Resolução de Problema de Decisão usando Programação em Lógica com Restrições: Gold Star

Daniel Garcia Silva[up201806524] e Mariana Truta[up201806543]

FEUP-PLOG, Turma 3MIEIC03, Grupo Gold_Star_1

Resumo. Este artigo consiste na descrição do problema de decisão Gold Star e na apresentação da abordagem utilizada para a resolução do mesmo. Usando programação em lógica com restrições, foi possível desenvolver um programa que gera aleatoriamente um puzzle Gold Star com uma dada dimensão e o resolve, mostrando, no fim, o puzzle e a solução encontrada na consola. O relatório contém uma secção onde são referidas várias experiências realizadas no programa desenvolvido com o objetivo de estudar o impacto do tamanho do puzzle e a combinação de argumentos de *labeling* no tempo de geração de um problema. Por fim, termina-se com uma breve conclusão onde se apresentam as principais aprendizagens e algumas melhorias que podiam ter sido feitas ao programa desenvolvido.

Keywords: Gold Star, SICStus, Prolog, Restrições, Decisão.

1 Introdução

No âmbito da unidade curricular de Programação em Lógica da Faculdade de Engenharia da Universidade do Porto, foi proposta a resolução e geração de um problema de decisão usando programação em lógica com restrições. O objetivo era explorar a utilidade das restrições num problema deste tipo bem como entender o impacto do uso das mesmas no tempo de execução de um programa.

Neste artigo, inicialmente, é feita uma breve descrição do problema escolhido, Gold Star, sendo também apresentada a abordagem que o grupo usou para o resolver. Tratando-se de um problema de satisfação de restrições, é necessário explicar quais as variáveis de decisão usadas e as restrições que limitam o valor destas. Mais à frente, apresentam-se os predicados utilizados para a visualização do puzzle, antes de se apresentar todas as experiências realizadas com instâncias do problema com diferentes dimensões e diferentes estratégias de pesquisa. Por fim, termina-se o artigo com algumas conclusões acerca do trabalho desenvolvido e como o melhorar.

2 Descrição do Problema

O puzzle **Gold Star** tem como objetivo resolver 5 equações, utilizando apenas uma vez cada dígito de 0 a 9. Isto é, pretende-se preencher a estrela, de forma a que as 5 equações sejam verdadeiras quando lidas da esquerda para a direita.

De forma a entender quais são as equações a resolver em cada puzzle, encontram-se representados um puzzle genérico, um puzzle por resolver e a sua respetiva solução.

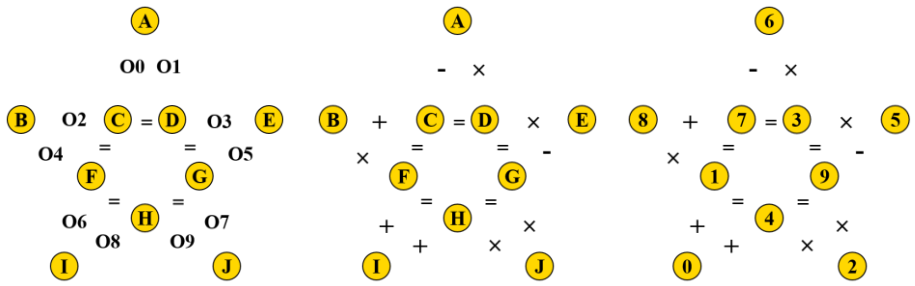

Fig. 1. Puzzle genérico à esquerda, exemplo de um puzzle ao centro e sua respetiva solução à direita

Assim, o sistema de equações a resolver, de acordo com a notação usada na fig.1, é o seguinte:

$$\begin{cases} B\ O2\ C\ =\ D\ O3\ E, \\ B\ O4\ F\ =\ H\ O9\ J, \\ I\ O6\ F\ =\ C\ O0\ A, \\ I\ O8\ H\ =\ G\ O5\ E, \\ A\ O1\ D\ =\ G\ O7\ J \end{cases} \quad (1)$$

Sendo A-J números de 0 a 9 e O*n* operadores (+, -, *, /).

É de notar que foi apresentada a descrição do puzzle de 5 pontas visto que este é o original, no entanto, o programa permite a resolução de puzzles com 3 a 6 pontas, usando esta mesma lógica para encontrar as equações, isto é, tem de se ler o puzzle da esquerda para a direita.

3 Abordagem

Para gerar um puzzle Gold Star e resolvê-lo, é usado o predicado *findSolution/1* que recebe o número de pontas do puzzle, tendo de ser um valor maior do que 3 e menor do que 7. Neste predicado, gera-se, inicialmente, uma lista aleatória de operadores, com o auxílio de *generateRandomOps/2*.

De seguida, procede-se à resolução do problema gerado. No predicado *solveStar/3*, encontra-se um valor para cada variável das equações, garantindo que são aplicadas as restrições corretas de forma a que seja encontrada uma solução válida.

Por fim, imprime-se o puzzle gerado e a solução encontrada como descrito no ponto 4.

Assuma-se que *N* é o número de pontas do puzzle.

3.1 Variáveis de Decisão

Um problema de satisfação de restrições é representado por variáveis que têm associados domínios (valores que estas variáveis podem tomar).

