Symmetry Puzzle

L.N.N. Almeida[201706908] e J.H.A.M. Reguengo da Luz[201703782]

Faculdade de Engenharia da Universidade do Porto
FEUP-PLOG, Turma 3MIEIC5, Grupo Symmetry_2

Resumo Este projeto foi desenvolvido no âmbito da unidade curricular de Programação Lógica com o objetivo de solucionar problemas de otimização/decisão. O problema selecionado tem como finalidade o preenchimento de um puzzle com algumas formas geométricas, tal que, as linhas e as colunas deste resultem em palíndromos. Foram testadas diferentes estratégias de pesquisa, assim como diferentes dimensões do tabuleiro, tendo-se concluido que a estratégia utilizada não é a mais eficiente e que esta diminui muito com o aumento do tamanho do tabuleiro.

Keywords: Prolog · SICStus · Problemas de Decisão · Restrições · Palíndromo · Autómato.

1 Introdução

Este projeto, desenvolvido no âmbito da unidade curricular de Programação Lógica, tem como objetivo obter uma solução válida para um problema de decisão. Este problema é o puzzle Symmetry que consiste em preencher o tabuleiro com formas geométricas diferentes, de modo a que cada linha e coluna seja um palíndromo.

O projeto é constituído, então, em duas partes: a pesquisa de uma solução para um problema e a geração aleatória de um problema.

A estrutura deste artigo é a seguinte:

- **Descrição do Problema:** descrição com detalhe do problema de otimização ou decisão em análise.
- **Abordagem:** descrição da modelação do problema como um PSR, de acordo com as seguintes subsecções:
  - **Variáveis de Decisão:** descrição das variáveis de decisão, dos seus domínios e do seu significado no contexto do problema em análise.
  - **Restrições:** descrição das restrições rígidas e flexíveis do problema e da sua implementação utilizando o SICStus Prolog.
  - **Estratégia de Pesquisa:** descrição da estratégia de etiquetagem (labeling) utilizada ou implementada, nomeadamente heurísticas de ordenação de variáveis e valores.
- **Visualização da Solução:** explicação dos predicados que permitem visualizar a solução em modo de texto.

- **Resultados:** Exemplificar a aplicação em instâncias do problema com diferentes dimensões e complexidade, bem como a análise dos resultados obtidos.
- **Conclusões e Trabalho Futuro:** indicação das conclusões retiradas do projeto, análise dos resultados obtidos e indicação das vantagens e limitações da solução proposta, assim como, de possíveis aspetos a melhorar.

2 Descrição do Problema

O puzzle Symmetry é um problema de decisão. Tem como base um tabuleiro de tamanho variável com espaços vazios e espaços preenchidos com uma das seguintes formas geométricas: um triângulo, um círculo ou um quadrado.

Este problema consiste, então, em preencher alguns dos espaços vazios com as formas geométricas referidas, tal que, no final, ignorando os espaços vazios restantes, cada linha e coluna seja um palíndromo, isto é, que cada linha e coluna tenha a mesma sequência de formas geométricas quer seja lida num sentido ou no outro.

3 Abordagem

Para solucionar este problema recorreu-se a um autómato com um contador. Este autómato vai receber todas as linhas e, posteriormente, todas as colunas após a matriz do tabuleiro ter sido transposta.

Para encontrar uma solução para um problema, é calculado o tamanho da matriz do tabuleiro fornecido e, de seguida, todos os espaços vazios são substituídos por variáveis, executando, por fim, as restrições necessárias e o processo de etiquetagem.

No caso da geração aleatória de problemas, é criada uma matriz de tabuleiro com o tamanho pretendido e, após realizar o mesmo processo que a situação anterior, é obtida uma solução aleatória de um puzzle. Para chegar a um problema com essa solução, é escolhido um número aleatório de posições que vão ser acedidas e para cada uma dessas posições é escolhida uma linha e uma coluna aleatória, na qual é colocado um espaço vazio, independentemente do seu valor anterior.

