Clouds

Resolução de um Problema de Decisão usando Programação em Lógica com Restrições

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Clouds\_4:

Andreia Cristina de Almeida Rodrigues - up201404691@fe.up.pt

Gonçalo da Mota Laranjeira Torres Leão - up201406036@fe.up.pt

Faculdade de Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 4200-465 Porto, Portugal

23 de Dezembro de 2016

Resumo

Este artigo complementa o segundo projeto da Unidade Curricular de Programação em Lógica, do Mestrado Integrado em Engenharia Informática e Computação. O objetivo deste trabalho é elaborar um programa usando Programação em Lógica com Restrições capaz de resolver problemas de decisão com eficácia, neste caso, resolver qualquer tabuleiro do jogo Clouds e gerar tabuleiros resolúveis deste mesmo jogo.

Conteúdo

1. Introdução

2. Descrição do Problema

3. Abordagem

3.1 Variáveis de Decisão

3.2 Restrições

3.3 Função de Avaliação

3.4 Estratégia de Pesquisa

4. Visualização da Solução

5. Resultados

6. Conclusões e Trabalho Futuro

Bibliografia

Anexo

1. Introdução

O objetivo deste trabalho é a construção de um programa em Programação em Lógica com Restrições para a resolução de um dos problemas de decisão ou optimização sugeridos.

O nosso grupo decidiu implementar o puzzle Clouds, um problema de decisão.

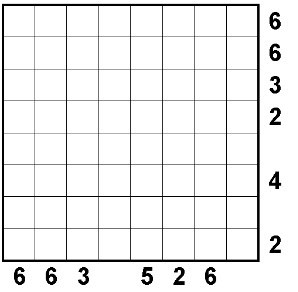
O ambiente de desenvolvimento escolhido foi o SICStus Prolog, que inclui um módulo de resolução de restrições sobre domínios finitos, o clp(FD), utilizado para a resolução deste trabalho devido à sua eficiência e às restrições já pré-definidas.

Este artigo descreve detalhadamente as regras do puzzle Clouds, a abordagem do grupo para a implementação do programa capaz de resolver qualquer instância deste jogo e as variáveis de decisão e restrições utilizadas na elaboração deste programa. Também é referida a função de avaliação da solução obtida e a sua implementação e a estratégia de etiquetagem (labeling) utilizada. É também apresentada a forma utilizada para visualização das soluções obtidas e alguns exemplos de resultados com estatísticas de resolução do puzzle, nomeadamente em relação à sua complexidade (neste caso, o tamanho do tabuleiro ou o número de pistas fornecidas). Finalmente, são apresentadas as conclusões sobre o projeto realizado.

2. Descrição do Problema

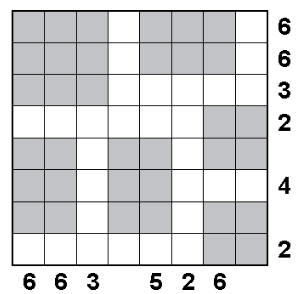
O puzzle Clouds consiste num tabuleiro quadrado de tamanho N onde existem NxN células. Juntamente a este tabuleiro, são colocadas pistas, ou não, junto a cada linha e coluna do tabuleiro, representando o número de células ocupadas com nuvens da linha/coluna respectiva, com o intuito do jogador preencher o tabuleiro baseando-se nessas pistas.

Para além das pistas, as nuvens colocadas no tabuleiro têm de ter a forma de um retângulo ou quadrado e o tamanho mínimo de cada nuvem é de 2 células de comprimento e 2 células de altura. As nuvens, quando posicionadas no tabuleiro, não podem ser adjacentes, nem na diagonal, mantendo sempre uma distância de pelo menos 1 célula vazia em todas as direções face às nuvens mais próximas.



(<https://gyazo.com/7236e8a680fea42101f52fd791214de1>)

Figura 1. Exemplo de um tabuleiro de Clouds com dimensões 8x8, com as respectivas pistas que permitem a resolução do puzzle.



