Bachelor's degree in Bioinformatics

Solving optimization problems with AMPL modeling language

Prof. Renato Bruni

bruni@dis.uniroma1.it

Department of Computer, Control, and Management Engineering (DIAG)

"Sapienza" University of Rome

What is a Modeling Language?

- It's a language to write an optimization **problem** in such a way that it can be handled by a software **solver**
- Much easier than implementing the solution algorithm in some programming language like python or c++
- There exist a number of programming languages, we will study **AMPL**, A Mathematical Programming Language, probably the most known
- **AMPL** is not a solver, it can use several solvers. For example **Cplex**, which solves LP and ILP, and is an implementation of simplex algorithm, branch-and-bound, branch-and-cut
- The **full version** of AMPL requires a license, the **student version** is limited but **free**
- Downloadable from https://ampl.com/try-ampl/download-a-free-demo/
- AMPL book is also downloadable from https://ampl.com/resources/the-ampl-book/

Write a model in AMPL

- First, we need to install AMPL. The command line version simply has to be unzipped, and it already contains some solvers
- As many other programming languages, it uses **plain text files**
- One **model file** (*.mod) containing the mathematical structure of the model but generally not the numerical data
- Numerical data may be in a **data file** (*.dat) or read in other ways
- There are some basic elements to express a model: **sets** (structures over which the data are organized), **param** (parameters, the numerical data), **var** (variables of the model)
- We need to declare every element we want to use (sets, parameters, variables, …), and then they can be used to write the objective function and the constraints in the model file

First Very Easy Example

- A company produces 2 products: **standard** and **deluxe**
- It needs 3 resources: **material**, **shaping machine**, **polishing machine**, as in the following table

	1kg standard	1kg deluxe
material	4Kg	4Kg
shaping	4 hours	2 hours
polishing	2 hours	5 hours
we sell at	10EUR/Kg	15EUR/Kg

- Resources are limited: in each week, we only have 75 Kg of material, we can use shaping at most 80 hours and polishing at most 60 hours
- We want to plan production in order to maximize income

Optimization model

- It is a very easy case, we should already know how to model this
- Variables:

  x_1 = **Kg** of standard to be produced weekly

  x_2 = **Kg** of deluxe to be produced weekly
- We obtain the following **Linear Programming problem**:

$$\max \; c^T x$$

$$Ax \leq b$$

$$x_i \geq 0 \quad i \in \{1,2\}$$

with c = (10, 15)^T b =(75, 80, 60)^T A = $\begin{pmatrix} 4 & 4 \\ 4 & 2 \\ 2 & 5 \end{pmatrix}$

File .mod

```
set PROD;      # set of the products
set RES;       # set of the resources
param b{RES};
param a{RES,PROD};
param c{PROD};
var x{j in PROD} >=0;
maximize profit: sum{j in PROD} c[j]*x[j];
s.t. constraints{i in RES}: sum{j in PROD} a[i,j]*x[j] <= b[i];
```

File .dat

```
set PROD:=STANDARD, DELUXE;
set RES:=MATERIAL, SHAPING, POLISHING;

param:          b:=
MATERIAL        75
SHAPING         80
POLISHING       60;

param a:  STANDARD  DELUXE:=
MATERIAL        4           4
SHAPING         4           2
POLISHING       2           5;

param:          c:=
STANDARD    10
DELUXE          15;
```

Execution

start ampl.exe , (opens a command-line window)

give the following commands (always a semicolon at the end):

option solver cplex; (chooses the solver)

model (path)plan.mod; (chooses model file)

data (path)plan.dat; (chooses data file)

solve; (calls the selected solver)

If the files are in the same folder than ampl we can omit the full path, otherwise we need it

We can even write the commands in a plan.run file, to be executed with: include plan.run;

Results

In case of errors, use reset; command (twice)

Then, to correct the errors, read error location, remove the first error, and try again. The subsequent errors may just be consequences of the first one. Note that AMPL cannot detect some errors, which may appear only at run time

After solving, use display command to see the optimal solution or any other object in the model:

display x; (shows the optimal solution x)

If all is correct, for this model we should obtain:

```
x [*] :=
 DELUXE   7.5
 STANDARD  11.25
```

Lot Sizing Example

- We are the owner of a **warehouse**, we receive goods from the **factories** and we ship them to the **shops**
- In the next 5 months (our **time horizon**), for each commodity, we have the **orders** from the shops. Let's focus for example on this commodity

	month1	month2	month3	month4	month5
order	12	4	18	17	7

- For each month, the **unit price** of that commodity from the factory is the following

	month1	month2	month3	month4	month5
price	8	12	7	11	10

Lot Sizing Example

- Every time we buy the commodity from the factory, we also need to pay the following **transportation cost** (a case of fixed cost)

	month1	month2	month3	month4	month5
T cost	10	10	10	10	10

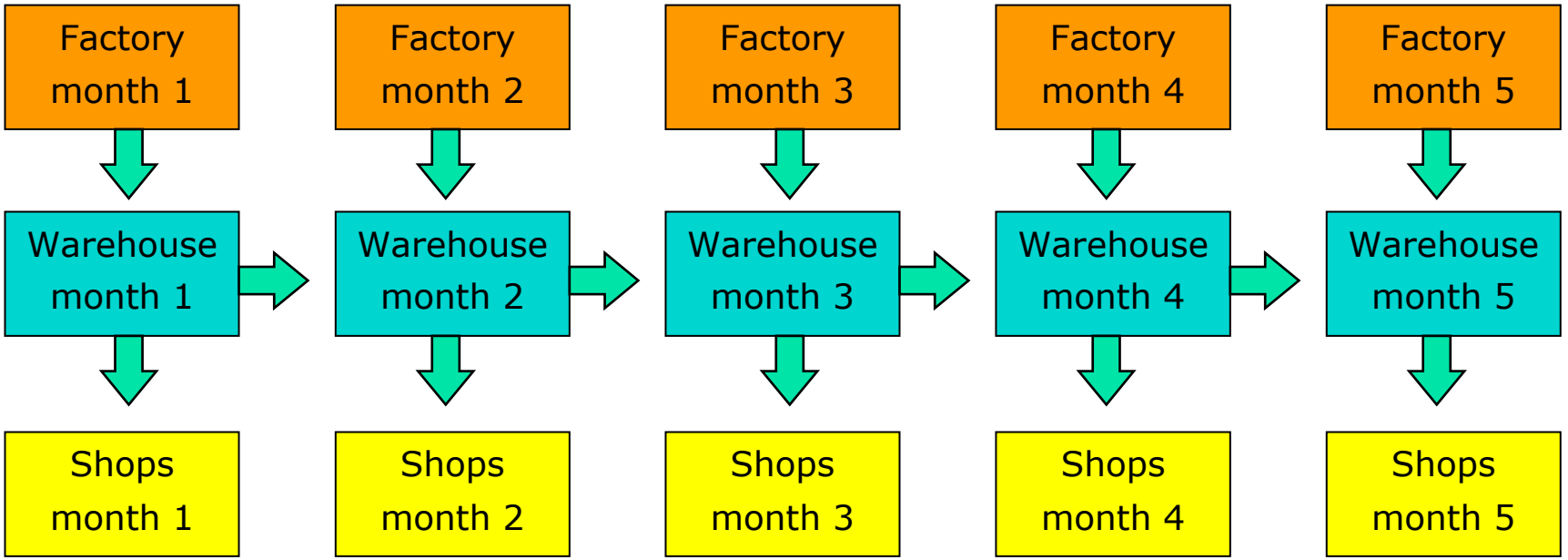
- Finally, keeping the commodity in the warehouse has the following **storage cost** for each unit of commodity and each month

