Four Winds

Resolução de Puzzles utilizando Programação em Lógica com Restrições

Bruno Marques and Diogo Silva

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

Resumo Para este trabalho foi proposta a resolução de um puzzle, utilizando programação em lógica com restrições em **SICStus Prolog**, permitindo resolver puzzles numéricos do tipo "Four Winds"de tamanho variável, retornando graficamente a solução para o mesmo. Foram implementados desta forma um conjunto de predicados de modo a aplicar as restrições inerentes ao funcionamento do puzzle e o correto funcionamento da aplicação.

Keywords: Prolog, restraints, logic programming, efficiency, puzzle solving

1 Introdução

Este projeto esta a ser realizado no âmbito da unidade curricular Programação em Lógica do 3º ano do Mestrado Integrado em Engenharia Informática e Computação, da Faculdade de Engenharia da Universidade do Porto.

Tem como objetivo a resolução de um puzzle do tipo "Four Winds", utilizando programação em lógica com restrições.

O problema consiste no preenchimento de um tabuleiro em grelha com setas direcionais de forma a preencher todo o tabuleiro de acordo com as condições impostas pelo puzzle (descritas á frente). Os puzzles podem ainda variar as suas dimensões e número de células capitais.

Ao longo deste artigo iremos reportar o trabalho desenvolvido pelo grupo, vamos iniciar com a descrição do o respetivo problema, procedendo com uma explicação da abordagem tomada pelo grupo para a implementação de uma solução, e finalizando com as conclusões retiradas acerca da solução implementada.

2 Descrição do Problema

O Problema do puzzle "Four Winds" prende-se com o preenchimento de um tabuleiro de dimensões variáveis, com uma estrutura em grelha e com dois tipos de células (numa fase inicial), células vazias e células numéricas com um valor associado. O puzzle impõe de seguida as seguintes restrições para uma resolução válida:

- Todas as células devem ser preenchidas, uma célula considera-se preenchida quando contém um algarismo ou é atravessada por uma linha direcional;
- A soma de todas as linhas direcionais que partem de uma célula deve ser igual ao algarismo da célula;
- As linhas direcionais não se devem cruzar nem sobrepor, formando áreas coesas;
- As linhas direcionais são válidas na direção horizontal e vertical, não sendo assim permitidas linhas diagonais.

O resultado final deve ser um tabuleiro completamente preenchido com células numéricas e células ocupadas por linhas direcionais ininterruptas que partem de cada célula numérica na horizontal ou vertical, ocupando um número de células equivalente ao algarismo presente na célula correspondente.

Deve ser ainda possível resolver tabuleiros de dimensões variáveis, e com células numéricas em disposições diferentes, desde que possíveis.

3 Abordagem

Para fazermos a abordagem a este problema inicialmente procedemos á avaliação das variáveis de decisão necessárias para a resolução do problema, prosseguindo para a elaboração de uma estrutura na qual fosse possível a imposição de restrições de um modo simples e eficiente.

Por fim passamos á elaboração de predicados para resolver os tabuleiros de forma dinâmica, isto é para um tamanho variável.

3.1 Variáveis de Decisão

Para a resolução deste problema o grupo optou pela criação de áreas no tabuleiro que correspondem a cada célula numérica, desta forma cada célula do tabuleiro irá ter um conjunto de variáveis de decisão que será o número de áreas existentes calculadas durante a execução do programa. Estas áreas terão um número igual ao numero de células numéricas.

O domínio será assim limitado pelo número de áreas que estas células poderão ocupar.

3.2 Restrições

As restrições referidas nesta secção do relatório encontram-se no ficheiro *restricts.pl*, á excepção da verificação de conectividade que se encontra em *boardDomain.pl*.

1. **Garantir que cada célula pode apenas pertencer a uma das áreas adjacentes.**

   A função *restrictRow* encarrega-se de aplicar esta restrição verificando as 4 células adjacentes, acima, abaixo, e em ambos os lados, restringindo o domínio da variável de decisão a estas.

2. **Garantir que a área total ocupada pelas células de uma área, corresponde ao algarismo indicado na célula numérica.**

   A função *getCount* encarrega-se de aplicar esta restrição verificando se a área correspondente á região que a célula numérica ocupa corresponde ao algarismo indicado.

