Programmation linéaire avec SCIP

Christoph Dürr
CentraleSupélec — 2018

Vous avez vraiment de bons yeux

Pour aller plus loin, consultez le
ZIMPL User Guide de Thorsten Koch

Format LP par l'exemple

- Une usine dispose d'un stock de roues, de matière première plastique et du fer
- Elle peut produire 3 sortes de jouets: voiture, camion, tricycle, vélo. Chaque jouet nécessite une quantité du stock et sa vente produit un profit.
- On sait qu'on ne vendra jamais plus que 20 tricycles.
- On veut écouler tout le fer.
- Combien produire de chaque sorte en respectant le stock et en maximisant le profit?

La vie serait triste si toutes les variables s'appelaient x1, x2,...

ou *Minimize*

multiplication sans *

pas d'accents

```
\Commentaire
Maximize
 obj: 17 voiture + 20 camion + 34.50 tricycle + 38 velo
Subject To
 roues: 4 voiture + 4 camion + 3 tricycle + 2 velo <= 1000
 plastique: 5 voiture + 17 camion + 8.5 tricycle <= 300
 fer: 20 camion + 35 velo = 500
Bounds
  0 <= tricycle <= 20
General
  voiture camion tricycle velo
Binary
End
```

noms contraintes optionnels

ou >= ou = mais pas > ou !=

par défaut 0<= var < +Infinity

section *General* contient les variables entières

Le langage ZIMPL

- Extension .zpl
- Permet de modéliser un programme linéaire
- Similaire à AMPL ou GNU MathProg
- Description compacte et paramétré des contraintes et variables
- Les données peuvent être dans un fichier séparé (en format ZIMPL ou CSV)
- *soplex* ne lit que .lp
  scip lit .zpl aussi.

Exemple plus court chemin

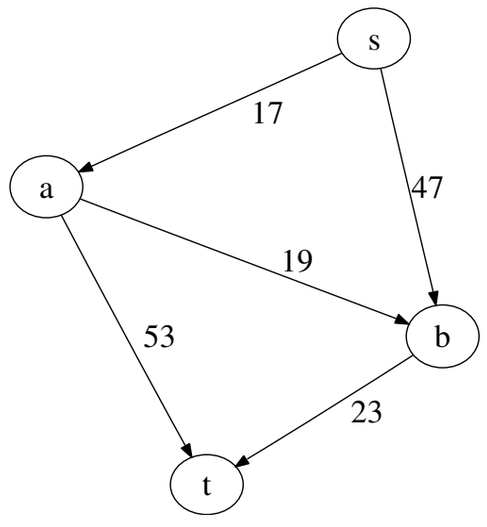
Consider the LP formulation of the shortest s, t-path problem, applied to some directed graph (V, A) with cost coefficient c_{ij} for all $(i, j) \in A$:

$$\min \quad \sum_{(i,j)\in A} c_{ij} x_{ij}$$

$$\sum_{(iv)\in\delta^-(v)} x_{iv} = \sum_{(vi)\in\delta^+(v)} x_{vi} \quad \text{for all } v \in V \setminus \{s, t\} \tag{1}$$

$$x_{ij} \in \{0, 1\}, \text{for all i,j in A}$$

where $\delta^+(v) := \{(v, i) \in A\}$, $\delta^-(v) := \{(i, v) \in A\}$ for $v \in V$. For a given graph the instantiation is



$$\begin{aligned} \min \quad & 17x_{sa} + 47x_{sb} + 19x_{ab} + 53x_{at} + 23x_{bt} \\ \text{subject to} \quad & x_{sa} = x_{ab} + x_{at} \\ & x_{sb} + x_{ab} = x_{bt} \\ & x_{ij} \in \{0, 1\}, \text{for all i,j} \end{aligned}$$

arcs sortants, entrants

variable indicatrice : xij=1 selectionne (i,j) pour le chemin

Oups!
Il manque la contrainte
Σ x_{si} = 1 [somme sur (si) ∈ δ⁺(s)]
qui assure la sélection *d'un* chemin

Exemple plus court chemin

autres notations (utiles pour le problème des n reines):
param n := 100;
set N := { 0 .. n-1 };
set G := N cross N;

ensemble d'éléments

ensemble de couples

notation comme un dictionnaire: liste de clé valeur, sans : comme en Python

```
set V :={"a","b","s","t"};
set A :={<"s","a">, <"s","b">, <"a","b">, <"a","t">, <"b","t">};
param c[A] := <"s","a"> 17, <"s","b"> 47, <"a","b"> 19, <"a","t"> 53, <"b","t"> 23;
defset dminus(v) := {<i,v> in A};
defset dplus(v) := {<v,j> in A};

var x[A] binary;

minimize cost: sum<i,j> in A: c[i,j] * x[i,j];

subto fc:
forall <v> in V - {"s","t"}:
   sum<i,v> in dminus(v): x[i,v] == sum<v,i> in dplus(v): x[v,i];

subto uf:
   sum<s,i> in dplus("s"): x[s,i] == 1;
```

le v est fixé, et le i est variable. Cette notation permet une sorte de pattern matching

notations alternatives

var x1;
var x2 binary;
var x3 integer >= -infinity;
var y[A] real >= 2 <= 18; \réel par défault

génère une contrainte par sommet v

2 fois =

Exemple pyscipopt

environment

```
#/usr/bin/env pypy

from pyscipopt import Model, quicksum

model = Model("try")

x = model.addVar("x")
y = model.addVar("y")
z = [model.addVar("z%i" % i) for i in range(10)]
```

```
addVar(self,            # the SCIP model
  name = '',            # name of the variable
  vtype = 'C',          #'C', 'I', or 'B'
  lb = 0.0,             # lower bound
  ub = None ,           # upper bound
  obj = 0.0,            # objective coefficient
  pricedVar = False ) # is it a pricing candidate?
```

résultat identique à sum, mais exécution plus rapide

```
model.addCons(2 * x + y <= 10, "costs")
model.addCons(x + quicksum(z[i] for i in range(10)) <= 30, "weight")

model.setObjective( x + y , "maximize")
model.optimize()

if model.getStatus() != 'optimal':
    print('LP is not feasible!')
else:
    print("Optimal value: %f" % model.getObjVal())
    print("x: = %f" % model.getVal(x))
    print("y: = %f" % model.getVal(y))
```

ou « minimize"

"optimal", "unbounded", "infeasible"

Pour aller plus loin :

- PySCIPOpt/examples
- PySCIPOpt.pdf *Mathematical Programming in Python with the SCIP Optimization Suite* par Stephen Maher et al. (explique les callbacks)