(<https://gyazo.com/434f61ad00c7de28c2fe34d1118ee511>)

Figura 2. Solução do puzzle apresentado na figura anterior, em que o número de células ocupadas por nuvens de cada linha e coluna corresponde ao número apresentado pela pista inicial e nenhuma das nuvens é adjacente a outra nuvem, cumprindo todas as regras acima enunciadas.

3. Abordagem de implementação do *solver*

Considera-se até ao final deste artigo que o número de linhas do tabuleiro é designado por *NLines*, e o número de colunas por *NCols*.

O *solver* recebe como valores de entrada duas listas, *LineClues* e *ColClues*, que indicam as pistas para as linhas e colunas respectivamente. Associado ao n-ésimo elemento de *LineClues* e *ColClues*, está a pista para a n-ésima linha e coluna do tabuleiro, respetivamente. Caso uma pista não seja fornecida para uma dada linha ou coluna, deve ser dado o valor *none* para a variável que lhe está associada.

Como *output*, o *solver* produz uma lista de listas chamadas *Lines*, composta pelas variáveis de decisão. O significado dessas variáveis será apresentado de seguida.

3.1. Variáveis de decisão

Cada variável de decisão corresponde uma célula do tabuleiro. Assim sendo, o número de variáveis de decisão é igual a NLines \* NCols.

O domínio de cada variável corresponde ao conjunto {0, 1}. Se uma variável tiver como valor 0, isso significa que a célula corresponde está pintada de branco, não estando a ser ocupada por uma nível. Por oposição, uma variável com valor a 1 significa que a célula correspondente está pintada de cinzento e é ocupada por uma nuvem.

Para facilidade de representação, estas variáveis de decisão estão organizadas numa lista de listas, em que cada sub-lista representa uma linha do tabuleiro, e cada elemento, uma célula. Esta lista bi-dimensional corresponde ao *output* já mencionado do solver, *Lines*.

Após serem definidas as restrições, para efetuar o *labeling*, é preciso que as todas as variáveis de decisão sejam incluídas numa única lista de inteiros, pelo que é usado o predicado *append/2* para achatar a lista *Lines*.

3.2. Restrições

A resolução de um *puzzle* pode ser resumido em duas restrições:

* as pistas das linhas e colunas devem ser respeitadas
* as nuvens devem ser “bem formadas”

**As pistas das linhas e colunas devem ser respeitadas**

Esta restrição impõe que, caso seja dada uma pista para uma dada linha ou coluna, esta deve ter tantas células a cinzento quanto o número indicado pela pista.

É possível decompor esta restrição em duas sub-restrições, uma para a conformidade do tabuleiro às pistas das linhas, outro, para as pistas das colunas.

Para a sub-restrição das pistas das linhas, percorreu-se a lista *Lines* de forma recursiva, e, para cada sublista (linha do tabuleiro), é imposto que a soma dos seus elementos tem que ser igual à pista da linha respectiva, com o auxílio da restrição aritmética *sum/3*. Caso a pista seja *none*, não é imposta esta restrição para essa linha do tabuleiro. Esta restrição é imposta com o auxílio do predicado *checkLines/2*, que percorre as linhas em *Lines*, e *checkClue/2*, que percorre cada elemento de uma sublista.

Para a segunda sub-restrição, para as pistas das colunas, trabalhou-se com a matriz transposta à de *Lines*, à qual se deu o nome de *Cols*. Nesta lista de listas, cada sublista representa uma coluna da matriz. Assim, adotando um método análogo ao da sub-restrição das pistas das linhas em lines, as restrições para as pistas das colunas são aplicadas em *Cols*.

**As nuvens devem ser “bem formadas”**

Entende-se por nuvem “bem formada” um retângulo que tem pelo menos duas linhas e duas colunas e que não é adjacente a mais nenhuma nuvem (nem na diagonal), tal como especificado pelas regras do jogo.

Esta restrição é mais complexa pois, antes do *labeling*, não são conhecidas as posições das nuvens.

Esta restrição foi implementada percorrendo a lista *Lines* elemento a elemento e impondo a seguinte implicação material:

a célula corresponde ao canto superior esquerdo de uma nuvem -> a nuvem associada é “bem formada”

A implicação material apresentada acima foi implementada usando uma restrição proposicional: *IsCorrectCloud #<= IsCloudULC*.

