Resolução de puzzles Domino usando Programação em Lógica com Restrições

Fábio Caramelo, Pedro Dias
FEUP-PLOG, Turma 3MIEIC01, Dominos1

Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias, s/n 4200-46,5 Porto, Portugal
fabiocaramelo18@gmail.com,pedrogoodwork@gmail.com

Abstract. No âmbito da unidade curricular de Programação em Lógica, do 3º ano do Mestrado Integrado em Engenharia Informática e Computação desenvolvemos uma aplicação capaz de resolver puzzles "Domino" utilizando *Programação em Lógica com Restrições*. O artigo segue a estrutura recomendada, contendo explicações sobre a descrição do problema, a abordagem seguida, a visualização da solução, demonstração de resultados obtidos e conclusões sobre o trabalho realizado.

Keywords: domino, prolog, feup, programação em lógica, restrições

1 Introdução

O objectivo deste trabalho era implementar a resolução de um problema de decisão ou optimização em Prolog com restriçoes.

O grupo optou por implementar o puzzle Domino, um problema de decisão. O puzzle consiste num tabuleiro constituido por quadrados numerados onde se tem de colocar peças de dominó para completar o tabuleiro.

Este artigo descreve o puzzle Domino, a abordagem do grupo para implementar capaz de resolver os tabuleiros deste puzzle, bem como a análise da mesma. Ainda é descrito como é feita a visualização dos tabuleiros e da resolução do mesmo , a análise das estatisticas da resolução de puzzles com diferentes complexidades e a conclusão do projecto realizado.

2 Descrição do problema

Os puzzles "Domino" são puzzles rectangulares divididos em vários quadrados numerados, podendo conter quadrados não numerados. Cada peça consiste num par de números. O objectivo do jogo consiste em cobrir todo o tabuleiro numerado com peças, podendo apenas utilizar uma peça uma vez.

Na figura 1 e 2 é apresentado um tabuleiro e a sua respectiva solução.

1	3	0	1	2
3	2	0	1	3
3	3	0	0	1
2	2	1	2	0

Fig. 1. Tabuleiro 5x4

1	3	0	1	2
3	2	0	1	3
3	3	0	0	1
2	2	1	2	0

Fig. 2. Solução do tabuleiro da figura 1

3 Abordagem

Na abordagem à resolução deste puzzle decidiu-se implementar uma solução baseada nas fronteiras entre os quadrados numerados, em que cada quadrado possuí no máximo quatro fronteiras. Contudo, alguns quadrados não possuem as 4 fronteiras por estarem em bordas do tabuleiro. Também se assumiu que não existe peças repetidas e simétricas (Ex: peça [2,0] e [0,2]).

Os tabuleiros são representados como uma lista de de listas como se pode verificar na figura 3.

```
table1([[1,3,0,1,2],
        [3,2,0,1,3],
        [3,3,0,0,1],
        [2,2,1,2,0]]).
```

Fig. 3. Representação do tabuleiro da figura 1

Na resolução do tabuleiro são criadas duas listas com tamanho igual ao número de quadrados do tabuleiro, que serão a solução.

3.1 Variáveis de decisão

O tamanho da solução é determinado pelo tamanho tamanho do tabuleiro. Se um tabuleiro tiver largura W e altura H, o tamanho de cada lista East e South será W * H. A solução será uma lista que contém ambas as listas. Desde modo o tamanho da lista resultante é dado pela seguinte expressão, onde N é o tamanho da lista East e South:

```
Solution = N * 2
```

Conforme previamente referido, as variaveis de decisão são as fronteiras representadas por duas listas, East and South, com dominio compreendido entre 0 e 1. Cada posição do tabuleiro corresponde a uma posição de cada uma das listas South e East, sendo que a lista East representa a margem direita das posições do tabuleiro e a lista South a margem inferior de cada posição do tabuleiro. Deste modo, uma posição que contenha o valor 0 na lista East e o valor 1 na lista South terá que conter uma peça colocada na vertical, enquanto que uma posição que contenha o valor 1 na lista East e o valor 0 na lista South terá que conter uma peça colocada na horizontal. Cada peça de dominó contém uma linha que separa os dois valores dessa peça. As fronteiras quando ocupadas representam a linha divisória do domino.

