

Inteligência Artificial

Problemas de Satisfação de Restrições



Sumário

- Definição e Exemplos
- Busca com Retrocesso (*Backtracking*)
- Busca Local para PSRs



Problemas de Satisfação de Restrições

Definição e Exemplos



Definição

- Um problema de satisfação de Restrições (PSR) é definido como:
 - Um conjunto de variáveis X_1, X_2, \dots, X_n
 - Cada variável possui um domínio não vazio de valores possíveis
 - Um conjunto de restrições C_1, C_2, \dots, C_m
 - Cada restrição envolve algum subconjunto de variáveis e especifica combinações possíveis de valores para aquele conjunto



Definição

- Um estado em um PSR
 - É um assinalamento de valores para algumas ou todas as variáveis X_i
 - Um assinalamento que não viola nenhuma restrição é chamado **consistente**
 - Um assinalamento que envolve todas as variáveis é chamado **completo**
 - Uma solução é um assinalamento **completo** que seja **consistente**



Exemplo 1

Colorindo um Mapa

- Colorir cada um dos estados e territórios da Austrália, usando as cores *verde*, *vermelho* ou *azul* sem que regiões vizinhas tenham a mesma cor



Exemplo 1

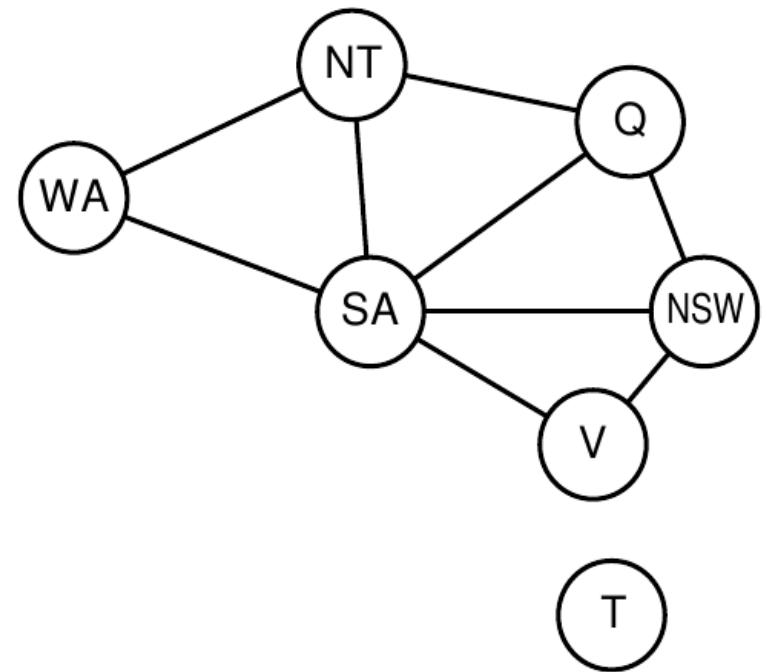
Colorindo um Mapa

- Variáveis
 - $X = \{WA, NT, SA, Q, NSW, V, T\}$
- Domínio
 - $D = \{\text{verde, vermelho, azul}\}$
- Restrição
 - $X_i \neq X_j$ para X_i e X_j adjacentes



Grafo de Restrições

- Restrições binárias
 - Vértices: Variáveis
 - Arestas: Restrições
- Algoritmos genéricos podem utilizar informação do grafo para agilizar a busca
 - A Tasmânia é um sub-problema!

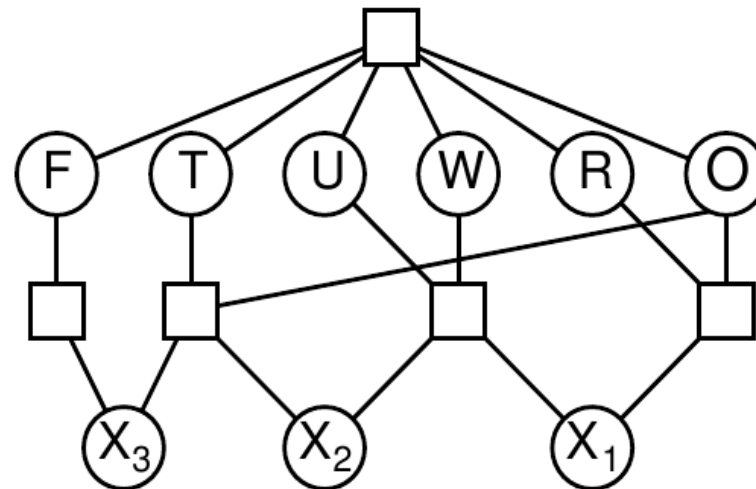


Exemplo 2

Cryptarithmic puzzle

- Encontrar um valor para as letras de forma que a soma correspondente seja aritmeticamente correta

$$\begin{array}{r} \text{T W O} \\ + \text{T W O} \\ \hline \text{F O U R} \end{array}$$