*IsCloudULC* é uma variável (booleano) que indica se a célula é o ULC (*upper left corner* - canto superior esquerdo) de uma nuvem. Para isso, testou-se se a própria célula é cinzenta e se a célula à sua esquerda (na mesma linha) e acima (na mesma coluna) são ambas brancas. No código, a obtenção da cor de uma célula é implementada pelo predicado *checkColor(Lines,LineIdx-ColIdx,Color)*, onde *Lineidx* e *ColIdx* são os índices da linha e coluna (a começar em 1) e *Color* é a cor da célula. Para facilitar a implementação, as célula cujo índice da linha ou coluna é 0 (células “fora do tabuleiro”) são sempre brancas.

*IsCorrectCloud* é uma variável (booleano) que indica se a nuvem que tem como ULC uma dada célula do tabuleiro é “bem formada”.   
Para testar isto, é primeiro contado, de forma recursiva, o número de células consecutivas à direita da célula que se pretende testar que são cinzentas (a contar com a célula da partida). A este número de celulas, chamemos *CloudWidth* O critério de paragem desta recursão é chegar-se ao final do tabuleiro (o índice da coluna é superior a *NCols*) e não ao final da nuvem pois, mais uma vez, não se sabe à partida onde serão desenhadas as nuvens. Esta contagem é feita usando o predicado *countShadedLine/4*.   
De seguida, usando um método análogo com a matriz transposta *Cols*, calcula-se o número de células consecutivas por baixo da célula a testar que são cinzentas, *CloudHeight*.  
Para testar se a nuvem não é adjacente com outras nuvens, é preciso calcular o *CloudTopWidth*, que corresponde ao número de células consecutivas a branco à direita da célula uma linha acima e uma coluna à esquerda da célula a testar. Tal como para o *CloudWidth*, o critério de paragem da recursão é chegar a uma célula fora do tabuleiro. É necessário também contar o número de células consecutivas a branco por baixo da célula uma unidade à esquerda e acima da célula a testar, *CloudSideHeight*. Estas contagens são é feitas com o predicado *countUnshadedLine/4*, sendo que, para *CloudTopWidth* é usada a lista *Lines*, e para *CloudSideHeight*, a lista *Cols*.  
Por fim, para garantir o formato rectangular da nuvem, é preciso contar o número de linhas (colunas, respetivamente) da nuvem que têm a mesma largura (altura, resp.) que *CloudWidth* (*CloudHeight*, resp.). A este número damos o nome de  *RectWidth* (*RectHeight*, resp.). Para isso, é usado o predicado *checkRectHeight/6* que percorre a nuvem linha a linha (coluna a coluna, resp.) usando a lista *Lines* (*Cols*, resp.).

Finalmente, tendo estas seis variáveis auxiliares definidas, é possível definir o valor de *IsCorrectCloud*, que vale 1, se todas as seguintes condições se verificarem:

* *CloudWidth >= 2* (a nuvem tem uma largura de pelo menos 2 unidades).
* *CloudHeight* >= 2 (a nuvem tem uma altura de pelo menos 2 unidades).
* *CloudTopWidth >= CloudWidth* + 2 (para garantir que o topo nuvem não é adjacente a nenhuma outra núvem; a contagem foi feita na célula 1 unidade à esquerda a acima da célula a testar para também cobrir os casos de haver adjacências na diagonal).
* *CloudSideHeight* >= *CloudHeight* + 2 (para impedir que o lado esquerdo da nuvem seja adjacente a outra nuvem, adjacências nas diagonais incluídas).
* *CloudWidth* = *RectWidth* e *CloudHeight* = *RectHeight* (para garantir que a nuvem tem um formato rectangular).

3.3. Estratégia de pesquisa

O uso de uma boa estratégia de etiquetagem (*labeling*) é importante para aumentar a eficiência temporal do programa.

