Optimization Theory (DS2) Lecture #2
Examples of Linear, Integer and Graph Problems

April 26, 2017

Abstract

Today we are primarily going to look at how to *formulate* problems, starting from word descriptions of the problem, into a rigorous mathematical form. How we *solve* those problems will come during the course of the semester.

Text and examples adapted from A Gentle Introduction to Optimization.

1 Introduction

Today we are going to run through a set of examples. Ideally, you should follow up on this by reading the corresponding sections in the book and trying some of the exercises.

Examples:

1. Linear Program: WaterTech (Sec. 1.1 from the textbook)
2. Integer Program: KitchTech, the knapsack problem (Sec. 1.3.2)
3. Graphs: shortest path problem (Sec. 1.4.1, 1.5.2)
4. Graphs: minimum cost perfect matching (Sec. 1.4.2, 1.5.1)

2 Linear Program: WaterTech

WaterTech makes four types of products, each of which requires some amount of time on Machine 1 (say, a milling machine) and a certain amount of time on Machine 2 (say, a polishing machine). It takes both skilled and unskilled labor to make the products, and of course each sells for a certain price. How can we maximize profit?

Table of times and selling price:

Product	Machine 1	Machine 2	Skilled labor	Unskilled labor	Unit sale price
1	11	4	8	7	$300
2	7	6	5	8	$260
3	6	5	5	7	$220
4	5	4	6	4	$180

We have the following constraints:

1. 700 hours of time are available on Machine 1.
2. 500 hours of time are available on Machine 2.
3. Up to 600 hours of skilled labor at $8/hour can be purchased.
4. Up to 650 hours of unskilled labor at $6/hour can be purchased.

Variables. Our product "names" are the numbers 1 to 4, so let's call x_i the number of each product we decide to make, for $i \in \{1, 2, 3, 4\}$. (Later we will see that sometimes you have to add artificial variables, but in this case we'll stick with the basics.)

Constraints. The total amount of time used on Machine 1 has to be less than 700 hours, and can be written

$$11x_1 + 7x_2 + 6x_3 + 5x_4 \leq 700. \tag{1}$$

Likewise, the constraint on Machine 2 is

$$4x_1 + 6x_2 + 5x_3 + 4x_4 \leq 500. \tag{2}$$

You can see these equations by going down the columns labeled Machine 1 and Machine 2 in the table above.

We don't yet know how much labor we're buying, so let's call unskilled labor y_u and skilled labor y_s. Going down the corresponding columns, we get the constraints

$$8x_1 + 5x_2 + 5x_3 + 6x_4 \leq y_s \tag{3}$$
$$7x_1 + 8x_2 + 7x_3 + 4x_4 \leq y_u \tag{4}$$

Objective function. This is the thing we're optimizing for: profit! Let's call our income from sales I, costs C, and profit P. Obviously,

$$P = I - C \tag{5}$$

and we want that to be positive and as big as possible! We can read the income off the right hand column of the table,

$$I = 300x_1 + 260x_2 + 220x_3 + 180x_4 \tag{6}$$

and the labor costs are

$$C = 8y_s + 6y_u. \tag{7}$$

2.1 Formulation

Finally, this gives us the formal formulation of the formulas for this problem:

Maximize

$$300x_1 + 260x_2 + 220x_3 + 180x_4 - 8y_s - 6y_u. \tag{8}$$

subject to the constraints

$$11x_1 + 7x_2 + 6x_3 + 5x_4 \leq 700 \tag{9}$$
$$4x_1 + 6x_2 + 5x_3 + 4x_4 \leq 500 \tag{10}$$
$$8x_1 + 5x_2 + 5x_3 + 6x_4 \leq y_s \tag{11}$$
$$7x_1 + 8x_2 + 7x_3 + 4x_4 \leq y_u \tag{12}$$
$$y_s \leq 600 \tag{13}$$
$$y_u \leq 650 \tag{14}$$
$$x_1, x_2, x_3, x_4, y_s, y_u \geq 0. \tag{15}$$

The first four of those constraints are the equations we derived above, but don't forget the bottom three – without them our techniques can easily yield an unphysical solution such as buying a *negative* amount of labor or building an infinite number of some product!

3 Knapsack Problem

(Our first real integer programming problem! Might not get there today, but the problem is from 1.3.2.)

4 Shortest Path Problem

A *graph* is a set of vertices V and a set of edges E, where each edge connects to two of the vertices, or nodes. For some problems, we may begin bg guaranteeing that the whole thing is connected, for others, we may not.