3. **Verificar se as áreas ocupadas são ininterruptas, ou não se sobrepõem**

   Esta restrição surge de modo tardio pois surge de uma situação pouco comum na qual as áreas se sobrepõem ou interesetam, verificando assim a função *checkConectivity* se existe conexão completa entre todas as áreas apresentadas

3.3 Função de Avaliação

Para a avaliação do problema utilizamos uma função disponibilizada online para verificar tempos de execução assim como o número de backtrackings executados, resumptions, entailments, prunings e constraints criadas. Foi através deste método que acabamos por fazer alguns ajustes ao labeling utilizado no programa, pois as diferenças em performance não eram muito pronunciadas. Os resultados que obtivemos foram os seguintes:

```
Time taken to find solution: 1.147 seconds
Resumptions: 7918772
Entailments: 2732007
Prunings: 3678063
Backtracks: 113983
Constraints created: 318368
```

3.4 Estratégia de Pesquisa

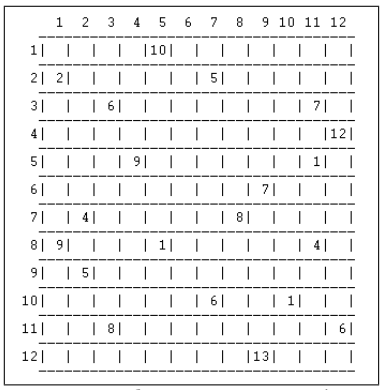
A estratégia de labeling passa por utilizar o predicado de *labeling* com a seguinte estrutura:

labeling([step], Solution)

A opção ***step*** melhora a eficiência ligeiramente comparada com outras estratégias e talvez devido ao grande número de backtracks ou a uma relativa dificuldade em aplicar mais restrições pela parte do grupo as mudanças ta etiquetagem do label não se traduzam numa variância de performance significativa.

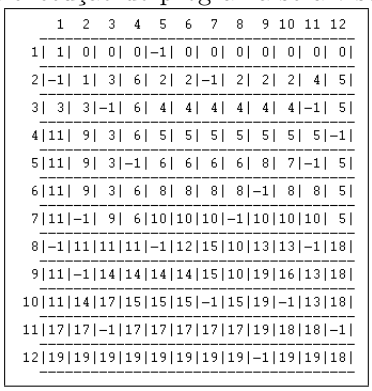
4 Visualização da Solução

Antes da execução do programa o tabuleiro será apresentado da seguinte forma:



	1	2	3	4	5	6	7	8	9	10	11	12
1					10							
2	2						5					
3			6								7	
4												12
5				9							1	
6									7			
7		4						8				
8	9				1						4	
9		5										
10							6			1		
11			8									6
12								13				

Sendo que depois da execução do programa será visualizado desta forma:



	1	2	3	4	5	6	7	8	9	10	11	12
1	1	0	0	0	-1	0	0	0	0	0	0	0
2	-1	1	3	6	2	2	-1	2	2	2	4	5
3	3	3	-1	6	4	4	4	4	4	4	-1	5
4	11	9	3	6	5	5	5	5	5	5	5	-1
5	11	9	3	-1	6	6	6	6	8	7	-1	5
6	11	9	3	6	8	8	8	8	-1	8	8	5
7	11	-1	9	6	10	10	10	-1	10	10	10	5
8	-1	11	11	11	-1	12	15	10	13	13	-1	18
9	11	-1	14	14	14	14	15	10	19	16	13	18
10	11	14	17	15	15	15	-1	15	19	-1	13	18
11	17	17	-1	17	17	17	17	17	19	18	18	-1
12	19	19	19	19	19	19	19	19	-1	19	19	18

Aqui podemos observar que as células vazias foram substituídas por áreas coesas numeradas, que representam as regiões correspondentes a cada célula numérica que agora passa a estar representada por um -1.

Esta representação embora na nossa opinião permita perceber que o programa executou corretamente não é ótima e reconhecemos isso, isto advém do facto de existirem dois tipos de *Boards* um Board sobre o qual as operações de labeling e as restrições são executadas, e outro com funcionalidades de display, isto advêm do facto da visão de projeto inicial passar a ser incompatível com algumas das características requeridas pelo programa o que levou a este conflito.

