amount leaving u (i.e., flows are conservative):

$$\sum_{(w,u) \in E} f_{wu} = \sum_{(u,z) \in E} f_{uz}.$$

Flows in Networks

Figure 7.4 (a) A network with edge capacities. (b) A *flow* in the network.



Definition

Size of a flow. The total quantity sent from s to t, i.e., the quantity leaving s:

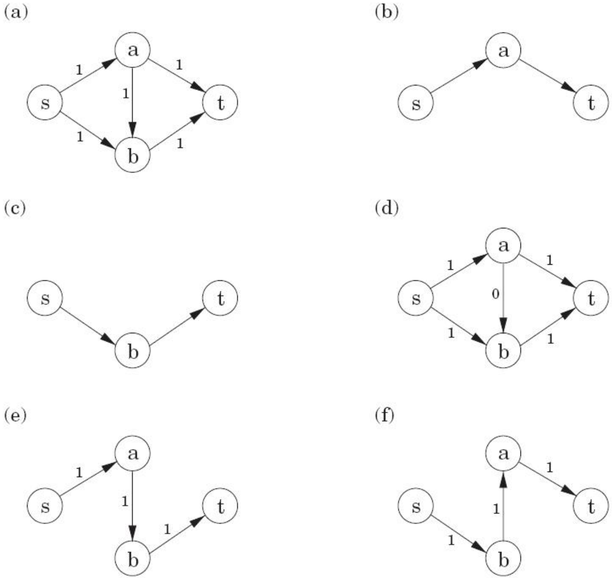
$$size(f) := \sum_{(s,u)\in E} f_{su}.$$

$$\begin{aligned} \max size(f) \quad &:= \quad \sum_{(s,u)\in E} f_{su} \\ \text{subject to} & \\ 0 \quad \leq \quad f_e \quad &\leq \quad c_e, \quad \forall e \in E \\ \sum_{(w,u)\in E} f_{wu} \quad &= \quad \sum_{(u,z)\in E} f_{uz}, \quad u \neq s, t \end{aligned}$$

Using the Interpretation of the Simplex Algorithm

1. Start with a zero flow.
2. Repeat: Choose an appropriate path from s to t, and increase flow along the edges of this path as much as possible.

Figure 7.5 An illustration of the max-flow algorithm. (a) A toy network. (b) The first path chosen. (c) The second path chosen. (d) The final flow. (e) We could have chosen this path first. (f) In which case, we would have to allow this second path.



Using the Interpretation of the Simplex Algorithm

1. Start with a zero flow.
2. Repeat: Choose an appropriate path from s to t, and increase flow along the edges of this path as much as possible. In each iteration, the simplex looks for an $s - t$ path whose edge (u, v) can be of two types:
   1. (u, v) is in the original network, and is not yet at full capacity. If f is the current flow, edge (u, v) can handle up to $c_{uv} - f_{uv}$ additional units of flow.
   2. The reverse edge (v, u) is in the original network, and there is some flow along it. Up to f_{vu} additional units (i.e., canceling all or part of the existing flow on (v, u)).

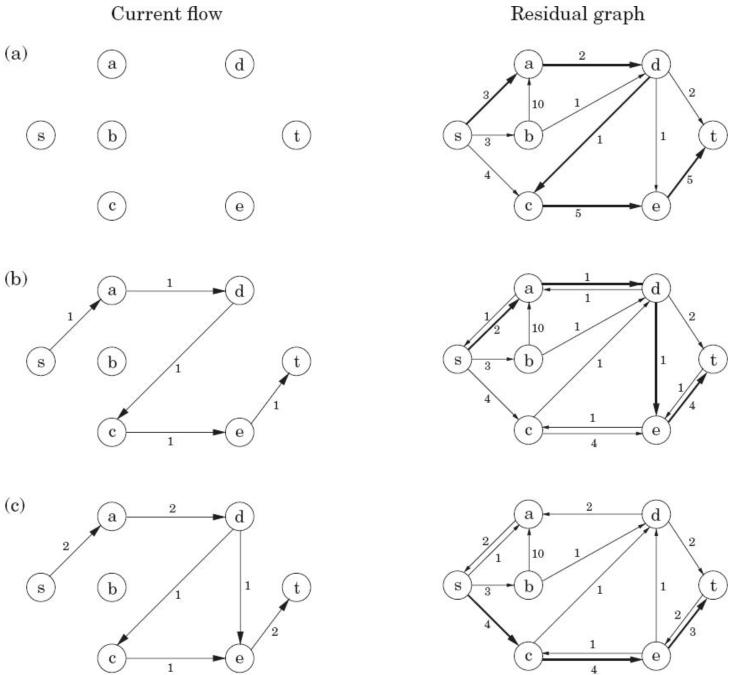
Definition

Residual network $G^f = (V, E^f)$. G^f has exactly the two types of edges listed, with residual capacity c^f:

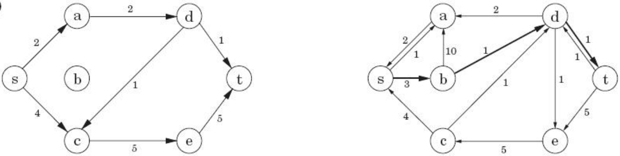
$$c^f := \begin{cases} c_{uv} - f_{uv}, & \text{if } (u, v) \in E \text{ and } f_{uv} < c_{uv} \\ f_{vu}, & \text{if } (v, u) \in E \text{ and } f_{vu} > 0 \end{cases} \tag{1}$$

Definition

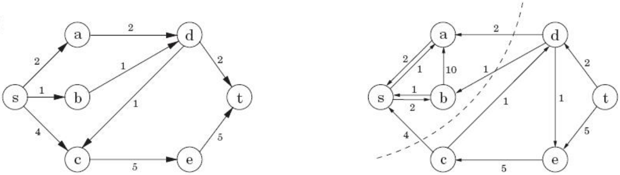
Augmenting path. An augmenting path p is a simple path from s to t in the residual network G^f.

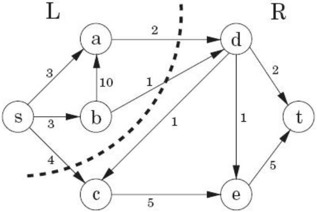

Current flow
Residual graph
(a)
(b)
(c)

(e)



(f)





Flows in Networks

Definition

Cuts. A $s - t$ cut partitions the vertices into two disjoint groups L and R such that $s \in L$ and $t \in R$. Its *capacity* is the total capacity of the edges from L to R, and it is an upper bound on *any* flow from s to t.

Theorem

Max-flow min-cut theorem. *The size of the maximum flow in a network equals the capacity of the smallest (s, t)-cut.*

Proof.

? □

Theorem

The running time of the augmentation-flow algorithm is $O(|V| \cdot |E|^2)$ over an integer-value graph.

Proof.

? □