Neste caso, dados os resultados que serão apresentados a secção 5, optou-se de não usar nenhuma opção de *labeling*. Com efeito, as observações empíricas mostram que não usar nenhuma opção, no caso concreto do *solver* desenvolvido, é o mais eficiente.

Em particular, as opções de *labeling* para escolha do próximo valor a atribuir a uma variável (*bisect*, *median*,...) não são muito interessantes dado que o domínio de todas as variáveis de decisão é apenas composto por dois valores: 0 e 1.

4. Abordagem de implementação do gerador

O gerador recebe como valores de entrada *NLines*, *NCols*, o número de nuvens a desenhar (*NClouds*), o número de pistas visíveis para as linhas (*NLineClues*) e o número de pistas visíveis para as colunas (*NColClues*).

Como *output*, o gerador produz as listas *LineClues* e *ColClues*, variáveis de *input* do *solver*.

A lógica do gerador implementado é o inverso da do *solver*: começa-se por gerar o tabuleiro (usando programação em lógica com restrições), e a partir deste derivam-se as pistas.

Nesta secção, vamos-nos focar apenas na geração aleatória do tabuleiro, dado que o cálculo das pistas através do tabuleiro é trivial e não usa programação em lógica com restrições.

4.1. Variáveis de decisão

As variáveis de decisão estão organizados em tuplos do tipo *cloud(Xi,Width,Yi,Height)*, que contêm a informação acerca da posição do canto superior esquerdo (*Xi* - índice da coluna, *Yi* - índice da linha), da largura (*Width*) e altura (*Height*) de cada nuvem. Assim, o número de variáveis de decisão é dado por 4\**NClouds*.

O domínio de cada variável *Xi* e *Yi* é o intervalo [1; *NCols* - 1] [1; *NLines* - 1], respetivamente (não são incluídos os valores *NCols* e *NLines* pois sabe-se que cada nuvem deve ter largura e altura maior ou igual a 2).

O domínio de cada variável *Width* e *Height* é o intervalo [3; *NCols* + 1] e [3; *NLines* + 1], respetivamente. Na realidade, o *Width* e o *Height* usados correspondem à largura e altura respetivas da nuvem incrementadas de uma unidade. Este incremento de uma unidade é útil para aplicar a restrição *disjoint2/1*, para garantir que as nuvens não são adjacentes. O limite inferior dos intervalos permite impor que as nuvens devem ter altura e largura de pelo menos 2 unidades.

Para facilidade de representação, estas variáveis de decisão estão organizadas numa lista de tuplos do tipo *Cloud* chamada *Clouds*. Tal como para o *solver*, antes da fase de *labeling* esta lista é achatada num lista de inteiros, *CloudsVars*.

A lista de nuvens é também convertida numa representação sob a forma de uma lista de listas de booleanos, *Lines*, que serve de *output* para a primeira fase do gerador, da criação do tabuleiro. Esta conversão é feita com o predicado *getMatrix/5*.

4.2. Restrições

A geração de um *puzzle* pode ser resumido em quatro restrições:

* as nuvens não podem ser adjacentes
* as nuvens devem estar completamente contidas no tabuleiro
* as nuvens devem estar ordenadas
* o tabuleiro não deve ser simétrico

**As nuvens não podem ser adjacentes**

Para aplicar esta restrição, basta aplicar a restrição combinatória *disjoint2/1* em *Clouds*, que já tem a estrutura apropriada para este predicado.

**As nuvens devem estar completamente contidas no tabuleiro**

Esta restrição permite eliminar algumas soluções redundantes (as porções das nuvens fora do tabuleiro, não afetam o seu aspeto). Para impor esta restrição, é preciso que cada nuvem em *Clouds* respeite as seguintes inequações: (Xi + W - 2) =< NCols, (Yi + H - 2) =< NLines.

Esta restrição é imposta com o predicado *checkCloudsInBounds/3*.

**As nuvens devem estar ordenadas**

Esta solução também permite eliminar soluções redundantes.

As nuvens estão ordenadas pela posição do seu canto superior esquerdo: a n-ésima nuvem da lista deve, ou ter um *Yi* inferior ao da (n+1)-ésima nuvem, ou devem ter o mesmo *Yi* e o *Xi* deve ser inferior ao da (n+1)-ésima nuvem.

