Optimisation Models: exercises

Dipartimento di Ingegneria e Architettura
Università degli studi di Trieste

30/4/2021, Trieste

Exercise 1: Economic lot sizing problem (ELS)

- production planning of 4 products over 7 time periods

Exercise 1: Economic lot sizing problem (ELS)

- production planning of 4 products over 7 time periods
- in every period, a given demand (tons) for every product must be satisfied by the **production in this period** and **by inventory carried over from previous periods**

		Time Periods						
	Prod	1	2	3	4	5	6	7
Demand	1	2	3	5	3	4	2	5
	2	3	1	2	3	5	3	1
	3	3	5	2	1	2	1	3
	4	2	2	1	3	2	1	2
Cost (k€)	1	5	3	2	1	3	1	4
	2	1	4	2	3	1	3	1
	3	3	3	3	4	4	3	3
	4	2	2	2	3	3	3	4
Set-up		17	14	11	6	9	6	15

Exercise 1: Economic lot sizing problem (ELS)

- production planning of 4 products over 7 time periods
- in every period, a given demand (tons) for every product must be satisfied by the **production in this period** and **by inventory carried over from previous periods**

- unit production cost per product and time period (**no inventory cost**)

		Time Periods						
	Prod	1	2	3	4	5	6	7
Demand	1	2	3	5	3	4	2	5
	2	3	1	2	3	5	3	1
	3	3	5	2	1	2	1	3
	4	2	2	1	3	2	1	2
Cost (k€)	1	5	3	2	1	3	1	4
	2	1	4	2	3	1	3	1
	3	3	3	3	4	4	3	3
	4	2	2	2	3	3	3	4
Set-up		17	14	11	6	9	6	15

Exercise 1: Economic lot sizing problem (ELS)

- production planning of 4 products over 7 time periods
- in every period, a given demand (tons) for every product must be satisfied by the **production in this period** and **by inventory carried over from previous periods**

- unit production cost per product and time period (**no inventory cost**)
- set-up cost (k€) associated **each product** and each period

	Prod	Time Periods 1	2	3	4	5	6	7
Demand	1	2	3	5	3	4	2	5
	2	3	1	2	3	5	3	1
	3	3	5	2	1	2	1	3
	4	2	2	1	3	2	1	2
Cost (k€)	1	5	3	2	1	3	1	4
	2	1	4	2	3	1	3	1
	3	3	3	3	4	4	3	3
	4	2	2	2	3	3	3	4
Set-up		17	14	11	6	9	6	15

Exercise 1: Economic lot sizing problem (ELS)

- production planning of 4 products over 7 time periods
- in every period, a given demand (tons) for every product must be satisfied by the **production in this period** and **by inventory carried over from previous periods**

- unit production cost per product and time period (**no inventory cost**)
- set-up cost (k€) associated **each product** and each period
- limited total production capacity (**12 tons per period**)

		Time Periods						
	Prod	1	2	3	4	5	6	7
Demand	1	2	3	5	3	4	2	5
	2	3	1	2	3	5	3	1
	3	3	5	2	1	2	1	3
	4	2	2	1	3	2	1	2
Cost (k€)	1	5	3	2	1	3	1	4
	2	1	4	2	3	1	3	1
	3	3	3	3	4	4	3	3
	4	2	2	2	3	3	3	4
Set-up		17	14	11	6	9	6	15

Exercise 1: Economic lot sizing problem (ELS)

- production planning of 4 products over 7 time periods
- in every period, a given demand (tons) for every product must be satisfied by the **production in this period** and **by inventory carried over from previous periods**

- unit production cost per product and time period (**no inventory cost**)
- set-up cost (k€) associated **each product** and each period
- limited total production capacity (**12 tons per period**)
- minimise the total cost

		Time Periods						
	Prod	1	2	3	4	5	6	7
Demand	1	2	3	5	3	4	2	5
	2	3	1	2	3	5	3	1
	3	3	5	2	1	2	1	3
	4	2	2	1	3	2	1	2
Cost (k€)	1	5	3	2	1	3	1	4
	2	1	4	2	3	1	3	1
	3	3	3	3	4	4	3	3
	4	2	2	2	3	3	3	4
Set-up		17	14	11	6	9	6	15

