Linear programming

Mathematical programming

Mathematical Programming (MP) is a branch of mathematics in which an optimization problem is encoded using a model consisting of three parts: the declaration of the domains of variables, the objective function defined using these variables, and constraints on the values of these variables. Linear Programming (LP) is a branch of MP in which the target function and constraints are linear with respect to variables.

Standard form of the LP model:

$$\max \sum_i w_i x_i$$

subject to:

$$\sum_i v_i x_i \leq b_1$$

$$\sum_i u_i x_i \leq b_2$$

$$\vdots$$

$$\sum_i z_i x_i \leq b_n$$

$$\forall_i \, x_i \geq 0$$

The solution to the LP model is a vector of values of the variables. *A feasible* solution is the solution for which all constraints in the model are met. *The optimal* solution is the feasible solution for which the objective function attains the minimum (maximum) value. To solve the LP model, tools called solvers are used. The solver finds the optimal solution or returns a proof of the absence of a feasible solution.

The LP model can be used to describe a real system or process:

- Variables represent the configuration parameters and internal state of the system,
- The objective function represents the value returned by the system, e.g., production volume, profit,
- Constraints represent dependencies between system variables.

LP format is a language for LP models understood by most solvers available on the market. The LP format allows you to declare variable domains, write the objective function in the form of a weighted sum of variables and the constraints in the form of linear inequalities, e.g.:

```
Max
550 x1 + 600 x2 + 350 x3 + 400 x4 + 200 x5
Subject to
12 x1 + 20 x2 +         25 x4 + 15 x5   <= 288
10 x1 +  8 x2 + 16 x3                   <= 192
20 x1 + 20 x2 + 20 x3 + 20 x4 + 20 x5   <= 384
End
```

A detailed description of the LP format can be found in the documentation.

ZIMPL

Unfortunately, the LP format does not use sum operators (∑), quantifiers (∀), sets, conditional expressions, functions, and other structures often found in the typical mathematical formulation of the LP model, e.g.:

$$\min \sum_{f \in F} c_f x_f$$

subject to:

$$\bigvee_{n \in N} \sum_{f \in F} \pi_{fn} x_f \geq \Pi_n$$

$$\bigvee_{f \in F} 0 \leq x_f \leq \Delta_f$$

$$\bigvee_{f \in F} x_f \in \mathbb{N}_{\geq 0}$$

For this reason, higher-level modeling languages were created, e.g.: AMPL, GAMS, ZIMPL. These languages transcode to LP format and thus the models represented in the high-level languages can be solved by ordinary solvers. AMPL and GAMS are commercial solutions, while ZIMPL is an open-source solution.

The above formulated LP model in ZIMPL can be written as follows:

```
set Food      := { "Oatmeal", "Chicken", "Eggs",
                   "Milk",    "Pie",     "Pork" };
set Nutrients := { "Energy",  "Protein", "Calcium" };
set Attr      := Nutrients + { "Servings", "Price" };

param needed[Nutrients] :=
   <"Energy"> 2000, <"Protein"> 55, <"Calcium"> 800;

param data[Food * Attr] :=
          |"Servings","Energy","Protein","Calcium","Price"|
|"Oatmeal"|         4 ,    110 ,       4 ,       2 ,     3 |
|"Chicken"|         3 ,    205 ,      32 ,      12 ,    24 |
|"Eggs"   |         2 ,    160 ,      13 ,      54 ,    13 |
|"Milk"   |         8 ,    160 ,       8 ,     284 ,     9 |
|"Pie"    |         2 ,    420 ,       4 ,      22 ,    20 |
|"Pork"   |         2 ,    260 ,      14 ,      80 ,    19 |;
#                       (kcal)      (g)      (mg) (cents)

var x[<f> in Food] integer >= 0 <= data[f, "Servings"];

minimize cost: sum <f> in Food : data[f, "Price"] * x[f];

subto need: forall <n> in Nutrients do
   sum <f> in Food : data[f, n] * x[f] >= needed[n];
```

Thanks to the use of sets, parameters, sum operators, and forall quantifiers, the model saved in this way is more generic than the analogous model in the LP format. A change in a single parameter value or a set does not require rewriting the objective function and constraints. A detailed description of the ZIMPL language can be found in the documentation.

