Resolução de um Problema de um Puzzle em Prolog com Restrições: Yin Yang

Inês Teixeira (up201404592) e **Pedro Costa**(up201403291)

Mestrado Integrado em Engenharia Informática e Computação - 3º Ano
Programação em Lógica 2016/2017
Yin Yang Grupo 1

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

Resumo Este projeto teve o objetivo de aplicar o conhecimento de Prolog com restrições lecionados durante as aulas para a resolução de um puzzle.

O puzzle escolhido pelo nosso grupo foi o ***Yin Yang***, que consiste num tabuleiro, por *default* 6x6, que tem de ser preenchido totalmente com peças brancas e pretas de forma em que num quadrado 2x2 no tabuleiro não tenha apenas só peças da mesma cor e que todos as peças da mesma cor estejam conectadas entre si verticalmente ou horizontalmente.

Através da manipulação de predicados disponibilizados pelo SICStus Prolog, mostramos neste artigo a resolução deste problema, além de algumas estatísticas e a sua análise.

1 Introdução

Este projeto, desenvolvido no âmbito de Programação em Lógica, tem como objetivo avaliar os alunos na sua capacidade de resolver um problema de otimização ou decisão, utilizando conceitos de Prolog com restrições, dando uso à biblioteca 'clpfd'.

O tema escolhido pelo grupo foi um problema de decisão, um puzzle intitulado de ***Yin Yang***.

O *Yin Yang*, é um enigma, que inicialmente tem peças brancas e pretas distribuídas aleatoriamente num tabuleiro. Sendo o objetivo deste puzzle preencher todo o tabuleiro com peças brancas e pretas de forma a que em cada quadrado 2x2 não haja peças apenas da mesma cor. Outra das regras deste puzzle é que peças da mesma cor estejam conectadas entre si na vertical ou na horizontal.

Este artigo, descreve detalhadamente a abordagem seguida pelo nosso grupo na resolução do problema, os resultados obtidos e conclusões destes mesmos.

2 Descrição do Problema

O Yin Yang é jogado no tabuleiro quadrado, por *default* 6x6, que inicialmente tem peças distribuídas aleatoriamente.

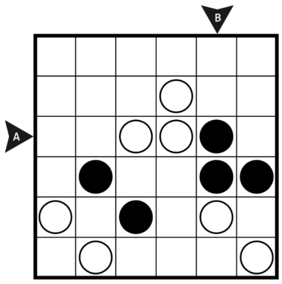


Figura 1: Exemplo de Puzzle Yin Yang

O objetivo é preencher o tabuleiro até não haver espaços sem peças e respeita todas as regras do puzzle.

Numa casa 2x2 as peças não podem ser todas da mesma cor e peças da mesma cor têm de estar conectadas horizontalmente e verticalmente.

Em baixo, está representada a solução do tabuleiro anterior.

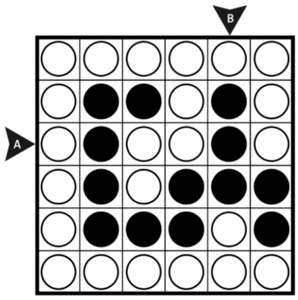


Figura 2: Exemplo de Puzzle Yin Yang

3 Abordagem

O primeiro passo na abordagem foi tentar perceber como modelar o puzzle como um problema de restrições. Entender as variáveis de decisão a usar no predicado *labeling*, as restrições necessárias para o problema e restringir essas variáveis.

Foi ainda tido em conta a melhor forma de interagir com os utilizadores, ou seja, a melhor forma de o puzzle ser visualizado. Sendo a consola do SICStus Prolog muito simples, a representação das peças é feita com X e O, em vez de peças brancas e pretas.

3.1 Variáveis de Decisão

A solução pretendida para este puzzle é o mesmo tabuleiro, mas com a resposta correta ao puzzle. No entanto, apenas uma das regras foi totalmente resolvida. Ou seja, a restrição de que a cada 2x2 quadrados não deve ter peças todas do mesmo tipo é respeitada.