3.1 Variáveis de Decisão

A solução do problema traduz-se numa lista de listas representante do tabuleiro *Lines*, posteriormente, convertida a uma lista unidimensional *LinesUni*. Para o problema ser resolvido é ainda necessário o auxílio de duas listas *CountersL* e *CountersC* que contêm os contadores resultantes do autómato de todas as linhas e colunas, respetivamente.

O domínio da lista *LinesUni* é de 0 a 3, tal que, 0 corresponde a um espaço em branco no tabuleiro e os restantes aos três tipos diferentes de formas geométricas das peças: quadrado, triângulo e circulo, por esta ordem. Quanto às listas dos

contadores, podem conter valores entre 0 e uma sequência de N números 3 em que N é o número de linhas/colunas do tabuleiro e 3 o maior valor que se pode encontrar numa posição do tabuleiro.

3.2 Restrições

A principal restrição do problema é que cada linha e cada coluna do tabuleiro tem de formar um palíndromo, não sendo incluídos os espaços vazios.

Para tal, recorreu-se a um autómato com um contador que, recebendo uma lista, cria um número representativo dos símbolos diferentes de zero, que corresponde à sequência de símbolos presentes numa fila ou coluna sem os espaços em branco. Para formar este número, a cada símbolo diferente de zero, o valor do contador é multiplicado por 10 e a este é somado o valor do símbolo. O código relativo ao autómato encontra-se na secção A.3 dos Anexos.

A verificação de que uma lista, pertencente a uma linha ou coluna, obdece à restrição exigida pelo problema é feita enviando esta lista para o autómato e, posteriormente, enviando também a lista invertida da mesma, sendo necessário garantir que, no final, os números resultantes dos dois contadores são iguais.

Este processo tem de ser realizado para todas as linhas e colunas do tabuleiro.

3.3 Estratégia de Pesquisa

O processo de etiquetagem foi implementado com a opção *leftmost*, por omissão, como forma de selecionar a próxima variável, ou seja, selecionando a variável mais à esquerda de entre todas as possíveis.

Quanto à forma como são selecionados os valores para uma variável, foi utilizada a opção *enum*, com recurso a uma heurística de seleção aleatória *selRandom*. Esta atribui um valor aleatório a cada uma das variáveis, de entre os possíveis resultanto, no final, num tabuleiro com uma solução aleatória, no caso de haver mais do que uma.

4 Visualização da Solução

A visuzalização da solução em modo de texto é toda realizada no ficheiro **display.pl**, sendo despoletada pelo predicado `display(+Board, +N)`, tal que N é o número de linhas/colunas do tabuleiro.

Como o predicado recebe uma matriz, tranforma-a numa lista unificada passando, após alguns predicados que auxiliam na visuzalização do tabuleiro, ao predicado `display_board(+Board, +Column, +N, +Line)`. Este percorre as N linhas do tabuleiro e para cada uma delas percorre as N colunas imprimindo, no ecrã, o caracter correspondente ao símbolo que se encontra nessa posição. Na secção A.2 dos Anexos é possível encontrar os principais predicados para este processo.

Há que considerar, ainda, dois tipos de visualizações diferentes: a solução para um problema fornecido e um problema gerado aleatoriamente.

No caso de ser necessária a solução para um problema, esta é apenas mostrada no ecrã como referida anteriormente. Já no caso de ter sido pedido a geração de um problema, além de apresentado o problema como no caso anterior, também é impressa a lista de listas correspondente para poder ser reutilizada no cálculo da solução desse problema gerado. Nas figuras seguinte, pode-se visualizar o resultado final nas duas situações:

