# Kropki
# Resolução de Problemas de Decisão com Restrições

João Guarda and Ricardo Lopes

Faculdade de Engenharia da Universidade do Porto,
Rua Dr. Roberto Frias, s/n 4200-465 Porto PORTUGAL
{FEUP-PLOG,Turma3MIEIC02,GrupoKropki_3
http://www.fe.up.pt

**Resumo** *Este trabalho apresenta-se com o objetivo principal de entender a abordagem necessária para a resolução de problemas de satisfação de restrições de forma a aplica-la em casos concretos e práticos como problemas de otimização ou decisão combinatória. Foi abordado o jogo Kropki, onde se utilizou programação em lógica com restrições para a sua resolução. É bastante simples escrever soluções para problemas complexos com a programação com restrições no entanto otimizar a rapidez dessas soluções pode ser algo bastante complexo.*

**Keywords:** kropki, prolog, feup, sictus, plog

## 1 Introdução

Foi proposto pelos docentes da unidade curricular Programação em Lógica o desenvolvimento de um programa em *Prolog* com restrições para a resolução de problemas de otimização ou decisão combinatória. De entre as várias opções, decidimos desenvolver o *Kropki*, um jogo semelhante ao *sudoku* porém com novas e diferentes regras.

O artigo está organizado primeiramente pela descrição do problema onde é estudado e explicado o problema em questão, seguidamente é apresentada a abordagem ao problema de satisfação de restrições. Subdividindo-se em variáveis de decisão, restrições, função de avaliação e estratégia de pesquisa. De seguida é apresentada a solução e para finalizar os resultados e conclusões.

## 2 Descrição do Problema

O problema é muito semelhante ao *Sudoku*. Em ambos os jogos os números de uma coluna e linha têm de ser diferentes. No entanto a regra dos quadrados, isto é em cada quadrado do sudoku não pode haver nenhum número de 1 a 9 repetido, no *Kropki* isto não se verifica.

Se no *Sudoku* tradicional algumas casas já estão preenchidas com números de modo a reduzir as soluções, no caso do *Kropki* apenas são mostrados círculos **brancos** e **pretos**.

Um círculo **branco** entre duas casas significa que contém números consecutivos (isto é, 1 e 2, 2 e 3, 3 e 4, 4 e 5, 5 e 6, 6 e 7, 7 e 8 ou 8 e 9). Caso o circulo seja **preto** entre duas casas significa que um dos número é exatamente o dobro do outro (1 e 2, 2 e 4, 3 e 6 ou 4 e 8). No caso de o número um e o número dois serem consecutivos então tanto pode ser um circulo **branco** como **preto** entre as casas.

Para diferentes tamanhos o números a colocar variam de 1 até N. Sendo N o tamanho do tabuleiro quadrado.

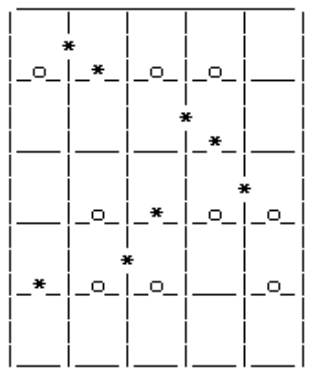

**Figura 1.** Exemplo de tabuleiro de tamanho 5, em estado inicial.

Conclui-se que entre casas que não possuem círculos, os números não são consecutivos nem o dobro um do outro.

# 3 Abordagem

## 3.1 Variáveis de Decisão

Neste jogo as variáveis de decisão correspondem às células do jogo. O objetivo é então decidir que número colocar na variável. O domínio da variável corresponde então à gama de valores que uma célula pode ter. O domínio varia de 1 a N, sendo N o tamanho do tabuleiro NxN.

## 3.2 Restrições

As retrições utilizadas neste jogo são as seguintes:

- Todos os elementos de uma linha têm de ser diferentes (aplicar all different a todas as linhas);
- Todos os elementos de uma coluna têm de ser diferentes (aplicar all different as linhas da matriz transposta do tabuleiro);
- Todos os elementos que estejam numa célula com um ponto preto à sua direita tem de ter um valor do dobro ou metade da célula da direita;
- Todos os elementos que estejam numa célula com um ponto branco à sua direita tem de ter um valor de mais ou menos 1 unidade da célula da direita;
- Todos os elementos que estejam numa célula com um ponto preto em baixo tem de ter um valor do dobro ou metade da célula imediatamente abaixo;
- Todos os elementos que estejam numa célula com um ponto branco em baixo tem de ter um valor de mais ou menos 1 unidade da célula imediatamente abaixo...

