

Métodos de Apoio à Decisão

Programação por restrições

João Pedro Pedroso

2023/2024

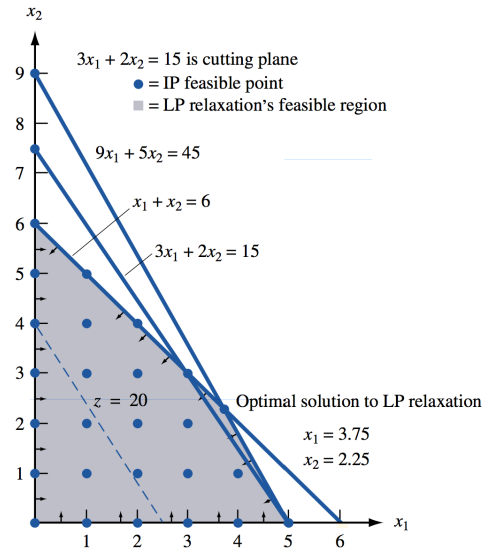
Última aula:

- problema de rotas de veículos (*vehicle routing problems*)
- algoritmo dos planos de corte de Gomory

Hoje:

- Introdução à programação por restrições

Planos de corte: exemplo



Corte válido: desigualdade que se adiciona tal que:

- 1 a solução da relaxação linear passa a ficar **fora** da região admissível;
- 2 todas as soluções inteiras do problema original continuam na região admissível.

10 simple rules:

- 1 For Every Result, Keep Track of How It Was Produced
- 2 Avoid Manual Data Manipulation Steps
- 3 Archive the Exact Versions of All External Programs Used
- 4 Version Control All Custom Scripts
- 5 Record All Intermediate Results, When Possible in Standardized Formats
- 6 For Analyses That Include Randomness, Note Underlying Random Seeds
- 7 Always Store Raw Data behind Plots
- 8 Generate Hierarchical Analysis Output, Allowing Layers of Increasing Detail to Be Inspected
- 9 Connect Textual Statements to Underlying Results
- 10 Provide Public Access to Scripts, Runs, and Results

source: <http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003285>

- **Otimização inteira:** ênfase em encontrar o *melhor objetivo*
- **Programação por restrições:** ênfase em encontrar *uma solução admissível*

Exemplos:

- coloração de grafos com número de cores fixo
- sudoku

Componentes fundamentais:

- **variáveis**
- **domínios**
- **restrições**

Restrições:

- Equações ou desigualdades (como as que vimos em otimização inteira)
- Restrições lógicas
- Restrições globais:
 - permitem especificar com um termo uma propriedade que determinadas variáveis devem possuir
 - muitas vezes, difíceis de exprimir com equações ou desigualdades
 - têm associados *solvers* específicos
 - exemplo: `AllDifferent(x1,x2,x3,x4,x5)` → as variáveis x_1, x_2, x_3, x_4, x_5 deverão todas ter valores diferentes
 - cada x_i deverá ter um domínio finito
 - e.g., $x_i \in \{1, \dots, 10\}$
 - sintaxe de `AllDifferent` depende do resolutor (iremos ver exemplo com AMPL)

- há muitos sistemas para resolução de problemas com programação por restrições
- alguns são específicos para um tipo de problemas (por exemplo, satisfazibilidade)
- os mais completos, implementam a maioria das restrições globais “*comuns*”
- infelizmente, não há uma interface comum a todos eles
 - AMPL suporta:
 - algumas restrições globais
 - operadores lógicos
 - sumário disponível aqui
[https://ampl.com/resources/
logic-and-constraint-programming-extensions/](https://ampl.com/resources/logic-and-constraint-programming-extensions/)

- Sintaxe para programação por restrições em AMPL:
 - Operadores lógicos:
 - and + or + not
 - iterados: exists + forall
 - Operadores condicionais:
 - if-then + if-then-else + ==> + ==> else + <== + <==>
 - Operadores de contagem:
 - iterados: count + atmost + atleast + exactly + numberof
 - coletivos: alldiff
- Alguns resolutores
 - IBM ILOG CP (comercial; disponível no software fornecido `option solver ilogcp;`)
 - Gecode (software livre, disponível aqui)
 - JaCoP (software livre, disponível aqui)

