# Inteligência Artificial

# Trabalho 2

# Problema de satisfação de restrições



Discentes:

João Santos nº 29634

André Gouveia nº 26918

# Índice

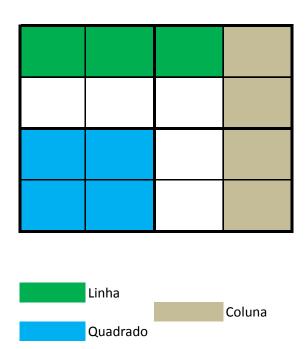
Introdução:	2
Respostas:	
1.(a)	
1.(b)	
1.(c)	
1.(e)	
i. 3 casas por preencher	
ii. 9 casas por preencher	8
iii. 16 casas por preencher:	9
Instrucões:	10

# Introdução:

Para este trabalho foi-nos solicitado a resolução de um problema de sudoku 4x4 encarando-o como um problema de satisfação de restrições.

As restrições num problema de sudoku são :

- Cada linha não pode repetir valores
- Cada coluna não pode repetir valores
- Cada quadrado não pode repetir valores



#### **Respostas:**

1.(a)

Os estados são definidos por listas. Temos uma lista para as posições já afectadas e outra para as posições a afectar. Cada posição é definida por v(c(i,j),D,V) em que i e j são as coordenadas da posição, D é o dominio afecto (neste caso será 1 a 4 uma vez que apenas esses valores podem ser tomados) e V que será o valor a atribuir à posição.

O estado inicial é dado à partida. Caso a posição esteja ocupada, é colocada na lista de afectados com valor atribuído. Caso contrário é colocada na lista de não afectados sem valor atribuído.

Ex:

#### O estado inicial:

 $estado\_inicial(e([v(c(1,1),[1,2,3,4],\_),v(c(1,2),[1,2,3,4],\_),v(c(1,3),[1,2,3,4],\_)],\\ [v(c(1,4),[4],4),v(c(2,1),[3],3),v(c(2,2),[4],4),v(c(2,3),[1],1),v(c(2,4),[2],2),v(c(3,1),[2],2),v(c(3,2),[1],1),v(c(3,3),[4],4),v(c(3,4),[3],3),v(c(4,1),[4],4),v(c(4,2),[3],3),v(c(4,3),[2],2),v(c(4,4),[1],1)])).$ 

Define o seguinte problema:

			4
3	4	1	2
2	1	4	3
4	3	2	1

O operador sucessor em backtracking foi definido como :

sucessor(e([v(N,D,V)|R],E),e(R1,[v(N,D,V)|E])):-member(V,D).

#### As restrições definidas da seguinte forma:

```
ve_restricoes(e(_,Afect)):-
    \+ (member(v(c(I,J),_,Vj), Afect), member(v(c(I,K),_,Vk),Afect), K \=J,Vk=Vj),
    \+ (member(v(c(I,J),_,Vi), Afect), member(v(c(K,J),_,Vk),Afect), K \=I,Vi=Vk),
    \+ (member(v(c(1,1),_,Vi), Afect), member(v(c(2,2),_,Vj),Afect), Vi=Vj),
    \+ (member(v(c(2,1),_,Vi), Afect), member(v(c(1,2),_,Vj),Afect), Vi=Vj),
    \+ (member(v(c(3,1),_,Vi), Afect), member(v(c(4,2),_,Vj),Afect), Vi=Vj),
    \+ (member(v(c(3,2),_,Vi), Afect), member(v(c(2,4),_,Vj),Afect), Vi=Vj),
    \+ (member(v(c(2,3),_,Vi), Afect), member(v(c(1,4),_,Vj),Afect), Vi=Vj),
    \+ (member(v(c(3,3),_,Vi), Afect), member(v(c(4,4),_,Vj),Afect), Vi=Vj),
    \+ (member(v(c(3,4),_,Vi), Afect), member(v(c(4,3),_,Vj),Afect), Vi=Vj),
    \+ (member(v(c(3,4),_,Vi), Afect), member(v(c(4,3),_,Vj),Afect), Vi=Vj).
```

Sendo que a 1ª linha verifica as restrições de linha, a 2ª linha as restrições de coluna e as remanescentes as diagonais de cada quadrado.

#### 1.(b)

Para a resolução em backtracking, é necessária a afectação de valores do dominio às diferentes posições e a sua confirmação através das restrições definidas. Para esse fim usamos o seguinte código:

```
p(Prg):- consult(Prg), estado\_inicial(E0), back(E0,A), esc(A). back(e([],A),A). back(E,SoI):- sucessor(E,E1), ve\_restricoes(E1), back(E1,SoI). sucessor(e([v(N,D,V)|R],E), e(R1,[v(N,D,V)|E])):- member(V,D).
```

#### 1.(c)

Para a resolução em forward checking é necessário, ao fazer uma atribuição, remover dos dominios com a qual interfere, todas as opções que causam conflito. Neste caso, ao atribuir um valor a uma posição, é necessário remover do dominio de toda a linha, toda a coluna e todo o quadrado associado a essa posição esse mesmo valor.

