Implementação do Puzzle *Walls* usando Programação em Lógica com Restrições

José Francisco Cagigal da Silva Gomes e Margarida Xavier Viterbo

FEUP-PLOG, Turma 3MIEIC4, Grupo Walls\_1

Faculdade de Engenharia da Universidade do Porto,

Rua Roberto Frias, sn, 4200-465, Paranhos, Porto, Portugal

{up201305016, [up201403205}@fe.up.pt](mailto:up201403205%7d@fe.up.pt)

<http://www.fe.up.pt>

**Resumo.** Serve o presente artigo como complemento ao Segundo trabalho prático da cadeira Programação em Lógica – resolução de um problema de decisão/otimização usando Programação em Lógica com Restrições. Neste caso especifico o problema trata-se de um puzzle 2D, de nome *Walls*, cujo objetivo é colocar linhas verticais ou horizontais em todas as células vazias. As células não vazias contêm números que indicam o número total de segmentos conectados a si (que tocam nas paredes de cada uma das células em causa).

**Keywords:** problemas de decisão, puzzle, PLR, SICStus-Prolog

1. Introdução

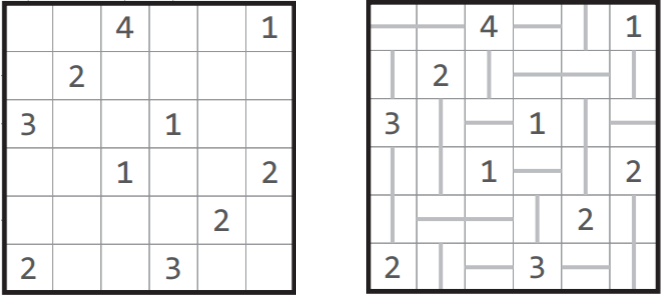
O projeto adotado pelo grupo, Walls, para o segundo trabalho prático da cadeira Programação em Lógica, tem como objetivo fundamental dotar os alunos de uma maior capacidade para resolver problemas de decisão, bem como aprofundar o conhecimento a aumentar a aptidão para programação em lógica usando restrições. Através da realização deste projeto os alunos ganham uma destreza para a utilização da bilblioteca ‘clpfd’ presente no SICStus-Prolog (biblioteca-chave para programação em lógica com restrições – PLR).

O desafio em questão, resumido anteriormente, será descrito abaixo. Consta também neste artigo a abordagem utilizada para a resolução do problema, bem como as restrições, função de avaliação e a estratégia de pesquisa. São descritos também a visualização da solução, os resultados e as conclusões.

1. Descrição do Problema

O puzzle Walls consiste numa grelha quadrada dividida em células quadradas. Algumas das células contêm números inscritos, outras encontram-se em branco. O tamanho (número de células do tabuleiro) pode variar, tal como os números e a sua disposição, podendo desta forma “jogar-se” com diferentes tabuleiros, criando-se diferentes desafios.

Para resolver o problema é necessário preencher todas as células vazias com traços verticais ou horizontais, de forma que cada célula com número tenha a apontar para si um comprimento total de traços igual ao número, estes traços não têm necessariamente de estar nas células adjacentes à célula numerada, basta que um deles esteja e que os restantes estejam ligados a ele. De forma a esclarecer melhor as regras do puzzle mostra se abaixo uma figura com um possível puzzle por resolver e a respetiva resolução.



**Fig. 1.** Exemplo de um possível puzzle (esquerda) e sua resolução (direita)

1. Abordagem
   1. Variáveis de Decisão
   2. Restrições
   3. Função de Avaliação
   4. Estratégia de Pesquisa
2. Visualização da Solução

Para visualização dos tabuleiros em modo de texto foram implementados os seguintes predicados: (só no fim é que cheguei à conclusão que não devo ter explicado bem os predicados!!!!)

* ***display\_board***: implementa os predicados *display\_line* e *display\_separator* de forma recursiva;

display\_board([L1|L2,Size) :-

diplay\_line(L1), write(':'), nl,

display\_separator(Size,1), display\_board(L2,Size).

display\_board([],\_).

* ***display\_line***: imprime o conteúdo de uma linha, escrevendo todos os seus elementos um por um, recursivamente;

display\_line([L1|L2]) :-

write(':'), (var(L1), write(' '), !;

write(L1)), display\_line(L2).

display\_line([]).

* ***display\_separator***: imprime os pontos que fazem separação das células de acordo com o número de células do tabuleiro;

display\_separator(Size, It) :-

It=<Size, !, write('..'), It1 is It+1,

display\_separator(Size,It1).

display\_separator(\_, \_) :- write('.'),nl.

* ***draw\_solution/draw\_solution\_line***: equivalentes aos predicados *display\_board* e *display\_line* mas aplicados ao tabuleiro da solução;

draw\_solution([L1|L2], [L3|L4], Size) :-

draw\_solution\_line(L1, L3), write(':'), nl,

display\_separator(Size, 1), draw\_solution(L2,L4,Size).

draw\_solution([], [], \_).

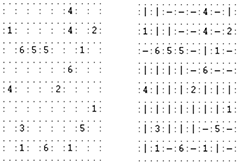
draw\_solution\_line([L1|L2], [L3|L4]) :-

write(':'), (var(L1), (L3==1, write('-'), !;

write('|')), !; write(L1)), draw\_solution\_line(L2,L4).

draw\_solution\_line([], []).

Estes predicados de visualização imprimem o seguinte:



**Fig. 2.** Visualização em modo texto dos tabuleiros inicial (esquerda) e final (direita) do puzzle

1. Resultados
2. Conclusões

Com a realização deste trabalho foi possível perceber a utilidade prática da programação em lógica com restrições e de que forma esta pode facilitar imenso e resolução de problemas de decisão. Foi igualmente conseguida uma ambientação e destreza claras com este tipo de programação e bibliotecas associadas.

Embora no início se tenha revelado um pouco complicado alterar o raciocino lógico do trabalho anterior e procurar soluções usando restrições, ao começar a ser explorada esta nova forma de programação e *prolog*, verificou-se a enorme diferença entre o resultado da implementação do puzzle com e sem restrições, tendo-se facilmente percebido a importância da PLR.

O problema proposto foi resolvido com sucesso e generalizado para todos os casos. O que começou por ser uma resolução bastante demorada passou foi otimizado e é agora uma solução eficiente para o puzzle em questão.

1. Bibliografia

[1] Regras do puzzle *Walls*,

<http://logicmastersindia.com/lmitests/dl.asp?attachmentid=523>

[2] SICStus Prolog, <https://sicstus.sics.se/>

[3] SWI-Prolog, <http://www.swi-prolog.org/>

[4] Elaboração do artigo,

<ftp://ftp.springer.de/pub/tex/latex/llncs/latex2e/instruct/authors/authors.pdf>

Anexos

Código-fonte segue nos ficheiros *main.pl*, *aula.pl* e *generator.pl*, anexadas ao artigo.