Resolução de Problemas de Decisão usando Programação em Lógica com Restrições: *Crypto-Product*

Ana Teresa Cruz[up201806460] e André Nascimento[up201806461]

FEUP-PLOG, Turma 3MIEIC06, Grupo Crypto_Product_2
Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias, 4200-465, Porto, Portugal
http://web.fe.up.pt

Resumo: O projeto foi desenvolvido no âmbito da unidade curricular de Programação em Lógica utilizando o SICStus como Sistema de Desenvolvimento. O objetivo deste é resolver o problema escolhido, *Crypto-Product*, que tem como objetivo resolver um puzzle de forma a que a multiplicação seja correta. Numa abordagem inicial foram escolhidas as variáveis de decisão e as restrições. Na visualização da solução são mencionados os predicados que permitem diferentes visualizações. Após, são explicados as experiências feitas e os respetivos resultados. Por fim, as conclusões e possível trabalho futuro.

Keywords: Multiplicação, Restrições, Programação em Lógica

1 Introdução

O projeto foi desenvolvido no âmbito da unidade curricular de Programação em Lógica do 3ºano do curso Mestrado Integrado em Engenharia Informática de Computação. O objetivo deste é resolver e gerar *puzzles Crypto-Product* [1] através de restrições.

Numa abordagem inicial, o objetivo foi desenvolver um predicado que resolve puzzles como os do enunciado [1]. De seguida, um predicado que gerasse problemas e as respetivas soluções. Posteriormente foram criados predicados que permitem a visualização dos puzzles e das soluções.

No relatório começa-se por descrever o problema e, de seguida, a abordagem seguida para o resolver incluindo as variáveis de decisão, restrições, funções de avaliação e estratégias de pesquisa. Por fim, é mencionada a visualização da solução, seguida pelos resultados e conclusões.

2 Descrição do Problema

O problema de decisão *Crypto-Product* [1] consiste em resolver a multiplicação na forma **A x B = C**, sendo A, B e C números cujos dígitos são representados por cores.

Os números podem ser número de dígitos variável, mas dígitos iguais são representados pela mesma cor.

Segue-se um exemplo de um puzzle e respetiva solução. Este puzzle é o produto entre um número, com dois dígitos diferentes, por outro número, com dois dígitos iguais ao segundo digito do primeiro número, resultando num número com três dígitos, sendo o segundo igual ao primeiro digito do primeiro número e o primeiro e último dígitos iguais entre si mas diferentes de todos os outros.

Figura 1: Exemplo de um puzzle

Figura 2: Solução do puzzle

Para que a solução seja válida, cada dígito tem de ser um número inteiro de 0 até 9, exceto o dígito mais à esquerda de cada número que não pode ser 0.

3 Abordagem

Este puzzle corresponde a um problema de satisfação de restrições (**PSR**). De seguida são explicadas as variáveis de decisão e restrições usadas.

3.1 Variáveis de Decisão

Para determinar a solução ao problema foi implementado o predicado *cp_solver(+L_digits_list,+R_digits_list,+Res_digits_list,-L_number,-R_number, -Res_number)* que recebe três listas, uma para o operando da esquerda, outra para o da direita e outra para o produto. A partir de cada uma das listas constituídas por variáveis que representam cada dígito, este predicado atribui a cada variável um dígito de forma a que a multiplicação seja correta, cada número é retornado nos três argumentos seguintes.

Assim sendo, o problema possui então um número de variáveis de decisão variável, correspondendo ao número de dígitos diferentes na multiplicação. O domínio das variáveis é [0,9], sendo o domínio da variável correspondente ao dígito mais à esquerda do número [1,9]. Os domínios são atribuídos dentro no predicado *cp_solver/6*.

3.2 Restrições

O problema contém apenas restrições rígidas, uma vez que têm obrigatoriamente de ser cumpridas.

Os dígitos têm de ser sempre inteiros de 0 a 9, e o dígito mais à esquerda do número não pode ser 0. Uma vez que o 0 é o elemento absorvente da multiplicação, caso o número tenha só um dígito e este seja 0, o produto será 0. Caso o número tenha mais que um dígito, mas o mais à esquerda for 0, então o tamanho real do número já não vai ser o pensado. O domínio das variáveis é garantido com o recurso ao predicado *domain/3* e, o dígito mais à esquerda não poder ser 0 com recurso ao *restrictAboveZero/3*. Através desta restrição, reduz-se as multiplicações que o *labeling* vai ter de fazer. Estas restrições são inerentes ao problema.

