Optimization Labs

Softwares for LP solving and optimization

Mixed Integer Linear Programming Solvers

- Several MILP solvers are available on the market, from open-source to commercial products.
- Among the best performing there are CPLEX (IBM), Gurobi, Xpress-MP.
- Most of those solvers are callable with APIs in different languages (Python, C, C++, C#, Java, etc).

- Let’s consider what Matlab offers.

Matlab Optimization Toolbox

Optimization Toolbox

- https://it.mathworks.com/help/optim/index.html

- The Optimization Toolbox provides functions for defining an optimization problem and solving it. Among other optimization problems, there is a MILP solver.

- The model definition can be *solver based* (input in matrix form, very complicate for large models) or *model based* (we will use this one, allowing to define variables and constraints one by one)

$$\min_x(-3x_1 - 2x_2 - x_3) \text{ subject to } \begin{cases} x_3 \text{ binary} \\ x_1, x_2 \geq 0 \\ x_1 + x_2 + x_3 \leq 7 \\ 4x_1 + 2x_2 + x_3 = 12 \end{cases}$$

Create an `OptimizationProblem` object named `prob` to represent this problem. To specify a binary variable, create an optimization variable with integer type, a lower bound of 0, and an upper bound of 1.

```
x = optimvar('x',2,'LowerBound',0);
xb = optimvar('xb','LowerBound',0,'UpperBound',1,'Type','integer');
prob = optimproblem('Objective',-3*x(1)-2*x(2)-xb);
cons1 = sum(x) + xb <= 7;
cons2 = 4*x(1) + 2*x(2) + xb == 12;
prob.Constraints.cons1 = cons1;
prob.Constraints.cons2 = cons2;
```

$$\min_x(-3x_1 - 2x_2 - x_3) \text{ subject to } \begin{cases} x_3 \text{ binary} \\ x_1, x_2 \geq 0 \\ x_1 + x_2 + x_3 \leq 7 \\ 4x_1 + 2x_2 + x_3 = 12 \end{cases}$$

Create an `OptimizationProblem` object named `prob` to represent this problem. To specify a binary variable, create an optimization variable with integer type, a lower bound of 0, and an upper bound of 1.

```
x = optimvar('x',2,'LowerBound',0);
xb = optimvar('xb','LowerBound',0,'UpperBound',1,'Type','integer');
prob = optimproblem('Objective',-3*x(1)-2*x(2)-xb);
cons1 = sum(x) + xb <= 7;
cons2 = 4*x(1) + 2*x(2) + xb == 12;
prob.Constraints.cons1 = cons1;
prob.Constraints.cons2 = cons2;
```

Defines an array x of variables with two elements (x(1) and x(2)).
Their values is continuous and with a lower bound of 0 (x(1) >= 0, x(2) >= 0)

$$\min_x(-3x_1 - 2x_2 - x_3) \text{ subject to } \begin{cases} x_3 \text{ binary} \\ x_1, x_2 \geq 0 \\ x_1 + x_2 + x_3 \leq 7 \\ 4x_1 + 2x_2 + x_3 = 12 \end{cases}$$

Create an `OptimizationProblem` object named `prob` to represent this problem. To specify a binary variable, create an optimization variable with integer type, a lower bound of 0, and an upper bound of 1.

```
x = optimvar('x',2,'LowerBound',0);
xb = optimvar('xb','LowerBound',0,'UpperBound',1,'Type','integer');
prob = optimproblem('Objective',-3*x(1)-2*x(2)-xb);
cons1 = sum(x) + xb <= 7;
cons2 = 4*x(1) + 2*x(2) + xb == 12;
prob.Constraints.cons1 = cons1;
prob.Constraints.cons2 = cons2;
```

Defines a variable xb (no dimension is specified, so, it's a single variable). Bounds are 0<=x<=1 and the variable type is integer.

