Resolução de Problemas de Decisão usando Programação em Lógica com Restrições: Shakashaka

João Álvaro Ferreira and João Augusto Lima

Faculdade de Engenharia da Universidade do Porto, R. Dr. Roberto Frias, 4200-465 Porto

FEUP-PLOG, Turma 3MIEIC1, Shakashaka 3

Resumo Este artigo pretende descrever o trabalho efetuado no âmbito do segundo projeto da unidade curricular de Programação em Lógica do Mestrado Integrado em Engenharia Informática e Computação. O objetivo do trabalho em questão 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 no enunciado. Para este efeito, o problema escolhido foi a resolução e geração de tabuleiros com soluções possíveis do puzzle Shakashaka, sendo que o desenvolvimento foi feito em SICStus Prolog.

Keywords: Shakashaka; SICStus, Prolog, FEUP

1 Introdução

O objetivo deste trabalho é a demonstração dos conhecimentos sobre a resolução de problemas de otimização ou decisão combinatória com restrições em Prolog. Foi dada ao grupo a opção entre problemas de otimização e decisão e um leque de problemas variádos a resolver dentro destes âmbitos. Tomando em conta os problemas disponíveis aquando da escolha, foi escolhido o puzzle Shakashaka, que consiste num tabuleiro retangular de tamanho variável com quadrados pretos e brancos, sendo que o objetivo do puzzle é preencher esse tabuleiro com triangulos de modo a que apenas formas retangulares restem nos espaços brancos. Este artigo explica o problema detalhadamente.

2 Descrição do Problema

O puzzle Shakashaka é um dos muitos quebra-cabeças popularizados pela publicadora Nikoli, que publicou também Sudoku.

O puzzle é jogado num tabuleiro retangular com uma disposição em grelha quadriculada. Nesta, podem-se encontrar quadrados brancos ou negros, sendo que os negros podem conter um número. O jogador pode preencher os quadrados brancos com um quadrado meio preenchido com um triangulo (sendo o quadrado

dividido ao meio com uma diagonal e um dos lados preenchidos), havendo quatro possíveis preenchimentos de um quadrado pelo jogador - canto superior esquerdo, canto superior direito, canto inferior equerdo e canto inferior direito.

O puzzle encontra-se resolvido quando as unicas áreas brancas presentes são retângulos e quadrados fechados. Para este efeito, quadrados brancos também podem ser deixados completamente brancos. Quanto à numeração em quadrados pretos, esta indica quantos quadrados preenchidos com um triangulo devem ter um dos seus lados em contacto com o quadrado preto em questão e é uma específicação extra não sempre presente, podendo ir de 1 a 4.

Para além de ser capaz de resolver instâncias do puzzle Shakashaka que lhe sejam fornecidas, o programa tem também de ser capaz de gerar instâncias resolvíveis.

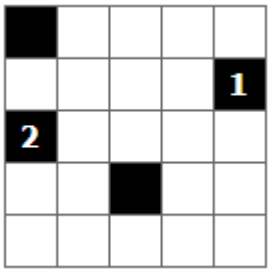

Fig. 1. Um exemplo do puzzle Shakashaka num tabuleiro 5x5

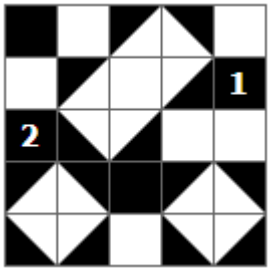

Fig. 2. Uma possível solução para a instância do Shakashaka mostrada na Fig. 1

3 Abordagem

Ao implementar o puzzle em questão em Prolog, o grupo decidiu que a representação do tabuleiro seria feita através de listas de listas. Na posição de um quadrado com uma determinada cor, número ou forma, utilizamos um número específico para cada um dos possíveis símbolos. Nomeadamente, para quadrados pretos com números utilizamos de números de 0 a 4, um quadrado preto sem número é o 5, um quadrado branco é representado pelo número 6 e os quadrados semi-preenchidos são repreentados pelos numeros 7 a 10. Servimo-nos também do número 11 para representar um quadrado branco re-escrito com branco, para efeito de lógica do programa.

```
tabuleiro(tabExemplo,[
    [5,6,6,6,6],
    [6,6,6,6,1],
    [2,6,6,6,6],
    [6,6,5,6,6],
    [6,6,6,6,6]
]).
```

