UNIVERSIDADE DE ÉVORA

Disciplina de Inteligencia Artificial

2º Trabalho de Inteligência Artificial

Alunos:
Tiago Ávila - Nº 31565
Tiago Baião - Nº 18860
Engenharia Informática

Trabalho:
Quadrado mágico e Sudoku

Docente:
Irene Pimenta Rodrigues
Engenharia Informática

5 de abril de 2016

Introdução

Neste primeiro trabalho de inteligência artificial iremos colocar em prática os conhecimentos adquiridos nas aulas para a resolução de problemas como o problema do quadrado mágico e o problema do sudoku. Consideramos o problema do quadrado magico (3X3, 4X4, ...)como um problema de CSP. Num quadrado magico todos os elementos são diferentes e a soma das linhas, das colunas e das duas diagonais principais são iguais, e consideramos o problema do sudoku. Iremos representar em ambos os problemas como problemas de satisfações de restrições em prolog. Iremos representar os estados, as variáveis e as restrições. Iremos resolver os problemas com o algoritmo de backtracking e forward checking.

Quadrado Mágico

Considere o problema do quadrado magico (3X3, 4X4, ...) como um problema de CSP. Num quadrado magico todos os elementos são diferentes e a soma das linhas, das colunas e das duas diagonais principais são iguais. Em seguida segue o código com os estados, as variáveis e as restrições em Prolog:

```
dominio([1,2,3,4,5,6,7,8,9]).
estado_inicial( e([v(n(1,1),D,_Val),
           v(n(1,2),D,_Val),
           v(n(1,3),D,_Val),
           v(n(2,1),D,_Val),
           v(n(2,2),D,_Val),
           v(n(2,3),D,_Val),
           v(n(3,1),D,_Val),
           v(n(3,2),D,_Val),
           v(n(3,3),D,_Val)],[]) ):- dominio(D).

 %Verifica se todos os elementos no quadrado sao diferentes e valida as somas
restricoes(e(NAfect,Afect)):- all_diff(Afect), valida_somas(Afect).


all_diff([]).
all_diff([v(_,_,V)|Afect]):- member(v(_,_,V),Afect),!,fail.
all_diff([_|Afect]):- all_diff(Afect).
 %quadrado magico somas de linha coluna todas iguais
valida_somas(L):- % linhas
                  findall(V,member(v(n(1,_),_,V), L),L1), sum_total(L1),
                  findall(V,member(v(n(2,_),_,V), L),L2), sum_total(L2),
                  findall(V,member(v(n(3,_),_,V), L),L3), sum_total(L3),

                  %colunas
                  findall(V,member(v(n(_,1),_,V), L),C1), sum_total(C1),
                  findall(V,member(v(n(_,2),_,V), L),C2), sum_total(C2),
                  findall(V,member(v(n(_,3),_,V), L),C3), sum_total(C3),

                  % diagonal X = y
                  findall(V,member(v(n(1,1),_,V), L),D1),
                  findall(V,member(v(n(2,2),_,V), L),D2),
                  findall(V,member(v(n(3,3),_,V), L),D3),append(D1,D2,M),append(M,D3,X),sum_total(X),

                  % diagonal
                  findall(V,member(v(n(0,2),_,V), L),D4),
                  findall(V,member(v(n(1,1),_,V), L),D5),
                  findall(V,member(v(n(2,0),_,V), L),D6),append(D4,D5,P),append(P,D6,J),sum_total(J).

sum_total( [V1,V2,V3] ):-!, 15 is  V1+V2+V3.
sum_total(_).
```

Quadrado Mágico

Para responder ao resto do exercício segue em seguida o código com o sucessor e os algoritmos de backtracking e forward checking.O backtracking é um algoritmo de pesquisa não informada. O algoritmo utilizado neste projecto, foi o fornecido pela docente em uma aula anterior.O forward checking, tem o objetivo de limitar o domínio das variáveis à medida que as variáveis vão sendo afetadas. Ou seja quando a variável é afetada, o domínio das outras variáveis tem os valores que se encontravam no domínio com exceção do valor colocado na variável afectada.Não é possível melhorar a complexidade porque o domínio é sempre igual para todas as variáveis, assim não é possível escolher o que tem menor domínio,para melhorar a complexidade espacial e temporal.

