Intro to LO, Lecture 2

Martin Böhm
University of Wroclaw, Winter 2023/2024

D(Linear program**):** A *linear program* is a model of an optimization problem over real-valued variables using linear constraints and a linear objective function.

Compact form:
$$\begin{aligned} &\min \boldsymbol{c}^T\boldsymbol{x} \\ &\text{s.t. } A\boldsymbol{x} \geq \boldsymbol{b} \\ &\boldsymbol{x} \geq 0 \end{aligned}$$

T: An optimal vector $\boldsymbol{x}^*$ to a linear program can be computed in polynomial time with respect to the program size.

A bakery example

A bakery produces four things: bread, bagels, baguettes and donuts. To bake a single bread, they need 500g of flour, 10 eggs and 50 grams of salt. To bake a bagel, they need 150 grams of flour, 2 eggs and 10g of salt. For a baguette, they need 230g of flour, 7 eggs and 15g of salt. For a donut, they need 100g of flour and 1 egg.

The bakery has a daily supply of 5 kg of flour, 125 eggs, and 500g of salt.

The bakery charges 5 PLN for one bread, 1 PLN for a bagel, 2.5 PLN for a baguette and 1.5 PLN for a donut. The bakery tries to maximize its profit.

$$\begin{aligned} &\max 5x_{ch} + 1x_o + 2.5x_b + 1.5x_d \\ &500x_{ch} + 150x_o + 230x_b + 100x_d \leq 5000 && \text{// flour} \\ &10x_{ch} + 2x_o + 7x_b + x_d \leq 125 && \text{// eggs} \\ &50x_{ch} + 10x_o + 15x_b \leq 500 && \text{// salt} \\ &x_{ch}, x_o, x_b, x_d \geq 0. \end{aligned}$$

For a quick online solver, we can use for example `https://cocoto.github.io/glpk-online/`.

A functional GNU MathProg code is:

```
var c >= 0;
var o >= 0;
var b >= 0;
var d >= 0;
maximize obj: 5*c + 1*o + 2.5*b + 1.5*d;
fl: 500*c + 150*o + 230*b + 100*d <= 5000;
eggs: 10*c + 2*o + 7*b + d <= 125;
salt: 50*c + 10*o + 15*b <= 500;
solve;
end;
```

From George Dantzig, one of the fathers of linear optimization:

I decided to use the simplex method and linear programming to solve a diet problem designed for me to lose weight. Anne agreed she would prepare my meals according to what the computer declared was the optimal diet. (...)

So it's getting late in the day and finally Anne calls me up and she says: "Well, what's for supper?" And I said: "Well, we ran the program. A couple of gallons of vinegar and some other stuff were the optimum diet. We'll just gonna have to take vinegar now as our food."

Source: `https://www.informs.org/Explore/History-of-O.R.-Excellence/Oral-Histories/George-Dantzig`.

Cutting paper rolls

A paper mill manufactures rolls of paper of a standard width 3 meters. But customers want to buy paper rolls of shorter width, and the mill has to cut such rolls from the 3 m rolls. One 3 m roll can be cut, for instance, into two rolls 93 cm wide, one roll of width 108 cm, and a rest of 6 cm (which goes to waste).

One order could be:

- 97 rolls of width 135 cm,
- 610 rolls of width 108 cm,
- 395 rolls of width 93 cm, and
- 211 rolls of width 42 cm.

What is the smallest number of 3 m rolls that have to be cut in order to satisfy this order, and how should they be cut?

The key question for this problem: *What should the variables be?* We cannot simply have a variable x_{135} of rolls of width 135 cm, because we know how many we need – namely 97 – and their number does not help to discover the best partition.

With only four types of rolls, there is only a limited number of ways to cut a single 3 m roll. We can call these *configurations*.

For example, we can cut one roll into one 108 cm roll, one 93 cm and two 42 cm rolls, leaving a wasted paper strip of width 15 cm. We plan to have a variable $y_7 \in \mathbb{R}^+$ for how many configurations of this type we shall use.

Where does this variable appear? If we have an inequality denoting that there are at least 211 rolls of width 42 cm, then it will look like this:

$$\cdots + 2y_7 + \cdots \geq 211.$$

Note the 2 on the left side, as we create 2 rolls of width 42 cm with this configuration. We also create rolls of width 108 cm and 93 cm, so it will appear with coefficient 1 in the corresponding inequalities.