3.2 Restrições

1. Todos os quadrados numerados tem só uma fronteira preenchida
Os quadrados numerados são afectados pela seguinte restrição:

```
NorthBorder + SouthBorder + EastBorder + WestBorder #= 1
```

Esta restrição tem o objectivo de restringir a cada quadrado numerado uma peça. Assim, esta equação apenas se verifica quando apenas 1 das fronteiras se encontra ocupada, ou seja, apenas 1 linha divisória associada.

Existem casos especificos desta regra em que estes quadrados nao tem todas a fronteiras, mais concretamente, as bordas do tabuleiro.

No canto superior esquerdo:

```
EastBorder + Southborder #=1
```

No canto inferior esquerdo:

```
EastBorder + Northborder #=1
```

No canto superior direito:

```
WestBorder + SouthBorder #= 1,
```

Nas restantes casas da primeira linha do tabuleiro:

```
EastBorder + WestBorder + Southborder #= 1
```

Nas restantes casas da primeira coluna do tabuleiro:

```
EastBorder + NorthBorder + SouthBorder #=1
```

Nas restantes casas da última coluna do tabuleiro:

```
WestBorder  + NorthBorder + SouthBorder #=1
```

Nas restantes casas da última linha do tabuleiro:

```
EastBorder + WestBorder + NorthBorder  #=1
```

SouthBorder representa o elemento da posição actual do tabuleiro na lista South, NorthBorder a o elemento da posição imediatamente acima da posição actual na lista South, EastBorder o elemento da posição actual do tabuleiro na lista East enquanto que WestBorder representa o elemento da posição imediatamente anterior na lista East.
O predicado que implementa esta restriçao percorre todo o tabuleiro aplicando as restrições anteriormente referidas e denomina-se:

```
restrictNeighbors(MaxLine,MaxCol,Line,Col,East,South, Board)
```

2. Uma peça só pode ocupar uma posição

A seguinte restrição relaciona o número lugares possiveis que uma determinada peça pode ocupar.

Assumindo que uma peça tem N opçoes possiveis. A partir destas N opçoes é possivel agrupar as fronteiras que vão originar as linhas divisorias, o somatório destas tem de ser igual a um, caso contrário a peça estaria colocada mais que uma vez em posiçoes diferentes.

Podendo concluir:

```
Solution :  o array constituido pelas fronteiras
sum(Solution, \#=, 1)
```

Os predicados envolvidos para aplicar esta restrição são os seguintes:

```
placement([P|Ps],East,South,MaxLine,MaxCol)
restrictPlacement(Ps,East,South,MaxLine,MaxCol)
compileOptions([L-C-0|Ps],East,South,[O|Os],MaxLine,MaxCol)
compileOptions([L-C-1|Ps],East,South,[O|Os],MaxLine,MaxCol)
listAllPlacements(Pieces, Board, Placements)
```

3.3 Função de Avaliação

De modo a avaliar o tempo de execução de cada um dos tabuleiros foram criados dois predicados:

```
reset_timer
    print_time
```

O primeiro tem como objectivo reiniciar o contador enquanto que o segundo tem como objectivo obter e imprimir o tempo de execução.

Foram ainda criados 3 predicados que executam estes predicados, um para cada teste:

```
stat1
stat2
stat3
```

3.4 Estratégia de Pesquisa

Na resolução deste problema não foi utilizada nenhuma estratégia de etiquetagem. Embora o grupo tenha testado várias opções possiveis, não conseguiu obter melhorias visiveis.

4 Visualização da Solução

Antes de ser mostrada a solução do puzzle é imprimido o tabuleiro inicial.
O predicado de visualização da solução imprime o tabuleiro-solução com as linhas divisórias das peças um dominó devidamente colocadas, sendo os contornos do tabuleiro e das peças ignorados para facilitar a visualização deste.

