Puzzle 2D - Wrong Products

Segundo Projeto

Hugo Miguel Monteiro Guimarães
Beatriz Costa Silva Mendes

Trabalho realizado no âmbito da
Unidade Curricular de Programação Lógica

Mestrado Integrado em Engenharia Informática e Computação
Faculdade de Engenharia da Universidade do Porto
Porto
28 de dezembro de 2020

Contents

Resumo

Este trabalho foi desenvolvido no âmbito da Unidade Curricular de Programação em Lógica, através do sistema SICStus Prolog, e o seu objetivo foi resolver um problema de decisão utilizando Programação de lógica com restrição sobre domínios finitos. O problema escolhido foi ***Wrong Products*** e tem como objetivo colocar números numa grelha de modo a que cada linha e coluna contenha apenas 2 dígitos e que o produto entre os mesmos corresponda a um determinado valor no exterior da grelha, com variação de uma unidade.

1 Introdução

O Problema de Otimização escolhido ,*Wrong Products*, tem como objetivo colocar números numa grelha de modo a que cada linha e coluna contenha apenas 2 dígitos e que o produto entre os mesmos corresponda a um determinado valor no exterior da grelha, com variação de uma unidade.

Este problema tem como objetivo a implementação de uma grelha matricial com restrições sobre linhas, colunas e toda a malha utilizando programação de lógica com restrições sobre domínios finitos.

Este artigo possui a seguinte estrutura:

- **Descrição do Problema:** Descrição com detalhe do problema de otimização ou decisão em análise, incluindo todas as restrições envolvidas.
- **Abordagem:** Descrição da modelação do problema como um Problema de Satisfação de Restrições(PSR)
  - **Variáveis de Decisão:** Descrição das variáveis de decisão e respetivos domínios, assim como o seu significado no contexto do problema em análise.
  - **Restrições:** Descrição das restrições rígidas e flexíveis do problema e a sua implementação utilizando o SICStus Prolog.
  - **Visualização da Solução:** Explicação dos predicados que permitem visualizar a solução em modo de texto.

- **Experiências e Resultados:** Análise de problema e resultados obtidos
  - **Análise Dimensional:** Exemplificação da execução de instâncias do problema com diferentes dimensões e análise dos resultados obtidos.
  - **Estratégias de Pesquisa:** Descrição de diferentes estratégias de pesquisa, comparando os resultados.
- **Conclusões e Trabalho Futuro:** Conclusões obtidas pela realização do trabalho
- **Referências:** Referências bibliográficas utilizadas
- **Anexos:** Anexos de resultados úteis para a resolução do problema.

2 Descrição do Problema

Wrong Products é um problema de decisão. Este problema pretende descobrir se é possível colocar números numa grelha de modo a que cada que cada número apareça uma única vez, e que cada linha e coluna contenha unicamente dois números, e que o seu produto corresponda a uma unidade acima ou abaixo de um determinado valor no exterior da grelha associado à respetiva linha ou coluna.

3 Abordagem

Wrong Products é um Problema de Satisfação de Restrições(PSR) e é modelado por:

3.1 Variáveis de Decisão

Na resolução deste problema, é necessário criar as seguintes variáveis:

Length - Tamanho da grelha, indicando o número de linhas e colunas. Estes valores são iguais dado que a grelha corresponde a uma matriz quadrada. O valor de *length* é passado como argumento do predicado, pelo que o seu domínio varia conforme o pretendido pelo utilizador.

RowValues - Lista com os valores iniciais cuja diferença entre o produto dos 2 valores da respetiva linha seja 1.

ColValues - Lista com os valores iniciais cuja diferença entre o produto dos 2 valores da respetiva coluna seja 1.

FinalRowValues - Lista com os valores finais cuja diferença entre o produto dos 2 valores da respetiva linha seja 1.

FinalColValues - Lista com os valores finais cuja diferença entre o produto dos 2 valores da respetiva coluna seja 1.

RowValues, *ColValues*, *FinalRowValues* e *FinalColValues* possuem o mesmo domínio, todos os valores inteiros positivos que não sejam superiores ao produto entre os 2 maiores valores permitidos, que corresponde a:

$$(Length * 2) * (Length * 2 - 1)$$

ListOfLists - Lista de Listas contendo a grelha do problema. Pode ser interpretada como uma representação matricial do problema. Contém valores inteiros a serem multiplicados em cada linha e coluna, e zeros que correspondem a casas vazias.

Transpose - Lista de Listas contendo a matriz transposta da grelha do problema. Tem como propósito facilitar a aplicação de restrições às colunas da grelha do problema.

Table - Forma achatada de ListOfLists. tem como objetivo permitir a aplicação de restrições sobre toda a grelha.

Dado que apenas podem existir 2 números por linha e coluna, o domínio de ListOfLists, Transpose e Table é o mesmo e corresponde a todos os valores inteiros entre 0 e $2 * Length$.

