Introdução ao XPress-Mosel

Flávio Keidi Miyazawa

IC - UNICAMP

Considere a formulação do Problema da Dieta

$$\begin{array}{ll} \text{minimize} & 4{,}78x_1 + 1{,}48x_2 + 0{,}85x_3 \\ \text{sujeito a} & \left\{ \begin{array}{rrrcl} 110x_1 + & 170x_2 + & 1150x_3 & \geq & 800 \\ 110x_1 + & 170x_2 + & 1150x_3 & \leq & 1200 \\ 29x_1 + & 15x_2 & & \geq & 10 \\ 29x_1 + & 15x_2 & & \leq & 18 \\ x_1 \geq 0 & x_2 \geq 0 & x_3 \geq 0 & & \end{array} \right. \end{array}$$

Resolvendo-se:

Obtemos uma dieta de 1,49 reais com

$x_1 = 0$ kg. de carne de boi,

$x_2 = 0.67$ kg. de feijão cozido e

$x_3 = 0.60$ lt. de leite desnatado.

A quantidade diária de cálcio nesta dieta é 800 e de ferro é 10.

Formulação do Problema da Dieta (por variáveis simples)

```
model "dieta"
 uses "mmxprs"    ! Indicando que vamos resolver P.L.

 ! Declaracao de algumas variaveis
 declarations
  q_carne, q_feijao, q_leite: mpvar    ! Variaveis do PL
 end-declarations

 ! Funcao objetivo
 obj:= 4.78*q_carne + 1.48*q_feijao + 0.85*q_leite

 ! Restricoes do PL
 l1 := 110*q_carne +170*q_feijao + 1150*q_leite >= 800
 l2 := 110*q_carne +170*q_feijao + 1150*q_leite <= 1200
 l3 :=  29*q_carne + 15*q_feijao >= 10
 l4 :=  29*q_carne + 15*q_feijao <= 18
```

```
 ! Restricoes de nao negatividade
 q_carne >= 0
 q_feijao >= 0
 q_leite >= 0

! Resolva o LP
 minimize(obj)

! Impressao dos resultados
 writeln("valor da dieta: ", getobjval)
 writeln("quantidade de carne = ",getsol(q_carne))
 writeln("quantidade de feijao = ",getsol(q_feijao))
 writeln("quantidade de leite = ",getsol(q_leite))
end-model
```

Alguns tipos de dados usados no XPress-Mosel

Tipos básicos

`mpvar`

variável de programa linear

`real`

números em ponto flutuante

`integer`

valores inteiros

`string`

cadeia de caracteres

Conjuntos

- $\langle \mathit{Inicio}\,..\,\mathit{Fim} \rangle$

  conjunto de números em um intervalo

- $\{\langle \mathit{Elem}_1 \rangle, \langle \mathit{Elem}_2 \rangle, \ldots, \langle \mathit{Elem}_k \rangle\}$

  conjunto de k elementos (todos os k elementos devem ser dados)

- `set of` $\langle \mathit{Tipo} \rangle$

Exemplos:

$1..100$

$-50..50$

{`"Norte","Sul","Leste","Oeste"`}

`set of string`

`set of real`

Comandos de Escrita e Leitura

```
...
declarations
  n,m : integer
end-declarations

write("Entre com a ordem da matriz (linha coluna): ")
readln(n,m)

writeln("O numero de linhas  lido foi ",n)
writeln("O numero de colunas lido foi ",m)
...
```

Manipulação de Conjuntos

```
model "Exemplo com Conjuntos"

declarations
   Cidades    = {"sao paulo", "campinas", "fortaleza",
              "jundiai", "recife", "rio de janeiro"}
   Litoraneas = {"rio de janeiro", "buzios", "recife"
              "salvador", "fortaleza"}
   Capitais   = {"sao paulo", "salvador", "fortaleza",
              "rio de janeiro"}
end-declarations


! Conjunto vazio
Vazio := {}
writeln("Conjunto vazio = ",Vazio)
```

```
! Uniao de conjuntos
Lugares := Cidades + Litoraneas + Capitais

! Intersecao de conjuntos
Inter := Cidades * Litoraneas * Capitais

! Diferenca
Nao_Capitais := Cidades - Capitais

writeln("Alguns lugares: ",Lugares)
writeln("Algumas cidades litoraneas que sao capitais: ",
        Inter)
writeln("Algumas cidades que nao sao capitais: ",
        Nao_Capitais)

end-model
```