Notation

Notation

$\mathcal{P}$: set of products

$\mathcal{T}$: set of time periods $= 1, \ldots, NT$

$DEM_{p,t}$: demand for every product p in period t

$PC_{p,t}$: unit production cost of product p in period t

SC_t : set-up up cost associated with production in period t

CAP_t : total production capacity in period t

Decision variables

Decision variables

$x_{p,t}$: amount of product p made in period t

Decision variables

$$x_{p,t}: \text{ amount of product } p \text{ made in period } t$$

$$y_{p,t} = \begin{cases} 1 & \text{if a setup takes places for} \\ & \quad \text{product } p \text{ in period } t \\ 0 & \text{otherwise} \end{cases} \qquad \forall p \in \mathcal{P}, t \in \mathcal{T}$$

Objective function

Objective function

$$min \sum_{p \in \mathcal{P}, t \in \mathcal{T}} (SC_t \cdot y_{p,t} + PC_{p,t} \cdot x_{p,t})$$

Objective function

$$min \sum_{p \in \mathcal{P}, t \in \mathcal{T}} (SC_t \cdot y_{p,t} + PC_{p,t} \cdot x_{p,t})$$

```
! Objective: minimize total cost
MinCost:= sum(p in P, t in T)
           (SC(t) * y(p,t) + PC(p,t) * x(p,t))

minimize(MinCost)
```

Decision variables constraints

Decision variables constraints

$$x_{p,t} \leq \mathbf{D_{p,t,NT}} \cdot y_{p,t} \qquad \forall p \in \mathcal{P}, t \in \mathcal{T}$$

Decision variables constraints

$$x_{p,t} \leq \mathbf{D_{p,t,NT}} \cdot y_{p,t} \qquad \forall p \in \mathcal{P}, t \in \mathcal{T}$$

D_{p,t_1,t_2} : total demand of product p in periods t_1 to t_2

Decision variables constraints

$$x_{p,t} \leq \mathbf{D_{p,t,NT}} \cdot y_{p,t} \qquad \forall p \in \mathcal{P}, t \in \mathcal{T}$$

D_{p,t_1,t_2} : total demand of product p in periods t_1 to t_2

```
 forall(p in P,s,t in T) D(p,s,t):= sum(k in s..t) DEM(p,k)

! If there is production during t then there is a setup in t
 forall(p in P, t in T)
  x(p,t) <= D(p,t,NT) * y(p,t)
```

Constraints

Constraints

$$\sum_{s=1}^{t} x_{p,s} \geq \mathbf{D_{p,1,t}} \qquad \forall p \in \mathcal{P}, t \in \mathcal{T}$$

Constraints

$$\sum_{s=1}^{t} x_{p,s} \geq \mathbf{D_{p,1,t}} \qquad \forall p \in \mathcal{P}, t \in \mathcal{T}$$

$$\sum_{p \in \mathcal{P}} x_{p,t} \leq CAP_t \qquad \forall t \in \mathcal{T}$$

Constraints

$$\sum_{s=1}^{t} x_{p,s} \geq \mathbf{D_{p,1,t}} \qquad \forall p \in \mathcal{P}, t \in \mathcal{T}$$

$$\sum_{p \in \mathcal{P}} x_{p,t} \leq CAP_t \qquad \forall t \in \mathcal{T}$$

```
! Satisfy the total demand
 forall(p in P,t in T)
   sum(s in 1..t) x(p,s) >= D(p,1,t)

! Capacity limits
 forall(t in T) sum(p in P) x(p,t) <= CAP(t)
```

ELS: (l,S)-inequalities

$$\forall p \in \mathcal{P}, \quad \mathbf{l} \in \mathcal{T}, \quad \mathbf{S} \subseteq \{1 \ldots \mathbf{l}\}$$

$$\sum_{t \in S} x_{p,t} + \sum_{t=1 | t \notin S}^{l} D_{p,t,l} \cdot y_{p,t} \geq D_{p,1,l}$$

ELS: (l,S)-inequalities

$$\forall p \in \mathcal{P}, \quad \mathbf{l} \in \mathcal{T}, \quad \mathbf{S} \subseteq \{1 \dots \mathbf{l}\}$$

$$\sum_{t \in S} x_{p,t} + \sum_{t=1 | t \notin S}^{l} D_{p,t,l} \cdot y_{p,t} \geq D_{p,1,l}$$

