Resolução de Problema de Decisão usando Programação em Lógica com Restrições: Close or Far

Diogo Silva & Mafalda Santos
FEUP-PLOG, Turma 3MIEIC05, Grupo Close or Far_2

Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias, s/n 4200-465 Porto PORTUGAL

Resumo. Este projeto foi desenvolvido no âmbito da unidade curricular de Programação em Lógica utilizando SICStus Prolog. O seu objetivo é a resolução de um problema de decisão através do uso de restrições. O *puzzle* a escolhido foi o "Close or Far", que passa por garantir que num tabuleiro N por N existem em cada linha e coluna exatamente duas letras 'C' e duas letras 'F', sendo a distância entre as letras 'C' obrigatoriamente menor do que a distância entre as letras 'F'. Assim, este artigo tem como objetivo explicar pormenorizadamente as etapas de resolução deste problema e as conclusões retiradas do mesmo.

Keywords: Close or Far, Prolog, SICStus, Restrições, FEUP

1 Introdução

Este projeto foi desenvolvido para a unidade curricular de Programação em Lógica do 1º semestre do 3º ano do Mestrado Integrado em Engenharia Informática e Computação (MIEIC) da Faculdade de Engenharia da Universidade do Porto (FEUP). Neste, foi necessário resolver um problema de decisão em Prolog utilizando restrições, tendo sido escolhido o *puzzle* "Close or Far", cujas regras serão descritas na próxima secção, entre um grupo de problemas semelhantes. Para além da resolução do problema e do escalamento efetuado para que fosse possível resolver qualquer tabuleiro N por N, é possível gerar dinamicamente novas instâncias do "Close or Far". Foram também efetuados alguns testes para saber como era afetado o algoritmo pelo aumento das dimensões do *puzzle* e variação da estratégia de pesquisa.

Posto isto, este artigo apresenta a seguinte estruturação:

- **Descrição do Problema.** Descrição com detalhe do problema de decisão em análise.
- **Abordagem.** Descrição da modelação do problema como um PSR/POR, de acordo com as subsecções seguintes:
  - **Variáveis de Decisão.** Descrição das variáveis de decisão, seus domínios e significado no contexto do problema em análise.
  - **Restrições.** Descrição das restrições rígidas e flexíveis do problema e a sua implementação utilizando o SICStus Prolog.

  - **Estratégia de Pesquisa.** Descrição da estratégia de etiquetagem (*labeling*) utilizada ou implementada, nomeadamente heurísticas de ordenação de variáveis e valores.
  - **Geração de Problemas.** Descrição da geração dinâmica de problemas, e.g. gerar aleatoriamente o problema a ser resolvido.
- **Visualização da Solução.** Explicação dos predicados que permitem visualizar a solução em modo de texto.
- **Resultados.** Exemplos de aplicação em instâncias do problema com diferentes dimensões e análise dos resultados obtidos.
- **Conclusões e Trabalho Futuro.** Conclusões retiradas da elaboração do projeto, vantagens, limitações e aspetos a melhorar da solução apresentada.
- **Bibliografia.** Livros, artigos e páginas Web usados para desenvolver o trabalho.
- **Anexo.** Código fonte, ficheiros, dados, resultados e outros elementos úteis.

2 Descrição do Problema

O *puzzle* "Close or Far" trata-se de um problema de decisão em que é apresentado um tabuleiro N por N com alguns dos espaços preenchidos com letras C e F, de Close e Far, respetivamente. É de salientar que só existem soluções para *puzzles* com o tamanho mínimo de 6x6.

F					
				F	
			C		
					C
	F				
		F			

Fig. 1. Exemplo de um *puzzle* "Close or Far" (6 x 6) não resolvido.

O objetivo passa então por analisar esse tabuleiro e concluir se existe uma solução que cumpra as regras estabelecidas pelo mesmo, i.e., cada linha e cada coluna tem de ter exatamente duas letras 'C' e duas letras 'F', sendo também necessário que a distância entre C's seja menor do que a distância entre F's.

F	C	C	F		
C	C	F		F	
C	F		C		F
F		C		F	C
	F		F	C	C
		F	C	C	F

Fig. 2. Solução do *puzzle* "Close or Far" anterior.

3 Abordagem

Para a representação de um tabuleiro foi utilizada uma lista. Dado um *puzzle*, a variável 'Vars' é unificada com a lista correspondente. Por exemplo, em **board_six(Vars)** estamos a unificar a lista que contém um tabuleiro fornecido de tamanho seis com 'Vars'.

