

Inteligência Artificial

Prof. Fabrício Olivetti de França Prof. Denis Fantinato

3º Quadrimestre de 2019

Problema de Satisfação de Restrição

Limitações das Buscas (Des)Informadas

- O estado de um problema tem uma estrutura arbitrária diferente para cada problema
- O teste para verificar se o objetivo foi atingido é específico para essa estrutura
- A função que escolhe o estado sucessor também é específico para o problema

Problema de Satisfação de Restrição (CSP)

O **Problema de Satisfação de Restrição** (CSP, do inglês *Constraint Satisfaction Problem*) formaliza os problemas de busca de acordo com:

- Conjunto de variáveis X do problema
- Conjunto de domínios D de cada variável
- Conjunto de restrições C determinando a validade de uma configuração de X

Problema de Satisfação de Restrição (CSP)

Ao definir um problema seguindo essa formalização, podemos pensar em um algoritmo único que funcione para todos os algoritmos.

Exemplo: Colorir um mapa

Dado o mapa da Australia abaixo, queremos colorir cada estado de uma cor diferente de seus vizinhos:



Exemplo: Colorir um mapa

Formulação CSP:

- $X = \{WA, NT, Q, SW, V, SA, T\}$
- $D_i = \{\text{vermelho, verde, azul}\}$ para todo i
- $C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$

Exemplo: Colorir um mapa

Computacionalmente pode ser interessante representar as restrições na forma:

$$C = \{(SA, WA, \{(verm, verd), (verm, azul), (verd, azul), (verd, verm), (azul, verd), (azul, verm)\}), (SA, NT, \{(verm, verd), \dots\}), \dots\}$$

Assim podemos padronizar nosso algoritmo para lidar com as restrições.

Exemplo: Colorir um mapa

É fácil perceber que ao definir $SA = \text{azul}$, nenhum de seus vizinhos pode usar essa cor.

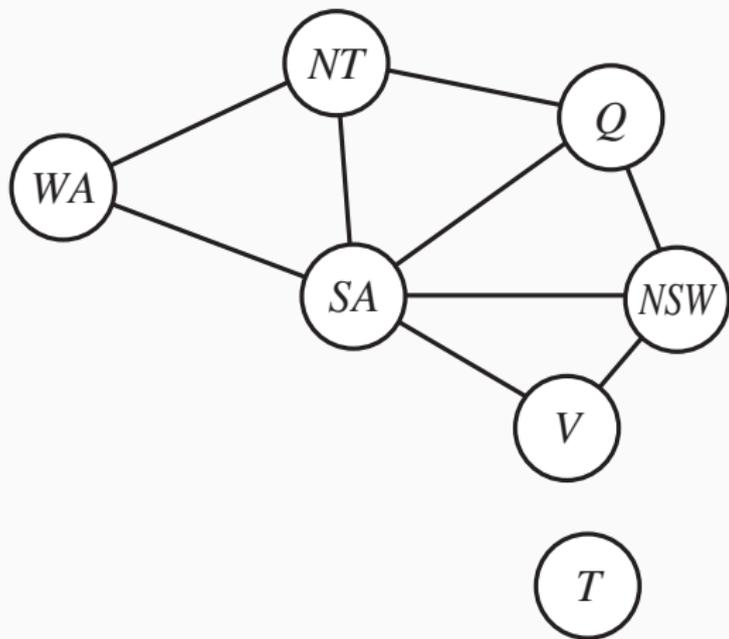
Exemplo: Colorir um mapa

Uma vez escolhido $\{SA = azul\}$:

- Em um algoritmo de busca (des)informado, teríamos que explorar para as 5 variáveis vizinhas $3^5 = 243$ possibilidades de alocações de cores.
- Com o uso de restrições (nenhum vizinho de SA pode ser *azul*), teríamos que explorar apenas $2^5 = 32$ alocações.

Grafo de restrições

Representa as restrições na forma de grafos em que cada nó representa uma variável do problema, arestas representam a restrição (ex.: \neq).



O mais comum é pensarmos sempre em **variáveis discretas**, ou seja, X_i é azul ou vermelho, X_i representa a quantidade de compra de certo material.

Os problemas discretos podem ter **domínio finito ou infinito**.

- Domínio finito: cores do mapa, posições da rainha no tabuleiro.
- Domínio infinito: quantidade de compra de material, tempo de início de cada tarefa.

Podemos ter variáveis contínuas em um CSP: tempo (com alta precisão), valor de medição como número real, etc.

Se as restrições forem lineares, existem algoritmos que encontram solução em tempo polinomial (veja Programação Matemática).