- $\sum_{t \in S} x_{p,t}$: actual production of product p in periods included in S
- $\sum_{t=1|t \notin S}^{l} D_{p,t,l} \cdot y_{p,t}$: maximum potential production of product p in the remaining periods (those not in S)
- $D_{p,1,l}$: total demand of product p in periods 1 to l

ELS: (l,S)-inequalities

$$\forall p \in \mathcal{P}, \quad \mathbf{l} \in \mathcal{T}, \quad \mathbf{S} \subseteq \{1 \ldots \mathbf{l}\}$$

$$\sum_{t \in S} x_{p,t} + \sum_{t=1 | t \notin S}^{l} D_{p,t,l} \cdot y_{p,t} \geq D_{p,1,l}$$

- $\sum_{t \in S} x_{p,t}$: actual production of product p in periods included in S
- $\sum_{t=1|t \notin S}^{l} D_{p,t,l} \cdot y_{p,t}$: maximum potential production of product p in the remaining periods (those not in S)
- $D_{p,1,l}$: total demand of product p in periods 1 to l

$$\sum_{t \in S} x_{p,t} + \sum_{t=1 | t \notin S}^{l} D_{p,t,l} \cdot y_{p,t} \geq \sum_{\mathbf{t=1}}^{\mathbf{l}} \mathbf{min(x_{p,t},\ D_{p,t,l} \cdot y_{p,t})} \geq \mathbf{D_{p,1,l}}$$

ELS: (I,S)-inequalities

$$p = 1, \quad l = 3, \quad \forall \mathbf{S} \subseteq \{1, 2, \mathbf{3}\}$$

t	1	2	3
$DEM_{1,t}$	2	3	5

$$\sum_{t \in S} x_{1,t} + \sum_{t=1 \mid t \notin S}^{3} D_{1,t,3} \cdot y_{1,t} \geq \sum_{\mathbf{t=1}}^{\mathbf{3}} \mathbf{min}(\mathbf{x_{1,t}}, \ \mathbf{D_{1,t,3}} \cdot \mathbf{y_{1,t}}) \geq \mathbf{D_{1,1,3}}$$

$$
\begin{array}{lrr}
S = \emptyset \rightarrow & 10 \cdot y_{1,1} + 8 \cdot y_{1,2} & +5 \cdot y_{1,3} \geq 10 \\
S = \{1\} \rightarrow & \mathbf{x_{1,1}} + 8 \cdot y_{1,2} & +5 \cdot y_{1,3} \geq 10 \\
S = \{2\} \rightarrow & 10 \cdot y_{1,1} + \mathbf{x_{1,2}} \quad & +5 \cdot y_{1,3} \geq 10 \\
S = \{3\} \rightarrow & 10 \cdot y_{1,1} + 8 \cdot y_{1,2} & +\mathbf{x_{1,3}} \geq 10 \\
S = \{1, 2\} \rightarrow & \mathbf{x_{1,1}} + \mathbf{x_{1,2}} \quad & +5 \cdot y_{1,3} \geq 10 \\
S = \{2, 3\} \rightarrow & 10 \cdot y_{1,1} + \mathbf{x_{1,2}} \quad & +\mathbf{x_{1,3}} \geq 10 \\
S = \{1, 3\} \rightarrow & \mathbf{x_{1,1}} + 8 \cdot y_{1,2} & +\mathbf{x_{1,3}} \geq 10 \\
S = \{1, 2, 3\} \rightarrow & \mathbf{x_{1,1}} + \mathbf{x_{1,2}} \quad & +\mathbf{x_{1,3}} \geq 10
\end{array}
$$

Using valid inequalities

$$\sum_{t \in S} x_{p,t} + \sum_{t=1 | t \notin S}^{l} D_{p,t,l} \cdot y_{p,t} \geq \sum_{\mathbf{t=1}}^{\mathbf{l}} \mathbf{min(x_{p,t},\ D_{p,t,l} \cdot y_{p,t})} \geq \mathbf{D_{p,1,l}}$$

