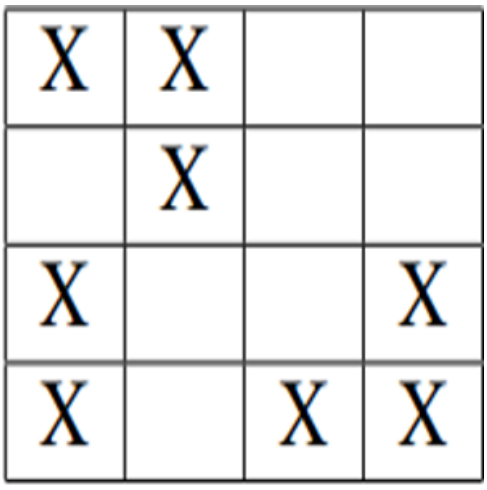
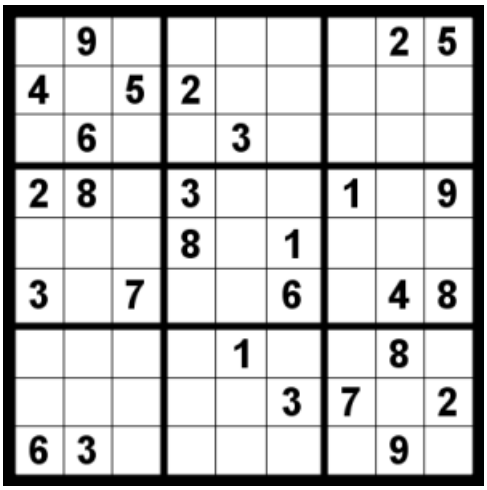
Universidade de Évora

Disciplina de Inteligência Artificial

Problema 2Xn peças e Sudoku





Autores:
Marcus Santos, 29764
Ricardo Fusco, 29263

Professor:
Irene Pimenta Rodrigues

28/03/2014

Índice

1 Respostas às perguntas do enunciado **3**
1.1 Problema 2Xn peças 3
1.1.1 Definição do problema (Estados, variáveis e restrições) 3
1.2 Problema Sudoku 5
1.2.1 Definição do problema (Estados, variáveis e restrições) 5

Lista de Figuras

1 Estado inicial 3
2 Restricoes 3
3 Solução 1 4
4 Solução 2 4
5 Solução 3 4
6 Solução 4 5
7 Estado inicial 5
8 Gerador de estado 6
9 Sudoku - solução 1 7
10 Sudoku - solução 2 7
11 Sudoku - solução 3 8
12 Sudoku - solução 4 8

1 Respostas às perguntas do enunciado

1.1 Problema 2Xn peças

1.1.1 Definição do problema (Estados, variáveis e restrições)

Para representar o problema através do uso de variáveis como um problema de satisfacção de restrições estabelecemos que uma variável tem 3 elementos, o primeiro é a posição p(X,Y) do tabuleiro, o segundo é a lista com o domínio que definimos como [0,1] e o terceiro é o valor do domínio que irá ser instanciado. Se o valor duma variável for 1 significa que nessa posição se encontra um X, se for 0 não tem nada.

Um estado irá ser representado da seguinte maneira (estado inicial):

```
(4 ms) yes
| ?- estado_inicial(E).

E = e([v(p(1,1),[0,1],_),v(p(2,1),[0,1],_),v(p(3,1),[0,1],_),v(p(4,1
),[0,1],_),v(p(1,2),[0,1],_),v(p(2,2),[0,1],_),v(p(3,2),[0,1],_),v(p
(4,2),[0,1],_),v(p(1,3),[0,1],_),v(p(2,3),[0,1],_),v(p(3,3),[0,1],_)
,v(p(4,3),[0,1],_),v(p(1,4),[0,1],_),v(p(2,4),[0,1],_),v(p(3,4),[0,1
],_),v(p(4,4),[0,1],_)],[]) ?
```

