# Resolução de problemas de decisão usando Programação em Lógica com Restrições: Gap Puzzle

Bernardo Santos[201706534] and Vítor Gonçalves[201703917]

[1] FEUP-PLOG, 3MIEIC06, Gap_Puzzle_2
[2] Faculdade de Engenharia da Universidade do Porto, Rua Doutor Roberto Frias, s/n 4200-465, Porto PORTUGAL incoming@fe.up.pt
http://www.fe.up.pt

**Resumo:** O projeto desenvolvido na unidade curricular de Programação em Lógica, usando o sistema de desenvolvimento SICSTUS Prolog, tem como finalidade resolver um puzzle utilizando restrições. O puzzle escolhido foi o 'Gap Puzzle' e tem como objetivo resolver um puzzle onde apenas podem existir dois blocos em cada linha e em cada coluna, e estes não podem tocar-se. Para além disso, quando for especificado, os blocos de uma linha ou coluna têm de respeitar a distância que é pedida. Para além da criação do solver do puzzle foi também implementado predicados que permitem gerar puzzles aletórios válidos. A abordagem utilizada permite lidar com tabuleiros de diversos tamanhos. Este artigo tem como objetivo mostrar a abordagem utilizada, bem como a análise das soluções obtidas.

**Keywords:** FEUP · Prolog · Sicstus · Restrições · Gap Puzzle · Lógica

## 1 Introdução

O projeto tem como objetivo a aplicação do conhecimento adquirido e abordado durante as aulas teóricas e práticas da unidade curricular Programação em Lógica do 1º semestre do 3º ano do Mestrado Integrado em Engenharia Informática e de Computação, tendo como foco a resolução de um Problema de Decisão/Otimização usando Programação em Lógica com Restrições.

O grupo optou pela escolha de um problema de decisão, escolhendo para tal efeito o Gap Puzzle.

O alvo deste artigo passa pela exposição e esclarecimento de qualquer dúvida que possa ter surgido, descrevendo a abordagem utilizada pelo grupo, focando essencialmente em Variavéis de Decisão, Restrições e Estratégia de Pesquisa, bem como análise aos resultados e principais conclusões retiradas.

O artigo está dividido da seguinte forma:

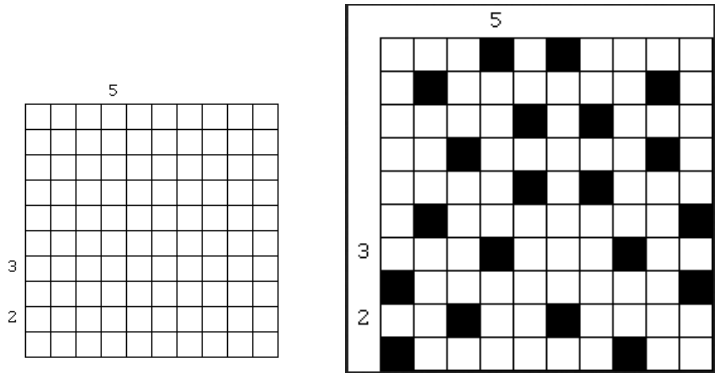
1. **Descrição do problema:** descrição do problema de decisão em análise.
2. **Abordagem:** descrição da modelação do problema, de acordo com as subsecções seguintes:
   - Variáveis de Decisão: descrição das variáveis de decisão e os seus domínios, bem como o seu significado no contexto do problema.
   - Restrições: descrição das restrições e a sua implementação.
   - Estratégia de Pesquisa: descrição da estratégia de etiquetagem utilizada.
3. **Geração Dinâmina de Problemas:** descrição das estratégias usadas para a geração de problemas.
4. **Visualização da Solução:** explicação dos predicados que permitem visualizar a solução em modo de texto.
5. **Resultados:** demonstração de exemplos de aplicação em instâncias do problema com diferentes complexidades e análise dos resultados obtidos.
6. **Conclusões e Trabalho Futuro:** conclusões retiradas ao longo do desenvolvimento do projeto, resultados obtidos e aspetos que podem ser melhorados.

## 2 Descrição do Problema

O Gap Puzzle caracteriza-se por ser um puzzle quadrangular, uma vez que o seu número de linhas e colunas é o mesmo. É um puzzle cujo tabuleiro tem um número mínimo de 8 células, mas que pode ter um tamanho variável superior ou igual a este valor.

O tabuleiro começa vazio e pode ou não apresentar algumas pistas na parte exterior do tabuleiro. Para resolver o puzzle, basta preencher cada linha e cada coluna exatamente com dois blocos, sendo que estes não se podem tocar. Os números que por vezes se encontram no exterior do puzzle, junto a linhas ou a colunas, servem para indicar a distância de células que tem de haver entre os dois blocos daquela linha ou coluna.