## 3.3 Função de Avaliação

Não é necessário uma função Prolog para avaliar a solução obtida. Como se trata de um jogo esta pode ser verificada visualmente.

## 3.4 Estratégia de Pesquisa

Optou-se por utilizar a estratégia ffc do labeling para ordenação das variáveis. Esta estratégia consiste em ordenar as variáveis da com mais restrições para a que tem menos restrições. Torna-se uma estratégia adequada pois assim reduz-se o número de backtrackings.

Para a ordenação de valores utilizou-se a padrão do labeling, ou seja da menor para a menor dos valores do domínio.

# 4 Visualização da Solução

Para a visualização da solução utilizamos o predicado printBoard(+Board) (+Board é uma lista de listas que representa o tabuleiro) do ficheiro que percorrer todas as linhas do tabuleiro desenhando-as de forma amigável ao utilizador.

Este predicado socorre-se de outros sendo que o mais importante é o printElem(+Elem) em que se o elemento a ser escrito na consola for um número este é escrito senão for a célula fica em branco. Isto permite que se passada uma lista sem valores atribuídos a variáveis, o predicado imprime um tabuleiro em branco.

Podemos ver dois exemplos de visualização:

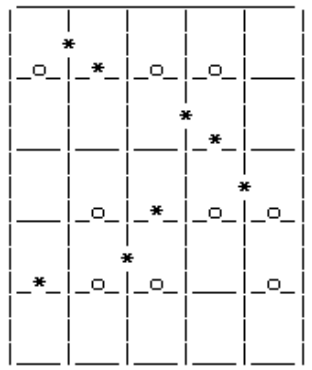

**Figura 2.** Exemplo de tabuleiro de tamanho 5, em estado inicial.

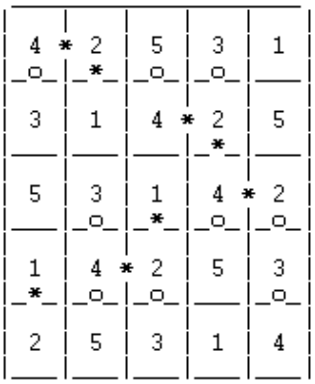

**Figura 3.** Exemplo de tabuleiro de tamanho 5, com solução.

## 5 Resultados

Após vários testes com vários tamanhos de tabuleiros, verificou-se que para um tamanho menor ou igual a oito o programa resolve o problema no tempo de um click. No entanto para valores de tamanho superiores a dez não é possível encontrar uma solução em tempo útil.

Decidiu-se então fazer um estudo sobre quais as opções de labeling são as mais eficazes para resolver o nosso jogo variando a estratégia de ordenação de variáveis e a estratégia de branching. As medições tiveram em conta apenas 10 valores para cada opção. Obtiveram-se os seguintes resultados:

| TAMANHO 9x9 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| TEMPO | | | | | | | | | | | |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Média |
| **ffc step** | 1,478 | 0,247 | 0,638 | 85,657 | 0,2 | 8,442 | 150,349 | 80,251 | 0,029 | 25,025 | 35,23 |
| **ff step** | 5,946 | 0,445 | 27,325 | 6,272 | 142,83 | 10,544 | 1,221 | 0,368 | 594,512 | 50,436 | 83,99 |
| **ffc enum** | 9,716 | 0,092 | 8,591 | 139,894 | 71,746 | 1,204 | 6,85 | 0,347 | 0,843 | 2,672 | 24,20 |
| **ffc bisect** | 23,705 | 3,272 | 3,54 | 15,864 | 3,882 | 0,117 | 1,39 | 13,024 | 9,103 | 35,133 | 10,90 |
| **ff bisect** | 62,314 | 134,573 | 4,077 | 13,611 | 11,081 | 188,007 | 2,384 | 6,454 | 2,355 | 12,797 | 43,77 |

**Figura 4.** Resultados do tempo médio com as diversas opções de labeling.

| TAMANHO 9x9 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| BACKTRACKS | | | | | | | | | | | |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Média |
| **ffc step** | 11194 | 2816 | 5639 | 702860 | 1823 | 67846 | 1369912 | 632593 | 196 | 196835 | 299171,40 |
| **ff step** | 59334 | 3436288 | 5627887 | 13800758 | 1241408 | 113387 | 10012 | 2752 | 5114182 | 7070472 | 3647648,00 |
| **ffc enum** | 93833 | 2327656 | 97294 | 1637644 | 724029 | 16429 | 7768343 | 3697 | 8410 | 22150 | 1269948,50 |
| **ffc bisect** | 149241 | 20636 | 34410 | 133231 | 26917 | 765 | 11366 | 6156452 | 74163 | 267136 | 687431,70 |
| **ff bisect** | 579528 | 1599184 | 36945 | 11246433 | 42097753 | 2119704 | 31323 | 70734 | 3818029 | 3764037 | 6536367,00 |

