Resolução e Geração de Puzzles Starry Night usando Programação em Lógica com Restrições

João Paulo Monteiro Leite and Márcia Isabel Reis Teixeira

FEUP-PLOG, Turma 3MIEIC4, Grupo Starry Night_1

Abstract. Projeto desenvolvido no âmbito da unidade curricular de Programação em Lógica, tendo como objetivo resolver um problema de decisão/otimização usando programação em lógica com restrições. Foi escolhido o problema do puzzle Starry Night, e desenvolvido um programa, no sistema de desenvolvimento SICStus Prolog, que resolve e gera estes puzzles.

1 Introdução

Este trabalho foi realizado no âmbito da unidade curricular Programação em Lógica, e tem como objetivo o desenvolvimento de um programa em Programação em Lógica com Restrições que resolva um problema de decisão combinatória, neste caso, puzzles Starry Night. O programa deve aceitar tabuleiros de tamanho variável e ser capaz de gerar puzzles.
O presente artigo está estruturado da seguinte forma:

2. Descrição do Problema: descrição detalhada do problema tratado.

3. Abordagem: apresentação da abordagem utilizada para modelar o problema como um problema de Programação em Lógica com Restrições, com descrição das variáveis de decisão, restrições e estratégia de pesquisa utilizada.

4. Visualização da Solução: descrição dos predicados usados para a visualização da solução em modo de texto.

5. Resultados: testes de diferentes estratégias de pesquisa com tabuleiros de tamanho variável e análise dos resultados.

6. Conclusões e Trabalho Futuro: conclusões do projeto e análise crítica dos resultados do mesmo.

2 Descrição do Problema

Puzzles Starry Night são puzzles 2D com tabuleiros quadrados com o mesmo número de células quadradas na vertical e na horizontal.

Para resolver o puzzle, deve por-se exatamente um círculo branco (representando o Sol), um círculo preto (representando a Lua) e uma estrela em cada coluna e linha do tabuleiro. Símbolos iguais não se podem tocar diagonalmente.

Um círculo ao lado do tabuleiro indica a cor do círculo mais próximo da estrela nessa linha ou coluna. Uma estrela indica que os círculos estão à mesma distância da estrela nessa linha ou coluna.

Pretende-se resolver e gerar puzzles deste tipo, com um tamanho de tabuleiro variável.

3 Abordagem

3.1 Variáveis de Decisão

Para a representação e resolução do problema consideram-se duas listas.

A primeira, normalmente referida no código como GivenBoard, representa os símbolos à volta do tabuleiro, ordenados da seguinte forma: primeiro os referentes às linhas, de cima para baixo, e de seguida os referentes às colunas, da esquerda para a direita.

A segunda, normalmente referida no código como SolutionBoard, representa os símbolos nas células do tabuleiro, ou seja, a solução. Os valores estão ordenados por linhas: primeiro os valores da primeira linha da esquerda para a direita, seguidos dos valores da segunda linha e assim sucessivamente.

Nas listas, um 0 representa um espaço vazio, 3 representa uma estrela, e 1 e 2 representam um círculo branco ou preto, respetivamente.

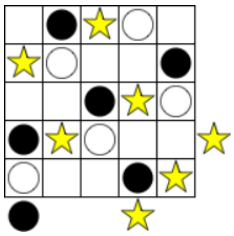


Fig. 1. Exemplo de puzzle Starry Night resolvido.

O puzzle da Figura 1 seria representado da seguinte forma: GivenBoard = [0,0,0,3,0,2,0,0,3,0], SolutionBoard = [0,2,3,1,0,3,1,0,0,2,0,0,2,3,1,2,3,1,0,0,1,0,0,3,0].

3.2 Restrições

Cada linha e coluna do SolutionBoard têm exatamente um círculo de cada cor e uma estrela. Esta restrição é implementada usando o predicado global_cardinality/2 sob as várias linhas e colunas do tabuleiro. A restrição está implementada no predicado checkOneSymbolPerLineAndColumn/3.