Comando Condicional

```
...
! se A for maior ou igual a 20
if A >= 20 then
   x <= 10  ! entao insira esta desigualdade
endif
...
```

```
...
! se e nao pertence ao conjunto Conj
if (not (e in Conj)) then
   x <= 10
else
   x >= 20
endif
...
```

Comando case

```
 ...
case A of
   -MAX_INT .. 10:  x >= 10
   11 .. 20:        x >= 20
   100 .. MAX_INT:  x >= 50
   30,40,50:        x >= 100

   else             x >= 200  ! se nao entrou em
                              ! nenhum caso anterior
endif
...
```

Comando forall

```
 ...
soma := 0
forall (p in 1..100)
   soma := soma + p

writeln("Media dos elementos de 1 a 100 = ",soma/100)
...
```

Vetores

`array` $(\langle Conjunto \rangle)$ `of` $\langle Tipo \rangle$

Vetor de elementos de tipo $\langle Tipo \rangle$,
cada elemento indexado por elemento em $\langle \text{Conjunto} \rangle$

```
...
 declarations
    Indices: 1..100
    Estados: {"Norte","Sul","Leste","Oeste"}
    V:          array (Indices) of integer
    Populacao: array (Estados) of real
 end-declarations

 forall (i in Indices)
    V(i) := i

 forall (est in Estados)
    Populacao(est) := 0

 forall (i in Indices)
    writeln(V(i))
...
```

Vetores Multidimensionais

`array` $(\langle Conj_1\rangle,\langle Conj_2\rangle)$ `of` $\langle Tipo\rangle$

Vetor bidimensional de elementos de tipo $\langle Tipo\rangle$

```
...
 declarations
    Indices: 1..100
    Estados: {"Norte","Sul","Leste","Oeste"}
    V:          array (Indices,Estados) of real
 end-declarations

 forall (i in Indices , est in Estados)
    V(i,est) := 123
...
```

Bloco de comandos

Quando queremos colocar vários comandos sob a ação dos comandos `while` e `forall`, devemos colocá-los em um *bloco de comandos*:

```
do
    ⟨lista de comandos⟩
end-do
```

Exemplo:

```
 ...
! Para todo estado est em Estados
forall (est in Estados) do
   ! os dois comandos abaixo estao dentro da execucao do f
   writeln("Nome do estado = ",est)
   V(est) := 0
end-do
...
```

Comando while

```
model "MDC"
declarations
  a,b,n,m : integer
end-declarations

write("Entre com dois numeros inteiros: ")
readln(a,b)
n:=a
m:=b

while (n<>m) do
   if (n > m) then
      n := n-m
   else
      m := m-n
   endif
end-do

writeln("O maximo divisor comum de ",a," e ",b," eh ",n)
end-model
```

Comando repeat

```
...
n := 0
repeat
   write("Entre com um numero maior que zero: ")
   readln(n)
   if (n <= 0) then
      writeln("Numero invalido")
   endif
until (n>0)
...
```

Operações com Conjuntos

Criando variáveis com novo índice: `create`(⟨variável com novo índice⟩)

Verificando existência de variável: `exists`(⟨variável⟩)

```
...
declarations
   S: set of string
   x: array(S) of mpvar
end-declarations
...
create(x("a"))
create(x("b"))
...
x("a")+x("b") <= 10
if (exists(x("a"))) then writeln("variavel x(a) existe"
if (exists(x("c"))) then writeln("variavel x(c) existe"
...
```

Percorrendo elementos de forma condicionada

Através do operador '|'

```
...
!
! O comando abaixo imprime os elementos
! V(1),...,V(100) que tem valores entre
! 50 e 200
!
forall( i in 1..100 | 50 <= V(i), V(i) <= 200 )
     writeln(V(i))
...
```

Adicionando novos elementos em um conjunto: +=

Removendo elementos de um conjunto: -=

Obtendo tamanho de um conjunto: `getsize`(⟨variável de conjunto⟩)

```
...
! Conjunto vazio
S := {1,2,3}
S += {2,4,6}
!conjunto esta com os elementos 1,2,3,4,6

S -= {1,6,7}
!conjunto esta com os elementos 2,3,4

writeln(getsize(S))    ! imprime o valor 3
...
```

Restrição para variáveis inteiras

variáveis binárias: `is_binary`

variáveis inteiras: `is_integer`

```
...
declarations
    x,y,z: mpvar
end-declarations
...
x is_binary     ! x so' tera' valores em {0,1}
y is_integer    ! y so' tera' valores inteiros
...
```

