

# **Busca com Satisfação de Restrições**

**Solange Rezende**

**LABIC-ICMC-USP**

# Até agora...

- Busca por um caminho: sequência de ações
  - Cada estado é considerado uma caixa preta;
  - Representação e rotinas específicas de problema;
  - O caminho até um objetivo é importante;
  - Caminhos diferentes podem possuir custos diferentes;
  - Heurísticas que auxiliam a busca são específicas para cada problema.
- Busca com restrição: atribuição de valores
  - O objetivo é importante, não o caminho;
  - Geralmente, todos os caminhos possuem a mesma profundidade;
  - Heurísticas de uso geral.

# Problemas de Satisfação de Restrições (PSR)

- O que é um PSR
  - *Conjunto finito de variáveis:*  $V_1, V_2, \dots, V_n$
  - *Domínio não vazio de valores possíveis* para cada variável  $D_{V_1}, D_{V_2}, \dots, D_{V_n}$
  - *Conjunto finito de restrições*  $C_1, C_2, \dots, C_m$
  - Cada *restrição*  $C_i$  limita os valores que uma variável pode assumir, por exemplo,  $V_1 \neq V_2$
- Um *estado* é uma *atribuição* de valores para algumas ou todas as variáveis.
- **Atribuição consistente:** atribuição que não viola nenhuma restrição.

# Problemas de Satisfação de Restrições

- Uma atribuição é *completa* quando toda variável possui um valor.
- Uma *solução* para um PSR é uma atribuição completa que satisfaz todas as restrições.
- Alguns PSRs requerem uma solução que maximiza uma *função objetivo*.
- Algumas Aplicações:
  - N-rainhas;
  - Coloração de mapas;
  - Criptografia.

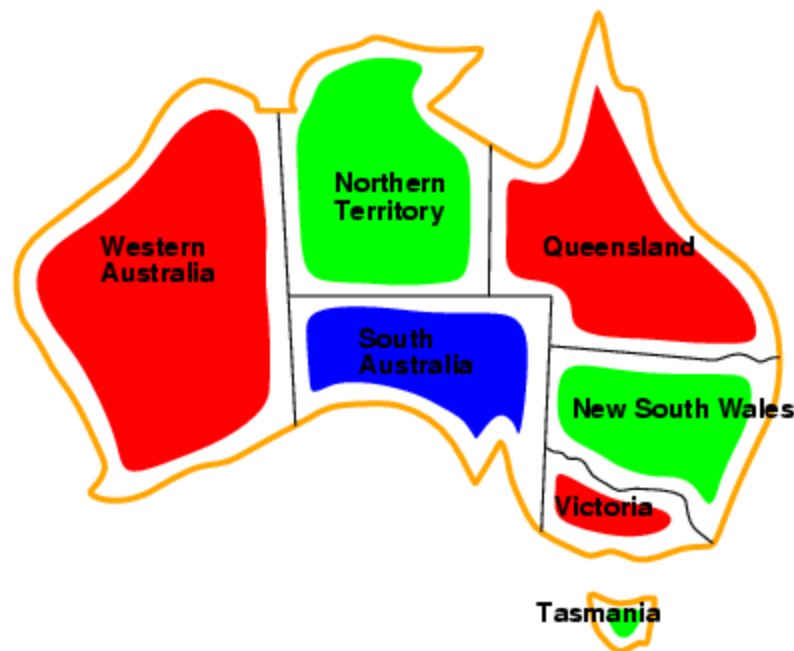
# Exemplo: Coloração de Mapas

Colorir **Mapa da Austrália** sem regiões vizinhas com a mesma cor



- **Variáveis:**  $WA, NT, Q, NSW, V, SA, T$
- **Domínio:**  $D_i = \{\text{vermelho}, \text{verde}, \text{azul}\}$
- **Restrições:** regiões adjacentes devem possuir cores diferentes por exemplo,  $WA \neq NT$ 
  - Portanto,  $(WA, NT)$  deve estar em  $\{(\text{vermelho}, \text{verde}), (\text{vermelho}, \text{azul}), (\text{verde}, \text{vermelho}), \dots\}$