No caso do Gold Star, são necessárias 2**N* variáveis de decisão, guardadas numa lista *Vars* e que representam os números de cada equação, que podem tomar os valores de 0 a 2**N*-1, ou seja, o seu domínio é [0, 2**N*-1]. Tomou-se a opção de adaptar o domínio ao tamanho do puzzle visto que se pretendia manter a regra de que cada número tem ser usado uma única vez. Exemplificando com o puzzle de 5 pontas descrito acima, cada uma das 10 variáveis representa uma variável de *A* a *J* do sistema de equações e o seu domínio é [0, 9].

No puzzle, são necessárias outras 2**N* variáveis de decisão, representando cada uma destas um operador. Neste caso, como só existem 4 possíveis operadores (+, -, *, /), o domínio destas variáveis é [0, 3], estando cada número associado a um único operador dos mencionados, respetivamente.

3.2 Restrições

Para a resolução do puzzle, é necessário impor limites e restrições aos valores que cada variável de decisão pode tomar. Assim, apresentam-se as restrições impostas e como foram conseguidas usando o *prolog*:

Cada número inteiro do domínio só pode ser usado uma única vez. De forma a cumprir esta restrição, foi usado o predicado *all_distinct/1*, que permite que as variáveis de decisão, que representam os números de cada equação, tomem valores distintos, fazendo com que cada número não seja usado mais do que uma vez.

As equações do puzzle têm de ser verdadeiras. Como descrito no ponto 2, o puzzle pode ser traduzido num sistema de equações. Sendo assim, no predicado *restriction*, cada equação foi usada como restrição de forma a que as variáveis, para além de só poderem usar um número do domínio uma única vez, têm de obrigatoriamente tornar verdadeiras as equações do sistema encontrado. Para compreender melhor o predicado *restriction*, é de notar que as variáveis usadas se relacionam da seguinte forma: *W Op1 X = Y Op2 Z*.

No caso da divisão e multiplicação, nenhum das variáveis pode tomar o valor 0. Sabendo que 0 é o elemento absorvente da multiplicação e que a divisão *X/Y=R* é equivalente a *R*Y=X*, pode-se afirmar que, no contexto deste problema, multiplicações e divisões não podem conter variáveis com o valor 0.

Passa-se a explicar em que situações é que cada equação pode resultar em 0:

- Adição: não existe nenhum *X* e *Y* positivos cuja soma resulte no valor 0;
- Subtração: não existe nenhum *X* e *Y* positivos e diferentes que subtraídos resultem em 0;
- Multiplicação: para que os números *X* e *Y* multiplicados resultem em 0, é necessário que, pelo menos, um deles tome o valor de 0;

– Divisão: para que o quociente entre *X* e *Y* seja 0, *X* tem de tomar obrigatoriamente o valor de 0 e, em nenhuma situação, *Y* pode ser igual 0.

Assim, conclui-se que, se uma multiplicação ou divisão contiver a variável 0, obrigatoriamente a equação equivalente terá de resultar em 0, o que levaria a que esta fosse também uma multiplicação ou divisão e que usasse o número 0. Esta situação não é possível no contexto deste problema uma vez que cada número só pode ser usado uma única vez. Para além disso, esta restrição, assegurada pelo predicado *restriction*, garante que a divisão de dois números é sempre válida e definida.

4 Visualização da Solução

O predicado *print_solution/2* foi criado com o objetivo de escrever o puzzle que foi gerado com a respetiva solução, em modo de texto e com uma formatação muito simples. Este pode ser usado por qualquer puzzle e tanto pode ser utilizado para guardar o puzzle e a solução num ficheiro de texto como para os imprimir na consola do *SICStus*.

```
[-,+,-,+,+,*,-,*][6,1,0,5,4,7,3,2]
```

Fig. 2. Representação de um puzzle e respetiva solução, em modo de texto

Quando o puzzle tem 5 pontas, pode-se usar o predicado *print5/2* que, com o auxílio do predicado *format*, imprime o sistema de equações em formato de estrela. Assim, é possível encontrar facilmente e intuitivamente as 5 equações que foram resolvidas com as respetivas soluções.

```
      6
    - *
8 + 7 = 3 * 5
 * =     = -
  1       9
    =   =
 +    4    *
   +   *
0           2
```

Fig. 3. Representação de um puzzle de 5 pontas resolvido

5 Experiências e Resultados

Todas os puzzles resolvidos nestas experiências foram guardados de forma organizada em vários ficheiros de texto na pasta *logs*.

5.1 Análise Dimensional

Como foi referido anteriormente, o predicado *solveStar/3* permite, como primeiro argumento, a escolha do número de pontas do puzzle a gerar. Passa-se a apresentar o

sistema de equações a resolver nos puzzles com 3, 4 e 6 pontas, respetivamente, sendo esta abordagem semelhante à resolução de uma estrela de 5 pontas:

$$\begin{cases} A\ O0\ B\ =\ E\ O5\ F, \\ D\ O2\ B\ =\ C\ O3\ F, \\ D\ O6\ E\ =\ C\ O1\ A \end{cases} \quad (2) \qquad \begin{cases} H\ O6\ F\ =\ B\ O0\ A, \\ A\ O1\ C\ =\ G\ O7\ H, \\ D\ O2\ B\ =\ C\ O3\ E, \\ D\ O4\ F\ =\ G\ O5\ E \end{cases} \quad (3)$$

$$\begin{cases} H\ O6\ F\ =\ C\ O0\ A, \\ A\ O1\ D\ =\ G\ O7\ K, \\ B\ O2\ C\ =\ D\ O3\ E, \\ B\ O4\ F = I\ O10\ L, \\ L\ O11\ J\ =\ G\ O5\ E, \\ \mathrm{H\ O8\ I} = J\ O9\ K \end{cases} \quad (4)$$

Sendo A-L números de 0 a 9 e O*n* operadores (+, -, *, /).

