Crazy Pavement

João Chaves (up201406225) e **Rui Araújo**(up201403263)

Mestrado Integrado em Engenharia Informática e Computação - 3º Ano
Programação em Lógica 2016/2017
Crazy Pavement Grupo 4

Faculdade de Engenharia da Universidade do Porto,
R. Dr. Roberto Frias, 4200-464 Porto, Portugal
`https://sigarra.up.pt/feup`

Resumo O trabalho foi desenvolvido no contexto do segundo projeto da Unidade Curricular de Programação em Lógica. O objetivo é a implementação das regras do puzzle ***Crazy Pavement*** utilizando a linguagem PROLOG com restrições.
O puzzle consiste em preencher um tabuleiro, dividido por áreas, de modo a que, no final, o número de células preenchidas em cada linha/coluna seja igual ao número identificado pelas pistas existentes nesta linha/coluna (se exisitirem). No entanto, ao preencher uma célula, é necessário preencher todas as outras células pertencentes a esta região.
Neste artigo mostramos a solução para três puzzles diferentes.

1 Introdução

O objetivo principal deste projeto é avaliar a capacidade de resolver um problema de otimização ou decisão, utilizando a linguagem Prolog com restrições.

O tema do grupo foi o ***Crazy Pavement***, que é um problema de decisão.

O objetivo é, a partir do tabuleiro inicial, vazio, pintar as células do tabuleiro de diferentes secções de forma a completá-lo corretamente (pintar todas as células de algumas regiões e que os números fora do tabuleiro indiquem os números das células pintadas naquela linha/coluna).

O relatório está dividido em cinco partes, a descrição do problema, a abordagem, que por sua vez apresenta as variáveis de decisão, as restrições, a função de avaliação e a estratégia de pesquisa. De seguida, a visualização da solução obtida, os resultados e, por fim, a conclusão.

2 Descrição do Problema

É apresentado um tabuleiro dividido em secções de diversas células , inicialmente em branco, que se pretende que sejam coloridas de modo que se obtenha todas as células de algumas regiões pintadas e que os números fora do tabuleiro indiquem os números das células pintadas naquela linha/coluna).

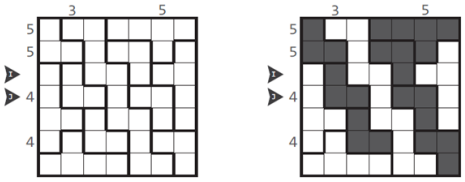

Figura 1: Exemplo de um puzzle com solução

3 Abordagem

O primeiro passo na resolução do puzzle foi tentar modelá-lo como um problema de restrições, começando por definir as variáveis de estado e pensar em todas as restrições necessárias para chegar a uma solução única e correta.

3.1 Variáveis de decisão

O tabuleiro foi dividido em linhas e, em cada linha, existem 7 células que correspondem a 7 variáveis. Logo, para um tabuleiro com 7 linhas e 7 colunas, são criadas 7 linhas e consequentemente 49 variáveis que têm domínio entre [0,1], 0 se estiver em branco e 1 se pintada.

```
1   L1 = [S1 , S2 , S2 , S3 , S3 , S3 , S3],
2   L2 = [S1 , S1 , S2 , S4 , S4 , S3 , S5],
3   L3 = [S6 , S7 , S2 , S2 , S4 , S5 , S5],
4   L4 = [S6 , S7 , S7 , S2 , S8 , S8 , S5],
5   L5 = [S6 , S6 , S9 , S2 , S2 , S8 , S5],
6   L6 = [S6 , S10, S9 , S9 , S2 , S11, S11],
7   L7 = [S10, S10, S10, S10, S2 , S2 , S11],
8
9   domain(L1, 0, 1),   domain(L5, 0, 1),
10  domain(L2, 0, 1),   domain(L6, 0, 1),
11  domain(L3, 0, 1),   domain(L7, 0, 1),
12  domain(L4, 0, 1),
```

3.2 Restrições

Para restrições são feitas contagens do numero de linhas e colunas e comparadas com as pistas dadas fora do tabuleiro para cada uma dessas linhas ou colunas. As pistas são definidas por uma lista com tamanho igual ao número de linhas e colunas e, para cada posição (que corresponde à posição da linha/-coluna do tabuleiro) estão definidas com o número de células a serem pintadas naquela linha/coluna, ou com 0, se não existe um número definido para aquela linha/coluna.

```
1   clue_line1([5,5,-1,4,-1,4,-1]).
2   clue_column1([-1, 3, -1, -1, -1, 5, -1]).
```

É usado o predicado check_lines_columns() para verificar o número de células pintadas. Este predicado é executado duas vezes, uma para as linhas e outra para as colunas, e percorre todas as linhas e colunas. Para cada linha/coluna, verifica se o número de células pintadas corresponde ao elemento correspondente da lista de pistas.

```
1   check_lines_columns([_|Ls], Lines, N):-
2     element(N, Lines, Nr),
3     Nr = -1,
4     N1 is N + 1,
5     check_lines_columns(Ls, Lines, N1).
6
7   check_lines_columns([L|Ls], Lines, N):-
8     element(N, Lines, Nr),
9     count(1, L, #=, Nr),
10    N1 is N + 1,
11    check_lines_columns(Ls, Lines, N1).
12
13  check_lines_columns([], _, _).
```

3.3 Estratégia de Pesquisa

É feito o labeling para cada uma das linhas.