Using valid inequalities

$$\sum_{t \in S} x_{p,t} + \sum_{t=1 | t \notin S}^{l} D_{p,t,l} \cdot y_{p,t} \geq \sum_{\mathbf{t=1}}^{\mathbf{l}} \mathbf{min}(\mathbf{x_{p,t}},\ \mathbf{D_{p,t,l}} \cdot \mathbf{y_{p,t}}) \geq \mathbf{D_{p,1,l}}$$

- add them to the initial formulation
  - creates a formulation with better LP relaxation
  - only when you have a **"small"** set of valid inequalities
  - easy to implement

Using valid inequalities

$$\sum_{t \in S} x_{p,t} + \sum_{t=1 | t \notin S}^{l} D_{p,t,l} \cdot y_{p,t} \geq \sum_{\mathbf{t=1}}^{\mathbf{l}} \mathbf{min(x_{p,t},\ D_{p,t,l} \cdot y_{p,t})} \geq \mathbf{D_{p,1,l}}$$

- add them to the initial formulation
  - creates a formulation with better LP relaxation
  - only when you have a **"small"** set of valid inequalities
  - easy to implement

- add them only as needed **to cut off fractional solutions**

Cut Manager

- the search for a solution of a MILP problem involves optimisation of a large number of LP problems (**nodes**)
- this process is often made more efficient by supplying additional rows (**cuts**) to the matrix which reduce the size of the feasible region, whilst ensuring that it still contains any optimal integer solution
- by default, cuts are automatically added to the matrix by the solver during a global search to speed up the solution process
- users may also write their own cut manager routines to be called at various points during the Branch and Bound search

Cut Manager

- users may also write their own cut manager routines to be called at various points during the Branch and Bound search

addcut

Purpose

Add a cut to the problem in the optimizer.

Synopsis

```
procedure addcut(cuttype:integer, type:integer, linexp:linctr)
```

Arguments

cuttype	Integer number for identification of the cut
type	Cut type (equation/inequality), which may be one of: CT_GEQ Inequality (greater or equal) CT_LEQ Inequality (less or equal) CT_EQ Equality
linexp	Linear expression (= unbounded constraint)

Further information

This procedure adds a cut to the problem in the Optimizer. The cut is applied to the current node and all its descendants.

Callback functions

"users may define their own routines which should be called at various stages during the optimisation process, prompting the solver to return to the user's program before continuing with the solution algorithm"

Callback functions

"users may define their own routines which should be called at various stages during the optimisation process, prompting the solver to return to the user's program before continuing with the solution algorithm"

setcallback

Purpose

Set optimizer callback functions and procedures.

Synopsis

```
procedure setcallback(cbtype:integer, cb:string)
```

Callback functions

"users may define their own routines which should be called at various stages during the optimisation process, prompting the solver to return to the user's program before continuing with the solution algorithm"

setcallback

Purpose

Set optimizer callback functions and procedures.

Synopsis

```
procedure setcallback(cbtype:integer, cb:string)
```

- **cbtype** : type of the callback

cbtype Type of the callback:

XPRS_CB_CHGNODE	User select node callback
XPRS_CB_PRENODE	User preprocess node callback
XPRS_CB_OPTNODE	User optimal node callback
XPRS_CB_INFNODE	User infeasible node callback
XPRS_CB_INTSOL	User integer solution callback
XPRS_CB_NODECUTOFF	User cut-off node callback
XPRS_CB_NEWNODE	New node callback
XPRS_CB_BARITER	Barrier iteration callback
XPRS_CB_CUTMGR	Cut manager (branch-and-bound node) callback

Callback functions

"users may define their own routines which should be called at various stages during the optimisation process, prompting the solver to return to the user's program before continuing with the solution algorithm"

setcallback

Purpose

Set optimizer callback functions and procedures.