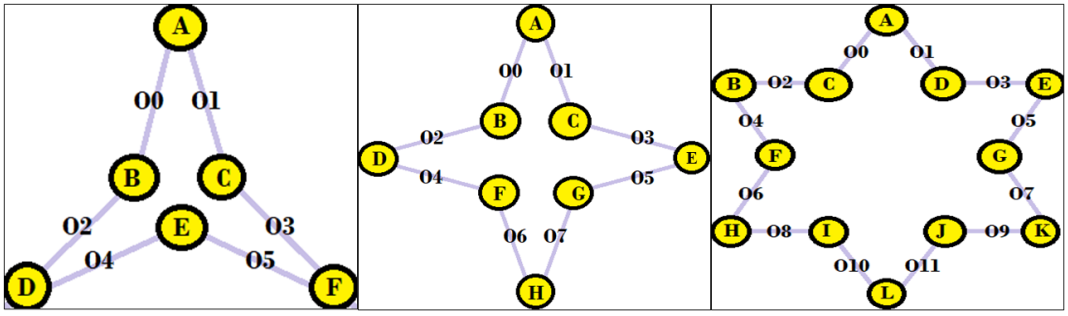

Fig. 4. Puzzles genéricos com: 3 pontas à esquerda, 4 pontas ao centro e 6 pontas à direita

Foram medidos os tempos para gerar e resolver todas as estrelas possíveis para cada uma destas opções, gerando apenas a primeira solução encontrada ou todas as possíveis soluções para cada puzzle. Analisando a Tabela 1, conclui-se que o tempo aumenta exponencialmente com o número de pontas da estrela.

Tabela 1. Tempos de execução em casos em que se pretende uma ou todas as soluções possíveis em puzzles de 3 a 6 pontas

Number of tips	Number of Solutions	Time (s)
3	one	0,179
	all	0,234
4	one	4,373
	all	4,746
5	one	135,143
	all	134,292
6	one	5857,771
	all	3629,125

5.2 Estratégias de Pesquisa

Foram experimentadas todas as 80 combinações possíveis para os argumentos do primeiro argumento do *labeling*, como forma de testar as várias estratégias de pesquisa, tanto no *labeling* dos operadores como no *labeling* dos números. Os testes foram realizados com estrelas de 5 pontas, sendo o tempo medido para cada combinação.

Dos resultados (ver Tabela 2, Tabela 3, Fig.5 e Fig.6), pode-se concluir que a melhor combinação para o *labeling* dos operadores é [*leftmost*/*step*/*up*] e para o *labeling* dos números é [*min*, *middle*, *up*].

6 Conclusões e Trabalho Futuro

Com este trabalho, pode-se concluir que o uso das restrições em *prolog* é realmente muito útil visto que facilita o desenvolvimento de problemas de decisão complexos e permite um menor tempo de execução quando comparado com o tempo de execução do mesmo problema resolvido sem restrições.

O programa desenvolvido pode ser melhorado com algumas restrições nos operadores, garantindo que todos os puzzles encontrados têm sistemas de equações diferentes. Há casos em que se se aplicar uma rotação na estrela, é possível encontrar um puzzle visualmente diferente mas que, no fundo, pode ser resolvido com o mesmo sistema de equações que o primeiro. Assim, não seria necessário gerar e resolver este segundo caso.

No entanto, considera-se que o programa desenvolvido cumpre os objetivos propostos, gerando e resolvendo puzzles de diferentes tamanhos com sucesso.

