Métodos de Apoio à Decisão

Programação por restrições

João Pedro Pedroso

2024/2025

Últimas aulas:

- Introdução à programação por restrições

Hoje:

- Extensões do AMPL (conclusão)

Programação matemática e por restrições: particularidades do AMPL

Fonte:
"Advances in Model-Based Optimization with AMPL"
Robert Fourer, Gleb Belov, Filipe Brandão 2022

Novas funcionalidades do AMPL

- Objetivo: permitir uma descrição mais natural de formulações
  - extensões: construções escritas em AMPL são traduzidas para modelo de otimização matemático
  - modelos AMPL $\neq$ modelos de *programação/otimização matemática*
    - permitem ir além de simples restrições lineares

Novas funcionalidades do AMPL: exemplo

Modelo com custos fixos:

```
sum {(i,j) in ARCS} fix_cost[i,j] * Use[i,j];
```

- precisa de uma restrição a *obrigar* `Use[i,j]` a ser igual a 1 quando há fluxo em `(i,j)`

Modelo com extensões do AMPL:

```
sum {(i,j) in ARCS}
    if exists {p in PRODUCTS} Flow[p,i,j] > 0 then fix_cost[i,j]
```

Funções max min abs

```
param n;
set I = 1..n;
param v{I};
var x{I} binary;

data;
param n := 5;
param v :=
    1   7
    2   4
    3   9
    4   6
    5   6;

model;
minimize z:
    max {i in I} v[i] * x[i];
OneX:
    sum {i in I} x[i] = 1;

option solver gurobi;
solve;
display z, v, x;
```

max(var-expr-list)

max {indexing} var-expr

```
minimize z:
    max {i in I} v[i] * x[i];

z = 4

:   v   x    :=
1   7   0
2   4   1
3   9   0
4   6   0
5   6   0
;
```

Funções max min abs

```
param n;
set I = 1..n;
param v{I};
var x{I} binary;

data;
param n := 5;
param v :=
    1   7
    2   4
    3   9
    4   6
    5   6;

model;
minimize z:
    max (v[1] * x[1],
         v[2] * x[2],
         v[3] * x[3],
         v[4] * x[4],
         v[5] * x[5]);

OneX:  sum {i in I} x[i] = 1;
```

max(var-expr-list)

max {indexing} var-expr

```
max (v[1] * x[1], v[2] * x[2], v[3] * x[3],

z = 4

:   v   x    :=
1   7   0
2   4   1
3   9   0
4   6   0
5   6   0
;
```

Funções max min abs

```
param n;
set I = 1..n;
param v{I};
var x{I} binary;

data;
param n := 5;
param v :=
    1   7
    2   4
    3   9
    4   6
    5   6;

model;
maximize z:
    sum {i in I} abs(8 - v[i]) * x[i];
OneX:  sum {i in I} x[i] = 1;

option solver gurobi;
solve;
display z, v, x;
```

abs(var-expr-list)

abs {indexing} var-expr

```
z = 4

:   v   x    :=
1   7   0
2   4   1
3   9   0
4   6   0
5   6   0
;
```

Operador count

```
param n;
set I = 1..n;
param v{I};
var x{I} binary;

data;
param n := 5;
param v :=
    1   7
    2   4
    3   9
    4   6
    5   6;

model;
maximize z:
    sum {i in I} v[i] * x[i];
OneX:
    count {i in I} (x[i] > 0) = 1;

option solver gurobi;
solve;
display z, v, x;
```

count {indexing}
(constraint-expr)

```
OneX:
    count {i in I} (x[i] > 0) = 1;


z = 9

:   v   x   :=
1   7   0
2   4   0
3   9   1
4   6   0
5   6   0
;
```

Operador atmost

```
param n;
set I = 1..n;
param v{I};
var x{I} binary;

data;
param n := 5;
param v :=
    1   7
    2   4
    3   9
    4   6
    5   6;

model;

maximize z:
    sum {i in I} v[i] * x[i];

MaxTwoX:
    atmost 2 {i in I} (x[i] > 0);

option solver gurobi;
solve;
```

atmost k {indexing}
(constraint-expr)

```
MaxTwoX:
    atmost 2 {i in I} (x[i] > 0);


z = 16

:   v   x   :=
1   7   1
2   4   0
3   9   1
4   6   0
5   6   0
;
```

Operador atleast

```
param n;
set I = 1..n;
param v{I};
var x{I} binary;

data;
param n := 5;
param v :=
    1   7
    2   4
    3   9
    4   6
    5   6;

model;

maximize z:
    sum {i in I} v[i] * x[i];

MinFourXAreZero:
    atleast 4 {i in I} (x[i] = 0);

option solver gurobi;
solve;
```

atleast k {indexing}
(constraint-expr)

```
MinFourXAreZero:
    atleast 4 {i in I} (x[i] = 0);


z = 9

:   v   x   :=
1   7   0
2   4   0
3   9   1
4   6   0
5   6   0
;
```

Operador exactly

```
param n;
set I = 1..n;
param v{I};
var x{I} binary;

data;
param n := 5;
param v :=
    1   7
    2   4
    3   9
    4   6
    5   6;

model;
maximize z:
    sum {i in I} v[i] * x[i];
ThreeXAreZero:
    exactly 3 {i in I} (x[i] = 0);

option solver gurobi;
solve;
display z, v, x;
```

exactly k {indexing}
(constraint-expr)

```
ThreeXAreZero:
    exactly 3 {i in I} (x[i] = 0);


z = 16

:   v   x    :=
1   7   1
2   4   0
3   9   1
4   6   0
5   6   0
;
```

Operador alldiff

```
param n;
set I = 1..n;
param v{I};
var x{I} binary;

data;
param n := 5;
param v :=
    1   7
    2   4
    3   9
    4   6
    5   6;

model;
maximize z:
    sum {i in I} v[i] * x[i];
AllDiff:
    alldiff {i in I} x[i] * v[i];

option solver gurobi;
solve;
display z, v, x;
```

alldiff(var-expr-list)

alldiff {indexing} var-expr

```
AllDiff:
    alldiff {i in I} x[i] * v[i];


z = 26

:   v   x    :=
1   7   1
2   4   1
3   9   1
4   6   0
5   6   1
;
```

Operador numberof

```
param n;
set I = 1..n;
param v{I};
var x{I} binary;

data;
param n := 5;
param v :=
    1   7
    2   4
    3   9
    4   6
    5   6;



model;
maximize z:
    sum {i in I} v[i] * x[i];
NumberOf:
    numberof 6 in ({i in I} v[i]*x[i]) <= 0;

option solver gurobi;
```

numberof k in (var-expr-list)

```
NumberOf:
    numberof 6 in ({i in I} v[i]*x[i]) <= 0;



z = 20

:   v   x     :=
1   7   1
2   4   1
3   9   1
4   6   0
5   6   0
;
```

Operadores de comparação: < > !=

```
param n;
set I = 1..n;
param v{I};
var x{I} binary;

data;
param n := 5;
param v :=
    1   7
    2   4
    3   9
    4   6
    5   6;

model;

maximize z:
    sum {i in I} v[i] * x[i];

Different {i in I}:
    v[i] != 9 * x[i];

option solver gurobi;
solve;
```

```
var-expr1!=var-expr2
var-expr1 > var-expr2
var-expr1 < var-expr2
```

```
Different {i in I}:
    v[i] != 9 * x[i];


z = 23

:   v   x    :=
1   7   1
2   4   1
3   9   0
4   6   1
5   6   1
;
```

Próximas aulas

- Otimização não linear