5 Resultados

Nesta secção serão mostrados os resultados da execução do programa para um tabuleiro de dimensões 12x12:

```
Time taken to find solution: 1.147 seconds
Resumptions: 7918772
Entailments: 2732007
Prunings: 3678063
Backtracks: 113983
Constraints created: 318368
    1  2  3  4  5  6  7  8  9 10 11 12
  ---------------------------------------
 1|  |  |  |  |10|  |  |  |  |  |  |  |
  ---------------------------------------
 2| 2|  |  |  |  |  | 5|  |  |  |  |  |
  ---------------------------------------
 3|  |  | 6|  |  |  |  |  |  |  | 7|  |
  ---------------------------------------
 4|  |  |  |  |  |  |  |  |  |  |  |12|
  ---------------------------------------
 5|  |  |  | 9|  |  |  |  |  |  | 1|  |
  ---------------------------------------
 6|  |  |  |  |  |  |  |  | 7|  |  |  |
  ---------------------------------------
 7|  | 4|  |  |  |  |  | 8|  |  |  |  |
  ---------------------------------------
 8| 9|  |  |  | 1|  |  |  |  |  | 4|  |
  ---------------------------------------
 9|  | 5|  |  |  |  |  |  |  |  |  |  |
  ---------------------------------------
10|  |  |  |  |  |  | 6|  |  | 1|  |  |
  ---------------------------------------
11|  |  | 8|  |  |  |  |  |  |  |  | 6|
  ---------------------------------------
12|  |  |  |  |  |  |  |  |13|  |  |  |
  ---------------------------------------
    1  2  3  4  5  6  7  8  9 10 11 12
  ---------------------------------------
 1| 1| 0| 0| 0|-1| 0| 0| 0| 0| 0| 0| 0|
  ---------------------------------------
 2|-1| 1| 3| 6| 2| 2|-1| 2| 2| 2| 4| 5|
  ---------------------------------------
 3| 3| 3|-1| 6| 4| 4| 4| 4| 4| 4|-1| 5|
  ---------------------------------------
 4|11| 9| 3| 6| 5| 5| 5| 5| 5| 5|-1|
  ---------------------------------------
 5|11| 9| 3|-1| 6| 6| 6| 6| 8| 7|-1| 5|
  ---------------------------------------
 6|11| 9| 3| 6| 8| 8| 8| 8|-1| 8| 8| 5|
  ---------------------------------------
 7|11|-1| 9| 6|10|10|10|-1|10|10|10| 5|
  ---------------------------------------
 8|-1|11|11|11|-1|12|15|10|13|13|-1|18|
  ---------------------------------------
 9|11|-1|14|14|14|14|15|10|19|16|13|18|
  ---------------------------------------
10|11|14|17|15|15|15|-1|15|19|-1|13|18|
  ---------------------------------------
11|17|17|-1|17|17|17|17|17|19|18|18|-1|
  ---------------------------------------
12|19|19|19|19|19|19|19|19|-1|19|19|18|
  ---------------------------------------
```

Os resultados mostram-se aceitáveis, mas no entanto existe por vezes uma alta variância dependendo das características do tabuleiro, acreditamos que isto se prende com o numero de soluções que cada tabuleiro oferece.

De seguida o programa é corrido para diferentes tabuleiros e são apresentadas algumas estatísticas:

Tamanho	Tempo	Backtracks	Resumptions	Entailments	Prunings
6	0,008	549	11830	4735	5687
8	0,014	1164	28245	10432	12875
10	0,227	33839	1099215	488866	557059
12	1,147	112334	7905860	2727862	3672403

Aqui podemos ver como se comporta o programa quando apresentado com tabuleiros de diferentes dimensões,e como podemos observar apenas se começam a notar um número significativo de alterações no tempo de execução e *backtracks* a partir dos tamanho número 10.

6 Instruções de Utilização

Para utilizar o software desenvolvido, este deve ser compilado através do ficheiro *fourWinds.pl*, depois de compilado pode-se então proceder á execução de qualquer um dos predicados de teste.

Estes devem ser chamados como *test(+Num)* definido no ficheiro *test.pl*. O argumento Num por sua vez corresponde a um dos teste preparados neste ficheiro para verificar a execução do programa. Os testes preparados pretendem abranger um diverso numero de tabuleiros com diferentes características daí o valor *+Num* corresponder a um tamanho de tabuleiro. Por exemplo *test(6)* irá testar um board de tamanho 6.

7 Conclusões e Trabalho Futuro

A elaboração deste projeto permitiu que aplicássemos os nosso conhecimentos de Prolog com Restrições, de uma forma orientada a um projeto real, o que provocou que fossem exploradas algumas interações entre o conhecimento obtido nas aulas práticas e a utilização do *Prolog* na resolução de um puzzle que não seriamos capazes de obter caso não tivéssemos elaborado este projeto. Foi possível observar como o *Prolog com Restrições* permite resolver problemas complexos com um número de linhas de código relativamente reduzido e com uma aproximação ao problema apresentado muito próxima do raciocínio do programador.

Tendo em conta os resultados obtidos, é possível afirmar que, de uma forma geral, o programa tem tempos de execução rápidos. No entanto verificamos variâncias que não conseguimos identificar especificamente mas que acreditamos que se prendem com o número de soluções possíveis.

Por fim acreditamos também que o número de células numéricas é algo que influencia a velocidade de execução pois quando estas se encontram em maior número é possível ter um maior número de restrições no programa logo aumentando a sua eficiência.

Conseguimos ver a utilidade do *Prolog e Prolog com Restrições* na resolução de problemas como o apresentado, e vamos ter em conta os conhecimentos adquiridos nesta unidade quando nos cruzarmos com problemas semelhantes.

Referências

1. SWI-Prolog, http://www.swi-prolog.org
2. SICStus-Prolog, https://sicstus.sics.se

Anexo

Código

O código fonte encontra-se submetido com este projeto no diretório ***code***.