4 Visualização da solução

Para a visualização em modo de texto no SICStus é usado o predicado display_full_board() que chama display_board() para mostrar o conteúdo do tabuleiro através do predicado display_line() e escreve, de seguida, os caracteres para construir os espaços do tabuleiro. Para mostrar os símbolos de cada célula são utilizados os predicados: translate(1, 'X') e translate(0, ' ').

```
display_f_board1([L1|Ls]):-
^^Iwrite('|'),
^^Idisplay_line(L1),
  nl,
  write(' --------------------------- '),
  nl,
^^Idisplay_f_board1(Ls).

display_f_board1([]).

display_full_board1(B):-
  nl,
  write(' --------------------------- '),nl,
  display_f_board1(B).
```

```
 ---------------------------
| X |   |   | X | X | X | X |
 ---------------------------
| X | X |   | X | X | X |   |
 ---------------------------
|   | X |   |   | X |   |   |
 ---------------------------
|   | X | X |   | X | X |   |
 ---------------------------
|   |   | X |   |   | X |   |
 ---------------------------
|   |   | X | X |   | X | X |
 ---------------------------
|   |   |   |   |   |   | X |
 ---------------------------
```

Figura 2: Exemplo de uma solução apresentada no SICSTUS

5 Resultados

Abaixo estão apresentados os resultados para cada um dos 3 puzzles presentes no programa.

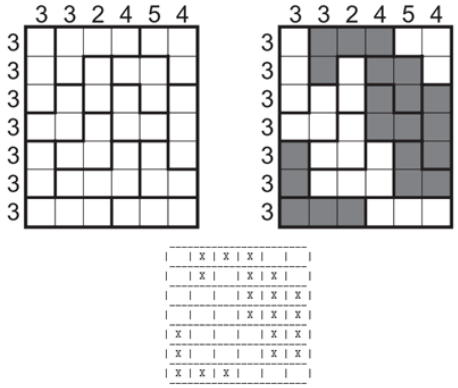

Figura 3: Solução do puzzle 7x6

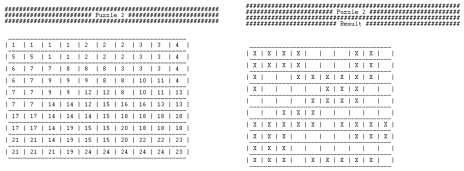

Figura 4: Solução do puzzle 10x10

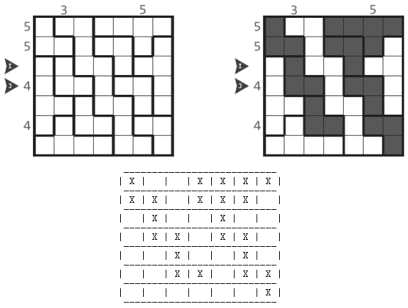

Figura 5: Solução do puzzle 7x7

6 Conclusões

Após a realização deste projeto pode-se confirmar a eficiência da linguagem Prolog na resolução de problemas de otimização ou decisão.

Os resultados obtidos demonstram 3 puzzles corretamente resolvidos. No entanto, a solução obtida tem como limitação o facto de ser necessário definir, para além das linhas, também as colunas de cada tabuleiro. Para melhorar o trabalho desenvolvido poderia ser feito um predicado para fazer visualização de todos os tabuleiros em vez de um para cada um e fazer uma verificação das colunas sem que se fosse preciso mapear em listas.

Apesar das dificuldades encontradas, de uma forma geral, o projeto foi bem sucedido e a solução implementada correspondeu às expetativas.

7 Bibliografia

Referências

1. Logic Master India, `http://logicmastersindia.com/lmitests/?test=M201309P2`
2. YoshiPuzzles, `http://yoshipuzzles.com/crazypaving-2/`
3. ShareLatex, `https://pt.sharelatex.com/learn/Inserting_Images`

Anexo

Em anexo seguem os ficheiros com código relativo ao projeto. Deve ser consultado, através do SICStus, o ficheiro main.pl, que inclui todos os outros. Para correr o programa chama-se o predicado crazy_pavement.