Resolução de Problema de Decisão usando Programação em Lógica com Restrições: Dominosweeper

Gustave Sena (up201704951) e Maria Baía (up201704951)

[1] FEUP-PLOG, Turma 3MIEIC06, Grupo Dominosweeper_2

[2] Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias, 4200-465 PORTO

Resumo: No âmbito da unidade curricular de Programação em Lógica, foi-nos proposto o desenvolvimento de um problema de decisão. O problema consiste num puzzle, ao qual foi escolhido o "Dominosweeper", sendo este uma variante do jogo "Minesweeper", onde as minas vêm aos pares. Assim, através do uso de Programação em Lógica por Restrições e utilizando o Sistema de Desenvolvimento SICStus Prolog, foi possível chegar a uma resolução do problema, ao qual será detalhadamente explicada a partir deste artigo.

Keywords: Dominosweeper, Restrições, Prolog, SICStus, Restrições, FEUP

1 Introdução

De um conjunto de problemas propostos, foi escolhido o tema "Dominosweeper", sendo este bastante intuitivo. Assim, este artigo terá como objetivo apresentar o problema de decisão escolhido, desenvolvido em Programação em Lógica. Após uma análise intensa, fomos capazes de resolver puzzles de diversos padrões de quadrados, e foi também criado um gerador de Puzzles.

Este artigo irá dividir-se em 4 grandes partes, ao qual em cada será feita uma descrição aprofundada do respetivo tópico.

Iniciaremos por descrever com detalhe o problema de decisão estudado e as suas respetivas regras, que nos irão conduzir a uma explicação da abordagem utilizada para com as restrições envolvidas de forma a chegar à resolução desse. De seguida iremos a expor as abordagens feitas ao problema, que irão englobar uma apresentação das variáveis de decisão escolhidas e os seus respetivos domínios, qual a abordagem para restringir essas variáveis e qual a estratégia de pesquisa utilizada de forma a chegar a

soluções concretas. Por fim, será feita uma explicação dos predicados utilizados que permitiram a visualização do problema em modo de texto.

O artigo irá finalizar com uma breve conclusão retirada através da elaboração deste projeto.

2 Descrição do Problema

Dominosweeper é um problema de decisão, que consiste num puzzle jogado por um único jogador. A zona de jogo consiste num tabuleiro quadrado de diversos padrões, em que cada padrão altera consoante o conteúdo de cada célula pertencente ao tabuleiro. Para cada puzzle há um conjunto de regras a serem seguidas:

- Cada mina ocupa uma célula e cada célula tem no máximo uma mina.
- Cada célula que contém uma pista não pode ter uma mina, sendo que essa pista consiste num número entre 0 e 8, ao qual esse indica o número possível de minas que cercam a respetiva célula.
- Cada mina é adjacente a exatamente uma outra mina.

O objetivo deste puzzle é encontrar todas as células não contenham uma mina, caso contrário, o jogo acaba.

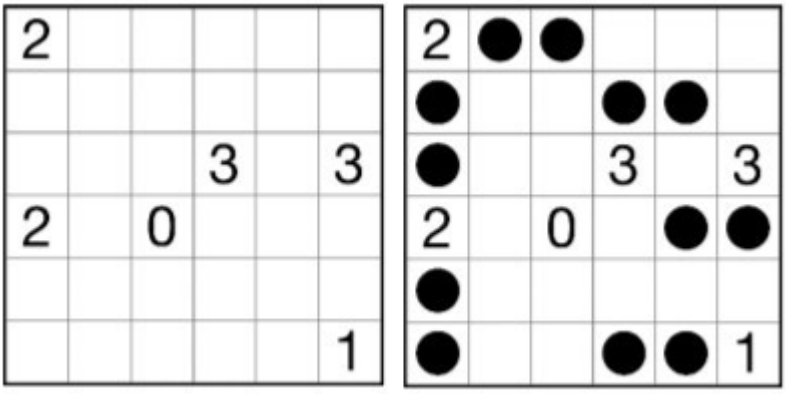


Fig. 1. Exemplo do problema dominosweeper

3 Abordagem

3.1 Variáveis de Decisão

Para a resolução deste problema começou-se por criar uma lista de tamanho NxN, em que cada elemento da lista representa uma célula do tabuleiro de jogo. Essa lista contêm dois tipos de células:

- Células já instanciadas que contêm um número que representam uma pista para a resolução do jogo.
- Células não instanciadas, sendo essas as nossas variáveis de decisão, com um domínio entre 9 e 10, sendo que o valor 9 identifica uma mina e o 10 representa um espaço livre no tabuleiro

