Geração e Resolução de puzzles do tipo *Grape Puzzles*

Gonçalo Alves[up201806451] e António Bezerra[up201806854]

FEUP-PLOG, Turma 3MIEIC03, Grupo Grape_3
https://web.fe.up.pt

Resumo Começa-se por definir o objetivo deste trabalho e pela apresentação de informações relevantes sobre cada ponto do trabalho. De seguida, é descrito o problema *Grape Puzzles* em detalhe. Na abordagem são descritas: as **variáveis de decisão** e as **restrições** aplicadas. Na secção de visualização da solução são explicados os predicados que permitem diferentes visualizações. De seguida, são apresentadas as experiências com os respetivos resultados. Finalmente, são apresentadas as conclusões e o possível trabalho futuro.

Keywords: Grape · Linhas · Restrição.

1 Introdução

Este artigo descreve o segundo trabalho realizado para a unidade curricular de Programação em Lógica, tendo como objetivo a resolução e geração de problemas do tipo *Grape Puzzles*, através de programação em lógica com restrições.

Inicialmente, o objetivo deste trabalho foi desenvolver um predicado que permitisse resolver os problemas apresentados no enunciado disponibilizado [1]. Após esta fase inicial, o próximo passo foi o melhoramento do nosso solucionador de modo a permitir a geração de problemas.

O artigo começa por descrever o problema em questão, seguido da abordagem tomada para o resolver. Após isto, são apresentadas as formas de visualização dos problemas e das soluções e também são apresentadas as experiências e os resultados. Finalmente, são discutidas as conclusões do trabalho.

2 Descrição do Problema

O problema de decisão *Grape Puzzle*, tal como descrito no site [1], tem como objetivo, colocar um número positivo em cada "uva". Cada número na primeira linha tem apenas um dígito e nas restantes linhas, os números são a soma das duas "uvas"da linha imediatamente acima. Além disso, "uvas"com o mesmo número têm a mesma cor. Assim, podemos interpretar este problema como a resolução de *n-1+...+1* somas, em que cores iguais representam variáveis iguais. Tomemos o exemplo, do problema seguinte, com 4 linhas:

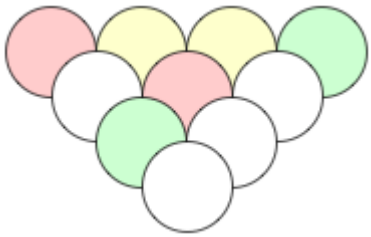

Figura 1. Problema de 4 Linhas

As somas correspondentes, apresentadas de linha a linha, serão:

$$\begin{cases} A + B = D \bigcap B + B = A \bigcap B + C = E \\ D + A = C \bigcap A + E = F \\ C + F = G \end{cases} \quad (1)$$

Tendo como solução:

Figura 2. Solução de um problema de 4 Linhas

3 Abordagem

3.1 Variáveis de Decisão

Tal como descrito na secção anterior, as variáveis de decisão serão os números de cada "uva". De modo, a representar um *puzzle* como o da Fig.1, optou-se por uma lista de listas, em que cada lista representa uma linha do problema.

O predicado `defineDomains/1` é responsável por definir os domínios para cada linha. Inicialmente, atribui à primeira linha o domínio `[1,9]` e depois vai atribuindo às outras listas o domínio `[2,MaxValue]`. A variável `MaxValue` é calculada consoante o número de linhas, através do predicado `defineUpperBound/2`.

3.2 Restrições

As restrições deste problema são as seguintes:

- Primeira linha apenas pode conter números positivos de um dígito
- A "uva"que se encontra debaixo de duas "uvas"é a soma destas
- "Uvas"com a mesma cor, contêm o mesmo número, com excepção da cor branca
- Há um número máximo de cores

As três primeiras restrições do problema são restrições rígidas. A última restrição foi assumida pelo grupo e isto deve-se ao facto de não estar presente no enunciado qualquer tipo de indicação sobre o número de cores obrigatórias que o problema deve ter. Assim, criou-se esta restrição flexível para poder gerar problemas inferiores a 4 linhas (problemas de 2 linhas não conseguiriam ter 3 cores) e problemas superiores a 6 linhas (problemas maiores vão necessitar de mais cores para não se tornarem demasiado difíceis para quem os está a resolver).