As cores para os dígitos são passadas como variáveis para o *cp_solver/6*. De maneira a que existam tantas variáveis distintas como o número de cores diferentes que lhe é passado, todos os dígitos dos três números são colocados numa só lista, recorrendo ao *append/3*, são removidos os duplicados através do *remove_dups/2*, e por fim, é garantido que todas as variáveis são distintas através do *all_distinct/1*. Para chegar à solução do puzzle da figura 1 seria *cp_solver([R, B],[B, B],[G, R, G], Left, Right, Result).*

O número de cores diferentes gerado pelo *cp_generator/6* vai desde 2 até ao número máximo de dígitos que o produto pode ter (por exemplo, um puzzle de dois dígitos vezes três dígito tem entre 2 a 5 cores). Tal é implementado recorrendo ao predicado *nvalue/2*.

Quando um dos operandos é uma potência de 10, o outro é restringido pelo predicado *restrictPower10/1*. Após a análise de alguns resultados verificou-se que multiplicações com potências de 10 geravam muitos *puzzles* repetidos, mas foi também

possível observar um padrão causador desta repetição. Assim, este predicado restringe de forma a que o operador multiplicado pela potência de 10 siga as seguintes regras: o dígito mais à esquerda será 1 ou 2 e os seguintes dígitos têm como limite máximo o dígito anterior, ou seja, o dígito à sua esquerda, somado de uma unidade.

4 Visualização da Solução

Para a visualização da solução foram criados dois predicados.

O predicado *printSolution/3* que escreve na consola no formato **Número x Número = Número** o produto pedido. Segue-se um exemplo de utilização do predicado.

```
printSolution(31,14,434).

Solution: 31 x 14 = 434
```

Figura 3: printSolution

O predicado *printPuzzle/3* vai pôr numa lista todos os dígitos, atribuir a cada dígito uma cor e imprimir na consola o produto, mas em vez de dígitos, a cor que lhe corresponde. Segue-se um exemplo da utilização do predicado.

```
printPuzzle([3,1], [1,4], [4,3,4]).

Puzzle: GR x RB = BGB
```

Figura 4: printPuzzle

5 Experiências e Resultados

De seguida apresentam-se as experiências efetuadas nas quais estão a ser gerados todos os problemas para uma dada configuração e todas as respetivas soluções, anotando os tempos de execução de cada experiência para, deste modo, testar o nosso programa em *puzzles* com tamanhos diferentes e estratégias de pesquisa diferentes. É importante mencionar que os tempos de execução foram tirados na mesma máquina e que estes podem mudar de máquina para máquina.

5.1 Análise Dimensional

O programa foi testado com *puzzles* de diferentes dimensões tendo sido usada a combinação heurística que se observou gerar menores tempos de execução nas estratégias de pesquisa, ou seja, a ***min-bisect-up*** (tópico abordado na secção seguinte).

Com os resultados demonstrados na tabela 1 e no gráfico 1 pode-se concluir que quanto menor forem os números de dígitos de cada operando mais rápido é gerar e solucionar *puzzles*.

5.2 Estratégias de Pesquisa

Para *puzzles* de dois dígitos a multiplicar por dois dígitos foram testadas todas as combinações heurísticas, estando os respetivos resultados na forma de gráfico no Anexo I, gráfico 2, e em tabela no Anexo II, tabelas 2 e 3.

Através dos resultados é possível concluir que a melhor combinação heurística foi ***min-bisect-up***, com a qual se alcançou o menor tempo de execução: 0.969 segundos.

Foi também testado para um *puzzle* de dois dígitos a multiplicar por três dígitos com a combinação heurística ***anti_first_fail-step-up***, cujo tempo de execução foi cerca de 1 hora e 13 minutos e, por isso, não foram feitos mais testes para essa dimensão nem superior.

6 Conclusões e Trabalho Futuro

O projeto serviu para praticar o conhecimento adquirido nas aulas e, conclui-se que a linguagem Prolog é bastante útil para a resolução de problemas do quotidiano, principalmente através do uso de restrições.

Durante o desenvolvimento deste foram surgindo algumas dificuldades que foram superadas pela análise da biblioteca *clpfd* e dos slides fornecidos.

É de mencionar que alguns aspetos podiam ser melhorados, por exemplo, na geração de *puzzles* ocorrem algumas repetições e existem simetrias. Ambas podiam ser evitadas com mais restrições além das já aplicadas. Tal fica para trabalho futuro.