De modo a facilitar a interação com cada célula do tabuleiro, converteu-se a lista para uma matriz. Podemos assim exemplificar, um tabuleiro completo 6x6, contendo uma lista composta por N listas com N elementos, em que que cada elemento representado por 'e' virá a ser uma variável de decisão dando-lhe o respetivo domínio.

```
[ 2,  e,  e,  e,  e,  e,
  e,  e,  e,  e,  e,  e,
  e,  e,  e,  3,  e,  3,
  2,  e,  0,  e,  e,  e,
  e,  e,  e,  e,  e,  e,
  e,  e,  e,  e,  e,  1 ]
```

Fig. 2. Representação do tabuleiro em lista

3.2 Restrições

De seguida, de modo a restringir todas as variáveis de decisão existentes, podendo chegar a uma resolução do puzzle, foram aplicadas restrições que se traduzem pelas regras já mencionadas específicas do puzzle.

As restrições foram aplicadas através do predicado ***restrictions*** que, para cada célula do tabuleiro, altera-a e/ou às células ao seu redor, consoante o seu estado e as respetivas restrições a aplicar.

Foram aplicadas dois tipos de restrições: para o caso de o elemento da célula selecionada ser uma pista, ou para o caso de ser uma variável que possa vir a ser uma mina ou um espaço vazio.

No caso de o elemento ser uma pista, ou seja, ser uma variável instanciada, todos os valores encontrados ao seu redor são guardados e, de seguida, verifica-se se o número de valores encontrados com o valor 9, correspondente a uma mina, é exatamente igual ao valor da pista dada.

Para o caso em que o elemento possa assumir os valores 9 e 10, o valor das células adjacentes a esse é verificado, sem contar com as diagonais, e caso o elemento selecionado tenha o valor de uma mina, deverá ser adjacente a exatamente a outra mina. Caso não seja, ocupará um espaço vazio.

4 Visualização

A representação interna do estado do jogo foi feita a partir de uma lista composta por N listas com N elementos. Ao decorrer do jogo cada elemento pode assumir valores diferentes. No caso de um elemento ainda não ser sido desvendado, assume o valor ‘O’. Caso contrário, poderá assumir o valor 9, correspondente a uma mina, o valor 10 sendo um espaço vazio, ou a uma pista, dando o respetivo valor desse mesmo.

No entanto, de modo a facilitar a vizualização das peças do jogo ao utilizador, para representar cada, peça usamos os seguintes átomos:

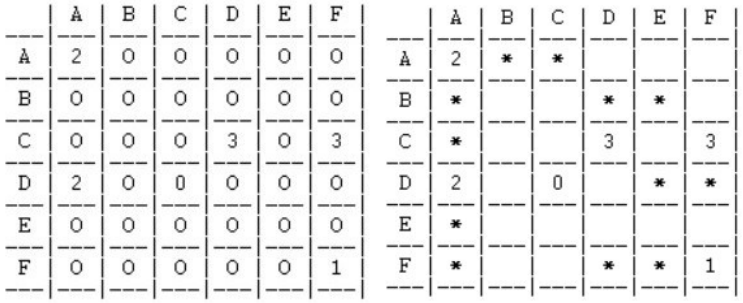
Atom	9	10	e	Track
Meaning	*	‘ ‘	‘O’	Track

Para a visualização do jogo, foi implementado o predicado **displayBoard(+Board)** que recebe uma matriz com o estado do jogo atual. De seguida, de forma a permitir um melhor alinhamento e visualização do tabuleiro, imprimiu-se este a partir dos seguintes predicados:

- **printHead(+Size):** Imprime o cabeçalho do tabuleiro, ordenado desde a letra A até à letra correspondente ao tamanho do tabuleiro.
- **printBar(+Size):** Imprime uma barra de separação entre as linhas do tabuleiro.
- **printBoard(+Size, +N, +Board):** Recursivamente imprime o estado do tabuleiro recebido, com auxílio de 2 predicados: **printLine(+X),** que imprime uma linha do tabuleiro, e novamente o predicado **printBar(+Size)**.

- **symbol(Atom, Meaning):** Consoante o átomo recebido, devolve o respetivo significado.

Desta forma, conseguimos chegar a um tabuleiro intuitivo para o jogador:



	A	B	C	D	E	F
A	2	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	3	0	3
D	2	0	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	1

	A	B	C	D	E	F
A	2	*	*			
B	*			*	*	
C	*			3		3
D	2		0		*	*
E	*					
F	*			*	*	1

Fig. 3. Visualização do tabuleiro

5 Experiências e Resultados

5.1 Análise Dimensional

Para o nosso problema, tendo em conta a variação dos tamanhos do tabuleiro, este seria uma aspeto a considerer que poderia alterar o tempo de execução consoante o seu tamanho. Para tentar perceber qual a influência que a dimensão do tabuleiro traz ao problema, fizemos um conjunto de testes, em que, para o controlo da ordem de seleção de variáveis e valores, foi utilizada uma configuração padrão com ordenação de variáveis estática. Os testes foram feitos para 4 tamanhos de tabuleiros já pré-definidos.