#### Para tal usamos o seguinte código:

```
quadrado(c(1,1),[c(1,2),c(2,1),c(2,2)]).
quadrado(c(1,2),[c(1,1),c(2,1),c(2,2)]).
quadrado(c(2,1),[c(1,2),c(1,1),c(2,2)]).
quadrado(c(2,2),[c(1,2),c(2,1),c(1,1)]).
quadrado(c(3,1),[c(3,2),c(4,1),c(4,2)]).
quadrado(c(3,2),[c(3,1),c(4,1),c(4,2)]).
quadrado(c(4,1),[c(3,2),c(3,1),c(4,2)]).
quadrado(c(4,2),[c(3,1),c(4,1),c(3,2)]).
quadrado(c(1,3),[c(1,4),c(2,3),c(2,4)]).
quadrado(c(1,4),[c(1,3),c(2,3),c(2,4)]).
quadrado(c(2,3),[c(1,4),c(2,4),c(1,3)]).
quadrado(c(2,4),[c(1,4),c(2,3),c(1,3)]).
quadrado(c(4,4),[c(3,4),c(3,3),c(4,3)]).
quadrado(c(4,3),[c(3,4),c(3,3),c(4,4)]).
quadrado(c(3,3),[c(3,4),c(4,3),c(4,4)]).
quadrado(c(3,4),[c(3,3),c(4,3),c(4,4)]).
p(Prg):- consult(Prg),estado_inicial(E0),back(E0,A), esc(A).
back(e([],A),A).
```

```
back(E,Sol):- sucessor(E,E1), ve_restricoes(E1),
back(E1,Sol).
sucessor(e([v(N,D,V)|R],E),e(R1,[v(N,D,V)|E])):-member(V,D), remove(N,V,R,R1).
remove(c(I,J),V,R,R3):-
 linha(I,V,R,R1),
 coluna(J,V,R1,R2),
 quadrados(c(I,J),V,R2,R3).
linha(_,_,[],[]).
linha(Linha,Valor,[v(c(Linha,Col),D,\_)|R],[v(c(Linha,Col),D2,\_)|R1]):-
 removeLista(Valor,D,D2),
 linha(Linha, Valor, R, R1),!.
linha(Linha, Valor, [I|R], [I|R1]):-
 linha(Linha, Valor, R, R1),!.
coluna(_,_,[],[]).
coluna(Col,Valor,[v(c(Linha,Col),D,_)|R],[v(c(Linha,Col),D2,_)|R1]):-
 removeLista(Valor,D,D2),
 coluna(Col, Valor, R, R1),!.
coluna(Col, Valor, [I|R], [I|R1]):-
 coluna(Col, Valor, R, R1),!.
quadrados(N,V,R,R1):-
 quadrado(N,Lista),
 removeQuads(Lista,V,R,R1).
removeQuads([],_,R,R).
removeQuads([N|L],V,R,R2):-
 member(v(N,_,_),R),
 findRemove(v(N,_,_),V,R,R1),
```

removeQuads(L,V,R1,R2).

removeQuads([N|L],V,R,R1):\+ member(v(N,\_,\_),R),

removeQuads(L,V,R,R1).

findRemove(v(N,\_,\_),V,[v(N,D,L)|R],[v(N,D2,L)|R]):
removeLista(V,D,D2),!.

findRemove(v(N,\_,\_),V,[I|R],[I|R1]):
findRemove(v(N,\_,\_),V,R,R1).

removeLista(\_,[],[]):-!.

removeLista(Valor,[Valor|R],R):-!.

removeLista(Valor,[X|R],[X|R1]):
removeLista(Valor,R,R1).

## 1.(e)

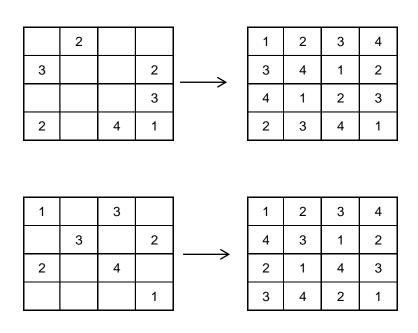
#### i. 3 casas por preencher

			4	1	2	3	4
3	4	1	2	 3	4	1	2
2	1	4	3	2	1	4	3
4	3	2	1	4	3	2	1

4	3	1	2		4	3	1	2
			4	,	1	2	3	4
2	1	4	3		2	1	4	3
4	3	2	1		4	3	2	1

4	3	1	2		4	3	1	2
2	1	4	3		2	1	4	3
			4		1	2	3	4
3	4	2	1		3	4	2	1
			T	Ī				
3	4	1			3	4	1	2
2	1		3	_	2	1	4	3
1		3	4		1	2	3	4
4	3	2	1		4	3	2	1

### ii. 9 casas por preencher



1	2		4	1	2	3	4
		1	2	4	3	1	2
	1			2	1	4	3
3			1	3	4	2	1

4	3		2	4	3	1	2
		4		2	1	4	3
	2			 1	2	3	4
3			1	3	4	2	1

## iii. 16 casas por preencher:

1	2	3	4
3	4	1	2
4	1	2	3
2	3	4	1

1	2	3	4
4	3	2	1
3	1	4	2
2	4	1	3

1	2	3	4
4	3	2	1
2	1	4	3
3	4	1	2

1	2	3	4
4	3	1	2
2	1	4	3
3	4	2	1

## Instruções:

Para correr o programa basta na linha de comandos introduzir:

swipl -s back.pl

Depois de compliado basta correr o predicado p da seguinte forma:

p('sudoku.pl').

Para alterar o sudoku base, é necessário aceder a sudoku.pl e alterar o estado inicial conforme indicado em Respostas: 1.(a) .