Resolução do problema Battleships (Moshe Rubin) usando restrições em Prolog

João Martins[up201605373@fe.up.pt] and João Brandão[up201705573@fe.up.pt]

FEUP-PLOG, Turma 3MIEIC07, Battleships_4

Abstract. Este projeto foi desenvolvido no âmbito da unidade curricular de Programação em Lógica, do 3º ano do Mestrado Integrado em Engenharia Informática e de Computação da Faculdade de Engenharia da Universidade do Porto, e tem como objetivo a resolução de um problema de decisão/otimização implementando restrições. Esse problema é o Battleships e vai ser descrito nas secções abaixo.

1 Introdução

O Battleships é dos puzzles lógicos mais reconhecidos aparecendo no primeiro World Puzzle Championship em 1992 em Nova Iorque [1]. De todas as implementações deste puzzle destaca-se a de Moshe Rubin, Fathom It!, um programa para Windows que permite gerar e resolver puzzles [2].

2 Descrição do Problema

O Battleships consiste em localizar a posição de um certo número de navios num tabuleiro. Tanto o tamanho do tabuleiro como o número de navios e o seu tamanho são fornecidos. Cada parte do navio ocupa uma célula do tabuleiro. Os navios tem orientação horizontal ou vertical no tabuleiro, não podem ocupar células com água e não podem ser adjacentes, mesmo na diagonal. Os números à direita e em baixo do tabuleiro representam o número de células ocupadas por navios na linha ou coluna correspondente.

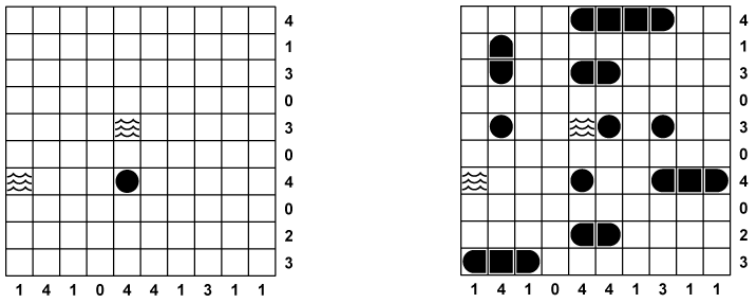

Fig. 1. Exemplo de um tabuleiro de jogo e sua resolução.

3 Abordagem

3.1 Variáveis de Decisão

Para resolver este problema em Prolog foi criada uma lista de tamanho igual ao número total de células no tabuleiro, cada valor representando uma célula do tabuleiro. A estas variáveis foi atribuído um domínio entre 0 a X, ambos inclusive, sendo X igual ao tamanho do maior navio usado no problema. Cada um destes valores tem o seu significado sendo que 0 é usado para representar uma célula vazia, ou seja, preenchida por água, e os números de 1 a X para indicar a presença de uma peça de um navio. Cada navio é representado num número de células igual ao seu tamanho com valores que vão desde o 1 ao seu tamanho, por exemplo, um navio de tamanho 1 é representado por uma célula com valor 1, um navio de tamanho 2 é representado por duas células, uma com valor 1 e outra com valor 2, e assim sucessivamente.

3.2 Restrições

As restrições usadas para resolver este problema são 4 e são todas rígidas:

Número de peças de navios por linha/coluna igual ao número à direita da linha/em baixo da coluna Esta restrição foi implementada em Prolog no predicado **constrain/4** que percorre cada linha/coluna do tabuleiro e restringe o número de peças de navios usando **global_cardinality/2**.

As células diagonais a células com peças de navios tem que ser água Esta restrição foi implementada no predicado **constrainDiagonals/5** que percorre o tabuleiro e restringe todas as células de modo a que o valor de uma célula multiplicado pelo valor de uma das suas diagonais (isto é feito para todas as diagonais) seja igual a 0, ou seja, pelo menos uma das células tem que ser 0.

Todas as peças dos navios fornecidos tem que estar no tabuleiro Restringir o tabuleiro de modo a que tenha o número certo de cada peça de navio. Para isso usou-se novamente o predicado **global_cardinality/2**.

