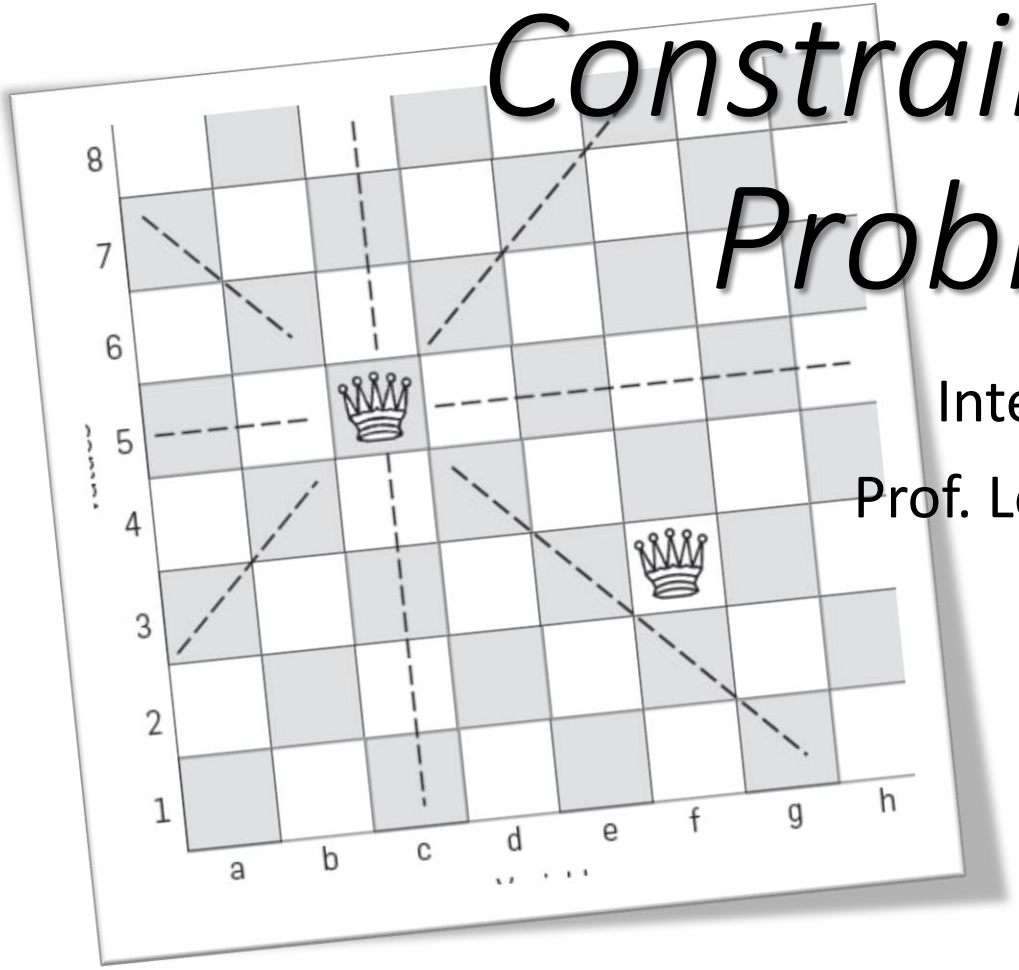
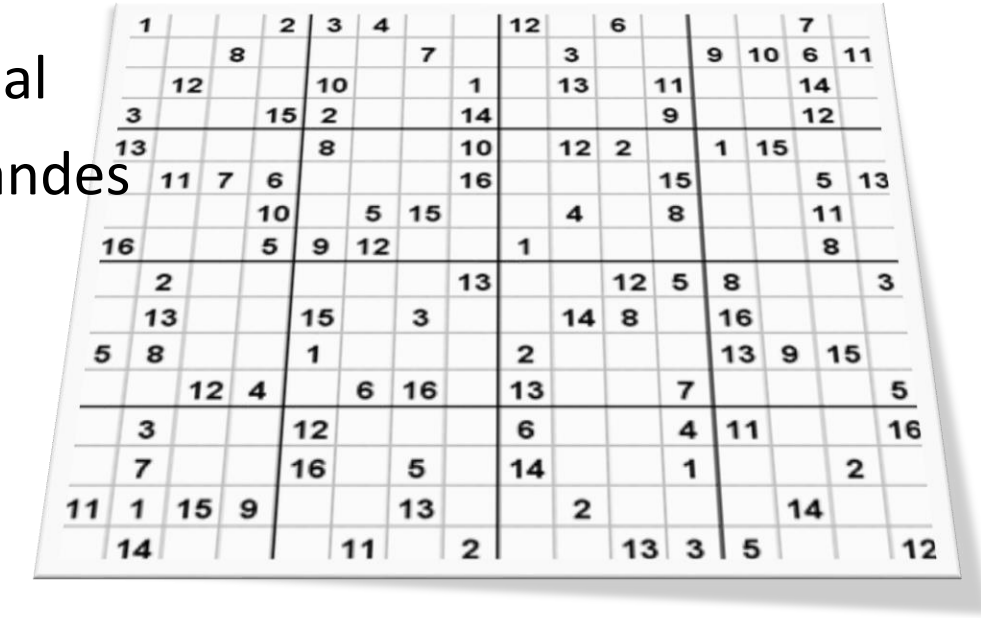
Constraint Satisfaction Problems – CSP