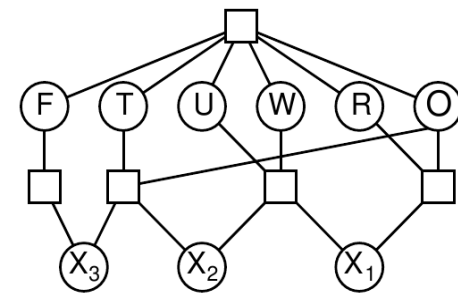


Exemplo 2

Cryptarithmic puzzle

- Variáveis
 - $X = \{F, T, U, W, R, O, X_1, X_2, X_3\}$
- Domínio
 - $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Restrições
 - Todas as letras com valores diferentes
 - $O + O = R + 10 \times X_1$
 - $X_1 + W + W = U + 10 \times X_2$
 - $X_2 + T + T = O + 10 \times X_3$
 - $X_3 = F$

$$\begin{array}{r}
 \text{TWO} \\
 + \text{TWO} \\
 \hline
 \text{FOUR}
 \end{array}$$



Tipos de PSRs

- Variáveis discretas
 - Domínios Finitos
 - Variáveis Booleanas: Problemas de SAT
 - Domínios Infinitos
 - Escalonamento de Tarefas
- Variáveis Contínuas
 - Escalonamento do tempo de cada tarefa no telescópio Hubble
 - Com PSR, reduziu-se o tempo de cálculo de 3 semanas para 10 minutos!



Tipos de Restrições

- Unárias
 - $SA \neq \text{verde}$
- Binárias
 - $SA \neq WA$
- Ordens mais altas
 - Envolvem 3 ou mais variáveis
 - Pode-se provar que podem ser modeladas com restrições binárias
- Preferências
 - Vermelho é melhor do que verde
 - Otimização com restrições



PSRs Reais

- Encargos didáticos
 - Quem dá aula em qual turma
- Horários de Aula
 - Qual horário e local de cada aula
- Floorplaning
 - Projetos VLSI
- Verificação de Circuitos Digitais
 - Problemas de SAT



Formulação Incremental de PSRs

- Inicia-se com um estado vazio {}
 - Nenhuma variável está assinalada
- Função sucessor
 - Um valor pode ser assinalado a qualquer variável não assinalada, respeitando-se às restrições
- Teste do objetivo
 - O estado atual possui um assinalamento completo?
- Custo do caminho
 - Custo constante para cada passo



Formulação Incremental de PSRs

- Vantagens
 - Formulação é igual para qualquer PSR!
 - Algoritmos genéricos
 - Todas as soluções ocorrem no nível n em problemas de n variáveis
 - Pode-se utilizar busca em profundidade!
 - Boa utilização de memória
 - Solução ótima, pois todas soluções estão no mesmo nível
- Desvantagens
 - $b = (n-l)d \rightarrow n!d^n$ folhas
 - l é o nível atual e d é o número de símbolos no domínio



Problemas de satisfação de Restrições

Busca com Retrocesso
(*Backtracking*)



Questão

- Porque foram geradas $n!d^n$ folhas na busca, se somente existem d^n assinalamentos completos?
- Porque a formulação foi ingênua
 - A ordem em que as restrições são aplicadas é irrelevante!
 - Em cada nó, o assinalamento de apenas uma variável é testado
 - Na raiz pode-se escolher entre SA = vermelho, SA = verde ou SA = azul
 - Nunca entre SA = verde e WA = azul