Fig. 3. Representacao do nosso trabalho do puzzle utilizado como exemplo nas figuras 1 e 2

3.1 Variaveis de Decisão

A solução pretendida para este puzzle é o tabuleiro inicial com alguns dos seus quadrados brancos substituídos por quadrados semi-preenchidos de modo a que este esteja de acordo com as regras para uma solução do puzzle válida, seguindo as indicações referidas nas secções anteriores.

Como tal, as únicas variáveis de decisão são a lista de listas que representa o tabuleiro e o seu length, sendo estes também os argumentos da função de labeling.

3.2 Restrições

Restrição 1 - Para cada quadrado branco, ou este é reescrito para branco novamente (ficando com o código 11, como mencionado anteriormente) ou contem um e apenas um dos triângulos que o preenchem. Para verificar esta restrição, verifica-se se o quadrado a analisar tem um valor de 7 a 11. Para isto,

utilizamos a função *constraint1* e a função getPecaConstraint, que verifica se o valor do quadrado em questão é igual ou não ao valor indicado e, se sim, retorna 1 num valor auxiliar (caso contrario retorna 0). Os valores auxiliares são então todos somados e, caso o valor da soma não seja igual a 1, a solução não se aplica.

Restrição 2 - Para cada quadrado negro, inicialmente considera-se se não tem número. Nesse caso, os seus vizinhos brancos têm algumas restrições. Por exemplo, supondo que um deles é um determinado triângulo (ou quadrado semi-preenchido), o vizinho deste branco adjacente ao quadrado preto tem de ter um triângulo tal que estes formem um angulo de 45 graus.

Restrição 3 - No caso do quadrado preto ter um número, este terá de ter um número de quadrados semi-preenchidos adjacentes igual ao número indicado.

Restrição 4 - Para todo o quadrado semi-preenchido,este tem de formar uma área branca com outros triangulos e, como tal, os triangulos semi-preenchidos adjacentes a este têm de ser iguais e formar continuídade com este (fazendo ambos parte do mesmo lado de um retângulo maior) ou serem-lhe perpendicular, de modo a formar um canto do retângulo ou quadrado.

Restrição 5 - Para todo o quadrado semi-preenchido, os quadrados semi-preenchidos que lhe são adjacentes não podem ser iguais se dessa forma não formarem continuidade, já que dessa forma teremos uma área com um ângulo de 45 graus como canto o que leva a ser impossível formar um retângulo ou quadrado naquela área branca.

Restrição 6 - Exclusão de cantos côncavos. Devido à restrição 5, este efeito só poderia acontecer com quadrados brancos, já que esta torna impossível a geração de uma solução com cantos concavos que contenham quadrados semi-preenchidos.

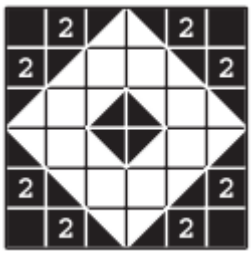

Fig. 4. Representacao de uma solução de uma instância do puzzle em que ocorre um quadrado nested sem a restrição 7

Restrição 7 - Exclusão de quadrados e retângulos brancos nested. Após todas as outras restrições, último problema que restava resolver para o bom solucionamento de todos os puzzles Shakashaka é a existência, em algumas soluções, de retângulos brancos que contêm outros quadrados, levando a áreas

que têm o contorno de um retÂngulo ou quadrado branco mas cujo interior não é completamente branco. A imagem indicada é um exemplo do que pode acontecer na resolução de um puzzle Shakashaka sem esta restrição.