Inteligência Artificial

Prof. Leandro C. Fernandes

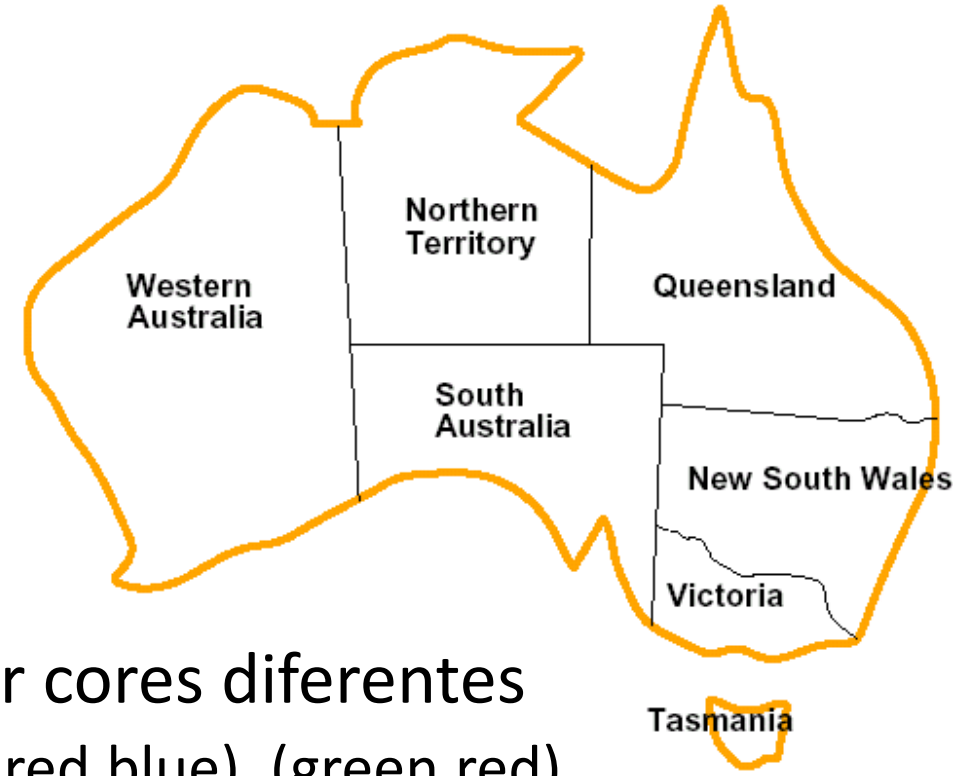


Problemas de Satisfação de Restrições – CSPs

- Problema de Busca Padrão:
  - estado é uma caixa preta “*black box*”
  - qualquer estrutura que suporte a função sucessor, heurísticas, e
  - teste de objetivo.

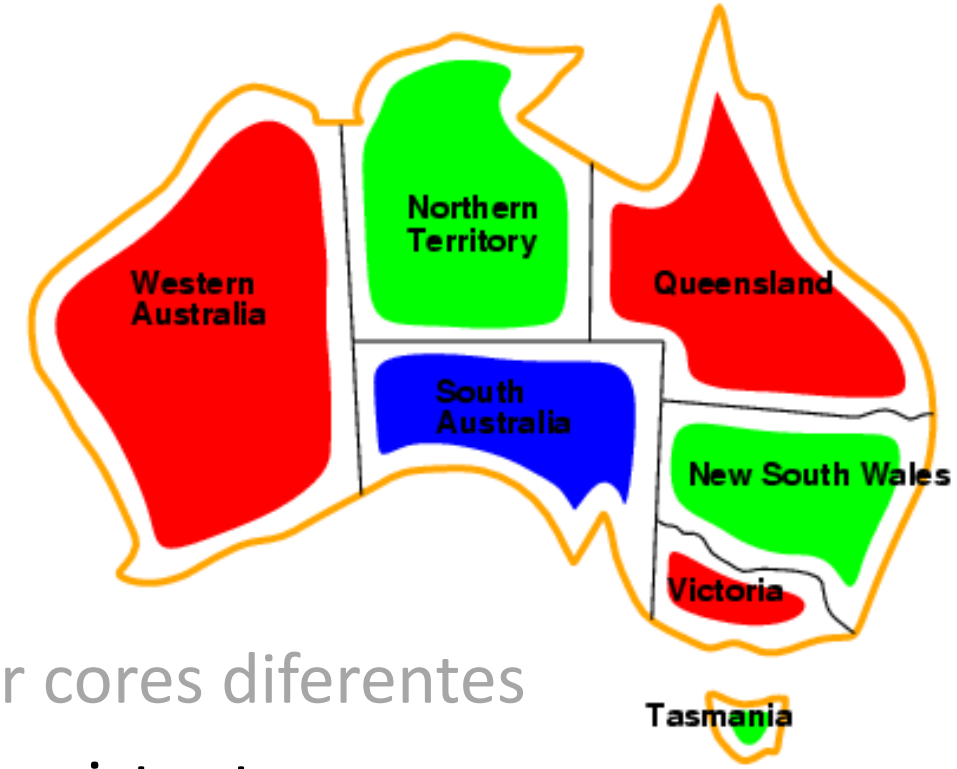
- CSP:
  - estado é definido por variáveis Xi com valores de um domínio Di
  - teste de objetivo é um conjunto de restrições especificando combinações permitidas de valores para as variáveis
- Representa uma linguagem formal de representação
- Permite algoritmos úteis de propósito geral com mais poder que algoritmos de busca padrão

Exemplo: Coloração de Mapas



- **Variáveis:** WA, NT, Q, NSW, V, SA, T
- **Domínio:** Di = {*red*, *green*, *blue*}
- **Restrições:** regiões adjacentes devem ter cores diferentes
  - Ex: WA ≠ NT, ou (WA,NT) em { (red,green), (red,blue), (green,red), (green,blue), (blue,red), (blue,green) }