Quanto as restrições, elas podem ser:

- **Unárias:** envolvendo uma única variável, $SA \neq verde$
- **Binárias:** envolvendo duas variáveis, $SA \neq WA$
- **Alta ordem (ou Global):** envolvendo 3 ou mais variáveis,
 $SA \neq WA \neq NT$

As restrições também podem modelar preferências (uso de diferentes pesos)!

Exercício

Modele o problema do Sudoku como um CSP

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

(b)

Resolvendo problemas CSP

Um nó do grafo de restrições é dito **nó consistente** se nenhum valor de seu domínio viola uma restrição unária envolvendo a variável.

É possível remover todas as restrições unárias de um grafo de restrições, removendo as restrições do domínio.

No exemplo de colorir o mapa, se definirmos a restrição $SA \neq verde$, então ele passa a não ser consistente pois seu domínio é $\{verde, vermelho, azul\}$.

Devemos então fazer com que o domínio de SA seja $D_{SA} = \{vermelho, azul\}$.

Uma variável é dita **arco consistente** se o domínio dessa variável satisfaz todas as restrições binárias envolvendo ela.

Da mesma forma é possível remover os valores do domínio de cada variável para torná-la consistente.

Imagine que temos duas variáveis X_1, X_2 e a restrição $X_2 = X_1^2$, sendo o domínio de ambas $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Para tornar as duas variáveis consistentes devemos fazer:

$$D_1 = \{0, 1, 2, 3\}$$

$$D_2 = \{0, 1, 4, 9\}$$

Algoritmo AC-3

Para tornar um problema arco consistente podemos seguir os seguintes passos (*queue* = fila com arcos):

```
def AC3(csp, queue):
    if empty(queue):
        return true
    (i, j) = queue.pop()
    revised, csp = Revise(csp, i, j)
    if revised:
        D = csp[1]
        if empty(D[i]): return false
        [queue.push( (k,i) )
         for k in neighbors(i, csp) if k!=i]
    return AC3(csp, queue)
```

Se C contém as restrições de diferença (Ex: $C = \{SA \neq WA\}$):

```
def Revise(csp, i, j):
    X, D, C = csp
    violate = set([x for x in D[i]
                   if any([(x,y) in C[i][j] for y in D[j]])])
    if not empty(violate):
        D[i] = D[i] - violate
        return true, (X, D, C)
    return false, csp
```

Se C contém as restrições na forma de conjuntos possíveis (Ex:

$C = \{(SA, WA, \{(verm, verd), (verm, azul), \dots\})\}$):

```
def Revise(csp, i, j):
    X, D, C = csp
    violate = set([x for x in D[i]
                   if len([(x,y) in C[i][j] for y in D[j]])==0])
    if not empty(violate):
        D[i] = D[i] - violate
        return true, (X, D, C)
    return false, csp
```

Podemos perceber que a consistência de arco não elimina nenhum valor dos domínios de nosso problema de colorir o mapa.

Precisamos de eliminação de valores de domínio mais fortes para resolver esse problema

Um par de nós (X_i, X_j) é dito possuir **consistência de caminho** levando em conta uma variável X_k se para todos os possíveis valores $X_i = a, X_j = b$ dentro de seus domínios, existe pelo menos um valor $X_k = c$ que não viola nenhuma restrição em (X_i, X_k) e (X_k, X_j) .

Se pensarmos no problema de colorir o mapa com apenas duas cores, para tornar as variáveis (WA, SA) consistentes no caminho com a variável NT, verificamos as duas possibilidades de atribuição para eles:

$\{WA = \textit{vermelho}, SA = \textit{azul}\}, \{WA = \textit{azul}, SA = \textit{vermelho}\}$

Qualquer um dos valores de NT violam suas restrições em qualquer uma das condições, tornando o domínio $D_{NT} = \emptyset$, mostrando que o problema não apresenta solução.

Nesse caso, ambas possíveis atribuições são removidas.

```
def PC2(csp, queue):
    if empty(queue):
        return true
    i,j,k = queue.pop()
    revised, csp = revise3(i,j,k)
    if revised:
        X, D, C = csp
        if empty(D[k]):
            return false
        queue.push([(l,i,j) for l in range(n)
                    if l != i and l != j])
        queue.push([(l,j,i) for l in range(n)
                    if l != i and l != j])
    return PC2(csp, queue)
```

```
Q = [(i,j,k) for i in range(n)
      for j in range(i+1,n)
      for k in range(n)
      if k != i and k != j]
```

```
PC2(csp, Q)
```

Exercício

Considere o Sudoku ilustrado abaixo. Dado as restrições do tipo *todosDiferentes* envolvendo as variáveis de uma mesma linha, de uma mesma coluna e de um mesmo bloco, transforme todos os domínios em arco-consistente:

1	2	3	4
3		1	2
2			
	1		3

Backtracking (Retrocesso)

Backtracking (Retracement)

O algoritmo **backtracking** é o algoritmo básico de busca não informada para um CSP.