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

%backtracking

b:- consult(square),
    estado_inicial(E0),
    back(E0,A),
    esc(A).

back(e([],A),A).
back(E,Sol):- sucessor(E,E1),
          restricoes(E1),
              back(E1,Sol).

%forward checking

f:-consult(sodoku),
     estado_inicial(E0),
     back1(E0,A),
     write(A),nl,nl,esc(A).

back1(e([],A),A).
back1(E,Sol):- sucessor(E,E1),
          restricoes(E1),
          forwardC(E1,E2),
              back(E2,Sol).

forwardC(e(NAfect,[v(N,D,V)|Afect]),e(NAfectS,[v(N,D,V)|Afect])):-
        actualizaDom(V, NAfect, NAfectS).

actualizaDom(_,[],[]).
actualizaDom(V,[v(N,D,_)|NAfect],[v(N,DS,_)|NAfectS]):-
    delete(D,V,DS),
    actualizaDom(V, NAfect, NAfectS).
```

Sudoku

Consideremos o problema do Sudoku. Em seguida segue o código com os estados, as variáveis e as restrições em Prolog:

```
tamanho_tabuleiro(9).

dominio([1,2,3,4,5,6,7,8,9]).

estado_inicial(e([v(n(1,1),D,_),v(n(1,3),D,_),v(n(1,4),D,_),v(n(1,5),D,_),v(n(1,7),D,_),
 v(n(2,1),D,_),v(n(2,2),D,_),v(n(2,3),D,_),v(n(2,5),D,_),v(n(2,7),D,_),v(n(2,8),D,_),v(n(2,9),D,_),
 v(n(3,2),D,_),v(n(3,3),D,_),v(n(3,4),D,_),v(n(3,5),D,_),v(n(3,6),D,_),v(n(3,8),D,_),
 v(n(4,1),D,_),v(n(4,2),D,_),v(n(4,3),D,_),v(n(4,4),D,_),v(n(4,5),D,_),v(n(4,7),D,_),v(n(4,8),D,_),v(n(
 v(n(5,1),D,_),v(n(5,2),D,_),v(n(5,3),D,_),v(n(5,4),D,_),v(n(5,7),D,_),
 v(n(6,2),D,_),v(n(6,3),D,_),v(n(6,5),D,_),v(n(6,6),D,_),v(n(6,7),D,_),v(n(6,8),D,_),v(n(6,9),D,_),
 v(n(7,1),D,_),v(n(7,2),D,_),v(n(7,3),D,_),v(n(7,5),D,_),v(n(7,6),D,_),v(n(7,7),D,_),v(n(7,8),D,_),v(n(
 v(n(8,3),D,_),v(n(8,4),D,_),v(n(8,6),D,_),v(n(8,7),D,_),v(n(8,9),D,_),
 v(n(9,1),D,_),v(n(9,2),D,_),v(n(9,4),D,_),v(n(9,5),D,_),v(n(9,7),D,_),v(n(9,8),D,_),v(n(9,9),D,_)],
[v(n(1,2),D,1),
v(n(1,6),D,8),
v(n(1,8),D,7),
v(n(1,9),D,3),
v(n(2,4),D,5),
v(n(2,6),D,9),
v(n(3,1),D,7),
v(n(3,7),D,9),
v(n(3,9),D,4),
v(n(4,6),D,4),
v(n(5,5),D,3),
v(n(5,6),D,5),
v(n(5,8),D,1),
v(n(5,9),D,8),
v(n(6,1),D,8),
v(n(6,4),D,9),
v(n(7,4),D,7),
v(n(8,1),D,2),
v(n(8,2),D,6),
v(n(8,5),D,4),
v(n(8,8),D,3),
v(n(9,3),D,5),
v(n(9,6),D,3)]) ):- dominio(D).

