Inteligência Artificial

Relatório do 2º trabalho prático
2022/2023

Resolução de problemas como problemas de satisfação de restrições

Trabalho realizado por:
- Rodrigo Alves, nº48681
- Luís Simões, nº48726

Introdução

Com a resolução deste trabalho é pretendido que sejam utilizadas as capacidades de resoluções de problemas como problemas de satisfação de restrições.

Para isso, deveremos conseguir perceber as diferenças entre as pesquisas backtracking e a backtracking com forward checking, percebendo também as variáveis existentes, como o nome, domínio e valor, e as restrições para cada exercício.

Resolução dos Exercícios

Exercício 1 - Quadrado Mágico

(a)

Para representar este problema como um problema de satisfação de restrições em prolog definimos o tamanho do nosso quadrado como sendo 3:

```
size(3).
```

Depois definimos o nosso estado inicial:

```
estado_inicial(
    e([
        v((1,1),[1,2,3,4,5,6,7,8,9],_), v((1,2),[1,2,3,4,5,6,7,8,9],_), v((1,3),[1,2,3,4,5,6,7,8,9],_),
        v((2,1),[1,2,3,4,5,6,7,8,9],_), v((2,2),[1,2,3,4,5,6,7,8,9],_), v((2,3),[1,2,3,4,5,6,7,8,9],_),
        v((3,1),[1,2,3,4,5,6,7,8,9],_), v((3,2),[1,2,3,4,5,6,7,8,9],_), v((3,3),[1,2,3,4,5,6,7,8,9],_)],
      [])).
```

O nosso estado inicial é composto por um estado ‘e’ que por sua vez é composto por duas listas de variáveis, sendo a primeira a lista de variáveis não instanciadas e a segunda a lista de variáveis instanciadas. Na primeira, como podemos ver acima temos as variáveis ‘v(N,D,V)’ onde N é o nome da variável, que neste caso tem a sua posição, D é o domínio da variável e V o valor. Inicialmente o valor está a ‘_’ uma vez que ainda não tem nenhum valor atribuído.

De seguida definimos o nosso predicado para o sucessor que por acaso é idêntico ao que estava presente nos algoritmos de pesquisa bactracking e pesquisa forwardchecking:

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

Depois implementámos as restrições do jogo:

```
ve_restricoes(e(_, [v((X,Y),_,V)|VariaveisInstanciadas])):-
    findall(V1, (member(v(_,_,V1), VariaveisInstanciadas), integer(V1)), Valores),
    all_distinct([V|Valores]),
    validar_linha([v((X,Y),_,V)|VariaveisInstanciadas]),
    validar_coluna([v((X,Y),_,V)|VariaveisInstanciadas]),
    validar_diagonal_principal([v((X,Y),_,V)|VariaveisInstanciadas]),
    validar_diagonal_secundaria([v((X,Y),_,V)|VariaveisInstanciadas]).

%Validar a linha quando esta está preenchida (tamanho 3), ou seja, verificar se a soma desta é igual a 15
validar_linha([v((X,_),_,V1)|VariaveisInstanciadas]):-
    findall(V, (member(v((X,_),_,V), VariaveisInstanciadas), integer(V)), Linha),
    tamanho([V1|Linha], T), T \= 3.

validar_linha([v((X,_),_,V1)|VariaveisInstanciadas]):-
    findall(V, (member(v((X,_),_,V), VariaveisInstanciadas), integer(V)), Linha),
    tamanho([V1|Linha], T), T == 3,
    soma([V1|Linha], 15).

%Validar a coluna quando esta está preenchida (tamanho 3), ou seja, verificar se a soma desta é igual a 15
validar_coluna([v((_,Y),_,V1)|VariaveisInstanciadas]):-
    findall(V, (member(v((_,Y),_,V), VariaveisInstanciadas), integer(V)), Coluna),
    tamanho([V1|Coluna], T), T \= 3.

validar_coluna([v((_,Y),_,V1)|VariaveisInstanciadas]):-
    findall(V, (member(v((_,Y),_,V), VariaveisInstanciadas), integer(V)), Coluna),
    tamanho([V1|Coluna], T), T == 3,
    soma([V1|Coluna], 15).

validar_diagonal_principal([v((X,Y),_,V1)|VariaveisInstanciadas]) :-
    X \= Y.

validar_diagonal_principal([v((X,Y),_,V1)|VariaveisInstanciadas]) :-
    X == Y,
    findall(V, (member(v((K,K),_,V), VariaveisInstanciadas), integer(V)), Diagonal),
    tamanho([V1|Diagonal], T), T \= 3.

validar_diagonal_principal([v((X,Y),_,V1)|VariaveisInstanciadas]) :-
    X == Y,
    findall(V, (member(v((K,K),_,V), VariaveisInstanciadas), integer(V)), Diagonal),
    tamanho([V1|Diagonal], T), T == 3,
    soma([V1|Diagonal], 15).

validar_diagonal_secundaria([v((X,Y),_,V1)|VariaveisInstanciadas]) :-
    X + Y \= 4.

validar_diagonal_secundaria([v((X,Y),_,V1)|VariaveisInstanciadas]) :-
    X + Y == 4,
    findall(V, (member(v((X,Y),_,V), VariaveisInstanciadas), integer(V)), Diagonal),
    tamanho([V1|Diagonal], T), T \= 3.

validar_diagonal_secundaria([v((X,Y),_,V1)|VariaveisInstanciadas]) :-
    X + Y == 4,
    findall(V, (member(v((X,Y),_,V), VariaveisInstanciadas), integer(V)), Diagonal),
    tamanho([V1|Diagonal], T), T == 3,
    soma([V1|Diagonal], 15).
```

