Resolução de Problema de Decisão usando Programação em Lógica com Restrições - Chess Num

Mariana Oliveira Ramos[up201806869],
Pedro Varandas da Costa Azevedo da Ponte[up201809694]

Mestrado Integrado em Engenharia Informática e Computação – 3º Ano
Programação em Lógica 2020/2021
Chess Num Grupo ?
Faculdade de Engenharia da Universidade do Porto, R. Dr. Roberto Frias,
4200-464 Porto, Portugal

Comentado [Springer-1]: Chinese authors should write their first names in front of their surnames. This ensures that the names appear correctly in the running heads and the author index.

Resumo. Este artigo tem como objetivo demostrar o processo de desenvolvimento do segundo projeto da Unidade Curricular de Programação em Lógica, assim como os resultados obtidos. O projeto consiste num programa escrito em Prolog, capaz de resolver qualquer instancia do puzzle Chess Num, cuja descrição poderá ser encontrada em https://erich-friedman.github.io/puzzle/chessnum/. Este for modelado como um PSR (Problema de Satisfação de Restrições) e resolvido utilizando a bibliotecao ‚clpfd' do SICStus Prolog, que permite implementação de PLR (Programação em Lógica com Restrições). O programa também possui a capacidade de gerar novos puzzle dinamicamente. Após análise de eficiencia, concluimos que o programa executa de forma quase instantanea para problemas com dimensao (?), subindo sempre de modo exponencial. O gerador de problemas consegue quase sempre concluir, em poucos segundos, a criação de novos puzzles até dimensão (?).

Palavras-chave: puzzle, clp, prolog, plog, feup

1 Introdução

Este trabalho foi desenvolvido no âmbito da unidade curricular de Programação em Lógica do Mestrado Integrado em Engenharia Informática e Computação, tendo como objetivo aprofundar o conhecimento teórico e prático da matéria lecionada referente à resolução de problemas com restrições em Prolog, utilizando a biblioteca ‚clpfd'.
O tema escolhido pelo grupo foi o puzzel 2D Chess Num. Este consiste em

Este artigo descreve detalhadamente a abordagem seguida na resolução do problema, os resultados obtidos e conclusões. Está estruturado na seguinte forma:

1. Descrição do problema
2. Abordagem
   - Variaveis de Decisão
   - Restrições
   - Função de Avaliação

Geração de Puzzels
3. Visualização da solução
4. Resultados Obtidos
5. Conclusões

2 Descrição do problema

Chess Num é um puzzle realizado num tabuleiro de xadrez, por default 8x8, que inicialmente é preenchido com números que indicam quantas vezes uma célula é atacada.

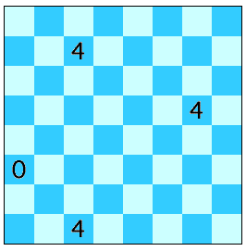

Figura 1: Exemplo do Puzzle Chess Num

O objetivo é preencher 6 casas com 6 peças (um Rei, uma Rainha, uma Torre, um Bispo, um Cavaleiro, e um Peão) de forma a que não haja duas peças no mesmo quadrado, nenhuma peça esteja num quadrado numerado e os ataques a cada célula correspondam ao numero de ataques indicado no tabuleiro inicial.

Em baixo está representada a solução do tabuleiro anterior.

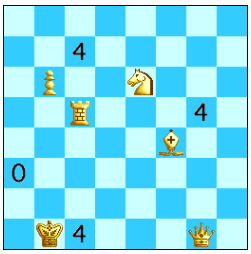

Figura 2: Exemplo do Puzzle Chess Num

Na célula [12] (linha 1 coluna 2) o número de ataques é 4, representando os possíveis ataques do Cavaleiro, Peão, Torre e Bispo. Na célula [51] o número de ataques é 0. Na célula [36] o número de ataques é 4, realizados pelo Bispo, Rainha, Cavaleiro e Torre.

Por último na célula [72] o número de ataques é 4 representando os ataques possíveis do Rei Rainha, Torre e Bispo.