Assim, para a primeira restrição fez-se uso do predicado `domain/3` da biblioteca **clpfd** do SICStus para limitar o domínio, como referido na subsecção anterior. Para a restrição da soma, desenvolveu-se o predicado `defineSumConstraints/1`, que percorre as listas e coloca a restrição `FirstUpper + SecondUpper #= Child`, para cada linha após a primeira. Por fim, a restrição de cor é feita com recurso ao predicado `global_cardinality/2`. Numa primeira fase, este predicado permite contar o número de ocorrências de números numa lista e com uma segunda chamada deste predicado, é possível restringir a ocorrência de pares de números ao número de cores necessárias.

4 Visualização da Solução

De modo a visualizar o "cacho de uvas"e tornar a experiência do utilizador mais apelativa, foram criados dois predicados `displayOutput/2` e `displayOutput/3`, que apresentam a solução e o puzzle seguido da solução, respetivamente.

A primeira versão do predicado apenas apresenta a solução do problema e é utilizada no predicado `grapesolver/1`.

A segunda versão apresenta tanto o problema, com as cores substituidas por letras, como a solução respetiva. Esta versão é utilizada no predicado `grapegenerator/2`.

```
 Puzzle:
(A)(C)(C)(B)
(B)(A)( )
( )( )
( )

 Solution:
(2)(1)(1)(3)
(3)(2)(4)
(5)(6)
(11)
```

Figura 3. Impressão de Resultados

5 Experiências e Resultados

O programa foi testado de modo a ser possível estudar o comportamento da nossa resolução não só com puzzles de tamanho diferente, mas também com estratégias de pesquisa diferentes. É de salientar que os resultados apresentados poderão diferir de máquina para máquina, devido aos diferentes tempos de processamento.

5.1 Análise Dimensional

O programa foi testado com problemas de 2 a 7 linhas.

Número de Linhas	Tempo de Execução (s)	Número de Soluções
2	0	9
3	0.015	80
4	0.125	240
5	1.438	9854
6	13.906	96134
7	128.641	212350

Tabela 1. Resultados de uma Análise Dimensional

Como podemos observar pela tabela acima e pelos gráficos respetivos [4,5], o crescimento do tempo e das soluções encontradas foi exponencial. Por esta razão, não se testaram problemas de tamanho superior a 7 linhas.

Devido à falta de tempo, não foi possível desenvolver uma restrição que nos permitisse remover puzzles "espelhados", ver Fig 7 e 8.

5.2 Estratégias de Pesquisa

O programa foi testado com todas as combinações de heurísticas, para problemas de 5 linhas.

Através da observação da tabela seguinte e do respetivo gráfico [6] , concluímos que a combinação mais eficiente seria **[ffc,enum,up]**.