Referências

1. Gold Star Rules, https://erich-friedman.github.io/puzzle/star/
2. SWI-Prolog, https://www.swi-prolog.org/

Anexos

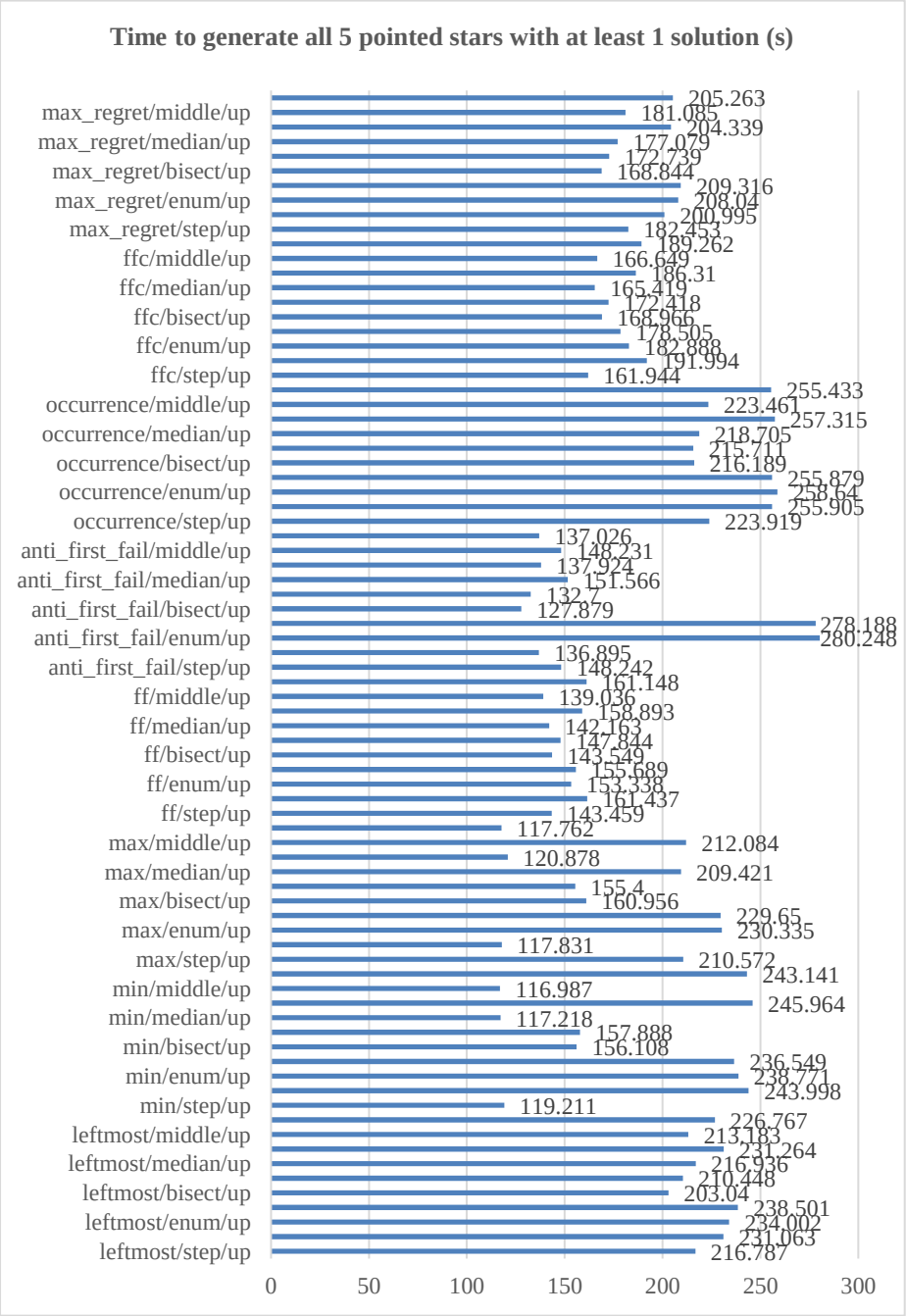

Fig. 5. Gráfico correspondente ao tempo de geração de todas os puzzles com pelo menos uma solução

Tabela 2. Tabela correspondente ao tempo de geração de todas os puzzles com pelo menos uma solução

Combination	Time (s)
leftmost/step/up	216,787
leftmost/step/down	231,063
leftmost/enum/up	234,002
leftmost/enum/down	238,501
leftmost/bisect/up	203,04
leftmost/bisect/down	210,448
leftmost/median/up	216,936
leftmost/median/down	231,264
leftmost/middle/up	213,183
leftmost/middle/down	226,767
min/step/up	119,211
min/step/down	243,998
min/enum/up	238,771
min/enum/down	236,549
min/bisect/up	156,108
min/bisect/down	157,888
min/median/up	117,218
min/median/down	245,964
min/middle/up	116,987
min/middle/down	243,141
max/step/up	210,572
max/step/down	117,831
max/enum/up	230,335
max/enum/down	229,65
max/bisect/up	160,956
max/bisect/down	155,4
max/median/up	209,421
max/median/down	120,878
max/middle/up	212,084
max/middle/down	117,762
ff/step/up	143,459
ff/step/down	161,437
ff/enum/up	153,338
ff/enum/down	155,689
ff/bisect/up	143,549
ff/bisect/down	147,844

Tabela 3. Continuação da página anterior

Combination	Time (s)
ff/median/up	142,163
ff/median/down	158,893
ff/middle/up	139,036
ff/middle/down	161,148
anti_first_fail/step/up	148,242
anti_first_fail/step/down	136,895
anti_first_fail/enum/up	280,248
anti_first_fail/enum/down	278,188
anti_first_fail/bisect/up	127,879
anti_first_fail/bisect/down	132,7
anti_first_fail/median/up	151,566
anti_first_fail/median/down	137,924
anti_first_fail/middle/up	148,231
anti_first_fail/middle/down	137,026
occurrence/step/up	223,919
occurrence/step/down	255,905
occurrence/enum/up	258,64
occurrence/enum/down	255,879
occurrence/bisect/up	216,189
occurrence/bisect/down	215,711
occurrence/median/up	218,705
occurrence/median/down	257,315
occurrence/middle/up	223,461
occurrence/middle/down	255,433
ffc/step/up	161,944
ffc/step/down	191,994
ffc/enum/up	182,888
ffc/enum/down	178,505
ffc/bisect/up	168,966
ffc/bisect/down	172,418
ffc/median/up	165,419
ffc/median/down	186,31
ffc/middle/up	166,649
ffc/middle/down	189,262
max_regret/step/up	182,453

Tabela 4. Continuação da página anterior

Combination	Time (s)
max_regret/step/down	200,995
max_regret/enum/up	208,04
max_regret/enum/down	209,316
max_regret/bisect/up	168,844
max_regret/bisect/down	172,739
max_regret/median/up	177,079
max_regret/median/down	204,339
max_regret/middle/up	181,085
max_regret/middle/down	205,263