Exemplo: Coloração de Mapas

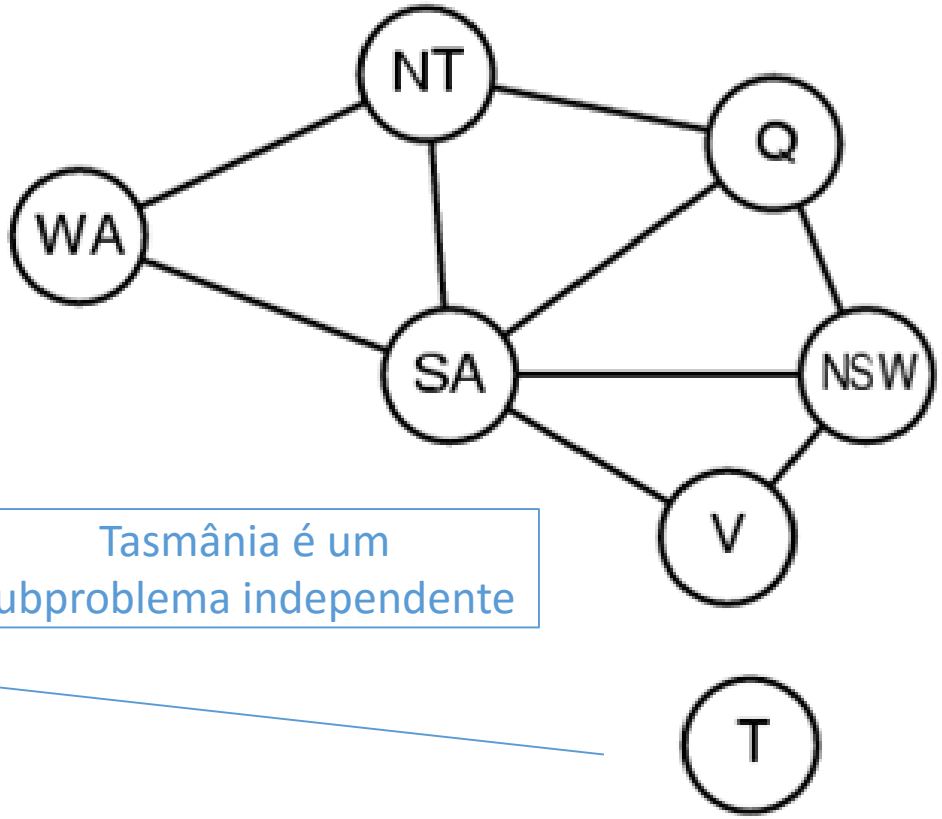


- **Variáveis:** WA, NT, Q, NSW, V, SA, T
- **Domínio:** Di = {*red*, *green*, *blue*}
- **Restrições:** regiões adjacentes devem ter cores diferentes
- **Soluções:** são atribuições completas e consistentes
  - Ex: WA = *red*, NT = *green*, Q = *red*, NSW = *green*, V = *red*, SA = *blue*, T = *green*

Grafo de Restrições

- **CSP Binário:** cada restrição relaciona duas variáveis.

- **Grafo de restrições** representa um CSP Binário:
  - nodos são variáveis, arcos são restrições



Tasmânia é um subproblema independente

Variáveis em CSPs

- Discretas com Domínio Finitos:
  - *n* variáveis, tamanho do domínio *d*
  - há O(d^n) atribuições completas
    - Ex.: CSP Booleanos, inclusive Satisfabilidade Booleana (NP-Completo)

- Discretas com Domínio Infinitos:
  - inteiros, strings, etc.
  - Ex: escalonamento de tarefas, agendamentos
    - Variáveis são o início/fim para cada tarefa
    - Precisa de uma linguagem de restrição, ex: *StartJob1* + 5 ≤ *StartJob3*

Variáveis em CSPs

- Variáveis Contínuas
  - Ex: tempo de inicio/fim para observações com o telescópio Hubble
  - restrições lineares são resolvíveis em tempo polinomial com programação linear

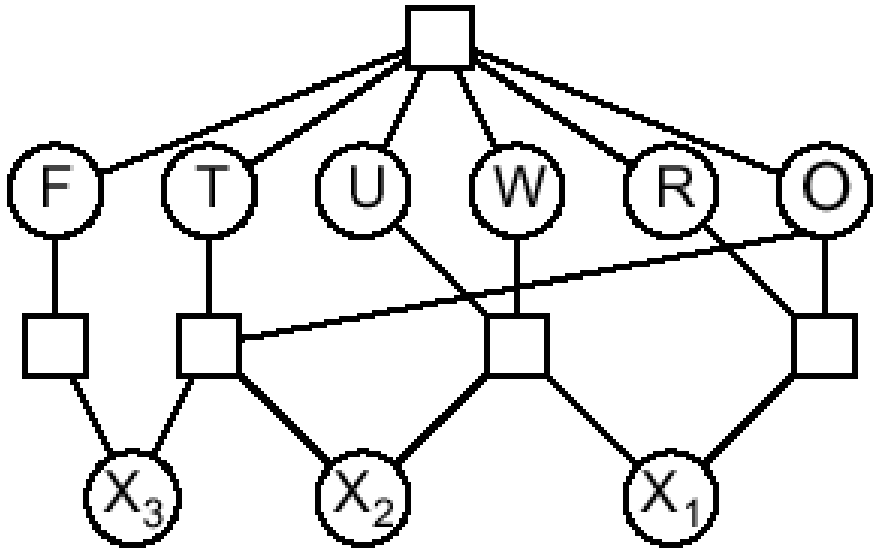
Tipos de Restrições

- Unárias: envolvem uma única variável
  - ex. SA ≠ *green*
- Binárias: envolvem pares de variáveis
  - ex. SA ≠ WA
- N-árias (alta ordem): envolvem 3 ou mais variáveis
  - ex. Restrições de cryptarithmetica
- Preferenciais: considera um fator de prioridade ou custo
  - ex. Vermelho é melhor do que verde, isto é, há uma função de custo nas atribuições.

Exemplo: Cryptarithmetica

```
  T W O
+ T W O
-------
F O U R
```

- Problema:
  - Associar a cada letra um dígito diferente tal que, substituindo as letras pelos seus dígitos associados, a adição esteja aritmeticamente correta.
- Formulação em CSP:
  - Variáveis: F, T, U, W, R, O, X1, X2, X3
  - Domínio: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
  - Restrições: *alldiff*(F, T, U, W, R, O)
    - O + O = R + 10 . X1
    - X1 + W + W = U + 10 . X2
    - X2 + T + T = O + 10 . X3
    - X3 = F



Problemas Reais de CSPs

- Problemas de Alocação
  - *Ex: quem ministrará qual disciplina?*
- Problemas de Oferta de Disciplinas
  - *Ex: quando e onde cada disciplina será oferecida?*
- Configuração de Equipamentos
- Escalonamento em Fábrica
- Logística de Transporte
  - *Geralmente problemas envolvem variáveis com valores reais!*

Formulação de busca padrão (Incremental)

Formulação de busca padrão (incremental)

Abordagem básica:

- Estados são definidos pelos valores atribuídos até o momento
- **Estado inicial:** atribuição vazia { }
- **Função Sucessor:** atribui um valor para uma variável não atribuída, sem entrar em conflito com as atribuições existentes
  - Falha, caso não haja atribuições legais
- **Teste de objetivo:** a atribuição está completa.
- **Caminho de Custo:** um valor constante para todo passo.

