UNIVERSIDADE DE ÉVORA

Inteligência Artificial
2015/2016

Problema Quadrado Mágico e Sudoku

Docente:

Irene Pimenta Rodrigues

Discentes:

Pedro Jacob - 27421
André Figueira - 31626

1. Respostas as perguntas do Enunciado

1.1 Problema do Quadrado Mágico

1.1.1 Pergunta 1

1.1.1.1 As variáveis

Foi estabelecido neste problema que as variáveis tem 2 componentes.

O primeiro componente corresponde à posição da variável no tabuleiro, e o segundo corresponde ao valor por instanciar.

v(n(X,Y),V)

X => valor do eixo do X
Y => valor do eixo do Y
V => valor

1.1.1.2 Os estados

Os estados estão divididos em três. A lista de variáveis não afetadas e a lista de variareis afetadas e domínio.

Exemplo de parte do estado inicial (para 3x3) :

```
| ?- estado_inicial(E0,3).

E0 = e([v(n(0,0),_),v(n(1,0),_),v(n(2,0),_),v(n(0,1),_),v(n(1,1),_),v(n(2,1),_),v(n(0,2),_),v(n(1,2),_),v(n(2,2),_)],[],[1,2,3,4,5,6,7,8,9])
```

Tendo em conta a natureza do problema em questão, decidimos que o domínio das variáveis deveria ser global, visto que qualquer instanciação de uma variável afecta directamente o domínio de todas por instanciar.

1.1.1.3 O estado inicial

```
estado_inicial(e(Vars, [], Domain),SideSize) :-
    gerarVars(Vars, SideSize),
    MaxD is SideSize*SideSize,
    findall(D, between(1, MaxD, D), Domain),!.

gerarVars(Vars, SideSize) :-
    gerarVars(Vars, SideSize, 0).

gerarVars([], SideSize, M) :-
    M is SideSize * SideSize.

gerarVars([v(n(CordX, CordY), _) | T], SideSize, Count) :-
    CordX is Count mod SideSize,
    CordY is Count // SideSize,
    Count2 is Count + 1,
    gerarVars(T, SideSize, Count2).
```

O estado inicial usa o predicado gerarVars, que permite a criação dum tabuleiro com as variáveis com os valores de coordenadas corretas dado como argumento o tamanho do lado do quadrado.

O Predicado faz divisões inteira de maneira a calcular o valor das coordenadas, incrementado o contador e chamando o resto da lista até todas as coordenadas estarem instanciadas.

1.1.1.4 O operador sucessor

Sem Forward Checking:

```
sucessor(e([v(N,V)|R],E,D),e(R,[v(N,V)|E],D)):- member(V,D).
```

Com Forward Checking:

```
sucessor2(e([v(n(CoordX,CoordY),V)|R],E,D),e(R,[v(n(CoordX,CoordY),V)|E],NewDom)):- member(V,D), subtract(D,[V],NewDom).
```

1.1.1.5 Restrições

```
ve_restricoes(E):-
    tamanho_linha(Linha),
    soma(Soma),
    ver_quadrado(E),
    ver_linhas(E,T,M),
    ver_colunas(E,T2,M2),
    ver_Diag1(E,T3,M3),
    ver_Diag2(E,T4,M4),

    ( T = Linha -> M = Soma; true),
    ( T2 = Linha -> M2 = Soma; true),
    ( T3 = Linha -> M3 = Soma; true),
    ( T4 = Linha -> M4 = Soma; true).

soma(Soma):- tamanho_linha(Linha), Linha = 3, Soma is 15.     % 3x3
soma(Soma):- tamanho_linha(Linha), Linha = 4, Soma is 34.     % 4x4
soma(Soma):- tamanho_linha(Linha), Linha = 5, Soma is 65.     % 5x5
soma(Soma):- tamanho_linha(Linha), Linha = 6, Soma is 111.    % 6x6
```

As restrições aplicadas são a verificação da soma das linhas, colunas e diagonais. Para tal usa-se Os predicados ver_linhas\3 ver_colunas\3, ver_Diag1\3 e ver_Diag2\3.

Conforme o tamanho do quadrado passado, a soma é verificada com um valor diferente.