It also will appear in the objective, because we wish to minimize the number of rolls used, with coefficient 1.

In the end, we have 12 maximal configurations:

C1:	2×135
C2:	$135 + 108 + 42$
C3:	$135 + 93 + 42$
C4:	$135 + 3 \times 42$
C5:	$2 \times 108 + 2 \times 42$
C6:	$108 + 2 \times 93$
C7:	$108 + 93 + 2 \times 42$
C8:	$108 + 4 \times 42$
C9:	3×93
C10:	$2 \times 93 + 2 \times 42$
C11:	$93 + 4 \times 42$
C12:	7×42

And the resulting LP is this:

$$\begin{aligned} \min y_1 + y_2 + \ldots + y_{12} \\ 2y_1 + y_2 + y_3 + y_4 \geq 97 \\ y_2 + 2y_5 + y_6 + y_7 + y_8 \geq 610 \\ y_3 + 2y_6 + y_7 + 3y_9 + 2y_{10} + y_{11} \geq 395 \\ y_2 + y_3 + 3y_4 + 2y_5 + 2y_7 + 4y_8 + 2y_{10} + 4y_{11} + 7y_{12} \geq 211 \\ y_1, \ldots, y_{12} \geq 0 \end{aligned}$$

The GNU MathProg code would be:

```
set Configurations := (1..12);

var y{i in Configurations}, >= 0;

minimize obj: sum{i in Configurations} y[i];

cond0: 2*y[1] + y[2] + y[3] + y[4] >= 97;
cond1: y[2] + 2*y[5] + y[6] + y[7] + y[8] >= 610;
cond2: y[3] + 2*y[6] + y[7] + 3*y[9] + 2*y[10] + y[11] >= 395;
cond3: y[2] + y[3] + 3*y[4] + 2*y[5] + 2*y[7] + 4*y[8]
+ 2*y[10] + 4*y[11] + 7*y[12] >= 211;
solve;
end;
```

Source: From Understanding and Using Linear Programming.

Maximum independent set

D(Independent set of vertices**):** For a graph $G = (V, E)$, a set $I \subseteq V$ of vertices is called independent if no two vertices of I are connected by an edge in G.

T: The problem of deciding whether there exists an independent set of size at least k in the graph (or not) is NP-complete.

Integer linear program:

$$\begin{aligned} \max \sum_{v \in V} x_v \\ \text{For each edge } \{u, v\} \in E : x_u + x_v \leq 1, \\ \text{For every variable } x_v : x_v \in \{0, 1\}. \end{aligned}$$

Shortest path

Given s and t in a directed graph $G = (V, E)$, with each edge having a positive distance d_e associated with it, we wish to find a shortest path from s to t.

First LP: Could we model it as a flow problem, with one unit of flow flowing from s to t? If we are allowed an integer linear program, this returns exactly one of the shortest paths:

$$\min \sum_{uv \in E} d_{uv} x_{uv}$$

$$\text{outflow one from source: } \sum_{v | sv \in E} x_{sv} = 1$$

$$\text{Kirchhoff's law: } \forall v \in V \setminus \{s,t\} \colon \sum_{u|uv \in E} x_{uv} - \sum_{w|vw \in E} x_{vw} = 0$$

$$\forall uv \in E \colon x_{uv} \in \{0,1\}$$

Whenever we deal with flows obeying Kirchhoff's law almost everywhere, we should be wary of *circulations*, which might appear in the solution. However, this is not the case here:

O: For the optimal solution to the integer LP above, or its *linear relaxation* with $x_{uv} \in [0,1]$, no circulations will occur.

We also observe that the length of the shortest path does not change if we consider the linear relaxation, which is an LP and thus solvable in polynomial time:

O: The optimal value of the integer LP above and its relaxation with $x_{uv} \in [0,1]$ are the same.

Second LP: Imagine that instead, we wish to store the length of the shortest path from s to v in each vertex v, much like a Bellman-Ford algorithm would. We would have variables y_v in each vertex, and could consider this LP:

$$\max y_t - y_s$$

$$\forall uv \in E \colon y_v - y_u \leq d_{uv}$$

$$\forall v \in V \colon y_v \in \mathbb{R}$$

T: The following linear program has the same optimum value as the length of the shortest path from s to t.