# Exemplo: Coloração de Mapas

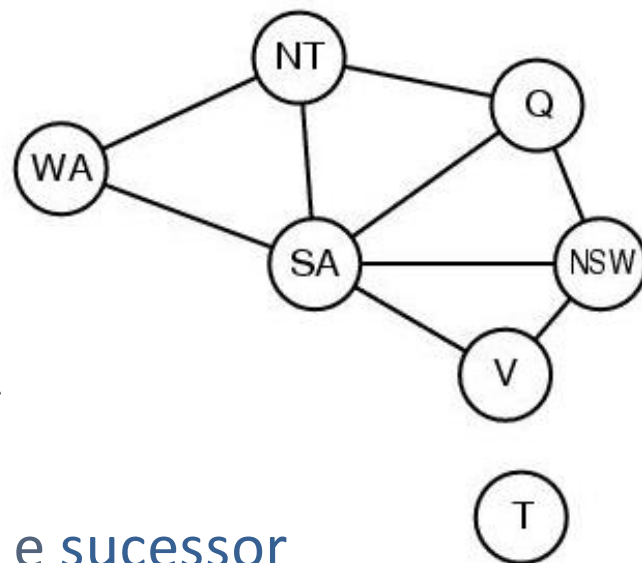


**Soluções** são atribuições **completas e consistentes**,

- Por exemplo, WA = vermelho, NT = verde, Q = vermelho, NSW = verde, V = vermelho, SA = azul, T = verde

# PSR como Grafos de Restrições

- *PSR binário*: cada restrição está relacionada com duas variáveis
- *Grafo de restrições*:
  - *Nós* são variáveis
  - *Arcos* são restrições
- Benefícios de utilizar um PSR
  - Padrão de representação
  - Funções genéricas de objetivo e sucessor
  - Heurísticas genéricas (sem necessidade de conhecimento profundo da aplicação).
- Grafo pode ser utilizado para simplificar a busca
  - Por exemplo, a Tasmânia é um subproblema independente



- **Variáveis discretas**

- **Domínios finitos:**

- $n$  variáveis, tamanho do domínio  $d \rightarrow O(d^n)$  atribuições completas
    - Ex de problemas: Coloração de mapas e 8-rainhas

- **Domínios infinitos:**

- inteiros, strings, etc.
    - Por exemplo, escalonamento de trabalho, variáveis são dia e início/fim de cada job
    - Necessidade de uma linguagem de restrições, por exemplo,  $IniciarAtiv_1 + 5 \leq IniciarAtiv_3$

- **Variáveis contínuas**

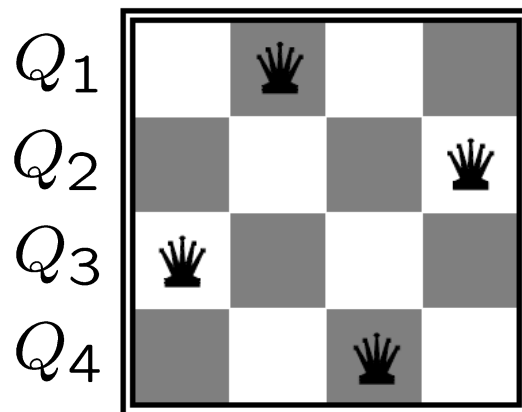
- Por exemplo, horários de início/fim para observações e manobra com o telescópio especial Hubble que exige sincronização muito precisa e devem obedecer restrições astronômicas, de precedência e energia
  - Restrições lineares são solucionáveis em tempo polinomial com programação inteira



# Variedade de Restrições

- **Unária** restrições que envolvem uma única variável,
  - Por exemplo.,  $SA \neq \text{verde}$
- **Binária** restrições que envolvem pares de variáveis,
  - Por exemplo,  $SA \neq WA$
- **Maior-ordem** restrições que envolvem 3 ou mais variáveis
  - Por exemplo, restrições do jogo sudoku
- **Preferência** (restrições leves)  
por exemplo *vermelho é melhor do que verde* pode ser representada por um custo diferente =>  
Problemas de **otimização de restrições**.

# Exemplo: N-Rainhas



- Variáveis:  $Q_k$
- Domínio:  $\{1, 2, 3, \dots, N\}$
- Restrições:
  - $\forall_{i,j}$  Não há ameaça  $(Q_i, Q_j)$

# Exemplo: Sudoku

	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		

	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

- **Variáveis:**  $A1, \dots, A9, B1, \dots, B9, \dots, I1, \dots, I9$
- **Domínio:**  $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- **Restrições:**  $Alldiff(A1, A2, A3, A4, A5, A6, A7, A8, A9)$

...