3 Abordagem

O primeiro passo na abordagem foi tentar perceber como modelar o puzzle como um problema de restrições. Entender as variáveis de decisão a usar no predicado labeling, as restrições necessárias para o problema e restringir essas variáveis.

Foi ainda tida em conta a melhor forma de interagir com os utilizadores, ou seja, a melhor forma do puzzel ser visualizado. Sendo a consola SICStus Prolog muito simples, a representação das peças é feita com as letras K (Rei), Q (Rainha), R (Torre), C (Cavaleiro), B (Bispo), P (Peão).

3.1 Variáveis de Decisão

A solução pretendida para este puzzel é o mesmo tabuleiro, mas com o preenchimento das peças nas posições (linha e coluna) corretas. Neste sentido, a única variável de domínio que o nosso problema necessita e utilizada no predicado labeling, é uma lista Positions[] que contém a linha e coluna de cada peça que deve ser colocada. A lista tem a seguinte forma:

```
Positions = [KingRow, KingCol, QueenRow, QueenCol, RookRow, Rook
Col, BishopRow, BishopCol, KnightRow, KnightCol, PawnRow, PawnCo
l].
```

3.2 Restrições impostas

Em primeiro lugar, na inicialização da variável de decisão foi imposto que em cada célula o domínio é entre 0 e 7, representando o número da linha/coluna.

De seguida, foi necessário garantir que não existem duas peças na mesma posição. Para isso, foi desenvolvido o predicado differentPossitions(+Possitions) que coloca restrições de forma a garantir que não existem duas peças com a mesma linha e coluna em simultâneo,

```
differentPositions([KingR, KingC, QueenR, QueenC, RookR, RookC,
BishopR, BishopC, KnightR, KnightC, PawnR, PawnC]) :-
    KingPos #= 10 * KingR + KingC,
    QueenPos #= 10 * QueenR + QueenC,
    RookPos #= 10 * RookR + RookC,
    BishopPos #= 10 * BishopR + BishopC,
    KnightPos #= 10 * KnightR + KnightC,
    PawnPos #= 10 * PawnR + PawnC,
    KingPos #\= QueenPos,
    KingPos #\= RookPos,
    KingPos #\= BishopPos,
```

```
KingPos #\= KnightPos,
KingPos #\= PawnPos,
QueenPos #\= RookPos,
QueenPos #\= BishopPos,
QueenPos #\= KnightPos,
QueenPos #\= PawnPos,
RookPos #\= BishopPos,
RookPos #\= KnightPos,
RookPos #\= PawnPos,
BishopPos #\= KnightPos,
BishopPos #\= PawnPos,
KnightPos #\= PawnPos.
```

3.3 Estratégia de Pesquisa

3.4 Gerador Aleatório do Puzzel a Resolver

4 Visualização da Solução

Para além do tabuleiro final a nossa solução mostra algumas estatísticas como o tempo que demora a resolver o puzzle, se foi efetuado algum backtracking, o número de restrições feitas, entre outras.

Alguns cuidados para a otimização da resolução do puzzle foram tidos em conta, desta forma, foram criados alguns gráficos que permitem visualizar a diferença entre tempos de resolução para diferentes tamanhos do tabuleiro.

Estatísticas para um tabuleiro 6x6 e 10x10:

Estatísticas para um tabuleiro 15x15 e 20x20:

Estatísticas para um tabuleiro 50x50:

4.1 Eficiência do gerador de puzzles

4.2 Eficiência do solver do puzzle

5 Conclusões e Considerações Finais

Após a realização deste projeto em Prolog, conclui-se que

Ao longo do desenvolvimento deste projeto, foram encontradas algumas dificuldades, particularmente a geração de novos puzzles, assim como a apresentação de resultados, que foram mais complexos e demorados do que inicialmente previsto.

Apesar de todas as dificuldades encontradas, de forma geral, achamos que realizamos com sucesso todos os objetivos do projeto e o seu desenvolvimento contribuirá para uma melhor compreensão do funcionamento do uso de restrições em Prolog.

Bibliografia

1.
2.
3.