Restrições do programa linear: Somatórias

Exemplo	Mosel
$\sum_{e \in S} x_i \leq 10$	`sum(e in S)  x(i)<=10`
$\sum_{e \in S:\ e <= 123} x_i \leq 10$	`sum(e in S|  e<=10) x(i)<=10`
$\sum_{i \in V} \sum_{j \in W} x_{ij} = 10$	`sum(i in V, j in W)  x(i,j)=10`
$\sum_{i \in V} \sum_{j \in W : i \neq j} x_{ij} = 10$	`sum(i in V, j in W| i<>j)  x(i,j)=10`

Considere a formulação do Problema da Dieta

$$
\begin{array}{ll}
\text{minimize} & 4{,}78x_1 + 1{,}48x_2 + 0{,}85x_3 \\
\text{sujeito a} & \left\{
\begin{array}{rrrcl}
110x_1 + & 170x_2 + & 1150x_3 & \geq & 800 \\
110x_1 + & 170x_2 + & 1150x_3 & \leq & 1200 \\
29x_1 + & 15x_2 & & \geq & 10 \\
29x_1 + & 15x_2 & & \leq & 18 \\
x_1 \geq 0 & x_2 \geq 0 & x_3 \geq 0 & &
\end{array}
\right.
\end{array}
$$

Resolvendo-se:

Obtemos uma dieta de 1,49 reais com

$x_1 = 0$ kg. de carne de boi,

$x_2 = 0.67$ kg. de feijão cozido e

$x_3 = 0.60$ lt. de leite desnatado.

A quantidade diária de cálcio nesta dieta é 800 e de ferro é 10.

Formulação do Problema da Dieta (vetores)

```
model "dieta2.mos"

 uses "mmxprs"

 ! Declaracao de algumas variaveis
 declarations
    ! declaracao do conjunto de indices
    ind = {"carne", "feijao", "leite"}

    ! Declaracao do vetor de variaveis do PL
    x: array(ind) of mpvar

    ! Declaracao de vetores de preco/calcio/ferro
    preco: array(ind) of real
    calcio: array(ind) of real
    ferro: array(ind) of real
 end-declarations
```

```
! Inicializacao do preco/calcio/ferro de cada alimento
preco  := [ 4.78, 1.48, 0.85 ]
calcio := [ 110, 170 , 1150 ]
ferro  := [ 29 , 15 , 0 ]

! Construcao da funcao objetivo
obj:= sum(i in ind) preco(i)*x(i)

! Construcao das restricoes para calcio
sum(i in ind) calcio(i)*x(i) <= 1200
sum(i in ind) calcio(i)*x(i) >= 800

! Construcao das restricoes para ferro
sum(i in ind) ferro(i)*x(i) <= 18
sum(i in ind) ferro(i)*x(i) >= 10

! Restricoes de nao negatividade
forall (i in ind)
   x(i)>=0
```

```
 ! Resolva o LP
 minimize(obj)

 ! Impressao do valor da solucao
 writeln("valor da dieta: ", getobjval)

 ! Impressao da quantidade de cada alimento
 forall (i in ind)
     writeln("quantidade de ",i," =", getsol(x(i)))

end-model
```

Formulação do Problema da Dieta (inicialização por arquivo I)

```
model "dieta3.mos"  ! exemplo de leitura de vetores em arq
 uses "mmxprs"

 ! Declaracao de algumas variaveis
 declarations
  ind = 1..3
  x: array(ind) of mpvar
  custo:  array(ind) of real
  calcio: array(ind) of real
  ferro:  array(ind) of real
 end-declarations

 ! leitura dos vetores atraves de dados que estao no arqui
 initializations from 'dieta3.dat'
   custo  as 'CUSTO'
   calcio as 'CALCIO'
   ferro  as 'FERRO'
 end-initializations
```

```
 ! funcao objetivo
 obj:= sum(i in ind) custo(i)*x(i)

 ! restricoes de calcio
 sum(i in ind) calcio(i)*x(i) >= 800
 sum(i in ind) calcio(i)*x(i) <= 1200

 ! restricoes de ferro
 sum(i in ind) ferro(i)*x(i) >= 10
 sum(i in ind) ferro(i)*x(i) <= 18

 ! Restricoes de nao negatividade
 forall (i in ind) x(i)>=0

! Resolva o LP
 minimize(obj)
! Impressao dos resultados
 writeln("valor da dieta: ", getobjval)
 forall (i in ind)
writeln("quantidade de x(",i,") =", getsol(x(i)))
end-model
```