	month1	month2	month3	month4	month5
S cost	5	5	5	5	5

- We want to satisfy orders from the shops and **minimize our expenses**
- How can we do?

The flow of commodity

- Every month, we could either **buy** from the factory or **use** the commodity that was in the warehouse from the previous month, we **send** to the shops, and we can also **leave** part in the warehouse for the future



- The warehouse was **empty** at the **beginning**, and it must remain empty at the **end** of the time horizon

Connection between variables

- What is under our control? What we have to **decide**?
- We need some **Real-valued variables**:

c_i = commodity that we buy at month i, with $i \in \{1,2,3,4,5\}$

s_i = commodity that we leave in stock at month i, with $i \in \{1,2,3,4,5\}$

Moreover, we need **binary variables** to model the transportation cost

$$t_i = \begin{cases} 1 & \text{if we pay transport at month } i, \text{ with } i \in \{1,2,3,4,5\} \\ 0 & \text{otherwise} \end{cases}$$

- Those variables are connected. For the generic month we have:

$$s_i = s_{i-1} + c_i - \text{order}(i) \qquad i \in \{2,3,4,5\}$$

and for the first month: $s_1 = c_1 - \text{order}(1)$

- To model the fixed cost we write: $c_i \leq M\, t_i \qquad i \in \{1,2,3,4,5\}$

Objective function

- Our aim is minimizing the sum of all expenses
- How many **types** of costs?
- Cost of the commodity (per unit) : $Cost_i$
- Transportation cost: $Trans_i$
- Storage cost (per unit per month): $Storage_i$
- So the **total cost** is:

$$\min \sum_{i=1..5} Cost_i \; c_i \quad + \sum_{i=1..5} Trans_i \; t_i \quad + \sum_{i=1..5} Storage_i \; s_i$$

The complete model

$$
\begin{cases}
\min \sum_{i=1..5} Cost_i \; c_i \quad + \sum_{i=1..5} Trans_i \; t_i \quad + \sum_{i=1..5} Storage_i \; s_i \\
s_1 = c_1 - \text{order}(1) \\
s_i = s_{i-1} + c_i - \text{order}(i) \qquad i \in \{2,3,4,5\} \\
c_i \leq M \, t_i \qquad i \in \{1,2,3,4,5\} \\
c_i \geq 0 \qquad i \in \{1,2,3,4,5\} \\
s_i \geq 0 \qquad i \in \{1,2,3,4,5\} \\
t_i \in \{0,1\} \qquad i \in \{1,2,3,4,5\}
\end{cases}
$$

- It is a **mixed-integer linear programming** model: some variables are real and some are integer (in particular binary)

.mod file

```
set O := 1..5;       # temporal horizon
param Order{O} ; # commodity needed
param Cost{O};    # unit cost of commodity
param Trans{O}; # transportation cost
param Storage{O}; # storage cost per unit
param M;             # big value for the fixed cost constraints
var c{i in O} >= 0; # commodity bought in month i
var s{i in O} >= 0; # commodity stocked at month i
var t{i in O} binary; # 1 if we must pay transportation , 0 otherwise
minimize total_cost: sum{i in O} Cost[i] * c[i] +
sum{i in O} Trans[i] * t[i] +
sum{i in O} Storage[i] * s[i];
s.t. month_1: s[1] = c[1] - Order[1];
s.t. month_i{i in O: i>1}: s[i] = s[i-1] + c[i] – Order[i];
s.t. transportation{i in O}: M * t[i] >= c[i];
```

.dat file

```
param Order:=
1       12
2       4
3       18
4       17
5       7;

param Cost :=
1       8
2       12
3       7
4       11
5       10;
```

```
param Trans :=
1       10
2       10
3       10
4       10
5       10;

param Storage :=
1       5
2       5
3       5
4       5
5       5;

param M := 60;
```

Solution

OBJECTIVES:

costi = 571

VARIABLES:

c [*] :=

1	16
2	0
3	18
4	17
5	7;

s [*] :=

1	4
2	0
3	0
4	0
5	0;

t [*] :=

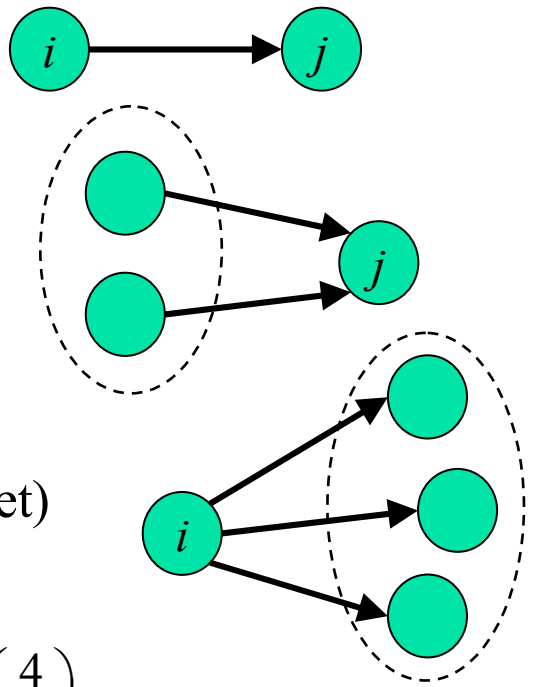
1	1
2	0
3	1
4	1
5	1;