['F', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'F', ' ', ' ', ' ', ' ', ' ', 'C', ' ', ' ',
' ', ' ', ' ', ' ', ' ', 'C', ' ', 'F', ' ', ' ', ' ', ' ', ' ', ' ', 'F', ' ', ' ', ' ']

Fig. 3. Lista do exemplo da Figura 1 após unificação com 'Vars'.

De seguida, utiliza-se o predicado **replace_all/8** que substitui os espaços em branco por '0', os C's por '1' e os F's por '2', de forma a ser possível processar o *puzzle*. Depois, procede-se à substituição dos '0' por variáveis com o predicado **maplist/3** e auxílio do predicado **make_vars/2**.

[2, _, _, _, _, _, _, _, _, 2, _, _, _, _, 1, _, _,
_, _, _, _, _, 1, _, 2, _, _, _, _, _, _, 2, _, _, _]

Fig. 4. Lista da Figura 3 após substituição.

3.1 Variáveis de Decisão

Como se trata de um tabuleiro N por N, facilmente sabemos quantas linhas e colunas existem de forma a obter uma matriz com o predicado **divide/3**, que divide a lista inicial em sublistas de tamanho N, cada uma representando uma linha do *puzzle*.

[[2, _, _, _, _, _],
[_, _, _, _, _, _],
[_, _, _, 1, _, _],
[_, _, _, _, _, 1],
[_, 2, _, _, _, _],
[_, _, 2, _, _, _]]

Fig. 5. Matriz final.

Cada uma destas variáveis de decisão representa o valor de um espaço do tabuleiro por preencher e, usando a mesma lógica do predicado **replace_all/8**, o domínio daquelas, definido em **domain/3**, está limitado aos valores '0', '1' e '2', simbolizando espaços em branco, 'C' e 'F', respetivamente.

Por fim, torna-se importante guardar as variáveis 'C' e 'F' encontradas no predicado **check_promixity/3** (que é explicado detalhadamente na próxima secção), pelo que existem duas listas – NF e NNF – de tamanho $R1 = 4\text{N}$ e $R2 = 2R1$, respetivamente, para a sua recolha.

3.2 Restrições

Para resolver este problema são fundamentais duas restrições, aplicadas às linhas pelo predicado **row_constraints/5** e às colunas por **column_constraints/5**. Por sua vez, estes predicados, aplicam essas restrições a cada uma das linhas e colunas, obtidas com o auxílio dos predicados **nth1/3** e **get_column/3**, respetivamente.

1. Só podem existir dois C's e dois F's.
Para tal, é usado o predicado **global_cardinality/2**, da biblioteca CLP(FD) SICStus, que restringe a presença em cada lista recebida a dois '2' (F's), dois '1' (C's) e N '0' (espaços em branco), em que N é calculado através da diferença entre o tamanho da lista e quatro (2xF + 2xC).

2. Os F's têm de estar mais afastados entre si do que os C's.
Esta restrição é implementada pelo predicado **check_proximity/3**, que procura dois '2' (F1 e F2) e dois '1' (C1 e C2), unificados pelo predicado **element/3**, da biblioteca CLP(FD) SICStus, sendo necessário que F2 #> F1 e C2 #> C1 (para evitar repetições e simetrias) e o valor absoluto do resultado da diferença entre o valor F2 e F1 seja maior que o mesmo valor entre C2 e C1.

3.3 Estratégia de Pesquisa

A etiquetação das variáveis é realizada pelo predicado **labeling/2**, da biblioteca CLP(FD) SICStus. Esta recebe uma lista com as todas variáveis recolhidas que acrescentam conhecimento acerca da solução. Para a sua obtenção é utilizado o predicado **flatten_list/2** para que a matriz tabuleiro fique numa lista simples e a essa mesma lista são concatenados – com o **append/3 – os** C's e F's recolhidos.
Foi acrescentado o predicado **my_sel/4** às opções de *labeling*, que permite adicionar aleatoriedade à solução encontrada. De forma a avaliar o impacto dessa opção e da variação do tamanho do *puzzle* na etiquetação, são utilizados o predicado **statistics/2** e **fd_statistics/0**, ambos da biblioteca CLP(FD) SICStus, para a medição do tempo decorrido, o número de retrocessos efetuados e restrições aplicadas.