Busca com Retrocesso

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment given CONSTRAINTS[csp] then
      add { var = value } to assignment
      result ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failure then return result
      remove { var = value } from assignment
  return failure
```

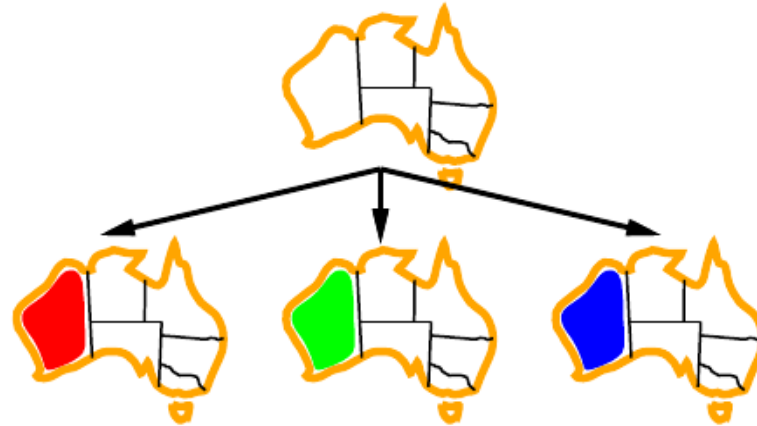


Exemplo

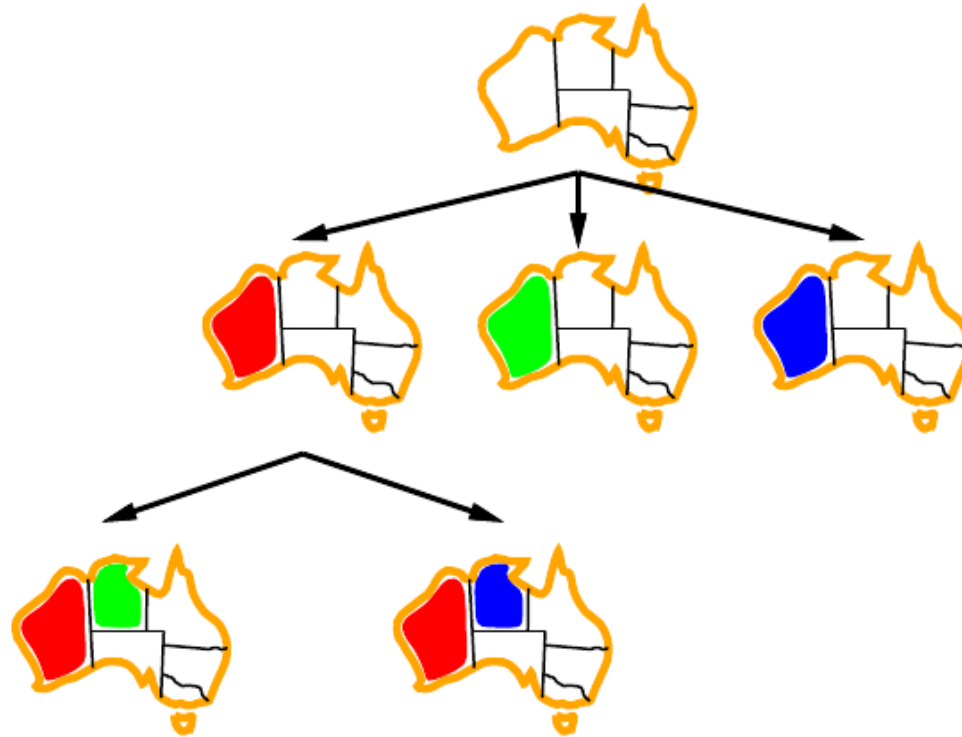
Colorindo um Mapa



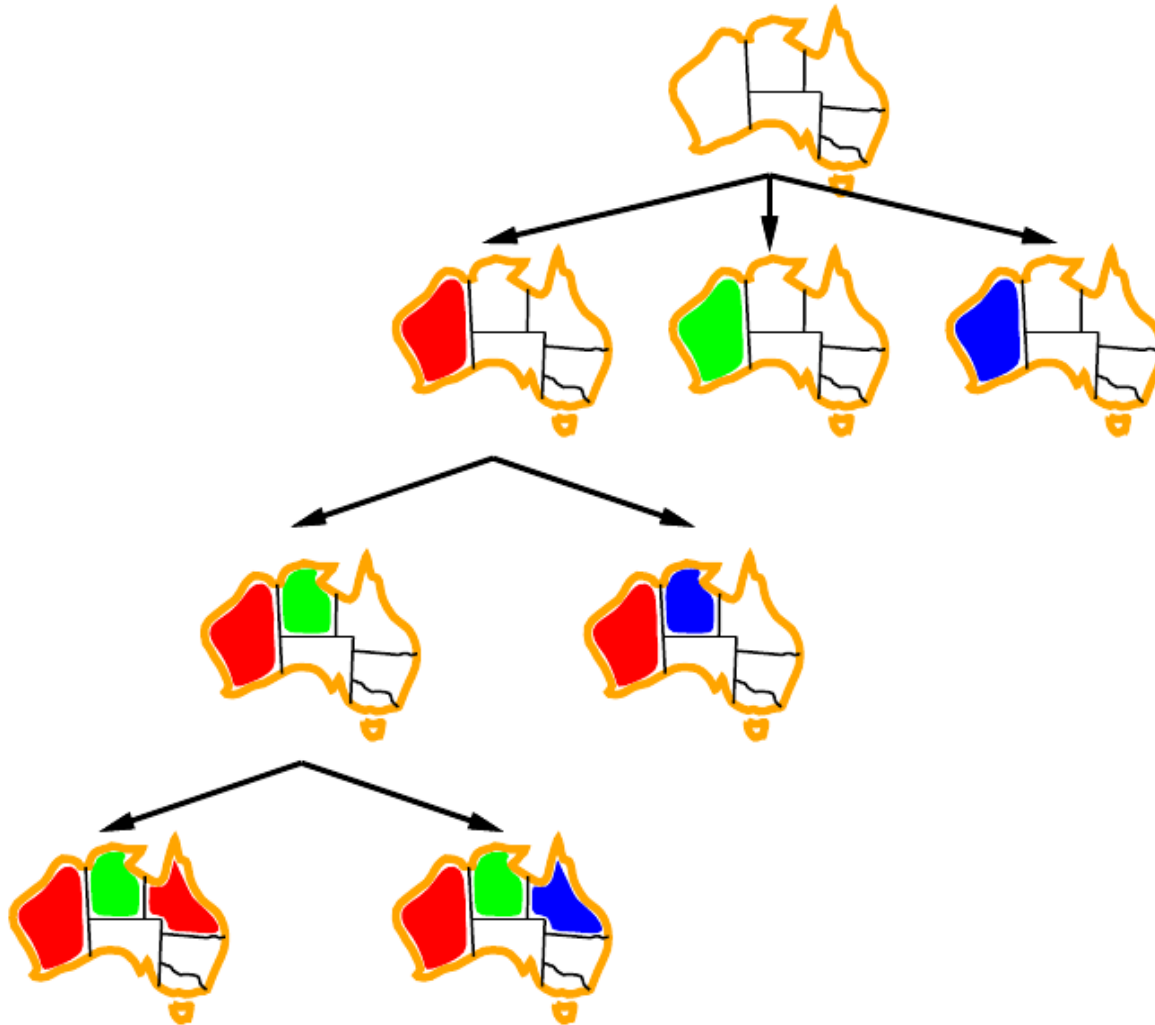
Exemplo Colorindo um Mapa