 %Verifica todas as restriçoes do sudoku
restricoes(e(NAfect,Afect)):-
ver_linhas(e(Nafect,Afect)),
ver_colunas(e(Nafect,Afect)),
ver_quadrantes(e(Nafect,Afect)).
```

```
all_diff([]).
all_diff([H|T]) :-
\+member(H, T), all_diff(T).

ver_linhas(e(Nafect,[v(n(X,Y), D, V)|R])):- findall(V1,member(v(n(X,_),_,V1),R),L), all_diff([V|L]).

ver_colunas(e(Nafect,[v(n(X,Y), D, V)|R])):- findall(V1,member(v(n(_,Y),_,V1),R),L), all_diff([V|L]).

ver_quadrantes(e(_, Afect)):-
%ve o primeiro quadrante
ver_quadrante(Afect, 1, 1, 3, Q1), all_diff(Q1),

%ve o segundo quadrante
ver_quadrante(Afect, 1, 4, 6, Q2),all_diff(Q2),

%ve o terceiro quadrante
ver_quadrante(Afect, 1, 7, 9, Q3),all_diff(Q3),

%ve o quarto quadrante
ver_quadrante(Afect, 4, 1, 3, Q4),all_diff(Q4),

%ve o quinto quadrante
ver_quadrante(Afect, 4, 4, 6, Q5),all_diff(Q5),

%ve o sexto quadrante
ver_quadrante(Afect, 4, 7, 9, Q6),all_diff(Q6),

%ve o sétimo quadrante
ver_quadrante(Afect, 7, 1, 3, Q7),all_diff(Q7),

%ve o oitavo quadrante
ver_quadrante(Afect, 7, 4, 6, Q8),all_diff(Q8),

%ve o nono quadrante
ver_quadrante(Afect, 7, 7, 9, Q9),all_diff(Q9).


ver_quadrante(L, X, Y, Y2, L2):-
Y = Y2, X1 is X+2,
ver_q_c(L, X, Y, X1, L2).

ver_quadrante(L, X, Y, Y2, L3):-
Y < Y2, Y1 is Y+1,
X1 is X+2,
ver_q_c(L, X, Y, X1, L1),
append(L1, L2, L3),
ver_quadrante(L, X, Y1, Y2, L2).

ver_q_c(L, X, Y, X2, []):-
```

```
X = X2,
\+member(v(n(X, Y), _, _), L).

ver_q_c(L, X, Y, X2, [V]):-
        X = X2,
        member(v(n(X,Y), _, V), L).

ver_q_c(L, X, Y, X2, T):-
        X < X2, X1 is X+1,
        \+member(v(n(X, Y), _, _), L),
ver_q_c(L, X1, Y, X2, T).

ver_q_c(L, X, Y, X2, [V|T]):-
X < X2,
        member(v(n(X,Y), _, V), L),
X1 is X+1,
        ver_q_c(L, X1, Y, X2, T).
```

Para responder ao resto do exercício segue em seguida o código com o sucessor e os algoritmos de backtracking e forward checking. O backtracking é um algoritmo de pesquisa não informada. O algoritmo utilizado neste projecto, foi o fornecido pela docente em uma aula anterior. O forward checking, tem o objetivo de limitar o domínio das variáveis à medida que as variáveis vão sendo afetadas. Ou seja quando a variável é afetada, o domínio das outras variáveis tem os valores que se encontravam no domínio com exceção do valor colocado na variável afectada. Não é possível melhorar a complexidade porque o domínio é sempre igual para todas as variáveis, assim não é possível escolher o que tem menor domínio,para melhorar a complexidade espacial e temporal.

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

%backtracking
b:- consult(sodoku),
    estado_inicial(E0),
    back(E0,A),
    esc(A).

back(e([],A),A).
back(E,Sol):- sucessor(E,E1),
          restricoes(E1),
              back(E1,Sol).

%forward checking
f:-consult(sodoku),
     estado_inicial(E0),
     back1(E0,A),
     write(A),nl,nl,esc(A).

back1(e([],A),A).
back1(E,Sol):- sucessor(E,E1),
          restricoes(E1),
          forwardC(E1,E2),
              back(E2,Sol).

forwardC(e(NAfect,[v(N,D,V)|Afect]),e(NAfectS,[v(N,D,V)|Afect])):-
        actualizaDom(V, NAfect, NAfectS).

actualizaDom(_,[],[]).
actualizaDom(V,[v(N,D,_)|NAfect],[v(N,DS,_)|NAfectS]):-
    delete(D,V,DS),
    actualizaDom(V, NAfect, NAfectS).
```