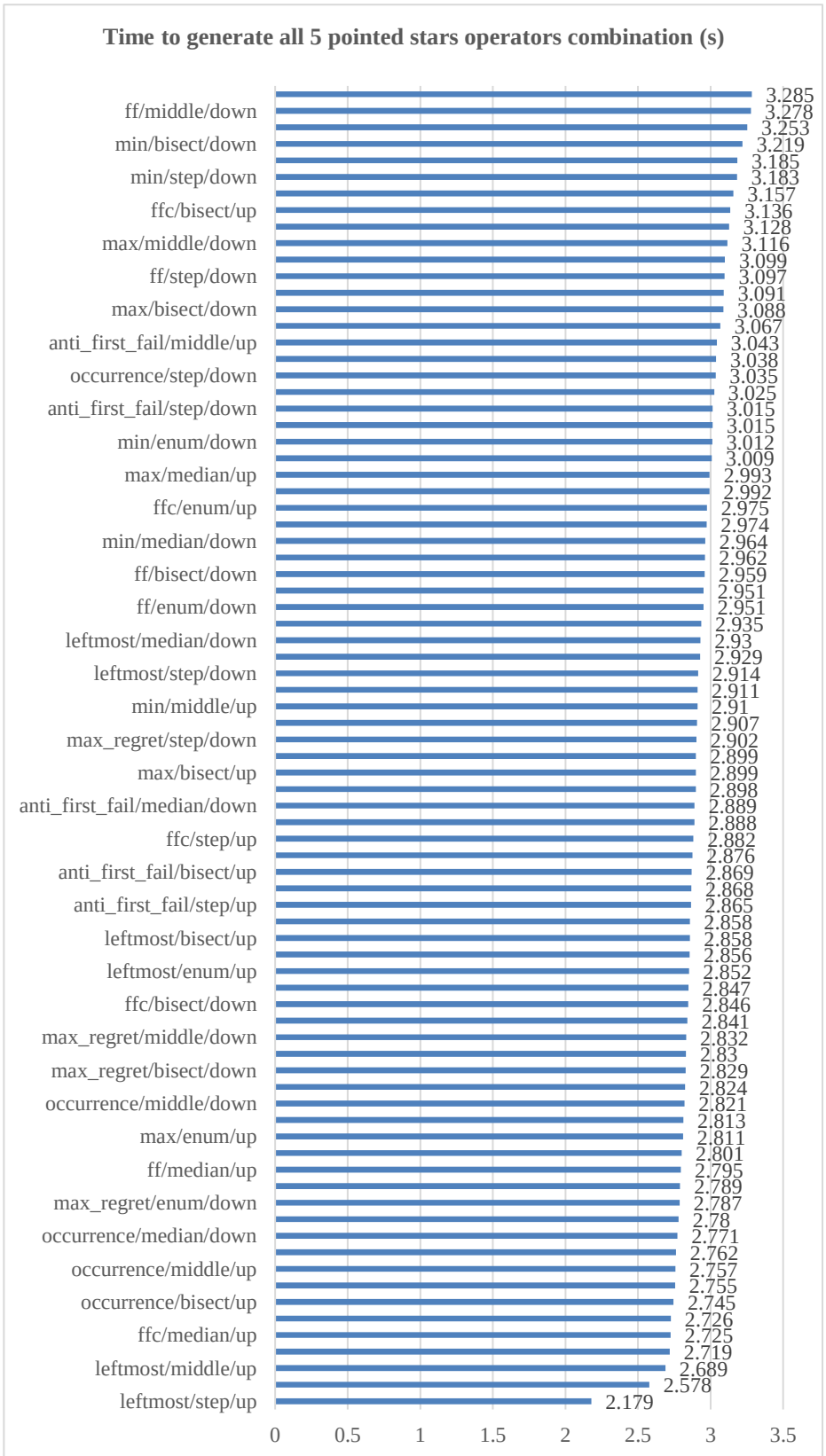
Time to generate all 5 pointed stars operators combination (s)
ff/middle/down 3.285
min/bisect/down 3.278
min/step/down 3.253
ffc/bisect/up 3.219
max/middle/down 3.185
ff/step/down 3.183
max/bisect/down 3.157
anti_first_fail/middle/up 3.136
occurrence/step/down 3.128
anti_first_fail/step/down 3.116
min/enum/down 3.099
max/median/up 3.097
ffc/enum/up 3.091
min/median/down 3.088
ff/bisect/down 3.067
ff/enum/down 3.043
leftmost/median/down 3.038
leftmost/step/down 3.035
min/middle/up 3.025
max_regret/step/down 3.015
max/bisect/up 3.015
anti_first_fail/median/down 3.012
ffc/step/up 3.009
anti_first_fail/bisect/up 2.993
anti_first_fail/step/up 2.992
leftmost/bisect/up 2.975
leftmost/enum/up 2.974
ffc/bisect/down 2.964
max_regret/middle/down 2.962
max_regret/bisect/down 2.959
occurrence/middle/down 2.951
max/enum/up 2.951
ff/median/up 2.935
max_regret/enum/down 2.93
occurrence/median/down 2.929
occurrence/middle/up 2.914
occurrence/bisect/up 2.911
ffc/median/up 2.91
leftmost/middle/up 2.907
leftmost/step/up 2.902
2.899
2.899
2.898
2.889
2.888
2.882
2.876
2.869
2.868
2.865
2.858
2.858
2.856
2.852
2.847
2.846
2.841
2.832
2.83
2.829
2.824
2.821
2.813
2.811
2.801
2.795
2.789
2.787
2.78
2.771
2.762
2.757
2.755
2.745
2.726
2.725
2.719
2.689
2.578
2.179
0
0.5
1
1.5
2
2.5
3
3.5