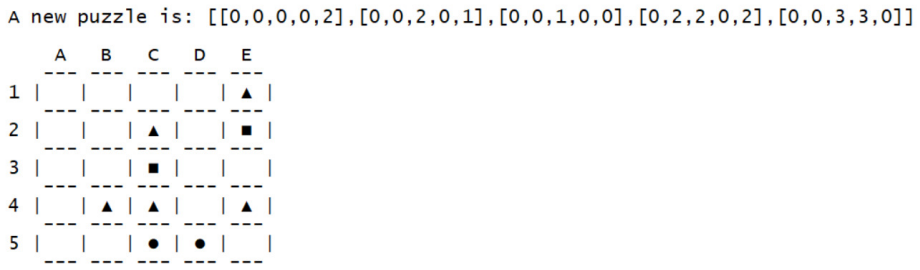

Figura 1. Visualização de um problema gerado aleatoriamente

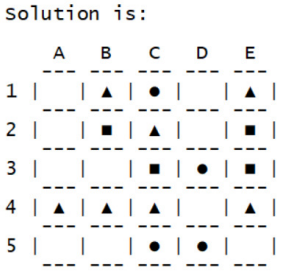

Figura 2. Visualização de uma solução para o problema obtido anteriormente

5 Resultados

Para a resolução deste problema, foram testadas várias opções de pesquisa, assim como a heurística para a seleção de valores aleatórios, com a dimensão do tabuleiro constante e igual a 6. Os resultados desses testes estão apresentados na seguinte tabela:

Tabela 1. Resultados dos testes com as diferentes estratégias de pesquisa.

	leftmost	min	ff	occurrence	ffc	max_regret
step	0,01	0,01	0,01	0,01	0,01	0,01
enum	0,01	0,01	0,01	0,01	0,01	0,01
bisect	0,01	0,01	0,02	0,01	0,01	0,01
median	0,07	0,08	0,04	0,07	0,04	0,07
middle	0,09	0,07	0,05	0,06	0,03	0,07
random	0,07	0,14	0,11	0,05	0,09	0,06

Em adição, foi também testada a solução para um problema com diferentes dimensões, estando os resultados nos gráficos das Figuras 3, 4 e 5 e na Tabela 2 dos Anexos.

6 Conclusões e Trabalho Futuro

A realização deste projeto possibilitou a implementação de programação em lógica com restrições, na resolução de problemas de decisão e de otimização, revelando a sua importância e utilidade.

De entre as opções de pesquisa testadas, no geral, o *step* e o *enum* foram as mais eficientes relativamente à seleção de valores, independentemente da forma como as variáveis eram ordenadas. Já nas opções utilizadas para selecionar a variável seguinte não existiu nenhum destaque entre as demais.

A heurística utilizada para selecionar valores aleatórios revelou-se menos eficiente em praticamente todas as situações, sendo o tempo de execução menor quando aplicada em conjunto com a opção *occurence*.

Com o aumento da dimensão do problema, tanto o tempo de execução como o número de retrocessos sofrem um aumento drástico, enquanto que o número de restrições criadas também tem um aumento embora não tão acentuado.

A solução aplicada, através de um autómato, tornou-se a mais simples, por outro lado, a estratégia de pesquisa utilizada não foi a mais eficiente.

Em suma, o projeto foi desenvolvido com sucesso, sendo possível obter uma solução para o problema do puzzle Symmetry, além de também ser possível gerar aleatoriamente problemas deste tipo com diferentes complexidades.

Referências

1. Symmetry Puzzles, https://www2.stetson.edu/~efriedma/puzzle/symmetry/. Last accessed 4 Jan 2020

A Anexos

A.1 Tabelas e gráficos

Figura 3. Tempo, em segundos, em função da dimensão do problema.

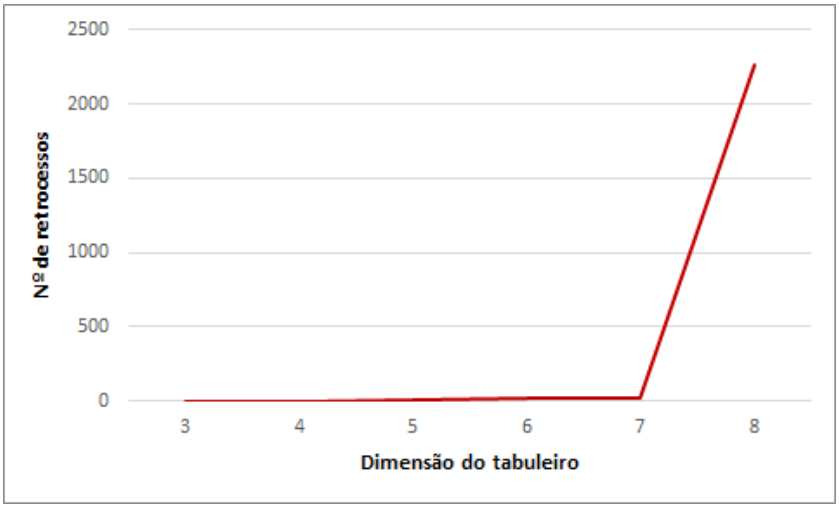

Figura 4. Número de retrocessos realizados em função da dimensão do problema.