Os predicados ver_quadrado\3 ver_colunas\3 e ve_linhas\3:

```
ver_quadrado(e(_,[v(n(X,Y), V)|R],_)) :-
    findall(V1,member(v(n(_,_),V1),R),L), diff([V|L]).

ver_linhas(e(_,[v(n(X,Y), V)|R],_),T,M)    :-
    findall(V1,member(v(n(X,_),V1),R),L), diff([V|L]),
    length([V|L],T), sumList([V|L],M).

ver_colunas(e(_,[v(n(X,Y), V)|R],_),T,M)   :-
    findall(V1,member(v(n(_,Y),V1),R),L), diff([V|L]),
    length([V|L],T), sumList([V|L],M).
```

Estes a excepção do ver_quadrado/3 que só verifica se todos os valores estão diferentes. Os restantes dois retornam o tamanho da lista obtida com findall e o valor da soma dessa mesma lista.
Isto para que, se o tamanho da lista não for igual ao tamanho da linha, por exemplo 3, falha. Se o tamanho se verificar verifica a soma, se não se verificar, falha.
As diagonais funcionam com um processo semelhante.

Os predicados ver_diag1/3 e ver_diag2/3 veem as duas diagonais dos quadrados, usando como auxiliar os predicados run_diag\2 e run_diag2\2, que fazem assertz dos valores das casas que pertencem as coordenadas.

```
ver_Diag2(e(_,[v(n(X,Y), V)|R],_),T,M)     :-
        findall((X1,Y1,V1),member(v(n(X1,Y1),V1),R),L),
        tamanho_quadrado(SquareSize), length([(X, Y, V)|L], SquareSize_2),
        ( SquareSize = SquareSize_2  -> run_diag2([(X, Y, V)|L], Result),diag2(Ls), diff(Ls), length(Ls,T), sum_list(Ls, M); true).

ver_Diag1(e(_,[v(n(X,Y), V)|R],_),T,M)     :-
        findall((X1,Y1,V1),member(v(n(X1,Y1),V1),R),L),
        tamanho_quadrado(SquareSize), length([(X, Y, V)|L], SquareSize_2),
        ( SquareSize = SquareSize_2 -> run_diag([(X, Y, V)|L], Result),diag(Ls), diff(Ls), length(Ls,T), sum_list(Ls, M); true).
```

O run_diag\2 corre o quadrado, verificando se as casas têm coordenadas que pertençam á diagonal “esquerda” em que x e y têm valores iguais. Se sim, guarda esse valor.

O run diag2\2 corre o quadrado também, mas agora procura as variáveis da outra diagonal, em que os valores de x e y são simétricos em relação ao tamanho do quadrado.

```
run_diag([], [V1, V2, V3,V4]) :- retractall(diag(_)), assertz(diag([V1, V2, V3,V4])).

run_diag([(X, Y, V)|L], []):-
    ( X = Y -> run_diag(L, [V]); run_diag(L, [])).

run_diag([(X, Y, V)|L], D) :-
    ( X = Y -> merge([V], D, N), run_diag(L, N); run_diag(L, D)).

run_diag2([], [V1, V2, V3,V4]) :- retractall(diag2(_)), assertz(diag2([V1, V2, V3,V4])).

run_diag2([(X, Y, V)|L], []):-
     tamanho_linha(C),
     F is C-1,
     Z is X+Y,
    ( Z = F -> run_diag2(L, [V]) ; run_diag2(L, [])).

run_diag2([(X, Y, V)|L], D) :-
        tamanho_linha(C),
        F is C-1,
        Z is X+Y,
    ( Z = F -> merge([V], D, N), run_diag2(L, N) ; run_diag2(L, D)).
```

1.1.2 Forward Checking

O forward checking o que faz é restringir o domínio das variáveis. O que faz é, sempre que instancia uma variavél, esse valor é removido do dominio.

```
sucessor2(e([v(n(CoordX,CoordY),V)|R],E,D),e(R,[v(n(CoordX,CoordY),V)|E],NewDom)):- member(V,D), subtract(D,[V],NewDom).
```

```
p2:- tamanho_linha(Linha),    estado_inicial(E0,Linha), fowarding(E0,A), esc(A).

fowarding(e([],A,D),A).
fowarding(E,Sol):- sucessor2(E,E1), ve_restricoes(E1),
                        fowarding(E1,Sol).
```

1.1.3 Complexidade

Uma hipótese para melhorar a complexidade do problema seria a instanciação antes de calcular seja o que for de o valor do meio em quadrados impares, pois o problema em questão causa com que esse valor seja o mesmo para todas as soluções. Em 3x3, por exemplo, esse valor seria 5.