Para que as nossas restrições funcionassem implementámos predicados auxiliares para calcular a soma de uma lista, para calcular o tamanho de uma lista e para verificar se todos os elementos de uma lista são distintos:

```
soma([], 0).
soma([X|Xs], Total) :- soma(Xs, R), Total is X + R.

all_distinct([]).
all_distinct([X|Xs]) :-
    \+ member(X, Xs),
    all_distinct(Xs).

tamanho([], 0).
tamanho([_ | Resto], Tamanho) :-
    tamanho(Resto, TamanhoResto),
    Tamanho is TamanhoResto + 1.
```

Por fim, fizemos uma função para printar o resultado de um estado:

```
%esc(_).
esc(L):-sort(L, L1), esc_a(L1),nl.

esc_a(L):- size(S), esc_l(L, 1, S).

esc_l([H], S, S):- H = v(_,_,X), write(X),nl.

esc_l([H|T], S, S):- H = v(_,_,X), write(X), nl,esc_l(T, 1, S).

esc_l([H|T], I, S):- I<S, I2 is I+1,
                     H = v(_,_,X), write(X),write(' | '),
                      esc_l(T, I2, S).
```

(b)

Por alguma razão com o algoritmo de backtracking, o nosso programa não funciona quando temos o estado inicial como o temos acima devido a algo que tenha a ver com diagonal secundária. Contudo, se instanciármos uma das variáveis da diagonal secundária com um valor aleatório já conseguimos obter resultados que estão corretos.

(c)

Por alguma razão, a nossa pesquisa forward só conseguiu chegar ao resultado final correto quando instaciamos também uma variável com a diagonal secundária também. Suspeitamos que o erro possa ser na mesma relacionado com o predicado que valida a diagonal secundária, contudo, como pensamos que o predicado está certo, é difícil ter a certeza.

(d)

Para este problema não percebemos como deveríamos proceder para aumentar para ainda melhor a complexidade de resolução do problema.

(e)

Na pesquisa com backtracking se instanciármos uma das variáveis da diagonal secundária com um valor aleatório, como por exemplo:

```
estado_inicial(
    e([
        v((1,1),[1,2,3,4,5,6,7,8,9],_), v((1,2),[1,2,3,4,5,6,7,8,9],_),
        v((2,1),[1,2,3,4,5,6,7,8,9],_), v((2,2),[1,2,3,4,5,6,7,8,9],_), v((2,3),[1,2,3,4,5,6,7,8,9],_),
        v((3,1),[1,2,3,4,5,6,7,8,9],_), v((3,2),[1,2,3,4,5,6,7,8,9],_), v((3,3),[1,2,3,4,5,6,7,8,9],_)],
      [v((1,3),[1,2,3,4,5,6,7,8,9],4)])).
```

Então nosso programa já corre e dá os outputs:

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

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

Na pesquisa com backtracking e forward checking obtemos os resultados para o estado inicial sem variáveis instanciadas como:

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

Que está claramente mal uma vez que a soma da diagonal secundária não dá 15 como é suposto mas sim 24. Contudo, como podemos verificar o resto dos resultados cumprem as restrições, ou seja, a soma dos valores de cada linha é igual à soma dos valores de cada coluna é igual à soma dos valores da diagonal principal.