$Alldiff(A1, B1, C1, D1, E1, F1, G1, H1, I1)$

...

$Alldiff(A1, A2, A3, B1, B2, B3, C1, C2, C3)$

# Exemplo: Sudoku

	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		

- Domínio de E6 =  $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$   
 $- \{1, 2, 3, 5, 6, 7, 8, 9\}$   
 $= \{4\}$

# PSR como Problemas de Busca

- Um PSR pode ser expresso como um problema de busca
  - *Estado inicial*: uma atribuição vazia { }
  - *Operadores*: Atribuir um valor a uma variável livre dado que não existe conflito
  - *Objetivo final*: atribuição completa e consistente
  - *Custo caminho*: custo constante para cada passo
  - Uma solução é encontrada em profundidade  $n$ , para  $n$  variáveis

# Busca Backtracking

## (Busca com Retrocesso)

- Atribuição das variáveis é *comutativa*,
  - Por exemplo  $[WA = \text{vermelho depois } NT = \text{verde}]$  equivalente a  $[NT = \text{verde depois } WA = \text{vermelho}]$
- Somente é necessário considerar atribuições a uma variável a cada nó
  - Existem  $d^n$  nós-folha ( $n$  = num. variáveis e  $d$  = tam. domínio)
- Busca em profundidade para PSRs com atribuições de uma variável por vez e checagem de restrições é a busca *backtracking*;
- Pode solucionar  $n$ -rainhas com  $n \approx 25$ .

# Busca Backtracking

**função** BUSCA-COM-RETROCESSO(*psr*) **retorna** uma solução ou falha  
**retornar** RETROCESSO-RECURSIVO( $\{\}$ , *psr*)

**função** RETROCESSO-RECURSIVO(*atribuição*, *psr*) **retorna** uma solução ou falha  
**se** *atribuição* é completa **então** **retornar** *atribuição*

*var*  $\leftarrow$  SELECIONAR-VARIÁVEL-NAO-TRIBUÍDA(VARIÁVEIS[*psr*], *atribuição*,  
*psr*)

**para** cada *valor* em VALORES-DE-ORDEM-NO-DOMINIO(*var*, *atribuição*, *psr*)  
**faça**

**se** *valor* é consistente com *atribuição* de acordo com RESTRIÇÕES[*psr*]  
**então**

adicionar {*var* = *valor*} a *atribuição*

*resultado*  $\leftarrow$  RETROCESSO-RECURSIVO(*atribuição*, *psr*)