Esta restrição é imposta pelo predicado *checkCloudsOrdered/1*.

**O tabuleiro não deve ser simétrico**

Esta restrição foi acrescentada para diminuir a probabilidade que seja gerado um tabuleiro com soluções múltiplas. Esta restrição não cobre todos os casos de multiplicidade de soluções, nomeadamente o caso de tabuleiros que tenham partições simétricas.

Entende-se por “não deve ser simétrico” que exista algum elemento [x][y] (posições a começar em 1) do tabuleiro tal que este e o seu elemento “homólogo”, em [*NCols* - x + 1][*NLines* - y +1] não tenham a mesma cor.

Esta restrição é implementada com o predicado *checkNoDiagonalSymetry/5*, que percorre a metade superior do tabuleiro.

4.3. Estratégia de pesquisa

Para opções do *labeing*, foram usadas as *flags ff* e *value(myValueSelector)*, em que *myValueSelector/4* é um predicado definido pelo grupo para que o posicionamento das nuvens seja aleatório.

5. Visualização da solução

Os predicados responsáveis pela visualização das soluções obtidas, através do solver e generator elaborados, em modo texto foram relativamente fáceis de implementar, sendo que já tínhamos trabalhado com uma representação semelhante no primeiro trabalho desta mesma unidade curricular.

Estes predicados podem ser encontrados no ficheiro printer.pl.

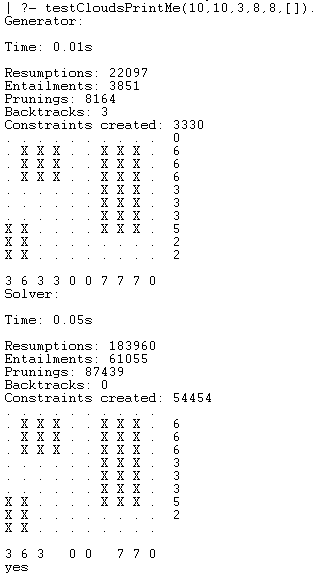


Figura 3. Geração de um tabuleiro e sua representação e resolução desse mesmo tabuleiro com seguida de representação do mesmo.

(<https://gyazo.com/95dd6145d4e78644abde60a67e4b2798>)

O predicado *printClouds(+LineClues, +ColClues, +Lines)* é responsável por imprimir o tabuleiro, linha a linha, incluindo na sua representação as pistas fornecidas para cada linha e coluna desse mesmo tabuleiro. As variáveis recebidas por este predicado são, por ordem, as pistas relativas às linhas, as pistas relativas às colunas e a matriz do tabuleiro.

Este predicado começa por invocar o predicado *printLinesClouds(+LineClues, +Lines)* que é responsável por imprimir o tabuleiro linha a linha e a pista respectiva, de forma recursiva. Por último invoca o predicado *printColClues(+ColClues)* que imprime as pistas de cada coluna do tabuleiro.

Na visualização do tabuleiro utilizamos ‘X’ para representar células que estejam ocupadas por partes de uma nuvem e ‘.’ para representar células vazias. Os números à frente de cada linha/coluna correspondem às pistas fornecidas para a linha/coluna respectiva. Alguns desses valores podem ser nulos e nesse caso nada é representado.

Juntamente com o tabuleiro são apresentados o tempo usado a executar o labeling, através do predicado *print\_time*, e as estatísticas, através do predicado *fd\_statistics*, que permitem uma melhor análise e discussão dos resultados obtidos.

6. Resultados

Para testar o *solver* desenvolvido, foi corrido um conjunto de testes com diferentes dimensões de tabuleiro, diferentes opções de etiquetagem (combinações das opções *down*, *ff* e *ffc*) e fornecendo ou não todas as pistas para as linhas e colunas.

Para cada teste, para diminuir a influência de fatores externos (nomeadamente a disposição das nuvens nos puzzles a resolver), foi recolhida uma amostra de 20 puzzles, gerados aleatoriamente com o gerador desenvolvido, que depois foram resolvidos usando o *solver*.