You will see this problem in several forms in different fields; the way networking people approach it is somewhat different from the way we will see it here. Today, we will formulate it as an integer problem. (Later in the semester, if we have time, we will see another means of solving this problem, called *shortest path first*, by a dude named Edsger Dijkstra.)

See Fig. 1.9 in the text. We have four vertices, s (our source), t (our target), a and b. Let's first see the *adjacency matrix*, which contains 1 if two vertices are connected by an edge, and 0 if not (and along the diagonal):

$$\begin{array}{c} \\ s \\ a \\ b \\ t \end{array} \begin{array}{c} \begin{array}{cccc} s & a & b & t \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \tag{16}$$

Obviously, this is an $n \times n$ matrix for a graph with n nodes. If the edges are *weighted*, e.g. representing distances between two points, we have the *weighted adjacency matrix*

(or sometimes the *distance matrix*):

$$\begin{array}{c} \\ s \\ a \\ b \\ t \end{array} \begin{array}{c} \begin{array}{cccc} s & a & b & t \end{array} \\ \begin{pmatrix} 0 & 3 & 4 & \infty \\ 3 & 0 & 1 & 2 \\ 4 & 1 & 0 & 2 \\ \infty & 2 & 2 & 0 \end{pmatrix} \end{array} \tag{17}$$

The numbers are known as the *cost* or *weight*. (Often edges in a graph have a direction, known as a *directed graph* or *digraph*, but here we are dealing only with undirected edges.)

4.1 Paths

A *path* is a list of edges that connects s to t. Their order matters, and the tail of each edge has to be the head of the next edge. No node can be repeated. So for this simple graph, we have four possible paths:

$$sa, at \tag{18}$$

$$sa, ab, bt \tag{19}$$

$$sb, bt \tag{20}$$

$$sb, ba, at. \tag{21}$$

Each edge has a cost, as above. If we establish the order of our edges to be sa, sb, ab, at, bt, then we can write the vector $\vec{x} = (1, 0, 0, 1, 0)^\intercal$ to represent the path sa, at. (Question: What is the length of this vector?)

If we write the edge costs as a vector, also, then we can calculate the *path cost* as

$$C = (3, 4, 1, 2, 2)\vec{x} \tag{22}$$

which is the same as

$$C = \sum_{e \in P} c_e = \sum_{e \in E} c_e x_e \tag{23}$$

where c_e is the cost for the edge e, P is the path, and E is the (complete) set of edges.

4.2 Cuts (or Partitions)

How many different ways can we split our graph into two parts, with s in one part and t in the other? If there are n total nodes in our graph, there are 2^{n-2} ways (because we start with the assumption that s is in one half and t in the other). In our example, we only have two other nodes, a and b, so we have $2^2 = 4$ possible sets. We'll refer to $\delta(U)$ as the *cut* or *st-cut* of the set U, and the four possibilities are s plus: $\{\}, \{a\}, \{b\}, \{a, b\}$. Then our possible cuts are:

$$\delta(\{s\}) = \{sa, sb\} \tag{24}$$

$$\delta(\{s, a\}) = \{at, ab, sb\} \tag{25}$$

$$\delta(\{s, b\}) = \{sa, ab, bt\} \tag{26}$$

$$\delta(\{s, a, b\}) = \{at, bt\}. \tag{27}$$

It turns out that a path that reaches from s to t has at least one edge in *every one* of those cuts above.

4.3 Formulation

Variables. As above, x_e is 0/1 for each edge e.

Constraints. Our strategy for the integer problem formulation of this is to have a constraint for each one of those sets U above, saying that we have at least one edge from each of them. (Question: how many constraints is this? Recall that we said last time that the number of constraints must be finite, but we didn't say it has to be small!)

Objective function. Our cost C, above.

We can write this down more mathematically as:

Minimize $(3, 4, 1, 2, 2)\vec{x}$ subject to

$$\begin{array}{c} \\ \{s\} \\ \{s,a\} \\ \{s,b\} \\ \{s,a,b\} \end{array} \begin{array}{c} \begin{array}{ccccc} sa & sb & ab & at & bt \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array} \vec{x} \geq \vec{1} \tag{28}$$

and

$$x_e \geq 0, x_e \in \mathbb{Z} \tag{29}$$

where I've used $\vec{1}$ to mean the vector of all ones of the appropriate length, and $\mathbb{Z}$ is the set of all integers, that is, every x_e has to be an integer. This big matrix constraint means that every element of the vector resulting from multiplying this matrix times our solution vector $\vec{x}$ has be at least one.

Note that many non-paths are technically solutions to this problem, but it can be proven that all *optimal* solutions will be st-paths.

5 Minimum Cost Perfect Matching

(Might not get there today, but the problem is from 1.4.2 and 1.5.1.)