Neste sentido, a única variável de domínio que o nosso problema necessita e utilizada no predicado *labeling*, é uma variável chamada L, o tabuleiro, que se trata de uma lista.

3.2 Restrições impostas

Em primeiro lugar, na inicialização da variável de decisão foi imposto que em cada célula, o domínio é 0 ou 1, que representam as peças necessárias para preencher o tabuleiro (X e O).

De seguida, foi necessário garantir que não exista num quadrado 2x2 apenas peças do mesmo tipo. Para isso, foi usado o predicado desenvolvido *square*, que coloca restrições nas células de forma a que a soma das 4 células não seja igual a 4 (4 casas com o valor a 1) ou a 0 (4 casas com o valor a 0).

```
square(L,Index,Size,TotalSize):-
  ceiling(Index/Size) < Size,
  (Index - (floor(Index/Size)*Size)) < Size,
  Index \= TotalSize,
  element(Index,L,A),
  Index1 is Index + 1,
  element(Index1,L,B),
  Index2 is Index+Size,
  element(Index2,L,C),
  Index3 is Index2+1,
  element(Index3,L,D),
  sum([A,B,C,D],#\=,0),
  sum([A,B,C,D],#\=,4),
  square(L,Index1,Size,TotalSize).
square(_,_,_,_).
```

Apesar de a restrição da conectividade não ter sido implementada corretamente devido à sua dificuldade, mesmo assim foram impostas algumas restrições. Restrições impostas no predicado *connected*, que obriga a que uma peça tenha pelo menos à sua direita, esquerda, em cima ou em baixo uma peça do mesmo tipo.

```
connected(L,Index,Size,TotalSize):-
    Index \= TotalSize,
    Index1 is Index + 1,
    Index2 is Index + Size,
    Index3 is Index - 1,
    Index4 is Index - Size,
    ceiling(Index/Size) < Size,
    (Index - (floor(Index/Size)*Size)) < Size,
    ceiling(Index/Size) > 1,
    (Index - (floor(Index/Size)*Size)) > 1,
    element(Index,L,A),
    element(Index1,L,B),
    element(Index2,L,C),
    element(Index3,L,D),
    element(Index4,L,E),
    (A#=B #\/ A#=C #\/ A#=D #\/ A#=E),
    connected(L,Index1,Size,TotalSize).
```

3.3 Estratégia de Pesquisa

De modo a tornar a pesquisa mais eficiente, foi usado no predicado *labeling*, a opção *ffc* - *first fail constraint*. Isto faz com que seja usada a restrição mais rápida: é escolhida a variável com o domínio mais pequeno, com menos restrições e mais à esquerda.

Comparação entre o tempo de pesquisa de 2 tabuleiros 20x20, *sem ffc* e *com fcc*, respetivamente.

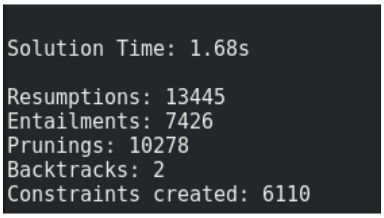


Figura 3: sem *labeling[ffc]*

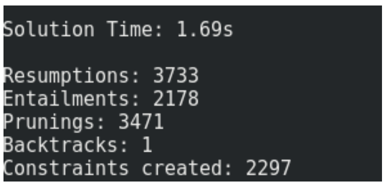


Figura 4: com *labeling[ffc]*

The most constrained heuristic is used: a variable with the smallest domain is selected, breaking ties by (a) selecting the variable that has the most constraints suspended on it and (b) selecting the leftmost one.

3.4 Gerador Aleatório do Puzzle a Resolver

Após serem chamadas os predicados que colocam as restrições, o tabuleiro é desenhado através de vários predicados.

O programa resolve qualquer puzzle independentemente do tamanho dado.