3.4 Geração de Problemas

Para além da resolução de *puzzles* dados – **find_CF(1, Board)** – procedemos à implementação de um gerador de problemas "Close or Far". Trata-se do **find_CF(2, Board)**, que começa por perguntar qual o tamanho N do *puzzle* a gerar no predicado **ask_size/1**. A implementação é exatamente a mesma do predicado de resolução, diferindo apenas na lista inicial, que neste caso começa vazia. Após gerar uma solução válida com o tamanho pedido, que, devido ao aumento de complexidade, pode demorar bastante para tabuleiros muito grandes, o predicado **unsolve/5** trata de remover um número aleatório de C's e F's de forma a gerar um *puzzle* incompleto, mas garantidamente solucionável.

4 Visualização da Solução

A solução implementada permite resolver o problema de decisão "Close or Far" e, para tal, existe uma interface associada para facilitar o processo. Para ser lançado é necessário instanciar o programa com o predicado '**closeOrFar.**', sendo apresentado um menu que permite escolher entre:

- Resolver, se possível, um *puzzle* dado.
- Gerar uma solução e respetivo *puzzle*, sendo preciso indicar o tamanho (N x N) que se pretende entre N = 6 e N = 20.

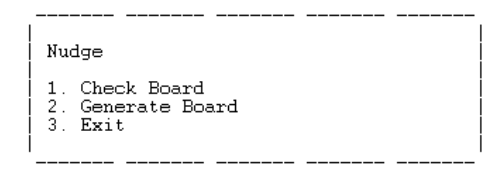

Fig. 6. Interface do programa.

O predicado **display_game/1**, junto com os predicados auxiliares **display_board/2**, **display_row/2**, **display_cell/1** e **display_border/1**, trata de apresentar a lista resultante da resolução ou geração do problema numa tabela de fácil visualização do *puzzle*. De salientar que estes predicados foram implementados de forma a poderem imprimir tabuleiros do formato da lista da Figura 3, de tamanho N por N, que recebam.

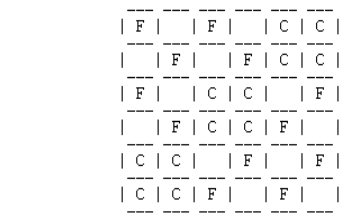

Fig. 7. Exemplo da visualização de um tabuleiro.

5 Resultados

De forma a avaliar os resultados obtidos, foram essencialmente medidos os tempos de resolução do problema, o número de restrições criadas e o número de retrocessos nas seguintes situações para *puzzles* fornecidos **25%** completos:

1. Variação do tamanho do *puzzle* a resolver (NxN).
Enquanto o número de restrições aumenta apenas linearmente com o aumento de tamanho, o tempo necessário para sua resolução e o número de retrocessos aumenta exponencialmente a partir de tabuleiros com dimensão superior a 10 por 10. Estes dois parâmetros estão diretamente relacionados como se pode ver no gráfico da Figura 8, baseado nos dados da Tabela 1 em Anexo.

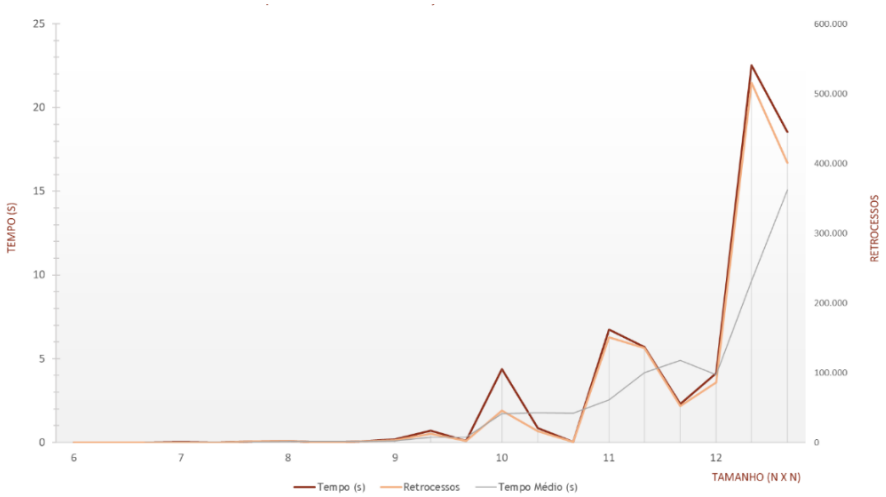

Fig. 8. Tempo decorrido e número de retrocessos em função do tamanho do tabuleiro.

2. Variação da estratégia de pesquisa.

Para perceber o efeito da estratégia de pesquisa na etiquetação, realizamos os mesmo testes do tópico anterior, mas sem utilizar o predicado **my_sel/4**, responsável por encontrar uma solução aleatória. Conclui-se assim que na maioria dos casos torna-se mais rápido encontrar uma solução, como se comprova pela Figura 9 com base nos dados da Tabela 2, em Anexo.