**se** *resultado*  $\neq$  falha **então** **retornar** *resultado*

remover {*var* = *valor*} de *atribuição*

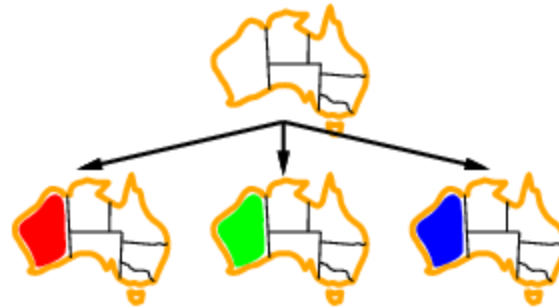
**retornar** falha

# Exemplo Backtracking

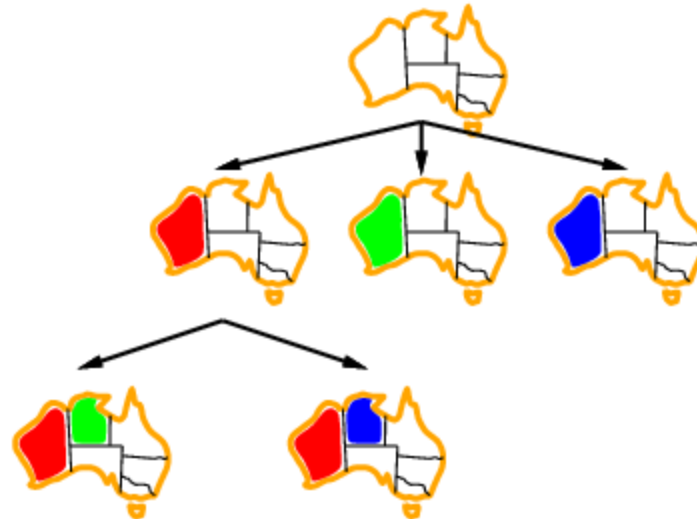


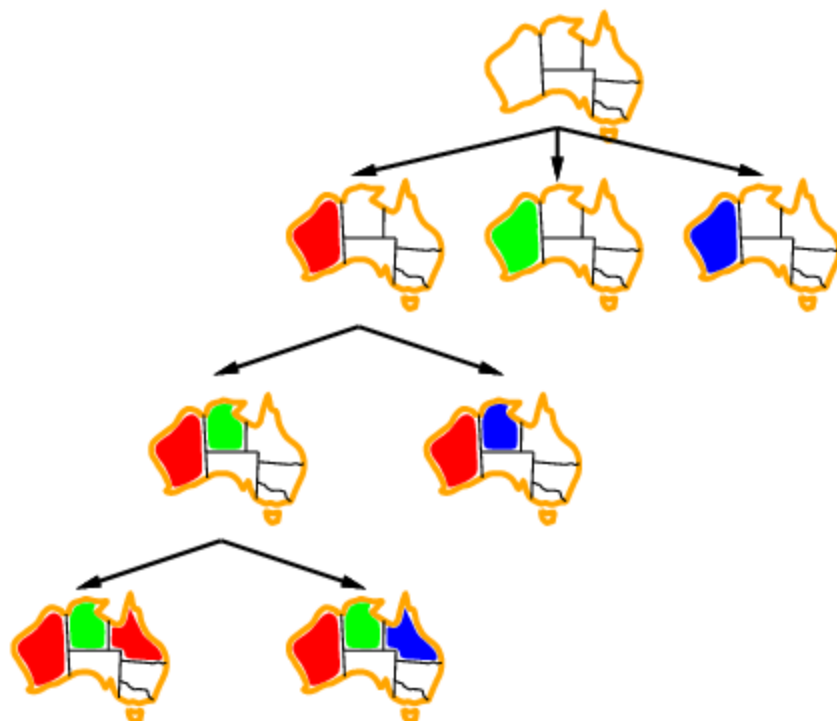


# Exemplo Busca Backtracking



# Exemplo Backtracking





# Melhorias ao Backtracking (retrocesso)

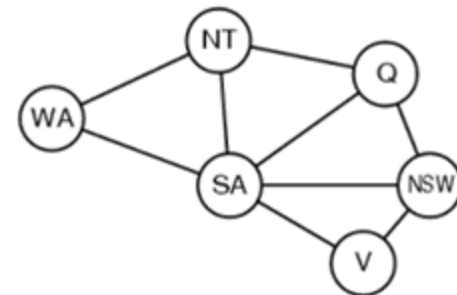
- Métodos de *propósito geral* podem fornecer grandes ganhos de desempenho detectando falhas inevitáveis mais cedo com modelos genéricos:
  - Qual variável deve ser a próxima a ser atribuída?
    - Heurística do maior grau
    - Heurística dos mínimos valores remanescentes
  - Em qual ordem os valores devem ser tentados?
    - Heurística do valor menos restritivo
  - Pode-se detectar falhas inevitáveis mais cedo?
    - Forward checking (verificação prévia)
    - Propagação de restrições (consistência de arcos)

## Qual variável deve ser a próxima a ser atribuída?

**função** BUSCA-COM-RETROCESSO(*psr*) **retorna** uma solução ou falha  
**retornar** RETROCESSO-RECURSIVO( $\{\}$ , *psr*)