De seguida, utilizou-se o predicado *display_board*, que é responsável por desenhar tabuleiro, com o tamanho dado. E coloca inicialmente peças aleatoriamente com o predicado *place_random_pieces*.

O puzzle é inicialmente mostrado ao utilizador apenas com algumas peças colocadas, pronto para ser resolvido.

4 Visualização da Solução

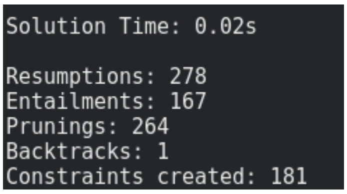


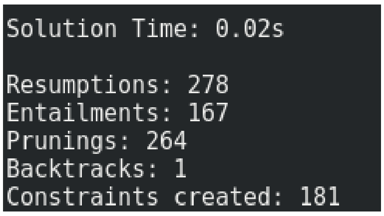
Figura 5: Exemplo de Puzzle Yin Yang

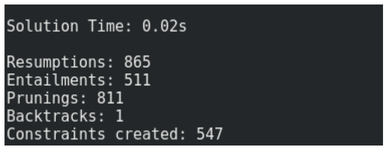
```
yinyang(Size):-
  reset_timer,
  TotalSize is Size*Size,
  length(L,TotalSize),
  domain(L,0,1),
  place_random_pieces(L,TotalSize,Size,Size),
  square(L,1,Size,TotalSize),
  connected(L,1,Size,TotalSize),
  labeling([],L),
  display_walls(Size),
  display_board(L,Size,Size,Size),
  print_time,
  fd_statistics.
```

Para além do tabuleiro final mostra algumas estatísticas como o tempo que demora a resolver o puzzle, se foi feito algum *backtracking*, numero de restrições feitas, entre outras.

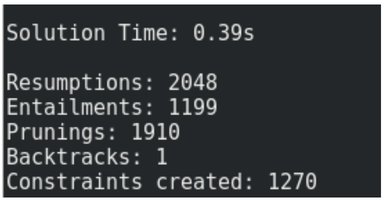
Alguns cuidados para a otimização da resolução do puzzle foram tidas em conta, desta forma, foi criado alguns gráficos que permitissem visualizar a diferença entre tempos de resolução para diferentes tamanhos de tabuleiro.

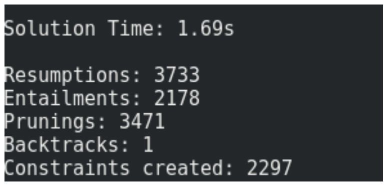
Estatísticas para um Tabuleiro 6x6 e 10x10:


```
Solution Time: 0.02s

Resumptions: 278
Entailments: 167
Prunings: 264
Backtracks: 1
Constraints created: 181
```

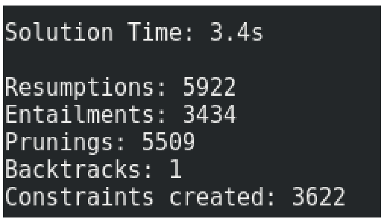

```
Solution Time: 0.02s

Resumptions: 865
Entailments: 511
Prunings: 811
Backtracks: 1
Constraints created: 547
```

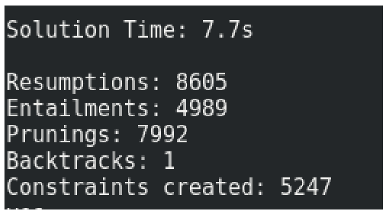
Estatísticas para um Tabuleiro 15x15 e 20x20:


```
Solution Time: 0.39s

Resumptions: 2048
Entailments: 1199
Prunings: 1910
Backtracks: 1
Constraints created: 1270
```


```
Solution Time: 1.69s

Resumptions: 3733
Entailments: 2178
Prunings: 3471
Backtracks: 1
Constraints created: 2297
```