Exemplo Colorindo um Mapa



Exemplo Colorindo um Mapa



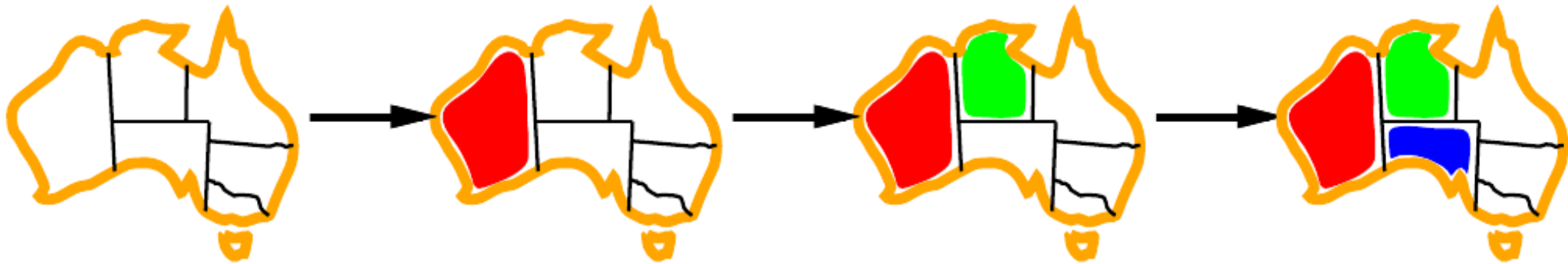
Aumentando a Eficiência

- Qual variável deve ser assinalada em seguida?
- Em qual ordem os valores devem ser tentados?
- Uma falha inevitável pode ser detectada precocemente?
- Pode-se tirar vantagem da estrutura do problema?



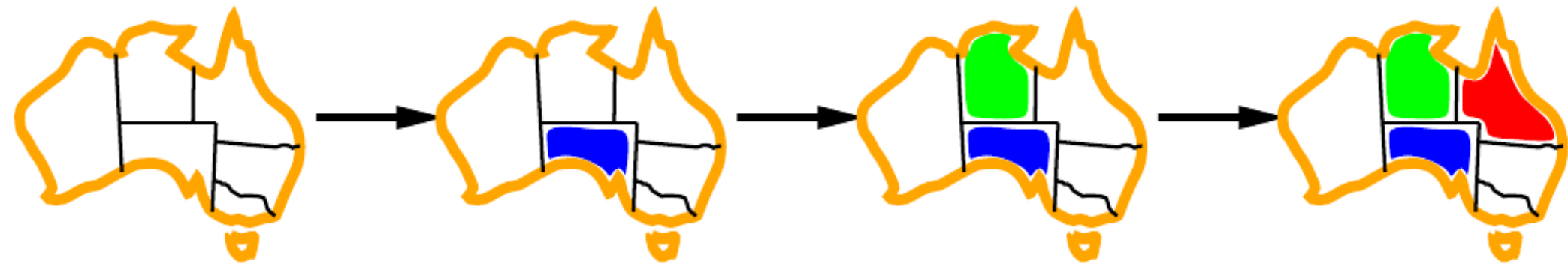
Escolha da Variável

- Menos Valores Restantes (*MRV*)
 - Escolhe a variável com menos valores legais