Em todos os testes descritos, o número de nuvens geradas varia entre 1 e *MaxNClouds* (número máximo de nuvens que podem ser colocadas no tabuleiro dividido por 4 e subtraído de 1) dado pela fórmula:

*MaxNClouds* = div(*MaxNCloudsLines* \* *MaxNCloudsCols*, 4) - 1, com

*MaxNCloudsLines* = div(*NLines* + 1, 3) + 1,

*MaxNCloudsCols* = div(*NCols* + 1, 3) + 1,

sendo que div(a,b) é o quociente da divisão inteira de a por b.

As duas figuras abaixos (figuras 4 e 5) apresentam os resultados dos testes realizados. As figuras apresentam o tempo total (tempo acumulado para os 20 puzzle’s gerados) e o tempo médio (em segundos) gastos pelo *solver* na fase de *labeling*.

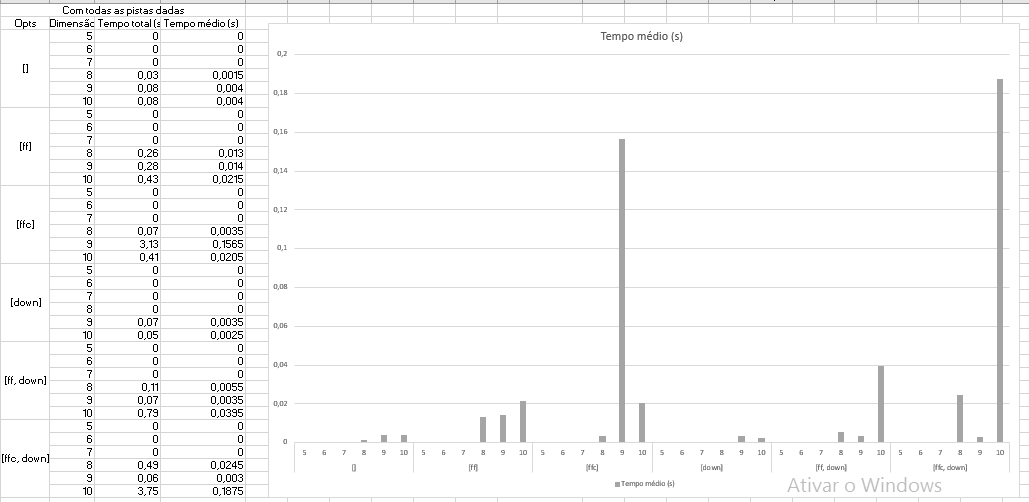


Figura 4. Tempo gasto pelo *solver* em tabuleiros de diferentes dimensões e opções de *labeling*, para tabuleiros onde todas as pistas para as linhas e colunas eram dadas,

<https://gyazo.com/5a2a473727cdbf80f9eacfb72b357133>

Na figura 5, o número de pistas fornecidas para as linhas e para as colunas é um número aleatório entre div(*NLines*, 2) e *NLines*, e div(*NCols*, 2) e *NCols*, respetivamente.