$$\min_x(-3x_1 - 2x_2 - x_3) \text{ subject to } \begin{cases} x_3 \text{ binary} \\ x_1, x_2 \geq 0 \\ x_1 + x_2 + x_3 \leq 7 \\ 4x_1 + 2x_2 + x_3 = 12 \end{cases}$$

Create an `OptimizationProblem` object named `prob` to represent this problem. To specify a binary variable, create an optimization variable with integer type, a lower bound of 0, and an upper bound of 1.

```
x = optimvar('x',2,'LowerBound',0);
xb = optimvar('xb','LowerBound',0,'UpperBound',1,'Type','integer');
prob = optimproblem('Objective',-3*x(1)-2*x(2)-xb);
cons1 = sum(x) + xb <= 7;
cons2 = 4*x(1) + 2*x(2) + xb == 12;
prob.Constraints.cons1 = cons1;
prob.Constraints.cons2 = cons2;
```

Defines an optimization problem.
Sets the objective function of the problem.

Note: problem and objective function definitions can be also written separately:

prob = optimproblem;
*obj = -3*x(1)-2*x(2)-xb;*
prob.Objective = obj;

$$\min_x(-3x_1 - 2x_2 - x_3) \text{ subject to } \begin{cases} x_3 \text{ binary} \\ x_1, x_2 \geq 0 \\ x_1 + x_2 + x_3 \leq 7 \\ 4x_1 + 2x_2 + x_3 = 12 \end{cases}$$

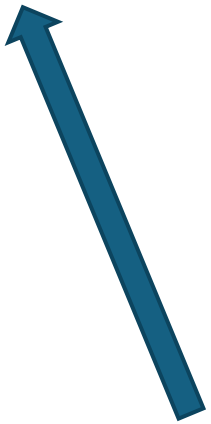
Create an `OptimizationProblem` object named `prob` to represent this problem. To specify a binary variable, create an optimization variable with integer type, a lower bound of 0, and an upper bound of 1.

```
x = optimvar('x',2,'LowerBound',0);
xb = optimvar('xb','LowerBound',0,'UpperBound',1,'Type','integer');
prob = optimproblem('Objective',-3*x(1)-2*x(2)-xb);
cons1 = sum(x) + xb <= 7;
cons2 = 4*x(1) + 2*x(2) + xb == 12;
prob.Constraints.cons1 = cons1;
prob.Constraints.cons2 = cons2;
```

The two constraints of the problem are defined with names cons1 and cons2.

cons1 and cons2 are added to the problem.

```
% Run the LP solver

[Sol fval exitflag]=solve(prob,'solver','intlinprog');
```



OUTPUT

Sol.x(1)
Sol.x(2) are the values of the optimal solution

fval is the value of the optimal objective function

exitflag is 'OptimalSolution' if the solver concluded the optimization.

Ask for solver «intlinprog» in the case of linear programming with integer/boolean/continuous variables.

Other possible values, among others, are «Unbounded», «IntegerFeasible», «NoFeasiblePointFound».

Examples

Example «Knapsack»

```
%% Example

clc
clear

% Decision variables
x=optimvar('x',6,'Type', 'integer', 'LowerBound',0);

% Problem definition
Problem=optimproblem('ObjectiveSense', 'Maximize');

% Obj and constraints
Problem.Objective       =   10*x(1)+30*x(2)+6*x(3)+3*x(4)+20*x(5)+8*x(6);
Problem.Constraints.c1 =   0.5*x(1)+x(2)+0.33*x(3)+0.1*x(4)+x(5)+0.5*x(6) <= 10;
Problem.Constraints.c2 =   x(1)>=2;
Problem.Constraints.c3 =   x(2)>=2;
Problem.Constraints.c4 =   x(3)>=6;
Problem.Constraints.c5 =   x(4)>=10;
Problem.Constraints.c6 =   x(5)>=1;
Problem.Constraints.c7 =   x(6)>=2;

% Run the LP solver

[Sol fval exitflag]=solve(Problem,'solver','intlinprog');
```