3.3 Função de Avaliação

Este é um problema de decisão e, como tal, a avaliação de uma solução passa meramente pela verificação de que esta se encontra correta, o que é feito pelas restrições. Devido à natureza do problema como um problema de programação de inteiros 0-1, é frequente existir apenas uma solução. Tendo isto em conta, mesmo que o nosso programa calculasse todas as soluções possíveis, não seria possível determinar uma solução superior já que o único critério a cumprir é se o puzzle se encontra resolvido ou não.

3.4 Estratégia de Pesquisa

O labeling para a resolução deste problema é feito através da função criarLabeling que, com o auxílio da função criarLabelingElemento, atribui valores sequencialmente a todos os valores de um tabuleiro NovoTab. Isto é feito após a criação dasrestrições e um cut, levando a que os únicos tabuleiros aceites sejam válidos.

4 Visualização da Solução

O programa é corrido através de dois predicados principais - o dynamic(N), que gera tabuleiros resolvidos com tamanhos NxN, e o shakashaka(Tab), que apresenta a resolução de um determinado Tabuleiro Tab que tenha sido previamente transcrito para o código. Para visualizar a solução, criamos um predicado imprimirTabuleiro que imprime o tabuleiro do jogo. Devido às conceções que tivemos de fazer para poder abstrair o puzzle de imagens com quadrados pretos, brancos e semi-preenchidos de várias fomas, a impressão do tabuleiro não seria facilmente interpretável sem algumas alterações.

Para o efeito, decidimos simular os caractéres do jogo original que pretendemos representar em ASCII com o predicado imprimirLine. Aqui se encontram alguns exemplos desse predicado:

Sendo que o resultado, utilizando o tabuleiro que resolvemos no início deste artigo, é o seguinte:

5 Resultados

Coisas a adicionar depois de fazer os testes

```
imprimirLine([9|Tail],Tamanho,2):-
    write('|'),
    write('# /'),
    imprimirLine(Tail,Tamanho,2).
imprimirLine([7|Tail],Tamanho,2):-
    write('|'),
    write('  /'),
    imprimirLine(Tail,Tamanho,2).
imprimirLine([8|Tail],Tamanho,2):-
    write('|'),
    write('\\  '),
    imprimirLine(Tail,Tamanho,2).
imprimirLine([10|Tail],Tamanho,2):-
    write('|'),
    write('\\ #'),
```

Fig. 5. Exemplos do código para imprimir um tabuleiro interpretável

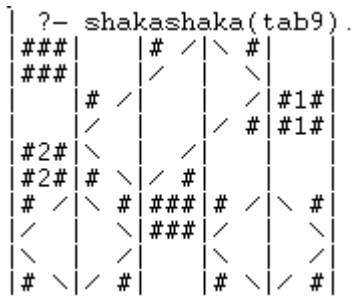

Fig. 6. Representacao da solução demonstrada nas Figuras 1 e 2 pelo nosso programa.

6 Conclusões e Trabalho Futuro

Este projeto levou o grupo à conclusão de que quanto à resolução de problemas de decisão e optimização, a linguagem PROLOG é muito vantajosa devido a todas as suas capacidades relevantes para a efetuação de labeling de variáveis e aplicação de restrições (não pudemos explorar com profundidade a questão da avaliaão de resultados pelo que não faria sentido ofercer uma conclusão relativa a estes). No que toca a rapidez em procesos lógicos e optimização, PROLOG é o ideal.

No entanto, embora seja poderosa nesses aspetos, é limitada em aspetos como o display de caractéres especiais e a maior dificuldae que existe em imprimir um tabuleiro, que embora não necessariamente difíceis podem-se tornar trabalhosas.

Podemos afirmando que julgamos ter cumprido os nossos objetivos neste trabalho de forma satisfatória.

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