Chess-Num Puzzles Solver

Diogo Samuel Gonçalves Fernandes and Paulo Jorge Salgado Marinho Ribeiro

Faculdade de Engenharia da Universidade do Porto, Portugal,
FEUP-PLOG, Turma 3MIEIC06, Grupo Chess-Num_2
up201806250@fe.up.pt
up201806505@fe.up.pt

Abstract. Este segundo projeto da unidade curricular Programação em Lógica (PLOG) consiste na resolução de problemas de otimização/decisão, recorrendo ao uso de restrições em Prolog. No caso do nosso grupo, o objetivo é resolver os puzzles do tipo "Chess-Num", no menor tempo possível, com recurso a restrições, através do uso da biblioteca clpfd do SICStus Prolog.

Keywords: Programação em Lógica · Prolog · Restrições · clpfd · SICStus · Problemas de Otimização · Problemas de Decisão.

1 Introdução

A Programação em Lógica com Restrições trata-se de uma classe de linguagens de programação, que combina a declaratividade característica da programação em lógica e a eficiência da resolução de restrições. As suas principais aplicações baseiam-se na resolução de problemas de pesquisa ou otimização combinatória, tal como problemas de escalonamento, geração de horários, alocação de recursos, gestão de produção, entre outros. Dada a sua distinção e eficiência, a Programação em Lógica com Restrições tem diversas aplicações industriais e comerciais na atualidade, destacando-se a sua utilização na Renault, para planeamento de produção a curto prazo, na Nokia, para configuração de software para telemóveis, e na Siemens, para verificação de circuitos. Neste trabalho, aplicaremos estas capacidades na resolução de puzzles Chess-Num, que consistem na colocação de uma peça de Xadrez de cada tipo (Peão, Torre, Bispo, Cavalo, Rainha, Rei) num tabuleiro com casas numeradas, de modo a que todas as casas numeradas sejam atacadas N vezes, sendo N o número apresentado nessas casas. Alguns exemplos de puzzles deste tipo podem ser observados aqui: https://erich-friedman.github.io/puzzle/chessnum/ Este relatório procura explicar a nossa abordagem do problema de forma aprofundada e organizada nos seguintes tópicos:

- Descrição do Problema, com explicação de todas as regras a cumprir - Abordagem, onde explicaremos a nossa implementação do problema, com enumeração das variáveis de decisão e dos seus domínios - Visualização da Solução, com exploração dos predicados que permitem a visualização do problema resolvido, e respetivas imagens exemplificativas - Experiências e Resultados, onde faremos

a análise dimensional do problema, para distintas quantidades de células numeradas, e - Conclusões e Trabalho Futuro, onde iremos debater as principais conclusões que retiramos deste projeto, com base nos resultados obtidos, e sugerir formas de melhorar o trabalho desenvolvido - Referências, com enumeração das várias fontes bibliográficas que utilizamos para a procura de conhecimento - Anexos, que contêm imagens explicativas de alguma secção do relatório, e imagens exemplificativas do programa em execução

2 Descrição do Problema

O nosso tema aborda um problema de otimização, que consiste na resolução de um tipo de puzzle envolvendo peças de xadrez. No tabuleiro vão existir casas numeradas, de 1 a 6. A solução consiste em colocar uma peça de cada tipo (Peão, Cavalo, Rei, Torre, Bispo, Rainha) no tabuleiro, de modo a que cada uma destas casas seja atacada N vezes, sendo N o número presente na casa. As peças atacam como num jogo de xadrez normal: - O peão ataca na diagonal, para cima. Ou seja, ataca 2 casas distintas. - O cavalo ataca em L, pelo que ataca 8 casas - O bispo ataca todas as diagonais - A torre ataca todas as verticais e horizontais - O rei ataca todas as casas à sua volta, num alcance de uma casa (ataca 8 casas) - A rainha todas as diagonais, verticais e horizontais. É importante referir ainda que ao contrário do jogo de xadrez, é possível colocar os peões na primeira e última linha e não é possível colocar duas peças na mesma casa, nem numa casa que tenha numeração. Deve-se ter também em conta que tanto a torre como o bispo e a rainha não atacam uma dada casa se existir alguma peça entre eles, bloqueando o caminho.