Ordenação de Variáveis	Seleção de Valores	Ordenação de Valores	Tempo de Execução (s)
leftmost	step	up	2.422
		down	2.563
	enum	up	1.531
		down	1.531
	bisect	up	1.875
		down	2.047
	median	up	2.469
		down	2.547
	middle	up	2.422
		down	2.578
min	step	up	2.656
		down	2.594
	enum	up	1.547
		down	1.578
	bisect	up	2.453
		down	2.531
	median	up	2.656
		down	2.594
	middle	up	2.656
		down	2.594
max	step	up	3.156
		down	3.782
	enum	up	1.875
		down	1.89
	bisect	up	3.203
		down	3.172
	median	up	3.266
		down	3.828
	middle	up	3.219
		down	3.781
ff	step	up	2.438
		down	2.562
	enum	up	1.516
		down	1.547
	bisect	up	1.89
		down	2.016
	median	up	2.453
		down	2.578
	middle	up	2.453
		down	2.578
anti_first_fail	step	up	12.672
		down	11.078
	enum	up	5.157
		down	5.187
	bisect	up	6.125
		down	6.297
	median	up	12.781
		down	11.156
	middle	up	12.735
		down	11.203
occurrence	step	up	2.406
		down	2.485
	enum	up	1.485
		down	1.5
	bisect	up	1.828
		down	1.922
	median	up	2.343
		down	2.438
	middle	up	2.344
		down	2.453
ffc	step	up	2.359
		down	2.453
	enum	up	1.469
		down	1.516
	bisect	up	1.796
		down	1.954
	median	up	2.375
		down	2.515
	middle	up	2.344
		down	2.484
$\max_r egret$	step	up	2.5
		down	2.672
	enum	up	1.578
		down	1.61
	bisect	up	1.922
		down	2.031
	median	up	2.484
		down	2.703
	middle	up	2.469
		down	2.656

Tabela 2. Resultados de diferentes Estratégias de Pesquisa

6 Conclusão e Trabalho Futuro

Este trabalho permitiu-nos aprender uma nova metodologia de programar, muito diferente e mais eficaz do que a abordagem tradicional de *Generate&Test*. Além disso, ficámos cientes da utilidade de programação em lógica com restrições, no mundo real.

Quanto a trabalho futuro, gostaríamos de melhorar a restrição flexível do número de cores e para além disso, realizar mais testes. Devido à falta de tempo, não conseguimos pensar numa relação entre o tamanho do problema e o número de cores e não conseguimos desenvolver a restrição de para análise dimensional referida na secção respetiva.

Referências

1. *Grape Puzzles*, https://erich-friedman.github.io/puzzle/grapes/. Last accessed 3 Jan 2021