Synopsis

```
procedure setcallback(cbtype:integer, cb:string)
```

- **cbtype** : type of the callback
- **cb** : name of the callback function; the parameters and the type of the return value vary depending on *cbtype*

cbtype Type of the callback:

XPRS_CB_CHGNODE	User select node callback
XPRS_CB_PRENODE	User preprocess node callback
XPRS_CB_OPTNODE	User optimal node callback
XPRS_CB_INFNODE	User infeasible node callback
XPRS_CB_INTSOL	User integer solution callback
XPRS_CB_NODECUTOFF	User cut-off node callback
XPRS_CB_NEWNODE	New node callback
XPRS_CB_BARITER	Barrier iteration callback
XPRS_CB_CUTMGR	Cut manager (branch-and-bound node) callback

Callback functions: example

```
!The following example defines a procedure to handle
    solution printing and sets it to be called whenever an
    integer solution is found using the integer solution
    callback
procedure printsol
 declarations
  objval:real
 end-declarations

 objval:= getparam("XPRS_lpobjval")
 writeln("Solution value: ", objval)
end-procedure

setcallback(XPRS_CB_INTSOL, "printsol")
```

Cut Manager

- the solver works with tolerance values for solution feasibility that are typically of the order of 10^{-6} by default. When evaluating a solution **it is important to take into account these tolerances**

Cut Manager

- the solver works with tolerance values for solution feasibility that are typically of the order of 10^{-6} by default. When evaluating a solution **it is important to take into account these tolerances**
- to run the cut manager from Mosel, it may be necessary to (re)set certain control parameters of the optimiser:
  - presolve
  - cut strategy
  - reserving space for extra rows in the matrix

```
! Switch presolve off
setparam("XPRS_PRESOLVE", 0)

! No cuts
setparam("XPRS_CUTSTRATEGY", 0)

! Reserve extra rows in matrix
setparam("XPRS_EXTRAROWS", 4000)
```

PRESOLVE

PRESOLVE	1	This control determines whether presolving should be performed prior to starting the main algorithm. Presolve attempts to simplify the problem by detecting and removing redundant constraints, tightening variable bounds, etc. In some cases, infeasibility may even be determined at this stage, or the optimal solution found. -1 = Presolve applied, but a problem will not be declared infeasible if primal infeasibilities are detected. The problem will be solved by the LP optimization algorithm, returning an infeasible solution, which can sometimes be helpful. 0 = Presolve not applied. 1 = Presolve applied. 2 = Presolve applied, but redundant bounds are not removed. This can sometimes increase the efficiency of the barrier algorithm.

CUTSTRATEGY

CUTSTRATEGY	-1	Branch and Bound: This specifies the cut strategy. A more aggressive cut strategy, generating a greater number of cuts, will result in fewer nodes to be explored, but with an associated time cost in generating the cuts. The fewer cuts generated, the less time taken, but the greater subsequent number of nodes to be explored. -1 = Automatic selection of the cut strategy. 0 = No cuts. 1 = Conservative cut strategy. 2 = Moderate cut strategy. 3 = Aggressive cut strategy.

EXTRAROWS

EXTRAROWS	N/A	The initial number of extra rows to allow for in the matrix, including cuts. If rows are to be added to the matrix, then, for maximum efficiency, space should be reserved for the rows before the matrix is input by setting the EXTRAROWS control. If this is not done, resizing will occur automatically, but more space may be allocated than the user actually requires. The space allowed for cuts is equal to the number of extra rows remaining after rows have been added but before the global optimisation starts. EXTRAROWS is set automatically by the optimiser when the matrix is first input to allow space for cuts, but if you add rows, this automatic setting will not be updated. So if you wish cuts, either automatic cuts or user cuts, to be added to the matrix and you are adding rows, EXTRAROWS must be set before the matrix is first input, to allow space both for the cuts and any extra rows that you wish to add. Default value depends on the matrix characteristics.

Cutting plane algorithm

1. solve the LP relaxation
2. identify violated (l, S)-inequalities by testing violations of

$$\sum_{t=1}^{l} min(x_{p,t},\ D_{p,t,l} \cdot y_{p,t}) \geq D_{p,1,l}$$

3. add violated inequalities as cuts to the problem
4. re-solve the LP problem

Cutting plane algorithm

```
parameters
 ALG = 1             ! Default algorithm: no user cuts
 CUTDEPTH = 10       ! Maximum tree depth for cut generation
 EPS = 1e-6          ! Zero tolerance
end-parameters

procedure tree_cut_gen
 setparam("XPRS_PRESOLVE", 0)          ! Switch presolve off
 setparam("XPRS_EXTRAROWS", 4000)      ! Reserve extra rows in matrix
 setcallback(XPRS_CB_CUTMGR, "cb_node")  ! Set the cut-manager callback function
end-procedure
```