As peças dos navios tem que estar bem colocadas no tabuleiro Cada navio tem que estar no tabuleiro com as peças que o representam. Por exemplo, um navio de tamanho 5 tem que estar no tabuleiro na forma 12345, representando cada número uma célula, horizontal ou verticalmente. Esta restrição está implementada no predicado **constrainShipsPieces/5**.

3.3 Função de Avaliação

Visto que para um dado puzzle que apresenta várias soluções não existe uma solução ótima uma vez que todas as diferentes soluções são válidas da mesma forma, uma função de avaliação não se aplica a este problema.

3.4 Estratégia de pesquisa

De forma a escolher a melhor estratégia de pesquisa foram realizados vários testes, chegando-se à conclusão que a estratégia de pesquisa mais eficiente é a escolha da opção ff como heurística de ordenação de variáveis e as opções step ou enum como heurísticas de seleção de valores. A justificação detalhada desta escolha pode ser verificada na secção 5.

4 Visualização da Solução

Foram criados dois predicados principais: **battleships/2** e **randomPuzzle/3**.

O primeiro predicado permite resolver puzzles de quaisquer dimensões e com qualquer números de navios. Este predicado recebe como argumento um ID de um tabuleiro a resolver que se encontra no ficheiro **data.pl**. O segundo predicado criado permite a geração automática de puzzles. Recebe como argumentos o número de linhas e colunas (os tamanhos do tabuleiro possíveis são 6x6, 7x7, 8x8, 10x10 ou 15x15) e com esta informação lê os navios a ser colocados no respetivo tabuleiro através do predicado **shipsData/3** no ficheiro **data.pl** e gera um tabuleiro aleatório.

Para representar o tabuleiro com os navios foi implementado o predicado **printBoard/3** que desenha o tabuleiro de forma clara e intuitiva.

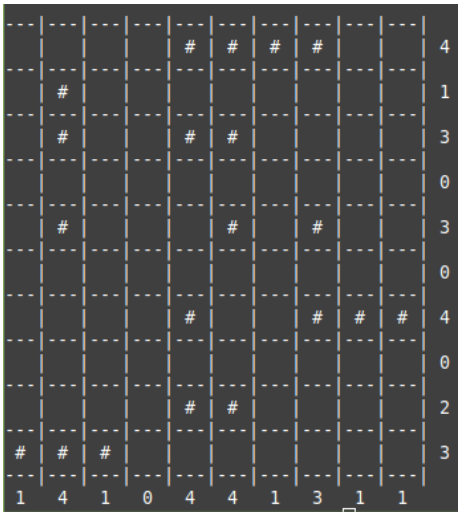

Fig. 2. Visualização da solução do problema da **Fig. 1**

5 Resultados

Para determinar as melhores heurísticas de escolha de variável e de valor, começou-se por testar o puzzle da **Fig. 1** com diferentes opções de ordenação de variáveis. Os resultados obtidos foram os seguintes:

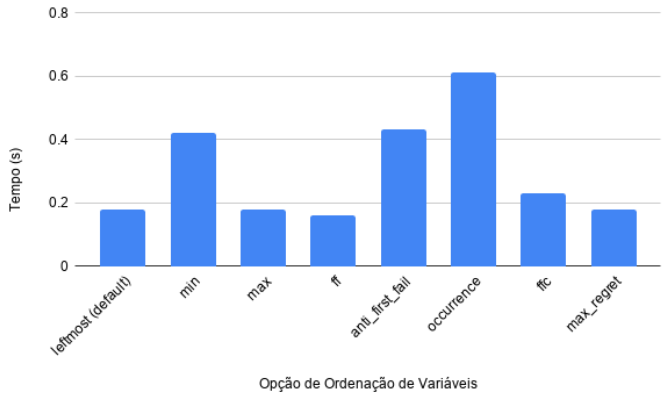

Fig. 3. Tempo de pesquisa com diferentes opções de ordenação de variáveis.

Verificamos que a opção ff é a melhor. Fizemos mais testes com outros tabuleiros mas este foi o que mostrou a maior discrepância entre valores.