A figura do lado esquerdo mostra o tabuleiro inicial e a do lado direito a resolução desse mesmo puzzle.




Tabuleiro Inicial Tabuleiro Final

Figura 2: Puzzle e sua respetiva solução

## 3 Abordagem

De forma a resolver este puzzle, foi necessário encontrar uma forma viavél e simples de passar a informação precisa para o Prolog.

Desta forma, a abordagem encontrada foi criar um ficheiro à parte, que contém apenas informação sobre puzzles, cada um deles com o número do puzzle em questão, e com uma lista de distâncias para as colunas, e outra para as linhas.

De salientar que todos os tabuleiros são quadrangulares e têm um tamanho mínimo de 8 células. Foi decidido também, que 1 servirá para representar as células com blocos, e células vazias serão representadas com um 0.

Na imagem abaixo pode ser visto o predicado que retrata aquilo que foi falado.

```
% puzzle(Number, Col, Row).
%
% Number - number of the puzzle
% Col - spacing between the corresponding column elements
% Row - spacing between the corresponding row elements

% ----------========== PUZZLE PROVIDED ON THE WEBSITE ==========----------
puzzle(1,
    [_, 5, _, _, _, _, _, _],
    [_, _, _, _, _, _, 4, _]
    ).

puzzle(2,
    [_, _, 5, _, _, _, _, _],
    [_, _, _, _, _, _, 4, _]
    ).
```

Figura 3: Parte do ficheiro puzzles.pl

### 3.1 Variavéis de Decisão

No nosso puzzle, variavéis de decisão correspondem às células do tabuleiro. O domínio das mesmas varia entre 0 e 1, sendo o 0 correspondente a células vazias e 1 às células que terão associado um bloco.

### 3.2 Restrições

Para resolver o puzzle corretamente foi necessário implementar algumas restrições.

Inicialmente, serão aplicadas restrições a todas a linhas do puzzle, através do predicado **itr_line**. Este predicado impõe todas as restrições.

Neste predicado é utilizado um outro predicado da biblioteca **clpfd**, o **global_cardinality**, que vai restringir o número de 1's por linha a 2 e todo o resto da linha a 0.

Seguidamente, é chamado um predicado novo, **space_rest(Space, Line, PrvPos, OutPos)**, que é responsável por fazer cumprir as distâncias mínimas entre blocos. O primeiro argumento é o espaçamento entre os 1's da linha **Line** que vamos iterar. Os últimos dois argumentos são tuplos com a posição dos blocos nas linhas, no caso do penúltimo argumento são em relação à linha anterior da que estamos a iterar, e o último argumento refere-se à linha atual.

Se o primeiro argumento passado for uma variável livre, significa que não há uma distância concreta entre os dois blocos, portanto, é aplicada a restrição de que o espaçamento entre os blocos dessa linha terá de ser maior que 1. Caso contrário, a diferença da posição do bloco final e da posição do bloco inicial terá de ser igual à distância passada como primeiro argumento.

Por fim, o predicado **space_rest** chama o predicado **surround_rest(CurPos, PrvPos, OutPos)**, que aplica a restrição de que cada 1 deve estar rodeado de 0's, ou seja, nenhum bloco se pode tocar, nem mesmo nos cantos. Este predicado aplica essa restrição para a linha atual calculando as posições nas quais os blocos não podem ser colocados nas posições em que os blocos das linhas anteriores foram colocados.

Após iterarmos sobre todas as linhas do puzzle, será feito o mesmo procedimento para todas as colunas do puzzle, tendo no final, uma solução válida.

### 3.3 Estratégia de Pesquisa

Após um longo período de debate e de testes, a estratégia de etiquetagem usada foi a **default**, onde são utilizadas as técnicas **up**, **leftmost** e **step**. No entanto, é importante referir que utilizando outras opções de *labeling* a performance do programa não é significativamente alterada na grande maioria dos puzzles testados devido à quantidade de restrições aplicadas e pela estratégia de implementação das mesmas.

## 4 Geração Dinâmica de Problemas

Para gerar novos puzzles foram implementadas duas estratégias de geração distintas.

A primeira resolve um puzzle vazio com as dimensões desejadas, utilizando uma estratégia de etiquetagem diferente da normal. O predicado que implementa esta estratégia é o **generate1(PuzzleNumber, Size, NumHints)** seleciona aleatoriamente as variáveis e os valores na fase de *labeling*, o que torna a geração um pouco lenta para puzzles de grandes dimensões. Após a resolução do puzzle, são selecionadas **NumHints** linhas e colunas aleatóriamente onde serão colocadas as pistas para a resolução do puzzle.

