Hitori

Resolução de um problema de decisão usando Programação em Lógica com Restrições

João Araújo and José Pedro Teles

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

Resumo Este artigo complementa o segundo projeto da Unidade Curricular de Programação em Lógica, do Mestrado Integrado em Engenharia Informática e Computação. O objetivo deste trabalho é desenvolver um programa usando Programação em Lógica com Restrições para a resolução de um problema de decisão combinatoria, o *Hitori*. Conclui-se que, usando restrições, este jogo é relativamente fácil de reproduzir.

Keywords: hitori, sicstus, prolog, PLR, feup

1 Introdução

O objetivo deste trabalho é a construção de um programa em Programação em Lógica com Restrições para a resolução de um dos problemas de otimização ou decisão combinatória sugeridos. O nosso grupo optou por um problema de decisão, o *Hitori*.

O sistema de desenvolvimento é o SICStus Prolog, que inclui um módulo de resolução de restrições sobre domínios finitos: clp(FD).

Este artigo descreve detalhadamente a implementação do *Hitori* em Prolog. São abordadas as variáveis de decisão e restrições utilizadas na resolução deste jogo. A estratégia de pesquisa e a visualização da solução são também tópicos importantes. Consideramos essencial acrescentar também estatísticas de resolução com diferentes complexidades. Finalmente, apresentamos os resultados obtidos, a conclusão e perspetivas de desenvolvimento.

2 Descrição do Problema

O *Hitori* é um puzzle bastante fácil de perceber que foi inventado pela editora japonesa *Nikoli*. Normalmente, são utilizados tabuleiros de 8x8, 12x12 ou 17x17 e o objetivo do jogo é sombrear quadrados de modo a que não apareça nenhum número repetido na mesma linha ou coluna. Para além disso, um quadrado sombreado não pode tocar noutro sombreado, tanto na vertical como na horizontal, e, quando completo, todos os quadrados não sombreados, a branco, têm que estar interligados entre si.

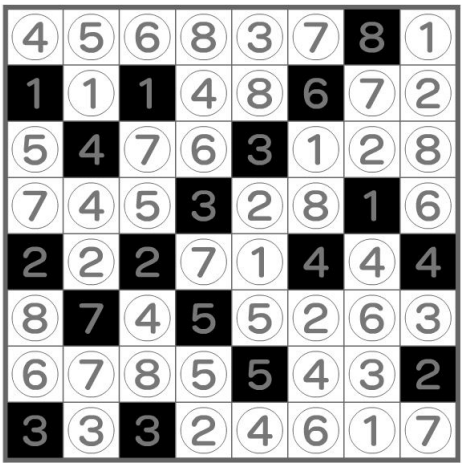

Figura 1. Exemplo de um estado final do jogo.

3 Abordagem

A implementação do tabuleiro em Prolog foi baseada numa matriz. Assim, representamos o tabuleiro como uma lista de listas, onde cada elemento é número correspondente àquela célula.

3.1 Variáveis de Decisão

Com a representação utilizada, a solução apresenta-se sobre a forma de uma lista de listas, todas elas com tamanho N. Cada linha é representada por uma lista onde estão guardados o números. O domínio da solução é definido pela dimensão do tabuleiro. As linhas e as colunas variam de 1 a N, quanto maior o tabuleiro mais complexa a solução se torna.

3.2 Restrições

A resolução do problema pode ser resumida às seguintes restrições:

Cada linha deve conter os números de 1 a N

Para um tabuleiro de tamanho N, percorrem-se recursivamente as linhas de 1 a N, e verifica-se se todos os números dessa linha são diferentes. Se não forem, aplicam-se os algoritmos para encontrar quais os números a sombrear e para circundar.

Cada número sombreado não pode tocar noutro sombreado na vertical nem na horizontal

Para um tabuleiro de tamanho N, percorrem-se recursivamente as linhas de 1 a N, e verifica-se em cada número sombreado se apenas está em contacto com outros sombreados nas diagonais ou com nenhum, ou seja, não está em contacto com outros sombreados na horizontal nem na vertical.

Quando resolvido, os números circundados devem formar uma única região

4 Visualização da Solução

Os predicados responsáveis pela visualização dos possíveis tabuleiros do puzzle em modo texto, apesar de serem uma grande parte do código, foram relativamente fáceis de implementar. O facto de termos trabalhado com uma representação de um tabuleiro mais complexo, acabou por facilitar o desenvolvimento neste segundo projeto.