Posteriormente fizemos para a melhor opção de ordenação de variáveis, ff, testes com diferentes opções de seleção de valores para o mesmo tabuleiro. Os resultados obtidos foram os seguintes:

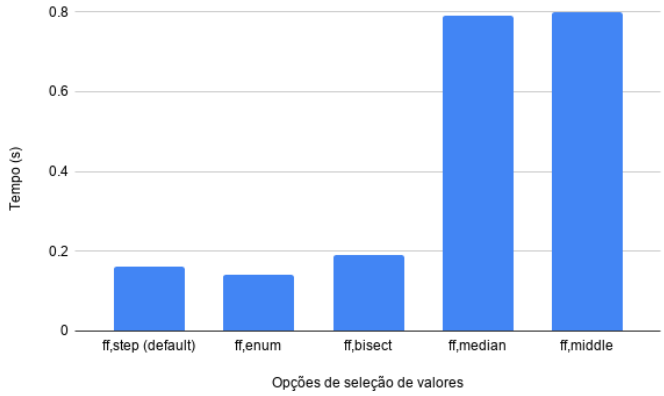

Fig. 4. Tempo de pesquisa com diferentes opções de seleção de valores.

Aqui verificamos que as melhores opções de seleção de valores são enum e step. Fizemos mais testes com outros tabuleiros e verificou-se o mesmo.

Chegamos assim à conclusão que as melhores estratégias de pesquisa são ff,step e ff,enum.

Também testamos a eficiência do programa para diferentes tamanhos de tabuleiro. Assim, usando ff,enum os resultados obtidos foram os seguintes:

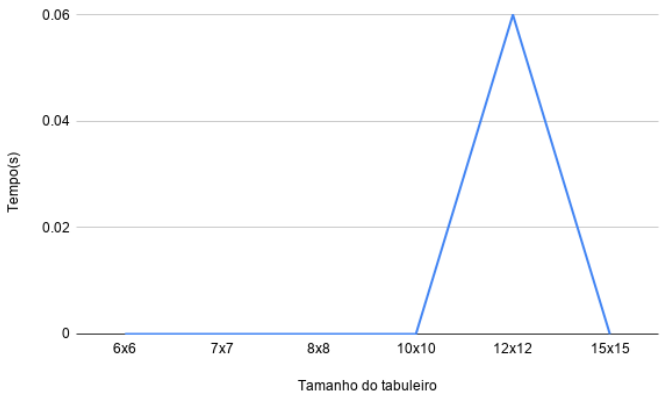

Fig. 5. Tempo de pesquisa com opções ff,enum com diferentes tamanhos de tabuleiro.

6 Conclusão

O projeto teve como principal objetivo aplicar o conhecimento adquirido nas aulas teóricas e práticas, e foi concluído que a linguagem Prolog, em particular, o módulo de restrições, é bastante útil na resolução de problemas de decisão e otimização. Ao longo do desenvolvimento deste projeto, foram encontradas algumas dificuldades, nomeadamente a escolha das restrições e a sua implementação. Após uma longa análise da biblioteca clpfd e dos slides fornecidos foi possível superar estas mesmas dificuldades.

Um aspeto que podia ser melhorado é a geração automática de puzzles que não foi muito trabalhado por falta de tempo e também por não ser o ponto principal deste projeto.

Em suma, o projeto foi concluído com sucesso, visto solucionar corretamente o problema proposto, e o seu desenvolvimento contribuiu positivamente para uma melhor compreensão do funcionamento do labeling e variáveis de decisão, assim como na aplicação de restrições.

References

1. Puzzle Battleships, https://en.wikipedia.org/wiki/Battleship_(puzzle). Último acesso 2 Jan 2020
2. Fathom It!, http://www.mountainvistasoft.com/inform.htm. Último acesso 2 Jan 2020

3. Puzzles usados nos testes, https://lukerissacher.com/battleships. Último acesso 2 Jan 2020
4. Apontamentos disponibilizados na plataforma Moodle na unidade curricular Programação em Lógica, https://moodle.up.pt/course/view.php?id=2133. Último acesso 2 Jan 2020
5. Documentação SWI Prolog, https://www.swi-prolog.org/. Último acesso 2 Jan 2020