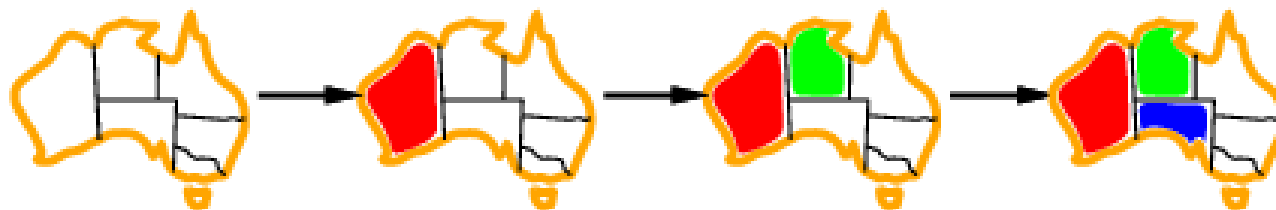
**função** RETROCESSO-RECURSIVO(*atribuição*, *psr*) **retorna** uma solução ou falha  
*se atribuição* é completa **então retornar** *atribuição*  
*var*  $\leftarrow$  SELECIONAR-VARIÁVEL-NAO-ATRIBUÍDA(VARIÁVEIS[*psr*], *atribuição*, *psr*)  
**para cada** *valor* em VALORES-DE-ORDEM-NO-DOMINIO(*var*, *atribuição*, *psr*)  
**faça**  
    *se valor* é consistente com *atribuição* de acordo com RESTRIÇÕES[*psr*]  
    **então**  
        adicionar {*var* = *valor*} a *atribuição*  
        *resultado*  $\leftarrow$  RETROCESSO-RECURSIVO(*atribuição*, *psr*)  
        *se resultado*  $\neq$  falha **então retornar** *resultado*  
        remover {*var* = *valor*} de *atribuição*  
**retornar** falha

# Variável mais Restrita



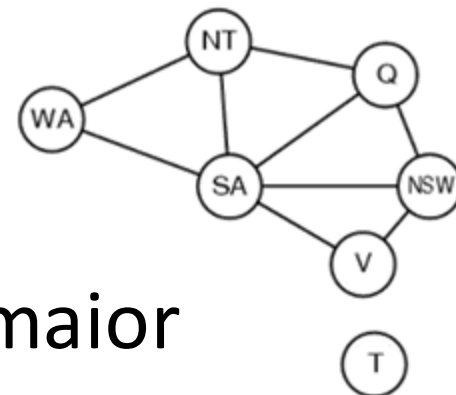
- Variável mais restrita:

Escolha a variável com menor número de valores “válidos”



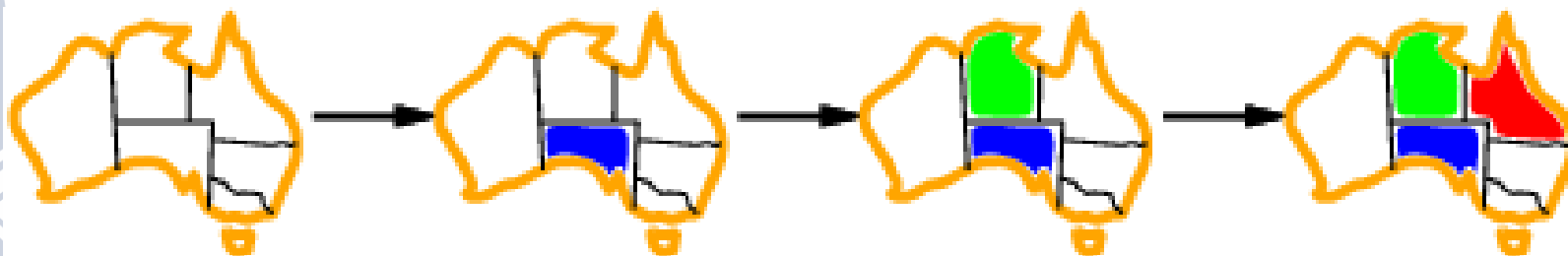
- Conhecida como heurística *mínimo valores remanescentes (MVR)*
  - Também conhecida como heurística de “variável mais restrita” ou de “primeira falha” – escolhe uma variável que tem maior probabilidade de provocar uma falha em breve

# Variável mais Restritiva



- Variável mais restritiva (heurística maior grau):

- Escolhe a variável envolvida no maior número de restrições sobre outras variáveis não-atribuídas
- No mapa da Australia SA é a variável com grau mais alto (5 - que está envolvida no maior número de restrições), T tem grau 0.



- No geral, MVR é melhor do que heurística grau, então utiliza-se grau como desempate para MVR. Bastante usada para desempatar as variáveis mais restritas

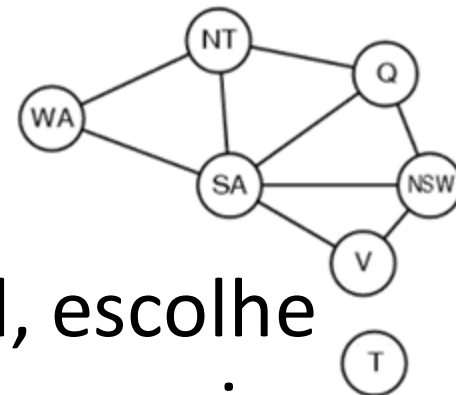
# Selecionada a variável - Em qual ordem os valores devem ser tentados?