Fig. 6. Gráfico correspondente ao tempo de geração de puzzles com todas as combinações de operadores possíveis

Tabela 3. Tabela correspondente ao tempo de geração de todas os puzzles com pelo menos uma solução

Combination	Time (s)
leftmost/step/up	2,179
min/enum/up	2,578
leftmost/middle/up	2,689
occurrence/enum/down	2,719
ffc/median/up	2,725
max_regret/median/down	2,726
occurrence/bisect/up	2,745
occurrence/enum/up	2,755
occurrence/middle/up	2,757
ff/enum/up	2,762
occurrence/median/down	2,771
leftmost/middle/down	2,78
max_regret/enum/down	2,787
anti_first_fail/middle/down	2,789
ff/median/up	2,795
max_regret/enum/up	2,801
max/enum/up	2,811
occurrence/bisect/down	2,813
occurrence/middle/down	2,821
max/enum/down	2,824
max_regret/bisect/down	2,829
min/middle/down	2,83
max_regret/middle/down	2,832
leftmost/enum/down	2,841
ffc/bisect/down	2,846
ff/bisect/up	2,847
leftmost/enum/up	2,852
max_regret/median/up	2,856
leftmost/bisect/up	2,858
max_regret/step/up	2,858
anti_first_fail/step/up	2,865
ffc/enum/down	2,868
anti_first_fail/bisect/up	2,869

Tabela 3. Continuação da página anterior

Combination	Time (s)
max_regret/middle/up	2,876
ffc/step/up	2,882
max_regret/bisect/up	2,888
anti_first_fail/median/down	2,889
leftmost/median/up	2,898
max/bisect/up	2,899
occurrence/step/up	2,899
max_regret/step/down	2,902
occurrence/median/up	2,907
min/middle/up	2,91
anti_first_fail/enum/up	2,911
leftmost/step/down	2,914
min/median/up	2,929
leftmost/median/down	2,93
min/bisect/up	2,935
ff/enum/down	2,951
ffc/step/down	2,951
ff/bisect/down	2,959
min/step/up	2,962
min/median/down	2,964
anti_first_fail/bisect/down	2,974
ffc/enum/up	2,975
anti_first_fail/median/up	2,992
max/median/up	2,993
ffc/middle/down	3,009
min/enum/down	3,012
max/middle/up	3,015
anti_first_fail/enum/down	3,025
occurrence/step/down	3,035
ffc/middle/up	3,038
anti_first_fail/middle/up	3,043
max/step/up	3,067
max/bisect/down	3,088
ff/step/up	3,091
ff/step/down	3,097

Tabela 3. Continuação da página anterior

Combination	Time (s)
ff/median/down	3,099
max/middle/down	3,116
leftmost/bisect/down	3,128
ffc/bisect/up	3,136
ff/middle/up	3,157
min/step/down	3,183
max/median/down	3,185
min/bisect/down	3,219
ffc/median/down	3,253
ff/middle/down	3,278
max/step/down	3,285