Referências

1. *Crypto-Product Puzzles*, https://erich-friedman.github.io/puzzle/crypto/.
2. *Constraint Logic Programming over Finite Domains, SICStus, https://sicstus.sics.se/sicstus/docs/4.1.0/html/sicstus/lib_002dclpfd.html*.

Anexo I – Gráficos

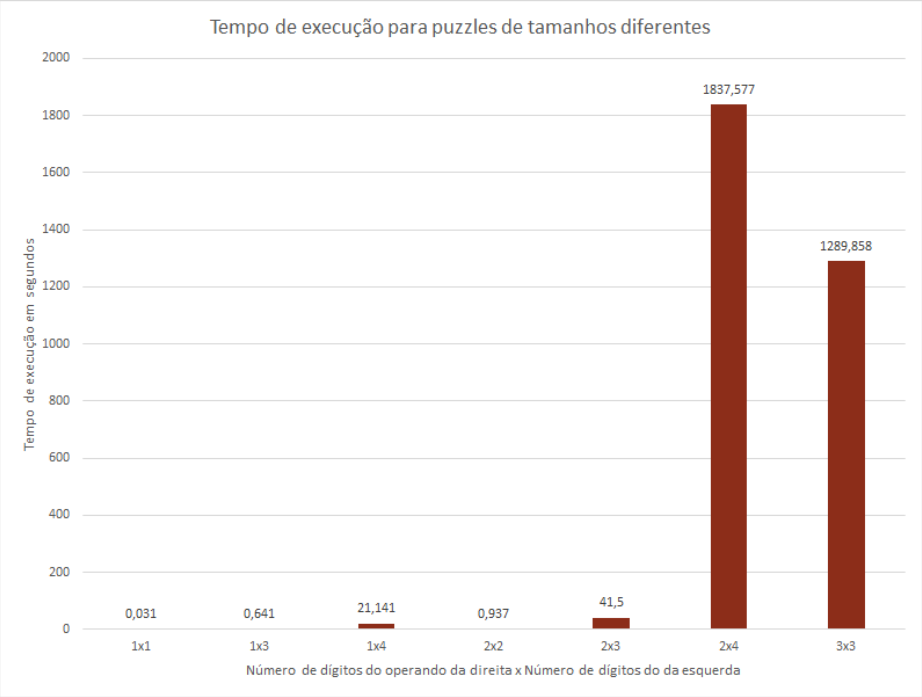

Gráfico 1: Análise Dimensional

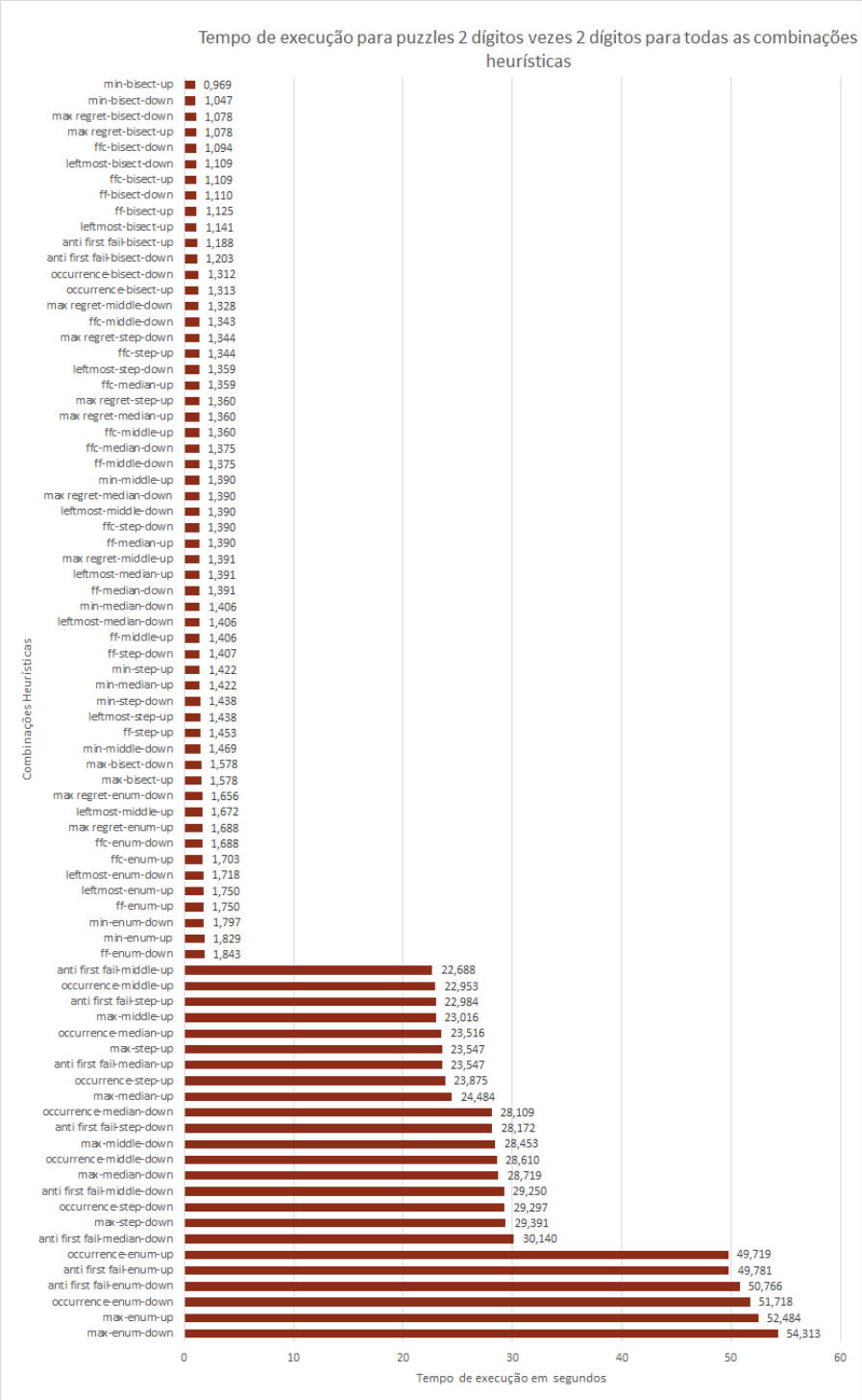

Gráfico 2: Estratégias de Pesquisa

Anexo II – Tabelas

Número de dígitos do operando da direita	Número de dígitos do operando da esquerda	Tempo de execução (s)
1	2	0,031
1	3	0,641
1	4	21,141
2	2	0,937
2	3	41,5
2	4	1837,577
3	3	1289,858

Tabela 1: Análise Dimensional

Ordenação das variáveis	Seleção de Valores	Ordenação de Valores	Tempo de execução (s)
anti first fail	bisect	up	1,188
		down	1,203
	enum	up	49,781
		down	50,766
	median	up	23,547
		down	30,140
	middle	up	22,688
		down	29,250
	step	up	22,984
		down	28,172
ff	bisect	up	1,125
		down	1,110
	enum	up	1,750
		down	1,843
	median	up	1,390
		down	1,391
	middle	up	1,406
		down	1,375
	step	up	1,453
		down	1,407
ffc	bisect	up	1,109