3 Abordagem ao Problema

Descrever a modelação do problema como um PSR / Portugal

Para a resolução deste problema utilizando a *CLPFD* em *PROLOG* foi utilizada uma lista de listas para representar a o tabuleiro de xadrez.

Além disso é importante realçar que este representa um problema de satisfação de restrições, uma vez que é modelizado por variáveis que representam os diferentes aspetos do problema juntamente com os seus domínios e as restrições que limitam os valores que as variáveis podem ser dentro dos seus domínios. A sua solução é a atribuição de um valor que pertença ao domínio a cada variável de forma a que todas as restrições impostas pelo programa sejam satisfeitas.

3.1 Variáveis de Decisão

A solução ao problema encontra-se nas seguintes variáveis listadas abaixo. As variáveis *piece*X, *piece*Y representam a posição X e Y no tabuleiro de xadrez da peça *piece*.

- PawnX, PawnY
- KnightX, KnightY
- KingX, KingY
- RookX, RookY
- BishopX, BishopY
- QueenX, QueenY

Estas variáveis de decisão são reunidas numa lista, Positions, que será depois utilizada ao efetuar o labeling. Cada par destas variáveis corresponde à posição de uma dada peça no tabuleiro, sendo o primeiro elemento do par a linha onde a peça se encontra, e o segundo elemento a coluna. Assim, as variáveis pertencem ao domínio [1, N] em que N é o tamanho do tabuleiro. Estes são os valores possíveis que a linha/coluna podem ser no tabuleiro.

3.2 Restrições

As restrições a definir devem garantir a solução do problema, isto é, que para cada casa numerada, esta seja atacada N vezes, sendo N o número que contém. Para isto, foi necessário definir a zona de ataque de cada tipo de peça. Nos anexos, é possível observar estes predicados para cada tipo de peça (Peão, Cavalo, Rei, Torre, Bispo, Rainha). Em todos usamos restrições materializadas (reified), para que a variável Attack ficasse definida com o valor 1 no caso de as restrições que verificam se a célula é atacada por uma dada peça serem cumpridas, e 0 caso contrário. Assim, após serem chamados todos os seis predicados, basta somar os seis ataques retornados nos predicados de cada peça, e o resultado será o número de vezes que a casa está atacada, que terá de ser igual ao número que essa casa contém. Isto tudo é realizado no predicado cell_attacks, que verifica estas condições para uma dada casa numerada. A verificação destas restrições

para todas as casas numeradas é realizada no predicado principal, solve, com recurso ao predicado maplist, que aplica este predicado cell_attacks a cada uma das casas numeradas, que foram reunidas numa lista, no início do programa, numa chamada ao predicado getCellsNumber. É necessário ter em conta também os possíveis bloqueios de peças, nos casos da Torre, Bispo e Rainha. Para isto, criamos um predicado nothing_between, que verifica se não há nenhuma peça a bloquear o caminho entre a Torre/Rainha e a casa numerada, na horizontal e vertical. Da mesma forma, foi necessário criar o predicado nothing_between_diagonal, que verifica se não há nenhuma peça a bloquear o caminho entre Bispo/Rainha e a casa numerada, nas diagonais. Estas duas funções que analisam os bloqueios podem também ser observadas nos anexos (METER IMAGEM)