Código Fonte

```
:- use_module(library(random)).
:- use_module(library(lists)).
:- use_module(library(clpfd)).

example(3, [+,-,-,/,-,+]).
example(4, [/,+,-,-,/,/,+,-]).
example(5, [+,+,-,-,*,+,/,+,/,+]).
example(6, [+,/,+,+,+,+,-,-,-,-,-,+]).

operation(0, +).
operation(1, *).
operation(2, -).
operation(3, /).

generateAllOps(Points, Params):-
    generateOps(Points, _, Params),
    fail.
generateAllOps(_, _).

generateOps(Points, OpsSymbols):-
    Points>2,
    Points<7,
```

```
    L is Points*2,
    length(Ops, L),
    domain(Ops, 0, 3),
    labeling([leftmost, step, up], Ops),
    opsConvert(Ops, OpsSymbols).

generateOps(Points, OpsSymbols, Params):-
    Points>2,
    Points<7,
    L is Points*2,
    length(Ops, L),
    domain(Ops, 0, 3),
    labeling(Params, Ops),
    opsConvert(Ops, OpsSymbols).

generateRandomOps(Points, OpsSymbols):-
    Points>2,
    Points<7,
    L is Points*2,
    length(Ops, L),
    createRandom(Ops),
    opsConvert(Ops, OpsSymbols).

createRandom([]).
createRandom([H | T]):-
    random(0, 4, H),
    createRandom(T).

opsConvert([], []).
opsConvert([HCode | TCodes], [HSymbol | TSymbols]):-
    operation(HCode, HSymbol),
    opsConvert(TCodes, TSymbols).

solveAll(Points, OneSolution):-
    Points>2,
    Points<7,
    generateOps(Points, Ops),
    solveStar(Points, Ops, OneSolution),
    fail.
solveAll(Points, _):-
    Points>2,
    Points<7.

solveAll(Points, OneSolution, Params):-
    Points>2,
```

```
    Points<7,
    generateOps(Points, Ops),
    solveStar(Points, Ops, OneSolution, Params),
    fail.

solveAll(Points, _, _):-
    Points>2,
    Points<7.

findSolution(Points):-
    Points>2,
    Points<7,
    repeat,
    generateRandomOps(Points, Ops),
    solveStar(Points, Ops, 1).

restriction(Op1, W, X, Op2, Y, Z):-
    restriction(Op1, W, X, Value),
    restriction(Op2, Y, Z, Value).

restriction(+, X, Y, R):-
    X + Y #= R.

restriction(-, X, Y, R):-
    X - Y #= R.

restriction(*, X, Y, R):-
    X #\= 0,
    Y #\= 0,
    X * Y #= R.

restriction(/, X, Y, R):-
    X #\= 0,
    Y #\= 0,
    R * Y #= X.

solveStar(3, Ops, OneSolution):-
    Ops = [O0, O1, O2, O3, O4, O5],
    Vars = [A, B, C, D, E, F],
    domain(Vars, 0, 5),
    all_distinct(Vars),
    restriction(O0, A, B, O5, E, F),
    restriction(O2, D, B, O3, C, F),
    restriction(O4, D, E, O1, C, A),
    !, labeling([min, middle, up], Vars),
```

```
    print_solution(Vars, Ops),
    ((OneSolution == 1, !);(OneSolution == 0)).

solveStar(3, Ops, OneSolution, Params):-
    Ops = [O0, O1, O2, O3, O4, O5],
    Vars = [A, B, C, D, E, F],
    domain(Vars, 0, 5),
    all_distinct(Vars),
    restriction(O0, A, B, O5, E, F),
    restriction(O2, D, B, O3, C, F),
    restriction(O4, D, E, O1, C, A),
    !, labeling(Params, Vars),
    print_solution(Vars, Ops),
    ((OneSolution == 1, !);(OneSolution == 0)).

solveStar(4, Ops, OneSolution):-
    Ops = [O0, O1, O2, O3, O4, O5, O6, O7],
    Vars = [A, B, C, D, E, F, G, H],
    domain(Vars, 0, 7),
    all_distinct(Vars),
    restriction(O6, H, F, O0, B, A),
    restriction(O1, A, C, O7, G, H),
    restriction(O2, D, B, O3, C, E),
    restriction(O4, D, F, O5, G, E),
    !, labeling([min, middle, up], Vars),
    print_solution(Vars, Ops),
    ((OneSolution == 1, !);(OneSolution == 0)).

solveStar(4, Ops, OneSolution, Params):-
    Ops = [O0, O1, O2, O3, O4, O5, O6, O7],
    Vars = [A, B, C, D, E, F, G, H],
    domain(Vars, 0, 7),
    all_distinct(Vars),
    restriction(O6, H, F, O0, B, A),
    restriction(O1, A, C, O7, G, H),
    restriction(O2, D, B, O3, C, E),
    restriction(O4, D, F, O5, G, E),
    !, labeling(Params, Vars),
    print_solution(Vars, Ops),
    ((OneSolution == 1, !);(OneSolution == 0)).

solveStar(5, Ops, OneSolution):-
    Ops = [O0, O1, O2, O3, O4, O5, O6, O7, O8, O9],
    Vars = [A, B, C, D, E, F, G, H, I, J],
    domain(Vars, 0, 9),
```

```
    all_distinct(Vars),
    restriction(O2, B, C, O3, D, E),
    restriction(O4, B, F, O9, H, J),
    restriction(O6, I, F, O0, C, A),
    restriction(O8, I, H, O5, G, E),
    restriction(O1, A, D, O7, G, J),
    !, labeling([min, middle, up], Vars),
    nl, print_solution(Vars, Ops),
    print5(Vars, Ops), nl,
    ((OneSolution == 1, !);(OneSolution == 0)).

solveStar(5, Ops, OneSolution, Params):-
    Ops = [O0, O1, O2, O3, O4, O5, O6, O7, O8, O9],
    Vars = [A, B, C, D, E, F, G, H, I, J],
    domain(Vars, 0, 9),
    all_distinct(Vars),
    restriction(O2, B, C, O3, D, E),
    restriction(O4, B, F, O9, H, J),
    restriction(O6, I, F, O0, C, A),
    restriction(O8, I, H, O5, G, E),
    restriction(O1, A, D, O7, G, J),
    !, labeling(Params, Vars),
    print_solution(Vars, Ops),
    ((OneSolution == 1, !);(OneSolution == 0)).

solveStar(6, Ops, OneSolution):-
    Ops = [O0, O1, O2, O3, O4, O5, O6, O7, O8, O9, O10,
O11],
    Vars = [A, B, C, D, E, F, G, H, I, J, K, L],
    domain(Vars, 0, 11),
    all_distinct(Vars),
    restriction(O6, H, F, O0, C, A),
    restriction(O1, A, D, O7, G, K),
    restriction(O2, B, C, O3, D, E),