1.1.4 Alguns resultados

Com 3x3

[v(n(0,0),2),v(n(0,1),9),v(n(0,2),4),v(n(1,0),7),v(n(1,1),5),v(n(1,2),3),v(n(2,0),6),v(n(2,1),1),v(n(2,2),8)]

2 9 4
7 5 3
6 1 8

[v(n(0,0),4),v(n(0,1),9),v(n(0,2),2),v(n(1,0),3),v(n(1,1),5),v(n(1,2),7),v(n(2,0),8),v(n(2,1),1),v(n(2,2),6)]

4 9 2
3 5 7
8 1 6

[v(n(0,0),4),v(n(0,1),3),v(n(0,2),8),v(n(1,0),9),v(n(1,1),5),v(n(1,2),1),v(n(2,0),2),v(n(2,1),7),v(n(2,2),6)]

4 3 8
9 5 1
2 7 6

Com 4x4

[v(n(0,0),1),v(n(0,1),12),v(n(0,2),13),v(n(0,3),8),v(n(1,0),2),v(n(1,1),14),v(n(1,2),7),v(n(1,3),11),v(n(2,0),15),v(n(2,1),3),v(n(2,2),10),v(n(2,3),6),v(n(3,0),16),v(n(3,1),5),v(n(3,2),4),v(n(3,3),9)]

1 12 13 8
2 14 7 11
15 3 10 6
16 5 4 9

2. Respostas as perguntas do Enunciado

2.1 Problema do Sudoku

2.1.1 Pergunta 1

2.1.1.1 As variáveis

Foi estabelecido neste problema que as variáveis tem 3 componentes.

O primeiro componente corresponde à posição da variável no tabuleiro,

n(X,Y,Quadrante)

X => coordenada no eixo X
Y => coordenada no eixo Y
Quadrante => quadrante a que corresponde a coordenada. Existem 9 quadrantes

O segundo componente é o domínio. Visto que se trata de um problema de sudoku, existem 9 valores possíveis para cada casa, sendo assim o domínio é

[1,2,3,4,5,6,7,8,9]

O terceiro componente é o valor do domínio. Este valor é o que irá ser instanciado.

Sendo assim o formato de uma variavel tem o seguinte formato:

v(n(X,Y,Quadrante),[1,2,3,4,5,6,7,8,9],_)

2.1.1.2 Os estados

Os estados estão divididos em dois. A lista de variáveis não afetadas e a lista de variareis afetadas.

Exemplo de parte do estado inicial:

```
E = e([v(n(0,0,q1),[1,2,3,4,5,6,7,8,9],_),v(n(2,0,q1),[1,2,3,4,5,6,7,8,9],_),v(n(3,0,q2),[1,2,3,4,5,6,7,8,9],_),v(n(4,0,q2),[1,2,3,4,5,6,7,8,9],_),
n(3,3,q5),[1,2,3,4,5,6,7,8,9],_),v(n(4,3,q5),[1,2,3,4,5,6,7,8,9],_),v(n(5,3,q5),[1,2,3,4,5,6,7,8,9],_),v(n(6,3,q6),[1,2,3,4,5,6,7,8,9],_),v(n(7,3,c
,[1,2,3,4,5,6,7,8,9],_),v(n(7,6,q9),[1,2,3,4,5,6,7,8,9],_),v(n(8,6,q9),[1,2,3,4,5,6,7,8,9],_),v(n(0,7,q7),[1,2,3,4,5,6,7,8,9],_),v(n(2,7,q7),[1,2,3
```

(para sudoku este estado é grande demais portanto foi cortado)

2.1.1.3 O estado inicial

O estado inicial é obtém-se usando os predicados **estadoinicial2(E1) e initializar(E1,E)**.

O predicado estadoinicial2(E1) calcula uma matriz, usando os mesmo predicados que foram usados para gerar a matriz do quadrado mágico com uma diferença. As diferença é associar a cada variável o quadrante correspondente.
Isto é possivél através do predicado **qual_quadrante/3.** Este predicado funciona em combinação com o **Count** e funciona da seguinte maneira:

Cada coluna está associada a um valor como mostra a figura em baixo:

0	1	2	3	4	5	6	7	8

Entre 0 e 2 pode corresponder aos quadrantes 1, 4 e 7.
Entre 3 e 5 pode corresponder aos quadrantes 2, 5 e 8.
Entre 6 e 8 pode corresponder aos quadrantes 3, 6 e 9.