4 Visualização da Solução

Para uma melhor compreensão das soluções encontradas, decidimos implementar duas formas de apresentação das soluções: - Forma Escrita: Apresenta-se no ecrã as posições de cada uma das seis peças, no formato [Linha, Coluna]. Isto é efetuado pelo predicado show_results, que recebe a lista das posições das peças e o número da peça a que a próxima posição corresponde. Trata-se de um ciclo simples, e recorre ao predicado piece, que recebe o número da peça e retorna o respetivo nome. Tanto a implementacão deste predicado como o seu funcionamento com o programa em execução podem ser visualizados nos anexos. (LINK IMAGENS) - Tabuleiro: É apresentado um tabuleiro com as células numeradas e com as seis peças já colocadas conforme a solução encontrada, com uma respetiva legenda. Isto é efetuado pelo predicado display_solution, que chama o predicado add_pieces, responsável por substituir os valores das células dos tabuleiros contidos em Positions, pela peça de Xadrez correspondente. De seguida, é chamado o predicado display_board, que representa visualmente o tabuleiro já preenchido. Da mesma forma, é apresentado o tabuleiro do problema (apenas com as células numeradas) antes de se iniciar a procura da solução.

5 Experiências e Resultados

5.1 Análise Dimensional

Para o estudo do comportamento do programa face à dimensão do problema, consideramos dois tipos de testes: variação da dimensão do tabuleiro, e variação do número de células numeradas. Nos anexos, (METER FIGURA), é possível verificar os resultados para os testes do primeiro tipo, sendo que testamos, para um mesmo número de casas numeradas, distintos tamanhos para os tabuleiros. É possível verificar que o tempo de execução aumenta com a dimensão do tabuleiro, o que seria de esperar, uma vez que aumenta também o domínio das variáveis de decisão (é de 1 a N, sendo N o tamanho do tabuleiro, que é o valor mínimo e máximo que a Linha/Coluna da peça pode tomar, respetivamente), e portanto aumenta o número de testes efetuados pelas restrições.

Quando à variação do número de células numeradas, testamos para um mesmo tabuleiro (8x8), diferentes valores, o que pode ser verificado nos anexos (METER FIGURA). Como seria também de esperar, o tempo de execução aumenta com o número de casas numeradas, uma vez que aumenta também o número de restrições a ter em consideração, e, consequentemente, o número de tentativas a serem efetuadas pelo programa.

5.2 Estratégias de Pesquisa

De modo a detetar possíveis melhorias no tempo de resolução dos problemas, foram testadas diversas combinações para as opções de pesquisa do labeling. Nos anexos, (METER FIGURA), pode-se verificar os tempos de execução do programa para um mesmo problema (Problema 11), para diferentes combinações de opções de pesquisa. Após realizar estes testes, chegamos à conclusão que no nosso caso a melhor combinação de opções do labeling seriam a anti_first_fail e a bisect. A opção escolhida para a ordenação de variáveis (anti_first_fail) define que a próxima variável a ser escolhida na colocação das restrições é a variável mais à esquerda das que têm o maior domínio. A complementá-la, a opção escolhida para a seleção de valores define que os valores de uma variável são decididos através de uma escolha binária entre X #=< M e X #> M, onde M é o ponto médio do domínio de X (média entre valores mínimo e máximo do domínio de X, com arredondamento para baixo).

6 Conclusões e Trabalho Futuro

A realização deste trabalho permitiu a resolução de um problema através da utilização de restrições lógicas na linguagem PROLOG, através da utilização do módulo CLPFD. Durante a realização do mesmo foram encontradas diversas dificuldades, nomeadamente na elaboração dos ataques para a torre e para o bispo e rainha, devido aos possíveis bloqueios de peças que se encontrem entre estas peças e as casas numeradas. Com o tempo, descobrimos solução para esta dificuldade, recorrendo a restrições materializadas (reified).

Apesar de cumprir todos os requisitos pedidos no enunciado, há certos aspetos que poderiam ser melhorados futuramente, nomeadamente a questão da eficiência do programa, tendo em conta que o tempo de execução do programa é elevado no caso de tabuleiros com um elevado número de casas numeradas.