**Figura 5.** Resultados dos backtrackings com as diversas opções de labeling.

| TAMANHO 9x9 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CONSTRAINTS | | | | | | | | | | | |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Média |
| **ffc step** | 311 | 261 | 276 | 276 | 316 | 291 | 261 | 246 | 281 | 276 | 279,50 |
| **ff step** | 291 | 602 | 4569661 | 1034 | 326 | 276 | 321 | 331 | 256 | 573 | 457367,10 |
| **ffc enum** | 276 | 753 | 286 | 266 | 271 | 266 | 532 | 286 | 281 | 241 | 345,80 |
| **ffc bisect** | 321 | 286 | 246 | 251 | 281 | 316 | 291 | 281 | 256 | 276 | 280,50 |
| **ff bisect** | 261 | 216 | 306 | 1044 | 1325 | 291 | 256 | 276 | 813 | 592 | 538,00 |

**Figura 6.** Resultados das contraints com as diversas opções de labeling.

Conclui-se então que as melhores opções são ffc com bisect.

## 6 Conclusões e Trabalho Futuro

Este projeto mostrou-se essencial para uma melhor compreensão do mecanismo por de trás da programação em lógica com restrições, estudado na unidade curricular *PLOG*, pois foi possível de uma forma prática estudar e testar conceitos através do desenvolvimento do jogo *Kropki*.

Conclui-se que é possível escrever programas para resolver problemas algo complexos em poucas linhas de código e que o Prolog é uma ferramenta muito poderosa neste tipo de situações.

Os resultados obtidos são bastante satisfatórios, visto que um problema de alguma complexidade consegue ser resolvido em relativo pouco tempo. No entanto a solução apresentada tem um limite máximo de tamanho para os tabuleiros. Estes não podem ser de tamanho superior a 9x9 visto que, não é apresentada uma solução em tempo útil.

Para além desta limitação o trabalho não precisa de melhoramentos significativos.

## 7 Referências

## Referências

1. Puzzle Ramayan, `http://logicmastersindia.com/lmitests/dl.asp?attachmentid=532&view=1`

## Apêndice

**dots.pl**

```
1  createDots(_, Size, Size, Size).
2
3  createDots(Board, Size, Col, Size):-
4    Col1 is Col + 1,
5    getCell(Board, Size, Col, Elem),
6    getCell(Board, Size, Col1, ElemRight),
7    dotRight(Size, Col, Elem, ElemRight),
8    createDots(Board, Size, Col1, Size).
9
10 createDots(Board, Row, Size, Size):-
11   Row1 is Row + 1,
12   getCell(Board, Row, Size, Elem),
13   getCell(Board, Row1, Size, ElemDown),
14   dotDown(Row, Size, Elem, ElemDown),
15   createDots(Board, Row1, 1, Size).
16
17 createDots(Board, Row, Col, Size):-
18   Row1 is Row + 1,
19   Col1 is Col + 1,
20   getCell(Board, Row, Col, Elem),
21   getCell(Board, Row, Col1, ElemRight),
22   getCell(Board, Row1, Col, ElemDown),
23   dotRight(Row, Col, Elem, ElemRight),
24   dotDown(Row, Col, Elem, ElemDown),
25   createDots(Board, Row, Col1, Size).
26
27 dotRight(Row, Col, Elem, ElemRight):-
28   Elem1 is Elem + 1,
29   ElemRight1 is ElemRight + 1,
30   Elem2 is Elem * 2,
31   ElemRight2 is ElemRight * 2,
32    if(Elem2 == ElemRight, assert(hor(Row,Col,black)), true),
33    if(Elem == ElemRight2, assert(hor(Row,Col,black)), true),
34    if(Elem1 == ElemRight, assert(hor(Row,Col,white)), true),
35    if(Elem == ElemRight1, assert(hor(Row,Col,white)), true).
36
37 dotDown(Row, Col, Elem, ElemDown):-
38   Elem1 is Elem + 1,
39   ElemDown1 is ElemDown + 1,
40   Elem2 is Elem * 2,
41   ElemDown2 is ElemDown * 2,
42    if(Elem2 == ElemDown, assert(ver(Row,Col,black)), true),
```

```
43    if(Elem == ElemDown2, assert(ver(Row,Col,black)), true),
44    if(Elem1 == ElemDown, assert(ver(Row,Col,white)), true),
45    if(Elem == ElemDown1, assert(ver(Row,Col,white)), true).
```