Example «Knapsack» (2)

```
% Dati del problema

C = [10  30 6    1    20  8];
W = [0.5 1  0.33 0.1  1   0.5];
MaxW = 10;
MinQ = [2 2 6 10 1 2];

% Decision variables
N = size(C,2);
x=optimvar('x',N,'Type', 'integer', 'LowerBound',0);

% Problem definition
Problem=optimproblem('ObjectiveSense', 'Maximize');

% Obj and constraints
Problem.Objective      =   C*x;
Problem.Constraints.w =   W*x <= MaxW;

for(i=1:N)
    WConstr(i) = x(i)>=MinQ(i);
end

Problem.Constraints.Wconstr = WConstr;

% Run the LP solver

[Sol fval exitflag]=solve(Problem,'solver','intlinprog');
```

- In this case we define separately the data of the problem (objects) and the model is generalized.
- In such a way it is easier to change instance of a problem.
- Note the operator * for directly write the product between a vector of a coefficients and a vector of variables.

Examples

An alternative way to write the constraints is to crate with a loop the sum of loaded objects, and then write the constraint with it.

$$\sum_{i=1}^{N} W_i x_i \leq MaxW$$

```
Problem.Constraints.w =    W*x <= MaxW;
```

```
loadedweight = 0;
for(i=1:N)
    loadedweight = loadedweight + W(i)*x(i);
end
Problem.Constraints.w = loadedweight <= MaxW;
```

In this case it’s an equivalente but longer implementation. But in could be helpful in other situations where you need to sum several terms according to conditions.

Scheduling example: single machine

$$\min \sum_{j=1}^{n} C_j$$

$$C_j \geq r_j + p_j \qquad \forall j = 1..n$$

$$C_i - p_i \geq C_j - M y_{ij} \qquad \forall i = 1..n,\ j = 1..n : i \neq j$$

$$C_j - p_j \geq C_i - M(1 - y_{ij}) \qquad \forall i = 1..n,\ j = 1..n : i \neq j$$

$$C_j \in R \qquad \forall j = 1..n$$

$$y_{ij} \in \{0,1\} \qquad \forall i = 1..n,\ j = 1..n$$

r_j = release time of job j
p_j = processing time of job j

Variables:

C_j = completion time of job j
x_{ij} = boolean variables for linearizing the disjunctive constraints.

Scheduling example: single machine

```
P(1)=3; % Processing times
P(2)=2;
P(3)=1;
P(4)=8;
P(5)=2;
P(6)=3;

R(1)=0; % Release times
R(2)=10;
R(3)=5;
R(4)=4;
R(5)=6;
R(6)=11;

[M N]=size(P); % Number of machines / jobs.
```

```
C=optimvar('C',N,'Type','continuous','LowerBound',0);
obj = sum(C);
for j=1:N
    RelTime(j)=C(j) >= P(j) + R(j);
end
x=optimvar('x',N,N,'Type','integer','LowerBound',0,'UpperBound',1);
BigM=50;

for j=1:N
    for i=1:N
        if not(i == j)
            D1(i,j)= C(j)-P(j)>=C(i)-BigM*(1-x(i,j));
            D2(i,j)= C(i)-P(i)>=C(j)-BigM*x(i,j);
        end
     end
end
SingleMachineProblem=optimproblem;

SingleMachineProblem.Objective=obj;

SingleMachineProblem.Constraints.RelTime=RelTime;
SingleMachineProblem.Constraints.D1=D1;
SingleMachineProblem.Constraints.D2=D2;

[Jobs fval]=solve(SingleMachineProblem,'solver','intlinprog');
```