    restriction(O4, B, F, O10, I, L),
    restriction(O11, L, J, O5, G, E),
    restriction(O8, H, I, O9, J, K),
    !, labeling([min, middle, up], Vars),
    print_solution(Vars, Ops),
    ((OneSolution == 1, !);(OneSolution == 0)).

solveStar(6, Ops, OneSolution, Params):-
    Ops = [O0, O1, O2, O3, O4, O5, O6, O7, O8, O9, O10,
O11],
    Vars = [A, B, C, D, E, F, G, H, I, J, K, L],
```

```
    domain(Vars, 0, 11),
    all_distinct(Vars),
    restriction(O6, H, F, O0, C, A),
    restriction(O1, A, D, O7, G, K),
    restriction(O2, B, C, O3, D, E),
    restriction(O4, B, F, O10, I, L),
    restriction(O11, L, J, O5, G, E),
    restriction(O8, H, I, O9, J, K),
    !, labeling(Params, Vars),
    print_solution(Vars, Ops),
    ((OneSolution == 1, !);(OneSolution == 0)).

print_solution(Vars, Ops):-
    print(Ops),
    print(Vars), nl.

print5(Vars, Ops):-
    Ops = [O0, O1, O2, O3, O4, O5, O6, O7, O8, O9],
    Vars = [A, B, C, D, E, F, G, H, I, J],
    format('~t~d~t~14|', [A]), nl,
    format('~4|~t~w~t~w~t~10|', [O0, O1]), nl,
    for-
mat('~t~d~t~2|~t~w~t~4|~t~d~t~6|~t=~t~8|~t~d~t~10|~t~w~t~
12|~t~d~t~14|', [B, O2, C, D, O3, E]), nl,
    format('~t~w~t=~t~6|~8|~t=~t~w~t~14|', [O4, O5]), nl,
    format('~t~d~t~6|~8|~t~d~t~14|', [F, G]), nl,
    format('~4|~t=~t~6|~8|~t=~t~10|', []), nl,
    format('~t~w~t~3|~t~d~t~11|~t~w~t~13|', [O6, H, O7]),
nl,
    format('~2|~t~w~t~w~t~12|', [O8, O9]), nl,
    format('~t~d~t~2|~12|~t~d~t~14|', [I, J]), nl.

generateSearch(Params):-
    search1(X),
    search2(Y),
    search3(Z),
    Params = [X, Y, Z].

search1(leftmost).
search1(min).
search1(max).
search1(ff).
search1(anti_first_fail).
search1(occurrence).
search1(ffc).
```

```
search1(max_regret).
search2(step).
search2(enum).
search2(bisect).
search2(median).
search2(middle).
search3(up).
search3(down).

testSearchOps:-
    generateSearch(Params),
    statistics(runtime, [TI | _]),
    generateAllOps(5, Params),
    statistics(runtime, [TF | _]),
    T is TF - TI,
    format('~w took ~3d seconds', [Params, T]), nl,
    fail.
testSearchOps.

testSearchStars(Points, OneSolution):-
    Points>2,
    Points<7,
    generateSearch(Params),
    statistics(walltime, _),
    solveAll(Points, OneSolution, Params),
    statistics(walltime, [_|T]),
    format('~w took ~3d seconds', [Params, T]), nl,
    fail.
testSearchStars(Points, _):-
    Points>2,
    Points<7.

getFileHeuristicsStar([S1, S2, S3], Path):-
    atom_concat('docs/logs/', S1, P1),
    atom_concat(P1, '/', P2),
    atom_concat(P2, S2, P3),
    atom_concat(P3, '/', P4),
    atom_concat(P4, S3, P5),
    atom_concat(P5, '/5points.txt', Path).


getFileHeuristicsOps([S1, S2, S3], Path):-
    atom_concat('docs/logs/', S1, P1),
    atom_concat(P1, '/', P2),
    atom_concat(P2, S2, P3),
```

```
    atom_concat(P3, '/', P4),
    atom_concat(P4, S3, P5),
    atom_concat(P5, '/ops.txt', Path).

getFilePoints(Points, 0, Path):-
    number_chars(Points, PArray),
    atom_chars(P0, PArray),
    atom_concat('docs/logs/', P0, P1),
    atom_concat(P1, '_points/all.txt', Path).

getFilePoints(Points, 1, Path):-
    number_chars(Points, PArray),
    atom_chars(P0, PArray),
    atom_concat('docs/logs/', P0, P1),
    atom_concat(P1, '_points/one.txt', Path).

saveLogsHeuristicsStar:-
    generateSearch(Params),

    getFileHeuristicsStar(Params, 1, Path),
    open(Path, write, S),
    current_output(Console),
    set_output(S),


    statistics(walltime, _),
    solveAll(5, 1, Params),
    statistics(walltime, [_|T]),
    format('~w took ~3d seconds', [Params, T]),

    close(S),
    set_output(Console),
    format('~w took ~3d seconds', [Params, T]), nl,

    fail.
saveLogsHeuristicsStar.

saveLogsOps:-
    generateSearch(Params),

    getFileHeuristicsOps(Params, Path),
    open(Path, write, S),
    current_output(Console),
    set_output(S),
```

```
    statistics(walltime, _),
    generateAllOps(5, Params),
    statistics(walltime, [_|T]),
    format('~w took ~3d seconds', [Params, T]),

    close(S),
    set_output(Console),
    format('~w took ~3d seconds', [Params, T]), nl,

    fail.
saveLogsOps.

points(3).
points(4).
points(5).
points(6).

saveLogsPoints:-
    points(Points),

    getFilePoints(Points, 0, Path),
    open(Path, write, S),
    current_output(Console),
    set_output(S),

    statistics(walltime, _),
    solveAll(Points, 0),
    statistics(walltime, [_|T]),
    format('~d Points, all solutions took ~3d seconds',
[Points, T]),

    close(S),
    set_output(Console),
    format('~d Points, all solutions took ~3d seconds',
[Points, T]), nl,

    getFilePoints(Points, 1, Path1),
    open(Path1, write, S1),
    current_output(Console),
    set_output(S1),

    statistics(walltime, _),
    solveAll(Points, 1),
    statistics(walltime, [_|T1]),
```

```
    format('~d Points, one solution took ~3d seconds',
[Points, T1]),

    close(S1),
    set_output(Console),
    format('~d Points, one solution took ~3d seconds',
[Points, T1]), nl,
    fail.
saveLogsPoints.
```