**interface.pl**

```
1  getChar(Input):-
2          get_char(Input),
3          get_char(_), !.
4
5  getInt(Input):-
6          get_code(TempInput),
7          Input is TempInput - 48,
8          get_code(_),
9           !.
10
11 waitForEnter:-
12         get_char(_).
13
14 printSeparator(1, 1):-
15         write(' --- ').
16 printSeparator(Size, Size):-
17         write('----').
18 printSeparator(1, Size):-
19         write(' --- '),
20         printSeparator(2, Size).
21 printSeparator(Index, Size):-
22         write('----'),
23         Index1 is Index + 1,
24         printSeparator(Index1, Size).
25
26 printLimits(1,1):-
27         write('|   |').
28 printLimits(1,Size):-
29         write('|   |'),
30         printLimits(2,Size).
31 printLimits(Size,Size):-
32         write('   |').
33 printLimits(Index,Size):-
34         write('   |'),
35         Index1 is Index + 1,
36         printLimits(Index1,Size).
37
38 printInitialSeparator (Size):- printSeparator(1,Size).
```

```
39
40 printElem(Elem):-
41         number(Elem),
42         write(Elem).
43
44 printElem(_):-
45         write(' ').
46
47 dotR(Row,Col):-
48         hor(Row,Col,Color),
49         if(Color == black, write('*'), write('o')).
50
51 dotR(_,_):-
52         write('|').
53
54 dotC(Row,Col):-
55         ver(Row,Col,Color),
56         if(Color == black, write('*'), write('o')).
57
58 dotC(_,_):-
59         write('_').
60
61 printRow([],_,_).
62 printRow([Elem|Rest],Row,Col):-
63         write(' '),
64         printElem(Elem),
65         write(' '),
66         dotR(Row,Col),
67         Col1 is Col + 1,
68         printRow(Rest, Row, Col1).
69
70 printDownLimits(Row, Size, Size):-
71         write('_'),
72         dotC(Row, Size),
73         write('_|').
74 printDownLimits(Row, Col, Size):-
75         write('_'),
76         dotC(Row, Col),
77         write('_|'),
78         Col1 is Col + 1,
79         printDownLimits(Row,Col1, Size).
80
81 printBoardAux([], _, _).
82 printBoardAux([Row|Tails], R, Size):-
83         printLimits(1,Size),nl,
```

```
84          write('|'),
85          printRow(Row, R, 1),
86          nl,
87          write('|'),
88          printDownLimits(R, 1, Size),
89          nl,
90          R1 is R+1,
91          printBoardAux(Tails, R1,Size).
92
93
94  printBoard(Board):-
95          length(Board, Size),
96           printInitialSeparator (Size),nl,
97          printBoardAux(Board,1,Size).
```

**kropki.pl**

```
1   :- include('interface.pl').
2   :- include('menus.pl').
3   :- include('lists.pl').
4   :- include('restrictions.pl').
5   :- include('dots.pl').
6
7
8   kropki:-
9     sizeMenu,
10    write('Size: '),
11    getInt(Size),
12    play(Size).
13
14  play(Size):-
15    retractall(hor(_,_,_)),
16          retractall(ver(_,_,_)),
17    retractall(time(_)),
18    newBoard(Size,Board),
19    generateBoard(Board, New),
20    createDots(New, 1, 1, Size),
21    newBoard(Size,SolBoard),
22          write('\33\[2J'),
23    printBoard(SolBoard),
24    solveBoard(SolBoard, Solved),
25    printBoard(Solved),
26     fd_statistics ,
27    time(T),
```

```
28   format('~n It took ~3d sec to solve this board.~n', T).
```

**lists.pl**

```
1  :- use_module(library( lists )).
2
3
4  newBoard(Size, Board):- newBoardAux(Size, Size, Board).
5
6  newBoardAux(0,_,[]).
7  newBoardAux(Pos, Size, [Head | Tails]):-
8    Pos > 0,
9    length(Head, Size),
10   Pos1 is Pos - 1,
11         newBoardAux(Pos1, Size, Tails).
```