**função** BUSCA-COM-RETROCESSO(*psr*) **retorna** uma solução ou falha  
**retornar** RETROCESSO-RECURSIVO({}, *psr*)

**função** RETROCESSO-RECURSIVO(*atribuição*, *psr*) **retorna** uma solução ou falha  
*se atribuição* é completa **então retornar** *atribuição*  
 $var \leftarrow \text{SELECIONAR-VARIÁVEL-NAO-TRIBUÍDA}(\text{VARIÁVEIS}[psr], \text{atribuição}, psr)$   
**para cada** *valor* em VALORES-DE-ORDEM-NO-DOMINIO(*var*, *atribuição*, *psr*)  
**faça**  
     *se valor* é consistente com *atribuição* de acordo com RESTRIÇÕES[*psr*]  
     **então**  
         adicionar {*var* = *valor*} a *atribuição*  
         *resultado*  $\leftarrow$  RETROCESSO-RECURSIVO(*atribuição*, *psr*)  
         *se resultado*  $\neq$  falha **então retornar** *resultado*  
         remover {*var* = *valor*} de *atribuição*  
**retornar** falha



# Valor menos restritivo



- Esta heurística, dada uma variável, escolhe o valor que tem menos restrições, ou seja, escolhe o valor que elimina menos valores no domínio das outras variáveis

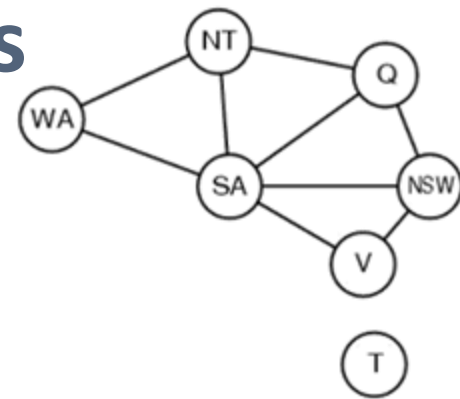


Permite um valor para SA

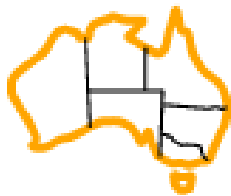
Permite nenhum valor para SA

- Combinar essas heurísticas faz com que o problema das 1000-rainhas seja possível.

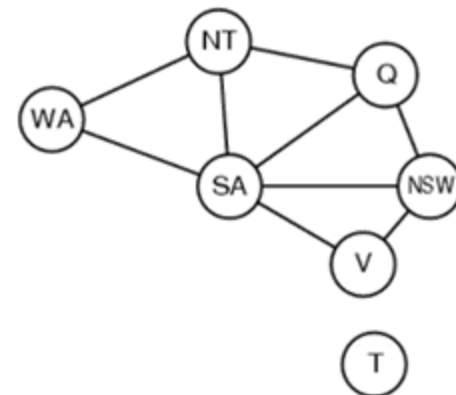
# Propagando informações por meio de restrições - Verificação Prévia



- **Ideia:**
  - Manter os valores legais remanescentes para variáveis não atribuídas
  - Retroceder a busca quando uma variável não possuir valores legais



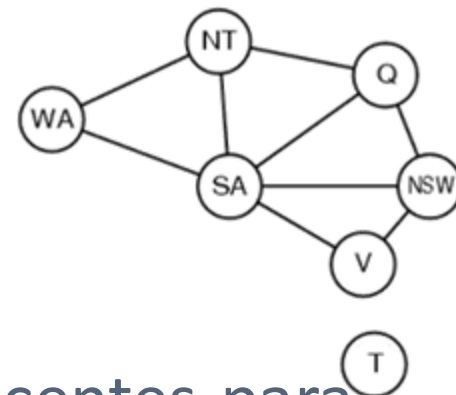
# Verificação Prévia



- **Ideia:**
  - Manter os valores legais remanescentes para variáveis não atribuídas
  - Retroceder a busca quando uma variável não possuir valores legais



