Métodos de Apoio à Decisão

Programação por restrições

João Pedro Pedroso

2024/2025

Última aula:

- Introdução à programação por restrições

Hoje:

- Programação por restrições: exemplos
- Extensões do AMPL

Aula passada: Queens

```
param n integer > 0;

var x {1..n} integer >= 1 <= n;

subject to
col_conflicts:
    alldiff {i in 1..n} x[i];
diag1_conflicts:
    alldiff {i in 1..n} (x[i] + i);
diag2_conflicts:
    alldiff {i in 1..n} (x[i] - i);
```

- domínio finito para a variáveis
- objetivo não está definido

SEND + MORE = MONEY

SEND + MORE = MONEY

```
  SEND
+ MORE
------
 MONEY
```

SEND + MORE = MONEY

```
  SEND
+ MORE
------
 MONEY
```

```
var S >= 1, <=9, integer;
var E >= 0, <=9, integer;
var N >= 0, <=9, integer;
var D >= 0, <=9, integer;
var M >= 1, <=9, integer;
var O >= 0, <=9, integer;
var R >= 0, <=9, integer;
var Y >= 0, <=9, integer;

s.t. sendmoremoney: 1000 * S + 100 * E + 10 * N + D +
                    1000 * M + 100 * O + 10 * R + E =
          10000 * M + 1000 * O + 100 * N + 10 * E + Y;

s.t. different: alldiff (S, E, N, D, M, O, R, Y);
```

SEND + MORE = MONEY

```
  SEND
+ MORE
------
 MONEY
```

```
# option solver gecode;
# option solver jacop;
ampl: option solver ilogcp;

ampl: solve;

ampl: display S, E, N, D, M, O, R, Y;
S = 9
E = 5
N = 6
D = 7
M = 1
O = 0
R = 8
Y = 2

ampl:
```

Constraint programming: Map coloring

```
param NumColors;
set Countries;
set Neighbors within Countries cross Countries;
var color{Countries} integer >= 1 <= NumColors;

s.t. different_colors{(c1, c2) in Neighbors}:
  color[c1] != color[c2];

data;

param NumColors := 4;
set Countries := Belgium Denmark France Germany Luxembourg Netherlands;
set Neighbors :=
  Belgium France
  Belgium Germany
  Belgium Netherlands
  Belgium Luxembourg
  Denmark Germany
  France  Germany
  France  Luxembourg
  Germany Luxembourg
  Germany Netherlands;
```

Constraint programming: open shop scheduling

Está a organizar uma sessão de entrevistas para os futuros diplomados do seu curso. Há um conjunto de empresas interessadas em conhecer cada um dos alunos e, com base nos dados de entrevistas anteriores, previu as próximas durações de cada uma dessas entrevistas. A que horas deverá cada entrevista começar por forma a que a última termine o mais cedo possível?

	Student		
Company	1	2	3
1	121	661	6
2	333	168	489
3	343	621	212
4	171	505	324

Constraint programming: open shop scheduling

Open-shop scheduling problem (OSSP)

- Dados:
  - conjunto de máquinas/estações de trabalho
  - conjunto de trabalhos, cada um a ser processado durante um determinado período de tempo em cada estação de trabalho
- O processamento pode ser feito numa ordem arbitrária
- Não pode haver sobreposições (*no overlaps*)
  - uma operação de cada vez em cada estação de trabalho
  - uma estação de trabalho de cada vez para as operações de cada trabalho
- Pretende-se determinar o instante em que cada trabalho deve ser processado em cada estação de trabalho, minimizando o makespan
  - data de conclusão da operação que termina mais tarde

3 ou mais estações de trabalho, ou 3 ou mais tarefas $\rightarrow$ OSSP é NP-hard

Modelos de otimização

- Com base em restrições que representam "ou-exclusivo":
  - ou a operação 1 é feita antes de 2, e então $t_2 \geq t_1 + d_1$
  - ou a operação 1 é feita após 2, e então $t_1 \geq t_2 + d_2$
- Os modelos são difíceis de formular
- Geralmente, causam grandes dificuldades aos resolutores...

Open shop scheduling: modelo em programação por restrições

```
param endTime integer > 0;
param nMach integer > 0;
param nJobs integer > 0;
param duration {1..nMach, 1..nJobs};
var Start {1..nMach, 1..nJobs} integer >= 0, <= endTime;
var Makespan integer >= 0, <= endTime;

minimize Objective: Makespan;

subject to
NoJobConflicts  {m in 1..nMach, j1 in 1..nJobs, j2 in j1+1..nJobs}:
   Start[m,j1] + duration[m,j1] <= Start[m,j2]  or
   Start[m,j2] + duration[m,j2] <= Start[m,j1];

NoMachineConflicts {m1 in 1..nMach, m2 in m1+1..nMach, j in 1..nJobs}:
   Start[m1,j] + duration[m1,j] <= Start[m2,j]  or
   Start[m2,j] + duration[m2,j] <= Start[m1,j];

MakespanDefn {m in 1..nMach, j in 1..nJobs}:
   Start[m,j] + duration[m,j] <= Makespan;
```

Open shop scheduling: dados

```
param endTime := 2809 ;
param nMach := 4 ;
param nJobs := 3 ;

param duration:
        1    2    3 :=
   1  121  661    6
   2  333  168  489
   3  343  621  212
   4  171  505  324 ;
```

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 hção 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]
```