Em suma, o projeto foi concluído com sucesso, tendo em conta que implementamos todos os requisitos do enunciado, e conseguimos ultrapassar todas as dificuldades que enfrentamos durante o seu desenvolvimento. Implementamos também certas funcionalidades extra e interessantes, como a geração aleatória de tabuleiros, de diferentes dimensões, e com números distintos de casas numeradas. Para além disto, tornamos a interface de interação com o utilizador bastante simples e fácil de compreender, com representação gráfica dos tabuleiros antes e após as soluções. Este projeto permitiu-nos aplicar o conhecimento obtido nas aulas teóricas e práticas da unidade curricular, e consolidá-lo para que possamos aplicá-lo em situações futuras.

Sample Heading (Third Level) Only two levels of headings should be numbered. Lower level headings remain unnumbered; they are formatted as run-in headings.

Sample Heading (Fourth Level) The contribution should contain no more than four levels of headings. Table 1 gives a summary of all heading levels.

Table 1. Table captions should be placed above the tables.

Heading level	Example	Font size and style
Title (centered)	**Lecture Notes**	14 point, bold
1st-level heading	**1 Introduction**	12 point, bold
2nd-level heading	**2.1 Printing Area**	10 point, bold
3rd-level heading	**Run-in Heading in Bold.** Text follows	10 point, bold
4th-level heading	*Lowest Level Heading.* Text follows	10 point, italic

Displayed equations are centered and set on a separate line.

$$x + y = z \tag{1}$$

Please try to avoid rasterized images for line-art diagrams and schemas. Whenever possible, use vector graphics instead (see Fig. 1).

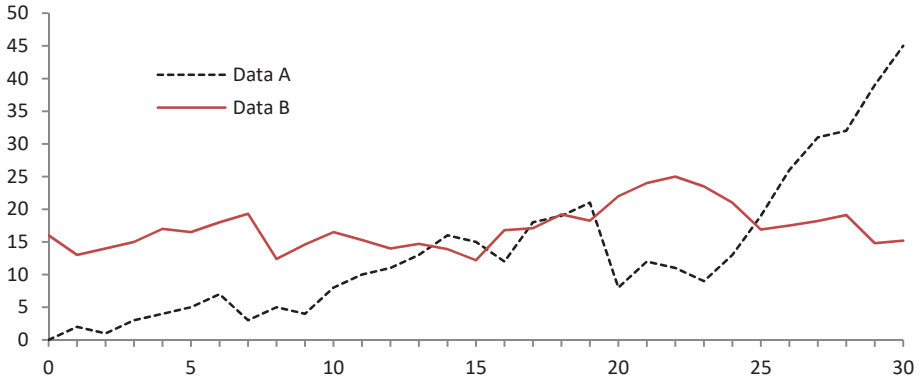

Fig. 1. A figure caption is always placed below the illustration. Please note that short captions are centered, while long ones are justified by the macro package automatically.

Theorem 1. *This is a sample theorem. The run-in heading is set in bold, while the following text appears in italics. Definitions, lemmas, propositions, and corollaries are styled the same way.*

Proof. Proofs, examples, and remarks have the initial word in italics, while the following text appears in normal font.

For citations of references, we prefer the use of square brackets and consecutive numbers. Citations using labels or the author/year convention are also acceptable. The following bibliography provides a sample reference list with entries for journal articles [1], an LNCS chapter [2], a book [3], proceedings without editors [4], and a homepage [5]. Multiple citations are grouped [1–3], [1, 3–5].

References

1. Author, F.: Article title. Journal **2**(5), 99–110 (2016)
2. Author, F., Author, S.: Title of a proceedings paper. In: Editor, F., Editor, S. (eds.) CONFERENCE 2016, LNCS, vol. 9999, pp. 1–13. Springer, Heidelberg (2016). https://doi.org/10.10007/1234567890
3. Author, F., Author, S., Author, T.: Book title. 2nd edn. Publisher, Location (1999)
4. Author, A.-B.: Contribution title. In: 9th International Proceedings on Proceedings, pp. 1–2. Publisher, Location (2010)
5. LNCS Homepage, http://www.springer.com/lncs. Last accessed 4 Oct 2017