Puzzle 2D Doppelblock

Resolução de Problemas de Decisão com Restrições

André Correia[1] e Pedro Faria[2]

[1] FEUP-PLOG, Turma 3MIEIC06, Grupo Doppelblock_2
up200706629@fe.up.pt
[2] FEUP-PLOG, Turma 3MIEIC06, Grupo Doppelblock_2
up201406992@fe.up.pt

Resumo. Este artigo completa a realização do segundo projeto da Unidade Curricular de Programação em Lógica do Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto. O projeto tem como objetivo principal a conceção da abordagem necessária para a resolução de problemas de satisfação de restrições, por forma a aplicá-la em casos concretos, como os problemas de otimização ou decisão combinatória. A abordagem foi aplicada ao puzzle 2D Doppelblock, construindo um programa que implementa um solver para a sua resolução. Através da manipulação de predicados disponibilizados pelo SICStus Prolog, mostramos neste artigo a resolução deste problema e respetiva análise.

Keywords: doppelblock, prolog, feup, sicstus, plog, solução, decisão.

1 Introdução

Este projeto, desenvolvido no âmbito da unidade curricular de Programação em Lógica, do Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto, tem como objetivo avaliar os alunos relativamente à sua capacidade para resolver problemas de otimização ou decisão, utilizando conceitos de programação em lógica com restrições.

De entre as várias opções disponibilizadas, o grupo optou por um problema de decisão: o puzzle 2D *Doppelblock*. Semelhante ao *Sudoku*, o *Doppelblock* consiste num enigma, em que inicialmente são afetos valores a cada uma das linhas e colunas do tabuleiro e a sua resolução consiste em, colocando exatamente duas casas pretas em cada linha e coluna, preencher as casas em branco por forma a que, a soma dos valores entre as casas pretas perfaça o valor inicialmente afeto à linha ou coluna em causa. Uma regra importante na resolução do puzzle, consiste na não repetição de valores em cada linha e em cada coluna.

Este artigo descreve, detalhadamente, o problema em análise, a abordagem seguida tendo em vista a resolução do mesmo, os resultados obtidos e respetiva análise e, ainda, as conclusões retiradas com a realização do projeto.

2 Descrição do Problema

O *Doppelblock* é um enigma semelhante ao *Sudoku*, consistindo num tabuleiro quadrado (por omissão, 6x6) em que todos os valores presentes, em cada linha e em cada coluna, não podem ser repetidos.

Inicialmente são afetos valores a cada uma das linhas e colunas, conforme ilustra a Fig. 1. O valor máximo que estes valores podem assumir, depende da dimensão do tabuleiro e representa a soma de todos os números até a dimensão do tabuleiro, excluindo as duas casas pretas (N-2, sendo N a dimensão do tabuleiro quadrado). Considerando o exemplo por omissão, em que o tabuleiro assume a dimensão 6x6 e em que cada linha e coluna contem duas casas pretas, o valor máximo que os valores afetos a cada uma das linhas e colunas podem assumir é 10, representando a soma de todos os números de 1 até 4 (6-2).

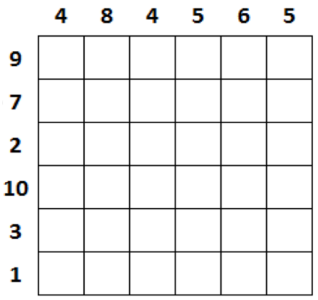

Fig. 1. Exemplo de configuração inicial do tabuleiro de tamanho 6

O objetivo consiste em preencher o tabuleiro com valores de 1 até N-2, de modo a que, em cada linha e em cada coluna, existam exatamente duas casas pretas e que a soma dos valores, que se encontram entre as casas pretas, seja igual ao valor inicialmente afeto à linha ou coluna em causa.

A Fig. 2, ilustra a solução para o tabuleiro anterior.

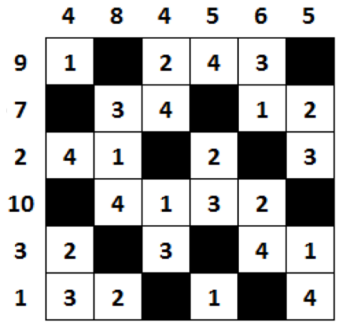

Fig. 2. Solução para o tabuleiro ilustrado na Fig. 1

3 Abordagem

O primeiro passo dado na conceção da abordagem foi tentar perceber como modelar o puzzle como um problema de restrições. Posteriormente, escolher as variáveis de decisão a usar no predicado *labeling*, entender as restrições necessárias para a resolução do problema e aplicá-las, restringindo as variáveis.

Foi ainda tida em consideração 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 casas pretas é feita com o valores -1 e 0. Por sua vez, a sua visualização é feita através de um X.