Heurística do Grau

- Em caso de empate, escolhe a variável com o maior número de restrições



Valor Menos Restritivo

- Escolhe o valor que menos restringe os assinalamentos nas variáveis restantes



Detectando Falhas

Forward Checking



WA

NT

Q

NSW

V

SA

T



Detectando Falhas Forward Checking



WA

NT









































Q

NSW

V

SA

T



Detectando Falhas Forward Checking



WA

NT

Q

NSW

V

SA

T



Detectando Falhas Forward Checking



WA

NT



















































































Q

NSW

V

SA

T



Propagação de Restrições

- *Forward Checking* não “olha” muito a frente, e pode não detectar inconsistências
 - NT e SA não podem ser azuis!

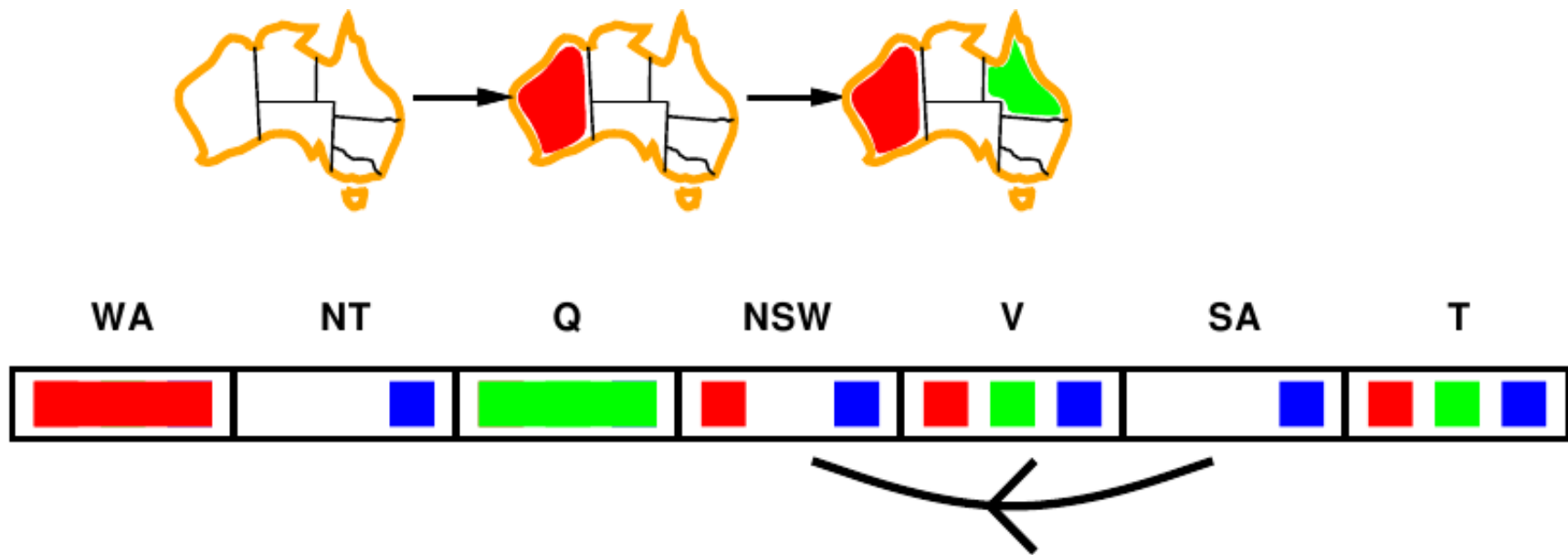


WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>



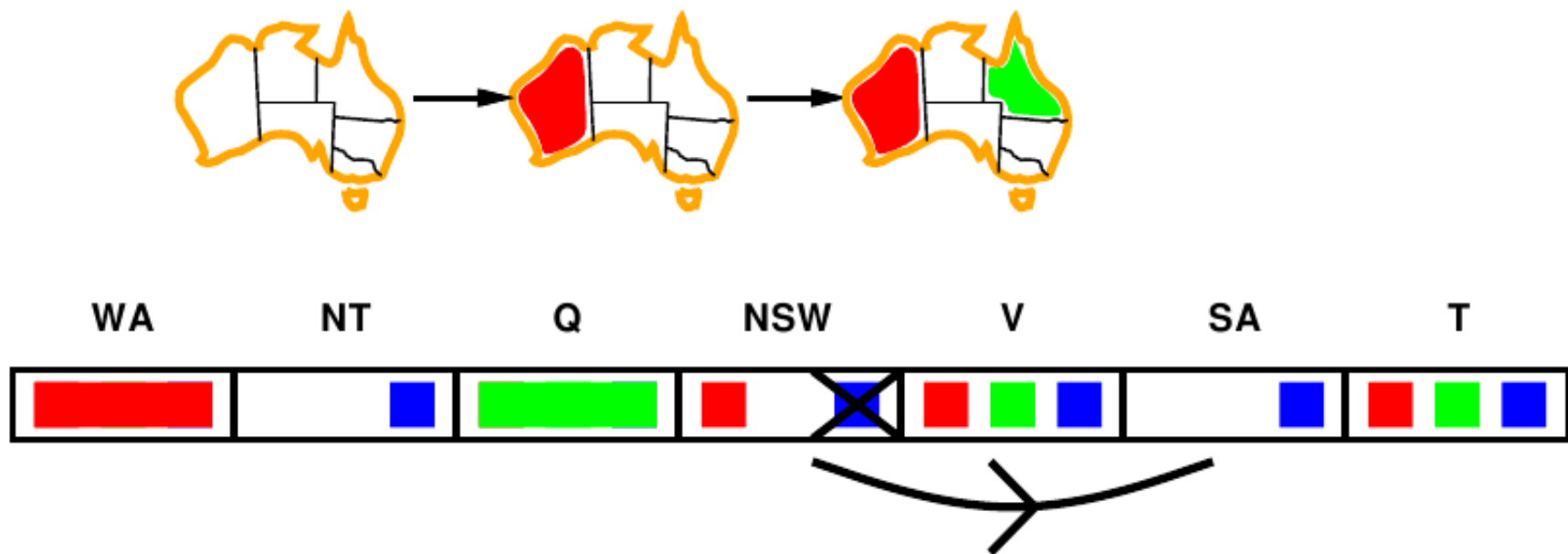
Propagação de Restrições

- Consistência de Arco



Propagação de Restrições