Nota: O sistema GLPK não suporta programação por restrições

Localização espacial:

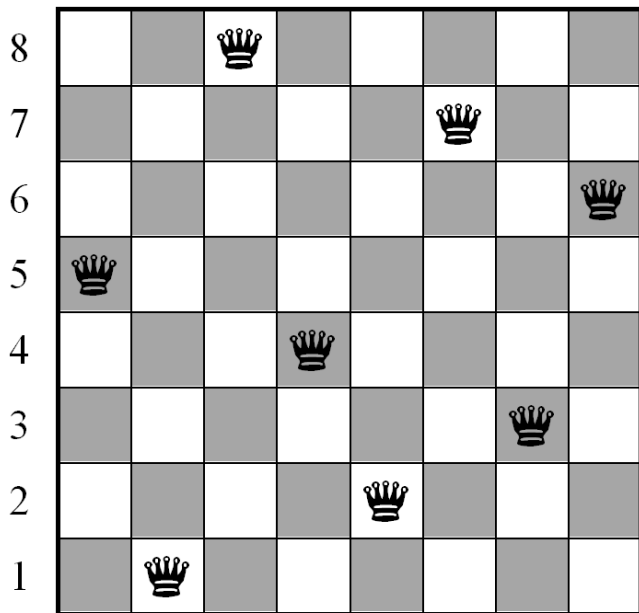
- grelha $n \times n$
- colocar n postos de observação em pontos da grelha
- custos de construção idênticos

Restrições de *não interferência*:

- postos não podem bloquear a vista de outros
 - linhas
 - colunas
 - diagonais

N-Queens problem

Queens



Queens: Mathematical programming

```
param n, integer, > 0, default 8; /* size of the chess board */

var x{1..n, 1..n}, binary; /* x[i,j] = 1 --> a queen is placed in square [i,j] */

s.t.
a{i in 1..n}: sum{j in 1..n} x[i,j] <= 1; /* at most one queen in each row */
b{j in 1..n}: sum{i in 1..n} x[i,j] <= 1; /* at most one queen in each column */
/* at most one queen can be placed in each "\"-diagonal */
c{k in 2..n-2}: sum{i in 1..n, j in 1..n: i-j == k} x[i,j] <= 1;
/* at most one queen can be placed in each "/"-diagonal */
d{k in 3..n+n-1}: sum{i in 1..n, j in 1..n: i+j == k} x[i,j] <= 1;

/* place as many queens as possible: */
maximize obj: sum{i in 1..n, j in 1..n} x[i,j];

solve; /* solve the problem */
for {i in 1..n} /* and print its optimal solution */
{ for {j in 1..n} printf " %s", if x[i,j] then "Q" else ".";
  printf("\n");
}
end;
```

Programação por restrições

Componentes fundamentais:

- **variáveis**
- **domínios**
- **restrições**

Restrições:

- Equações ou desigualdades (como as que vimos em otimização inteira)
- Restrições lógicas
- Restrições globais:
 - permitem especificar com um termo uma propriedade que determinadas variáveis devem possuir
 - muitas vezes, difíceis de exprimir com equações ou desigualdades
 - têm associados *solvers* específicos
 - exemplo: `AllDifferent(x1,x2,x3,x4,x5)` → as variáveis x_1, x_2, x_3, x_4, x_5 deverão todas ter valores diferentes
 - cada x_i deverá ter um domínio finito
 - e.g., $x_i \in \{1, \dots, 10\}$
 - sintaxe de `AllDifferent` depende do resolutor (iremos ver exemplo com AMPL)

Queens: *Constraint programming*

```
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);
```

```
ampl: model queensMIP.mod;
ampl: let n := 8;
ampl: option solver cplex;
ampl: solve;
[...]
```

Objective = find a feasible point.

```
ampl: model queensCP.mod;
ampl: let n := 8;
ampl: option solver ilogcp;
ampl: solve;
ilogcp 12.4.0: feasible solution
1731 choice points, 1458 fails
```

	CP	MIP
n	ilogcp	cplex
50	0.27	0.94
100	0.39	1.36
150	0.61	5.73
200	1.20	125.86
250	1.11	441.39
300	1.62	1470.29
350	1.56	
400	2.18	
450	2.70	
500	4.15	

- Programação por restrições: sudoku.