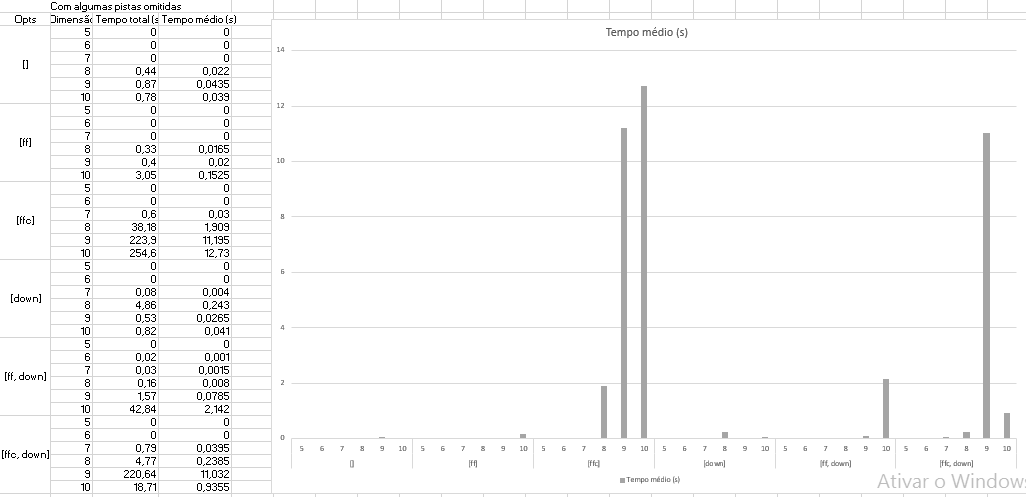


Figura 5. Tempo gasto pelo *solver* em tabuleiros de diferentes dimensões e opções de *labeling*, para tabuleiros onde algumas das pistas para as linhas e colunas foram omitidas, de forma aleatória.

<https://gyazo.com/a33721f8d3d33fd12aa4d45bef1790a8>

Com observações empíricas, foi possível determinar que os valores excepcionalmente elevados para o tempo total e tempo médio decorridos em alguns dos testes devem-se normalmente a *puzzles* específicos da amostra que demoraram mais tempo a ser resolvidos, ou seja, o tempo gasto pelo *solver* pelos diferentes *puzzles* não foi homogéneo. Seria necessário efetuar testes com amostras de maior dimensão para comparar com mais precisão a eficiência das diferentes opções de etiquetagem.

Com os resultados obtidos, é possível, com um certo grau de confiança, tirar as seguintes conclusões:

* o tempo gasto pelo *solver* a resolver um puzzle aumenta quanto maior forem as dimensões do tabuleiro.
* o uso das opções de *labeling* *ff* e *ffc* tornam a fase de *labeling* mais lenta, sendo a opção *ffc* menos eficiente que *ff*. Uma possível explicação é que o tamanho máximo inicial do domínio de todas as variáveis de decisão já é bastante pequeno (2) e o processo de procura da variável de menor domínio e das variáveis envolvidas no maior número de restrições (para o *ffc*) demoram um tempo considerável, dado o número elevado de variáveis de decisão em tabuleiros de maior dimensão.
* o tempo necessário para resolver puzzles em que nem todas as pistas são dadas é maior do que em puzzles onde todas as pistas são fornecidas. Uma explicação plausível é que um menor número de pistas implica um menor número de restrições a serem aplicadas.

Com os resultados produzidos, não é possível afirmar se há alguma diferença em termos de eficiência em usar a opção de *labeling* *down* ou não.

7. Conclusões e Trabalho Futuro

Com a conclusão deste projeto, o grupo concluiu que o uso de PLR (Programação em Lógica com Restrições) é bastante útil para determinadas situações, facilitando muito a programação, juntando a declaratividade da Programação em Lógica com a eficiência da resolução de restrições. É portanto uma extenção à linguagem bastante poderosa para resolver uma ampla variedade de questões de decisão e otimização.

Em relação à implementação, o nosso grupo encontra-se satisfeito com o trabalho realizado e sente que atingiu os objetivos propostos: foi desenvolvido um *solver* e um gerador de puzzles com uma boa eficiência temporal.

Os resultados obtidos provam o bom funcionamento do *solver* e do gerador e permitem justificar as decisões tomadas para a etiquetagem.

Como possíveis melhorias deste trabalho, seria interessante realizar as experiências apresentadas com amostras de maior dimensão, para dar um maior grau de confiança às conclusões retiradas.

Outra possível melhoria seria permitir ao gerador detetar mais casos de puzzles com soluções múltiplas. Em particular, poderiam ser detectadas simetrias em partições do tabuleiro, e não só no tabuleiro como um todo.

relatórios de anos anteriores:

<https://github.com/ferrolho/feup-plog/blob/master/project-2/paper/PLOG_TP2_Star_Battle_5.pdf>

<https://github.com/RuiVilares/PLOG/blob/master/T2/Relat%C3%B3rio/paper.pdf>

<https://github.com/pedrofraga/PLOG-FEUP/blob/master/Pinwheel/report.pdf>