- This means that the best solution is to **buy** at month 1, 3, 4, 5 and use the **storage** for month 2. Of course, this may change if storage and transportation costs vary

Combinatorial Example

- Consider the following model, with binary variables x_{ij} representing for example connections between couples of items i and j

$$x_{ij} = \begin{cases} 1 & \text{if connection activated} \\ 0 & \text{otherwise} \end{cases}$$



$\max \sum_i \sum_j x_{ij}$

$\sum_i x_{ij} \le b_1 \quad \forall j$ (number of connections to item j)

$\sum_j x_{ij} \le b_2 \quad \forall i$ (number of connections from item i)

$\sum_i \sum_j c_{ij} x_{ij} \le b_3$ (total cost less than or equal to a budget)

$x_{ij} \in \{0,1\}$

The numerical values are $c = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 1 & 3 \\ 3 & 1 & 4 & 2 \\ 4 & 3 & 2 & 1 \end{pmatrix}$ and $b = \begin{pmatrix} 4 \\ 3 \\ 10 \end{pmatrix}$

- Let's write it into AMPL in order to solve it numerically

File .mod version 1

```
param n;          # number of items
set O := 1..n;    # items
param c{O,O};     # cost
param b{1..3};    # right-hand side values
var x{O,O} >= 0, binary; # connections on/off

maximize obj: sum{i in O,j in O} x[i,j];
s.t. to{j in O}: sum{i in O} x[i,j] <= b[1];
s.t. from{i in O}: sum{j in O} x[i,j] <= b[2];
s.t. cost: sum{i in O,j in O} c[i,j]*x[i,j] <= b[3];
```

File .mod version 2

- If the connection between an item and itself cannot exist

```
param n;            # number of items
set O := 1..n;      # items
param c{O,O};       # cost
param b{1..3};      # right-hand side values
var x{i in O, j in O: i<>j} >= 0, binary; #connections on/off

maximize obj: sum{i in O,j in O: i<>j} x[i,j];
s.t. to{j in O}: sum{i in O: i<>j} x[i,j] <= b[1];
s.t. from{i in O}: sum{j in O: i<>j} x[i,j] <= b[2];
s.t. cost: sum{i in O,j in O: i<>j} c[i,j]*x[i,j] <= b[3];
```

File .mod version 3

- if the two connections ij and ji are mutually exclusive

```
param n;            # number of items
set O := 1..n;      # items
param c{O,O};       # cost
param b{1..3};      # right-hand side values
var x{i in O, j in O: i<>j} >= 0, binary; # connections on/off

maximize obj: sum{i in O,j in O: i<>j} x[i,j];
s.t. to{j in O}: sum{i in O: i<>j} x[i,j] <= b[1];
s.t. from{i in O}: sum{j in O: i<>j} x[i,j] <= b[2];
s.t. cost: sum{i in O,j in O: i<>j} c[i,j]*x[i,j] <= b[3];
s.t. exclusion{i in O, j in O: i<j}: x[i,j] + x[j,i] <= 1;
```

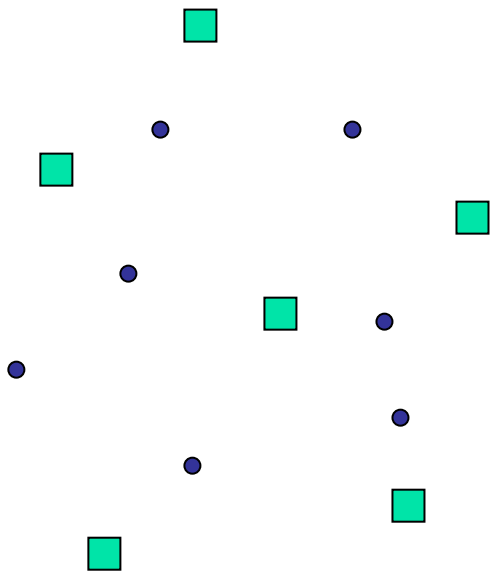
File .dat for all versions

```
param n := 4;

param b :=
      1  4
      2  3
      3 10;

param c:  1  2  3  4 :=
      1     1  2  3  4
      2     2  4  1  3
      3     3  1  4  2
      4     4  3  2  1;
```

- The data may remain the same in the 3 versions of the model

Advanced example: Plant Location

- We have a **Plant Location** problem:
- a set I of m **costumers** must be connected to plants (blue dots)
- and a set J of n sites where we can activate those **plants** (green squares)



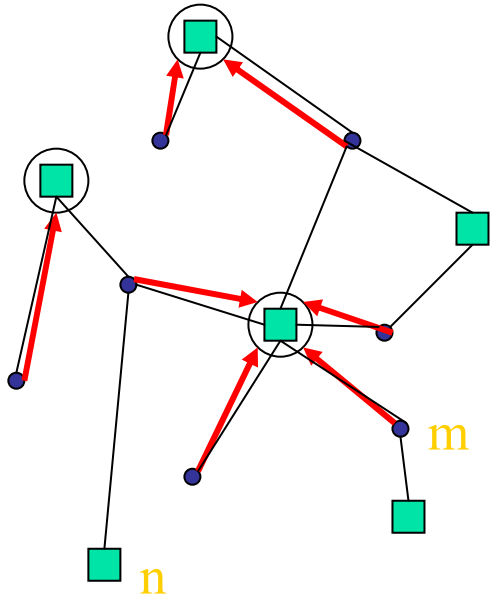
- Plant j costs f_j
- Connection ij costs c_{ij}
- We want to serve all customers and minimize the **total cost**
- This problem represents **many important practical problems:** power plants, hospitals, stores, etc.

$$\text{- } Plant\ j\ active\ or\ not \quad \Rightarrow x_j \in \{1, 0\}$$

$$\text{- } Customer\ i\ assigned\ to\ plant\ j \quad \Rightarrow y_{ij} \in \{1, 0\}$$