- Consistência de Arco

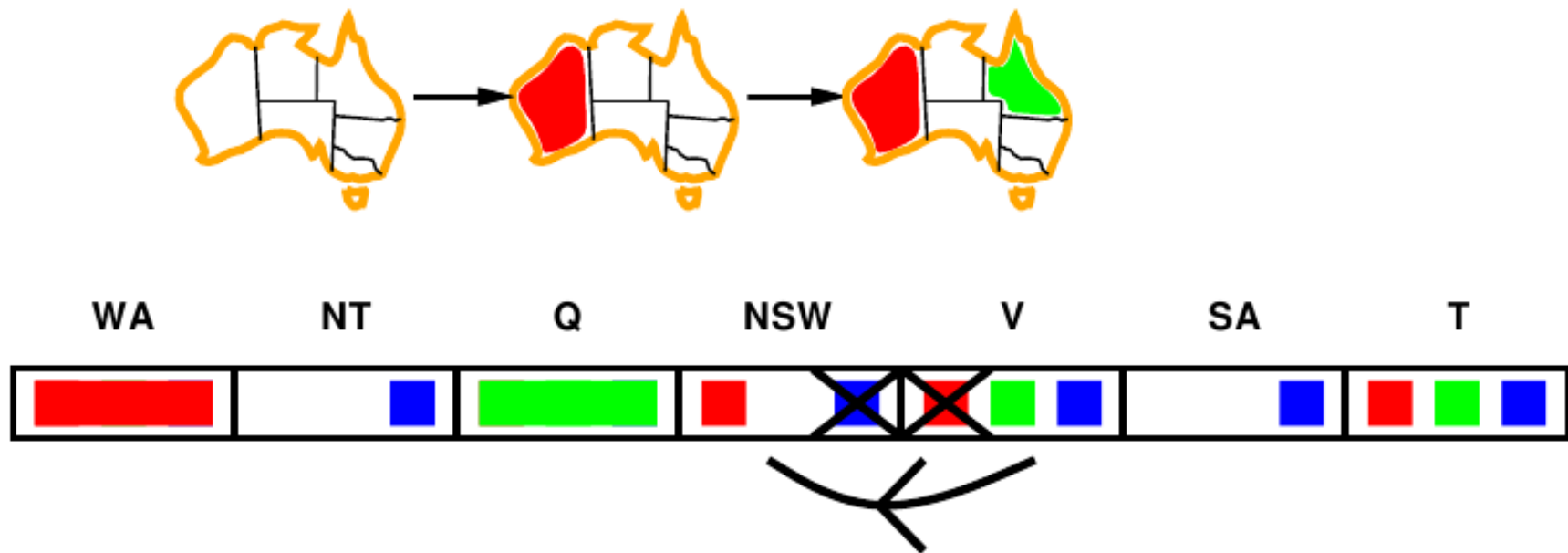


- Remove valores não válidos dos domínios das variáveis



Propagação de Restrições

- Consistência de Arco

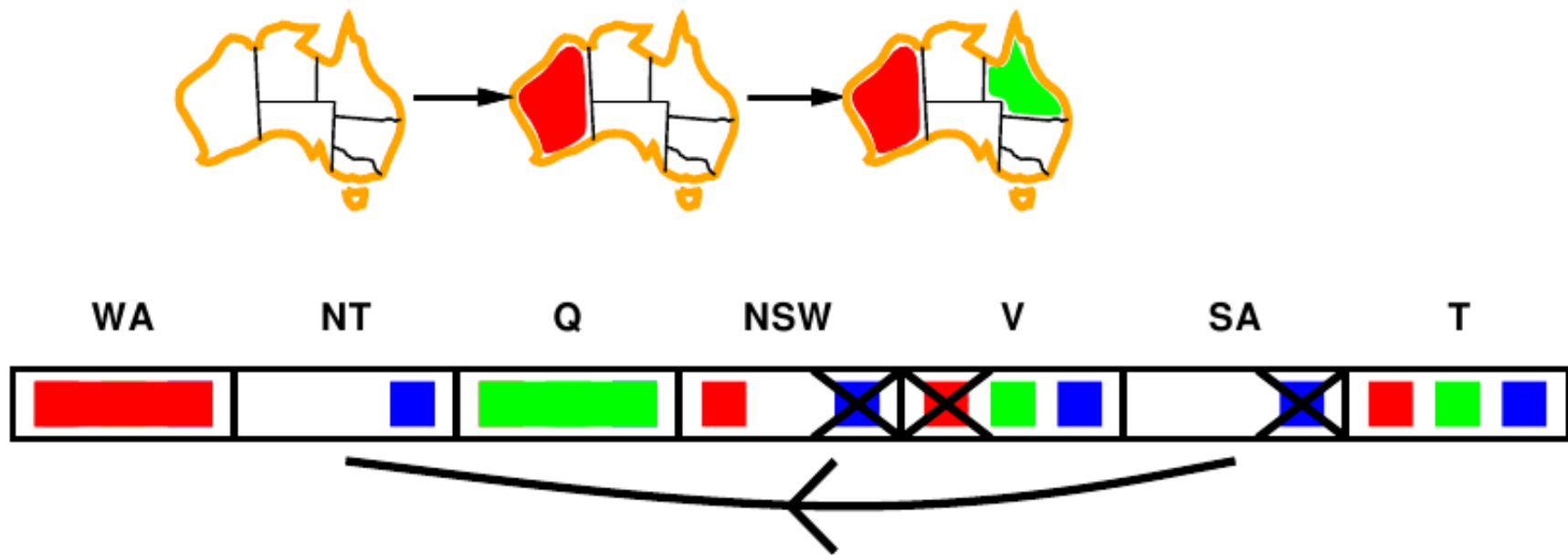


- Se um vértice perde um valor, seus vizinhos devem ser reverificados



Propagação de Restrições

- Consistência de Arco



- Detecta inconsistências antes do *Forward Checking*
 - Pode ser utilizado como pré-processamento ou depois de cada assinalamento



Algoritmo AC-3