Formulação de busca padrão (incremental)

- Esta formulação serve para todos os CSPs
- Uma solução aparece no nível *n* com *n* variáveis
  - Busca em profundidade é uma boa escolha
- O caminho é irrelevante, então pode-se utilizar uma formulação de estado completo.
  - Cada estado é uma atribuição completa
- Fator de ramificação *b* decresce com profundidade no nó da árvore de busca
- O número de nodos na árvore de busca é de $O(n!d^n)$

Busca com *Backtracking*

Busca com *Backtracking*

Características

- As atribuições de variáveis são comutativas
  - [WA = *red* seguido de NT = *green*]
  - [NT = *green* seguido de WA = *red*]
- É considerada apenas a atribuição de uma variável por vez em cada nodo.
- Assim o número de nodos na árvore de busca é $O(d^n)$

Definição

- Busca em profundidade para CSPs com atribuição simples a variáveis.
- É o algoritmo de busca cega mais básico para CSPs.
- Pode resolver o problema das *n-queens* para $n \approx 25$

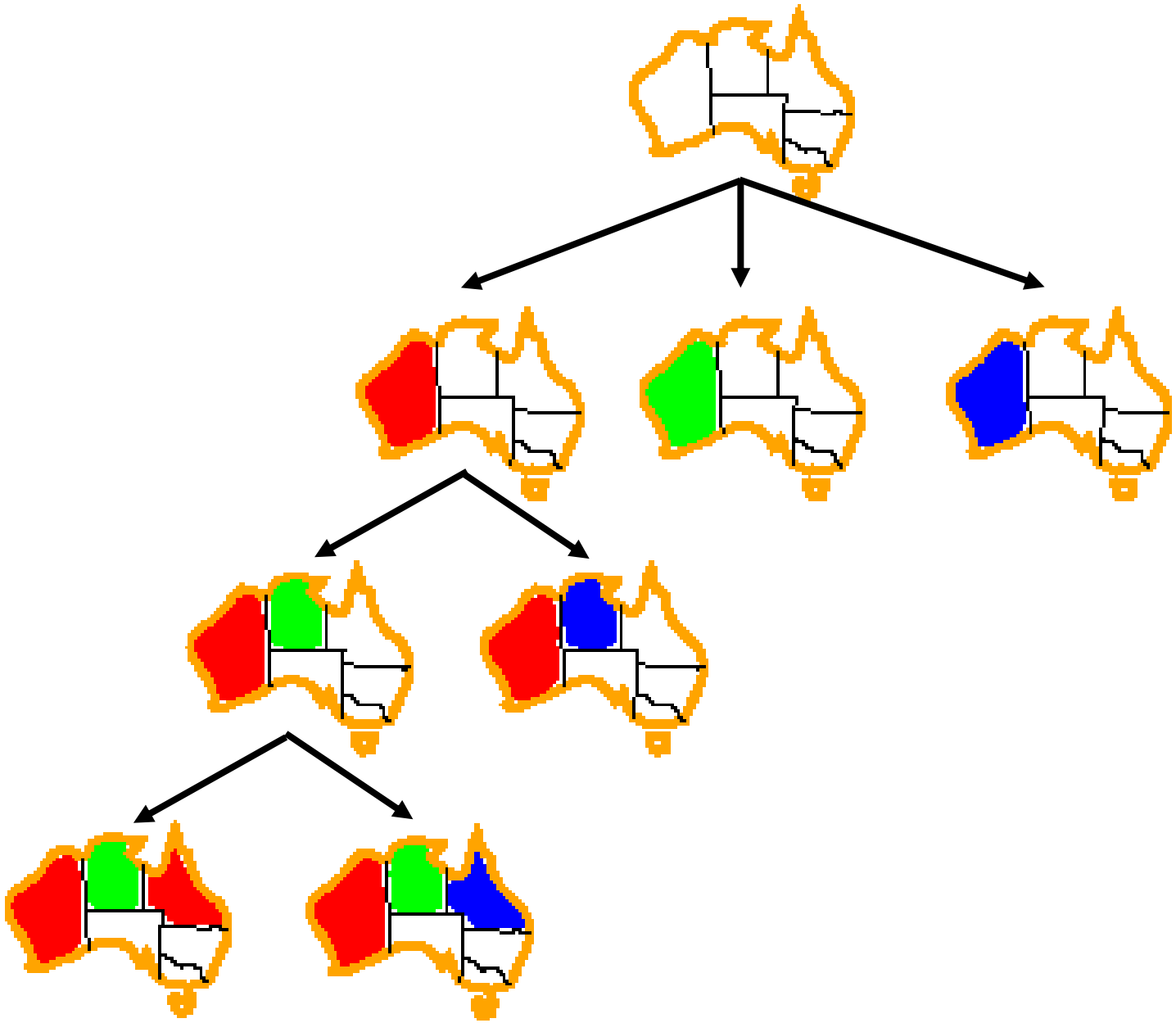
Busca com *Backtracking*

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

function RECURSIVE-BACKTRACKING( assignment,csp) returns a solution, or
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 according to Constraints[csp] then