Constraints

- A solution is **feasible** only if it satisfies two types of constraints (look at the figure for an example)



- *A customer can only be assigned to an active plant* (activation constraints)

$$y_{ij} \leq x_j \qquad i \in I \qquad j \in J$$

- *Every customer must be assigned to one plant* (service constraints)

$$\sum_{j \in J} y_{ij} = 1 \qquad i \in I$$

- So we can now write the model of the problem

Plant Location (binary version)

$$\min \sum_{j \in J} f_j x_j + \sum_{j \in J} \sum_{i \in I} c_{ij} y_{ij}$$

$$\sum_{j \in J} y_{ij} = 1 \qquad i \in I$$

$$x_j - y_{ij} \geq 0 \qquad i \in I \quad j \in J$$

$$y_{ij} \in \{0,1\} \quad x_j \in \{0,1\} \quad i \in I \quad j \in J$$

- This model is usually very large: if we have 100 plants and 1000 costumers (medium size for a practical case) we obtain:
- 100 + 100,000 variables!
- 1000 + 100,000 constraints!
- So, we can solve the binary version only when the problem is small. When it is large, we need to consider its **linear relaxation** (= the variables are real, not binary) which is always much easier to solve

Strong formulation of Plant Location

$$\min \sum_{j \in J} f_j x_j + \sum_{j \in J} \sum_{i \in I} c_{ij} y_{ij}$$

$$\sum_{j \in J} y_{ij} = 1 \qquad i \in I$$

$$x_j - y_{ij} \geq 0 \qquad i \in I \quad j \in J$$

$$y_{ij} \geq 0 \quad x_j \geq 0 \quad i \in I \quad j \in J$$

- In the **linear relaxation** the variables are real, ≥ 0 and ≤ 1 (box constraints) however we can omit ≤ 1 because, if you look at the objective, the larger the variables the higher the cost, so variables will never be larger than what is strictly necessary to satisfy constraints
- This version is called Strong Formulation of the problem

pl.mod in AMPL

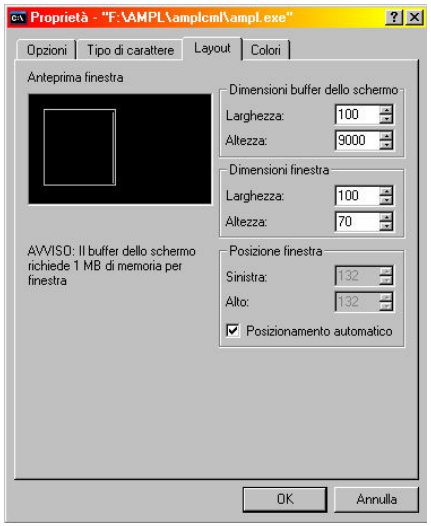
```
param m:=  6;   # number of customers
param n:=  5;    # number of plants

set I := 1..m;  # customers
set J := 1..n;  # plants

param c{I,J};   # connection costs
param f{J};     # activation costs

var y{I,J} >=0; # or binary if the problem is small
var x{J} >=0;   # or binary if the problem is small

minimize cost: sum{j in J} (f[j]*x[j] + sum{i in I} c[i,j]*y[i,j]);

s.t. service{i in I}: sum{j in J} y[i,j] = 1;

s.t. activation{i in I, j in J}: x[j] - y[i,j] >= 0;
```

pl.dat in AMPL

```
param:  f :=
      1   4
      2   3
      3   1
      4   4
      5   7;

param c:   1   2   3   4   5 :=
        1  12  13   6   0   1
        2   8   4   9   1   2
        3   2   6   6   0   1
        4   3   5   2  10   8
        5   8   0   5  10   8
        6   2   0   3   4   1;
```

Solving…

- Now, we can just type all the commands in AMPL terminal (**option solver cplex;** etc)
- However, we could save time by using a script: we write all those commands in a **.run** file (for example pl.run) and then we simply use that file
- Doing many experiments becomes much **faster**!
- We launch it by writing in the terminal **include pl.run;**
- Also, it's better to set, in the **properties** of AMPL window, many more rows and columns, especially for the rows of the screen buffer…



pl.run

```
option solver cplex;           # select the solver
#option omit_zero_rows 1; # display does not print variables at 0
#option solver_msg 0;        # no solver messages

model plant_location.mod;  # reads model
data plant_location.dat;       # reads data

solve;

printf"\nValues of the variables:\n"; display x, y;
printf“Total activation cost: %f \n", sum{j in J} f[j]*x[j];
printf“Total connection cost: %f \n", sum{i in I, j in J} c[i,j]*y[i,j];
```

Solution of the linear relaxation

CPLEX 11.2.0: optimal solution; objective 11
9 dual simplex iterations (0 in phase I)

The variables are:

```
x [*] :=
1  0
2  1
3  1
4  1
5  0;

y [*,*]
:   1   2   3   4   5   :=
1   0   0   0   1   0
2   0   0   0   1   0
3   0   0   0   1   0
4   0   0   1   0   0
5   0   1   0   0   0
6   0   1   0   0   0;
```

Total cost is: 11 (in this case we were lucky and we had a binary solution: it is the solution to the original binary problem. Otherwise, we have an approximation, which can be useful to estimate the cost for example)

How is the model of this example?

Min 12 y(1,1) + 13 y(1,2) + 6 y(1,3) + y(1,5) + 8 y(2,1) + 4 y(2,2)
+ 9 y(2,3) + y(2,4) + 2 y(2,5) + 2 y(3,1) + 6 y(3,2) + 6 y(3,3) + y(3,5)
+ 3 y(4,1) + 5 y(4,2) + 2 y(4,3) + 10 y(4,4) + 8 y(4,5) + 8 y(5,1)
+ 5 y(5,3) + 10 y(5,4) + 8 y(5,5) + 2 y(6,1) + 3 y(6,3) + 4 y(6,4)
+ y(6,5) + 4 x(1) + 3 x(2) + x(3) + 4 x(4) + 7 x(5)

Subject To
service(1): y(1,1) + y(1,2) + y(1,3) + y(1,4) + y(1,5) = 1
service(2): y(2,1) + y(2,2) + y(2,3) + y(2,4) + y(2,5) = 1
service(3): y(3,1) + y(3,2) + y(3,3) + y(3,4) + y(3,5) = 1
service(4): y(4,1) + y(4,2) + y(4,3) + y(4,4) + y(4,5) = 1
service(5): y(5,1) + y(5,2) + y(5,3) + y(5,4) + y(5,5) = 1
service(6): y(6,1) + y(6,2) + y(6,3) + y(6,4) + y(6,5) = 1

activat(01): x(1) – y(1,1) >= 0
activat(02): x(1) – y(2,1) >= 0
activat(03): x(1) – y(3,1) >= 0
activat(04): x(1) – y(4,1) >= 0
activat(05): x(1) – y(5,1) >= 0
activat(06): x(1) – y(6,1) >= 0

[to be continued]

How is the model of this example?