O tabuleiro está a ser impresso linha a linha . O predicado responsável por desenhar o tabuleiro é o **printBoard(Board)** que chama o **printBorder(Size)** para a borda do tabuleiro e os elementos do tabuleiro, são impressos pelo predicado **printElementMid(Value)**.

Todo o código relativo à visualização do tabuleiro encontra-se no ficheiro *Board.pl*. O tabuleiro apresentado na consola é facilmente entendido, no entanto, é importante esclarecer que quadrados sombreados aparecem representados por um **X**.

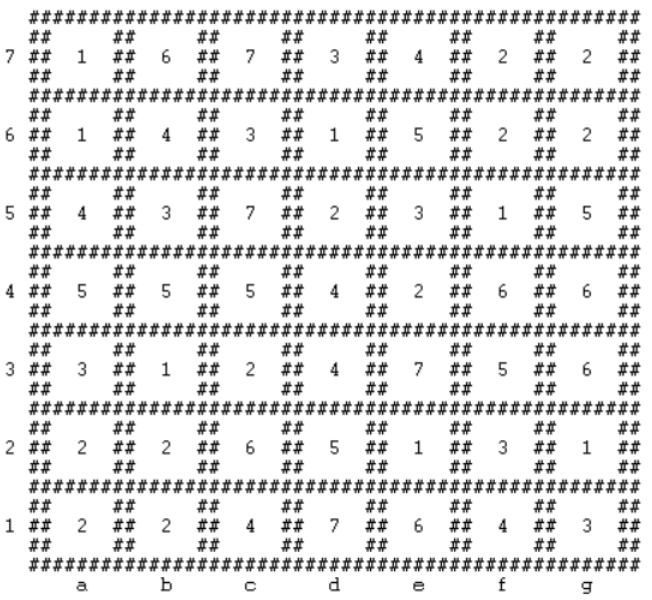

Figura 2. Representação de um tabuleiro, por resolver.

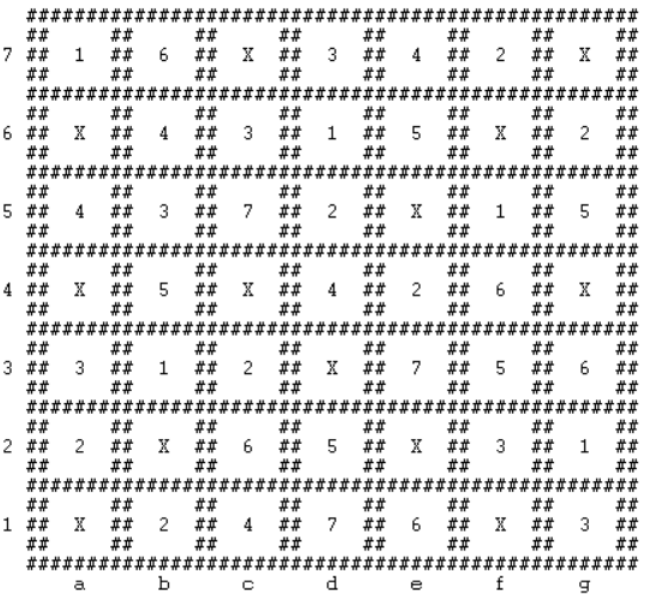

Figura 3. Representação de um tabuleiro, resolvido.

5 Resultados

Para testar a aplicação desenvolvida, corremo-la 10 vezes com cada tamanho N e apontámos o tempo que cada uma delas demora a encontrar a solução. Depois, usámos esses valores para calcular a média e elaborar o seguinte gráfico.

Figura 4. Resultados obtidos.

6 Conclusões e Trabalho Futuro

O uso de PLR (Programação em Lógica com Restrições) é muito útil para determinadas situações, uma vez que facilita imenso a programação. O nível de abstração com que o programador trabalha em PLR permite gerir problemas complexos de uma forma simples, rápida e com muito menos código do que usando linguagens imperativas.

No entanto, o facto de utilizarmos recursividade para imprimir o tabuleiro acaba por ser mais trabalhoso. O uso de ciclos, típicos das linguagens imperativas, facilitariam esta tarefa e reduziriam a quantidade de código escrita.

Em suma, consideramos o trabalho desenvolvido muito satisfatório, tal como a experiência com a programação lógica.

Referências

1. Hitori rules,
   http://www.conceptispuzzles.com/index.aspx?uri=puzzle/hitori/rules
2. Hitori techniques,
   http://www.conceptispuzzles.com/index.aspx?uri=puzzle/hitori/techniques
3. Nikoli, http://www.nikoli.com/en/puzzles/hitori/
4. SICStus Prolog, https://sicstus.sics.se/