            add { var = value } to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failue then return result
            remove { var = value } from assignment
    return failure
```

Busca com *Backtracking*

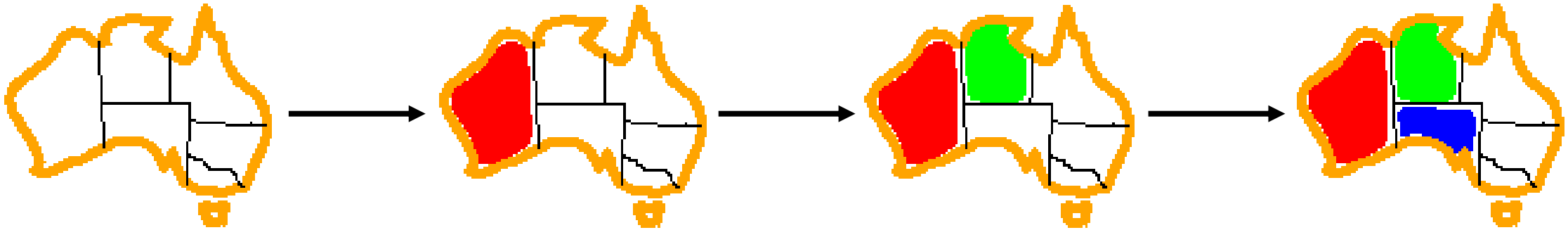
Processo de construção da árvore de busca para o problema de Coloração de Mapas



Melhorando a eficiência da busca com *backtracking*

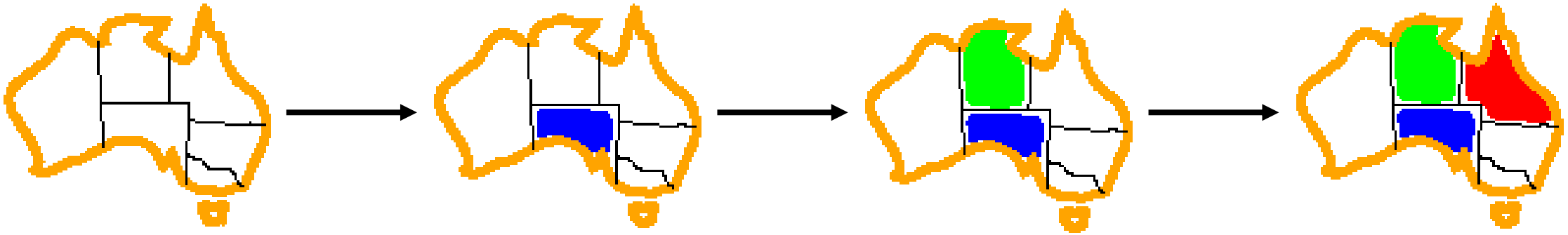
- Métodos de propósito geral podem oferecer ganhos significativos de velocidade e desempenho.
- Alguns aspectos que poderiam ser explorados …
  - Qual a próxima variável devemos atribuir?
  - Em qual ordem os valores devem ser tentados?
  - Podemos detectar uma falha inevitável cedo?
  - Podemos tirar vantagens da estrutura do problema?

Que variável será atribuída a seguir?



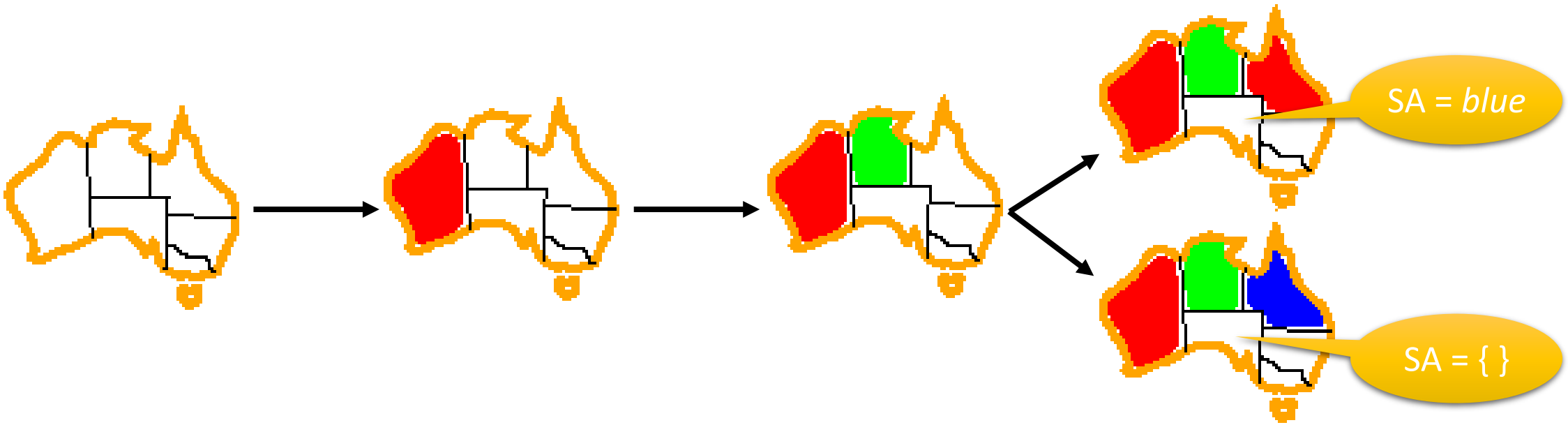
- A variável mais restrita (unária)
  - Escolha a variável com o menor número de valores legais
  - Abordagem também conhecida como heuristica dos Valores Remanescentes Mínimos – MRV (*Minimum Remaining Values*)

Que variável será atribuída a seguir?



- A variável mais restringida (binária)
  - Escolha a variável com o maior número de restrições das variáveis restantes.