3.2 Restrições

all_distinct_except_0(List) Permite aplicar uma restrição a todos os valores da grelha, de modo a que, tal como o nome indica, todos os valores sejam distintos exceto os zeros. Desta forma é possível garantir que não há valores semelhantes e, simultaneamente, utilizar o valor zero como representação de uma casa não ocupada.

line_restriction(List,Amount) Recebe como argumentos uma lista de listas e um valor inteiro *amount*, e restringe a quantidade de zeros em cada linha de uma Lista de Listas, consequentemente garantindo que existem apenas 2 valores em cada linha. Este predicado é evocado duas vezes, a primeira para *ListOfList*, e a segunda para *Transpose*, de modo a que a restrição seja aplicada quer a linhas quer a colunas.

multiplication_restriction(ListOfLists,List) Recebe uma Lista de Listas e verifica se o produto dos valores diferentes de zero de cada linha difere uma unidade em relação ao respetivo valor de uma Lista. Este predicado é evocado duas vezes, a primeira para *ListOfList* e *RowValues*, e a segunda para *Transpose* e *Colvalues*, garantindo que tanto o produto de uma linha como o de uma coluna cumprem a restrição da multiplicação enunciada anteriormente.

4 Visualização da Solução

Para visualizar a solução do solução do nosso problema, é necessário primeiro seguir os seguintes passos:

- Iniciar o SICStus Prolog.
- Dar **consult** ao ficheiro **wrongproducts.pl**.
- Selecionar a fonte através de: $Settings \rightarrow Font \rightarrow Consolas \rightarrow ok$ (Este passo é necessário para que a formatação dos resultados esteja correta).
- Executar o predicado **main** para executar o programa.

Ao executar o nosso programa (se quiser ver um dos problemas *default* ou resolver um problema gerado aleatoriamente), primeiro é mostrado um problema por resolver, ou seja, com os espaços dentro da matriz em branco. Este problema por resolver é obtido através do predicado **display**, que além de demonstrar a solução, faz também um *refresh* ao ecrã. Posteriormente, quando o utilizador pede a solução do problema, esta é apresentada graças ao predicado **displayWithoutClean**, que faz o mesmo que o predicado **display**, no entanto não faz um *refresh* ao ecrã.

Ambos os predicados mencionados anteriormente imprimem os valores das colunas inicialmente e imprimem recursivamente os valores da tabela juntamente com os valores por cada linha.

5 Experiências e Resultados

5.1 Análise Dimensional

Incluir exemplos de execução em instâncias do problema com diferentes dimensões e analisar os resultados obtidos.

Embora Wrong Products seja um problema apresentado sobre a forma de uma matriz quadrada de dimensão 4, resolvemos este problema de decisão dinamicamente, sendo possível fazer variar a dimensão da grelha e obter as soluções existentes, se possível.

O predicado ***generatePuzzle(Length,RowValues,ColValues)*** recebe como argumento o parâmetro *Length*, que corresponde à dimensão da matriz, gerando parâmetros aleatórios de *RowValues* e *ColValues* que garantem a existência de uma solução válida para o problema, sendo então possível gerar um problema resolúvel de qualquer dimensão.

O predicado ***solver(Length,RowValues,ColValues)*** recebe como argumento as variáveis de decisão obtidas pelo predicado *generatePuzzle*, obtendo uma solução possível para o problema. No fim, é chamado o predicado *displayWithoutClean* que permite visualizar a solução do problema conforme explicado na secção anterior. Tal como o *generatePuzzle*, possui como argumento a variável *Length*, que indica a dimensão da matriz.

Exemplos de execução de um problema com diferentes dimensões encontram-se na secção de anexos

5.2 Estratégias de Pesquisa

Após encontrarmos uma forma de resolver o problema, decidimos analisar diferentes estratégias de pesquisa, de modo a obter uma solução da maneira mais eficiente.

Para tal ser possível, utilizamos os predicados *reset_time* e *print_time(Msg)* disponibilizados no *Moodle* para obter tempos de execução do *solver* do problema.

Deste modo, decidimos variar as opções de escolha de variável e valor do *labeling* do *solver* do nosso problema, analisando o tempo de execução da resolução de problemas de diferentes dimensões. Concluímos então que a opção que gerava os tempos de execução da forma mais eficiente eram os valores default de *labeling*:

$$labeling([leftmost, step, down, satisfy], Table)$$

As restantes combinações de valores testadas acabam por ser sempre menos eficientes que a *default*, sendo a discrepância notável quando maior a dimensão do problema gerado.

O valor de $labeling([max, step, down, satisfy], Table)$ surpreendeu-nos pela negativa, dado que não era possível resolver o problema com a dimensão 30, enquanto as outras combinações conseguiam calcular o resultado em frações de tempo inferiores a 1 segundo, o que o que nos permitiu entender a importância da busca por uma estratégia de pesquisa eficiente.

De seguida encontram-se 2 gráficos comparando tempos de execução de diferentes opções de pesquisa, fazendo variar a dimensão do problema.