Scheduling example: single machine

```
P(1)=3; % Processing times
P(2)=2;
P(3)=1;
P(4)=8;
P(5)=2;
P(6)=3;

R(1)=0; % Release times
R(2)=10;
R(3)=5;
R(4)=4;
R(5)=6;
R(6)=11;

[M N]=size(P); % Number of machines / jobs.
```

```
C=optimvar('C',N,'Type','continuous','LowerBound',0);
obj = sum(C);
for j=1:N
    RelTime(j)=C(j) >= P(j) + R(j);
end
x=optimvar('x',N,N,'Type','integer','LowerBound',0,'UpperBound',1);
BigM=50;

for j=1:N
    for i=1:N
        if not(i == j)
            D1(i,j)= C(j)-P(j)>=C(i)-BigM*(1-x(i,j));
            D2(i,j)= C(i)-P(i)>=C(j)-BigM*x(i,j);
        end
     end
end
SingleMachineProblem=optimproblem;

SingleMachineProblem.Objective=obj;

SingleMachineProblem.Constraints.RelTime=RelTime;
SingleMachineProblem.Constraints.D1=D1;
SingleMachineProblem.Constraints.D2=D2;

[Jobs fval]=solve(SingleMachineProblem,'solver','intlinprog');
```

Multidimensional vector of decisional variables (you can use as many indices you wish)

Solver output

During the execution of the algorithm, it is possible to see the best solution found so far, the best lower bound, and the gap between them.

```
Solving MIP model with:
   602 rows
   625 cols (601 binary, 0 integer, 0 implied int., 24 continuous)
   2858 nonzeros

        Nodes      |    B&B Tree     |            Objective Bounds              |  Dynamic Constraints |
     Proc. InQueue |  Leaves   Expl. | BestBound       BestSol              Gap |   Cuts   InLp Confl. | Lpl
         0       0         0   0.00%   0               inf                  inf        0      0       0
         0       0         0   0.00%   388.4886828     inf                  inf        0      0       0
 L       0       0         0   0.00%   459.29074       567.2221056        19.03%     2389    122      85
 L       0       0         0   0.00%   459.29074       522.4335552        12.09%     2389    122      85
```

Limits of Matlab Optimization Studio

- The default solver *intlinprog* is not very efficient, compared to state-of-the-art ones.
- There are not many tools for analysis/debugging the model, and most are associated with a matrix rapresentation of the model and not with the model based design.
- If you need to solve a small model, not particularly complicated to be implemented, M.O.S. is probably enough (for the exercises of this course, it’s enough to use M.O.S.)
- In case you will ever need, let’s see more options of solvers callable from Matlab.

Gurobi

- If the modelling capabilities of Matlab are enough and just need a more efficient solver, you can install GUROBI:

https://www.gurobi.com/

- Gurobi is free for academic use (when installing the solver you will need to activate the licence from inside the Politechnic network).

- In order to run Gurobi from Matlab, just add two lines before running «solve».

Gurobi

- Add two Gurobi directories to the matlab path:

```
addpath(fullfile('C:\gurobi1003', 'win64', 'examples', 'matlab'));
addpath(fullfile('C:\gurobi1003', 'win64', 'matlab'));
```