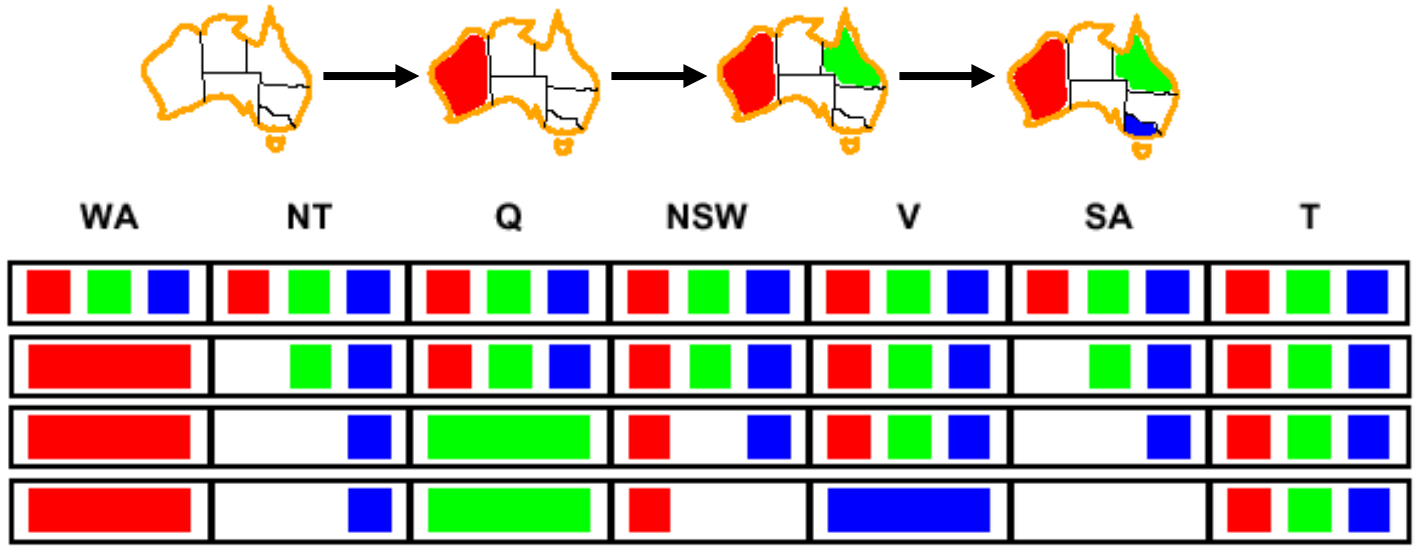
Em que ordem seus valores serão tentados?

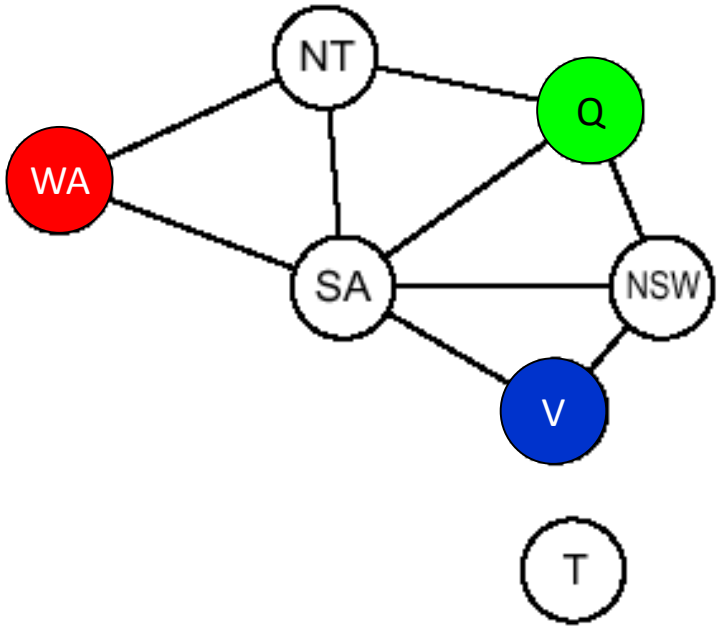


- Dada uma variável, escolha o valor menos restritivo:
  - O que descarta menos valores para as variáveis restantes, isto é, aquele que provoque o menor número de restrições possíveis nas variáveis restantes

Combinando estas duas heurísticas é possível resolver 1000 queens!!!

Em que ordem seus valores serão tentados?



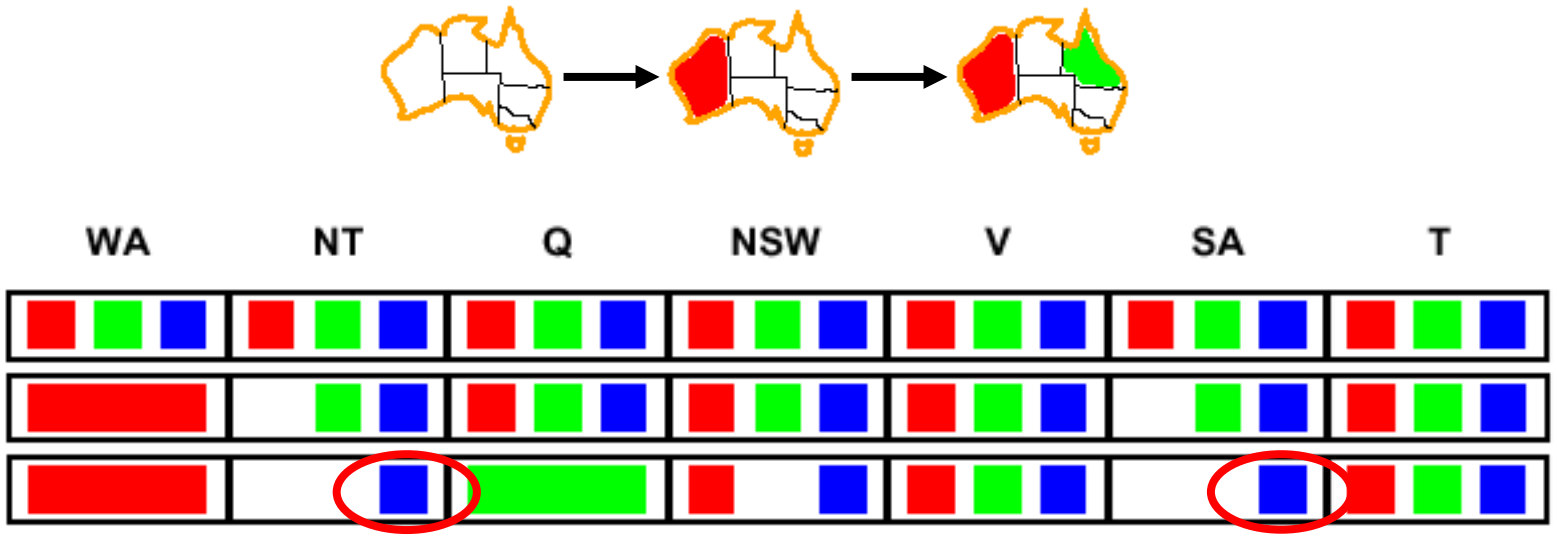


- Verificação Adiante (*Forward Checking*)

  Ideia:

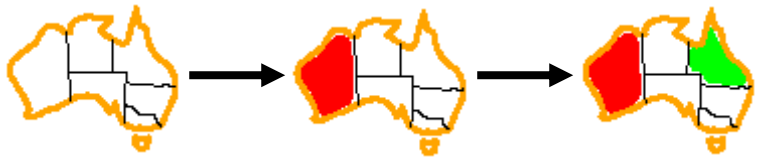
  - Mantenha registro dos valores legais que ainda podem ser utilizados para as variáveis não atribuídas
  - Encerre a busca quando qualquer variável não tiver valores legais

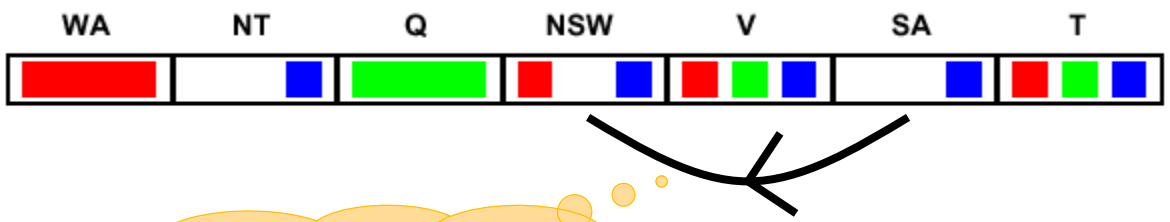
Propagação de Restrições



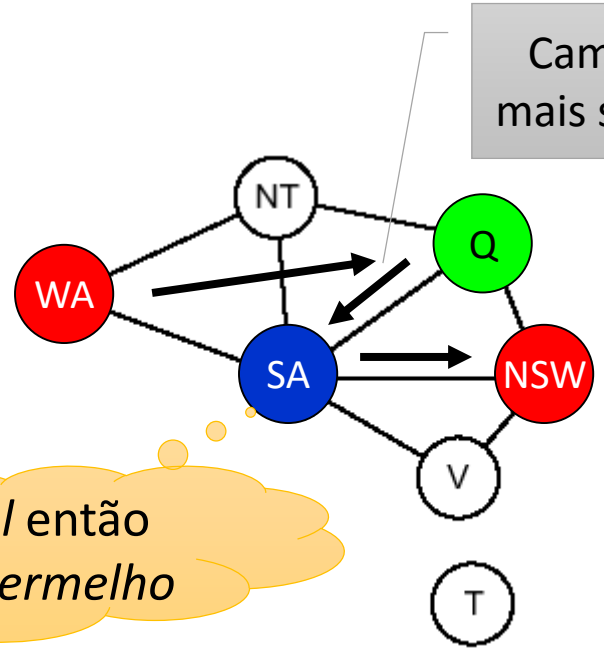
- Verificação adiante propaga informação das variáveis atribuídas para as não-atribuídas, mas não detecta cedo todas as falhas:
  - NT e SA não pode ser ambos azuis!
  - Propagação de restrição repetidamente reforçam as restrições localmente

É possível detectar falhas inevitáveis cedo?





Logo o arco de SA para NSW é **consistente!**