Dimensao	Média default	Média usando down
10	0.008	0.01
20	0.054	0.08
30	0.362	0.554
40	1.224	2.02
50	4.628	6.602
60	9.316	18.82
70	20.256	35.096
80	41.1	93.042

Figure 1: Tabela com Média de Tempos de Execução, em segundos

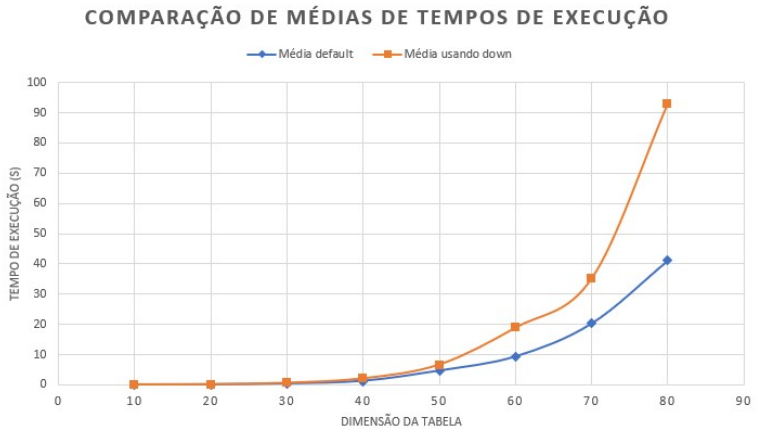

Figure 2: Grafico com Média de Tempos de Execução

6 Conclusões e Trabalho Futuro

Este trabalho permitiu-nos , por um lado, melhorar os nossos conhecimentos de Programação de lógica com restrições sobre domínios finitos, nomeadamente a sua utilização para resolução de Problema de Satisfação de Restrições. Por outro lado, também nos permitiu verificar na prática a influência de diferentes estratégias de pesquisa na resolução de um problema.

Os resultados obtidos demonstram que o problema proposto é facilmente resolúvel computacionalmente em tempos reduzidos, inferiores à décima do segundo para tabuleiros de dimensão inferior a 10, pelo que podemos concluir que tivemos sucesso no desenvolvimento da estratégia da solução do problema escolhido. Como seria de esperar, a complexidade temporal da solução proposta aumenta exponencialmente em função da dimensão do tabuleiro, pelo que o nosso programa tem dificuldade em encontrar soluções em tempo útil para malhas de dimensão superior a 60.

7 Referências

We Are Puzzlers Club: Part 2 Instructions Booklet. (2019, August).

Programação em Lógica com Restrições. (2020, November).

Programação em Lógica com Restrições no SICStus Prolog. (2020, November).

8 Anexos

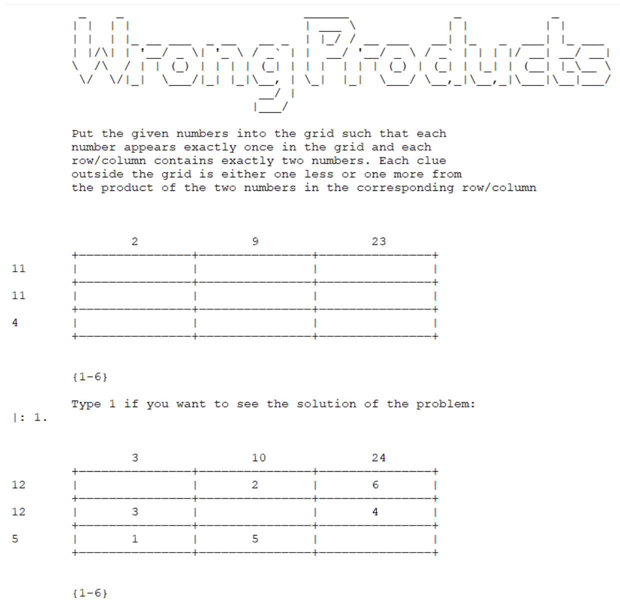

Figure 3: Execução do programa com dimensão 3

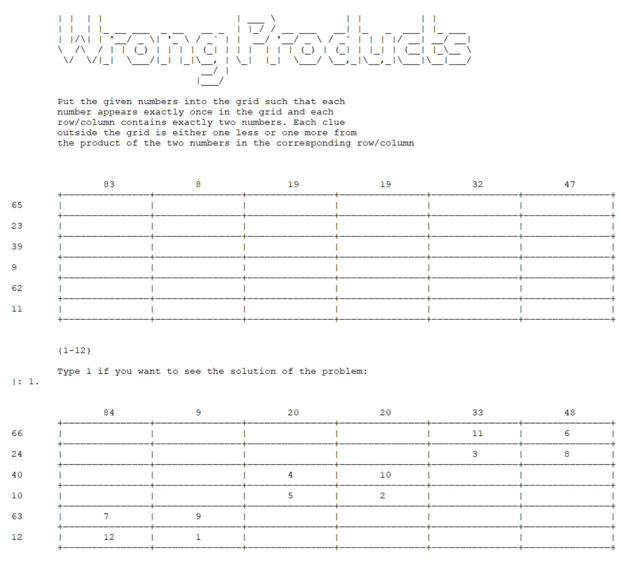

Figure 4: Execução do programa com dimensão 6