Hoo-Doo Solver

Daniel Mendonça and José Pedro Moreira

FEUP-PLOG, Turma 3MIEIC9, Grupo 123

Abstract. Este projecto consiste na implementação de um *solver* para o jogo de tabuleiro *Hoo-Doo*. O solver funciona para uma dimensão arbitrária do tabuleiro. A implementação foi feita usando Prolog, mas concretamente a plataforma *Sicstus Prolog* tendo sido usados para tal os módulos desta mesma ferramenta para Pragramação em Lógica com Restrições sobre domínios finitos.

1 Introdução

A realização deste trabalho teve como um dos objectivos, perceber a importância, potêncialidades e utilidade da Programação em Lógica com restrições. Outro dos objectivos foi o uso e conhecimento de bibliotecas do SICStus, que auxiliam na resolução dos problemas existentes e o alcance dos seus predicados, que, por vezes, mesmo não estando directamente relacionados com um problema em concreto, sendo interpretados de formas alternativas se tornam bastante úteis, quer na simplificação do problema, quer na procura de uma resolução mais eficiente. O Hoo-Doo, dependendo da sua dimensão, pode ter várias, uma ou até nenhuma resolução se não forem usadas *pegs* transparentes(peg é uma peça de cor, será explicado em detalhe na descrição do jogo). Quando foi lançado o jogo de tabuleiro Hoo-doo, os seus criadores ofereciam 1000$ à primeira pessoa que conseguisse resolver um tabuleiro de 8x8 sem recurso a *pegs* transparentes, e, os elementos do grupo acharam que seria interessante e até certo ponto divertido verificar como a implementação deste jogo em Prolog, usando restrições seria uma mais valia para vencer o prémio que era na altura oferecido. A pouca informação encontrada sobre este jogo de tabuleiro também despertou curiosidade. . . . O trabalho desenvolvido pelos elementos do grupo tem trs partes distintas:

– Parte de visualização do jogo, em muito semelhante ao já desenvolvido num outra trabalho para a disciplina.
– Parte do *solver*, que resolve o tabuleiro que lhe é passado
– Parte de conversão de tabuleiro *flat* para tabuleiro bi-dimensional e vice-versa.

Na implementação do jogo é suportado o funcionamento em dois modos, um que não usa *pegs* transparentes, e outro que usa, tentando no entanto minimizar a sua utilização.

2 Descrição

O jogo Hoo-Doo foi criado pela empresa(?) Tryne Games, lançado na década de 50. Hoo-Doo é um jogo de tabuleiro, para um único jogador, normalmente quadrado e que tem pelo menos tantas cores quanto o número de colunas no tabuleiro, e o numero de pegs de cada cor também é o numero de colunas do tabuleiro. Os tamanhos de tabuleiro mais frequentes são os de 4x4, 6x6 e 8x8. O jogo tem como início um tabuleiro vazio, e o objectivo é preencher todas as posições do tabuleiro com as *pegs* disponíveis, sem nunca repetir peças da mesma cor na mesma linha, coluna, ou qualquer uma das diagonais. Para auxiliar na resolução do tabuleiro, existem as denominadas *pegs* transparentes, cuja característica é preencher uma posição sem lhe atribuir uma cor. O uso de *pegs* transparentes é absolutamente necessário para a resolução de tabuleiros com determinados tamanhos, cuja resolução é possível apenas com o uso de *pegs* transparentes(como exemplo temos um tabuleiro de 6x6, que é impossível resolver mesmo com duas *pegs* transparentes). É sempre considerada como a melhor resolução, aquela que usar menos *pegs* transparentes.

3 Visualização

Existem seis predicados utilizados para a construção visual do tabuleiro em modo de texto. O primeiro predicado a ser executado é o *print_tab(+board)* que recebe como argumento um tabuleiro representado por uma lista de listas. Este predicado calcula o comprimento da lista, que determina o número de linhas, colunas e respectivos índices a serem imprimidos, e de seguida passa-os como argumentos para as funções auxiliares que controlam a impressão, descritas em baixo.

- *print_tab_aux(+Board,?LineI,?ColumnI)*: coordena a utilização dos seguintes predicados para a construção visual do tabuleiro.
- *tab_map(+Symb)*: Imprime o número ou correspondente no tabuleiro.
- *print_line(+Line)*: imprime uma linha do tabuleiro, fazendo uso do tab_map(+Symb) para a impressão numérica.
- *print_empty_line(+Length)*: imprime uma linha horizontal.
- *print_column_index(+ASCIICode,+Index)*: imprime o índice das colunas.

4 Variáveis de Decisão

Para modelar o problema em causa, usamos uma variável por cada posição no tabuleiro, perfazendo para um tabuleiro $n*m$, $n*m$ variáveis.O domínio dessas variáveis depende também ele do tamanha do tabuleiro, sendo que existem para cada tipo de tabuleiro dois domínios distintos dependendo de o jogo ser resolvido usando peças transparentes ou não. Para o caso de serem usadas peças transparentes o domínio de cada uma das variáveis é $[0, n]$. No caso de se recorrer somente a peças com cor o domínio de cada uma das variáveis passa a ser $[1, n]$.

No caso de o jogo ser resolvido recorrendo aos *pegs* transparentes é ainda utilizada uma variável extra, que é a soma do valor de todas as outras. Na resoluão do jogo tentar-se-á minimizar o valor desta variável que tem um domínio que é $[0, n^3]$ num tabuleiro de $n * m$ onde n é a maior dimensão.

5 Restrições

Tratando-se de um jogo todas as restrições que incluímos são rígidas, á excepção de uma. As restrições rígidas são:

- A restrição que impede que duas peças da mesma linha tenham valores iguais
- A restrição que impede que duas peças da mesma coluna tenham valores iguais
- A restrição que impede que duas peças da mesma diagonal tenham valores iguais

A restrição flexível, diz respeito á tentativa de minimizar o número de peças transparentes envolvidas na resolução do tabuleiro (que é por si também um dos objectivos do jogo).

A implementação das restrições rígidas é feita agrupando as variáveis em listas, ora por linha, ora por coluna, ora por diagonal , e chamando o predicado all_distinct sobre essas listas, obrigando assim que nunca hajam duas peças que partilhem a mesma linha/coluna/diagonal que tenham o mesmo valor.

A implementação da restrição flexível, é feita pela introdução de uma variável antes do *labeling*, a qual se passa ao predicado sum para que represente a soma do valor de todas as variáveis do tabuleiro. Após a aplicação desta restrição o *labeling* é chamado dentro do *min*, permitindo assim obter a solução cuja soma dos valores das peças é maior, valor esse que necessáriamente inclui o menor número possível de peças transparente (peças com o valor zero).

6 Estratégia de Pesquisa

A estratégia utilizada para a etiquetarem das variáveis foi tentar atribuir valores primeiro, aquelas que estão mais restritas por forma a tentar minimizar a árvore de pesquisa, "cortando" nós o mais cedo possível. Isto é conseguido usando a opção *ffc* que tentar atribuir valores primeiro as variáveis com um domínio menor, sendo que o critério de desempate seleciona a variável com mais restrições suspensas.