A segunda forma de geração de puzzles é implementada pelo predicado **generate2(PuzzleNumber, Size, NumHints)**, é menos eficiente mas tende a ser mais rápida para puzzles de grande dimensão e com poucas pistas, isto porque primeiro são geradas pistas aleatoriamente e só depois é que se verifica se o puzzle tem solução ou não.

Após um puzzle ser gerado, tanto a primeira como a segunda estratégias guardam o puzzle no ficheiro *puzzles.pl*. O puzzle é guardado com recurso ao predicado **save_puzzle(PuzzleNumber, Col, Row)** que guarda um puzzle com o número **PuzzleNumber** no ficheiro de puzzles com o formato correto.

## 5 Visualização da Solução

De forma a facilitar a validação da solução encontrada, foram implementados alguns predicados que permitem visualizar a solução em modo de texto.

Os predicados encontram-se todos num ficheiro separado, display.pl, e o predicado principal que chama todos os outros predicados auxiliar é *print_board(Board, Col, Row)*. Este predicado, começa por escrever todas as distâncias que devem ser cumpridas, primeiro as correspondentes às colunas e depois às linhas. É ainda apresentando um separador entre cada linha do tabuleiro, função pertencente ao predicado *print_line(Col)*. Finalmente, para se conseguir visualizar a solução, é usado o predicado *print_row(Row, Board)*, que percorre todas as linhas do tabuleiro e as desenha, com respetivos separadores.

Para além do puzzle resolvido, é ainda mostrado ao utilizador o tempo, em segundos, que demorou a ser encontrada a solução.

Abaixo, uma imagem que mostra o resultado da visualização em modo texto.


Figura 4: Resultado final

## 6 Resultados

De modo a ser possível retirar conclusões sobre os resultados do nosso programa, foram medidos vários tempos de resolução do puzzle, tendo estes sido retirados com diferentes tamanho de puzzles e com várias opções de *labeling*.

Independentemente do tamanho do puzzle, a procura de uma solução válida é bastante rápida. Em relação às opções etiquetagem, foram usadas as **default**, no entanto, qualquer outras opções que fossem utilizadas não afetavam o tempo de procura significativamente, sendo os resultados muito parecidos.

A tabela com os tempos obtidos, em milissegundos, está disponível no anexo A. para obter estes resultados foi executada a resolução de cada puzzle cinco vezes e foi calculada a média dos tempos obtidos através do predicado **it_test(Stream, Iterations, Options, Number)**.

## 7 Conclusões e Trabalho Futuro

No decorrer do projeto foram encontradas algumas dificuldades, principalmente com a escolha das opções de *labeling*, uma vez que os tempos de pesquisa apresentados variavam ligeiramente. No entanto, com a consulta à documentação do SICSTUS e da biblioteca clpfd foi encontrada a solução que mais beneficiava a nossa solução.

Os objetivos propostos e delineados pelo grupo foram alcançados com sucesso, uma vez que é possível resolver diversos tipos de puzzles com vários tamanhos, bem como a geração de puzzles válidos e aleatórios, com ótimos resultados.

O trabalho que agora se conclui exigiu um grande esforço de ambos os elementos do grupo, mas no final achamos que foi uma experiência diferente mas ao mesmo tempo bastante enriquecedora e satisfatória.

Em suma, estamos extremamente contentes e satisfeitos com o resultado final apresentado, tendo este contribuindo bastante para a melhor compreensão do paradigma dos Problemas de Otimização/Decisão com Restrições.

## 8 Bibliografia

1. Gap Puzzle rules, https://www2.stetson.edu/ efriedma/puzzle/gap/. Last accessed 31 Dec 2019 Oct 2017
2. Biblioteca clpfd, https://sicstus.sics.se/sicstus/docs/4.1.0/html/sicstus/lib_002dclpfd.html. Last accessed 1 Jan 2020
3. Manual SICSTUS, https://sicstus.sics.se/sicstus/docs/3.12.5/pdf/sicstus.pdf. Last accessed 28 Dec 2019
4. Slides de apoio das aulas teóricas, https://moodle.up.pt/course/view.php?id=2133. Last accessed 1 Jan 2020