Se SA = *azul* então NSW = *vermelho*

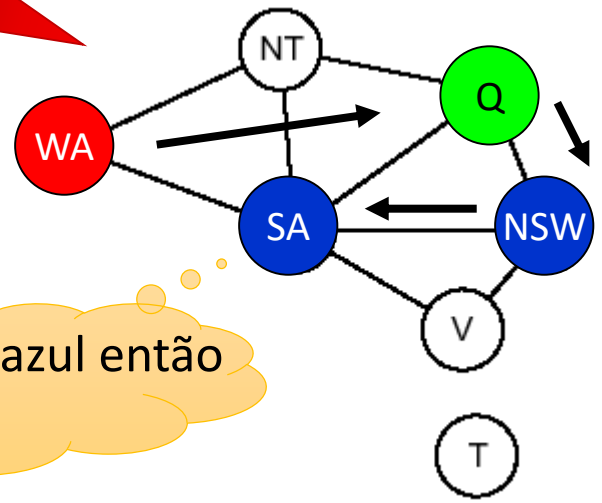
- Consistência de Arco:
  - A maneira mais simples de propagação torna cada arco consistente
  - X→Y é consistente sse para cada valor de x há algum valor permitido de y

É possível detectar falhas inevitáveis cedo?

Restrição:
SA ≠ NSW

Logo o arco de NSW para SA é **INconsistente!**

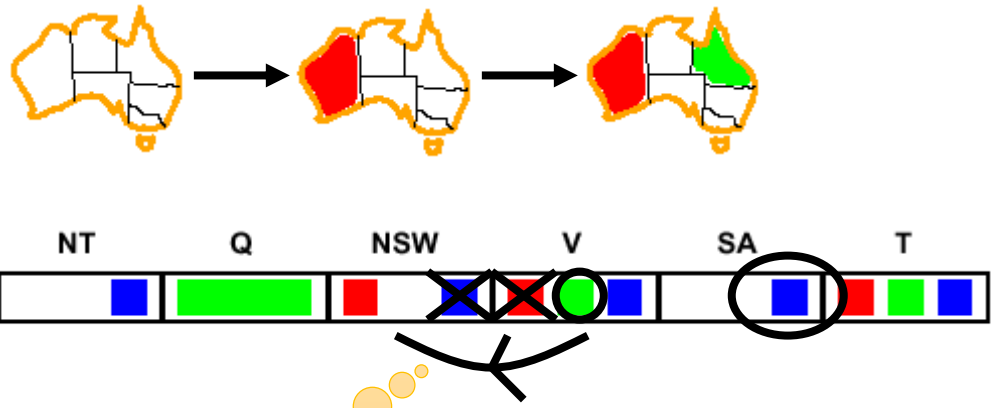
Caminho menos simples



Se NSW = azul então SA = { }

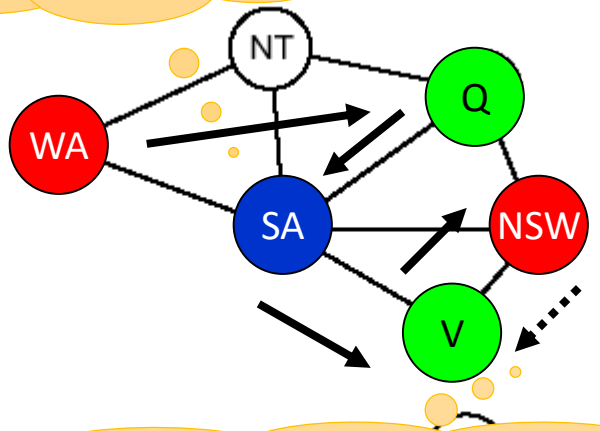
- Consistência de Arco:
  - A maneira mais simples de propagação torna cada arco consistente
  - X→Y é consistente sse para cada valor de x há algum valor permitido de y

É possível detectar falhas inevitáveis cedo?



Com a checagem da **vizinhança** de NSW,
o arco de V para NSW **tornou-se consistente!**

Se SA = *azul* então
NSW = {*vermelho*}



Se *vermelho* ∈ V={*vermelho*, *verde*, *azul*}
então V={*verde*, *azul*}

- Consistência de Arco:
  - A maneira mais simples de propagação torna cada arco consistente
  - X→Y é consistente sse para cada valor de x há algum valor permitido de y
  - Se *X* perde um valor, vizinhança de *X* precisa ser checada novamente.

Algoritmo AC-3: Consistência de Arcos