No entanto, quando instanciamos uma variável na diagonal secundária, como por exemplo:

```
estado_inicial(
    e([
        v((1,1),[1,2,3,4,5,6,7,8,9],_), v((1,2),[1,2,3,4,5,6,7,8,9],_), v((1,3),[1,2,3,4,5,6,7,8,9],4),
        v((2,1),[1,2,3,4,5,6,7,8,9],_), v((2,3),[1,2,3,4,5,6,7,8,9],_),
        v((3,1),[1,2,3,4,5,6,7,8,9],_), v((3,2),[1,2,3,4,5,6,7,8,9],_), v((3,3),[1,2,3,4,5,6,7,8,9],_)],
      [v((2,2),[1,2,3,4,5,6,7,8,9],5)])).
```

Obtemos outputs como:

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

Exercício 2 - Sudoku

(a)

De forma a representar os estados neste problema, decidimos ter variáveis com um tuplo que é definido por 3 números (X,Y,Z). O X e o Y representam as coordenadas do quadrado da tabela do sudoku, e o terceiro, o Z, representa o quadrante onde está inserido, sendo que o tabuleiro do sudoku tem 9 quadrantes.
O domínio é definido pelo o número de quadrantes que o sudoku possui, logo é definido pelo seguinte intervalo: [1,2,3,4,5,6,7,8,9]. Relativamente ao valor, apenas dizemos o número relativo ao mesmo caso existir no sudoku inicial, ou "_", no caso de não existir.

Para impor as restrições neste jogo, utilizamos a função *findall* para encontrar todos os valores existentes em cada posição com o mesmo valor de X (que estão na mesma linha). Em seguida, aplicamos o predicado *all_diff* para verificar se todos os valores encontrados são diferentes. Repetimos esse processo para as colunas (Y) e os quadrantes (Z) do tabuleiro.

O estado inicial, com o domínio já mencionado anteriormente:

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

estado_inicial(e([v(c(1,1,1),D,_),v(c(1,3,1),D,_),v(c(1,4,2),D,_),v(c(1,5,2),D,_),v(c(1,7,3),D,_),
                  v(c(2,1,1),D,_),v(c(2,2,1),D,_),v(c(2,3,1),D,_),v(c(2,5,2),D,_),v(c(2,7,3),D,_),v(c(2,8,3),D,_),v(c(2,9,3),D,_),
                  v(c(3,2,1),D,_),v(c(3,3,1),D,_),v(c(3,4,2),D,_),v(c(3,5,2),D,_),v(c(3,6,2),D,_),v(c(3,8,3),D,_),
                  v(c(4,1,4),D,_),v(c(4,2,4),D,_),v(c(4,3,4),D,_),v(c(4,4,5),D,_),v(c(4,5,5),D,_),v(c(4,7,6),D,_),v(c(4,8,6),D,_),v(c(4,9,6),D,_),
                  v(c(5,1,4),D,_),v(c(5,2,4),D,_),v(c(5,3,4),D,_),v(c(5,4,5),D,_),v(c(5,7,6),D,_),
                  v(c(6,2,4),D,_),v(c(6,3,4),D,_),v(c(6,5,5),D,_),v(c(6,6,5),D,_),v(c(6,7,6),D,_),v(c(6,8,6),D,_),v(c(6,9,6),D,_),
                  v(c(7,1,7),D,_),v(c(7,2,7),D,_),v(c(7,3,7),D,_),v(c(7,5,8),D,_),v(c(7,6,8),D,_),v(c(7,7,9),D,_),v(c(7,8,9),D,_),v(c(7,9,9),D,_),
                  v(c(8,3,7),D,_),v(c(8,4,8),D,_),v(c(8,6,8),D,_),v(c(8,7,9),D,_), v(c(8,9,9),D,_),
                  v(c(9,1,7),D,_),v(c(9,2,7),D,_),v(c(9,4,8),D,_),v(c(9,5,8),D,_), v(c(9,7,9),D,_), v(c(9,8,9),D,_),v(c(9,9,9),D,_)],