Figure 1: Estado inicial

Na figura anterior pode-se observar como está representado um estado, nesta figura as variáveis estão ainda todas por instanciar. Para o problema ser mais geral fizemos um gerador do estado inicial dado um tamanho_tabuleiro(D), em que D representa as dimensões do tabuleiro, por exemplo, se D = 4 significa que o tabuleiro é 4x4. No que diz respeito às restrições temos um ver_linhas e um ver_colunas que basicamente vai iterando pelas linhas/colunas e incrementando o número de X's seguidos na mesma linha/coluna de acordo com o valor que está na variável (0 ou 1).

```
ve_restricoes(e(_NAssign, Assigned)):-
        ver_linhas(Assigned), ver_colunas(Assigned).

% ------------------------------------------------
ver_colunas(L):-
        tamanho_tabuleiro(S),
        ver_colunas(L, 1, S).
ver_colunas(L, S, S):-
        ver_coluna(L, S).
ver_colunas(L, J, S):-
        J < S,
        ver_coluna(L, J),
        J2 is J+1,
        ver_colunas(L, J2, S).
% ------------------------------------------------
ver_linhas(L):-
        tamanho_tabuleiro(S),
        ver_linhas(L, 1, S).
ver_linhas(L, S, S):-
        ver_linha(L, S).
ver_linhas(L, I, S):-
        I < S,
        ver_linha(L, I),
        I2 is I+1,
        ver_linhas(L, I2, S).
```

Figure 2: Restricoes

O operador sucessor é o mesmo que foi utilizado para o problema das 8 rainhas: sucessor(e([p(N,D,V)|R],E),e(R,[p(N,D,V)|E])):- member(V,D).

O sucessor indica que um valor pode ser atribuído a qualquer variável não instanciada, desde que não entre em conflito com as restrições pré estabelecidas, ou seja basicamente instancia uma variável e coloca-a nos Afect, a lista com as variáveis já instanciadas.

O problema é resolvido facilmente usando a pesquisa backtracking. Nas figuras seguintes (3,4,5 e 6) podem-se observar algumas soluções que obtemos após a resolução do problema para uma tabuleiro 4x4.