```
function AC-3( csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X_1, X_2, ..., 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) ← REMOVE-FIRST(queue)
    if RM-INCONSISTENT-VALUES(X_i, X_j) then
      for each X_k in NEIGHBORS[X_i] do
        add (X_k, X_i) to queue

---

function RM-INCONSISTENT-VALUES( X_i, X_j) returns true iff remove a value
  removed ← false
  for each x in DOMAIN[X_i] do
    if no value y in DOMAIN[X_j] allows (x,y) to satisfy constraint(X_i, X_j)
      then delete x from DOMAIN[X_i]; removed ← true
  return removed
```

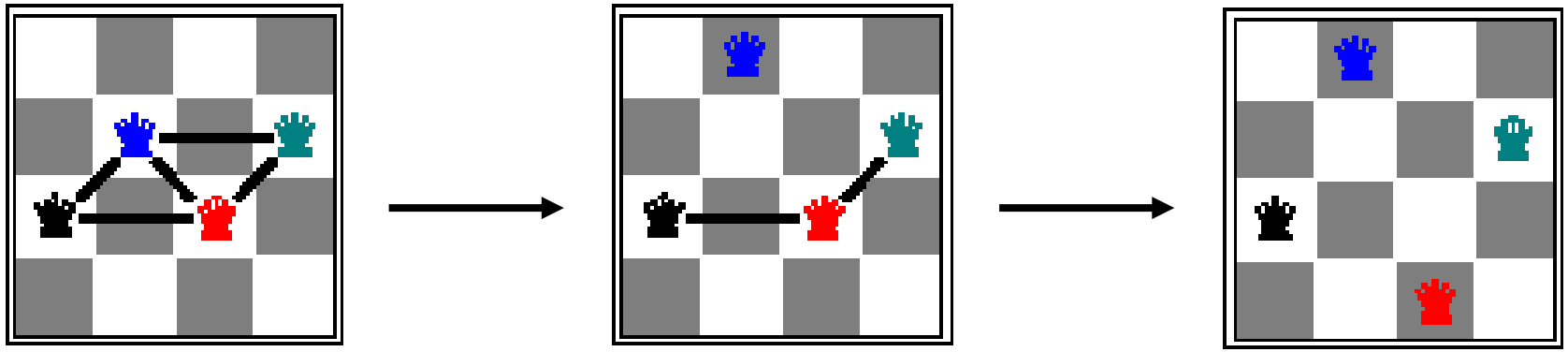
Complexidade de tempo: $O(n^2d^3)$

Busca Local para CSPs

Algoritmos Iterativos para CSPs

- Busca local funciona tipicamente com estados "completos", i.e., todas as variáveis atribuídas.
  - *Hill-climbing* e *Simulated Annealing*
- Adaptando para CSPs:
  - Permitir estados com restrições não satisfeitas
  - Operadores reatribuem valores
- Seleção de variáveis:
  - selecionar randomicamente qualquer variável em conflito
  - Heurística *min-conflicts* para seleção:
    - Escolha a variável que viole menos restrições
- Ex: *hill-climb* com h(n) = total de restrições violadas

Exemplo: Problema das 4-Rainhas



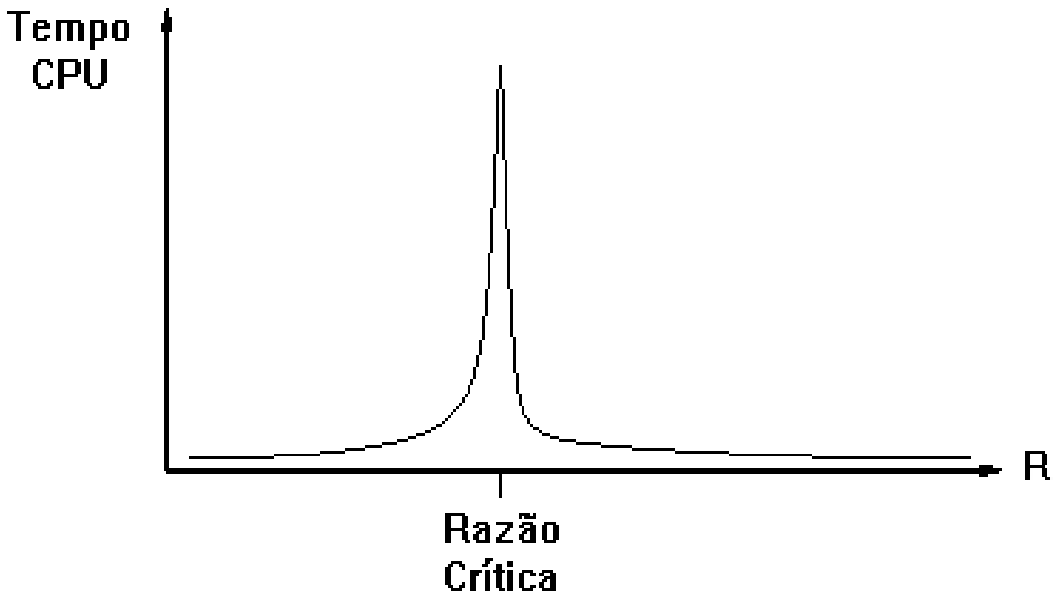
- **Estados:** 4 rainhas em 4 colunas (i.e., $4^4 = 256$ estados)
- **Ações:** mover rainha em uma coluna (ou seja, trocá-la de linha)
- **Teste de objetivo:** não há ataques
- **Avaliação:** h(n) = número de ataques

Desempenho do *min-conflicts*

- Estado Inicial Aleatório
  - Pode resolver o problema das *n-queens* em tempo quase sempre constante para um *n* arbitrário com alta probabilidade (ex: n = 10.000.000)

- Qualquer CSP gerado aleatoriamente
  - O mesmo aparenta ser verdade, exceto na faixa estreita de razão:

$$r = \frac{número\ de\ restrições}{número\ de\ variáveis}$$



Resumo

- CSPs são tipo especial de problema:
  - estados são definidos por valores de variáveis.
  - teste de objetivo é definido por restrições nos valores das variáveis.
- *Backtracking*: busca em profundidade com uma variável atribuída por vez.
- Verificação adiante previne atribuições que levam a falha futura
- Propagação de restrições ajuda a restringir valores e detectar inconsistências.
- Heurística de minimização de conflitos é bastante efetiva na prática.