[continued]

activat(07): x(2) – y(1,2) >= 0
activat(08): x(2) – y(2,2) >= 0
activat(09): x(2) – y(3,2) >= 0
activat(10): x(2) – y(4,2) >= 0
activat(11): x(2) – y(5,2) >= 0
activat(12): x(2) – y(6,2) >= 0

activat(13): x(3) – y(1,3) >= 0
activat(14): x(3) – y(2,3) >= 0
activat(15): x(3) – y(3,3) >= 0
activat(16): x(3) – y(4,3) >= 0
activat(17): x(3) – y(5,3) >= 0
activat(18): x(3) – y(6,3) >= 0

activat(19): x(4) – y(1,4) >= 0
activat(20): x(4) – y(2,4) >= 0
activat(21): x(4) – y(3,4) >= 0
activat(22): x(4) – y(4,4) >= 0
activat(23): x(4) – y(5,4) >= 0
activat(24): x(4) – y(6,4) >= 0

activat(25): x(5) – y(1,5) >= 0
activat(26): x(5) – y(2,5) >= 0
activat(27): x(5) – y(3,5) >= 0
activat(28): x(5) – y(4,5) >= 0
activat(29): x(5) – y(5,5) >= 0
activat(30): x(5) – y(6,5) >= 0

Constraint generation

- We managed to solve this model, but we can observe that, from a very **small** practical problem, we obtained a **large** model
- Unfortunately, this happens quite often: many linear programming problems tend to have many constraints, because the real problems actually contain a lot of implicit conditions that must be respected
- So, we will now see a technique to solve linear programming problems with a huge number of constraints
- It is called **Constraint generation**: we do not write all the constraint and we add only those which are needed!

1. *Take only a small set of constraint from the set of the original ones*
2. *Solve and find a solution x'*
3. *Search for an original constraint violated by x'*
4. *If we find this constraint, add it and solve again*
5. *Otherwise, x' is the solution to the original problem!*

pld.mod (constraint generation)

```
param m:=  6;   # number of customers
param n:=  5;    # number of plants

set I := 1..m;  # customers
set J := 1..n;  # plants

param c{I,J};   # connection costs
param f{J};     # activation costs

var y{I,J} >=0;
var x{J} >=0;

minimize cost: sum{j in J} (f[j]*x[j] + sum{i in I} c[i,j]*y[i,j]);

s.t. service{i in I}: sum{j in J} y[i,j] = 1;

# no activation constraints !
```

pld.run 1/2

```
option solver cplex;
model plant_location.mod;  # reads model
data plant_location.dat;   # reads data

param found;   # whether we have the violated constraint or not
let found:=0;

option cplex_auxfiles 'rc';   # to have in cplex the AMPL names
option cplex_options 'writeprob solvedmodel.lp'; # writes current model

solve;

printf"\n\Values of the variables:\n"; display x, y;
```

[to be continued]…

pld.run 2/2

…[continued]

```
for{j in J, i in I}       # searches for a violated activation constraint
{
  if(x[j] < y[i,j]) then  # if this is the violated constraint
  {
    printf"\nThe violated constraint is: x[%d] >= y[%d,%d] ", j,i,j;
    printf"\nTo add it write\n name: x[%d] >= y[%d,%d];
                                                commands go.run; \n", j,i,j;
    let found := 1;
    break;
  }
}
if(found == 0) then printf"\nAll constraints are satisfied \n";
```

go.run (repeated each time)

```
let found:=0;
option cplexamp_auxfiles 'rc';
option cplex_options 'writeprob solvedmodel.lp';
solve;
printf"\n\Values of the variables:\n"; display x, y;
for{j in J, i in I}
{
  if(x[j] < y[i,j]) then
  {
     printf"\nThe violated constraint is: x[%d] >= y[%d,%d] ", j,i,j;
     printf"\nTo add it write name: x[%d] >= y[%d,%d];
                                                    commands go.run; \n", j,i,j;
                let found := 1;
                break;
        }
}
if(found == 0) then printf"\All constraints are satisfied \n";
```

Solvedmodel 1

Minimize

obj: 12 y(1,1) + 13 y(1,2) + 6 y(1,3) + y(1,5) + 8 y(2,1) + 4 y(2,2)
+ 9 y(2,3) + y(2,4) + 2 y(2,5) + 2 y(3,1) + 6 y(3,2) + 6 y(3,3) + y(3,5)
+ 3 y(4,1) + 5 y(4,2) + 2 y(4,3) + 10 y(4,4) + 8 y(4,5) + 8 y(5,1)
+ 5 y(5,3) + 10 y(5,4) + 8 y(5,5) + 2 y(6,1) + 3 y(6,3) + 4 y(6,4)
+ y(6,5) + 4 x(1) + 3 x(2) + x(3) + 4 x(4) + 7 x(5)

Subject To

service(1): y(1,1) + y(1,2) + y(1,3) + y(1,4) + y(1,5) = 1
service(2): y(2,1) + y(2,2) + y(2,3) + y(2,4) + y(2,5) = 1
service(3): y(3,1) + y(3,2) + y(3,3) + y(3,4) + y(3,5) = 1
service(4): y(4,1) + y(4,2) + y(4,3) + y(4,4) + y(4,5) = 1
service(5): y(5,1) + y(5,2) + y(5,3) + y(5,4) + y(5,5) = 1
service(6): y(6,1) + y(6,2) + y(6,3) + y(6,4) + y(6,5) = 1

Solution 1

CPLEX 11.2.0: optimal solution; objective 3

0 dual simplex iterations (0 in phase I)