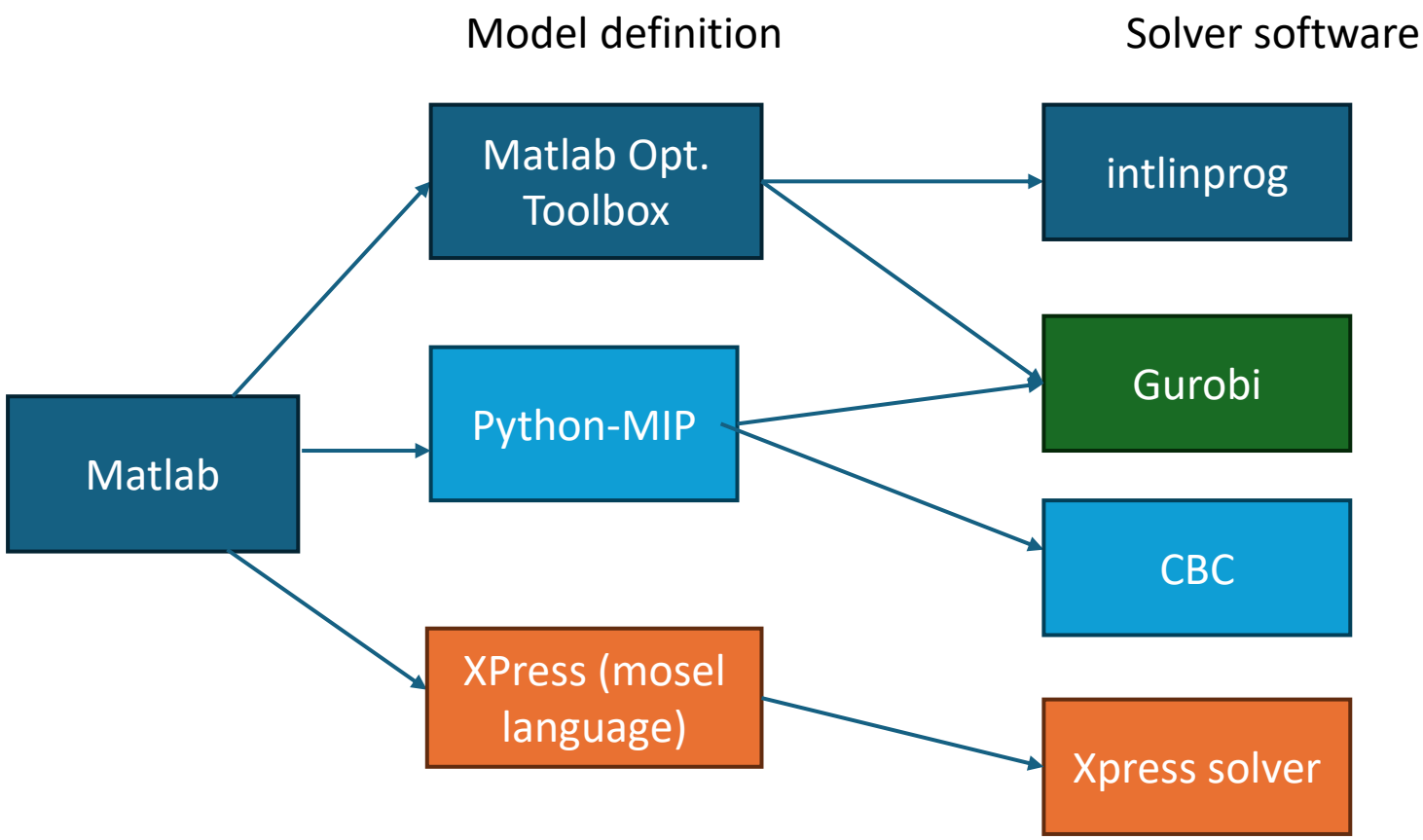
Check the name of the installation directory of Gurobi on your PC

- Then, when you will run *solve, gurobi will be called instead (intlinprog* is automatically substituted)
- Restarting matlab, the path is reset.
- If you wish to make the path change permanent do *savepath*

Other solvers callable from Matlab

- Xpress MP
  - Define the problem in a specific ad-hoc language (**Mosel**) and call it with one line from Matlab.
  - Up: specialized easy language for model definition, debugging and analysis tools, efficient solver.
  - Down: the free «community» version is limited in the dimensions of the models it can solve (LADISPE computers have an unlimited license).
- Python-MIP
  - If you like Python, the best option is **Python-MIP (**https://www.python-mip.com/**).** It uses one of the most efficient free solver (**CBC**), or **Gurobi** can be called instead.
  - Up: code in Python if you are used to it, free software.
  - Down: none that I know.

Other solvers callable from Matlab



Software to install

- Matlab (academic lic.)
- Python-MIP (free)
- Gurobi (academic lic.)
- Xpress (community limited lic.)