Símbolos iguais não se tocam diagonalmente. Esta verificação é feita célula a célula, começando na última linha do tabuleiro e terminando na segunda. Para cada célula, caso o seu conteúdo seja diferente de 0 (não esteja vazia), o conteúdo das células nos seus cantos superiores esquerdo e direito deve ser diferente do seu. A restrição está implementada no predicado checkNoDiagonalsBoard/3.

Regras relativas aos símbolos fora do tabuleiro. Para cada linha e coluna do SolutionBoard é verificado o símbolo correspondente no GivenBoard. Se for uma estrela ou um círculo, é calculada a distância entre os círculos e a estrela da linha/coluna do SolutionBoard através dos seus índices, e depois colocadas as restrições a estas distâncias de acordo com as regras explicadas na secção 2. São usados os predicados domain/3, all_distinct/1 e element/3, e a restrição está implementada no predicado checkOffBoardSymbols/3.

3.3 Estratégia de Pesquisa

Relativamente ao solucionador, foram testadas várias estratégias de etiquetagem, tendo sido estudadas em mais detalhe as opções de ordenação de variáveis leftmost e occurrence e as opções de seleção de valores step e middle. Durante os testes, depois de eliminadas várias opções que eram claramente menos eficientes, concluiu-se que o fator que mais influenciava a eficiência era a dificuldade específica do puzzle, o que seria difícil avaliar (de forma a escolher a melhor estratégia para o puzzle em causa) sem o resolver. Devido a esta condição, optou-se pela opção que produziu resultados mais consistentes para os vários puzzles de diferentes tamanhos e dificuldades, sendo esta a opção de ordenação de variáveis leftmost e a opção de seleção de valores middle.

Para o gerador, pretendia-se que a escolha do puzzle a apresentar ao utilizador fosse aleatória, pelo que foi utilizada a implementação da opção selRandom presente nos diapositivos da unidade curricular.

4 Visualização da Solução

A visualização do tabuleiro é feita com recurso ao predicado displayPuzzle/2, que apresenta o tabuleiro em modo de texto, numa forma aproximada à apresentada na figura 1. Nesta apresentação, '*' representa uma estrela e '0' e '@' representam, respetivamente, os círculos brancos e pretos.

O predicado displayPuzzle/2 recebe as listas correspondentes ao GivenBoard e ao SolutionBoard. Usa-se o predicado boardToLists/2 para organizar o SolutionBoard numa lista de listas, em que cada lista é uma linha do tabuleiro, de forma a facilitar os restantes predicados de visualização. Usa-se depois o predicado printBoard/4, que imprime o tabuleiro linha a linha, imprime os símbolos do GivenBoard correspondentes depois de cada linha e, por fim, imprime a linha de símbolos do GivenBoard relativos às colunas.

No solucionador starry_night/1 é impresso o tabuleiro resolvido completo e o tempo de resolução do mesmo em milissegundos.

```
| ?- starry_night([0,0,0,3,0,2,0,0,3,0]).
Time: 15ms
```

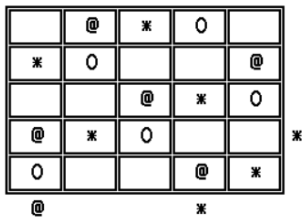
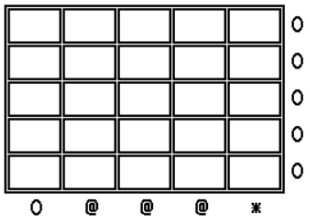


Fig. 2. Exemplo de output do solucionador.

No caso do gerador generate_starry_night/1 é impresso o tabuleiro por resolver, apenas com os símbolos gerados à volta, o tempo de geração em milissegundos, e o GivenBoard gerado em formato de lista, de forma a facilitar uma chamada ao solucionador com o tabuleiro gerado.

```
| ?- generate_starry_night(5,B).
Time: 703ms
```




```
[1,1,1,1,1,1,2,2,2,3]
B = [1,1,1,1,1,1,2,2,2,3] ?
```

Fig. 3. Exemplo de output do gerador.