Values of the variables:

x [*] := 0 0 0 0 0 ;

y [*,*]

:	1	2	3	4	5	:=
1	0	0	0	1	0	
2	0	0	0	1	0	
3	0	0	0	1	0	
4	0	0	1	0	0	
5	0	1	0	0	0	
6	0	1	0	0	0;	

The violated constraint is: x[2] >= y[5,2]

To add it write name: x[2] >= y[5,2]; commands go.run;

ampl: one: x[2] >= y[5,2]; commands go.run;

Solvedmodel 2

Minimize
obj: 12 y(1,1) + 13 y(1,2) + 6 y(1,3) + y(1,5) + 8 y(2,1) + 4 y(2,2)
+ 9 y(2,3) + y(2,4) + 2 y(2,5) + 2 y(3,1) + 6 y(3,2) + 6 y(3,3) + y(3,5)
+ 3 y(4,1) + 5 y(4,2) + 2 y(4,3) + 10 y(4,4) + 8 y(4,5) + 8 y(5,1)
+ 5 y(5,3) + 10 y(5,4) + 8 y(5,5) + 2 y(6,1) + 3 y(6,3) + 4 y(6,4)
+ y(6,5) + 4 x(1) + 3 x(2) + x(3) + 4 x(4) + 7 x(5)
Subject To
service(1): y(1,1) + y(1,2) + y(1,3) + y(1,4) + y(1,5) = 1
service(2): y(2,1) + y(2,2) + y(2,3) + y(2,4) + y(2,5) = 1
service(3): y(3,1) + y(3,2) + y(3,3) + y(3,4) + y(3,5) = 1
service(4): y(4,1) + y(4,2) + y(4,3) + y(4,4) + y(4,5) = 1
service(5): y(5,1) + y(5,2) + y(5,3) + y(5,4) + y(5,5) = 1
service(6): y(6,1) + y(6,2) + y(6,3) + y(6,4) + y(6,5) = 1
one: - y(5,2) + x(2) >= 0

Solution 2

CPLEX 11.2.0: optimal solution; objective 6

1 dual simplex iterations (0 in phase I)

Values of the variables:

x [*] := 0 1 0 0 0 ;

y [*,*]

:	1	2	3	4	5	:=
1	0	0	0	1	0	
2	0	0	0	1	0	
3	0	0	0	1	0	
4	0	0	1	0	0	
5	0	1	0	0	0	
6	0	1	0	0	0;	

The violated constraint is: x[3] >= y[4,3]

To add it write name: x[3] >= y[4,3]; commands go.run;

ampl: two: x[3] >= y[4,3]; commands go.run;

Solvedmodel 3

Minimize

obj: 12 y(1,1) + 13 y(1,2) + 6 y(1,3) + y(1,5) + 8 y(2,1) + 4 y(2,2)
+ 9 y(2,3) + y(2,4) + 2 y(2,5) + 2 y(3,1) + 6 y(3,2) + 6 y(3,3) + y(3,5)
+ 3 y(4,1) + 5 y(4,2) + 2 y(4,3) + 10 y(4,4) + 8 y(4,5) + 8 y(5,1)
+ 5 y(5,3) + 10 y(5,4) + 8 y(5,5) + 2 y(6,1) + 3 y(6,3) + 4 y(6,4)
+ y(6,5) + 4 x(1) + 3 x(2) + x(3) + 4 x(4) + 7 x(5)

Subject To

service(1): y(1,1) + y(1,2) + y(1,3) + y(1,4) + y(1,5) = 1
service(2): y(2,1) + y(2,2) + y(2,3) + y(2,4) + y(2,5) = 1
service(3): y(3,1) + y(3,2) + y(3,3) + y(3,4) + y(3,5) = 1
service(4): y(4,1) + y(4,2) + y(4,3) + y(4,4) + y(4,5) = 1
service(5): y(5,1) + y(5,2) + y(5,3) + y(5,4) + y(5,5) = 1
service(6): y(6,1) + y(6,2) + y(6,3) + y(6,4) + y(6,5) = 1
one: - y(5,2) + x(2) >= 0
two: - y(4,3) + x(3) >= 0

Solution 3

CPLEX 11.2.0: optimal solution; objective 7

1 dual simplex iterations (0 in phase I)

Values of the variables:

x [*] := 0 1 0 0 0 ;

y [*,*]

:	1	2	3	4	5	:=
1	0	0	0	1	0	
2	0	0	0	1	0	
3	0	0	0	1	0	
4	1	0	0	0	0	
5	0	1	0	0	0	
6	0	1	0	0	0;	

The violated constraint is: x[1] >= y[4,1]

To add it write name: x[1] >= y[4,1]; commands go.run;

ampl: three: x[1] >= y[4,1]; commands go.run;

Solvedmodel 4

Minimize
obj: 12 y(1,1) + 13 y(1,2) + 6 y(1,3) + y(1,5) + 8 y(2,1) + 4 y(2,2)
+ 9 y(2,3) + y(2,4) + 2 y(2,5) + 2 y(3,1) + 6 y(3,2) + 6 y(3,3) + y(3,5)
+ 3 y(4,1) + 5 y(4,2) + 2 y(4,3) + 10 y(4,4) + 8 y(4,5) + 8 y(5,1)
+ 5 y(5,3) + 10 y(5,4) + 8 y(5,5) + 2 y(6,1) + 3 y(6,3) + 4 y(6,4)
+ y(6,5) + 4 x(1) + 3 x(2) + x(3) + 4 x(4) + 7 x(5)
Subject To
service(1): y(1,1) + y(1,2) + y(1,3) + y(1,4) + y(1,5) = 1
service(2): y(2,1) + y(2,2) + y(2,3) + y(2,4) + y(2,5) = 1
service(3): y(3,1) + y(3,2) + y(3,3) + y(3,4) + y(3,5) = 1
service(4): y(4,1) + y(4,2) + y(4,3) + y(4,4) + y(4,5) = 1
service(5): y(5,1) + y(5,2) + y(5,3) + y(5,4) + y(5,5) = 1
service(6): y(6,1) + y(6,2) + y(6,3) + y(6,4) + y(6,5) = 1
one: - y(5,2) + x(2) >= 0
two: - y(4,3) + x(3) >= 0
three: - y(4,1) + x(1) >= 0