Xpress

- https://community.fico.com/s/fico-xpress-optimization-licensing-optio

- From FICO website you can download:
  - A free 60-days full trial is downloadable, or
  - a LIMITED DIMENSION «Community License» (Win64 only –enough for course exercises)

Xpress (called from Matlab) example

```
addpath(fullfile(getenv('XPRESSDIR'),'/matlab'))
% use savepath to make it permanent

% Set covering problem: select the minimal
% number of rows of M having at least one
% "1" for each column.

M=[1 0 0 1 0 0 0;
   0 0 1 0 1 0 1;
   0 1 0 0 1 0 1;
   1 0 0 1 0 1 0;
   0 1 1 0 0 1 0;
   ];

moselexec('setcovering.mos')

x
```

```
model ModelName
uses "mmxprs","mmsystem";

! data and variables definition
declarations
    ROWS, COLS: set of integer
    Matrix: array(ROWS,COLS) of integer
    x:array(ROWS) of mpvar
end-declarations

! input from Matlab
initializations from "matlab.mws:"
    Matrix as "M"
end-initializations
```

MODEL GOES HERE (see next slide)

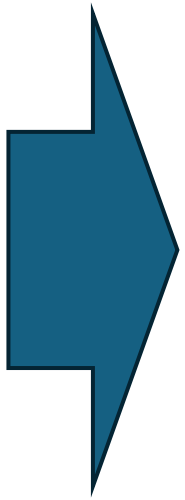
```
initializations to "matlab.mws:"
  evaluation of getobjval as "objval"
  evaluation of array(i in ROWS) x(i).sol as "x"
end-initializations

exit(getprobstat)

end-model
```

Xpress (called from Matlab) example

$$\min \sum_{i=1}^{Rows} x_i$$

$$\sum_{i=1}^{Rows} M_{i,j} x_i \geq 1 \quad j = 1..Cols$$

$$x_i \in \{0,1\} \quad \forall i$$



```
forall (i in ROWS) x(i) is_binary

forall (j in COLS) do
    sum (i in ROWS) Matrix(i,j)*x(i) >= 1
end-do

minimize(sum(i in ROWS) x(i))
```

Python-MIP

The main info if you wish to try using python are:

- **Python-MIP** https://www.python-mip.com/
- Gurobi https://www.gurobi.com/ (to test the external more efficient solver)
- You will need to call a python script from Matlab and pass data. The interface you could use is: https://it.mathworks.com/help/matlab/call-python-libraries.html

- If you do, tell me how it works. Ask me if you wish to try Python-MIP as the project for the course. (producing a small documentation with setup instructions and a few examples like the ones of this presentation and of the laboratory exercises)

Pyton-MIP

Example of a 0/1 knapsack problem implemented in Python

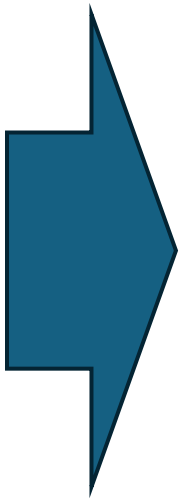
Maximize:

$$\sum_{i \in I} p_i \cdot x_i$$

Subject to:

$$\sum_{i \in I} w_i \cdot x_i \leq c$$

$$x_i \in \{0,1\} \ \forall i \in I$$



```python
from mip import Model, xsum, maximize, BINARY

p = [10, 13, 18, 31, 7, 15]
w = [11, 15, 20, 35, 10, 33]
c, I = 47, range(len(w))

m = Model("knapsack")

x = [m.add_var(var_type=BINARY) for i in I]

m.objective = maximize(xsum(p[i] * x[i] for i in I))

m += xsum(w[i] * x[i] for i in I) <= c

m.optimize()

selected = [i for i in I if x[i].x >= 0.99]
print("selected items: {}".format(selected))
```