3.1 Variáveis de Decisão

Neste puzzle as variáveis de decisão correspondem às células do tabuleiro. O objetivo passa por decidir que número atribuir à variável, correspondendo o domínio da variável à gama de valores que uma célula pode ter. O domínio varia de 1 a N-2, sendo N a dimensão do tabuleiro NxN.

3.2 Restrições

As restrições utilizados na resolução deste puzzle são as seguintes:

- Todos os elementos de uma linha têm de ser diferentes (aplicar *all_distinct* a todas as linhas);
- Todos os elementos de uma coluna têm de ser diferentes (aplicar *all_distinct* a todas as colunas);
- A soma de todos os elementos entre duas casas pretas tem de ser igual ao valor inicialmente afeto às linhas ou colunas. Para tal, foi utilizado um predicado (*verifylist*/2), que percorre uma linha ou coluna, e devolve, no segundo argumento, a lista dos valores entre as casas pretas. Em seguida, é aplicado o predicado *sum*/3 para verificar que a soma dos elementos da lista corresponde, efetivamente, ao valor inicialmente atribuído à linha ou coluna em causa;
- Em cada linha ou coluna, as casas pretas não podem ser colocadas em células consecutivas.

3.3 Função de Avaliação

Para a resolução do presente problema não é necessária uma função para avaliar a solução obtida. Tratando-se de um puzzle, esta pode ser verificada visualmente.

3.4 Estratégia de Pesquisa

Por forma a tornar a pesquisa mais eficiente, foi utilizada a opção *ffc – first fail constraint*, no predicado *labeling* para ordenação das variáveis. Esta estratégia consiste em ordenar as variáveis, das que contêm mais restrições para as que têm menos restrições..

Para a ordenação de valores foi utilizada a estratégia por omissão do predicado *labeling*, ou seja, o domínio é explorado por ordem ascendente.

3.5 Gerador Aleatório do Puzzle a Resolver

Para além da resolução do tabuleiro de dimensão 6x6, foi implementada a resolução para tabuleiros de qualquer tamanho superior a 3. Tabuleiros com dimensão inferior a 3 não apresentam solução uma vez que, como em cada linha e coluna é obrigatória a existência de 2 casas pretas, o domínio das variáveis associadas às células brancas seria apenas o valor 1 (3-2).

Assim, para a resolução de tabuleiros de dimensão variável, o programa inicia perguntando ao utilizador qual a dimensão pretendida para o tabuleiro.

Em seguida, após serem chamados os predicados que colocam as restrições, o tabuleiro é desenhado através de vários predicados. Para a dimensão introduzida pelo utilizador, são mostradas todas as soluções para essa dimensão, variando os valores que inicialmente são atribuídos às linhas e colunas (correspondentes à soma dos valores entre as casas pretas).

O programa resolve qualquer puzzle independentemente do tamanho dado (superior a 3).

4 Visualização da Solução

Tendo em vista a visualização da solução é utilizado o predicado *printValues(+Tabuleiro, +ValoresLinhas, +ValoresColunas)* ou o predicado *printValuesDynamic(+Tabuleiro, +ValoresLinhas, +ValoresColunas, +DimensaoTabuleiro)* dependendo se se está a resolver o tabuleiro 6x6 ou um tabuleiro com dimensão introduzida pelo utilizador. Estes predicados percorrem todas as linhas do tabuleiro, desenhando-as de forma amigável para o utilizador. Este predicado socorre-se de outros sendo que, no caso dinâmico, o mais importante é o *ciclo_imprime(+Tabuleiro*, *+ValoresLinhas*, *+DimensaoTabuleiro*, *+DimensaoFixaTabuleiro*, *+LinhaAtual*) que percorre o tabuleiro, tendo em conta a sua dimensão.

Para além do tabuleiro, são ainda mostradas algumas estatísticas como o tempo que demora a resolver o puzzle, se foi feito algum *backtracking*, número de restrições feitas, entre outras.

Alguns cuidados para a otimização da resolução do puzzle foram tidas em consideração, através da análise dessas estatísticas.

As Fig. 3, Fig. 4 e Fig. 5 ilustram alguns exemplos de visualização.