Solution 4

CPLEX 11.2.0: optimal solution; objective 7

1 dual simplex iterations (0 in phase I)

Values of the variables:

x [*] := 0 1 1 0 0 ;

y [*,*]

:	1	2	3	4	5	:=
1	0	0	0	1	0	
2	0	0	0	1	0	
3	0	0	0	1	0	
4	0	0	1	0	0	
5	0	1	0	0	0	
6	0	1	0	0	0;	

The violated constraint is: x[4] >= y[1,4]

To add it write name: x[4] >= y[1,4]; commands go.run;

ampl: four: x[4] >= y[1,4]; commands go.run;

Solvedmodel 5

Minimize

obj: 12 y(1,1) + 13 y(1,2) + 6 y(1,3) + y(1,5) + 8 y(2,1) + 4 y(2,2)
+ 9 y(2,3) + y(2,4) + 2 y(2,5) + 2 y(3,1) + 6 y(3,2) + 6 y(3,3) + y(3,5)
+ 3 y(4,1) + 5 y(4,2) + 2 y(4,3) + 10 y(4,4) + 8 y(4,5) + 8 y(5,1)
+ 5 y(5,3) + 10 y(5,4) + 8 y(5,5) + 2 y(6,1) + 3 y(6,3) + 4 y(6,4)
+ y(6,5) + 4 x(1) + 3 x(2) + x(3) + 4 x(4) + 7 x(5)

Subject To

service(1): y(1,1) + y(1,2) + y(1,3) + y(1,4) + y(1,5) = 1
service(2): y(2,1) + y(2,2) + y(2,3) + y(2,4) + y(2,5) = 1
service(3): y(3,1) + y(3,2) + y(3,3) + y(3,4) + y(3,5) = 1
service(4): y(4,1) + y(4,2) + y(4,3) + y(4,4) + y(4,5) = 1
service(5): y(5,1) + y(5,2) + y(5,3) + y(5,4) + y(5,5) = 1
service(6): y(6,1) + y(6,2) + y(6,3) + y(6,4) + y(6,5) = 1
one: - y(5,2) + x(2) >= 0
two: - y(4,3) + x(3) >= 0
three: - y(4,1) + x(1) >= 0
four: - y(1,4) + x(4) >= 0

Solution 5

CPLEX 11.2.0: optimal solution; objective 8

1 dual simplex iterations (0 in phase I)

Values of the variables:

x [*] := 0 1 1 0 0 ;

y [*,*]

:	1	2	3	4	5	:=
1	0	0	0	0	1	
2	0	0	0	1	0	
3	0	0	0	1	0	
4	0	0	1	0	0	
5	0	1	0	0	0	
6	0	1	0	0	0;	

The violated constraint is: x[4] >= y[2,4]

To add it write name: x[4] >= y[2,4]; commands go.run;

ampl: five: x[4] >= y[2,4]; commands go.run;

Solvedmodel 6

Minimize

obj: 12 y(1,1) + 13 y(1,2) + 6 y(1,3) + y(1,5) + 8 y(2,1) + 4 y(2,2)
+ 9 y(2,3) + y(2,4) + 2 y(2,5) + 2 y(3,1) + 6 y(3,2) + 6 y(3,3) + y(3,5)
+ 3 y(4,1) + 5 y(4,2) + 2 y(4,3) + 10 y(4,4) + 8 y(4,5) + 8 y(5,1)
+ 5 y(5,3) + 10 y(5,4) + 8 y(5,5) + 2 y(6,1) + 3 y(6,3) + 4 y(6,4)
+ y(6,5) + 4 x(1) + 3 x(2) + x(3) + 4 x(4) + 7 x(5)

Subject To

service(1): y(1,1) + y(1,2) + y(1,3) + y(1,4) + y(1,5) = 1
service(2): y(2,1) + y(2,2) + y(2,3) + y(2,4) + y(2,5) = 1
service(3): y(3,1) + y(3,2) + y(3,3) + y(3,4) + y(3,5) = 1
service(4): y(4,1) + y(4,2) + y(4,3) + y(4,4) + y(4,5) = 1
service(5): y(5,1) + y(5,2) + y(5,3) + y(5,4) + y(5,5) = 1
service(6): y(6,1) + y(6,2) + y(6,3) + y(6,4) + y(6,5) = 1
one: - y(5,2) + x(2) >= 0
two: - y(4,3) + x(3) >= 0
three: - y(4,1) + x(1) >= 0
four: - y(1,4) + x(4) >= 0
five: - y(2,4) + x(4) >= 0

Solution 6

CPLEX 11.2.0: optimal solution; objective 9

0 dual simplex iterations (0 in phase I)

Values of the variables:

x [*] := 0 1 1 0 0 ;

y [*,*]

:	1	2	3	4	5	:=
1	0	0	0	0	1	
2	0	0	0	0	1	
3	0	0	0	1	0	
4	0	0	1	0	0	
5	0	1	0	0	0	
6	0	1	0	0	0;	

The violated constraint is: x[4] >= y[3,4]

To add it write name: x[4] >= y[3,4]; commands go.run;

ampl: six: x[4] >= y[3,4]; commands go.run;

Solvedmodel 7

Minimize
obj: 12 y(1,1) + 13 y(1,2) + 6 y(1,3) + y(1,5) + 8 y(2,1) + 4 y(2,2)
+ 9 y(2,3) + y(2,4) + 2 y(2,5) + 2 y(3,1) + 6 y(3,2) + 6 y(3,3) + y(3,5)
+ 3 y(4,1) + 5 y(4,2) + 2 y(4,3) + 10 y(4,4) + 8 y(4,5) + 8 y(5,1)
+ 5 y(5,3) + 10 y(5,4) + 8 y(5,5) + 2 y(6,1) + 3 y(6,3) + 4 y(6,4)
+ y(6,5) + 4 x(1) + 3 x(2) + x(3) + 4 x(4) + 7 x(5)
Subject To
service(1): y(1,1) + y(1,2) + y(1,3) + y(1,4) + y(1,5) = 1
service(2): y(2,1) + y(2,2) + y(2,3) + y(2,4) + y(2,5) = 1
service(3): y(3,1) + y(3,2) + y(3,3) + y(3,4) + y(3,5) = 1
service(4): y(4,1) + y(4,2) + y(4,3) + y(4,4) + y(4,5) = 1
service(5): y(5,1) + y(5,2) + y(5,3) + y(5,4) + y(5,5) = 1
service(6): y(6,1) + y(6,2) + y(6,3) + y(6,4) + y(6,5) = 1
one: - y(5,2) + x(2) >= 0
two: - y(4,3) + x(3) >= 0
three: - y(4,1) + x(1) >= 0
four: - y(1,4) + x(4) >= 0
five: - y(2,4) + x(4) >= 0
six: - y(3,4) + x(4) >= 0