Dimension	Time	Resumptions	Entailments	Prunings	Backtracks	Constrains
3x3	0.0s	366	203	301	2	219
4x4	0.0s	700	337	520	5	356
5x5	0.0s	933	445	753	6	472
6x6	0.0s	2024	1359	1648	9	875

Fig. 4. Tabela de dados estatísticos

Com estes testes concluímos que a variação do tamanho dos tabuleiros não influencia de forma muito significativa a eficiência da solução, sendo o cálculo quase instantâneo. No entanto, relativamento aos outros parâmetros, podemos verificar um ligeiro acréscimo dos valores com o aumento do tamanho dos tabuleiros.

5.2 Estratégia de Pesquisa

De forma a testar diferentes estratégias de pesquisa, configuramos o argumento *Option* do predicado *labeling*, controlando a ordem de seleção de variáveis e valores, e tentamos diferentes abordagens e combinações para aumentar a eficiência do programa.

Começamos por alterar estratégias tanto para a forma de ordenação de variáveis, como para a forma de seleção e a ordenação de valores, sendo que não houve nenhuma estratégia que marca-se significativamente uma melhoria nos parâmetros de avaliação.

Porém, a combinação que mais se destacou foi a [ff, median, down], na qual foi possivel verificar uma melhoria apenas no tabuleiro de maior tamanho.

A conjunção de estratégias usadas consiste na estratégia ff, que por comparacão à estratégia leftmost, usada por omissão, não só escolhe a variável mais à esquerda, mas também de entre as que têm o menor domínio. Para alterar a estratégia de seleção de valor, foi utilizada a estratégia median que faz com que para cada variável, a escolha seja feita através da mediana do domínio, e por fm a estratégia de ordenação de valores down, que explora por ordem decrescente o domínio das variáveis.

Dimension	Time	Resumptions	Entailments	Prunings	Backtracks	Constrains
3x3	0.0s	366	203	301	2	219
4x4	0.0s	700	337	520	5	356
5x5	0.0s	933	445	753	6	472
6x6	0.0s	1533	881	1291	7	875

Fig. 5. Tabela de dados estatísticos alternando a estratégia de pesquisa

6 Conclusão e Trabalho Futuro

O desenvolvimento deste projeto permitiu-nos retirar a importância e a utilidade do uso de restrições para a resolução de problemas de decisão e otimização, sendo uma forma mais intuitiva e direta de abordar um dado problema.

No entanto, há partes do nosso projeto que poderiam vir a ser melhoradas, nomeadamente a parte da geração de tabuleiros aleatórios. A geração foi feita baseada nas restrições colocadas para a resolução do problema, só que em vez de as pistas serem variáveis instanciadas, passavam a ser as variáveis de decisão a encontrar, e as minas já estariam colocadas no tabuleiro de forma aleatória. Uma vez o ponto de referência para gerar um tabuleiro ser apenas as posições geradas aleatoriamente das minas, verificamos que o puzzle gerado consoante as posições das minas poderá ter mais que uma solução, não sendo esta a situação ideal. Não conseguimos encontrar uma forma genérica de garantir unicidade de solução dos puzzles criados, o que seria um ponto fundamental a melhorar. Para além disso, verificamos que na geração de tabuleiros de grandes dimensões, dado um determinado tabuleiro de minas gerado, tanto o tempo de execução como o número de retrocessos podem aumentar drasticamente até encontrar uma solução, o que por vezes nem é possível para o tabuleiro com as dadas minas.

Ao longo do desenvolvimento deste trabalho encontramos algumas dificuldades, nomeadamente na forma devida e eficaz da colocação de restrições. Contudo, todas as dificuldades foram ultrapassadas e o projeto foi finalizado com sucesso e com todos os parâmetros pedidos concluídos, conseguindo chegar a uma solução correta para o problema dado.

7 Referências

1. https://logicmastersindia.com/lmitests/dl.asp?attachmentid=790&view=1
2. LNCS Homepage, http://www.springer.com/lncs. Last accessed 4 Oct 2017
3. https://sicstus.sics.se/sicstus/docs/latest4/html/sicstus.html/
4. Slides da disciplina de PLR