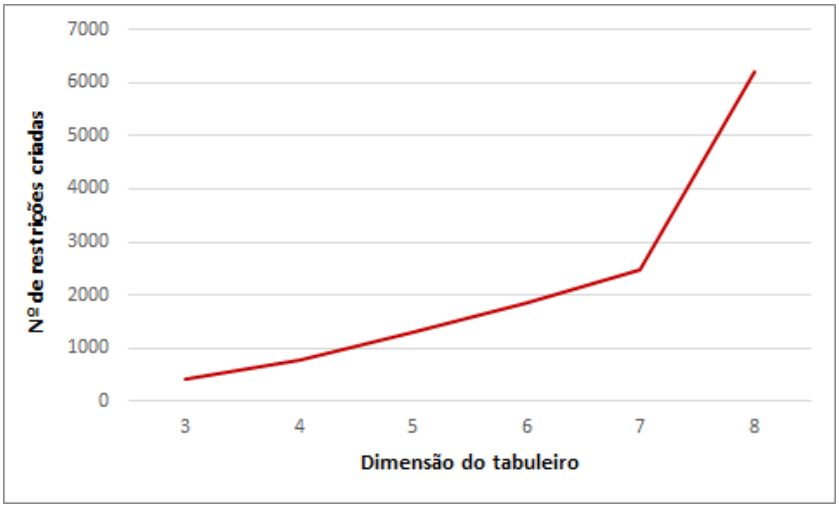

Figura 5. Número de restrições criadas em função da dimensão do problema.

Tabela 2. Resultados dos testes com as diferentes dimensões do problema.

Dimensão	Tempo	Retrocessos	Restrições
3	0	1	434
4	0	3	783
5	0,01	9	1297
6	0,07	23	1852
7	0,33	23	2474
8	4,66	2257	6204

A.2 Código-fonte referente à visualização do tabuleiro

```
display(Board, N):-
    append(Board, BoardFlat),
    write('\n\n   '),
    display_column_number(1,N),
    write('    '),
    display_line_separation(1,N),
    display_board(BoardFlat, N, N, 1),nl.

display_board([],_, _,_):-!.
display_board([Head|Tail], 1, N, Line):-
    write('  |  '),
    cell_symbol(Head),
    write('  |  '),
    write('\n    '),
```

```
    display_line_separation(1,N),
    NewLine is Line + 1,
    display_board(Tail, N, N, NewLine).

display_board([Head|Tail], N, N, Line):-
    write(Line),
    write(' | '),
    cell_symbol(Head),
    New is N -1,
    display_board(Tail, New, N, Line).

display_board([Head|Tail], Index, N, Line):-
    write(' | '),
    cell_symbol(Head),
    New is Index -1,
    display_board(Tail, New, N, Line).
```

A.3 Código-fonte referente ao autómato

```
% Checks if List is a palindrome
check_palindrome(List, Counter1, Counter2):-
    reverse(List, ReversedList),
    palind_automaton(List, Counter1),
    palind_automaton(ReversedList, Counter2),
    Counter1 #= Counter2.

% Automaton to remove zeros from List
palind_automaton(List, Counter):-
    palind_signature(List, Sign),
    automaton(Sign, _, Sign,
        [source(i), sink(i)],
        [arc(i,0,i),
         arc(i,1,i, [C*10 + 1]),
         arc(i,2,i, [C*10 + 2]),
         arc(i,3,i, [C*10 + 3])],
        [C], [0], [Counter]).

palind_signature([],[]).
palind_signature([X|Xs], [S|Ss]) :-
    S in 0..3,
    X#=0 #=> S#=0,
    X#=1 #=> S#=1,
    X#=2 #=> S#=2,
    X#=3 #=> S#=3,
    palind_signature(Xs,Ss).
```