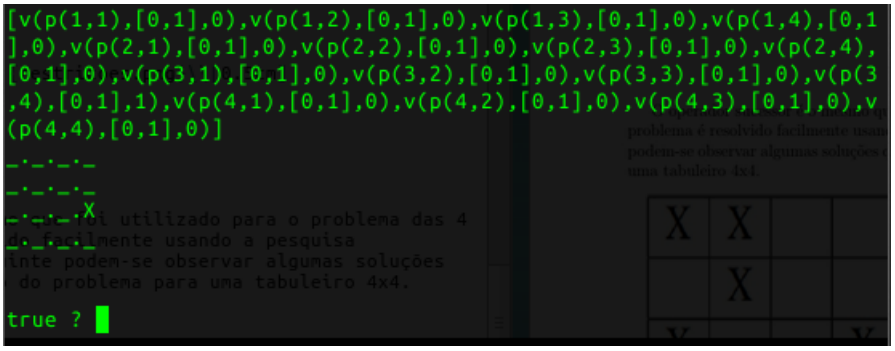


Figure 3: Solução 1

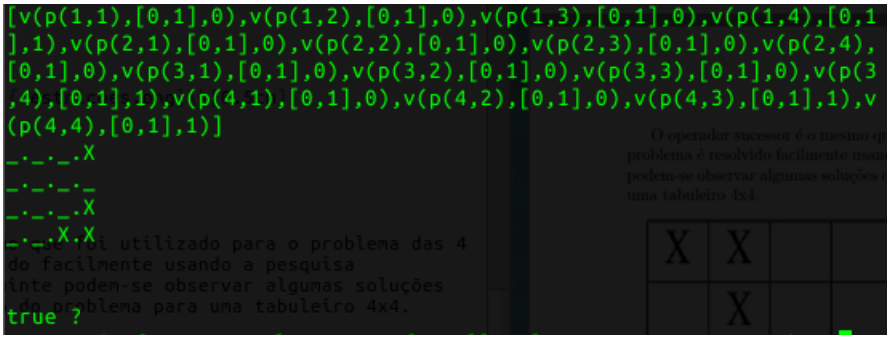


Figure 4: Solução 2

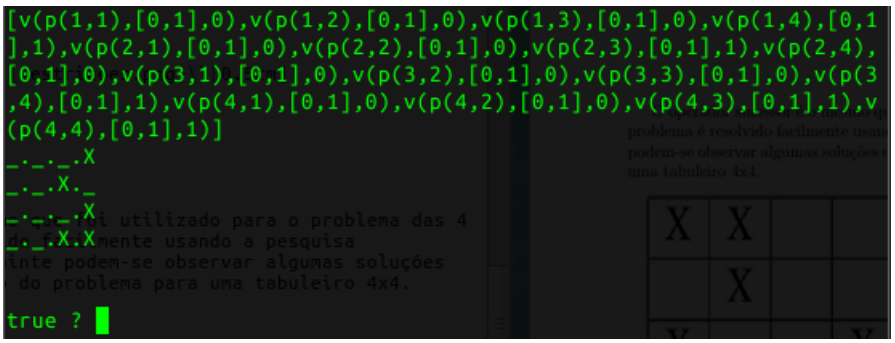


Figure 5: Solução 3

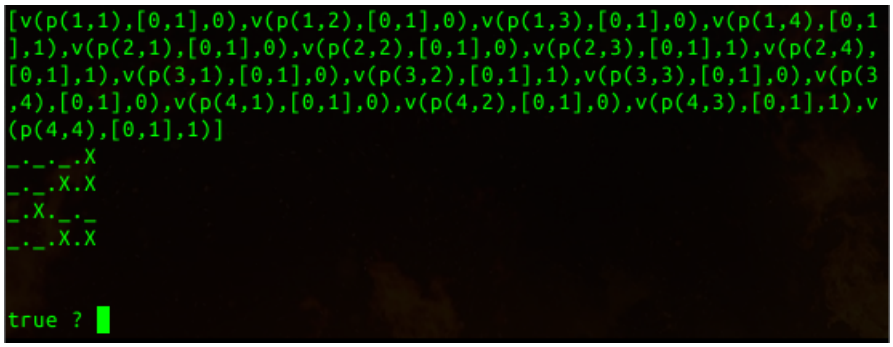


Figure 6: Solução 4

A primeira solução que mostra é a solução com todas as variáveis instanciadas com 0 ou seja o tabuleiro sem cruzes.

Relativamente ao forward checking começamos a pensar numa forma de o fazer mas não chegamos a acabar, como tal, este algoritmo não funciona se for testado.

1.2 Problema Sudoku

1.2.1 Definição do problema (Estados, variáveis e restrições)

Para resolver este problema como um problema de satisfacção de restrições usando variáveis decidimos que uma variável é composta por 3 elementos, à semelhança do problema das cruzes, o primeiro é a posição p(X,Y) do tabuleiro, o segundo é a lista com o domínio que vai de 1 a 9 e o terceiro é o valor do domínio que determinada variável irá assumir quando for instanciada. Para gerar o estado inicial usámos os mesmos predicados usados para gerar o tabuleiro para o problema das cruzes sendo apenas necessário mudar o dominio com que cada variável ia ser criada de [0,1] para [1,2,3,4,5,6,7,8,9] visto que o tabuleiro do sudoku é, por convenção, 9x9. O tamanho_tabuleiro foi alterado de 4 para 9. Na figura seguinte pode-se observar como é representado um estado tomando como exemplo parte do estado inicial.

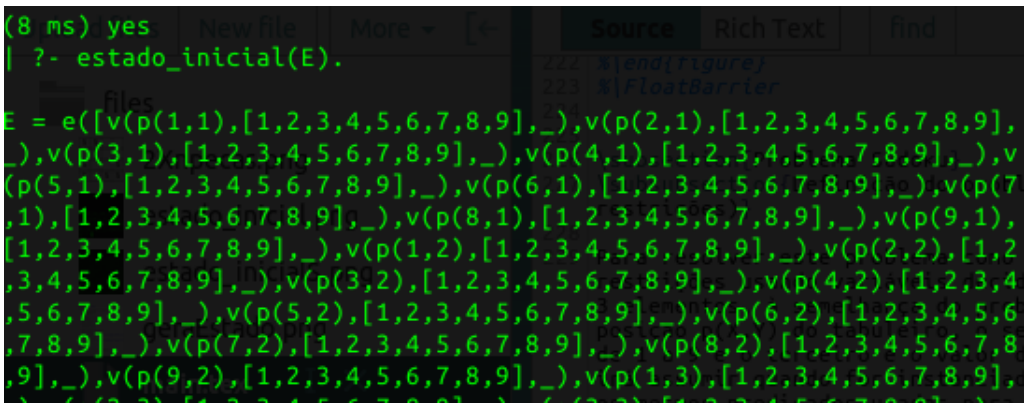


Figure 7: Estado inicial

Como já foi referido anteriormente o estado inicial é gerado usando os mesmos predicados utilizados para o problema das cruzes:

```
tamanho_tabuleiro(9). %(9x9)

estado_inicial(E):-
        functor(E, e, 2), arg(1, E, T), arg(2, E, []),
        tamanho_tabuleiro(S),
        gerar_tab(T2, S),
        flatten(T2, T).

gerar_tab(L, Size):-
        gerar_tab(L, 1, Size).
gerar_tab([H], N, N):-
        gerar_coluna(H, N, N).
gerar_tab([H|T], J, N):-
        J < N, J2 is J+1,
        gerar_coluna(H, J, N),
        gerar_tab(T, J2, N).

gerar_coluna(L, J, NColunas):-
        gerar_coluna(L, 1, J, NColunas).
gerar_coluna([H], N, J, N):-
        tamanho_tabuleiro(S),
        range(1, S, D), %cria o dominio da variavel dado um maximo como arg
        H = v(p(N, J), D, _).
gerar_coluna([H|T], I, J, N):-
        tamanho_tabuleiro(S),
        range(1, S, D), %cria o dominio da variavel dado um maximo como arg
        I < N, I2 is I+1,
        H = v(p(I, J), D, _),
        gerar_coluna(T, I2, J, N).

%devolve uma lista com numeros de I a J
range(I,I,[I]).
range(I,J,[I|L]) :- I < J, I1 is I + 1, range(I1,J,L).
```