% Posições já preenchidas na tabela dada
[v(c(1,2,1),D,1),v(c(1,6,2),D,8), v(c(1,8,3),D,7), v(c(1,9,3),D,3),
v(c(2,4,2),D,5), v(c(2,6,2),D,9),
v(c(3,1,1),D,7), v(c(3,7,3),D,9), v(c(3,9,3),D,4),
v(c(4,6,5),D,4),
v(c(5,5,5),D,3), v(c(5,6,5),D,5), v(c(5,8,6),D,1), v(c(5,9,6),D,8),
v(c(6,1,4),D,8), v(c(6,4,5),D,9),
v(c(7,4,8),D,7),
v(c(8,1,7),D,2), v(c(8,2,7),D,6), v(c(8,5,8),D,4), v(c(8,8,9),D,3),
v(c(9,3,7),D,5), v(c(9,6,8),D,3)])):- dominio(D).
```

As restrições que foram definidas:

```
ve_restricoes(e(_,[v(c(X,Y,Z), _, V)|Afect])):-
    findall(V1,member(v(c(X,_,_),_,V1),Afect),L), all_diff([V|L]),
    findall(V2,member(v(c(_,Y,_),_,V2),Afect),L2), all_diff([V|L2]),
    findall(V3,member(v(c(_,_,Z),_,V3),Afect),L3), all_diff([V|L3]),
    L \= L2, L3\=L, L3\=L2.

all_diff([]).
all_diff([X|Afect]):-
    \+ member(X,Afect), all_diff(Afect).
```

O operador sucessor:

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

As células do sudoku que já estão preenchidas estão definidas no estado inicial, marcadas com o respectivo valor..

(b)

Problema resolvido com o algoritmo de backtracking:

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

O algoritmo demorou cerca de 87 ms a retornar este tabuleiro como output e escreveu o mesmo em 26565 bytes.

(c)

Por alguma razão, a nossa pesquisa forward não conseguiu chegar ao resultado final do tabuleiro, apesar de escrever uma grande quantidade de bytes(26565), e de demorar ainda algum tempo a correr(76 ms).

(d)

De forma a melhorar a complexidade do algoritmo do sudoku, podemos fazer algumas alterações no código principal. O uso de *findall* para obter todos os valores das linhas e das

tabelas pode ser dispendioso em termos de complexidade espacial, o que pode levar a um aumento na complexidade temporal ao procurar entre todos os valores.
Uma abordagem que pode ser adotada é a pesquisa com "forward checking", pois à medida que escrevemos um número em uma determinada célula, podemos remover do domínio todos os números que já estão presentes na mesma linha, coluna e quadrante. Desta forma, reduz significativamente o tempo de procura, uma vez que temos menos opções disponíveis para preencher cada célula à medida que avançamos no jogo.
Para além disso, ao restringir o domínio cada vez que introduzimos um novo valor na linha, garantimos que não haverá duplicatas em cada linha do tabuleiro.

Conclusão

Com a realização deste trabalho ficámos a ter mais conhecimento sobre a resolução de problemas como problemas de satisfação de restrições, pois aprofundámos e usámos numa vertente mais prática todos os tipos de pesquisa dados nas aulas em problemas concretos (pesquisa backtracking e pesquisa backtracking com forward checking).

Para além disso, tivemos que nós próprios pensar nas melhores restrições para cada um dos problemas, o que foi bastante interessante de fazer, puxando bastante pelo nosso raciocínio.

Pensamos ter atingido a maioria dos objetivos do trabalho, principalmente para o problema do sudoku, onde chegámos à solução correta com a pesquisa com backtracking.

Já no problema do Quadrado Mágico apesar de percebermos muito bem as restrições, não chegámos a grandes valores concretos uma vez que o nosso predicado para validar diagonais secundárias deve ter algum problema que não conseguimos identificar.