Solution 7

CPLEX 11.2.0: optimal solution; objective 10

1 dual simplex iterations (0 in phase I)

Values of the variables:

x [*] := 0 1 1 0 0 ;

y [*,*]

:	1	2	3	4	5	:=
1	0	0	0	0	1	
2	0	0	0	0	1	
3	0	0	0	0	1	
4	0	0	1	0	0	
5	0	1	0	0	0	
6	0	1	0	0	0;	

The violated constraint is: x[5] >= y[1,5]

To add it write name: x[5] >= y[1,5]; commands go.run;

ampl: seven: x[5] >= y[1,5]; commands go.run;

Solvedmodel 8

Minimize

obj: 12 y(1,1) + 13 y(1,2) + 6 y(1,3) + y(1,5) + 8 y(2,1) + 4 y(2,2)
+ 9 y(2,3) + y(2,4) + 2 y(2,5) + 2 y(3,1) + 6 y(3,2) + 6 y(3,3) + y(3,5)
+ 3 y(4,1) + 5 y(4,2) + 2 y(4,3) + 10 y(4,4) + 8 y(4,5) + 8 y(5,1)
+ 5 y(5,3) + 10 y(5,4) + 8 y(5,5) + 2 y(6,1) + 3 y(6,3) + 4 y(6,4)
+ y(6,5) + 4 x(1) + 3 x(2) + x(3) + 4 x(4) + 7 x(5)

Subject To

service(1): y(1,1) + y(1,2) + y(1,3) + y(1,4) + y(1,5) = 1
service(2): y(2,1) + y(2,2) + y(2,3) + y(2,4) + y(2,5) = 1
service(3): y(3,1) + y(3,2) + y(3,3) + y(3,4) + y(3,5) = 1
service(4): y(4,1) + y(4,2) + y(4,3) + y(4,4) + y(4,5) = 1
service(5): y(5,1) + y(5,2) + y(5,3) + y(5,4) + y(5,5) = 1
service(6): y(6,1) + y(6,2) + y(6,3) + y(6,4) + y(6,5) = 1
one: - y(5,2) + x(2) >= 0
two: - y(4,3) + x(3) >= 0
three: - y(4,1) + x(1) >= 0
four: - y(1,4) + x(4) >= 0
five: - y(2,4) + x(4) >= 0
six: - y(3,4) + x(4) >= 0
seven: - y(1,5) + x(5) >= 0

Solution 8 (final)

CPLEX 11.2.0: optimal solution; objective 11

3 dual simplex iterations (0 in phase I)

Values of the variables:

x [*] := 0 1 1 1 0 ;

y [*,*]

:	1	2	3	4	5	:=
1	0	0	0	1	0	
2	0	0	0	1	0	
3	0	0	0	1	0	
4	0	0	1	0	0	
5	0	1	0	0	0	
6	0	1	0	0	0;	

All constraints are satisfied

Optimal solution found after adding 7 activation constraints instead of 6x5=30

For very large problems it makes the difference between solving and not solving

Other options to provide Data

- Instead of using a **.dat** file, we can read a **.txt** file

```
read{j in J} f[j] < activation_costs.txt;
read{i in I, j in J} c[i,j] < connection_costs.txt;
```

- Or we can **assign** them with "let" command

```
for{j in J}
{
let f[j] := 3+j;
}
```

- Or we can generate **random numbers**

```
option randseed 0;    # set random seed
param mean := 4;      # mean of the random number
param variance := 1.5;  # variance of the random number
printf"\nf [*] :=\n";     # we print the random vector f
for{j in J}
{
let f[j] := Normal (mean, variance); # generate a random value
printf" %f\n", f[j];  # print it
}
```

.mod Automatic

```
param m:= 16; # number of customers
param n:= 16; # number of plants
set I := 1..m; # customers
set J := 1..n; # plants

param c{I,J}; # connection costs
param f{J}; # activation costs
read {i in I,j in J} c[i,j] < matrix.txt;
read {j in J} f[j] < vector.txt;

param k, default 0; # number of generated constraints
param A{I,J}, default 0; # indicator of generated constraints

var y{I,J} >=0; # connection variables
var x{J} >=0; # activation variables

minimize cost: sum{j in J} (f[j]*x[j] + sum{i in I} c[i,j]*y[i,j]);
s.t. service{i in I}: sum{j in J} y[i,j] = 1;
s.t. activation{i in I, j in J: A[i,j] > 0}: x[j] - y[i,j] >= 0;
# matrix A tells which constraints have been generated
```

.run Automatic

```
option solver cplex;
model pld.mod;   # reads model
param found;    # whether we have the violated constraint or not
param epsilon := 0.001;  # numerical tolerance

option cplex_auxfiles 'rc'; # to have in cplex the AMPL names
option cplex_options 'writeprob solvedmodel.lp'; # writes current model
solve;
printf"\n\Values of the variables:\n"; display x, y;

repeat
{
        let found:=0;
        for{i in I, j in J} # searches for a violated activation constraint
        {
                if( x[j] < y[i,j] - epsilon ) then  # if this is the violated constraint
                {
                        printf"\nThe violated constraint is: x[%d] >= y[%d,%d] \n", j,i,j;
                        let found:=1;
                        let k := k + 1;  # counter for generated constraints
                        let A[i,j] := 1; # marks the generated constraint
                        break;
                }
        }
        if (found == 1) then
        {
                option cplex_auxfiles 'rc'; # to have in cplex the AMPL names
                option cplex_options 'writeprob solvedmodel.lp'; # writes current model
                solve;
                printf"\n\Values of the variables:\n"; display x, y;
        }
}
while (found == 1);
printf"\nAll service constraints are satisfied after adding %d of %d\n", k, m*n;
```