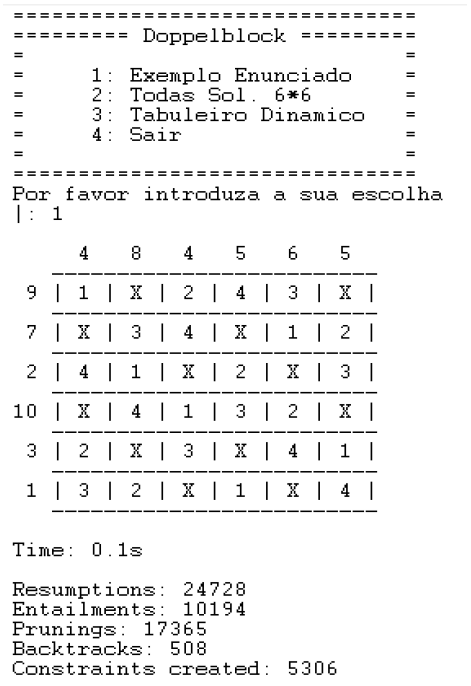

Fig. 3. Visualização da resolução do exemplo do enunciado

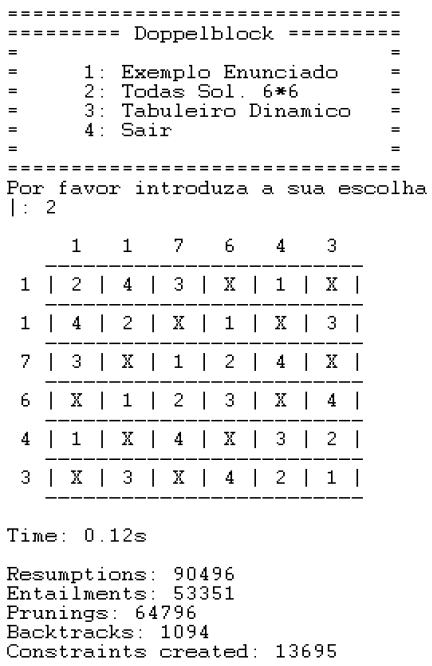

Fig. 4. Visualização da resolução de um tabuleiro 6x6

```
==============================
========= Doppelblock =========
=                            =
=     1: Exemplo Enunciado   =
=     2: Todas Sol. 6*6      =
=     3: Tabuleiro Dinamico  =
=     4: Sair                =
=                            =
==============================
Por favor introduza a sua escolha
|: 3
Por favor introduza o tamanho do tabuleiro: 7

     5   4   2   8   5   5   2
   ---------------------------
 5 | X | 5 | X | 4 | 2 | 3 | 1 |
   ---------------------------
 2 | 5 | X | 2 | X | 3 | 1 | 4 |
   ---------------------------
 4 | X | 4 | X | 5 | 1 | 2 | 3 |
   ---------------------------
 7 | 3 | X | 1 | 2 | 4 | X | 5 |
   ---------------------------
 5 | 2 | 3 | 4 | 1 | X | 5 | X |
   ---------------------------
 5 | 4 | 1 | 3 | X | 5 | X | 2 |
   ---------------------------
 4 | 1 | 2 | 5 | 3 | X | 4 | X |
   ---------------------------

Time: 0.17s

Resumptions: 1789024
Entailments: 1042145
Prunings: 1383245
Backtracks: 26788
Constraints created: 285611
```

Fig. 5. Visualização da resolução de um tabuleiro 7x7

5 Resultados

Após vários testes com diversas dimensões de tabuleiros, verificou-se que para um tamanho menor ou igual a sete, o programa resolve o problema em tempo útil aceitável. No entanto, para dimensões superiores a dez não é possível encontrar solução em tempo útil.

Decidiu-se, então, elaborar um estudo sobre as quais as opções de *labeling* mais eficazes para resolver o puzzle, variando a estratégia de ordenação de variáveis e a estratégica de *branching*.

Foram obtidos os resultados ilustrados na Fig. 6.

7x7			
	Tempo	**Backtracks**	**Constraints**
ffc step	0,14	26788	285611
ff step	0,17	26788	285611
ffc enum	0,15	26788	285611
ffc bisect	0,15	26788	285611
ff bisect	0,15	26788	285611

Fig. 6. Resultados de tempo, backtracks e restrições para diversas opções de *labeling*

Não existindo grande variação nos resultados, conclui-se que a melhor opção é *ffc step*.

6 Conclusões e Trabalho Futuro

Este projeto mostrou-se essencial para uma melhor compreensão do mecanismo subjacente à programação em lógica com restrições, tendo sido possível, através do desenvolvimento de um programa para a resolução do puzzle 2D *Doppelblock*, estudar e testar conceitos de uma forma prática.

Conclui-se, ainda, que a linguagem Prolog é uma linguagem muito poderosa e eficiente para a resolução de problemas de satisfação de restrições, como os problemas de otimização ou decisão combinatória. Esta permite resolver problemas complexos em poucas linhas de código.

Os resultados obtidos são bastante satisfatórios, visto que um problema de alguma complexidade consegue ser resolvido em relativamente pouco tempo. No entanto, a solução apresentada tem uma limitação relativa à dimensão máxima dos tabuleiros. Esta não pode ser superior a 9x9, visto não ser apresentada uma solução em tempo útil.

Para além da limitação identificada anteriormente, o trabalho não necessita de melhoramentos significativos.

7 Referências

1. Puzzle 2D *Doppelblock, http://logicmastersindia.com/lmitests/dl.asp?attachmentid=659&view=1*