NEOS server

The NEOS server offers cloud-based access to many MP solvers. Using the server is free, however, there are limits on the model size and execution time. See the documentation for details.

Exercises

1. The composition of food products indicated on the packaging label indicates the ingredients in the order of decreasing weight, but it is not required to specify the specific weights of the individual ingredients. Using linear programming, try to determine the weight of individual ingredients in a chocolate bar based on the data from the label below.

Dark chocolate with orange and almond slivers

Ingredients: sugar, cocoa mass, orange preparation 7% (orange 34%, sugar, apple, pineapple fibres, acidity regulator [citric acid], gelling agent [sodium alginate], stabiliser [calcium phosphate], flavouring), **almonds** 7%, cocoa butter, anhydrous **milk** fat, emulsifier (**soya** lecithin), flavourings.
May contain **hazelnuts**, **milk**.
Dark chocolate contains: Cocoa solids: 48% min.

NUTRITION INFORMATION PER 100g:	
Energy	2193 kJ / 526 kcal
Fat	32 g
- of which saturates	18 g
Carbohydrate	49 g
- of which sugars	42 g
Protein	6.7 g
Salt	0.09 g

Save the LP model for chocolate bar composition in the LP format. Take into account:

- Domains of variables representing chocolate ingredients resulting from the label; assume that each of the food additives (acidity regulator, gelling agent, stabilizer, flavorings) does not constitute more than 2g of the product; note that some of the additives are in the composition of the orange preparation, which constitutes only 7% of chocolate bar (7% * 2g = 0.14g),
- The weight of the chocolate bar is 100g (the fixed scale for model),
- The order of the listed ingredients,
- The minimum content of cocoa solids (consisting of cocoa mass and cocoa butter) in the dark chocolate (i.e. 86% of the product),
- The total sugar content (note that sugar is also in fruits),
- The total fat content (note that fat is in cocoa solids and almonds),
- The total salt content (sodium alginate and calcium phosphate are salts); salt may also occur naturally in other ingredients.
- Model the composition of the orange preparation analogously.

Using the NEOS Server and the Gurobi solver, try to get a feasible solution to your model. Does it exist? What does it look like?

2. Find the range of possible weight of added sugar and cocoa solids by extending the model from Exercise 1 using an objective function that either minimizes or maximizes the weight, separately for each component.

3. To narrow down the range of possible weight of ingredients, copy the chocolate model from Exercise 2 and extend it with additional constraints using the nutrient data from the table below and from the chocolate label.

Content in 100g of product	Sugar	Cocoa mass	Orange	Apple	Almond	Cocoa butter	Milk fat	Soy lecithin
Energy value [kcal]	388.9	227.50	47.00	52.10	585.0	900.0	900.0	784.0
Fat [g]	0.0	14.00	0.12	0.20	51.0	99.9	99.9	84.0
Protein [g]	0.0	16.55	0.94	0.26	17.1	0.0	0.0	0.0
Sugar [g]	99.8	1.75	9.35	10.40	4.8	0.0	0.0	5.0

Using the NEOS Server and the Gurobi solver, try to get a feasible solution to your model. Does it exist? What could be the cause?

4. The figures on the label may be rounded. Extend the domains of variables and modify the values in constraints so that the constraints become less restrictive, but at the same time the rounding of numeric values matches the label. Does the model obtained in this way have a solution? What are the values of the variables?

5. Determine the possible weight of added sugar and cocoa solids by adding to the model from Example 4 a function that either minimizes or maximizes the weight, separately for each component. Are the results more reliable than the results of Exercise 2?

6. Rewrite the model from Exercise 5 in ZIMPL and optimize it using the scip solver. Note: If scip does not return the value of a variable in the solution, the variable attains the value of 0.