Exercises

Exercise one. A cargo plane has three compartments for storing cargo: front, center and rear. These compartments have the following limits on both weight and space:

Compartment	Weight capacity (tons)	Space capacity (m^3)
Front	10	6800
Center	16	8700
Rear	8	5300

Furthermore, the weight of the cargo in the respective compartments must be the same proportion of that compartment's weight capacity to maintain the balance of the plane.

The following four cargoes are available for shipment on the next flight:

Cargo	Weight (tons)	Volume (m^3/ton)	Profit (EUR/ton)
C1	18	480	310
C2	15	650	380
C3	23	580	350
C4	12	390	285

Any proportion of these cargos can be accepted. The objective is to determine how much (if any) of each cargo C1, C2, C3 and C4 should be accepted and how to distribute each among the compartments so that the total profit for the flight is maximised.

1. Model this task as a linear program.
2. Write this linear program in the GNU MathProg format and present the optimal solution from the solver in class.

Exercise two. Let's make some chocolate! As chocolate is in demand throughout the year, it is important to plan the production properly.

The predicted demand according to which we want to plan production is $d_i > 0$ for the i-th month in tons.

Use a linear program to express how much chocolate is to be produced in each month in the following year, so that we always meet demand while spending as little as possible on production.

A change in production volume of 1 ton between the following months costs 1500 PLN (due to redundancies or recruitment etc.) and storing 1 ton of chocolate costs 600PLN (counting from one month to the nexth).

Don't count the production of the chocolate itself, as it is paid for out of sales. Also assume that the chocolate does not spoil and you should not have any left at the end of December.

Hint: Define s_i as the surplus of chocolate in the ith month, and set $s_0 = s_{12} = 0$.

Exercise three. A classical NP-complete problem is Graph 3-Coloring. It is a decision problem: given a graph G, decide if it its vertices can be colored with three colors such that, if we look at each edge in G, this edge is *not* monochromatic, thus it sees two colors on its endpoints.

Model this problem as an integer linear program. Since this is a decision version, there will be no objective function – this is still acceptable for a solver, it will just find any feasible solution.

Exercise four. Josef K. got an exercise at his Intro to Linear Optimization class:

Design an integer program for the travelling salesman problem: For a given graph with distances $G = (V, E, f)$, where $f \colon E \to \mathbb{R}_0^+$, find a Hamiltonian cycle with the shortest length.

He suggests the following:

"For every edge uv we have a variable $x_{uv} \in \{0, 1\}$, the target function is min $\sum_{uv \in E} f(uv)x_{uv}$ and for every vertex u we create a condition of the form $\sum_{i|ui \in E} x_{ui} = 2$."

Prove that Josef K. got the right solution – or prove him wrong.

Exercise five. A *matching* in a graph is a collection of edges such that no edge shares an endpoint with any other edge. We can imagine this as assigning the vertices of the graph to pairs, much like in kindergarten.

A *perfect* matching is a matching such that every vertex is a member of some pair. All graphs have at least some matchings (since an empty set is also a matching, technically) but not all graphs have a perfect matching.

A graph is *bipartite* if its vertices belong to two groups (parts) A and B, and all edges are going from one vertex $a \in A$ to a vertex in $b \in B$ – so there are no edges inside each part.

It is easier to look for perfect matchings in bipartite graphs compared to general graphs – we will talk about this later in the semester.

Your task is to write an *integer* linear program that computes the minimum-cost perfect matching on a bipartite graph. In other words, you have a bipartite graph $G = (A \cup B, E)$ and a cost function $c : E \to \mathbb{R}^+$ and you wish to find the perfect matching with the cheapest total cost.

Exercise six. Let us explore matching a little bit more, but in a simpler setting. Suppose that instead of a perfect matching, we care about *maximum* matching (without costs) – in other words, for an unweighted graph G, since there always exists at least some matching, we wish to find the matching of maximum size.

Consider the following linear program, which "sounds" like a natural candidate for the problem:

$$\max \sum_{uv \in E} x_{uv}$$

$$\forall v \in V \colon \sum_{uv \in E} x_{uv} \leq 1$$

$$\forall uv \in E \colon x_{uv} \geq 0$$

What is the optimum objective value of this linear program on cycles C_k for $k \geq 3$ and what are the corresponding optimal values of x_{uv} ?

Hint: You can model some small examples (C_3, C_4, C_5) with a solver and then infer the general form from them.