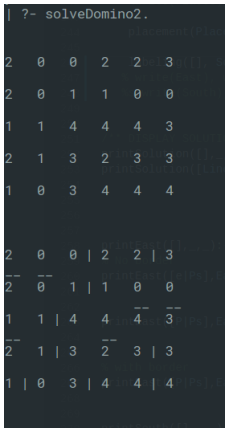

Fig. 4. Tabuleiro 6x5 e respectiva solução

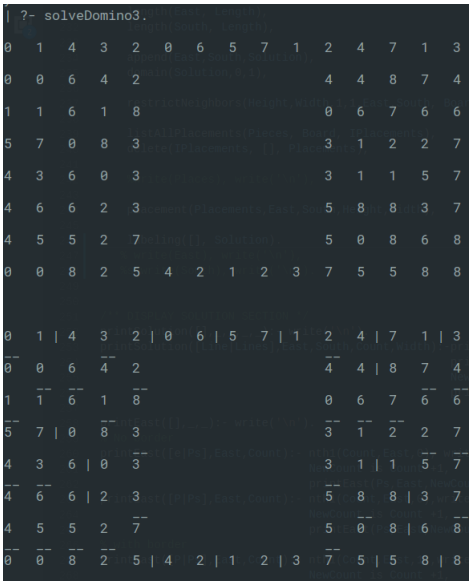

Fig. 5. Tabuleiro 15x8 e respectiva solução

5 Resultados

No seguinte gráfico serão mostrados os tempos de execução médios para os 3 casos testados no programa, ou seja, para um tabuleiro 5x4, 6x5 e ainda para um tabuleiro 15x8. Cada teste foi executado 10 vezes, sendo os tempos apresentados uma média de todos os tempos obtidos.

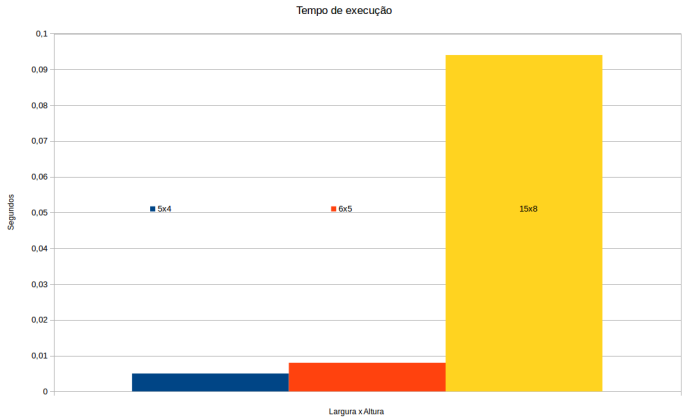

Fig. 6. Tempos de Execução

Como se pode verificar, os tempos de execução são bastante rápidos, sendo que nos dois testes mais pequenos foi obtida a solução do tabuleiro em menos de 1 centésimo de segundo e para o tabuleiro maior o resultado foi alcançado em menos de 1 décimo de segundo.

6 Conclusões e Trabalho Futuro

O desenvolvimento do projecto revelou-se importante para compreender as capacidade da programação com restrições, bem como o desenvolvimento do pensamento de forma lógica.

Além disso foi possivel aprender os pontos fortes do Prolog, mais concretamente do predicado labeling, pois problemas complexos podem ser resolvidos em poucas linhas de código.

Analisando os resultados obtidos é possivel afirmar que o programa tem tempo de execução rápidos embora a amostra de tabuleiros fosse pequena. Mas é importante verificar que o aumento das dimensões tabuleiro aumenta significativamente o tempo de execução.

As melhorias futuras para este projecto passariam por implementar um gerador de tabuleiro e além disso, melhorar o desempenho da aplicação.

7 The References Section

References

1. Szeredi, Peter, Teaching Constraints through Logic Puzzles `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.97.9715&rehttps://preview.overleaf.com/public/vsgdddhzbtnh/images/00064ed031fed6d6b99ed7e6618926977e43557a.jpegp=rep1&type=pdf`
2. Kenneth E. Caviness, Three Ways to Solve Domino Grids `http://www.mathematica-journal.com/2014/10/three-ways-to-solve-domino-grids/`
3. Camacho Rui, Cardoso Henrique L., Silva Daniel, Slides da Unidade Curricular disponibilizados no Moodle