**menus.pl**

```
1  mainMenu:-
2      write('\33\[2J'),
3      write('*****************************************'), nl,
4      write('********************* *********************'), nl,
5      write('******************** ********************'), nl,
6      write('********************   ******************'), nl,
7      write('******************     ******************'), nl,
8      write('****************       ****************'), nl,
9      write('***************         ***************'), nl,
10     write('***************         ***************'), nl,
11     write('**************           **************'), nl,
12     write('************             ************'), nl,
13     write('************             ************'), nl,
14     write('**********               **********'), nl,
15     write('*********                 *********'), nl,
16     write('********         KROPKI!!      ********'), nl,
17     write('*******                     *******'), nl,
18     write('********      Choose a size      ********'), nl,
19     write('*********                   *********'), nl,
20     write('**********                  **********'), nl,
21     write('************                ************'), nl,
22     write('************                ************'), nl,
23     write('**************              **************'), nl,
24     write('***************             ***************'), nl,
25     write('*****************           ***************'), nl,
```

```
26      write('*****************          *****************'), nl,
27      write('******************       ******************'), nl,
28      write('*******************    *******************'), nl,
29      write('******************** ********************'), nl,
30      write('********************* *********************'), nl,
31      write('*****************************************'), nl.
```

**restrictions.pl**

```
1  :- use_module(library(clpfd)).
2  :- use_module(library(random)).
3
4  setDomains([], _).
5  setDomains([H|T], Size):-
6    domain(H, 1, Size),
7    setDomains(T, Size).
8
9  rowRestrictions ([]) .
10 rowRestrictions([H|T]):-
11    all_different (H),
12   rowRestrictions(T).
13
14
15
16 randomVar(Vars,Var,Remaining) :-
17   random_select(Var,Vars,Remaining),
18           var(Var).
19
20 label ([]) .
21 label ([H|T]):-
22   labeling ([variable(randomVar)], H),
23   label(T).
24
25 generateBoard(Board,NewBoard):-
26   length(Board, Size),
27   setDomains(Board, Size),
28   rowRestrictions(Board),
29   transpose(Board, NewBoard1),
30   rowRestrictions(NewBoard1),
31   transpose(NewBoard1, NewBoard),
32   label(NewBoard).
33
34 getCell(Board, Row, Col, Cell):-
35   nth1(Row, Board, Elem),
```

```
36    nth1(Col, Elem, Cell).
37
38  double(Elem, Elem2):-
39    Elem #= Elem2 * 2 #\/ Elem2 #= Elem * 2.
40
41  consecutive(Elem, Elem2):-
42    Elem #= Elem2 + 1 #\/ Elem2 #= Elem + 1.
43
44   verticalRestricitons (_, Size, _, Size).
45   verticalRestricitons (Solved, Row, Size, Size):-
46    Row1 is Row + 1,
47    (ver(Row,Size,Color) ->
48      getCell(Solved, Row, Size, Elem),
49      getCell(Solved, Row1, Size, ElemDown),
50      if(Color == black, double(Elem, ElemDown), consecutive(Elem,
            ElemDown));
51      true),
52     verticalRestricitons (Solved, Row1, 1, Size).
53
54   verticalRestricitons (Solved, Row, Col, Size):-
55    Col1 is Col + 1,
56    Row1 is Row + 1,
57    (ver(Row,Col,Color) ->
58      getCell(Solved, Row, Col, Elem),
59      getCell(Solved, Row1, Col, ElemDown),
60      if(Color == black, double(Elem, ElemDown), consecutive(Elem,
            ElemDown)); true),
61     verticalRestricitons (Solved, Row,Col1,Size).
62
63   horizontalRestricitons(_, Size, Size, Size).
64
65   horizontalRestricitons(Solved, Row, Size, Size):-
66    Row1 is Row + 1,
67     horizontalRestricitons(Solved, Row1, 1, Size).
68
69   horizontalRestricitons(Solved, Row, Col, Size):-
70    Col1 is Col + 1,
71    (hor(Row,Col,Color) ->
72      getCell(Solved, Row, Col, Elem),
73      getCell(Solved, Row, Col1, ElemRight),
74      if(Color == black, double(Elem, ElemRight), consecutive(Elem,
            ElemRight));
75      true),
76     horizontalRestricitons(Solved, Row,Col1,Size).
77
```

```
78
79
80 dotRestrictions(Solved,Size):-
81     verticalRestricitons (Solved, 1, 1, Size),
82     horizontalRestricitons (Solved, 1, 1, Size).
83
84 solveBoard(Board, Solved):-
85     length(Board,Size),
86     append(Board, L1),
87     domain(L1, 1, Size),
88     rowRestrictions(Board),
89     transpose(Board, NewBoard1),
90     rowRestrictions(NewBoard1),
91     transpose(NewBoard1, Solved),
92     dotRestrictions(Solved, Size),
93     append(Solved, L),
94     statistics(walltime,_),
95     labeling ([ ffc , bisect ], L),
96     statistics(walltime,[_,T]),
97     assert(time(T)).
```