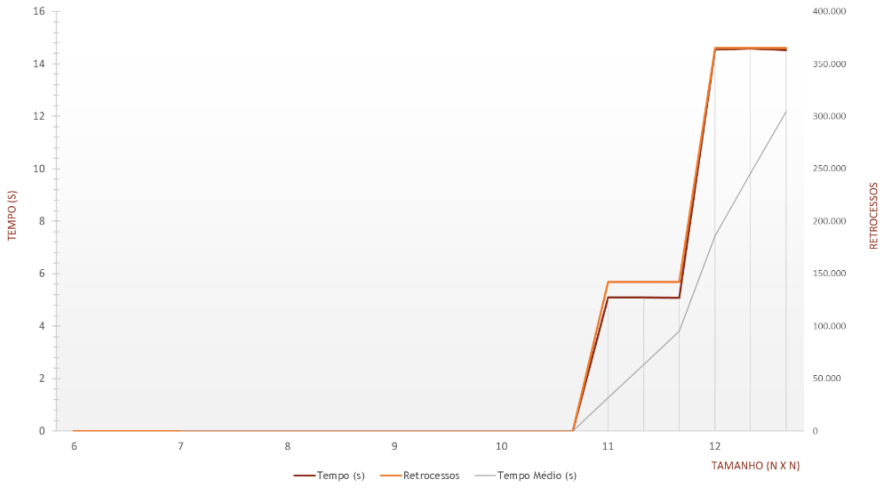

Fig. 9. Tempo decorrido e número de retrocessos em função do tamanho do tabuleiro (solução não aleatória).

6 Conclusões e Trabalho Futuro

Este trabalho tem como principal objetivo aplicar a matéria lecionada nas aulas práticas e teóricas da cadeira. Apesar das diversas dificuldades encontradas na implementação das restrições necessárias à resolução do problema, com o tempo estas foram ultrapassadas com a ajuda dos docentes da cadeira.

A solução que propomos poderia eventualmente ser otimizada, uma vez que, para *puzzles* de grandes dimensões, o tempo de resolução aumenta consideravelmente. Ainda assim, estamos satisfeitos com o algoritmo desenvolvido que se mostrou eficiente na maioria das situações, tendo assim cumprido os objetivos estabelecidos. Além disto, damos a possibilidade de gerar novas instâncias do problema.

Em suma, com este trabalho conseguimos não só aprimorar ainda mais o nosso conhecimento da linguagem Prolog, mas também a nossa adaptabilidade a novos conceitos.

7 Anexo

Tabela 1. Tempo decorrido e número de retrocessos em função do tamanho do tabuleiro.

Tamanho (N x N)	Restrições	Retrocessos	Tempo (s)	Tempo Médio (s)
6	156	44	0,00	0,0000
	156	45	0,00	
	156	55	0,00	
7	182	320	0,03	0,0367
	182	112	0,01	
	182	1.341	0,07	
8	206	1.983	0,10	0,0600
	206	155	0,01	
	206	1.509	0,07	
9	230	3.294	0,17	0,3200
	230	12.185	0,70	
	230	1.881	0,09	
10	256	45.659	4,37	1,7567
	256	16.388	0,86	
	256	704	0,04	
11	283	150.479	6,73	4,9100
	283	135.679	5,71	
	283	51.909	2,29	
12	308	86.213	4,15	15,0733
	308	515.336	22,53	
	308	400.822	18,54	

Tabela 2. Tempo decorrido e número de retrocessos em função do tamanho do tabuleiro (solução não aleatória).

Tamanho (N x N)	Restrições	Retrocessos	Tempo (s)	Tempo Médio (s)
6	156	24	0,00	0,0000
	156	24	0,00	
	156	24	0,00	
7	182	220	0,01	0,0067
	182	220	0,00	
	182	220	0,01	
8	206	52	0,01	0,0067
	206	52	0,01	
	206	52	0,00	
9	230	47	0,00	0,0000
	230	47	0,00	
	230	47	0,00	
10	256	23	0,00	0,0033
	256	23	0,00	
	256	23	0,01	
11	283	141.931	5,09	5,0833
	283	141.931	5,09	
	283	141.931	5,07	
12	308	365.248	14,54	14,5500
	308	365.248	14,59	
	308	365.248	14,52	

Bibliografia

1. Friedman, E.: https://stetson.edu/~efriedma/puzzle/closefar/, last accessed 2020/01/01.
2. LNCS Homepage, http://www.springer.com/lncs, last accessed 2020/01/01.