5 Resultados

Foram executados testes ao predicado starry_night/1 com várias estratégias de etiquetagem de forma a determinar qual a mais adequada ao problema. Foram testadas em mais detalhe as opções de ordenação de variáveis leftmost e occurrence e as opções de seleção de valores step e middle.

Os testes foram realizados fazendo variar o tamanho do tabuleiro, e foram observados os valores do tempo de execução e o número de retrocessos (backtracks). Os resultados destes testes encontram-se ilustrados nos gráficos da figura . Não se colocaram dados referentes às opções occurrence, middle com um tabuleiro 9x9, dado que o predicado não terminou com um dos tabuleiros usados para teste neste caso.

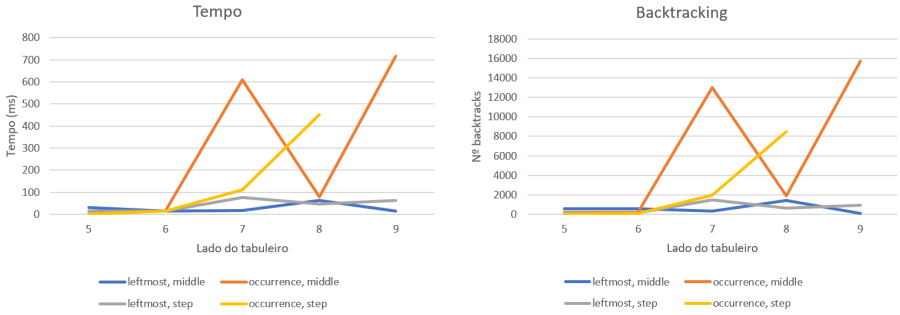


Fig. 4. Gráficos da variação do tempo de execução e número de retrocessos com o tamanho do tabuleiro.

Verifica-se que os resultados para a opção occurrence juntamente com a opção middle variam de forma bastante inconsistente com o tamanho. Alterando para a opção step, verifica-se um aumento muito significativo dos dois parâmetros estudados com o aumento do tamanho, chegando mesmo a não terminar. Estas opções não são, portanto, as mais adequadas ao problema.

Para a opção leftmost verifica-se que, tanto juntamente com a opção middle como com a opção step, os resultados são mais consistentes e eficientes. Pela sua maior eficiência com tabuleiros 7x7 e 9x9, considerou-se a opção middle a mais adequada.

6 Conclusões e Trabalho Futuro

Com o desenvolvimento deste projeto foram consolidados e aprofundados os conhecimentos adquiridos sobre Programação em Lógica com restrições ao longo das aulas teóricas e práticas da unidade curricular. Concluiu-se que Prolog é uma linguagem adequada e eficiente para a resolução de problemas do tipo dos abordados neste projeto.

Considera-se que o projeto foi realizado com sucesso, tendo-se conseguido a geração e resolução dos puzzles pretendidos de forma eficiente. Poderiam ser feitas algumas melhorias na organização do código.

O trabalho desenvolvido tem como uma limitação a utilização da mesma estratégia de etiquetagem para todos puzzles. Esta limitação poderia ser ultrapassada estudando de maneira mais aprofundada as várias técnicas e encontrar

padrões nos puzzles para que cada uma é mais eficiente, possibilitando assim a escolha de uma técnica de etiquetagem mais apropriada para cada puzzle. Isto não foi possível dentro dos limites de tempo para a realização do trabalho.

Bibliografia

1. Starry Night Puzzles, https://www2.stetson.edu/ efriedma/puzzle/night/. Último acesso 5 Jan 2020
2. Página Moodle da Unidade Curricular de Programação em Lógica, https://moodle.up.pt/course/view.php?id=2133. Último acesso 5 Jan 2020
3. Documentação de Prolog em SWI Prolog, https://www.swi-prolog.org/. Último acesso 5 Jan 2020