# 9 Anexos

## A Resultados da variação das opções de labeling

| Tamanho | Nº | Ordenação de Variáveis | | | | | | | | Seleção de Valores | | | | Ordenação de Valores | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | leftmost | min | max | ff | anti_first_fail | occurrence | ffc | max_regret | step | enum | bisect | median | up | down |
| 9x9 | 1 | 0.004 | 0.005 | 0.004 | 0.003 | 0.005 | 0.004 | 0.005 | 0.004 | 0.004 | 0.004 | 0.004 | 0.004 | 0.004 | 0.023 |
| | 2 | 0.014 | 0.014 | 0.014 | 0.014 | 0.014 | 0.015 | 0.014 | 0.014 | 0.014 | 0.014 | 0.014 | 0.014 | 0.014 | 0.011 |
| | 3 | 0.008 | 0.008 | 0.008 | 0.008 | 0.009 | 0.008 | 0.008 | 0.008 | 0.007 | 0.008 | 0.008 | 0.008 | 0.008 | 0.013 |
| | 4 | 0.015 | 0.015 | 0.015 | 0.015 | 0.016 | 0.015 | 0.015 | 0.015 | 0.015 | 0.015 | 0.015 | 0.015 | 0.015 | 0.040 |
| | 5 | 0.043 | 0.042 | 0.043 | 0.043 | 0.043 | 0.043 | 0.045 | 0.043 | 0.043 | 0.042 | 0.043 | 0.043 | 0.042 | 0.019 |
| | 6 | 0.030 | 0.030 | 0.031 | 0.032 | 0.031 | 0.032 | 0.031 | 0.030 | 0.033 | 0.031 | 0.030 | 0.030 | 0.031 | 0.014 |
| | 17 | 0.004 | 0.005 | 0.004 | 0.004 | 0.004 | 0.004 | 0.004 | 0.004 | 0.004 | 0.004 | 0.004 | 0.005 | 0.004 | 0.005 |
| 10x10 | 7 | 0.506 | 0.502 | 0.507 | 0.526 | 0.511 | 0.522 | 0.513 | 0.510 | 0.523 | 0.512 | 0.515 | 0.517 | 0.522 | 0.733 |
| | 8 | 0.527 | 0.523 | 0.519 | 0.536 | 0.524 | 0.536 | 0.523 | 0.528 | 0.522 | 0.523 | 0.527 | 0.540 | 0.537 | 0.159 |
| | 9 | 0.064 | 0.065 | 0.064 | 0.069 | 0.064 | 0.065 | 0.064 | 0.063 | 0.066 | 0.066 | 0.065 | 0.068 | 0.064 | 0.706 |
| | 10 | 0.114 | 0.119 | 0.115 | 0.121 | 0.116 | 0.115 | 0.123 | 0.115 | 0.117 | 0.117 | 0.123 | 0.121 | 0.114 | 0.037 |
| | 11 | 0.270 | 0.299 | 0.271 | 0.281 | 0.272 | 0.272 | 0.276 | 0.270 | 0.271 | 0.276 | 0.272 | 0.277 | 0.287 | 0.100 |
| | 12 | 0.029 | 0.029 | 0.028 | 0.029 | 0.029 | 0.028 | 0.030 | 0.029 | 0.030 | 0.033 | 0.029 | 0.028 | 0.030 | 0.113 |
| | 13 | 0.065 | 0.067 | 0.066 | 0.069 | 0.065 | 0.066 | 0.070 | 0.066 | 0.070 | 0.066 | 0.069 | 0.066 | 0.066 | 0.142 |
| | 14 | 0.006 | 0.007 | 0.006 | 0.006 | 0.007 | 0.006 | 0.007 | 0.006 | 0.006 | 0.006 | 0.007 | 0.007 | 0.007 | 0.007 |
| | 18 | 0.129 | 0.139 | 0.128 | 0.129 | 0.129 | 0.129 | 0.130 | 0.129 | 0.132 | 0.132 | 0.136 | 0.142 | 0.138 | 0.033 |
| 15x15 | 15 | 0.046 | 0.048 | 0.048 | 0.048 | 0.047 | 0.047 | 0.049 | 0.048 | 0.049 | 0.048 | 0.049 | 0.047 | 0.046 | 0.042 |
| | 16 | 0.049 | 0.050 | 0.049 | 0.050 | 0.049 | 0.049 | 0.049 | 0.049 | 0.051 | 0.050 | 0.052 | 0.050 | 0.048 | 0.032 |
| 100x100 | 100 | 0.653 | 1.015 | 1.034 | 0.653 | 1.041 | 1.324 | 1.330 | 1.076 | 0.660 | 0.676 | 0.647 | 0.648 | 0.641 | 1.508 |

Figura 5: Tabela de resultados