```
function AC-3(csp) returns the CSP, possibly with reduced domains
inputs: csp, a binary CSP with variables  $\{X_1, X_2, \dots, X_n\}$ 
local variables: queue, a queue of arcs, initially all the arcs in csp

while queue is not empty do
     $(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\textit{queue})$ 
    if REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) then
        for each  $X_k$  in NEIGHBORS[ $X_i$ ] do
            add  $(X_k, X_i)$  to queue



---


function REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) returns true iff succeeds
    removed  $\leftarrow$  false
    for each  $x$  in DOMAIN[ $X_i$ ] do
        if no value  $y$  in DOMAIN[ $X_j$ ] allows  $(x, y)$  to satisfy the constraint  $X_i \leftrightarrow X_j$ 
            then delete  $x$  from DOMAIN[ $X_i$ ]; removed  $\leftarrow$  true
    return removed
```



Busca Local Para PSR

- A formulação anterior era incremental
 - Iniciava-se com nenhuma variável assinalada
 - Assinalava-se uma variável de cada vez
- Outra formulação trabalha com o estado inicial onde todas as variáveis são assinaladas
 - A função sucessor retorna estados vizinhos, modificando-se o valor de uma variável



Algoritmo Min-Conflitos

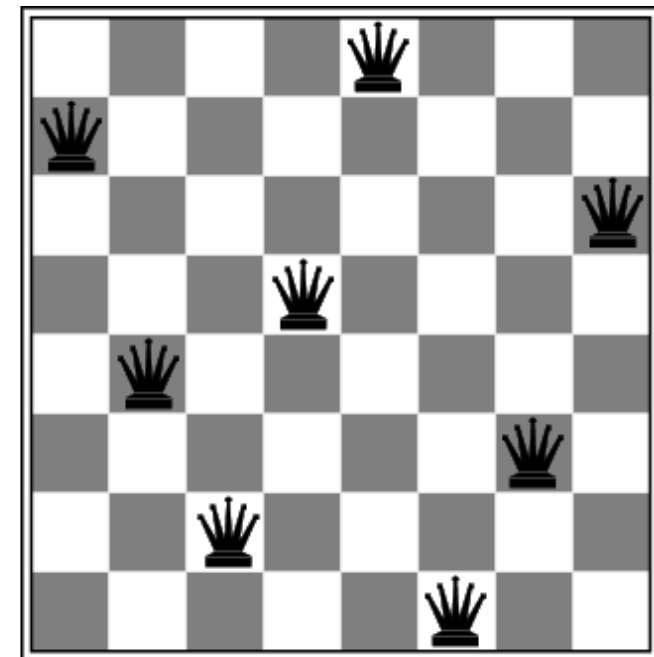
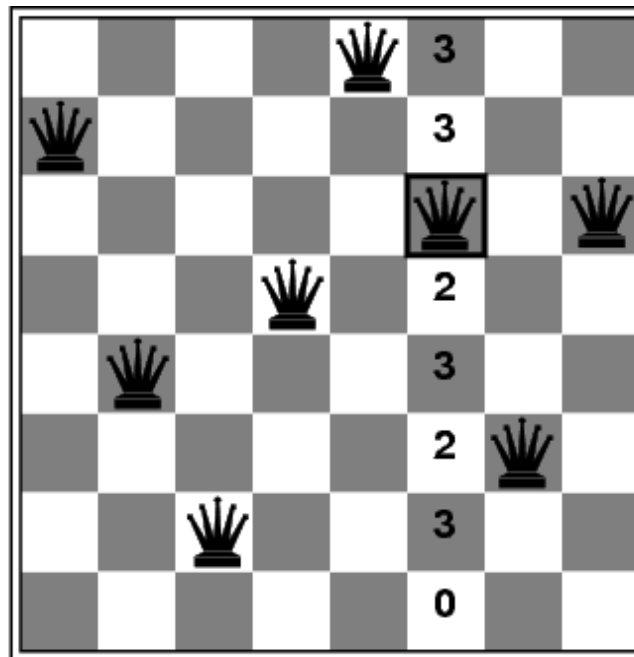
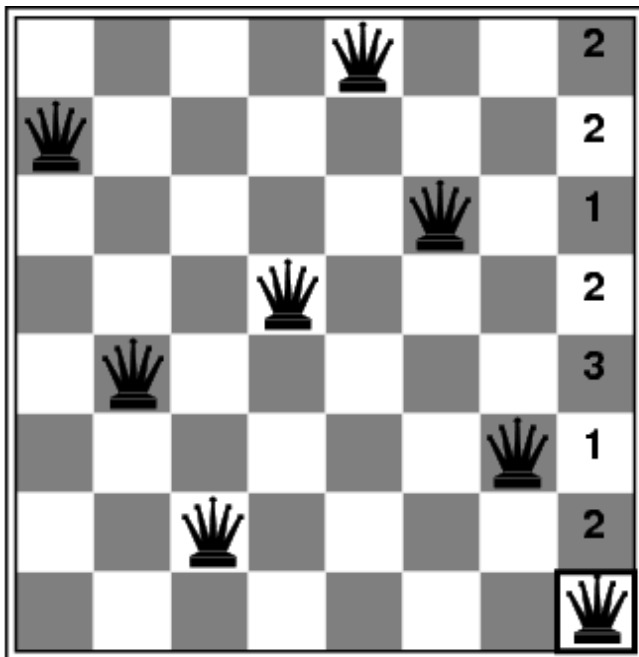
```
função MIN-CONFLITOS(psr, max_passos) retorna uma solução ou falha
  entradas: psr, um problema de satisfação de restrições
           max_passos, número máximo de passos antes de desistir

  atual ← um assinalamento aleatório para o psr
  Para i = 0 até max_passos faça
    se atual é uma solução para o psr então retorne atual
    var ← uma variável escolhida aleatoriamente de VARIÁVEIS[psr]
    valor ← o valor v para var que minimiza CONFLITOS(var, v, atual, psr)
    var ← valor

  retorne falha
```



Exemplo de Utilização



Algoritmo queen_search

```
1.  function queen_search(queen : array [1..n] of integer)
2.  begin
3.      repeat
4.          Generate a random permutation of queen1 to queenn;
5.          forall i, j; where queeni or queenj is attacked do
6.              if swap(queeni, queenj) reduces collisions
7.              then perform_swap(queeni, queenj);
8.          until no collisions;
9.  end;
```

```
1.  repeat
2.      swaps_performed := 0;
3.      for i in [1..n] do
4.          for j in [(i + 1)..n] do
5.              if queeni is attacked or queenj is attacked then
6.                  if swap(queeni, queenj) reduces collisions then begin
7.                      perform_swap(queeni, queenj);
8.                      swaps_performed := swaps_performed + 1;
9.                  end;
10. until swaps_performed = 0;
```