Usando o **Count** sabe-se a cada 9 posições, ou seja, Y = 8, corresponde a um numero múltiplo de 9 (9, 18, 27, ...). À medida que o count vai incrementado uma outra variável vai incrementado também, **ValorPos**, este valor corresponde ao valor da coluna, que faz reset (volta a 0) sempre que Count é multiplo de 9. o que significa que juntamente com o Count podemos saber a qual quadrante corresponde qual coordenada.

```
% verifica em qual conjunto de linhas está
% por exemplo
% < 28 -> entre as 3 primeiras linhas, isto é, está entre o quadrante 1, 2 e 3

qual_quadrante(ValorPos,Counter, Q):-
   ((Counter < 28   -> qual_quadrante2(ValorPos,Q));
    (Counter < 55   -> qual_quadrante3(ValorPos,Q));
    (Counter < 82   -> qual_quadrante4(ValorPos,Q) ; true)).

% verifica qual quadrante pertence a coordenada

qual_quadrante2(Valor,Q):- Valor < 3, Q = q1.
qual_quadrante2(Valor,Q):- Valor < 6, Q = q2.
qual_quadrante2(Valor,Q):- Valor < 9, Q = q3.

qual_quadrante3(Valor,Q):- Valor < 3, Q = q4.
qual_quadrante3(Valor,Q):- Valor < 6, Q = q5.
qual_quadrante3(Valor,Q):- Valor < 9, Q = q6.

qual_quadrante4(Valor,Q):- Valor < 3, Q = q7.
qual_quadrante4(Valor,Q):- Valor < 6, Q = q8.
qual_quadrante4(Valor,Q):- Valor < 9, Q = q9.
```

Como se trata de um problema de sudoku, isto implica já haver casas preenchidas, para resolver o problema do sudoku com casas preenchidas, usou-se o predicado **initializar/2** , que preenche posições. Isto funciona da seguinte maneira. Dado um valor que queremos preencher numa dada casa, cria-se uma nova variável com o formato descrito anteriormente. Por exemplo o valor 9 na casa (1,1) seria algo do gênero:

v(n(1,1,q1),[1,2,3,4,5,6,7,8,9],9)

O que este predicado faz é receber o estado inicial, em que percorre a lista de variáveis não afectadas, e remove a variável **v(n(1,1,q1),[1,2,3,4,5,6,7,8,9],_)** e insere a variável acima na lista de variáveis afectadas.

2.1.1.4 O operador sucessor

Sem usar forward checking

```
sucessor(e([v(N,D,V)|R],E),e(R,[v(N,D,V)|E])):- member(V,D).
```

Usando forward checking

```
sucessor2(e([v(N,D,V)|R],E),e(NovoR,[v(N,D,V)|E])):-
 member(V,D), forward_checking(N,V,R,NovoR).
```

Para lidar com as casas preenchidas, foi um processo simples. Basta instanciar uma variável, como demonstra na figura e a adicionar a cabeça da lista E que contém variáveis instanciadas (inicialmente contem as casas previamente preenchidas).

2.1.1.5 Restrições

As restrições são semelhantes aos do quadrado mágico. A maneira de lidar com as linhas e colunas é feita usando o mesmo processo mas adaptando as variaveis para corresponderem as do sudoku. O ver quadrantes é semelhante ao ver_quadrado do quadrado mágico só que neste caso temos de repetir para todos os quadrantes e verificar individualmente se cada quadrante tem os valores diferentes.

```
ver_q(LI,QualQ,ListQ):-
    findall(V1,member(v(n(X,Y,QualQ),_,V1),LI),ListQ).

ver_quadrantes(e(_,LI)):-
    ver_q(LI,q1,Q1),
    diff(Q1),

    ver_q(LI,q2, Q2),
    diff(Q2),

    ver_q(LI,q3, Q3),
    diff(Q3),

    ver_q(LI,q4, Q4),
    diff(Q4),

    ver_q(LI,q5, Q5),
    diff(Q5),

    ver_q(LI,q6, Q6),
    diff(Q6),

    ver_q(LI,q7, Q7),
    diff(Q7),

    ver_q(LI,q8, Q8),
    diff(Q8),

    ver_q(LI,q9, Q9),
    diff(Q9).
```

2.1.2 Backtracking

Uma solução com backtracking:

```
[v(n(0,0,q1),[1,2,3,4,5,6,7,8,9],1),v(n(0,1,q1),[1,2,3,4,5,6,7,8,9],3),v(n(0,2,q1),[1,2,3,4,5,6,7,8,9],6),
q2),[1,2,3,4,5,6,7,8,9],1),v(n(3,3,q5),[1,2,3,4,5,6,7,8,9],4),v(n(3,4,q5),[1,2,3,4,5,6,7,8,9],2),v(n(3,5,
3,4,5,6,7,8,9],5),v(n(6,5,q6),[1,2,3,4,5,6,7,8,9],2),v(n(6,6,q9),[1,2,3,4,5,6,7,8,9],9),v(n(6,7,q9),[1,2,

1 3 6 | 2 4 5 | 7 8 9
7 4 8 | 1 6 9 | 3 5 2
2 5 9 | 3 7 8 | 1 6 4
3 6 1 | 4 2 7 | 8 9 5
4 8 2 | 5 9 1 | 6 3 7
5 9 7 | 6 8 3 | 4 2 1
6 1 3 | 7 5 2 | 9 4 8
8 2 4 | 9 1 6 | 5 7 3
9 7 5 | 8 3 4 | 2 1 6
```