		down	1,094
	enum	up	1,703
		down	1,688
	median	up	1,359
		down	1,375
	middle	up	1,360
		down	1,343
	step	up	1,344
		down	1,390
leftmost	bisect	up	1,141
		down	1,109
	enum	up	1,750
		down	1,718
	median	up	1,391
		down	1,406
	middle	up	1,672
		down	1,390
	step	up	1,438
		down	1,359

Tabela 2: Estratégias de Pesquisa (1)

Ordenação das variáveis	Seleção de Valores	Ordenação de Valores	Tempo de execução (s)
max	bisect	up	1,578
		down	1,578
	enum	up	52,484
		down	54,313
	median	up	24,484
		down	28,719
	middle	up	23,016
		down	28,453
	step	up	23,547
		down	29,391
max regret	bisect	up	1,078
		down	1,078
	enum	up	1,688
		down	1,656
	median	up	1,360
		down	1,390
	middle	up	1,391
		down	1,328
	step	up	1,360
		down	1,344
min	bisect	up	0,969
		down	1,047
	enum	up	1,829
		down	1,797
	median	up	1,422
		down	1,406
	middle	up	1,390
		down	1,469
	step	up	1,422
		down	1,438
occurrence	bisect	up	1,313
		down	1,312
	enum	up	49,719
		down	51,718
	median	up	23,516
		down	28,109
	middle	up	22,953
		down	28,610
	step	up	23,875
		down	29,297

Tabela 3: Estratégias de Pesquisa (2)