Basicamente é uma busca em profundidade levando em conta:

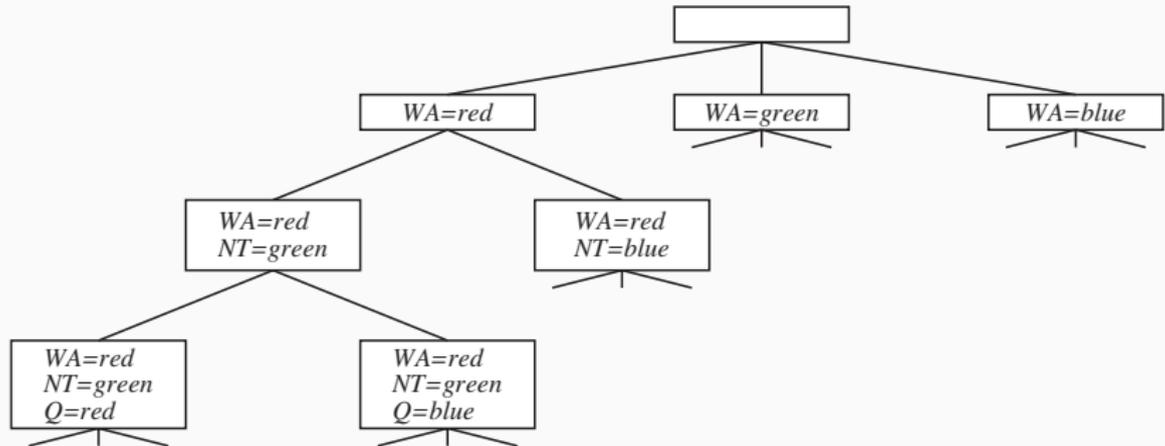
- Cada nó atribui valores para apenas uma variável
- Não atribui valores que conflitam com alguma restrição
- Para de expandir um nó ao atingir uma solução infactível

Backtracking

```
def Backtracking(sol, csp):
    X, D, C = csp
    if length(sol) == length(X):
        return sol
    i = selezioneVar(csp, sol)
    for v in sortValues(i, sol, csp):
        if consistent(i, v, csp):
            sol_p = Backtracking(sol + [(i,v)], csp)
            if not empty(sol_p):
                return sol_p
    return []
```

Backtracking

No nosso problema de colorir mapas:



A função `selecioneVar` em sua forma mais simples seleciona cada variável do problema na ordem.

Porém uma estratégia é escolher a variável que tem a menor quantidade de valores válidos para a solução parcial atual.

No nosso problema, se fizermos $WA = \textit{vermelho}$ e $NT = \textit{verde}$, SA só poder ter o valor *azul*, e as variáveis Q, NSW, V poderão assumir apenas um único valor, levando a uma solução de forma mais rápida.

Outra heurística possível é escolher a variável que faz parte de um maior número de restrições (variável com maior grau), pois uma vez que um valor é designado para ela, muitas opções de valores serão eliminadas ao mesmo tempo.

Uma vez escolhida a variável, uma boa heurística para escolha de valores é começar por aqueles valores que restringem menos os valores de seus vizinhos.

Suponha $WA = \textit{vermelho}$, $NT = \textit{verde}$, e agora queremos dar um valor a Q . A escolha de $Q = \textit{azul}$ elimina o último valor legal para seu vizinho SA , portanto essa heurística exploraria primeiro *vermelho*.

Note que a escolha da variável deve ser a que “falha-primeiro”, para resolvermos os problemas mais críticos primeiro.

Por outro lado, a escolha dos valores deve seguir “falha-por-último”, pois queremos evitar de não satisfazer nenhum de seus vizinhos.

Uma outra forma de tentar resolver um problema CSP é através de algoritmos de **busca local**, tais algoritmos partem de uma solução completa e inactível e tentam factibilizar iterativamente.

```
def HillClimbing(sol, csp, it):  
    if it == 0:  
        return []  
    i, v = minConflito(sol, csp)  
    sol[i] = v  
    if solved( sol, csp ):  
        return sol  
    return HillClimbing(sol, csp, it-1)
```

Aplique o algoritmo de Backtracking e o HillClimbing para o problema de colorir o mapa. Inicie a solução de HillClimbing com todas as cores vermelhas.