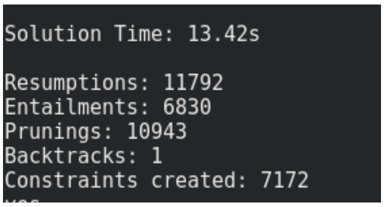
Estatísticas para um Tabuleiro 25x25 e 30x30:


```
Solution Time: 3.4s

Resumptions: 5922
Entailments: 3434
Prunings: 5509
Backtracks: 1
Constraints created: 3622
```

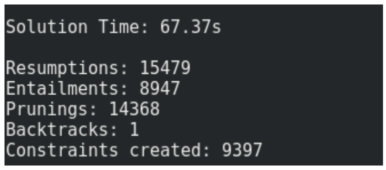

```
Solution Time: 7.7s

Resumptions: 8605
Entailments: 4989
Prunings: 7992
Backtracks: 1
Constraints created: 5247
```

Estatísticas para um Tabuleiro 35x35 e 40x40:

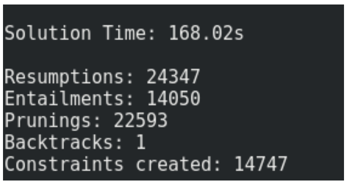
Estatísticas para um Tabuleiro 50x50:



```
Solution Time: 13.42s

Resumptions: 11792
Entailments: 6830
Prunings: 10943
Backtracks: 1
Constraints created: 7172
```



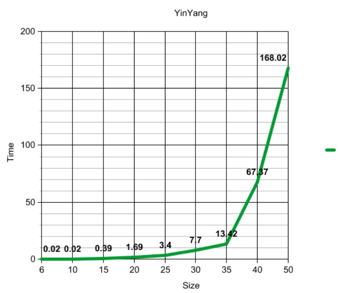
```
Solution Time: 67.37s

Resumptions: 15479
Entailments: 8947
Prunings: 14368
Backtracks: 1
Constraints created: 9397
```

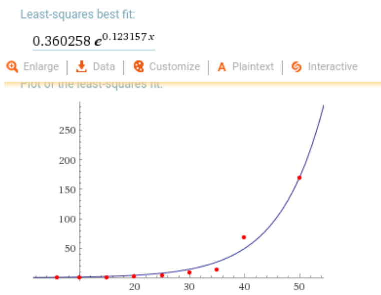


```
Solution Time: 168.02s

Resumptions: 24347
Entailments: 14050
Prunings: 22593
Backtracks: 1
Constraints created: 14747
```

Figura 6: Exemplo de Estatísticas

Gráficos:





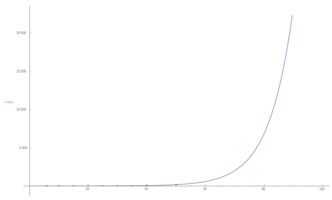


Figura 7: Exemplo de Estatísticas

Após correr varias vezes a nossa solução para tamanhos de tabuleiro diferentes, obtivemos os resultados apresentados na figura 15. Colocando estes valores no Wolfram Alpha, obtivemos uma função aproximada do crescimento do tempo que demora a resolver o problema em relação ao tamanho do tabuleiro. Podemos averiguar que este crescimento se assemelha a uma função exponencial.

5 Conclusões e Considerações Finais

Após a realização deste projeto em Prolog, concluí-se que a linguagem de Prolog é uma linguagem muito poderosa e eficiente para a resolução de problemas de lógica, como outras questões de decisão e otimização.

O uso de predicados fortes pela biblioteca 'clpfd', a compreensão do funcionamento do *labeling* e elaboração de variavéis de decisão e restriçoes, tornou-se maid fácil a resolução do professor. No entanto, a regra da conectividade não foi completamente resolvida apesar do esforço e empenho do grupo, assim como o auxilio do professor. Desta forma, o nosso puzzle coloca peças que estejam no mínimo par a par, sem estarem todas ligadas entre si as peças da mesma cor.

Apesar de todas as dificuldades encontradas, de forma geral, este projeto foi importante para a solução implementada pelo nosso grupo correspondeu às expetativas.