Figure 8: Gerador de estado

Para que possa ser resolvido o problema do sudoku para os casos em que no estado_inical há posições que já estão preenchidas, ou seja, com variáveis já instanciadas basta fazer uma pequena alteração no gerar_tab para que crie as variáveis não instanciadas menos as que queremos que estejam instanciadas no estado inicial, ou então outra maneira seria gerar o tabuleiro normalmente com o gerador e no fim pegar nas variáveis da lista NAfect e instanciar as variáveis que queremos que o estado inicial tenha e passá-las para os Afect. Para instanciar variáveis no estado_inicial foi feito o predicado preencher_posicoes que devolve o estado com posições do tabuleiro já preenchidas.

No que diz respeito às restrições fizemos um ver_linhas, ver_colunas e ver_quadrantes. No ver_linhas fizemos um findall do valor de todas as variáveis para uma determinada linha e colocámos o X da posição a variar e verificamos se os valores na lista que o findall devolveu são todos diferentes, o mesmo para a coluna a variar o Y. Para o ver_quadrantes vemos os 9 quadrantes veriando o X e o Y, por exemplo o ver_quadrante recebe como argumento a lista dos Afect, o valor X de onde começa o quadrante, o valor Y onde começa o quadrante e o valor Y onde acaba o quadrante e o valor X varia sempre até X+2.

Nas figuras seguintes pode-se observar algumas das soluções obtidas para este problema.

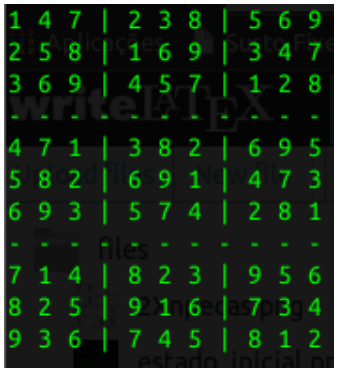


Figure 9: Sudoku - solução 1

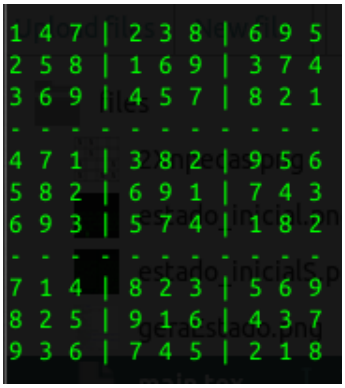


Figure 10: Sudoku - solução 2

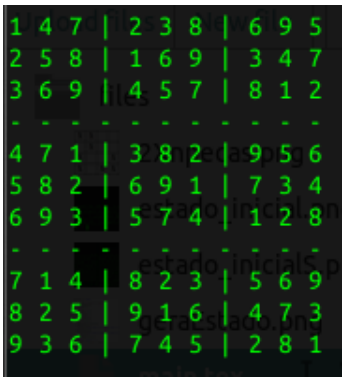


Figure 11: Sudoku - solução 3

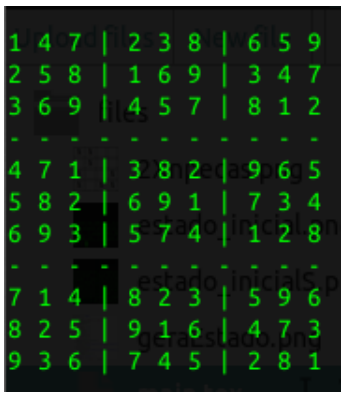


Figure 12: Sudoku - solução 4

No que diz respeito ao forward checking não conseguimos acabá-lo mas pelos testes que fizemos o predicado cut_linha e cut_coluna devolviam o novo estado com os domínios já restringidos, o problema foi depois quando testamos o forwardchecking na pesquisa backtrack não resultou e devolveu No. O que não fizemos mesmo foi restringir o domínio nos quadrados. O operador sucessor é igual ao que foi usado no problema anterior.