2.1.3 Forward Checking

O forward checking o que faz é restringir o domínio das variáveis. Este problema foi semelhante ao quadrado mágico, se comparamos com o quadrado mágico que o que tamos a fazer é restringir o domínio em todo o quadrante. Neste caso temos 8 quadrantes adicionais portanto temos de diminuir o domínio nas linhas, colunas e quadrantes. O operador sucessor chamada o forward checking.

```
sucessor2(e([v(N,D,V)|R],E),e(NovoR,[v(N,D,V)|E])):-
 member(V,D), forward_checking(N,V,R,NovoR).
```

```
% reduz o dominio de uma linha , coluna e quadrante

forward_checking(_,_,[],[]):- !.
forward_checking(n(X,Y,Q),V,OldR,NewR):-
    change_linha(X,Y,V,OldR,NewR2),
    change_coluna(X,Y,V,NewR2,NewR3),
    change_quadrante(Q,V,NewR3,NewR).
```

Para mudar quadrantes:

```
change_quadrante(_,_,[],[]):- !.

change_quadrante(Q1,V,[v(n(CX,CY,Q),D,_)|R],[v(n(CX,CY,Q),D,_)|NR]):-
    Q1 \= Q,
    change_quadrante(Q1,V,R,NR).

change_quadrante(Q1,V,[v(n(CX,CY,Q),D,_)|R],[v(n(CX,CY,Q),NovoD,_)|NR]):-
    Q1 = Q,
    subtract(D,[V],NovoD),!,
    change_quadrante(Q1,V,R,NR).
```

Para mudar linhas:

```
% change_linha altera só a linha em questao, como obvio

change_linha(X,Y,V,[H|R],[H|R]):-
    X = 8,!.

change_linha(X,Y,V,[v(n(CX,CY,Q),D,_)|R],[v(n(CX,CY,Q),NovoD,_)|R]):-
    \+(X=8),
    CX = 8,
    subtract(D,[V],NovoD),!.

change_linha(X,Y,V,[v(n(CX,CY,Q),D,_)|R],[v(n(CX,CY,Q),NovoD,_)|NR]):-
    CX < 8,
    subtract(D,[V],NovoD),
    !,
    change_linha(X,Y,V,R,NR).
```

Para mudar as colunas:

```
% change_coluna

change_coluna(X,Y,V,[v(n(CX,CY,Q),D,_)|R],[v(n(CX,CY,Q),D,_)|NR]):-
    \+(X = CX),
    change_coluna(X,Y,V,R,NR).

change_coluna(X,Y,V,[H|R],[H|R]):-
    Y = 8,!.

change_coluna(X,Y,V,[v(n(CX,CY,Q),D,_)|R],[v(n(CX,CY,Q),NovoD,_)|NR]):-
    X = CX,
    CY < 8,
    subtract(D,[V],NovoD),!,
    change_coluna(X,Y,V,R,NR).

change_coluna(X,Y,V,[v(n(CX,CY,Q),D,_)|R],[v(n(CX,CY,Q),NovoD,_)|R]):-
    \+(Y=8),
    X = CX,
    CY = 8,
    subtract(D,[V],NovoD),!.
```

Uma solução backtracking com forward checking:

```
[v(n(0,0,q1),[1,2,3,4,5,6,7,8,9],1),v(n(0,1,q1),[3,4,5,6,7,8,9],3),v(n(0,2,q1),[6,7,8,9],6),v(n(0,3,q4),[2,4,5
v(n(4,5,q5),[1,3],1),v(n(4,6,q8),[6],6),v(n(4,7,q8),[3,7],3),v(n(4,8,q8),[7],7),v(n(5,0,q2),[5,6,7,8,9],5),v(n

1 3 6 | 2 4 5 | 7 8 9
7 4 8 | 1 6 9 | 3 5 2
2 5 9 | 3 7 8 | 1 6 4
3 6 1 | 4 2 7 | 8 9 5
4 8 2 | 5 9 1 | 6 3 7
5 9 7 | 6 8 3 | 4 2 1
6 1 3 | 7 5 2 | 9 4 8
8 2 4 | 9 1 6 | 5 7 3
9 7 5 | 8 3 4 | 2 1 6
```