A Gráficos

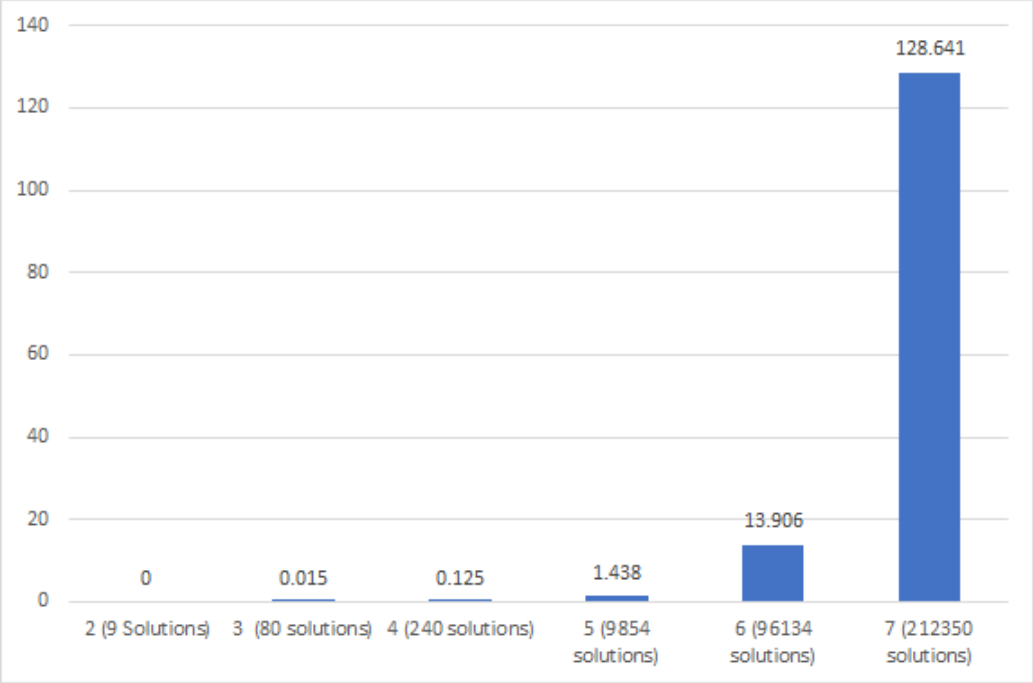

Figura 4. Tempos de execução por Número de linhas do problema

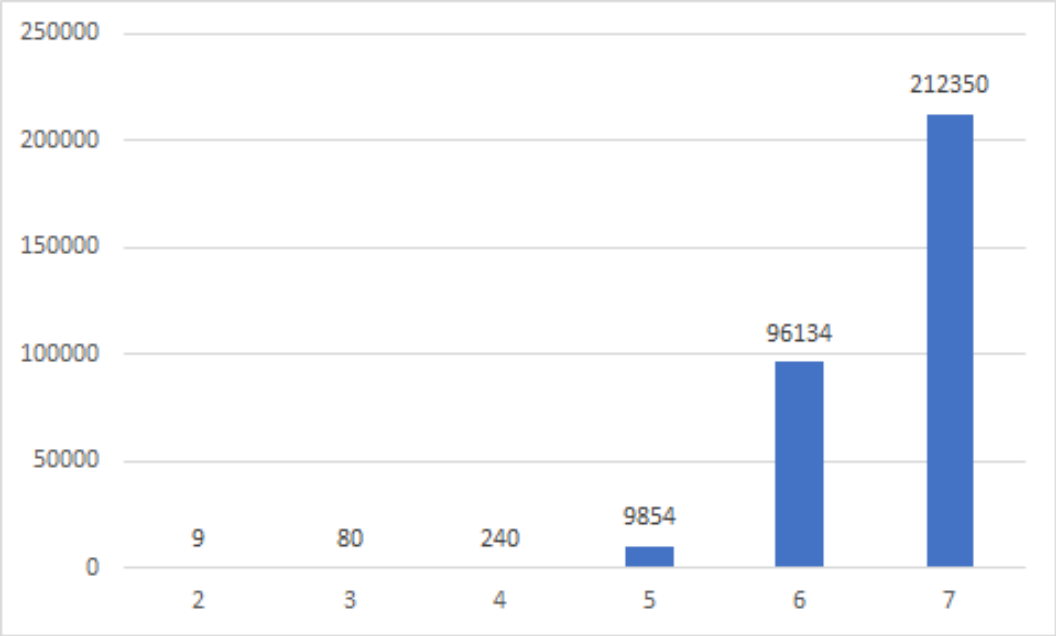

Figura 5. Número de soluções por Número de linhas do problema

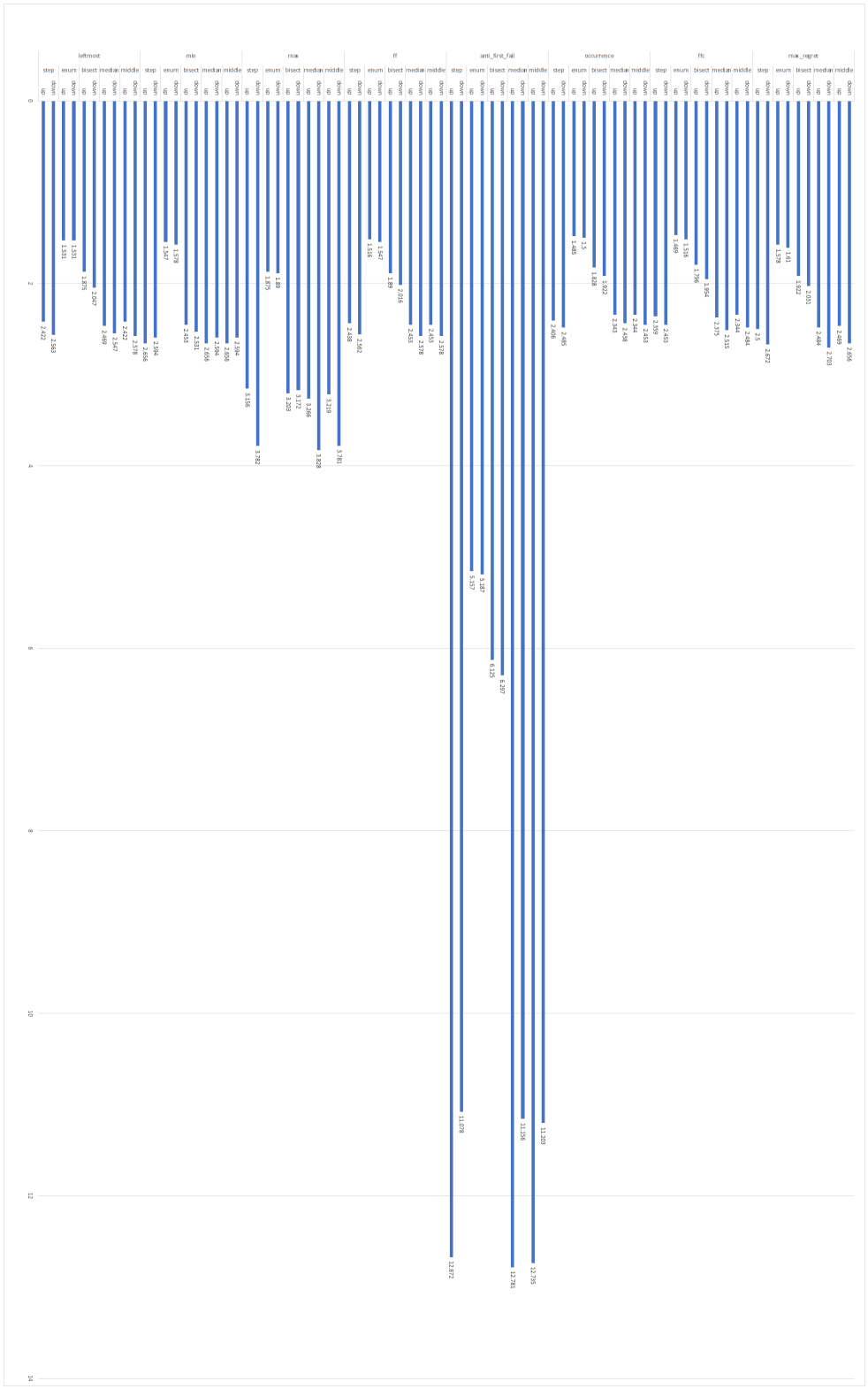

Figura 6. Análise temporal de estratégias de pesquisa, para um problema de 5 linhas

B Imagens

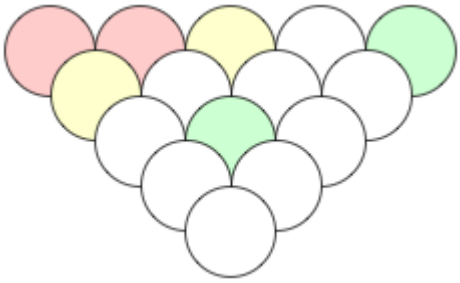

Figura 7. Problema original

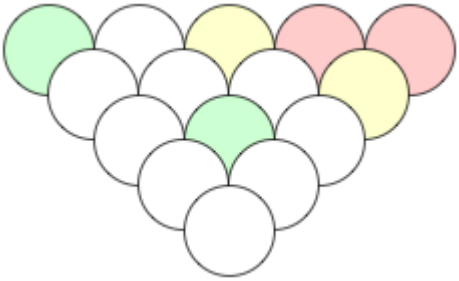

Figura 8. Problema invertido