Arquivo dieta3.dat

```
!dados para o programa dieta3.mos
!---- veja abaixo como foi feita a declaracao no programa
!
!    declarations
!        ind = 1..3
!        custo:  array(ind) of real
!        calcio: array(ind) of real
!        ferro:  array(ind) of real
!    end-declarations
!
!    initializations from 'dieta3.dat'
!        custo  as 'CUSTO'
!        calcio as 'CALCIO'
!        ferro  as 'FERRO'
!    end-initializations
!--------------------------------------------------------------
CUSTO:  [ 4.78  1.48  0.85 ]
CALCIO: [  110  170   1150 ]
FERRO:  [   29   15      0 ]
```

Formulação do Problema da Dieta (inicialização por arquivo II)

```
 ...
 ! Declaracao de algumas variaveis
 declarations
  ind = 1..3
  x: array(ind) of mpvar
  custo:  array(ind) of real
  calcio: array(ind) of real
  ferro:  array(ind) of real
 end-declarations

 ! leitura dos vetores, como matriz
 initializations from 'dieta4.dat'
   [custo,calcio,ferro] as 'ALIMENTO'
 end-initializations
...
```

Arquivo dieta4.dat)

```
! cada linha representa os dados de um alimento [preco cal
ALIMENTO: [ [ 4.78   110  29 ]
            [ 1.48   170  15 ]
            [ 0.85  1150   0 ] ]
```

Problema do Emparelhamento

Def.: Dado um grafo $G = (V, E)$, dizemos que $M \subseteq E$ é um emparelhamento de G se M não tem arestas com extremos em comum.

Formulação Inteira do problema do Emparelhamento:

$$
\begin{array}{ll}
\text{maximize} & \sum_{e \in E} c_e x_e \\
\text{sujeito a} & \left\{ \begin{array}{rcll} \sum_{e \in \delta(v)} x_e & \leq & 1 & \forall v \in V, \\ x_e & \in & \{0, 1\} & \forall e \in E. \end{array} \right.
\end{array}
$$

Formulação em Mosel

```
model "emparelhamento"
 uses "mmxprs"

 ! Declaracao de variaveis
 declarations
   V: set of string              ! conj. de vertices
   custo: array(V,V) of real     ! custo para cada aresta
   x: array(V,V) of mpvar        ! var. do LP para cada arest
 end-declarations

 ! leitura dos vetores atraves de dados que estao no arqui
 initializations from 'grafo.dat'
   V  as 'VERTICES'
   custo as 'CUSTO'
 end-initializations

! se custo(i,j) existe (i.e., existe a aresta (i,j) )
! criamos a variavel x(i,j) e dizemos que e' binaria
forall (i in V, j in V | exists(custo(i,j))) do
       create(x(i,j))
       x(i,j) is_binary
end-do
```

```
! para todo vertice i, a soma das variaveis de arestas
! que incidem em i e' <= 1
forall (i in V) do
        sum(j in V | exists(x(i,j))) x(i,j) +
        sum(j in V | exists(x(j,i))) x(j,i) <= 1
end-do

! Funcao objetivo: soma das arestas escolhidas
obj := sum(i in V, j in V | exists(x(i,j)))
           custo(i,j)*x(i,j)

! Queremos um emparelhamento de peso maximo
maximize(obj)

! Imprimimos as variaveis (arestas) com valor 1 e seu pes
forall (i in V, j in V | exists(x(i,j))) do
   if (getsol(x(i,j))=1) then
      writeln('aresta (',i,',',j,' , ',custo(i,j),')')
   end-if
end-do

end-model
```

Exemplo de entrada (grafo) para programa do emparelhamento

```
!   Um grafo de entrada: grafo.dat
!
!   Lista de vertices
!
VERTICES: [1 2 3 4 5 6 7 8]
!
!   Lista de arestas (i j) e seus custos [<custo(i,j)>]
!
CUSTO:   [    (1 2)  [ 500.0]
              (2 3)  [ 400.0]
              (3 4)  [ 945.0]
              (4 5)  [ 255.0]
              (5 6)  [ 323.0]
              (6 7)  [ 562.0]
              (7 8)  [ 236.0]
              (8 1)  [ 354.0]
              (1 3)  [ 001.0]
              (2 4)  [ 005.0]
              (3 5)  [ 005.0]
              (4 7)  [ 002.0]
              (5 8)  [ 006.0]
              (6 1)  [ 004.0] ]
```