- *cb_node* will be called by the solver from every node of the Branch-and-Bound search tree (**XPRS_CB_CUTMGR**)
- the prototype of this function is prescribed by the type of the callback

Cutting plane algorithm options

- *TOPONLY* - generation of cuts only in the root node (**Cut-and-Branch**) or also during the search (**Branch-and-Cut**)
- *SEVERALROUNDS* - number of cut generation passes at a node
- *CUTDEPHT* - search tree depth for cut generation
- exclusive use of (I,S)-cuts or combination with others (e.g.default cuts generated by the solver)

Cutting plane algorithm

```
function cb_node:boolean
  declarations
   solx: array(P,T) of real    ! Sol. values for var.s x
   soly: array(P,T) of real    ! Sol. values for var.s y
   ncut:integer                     ! Counter for cuts
   cut: array(range) of linctr      ! Cuts
   cutid: array(range) of integer   ! Cut type identification
   type: array(range) of integer    ! Cut constraint type
   objval,ds: real
  end-declarations
  depth:=getparam("XPRS_NODEDEPTH")

  if((TOPONLY and depth<1) or (not TOPONLY and depth<=
    CUTDEPTH)) then
   ncut:=0
   forall(t in T, p in P) do    ! Get the solution values
     solx(p,t):=getsol(x(p,t))
     soly(p,t):=getsol(y(p,t))
   end-do
```

Cutting plane algorithm

```
! Search for violated constraints
forall(p in P,l in T) do
  ds:=0
  forall(t in 1..l)
   if(solx(p,t)<D(p,t,l)*soly(p,t)+EPS) then ds+=solx(p,t)
    else  ds += D(p,t,l)*soly(p,t)
   end-if

  ! Generate the violated inequality
  if(ds < D(p,1,l) - EPS) then
    cut(ncut):= sum(t in 1..l)
    if(solx(p,t)<(D(p,t,l)*soly(p,t))+EPS, x(p,t),
        D(p,t,l)*y(p,t)) - D(p,1,l)
    cutid(ncut):= 1
    type(ncut):= CT_GEQ
    ncut+=1
  end-if
end-do
```

Cutting plane algorithm

```
    returned:=false  ! Call this function once per node

    ! Add cuts to the problem
    if(ncut>0) then
     addcuts(cutid, type, cut);

     if SEVERALROUNDS then
      returned:=true   ! Repeat until no new cuts generated
     end-if
    end-if
   end-if
  end-function
```

- at every node this function is called repeatedly, followed by a re-solution of the current LP, as long as it returns true.

Cutting plane algorithm options

```
SEVERALROUNDS:=false
TOPONLY:=false

case ALG of
 0: break
 1: setparam("XPRS_CUTSTRATEGY", 0) ! No cuts
 2: setparam("XPRS_PRESOLVE", 0)  ! No presolve
 3: tree_cut_gen  ! User branch-and-cut + automatic cuts
 4: do  tree_cut_gen !User branch-and-cut (several rounds)
        setparam("XPRS_CUTSTRATEGY", 0)  !no automatic cuts
        SEVERALROUNDS:=true  end-do
 5: do  tree_cut_gen  !User cut-and-branch (several rounds)
        SEVERALROUNDS:=true  ! + automatic cuts
        TOPONLY:=true  end-do
 6: do  tree_cut_gen  !User branch-and-cut (several rounds)
        SEVERALROUNDS:=true   end-do  ! + automatic cuts
end-case
```

Cutting plane algorithm results

ALG	First	Best	Opt./Best bound
0	817 (2s/53)	788 (703s/187933)	774.75 (1800s/358969)
1	852 (0s/77)	800 (592s/244761)	723.13 (1800s/440753)
2	845 (0s/77)	803 (235s/121778)	727.95 (1800s/412300)
3	818 (1s/50)	788 (271s/27653)	783.06 (1800s/190171)
4	831 (1s/62)	788 (253s/19834)	779.95 (1800s/146871)
5	809 (0s/45)	790 (984s/118846)	784.07 (1800s/214400)
6	793 (1s/32)	788 (476s/45495)	Opt. (1571s/168820)

- larger problem (20 time periods)
- First, Best solution found and Best lower bound (*Opt.* if optimality was proven)
- **solution value (running time / number of nodes)**
- all runs were stopped after 1800 seconds