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>



- **Ideia:**
  - Manter os valores legais remanescentes para variáveis não atribuídas
  - Retroceder a busca quando uma variável não possuir valores legais



WA	NT	Q	NSW	V	SA	T
						
						
						

# Verificação Prévia



- **Ideia:**
  - Manter os valores legais remanescentes para variáveis não atribuídas
  - Retroceder a busca quando uma variável não possuir valores legais

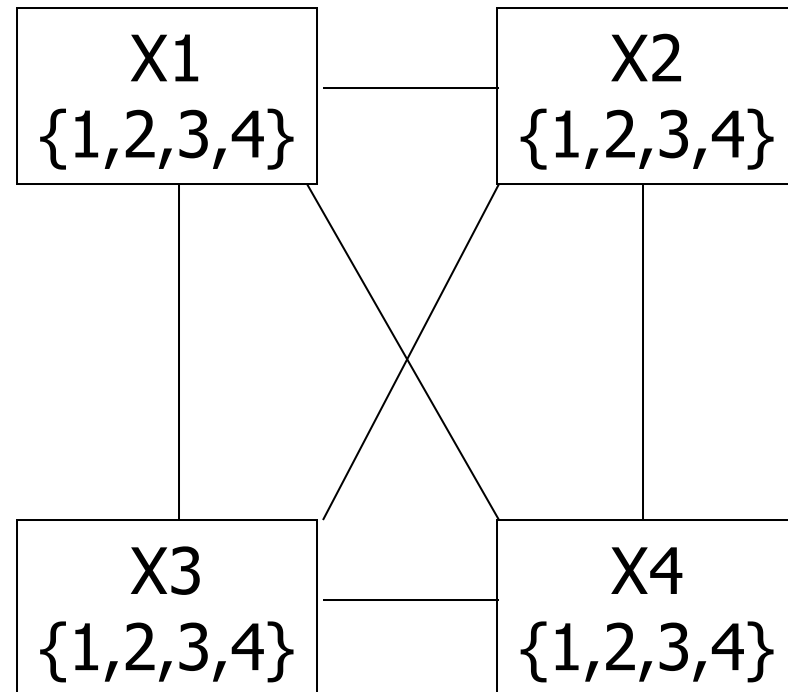


WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div>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








- Atribuição prévia detectou que  $\{WA=\text{vermelho}; Q=\text{verde}, V=\text{azul}\}$  é inconsistente com as restrições do problema (Nenhum valor possível para SA): backtracking

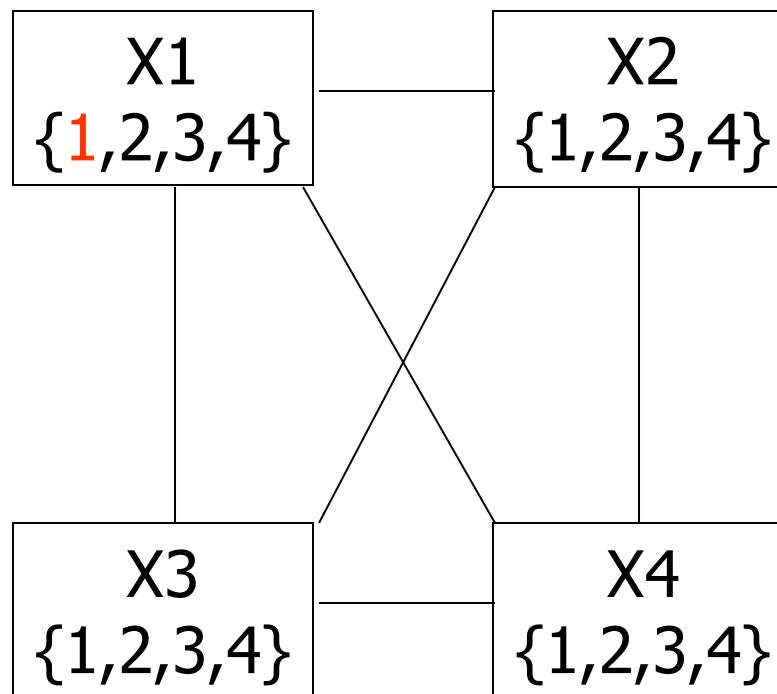
# Exemplo: Problema das 4-Rainhas

	1	2	3	4
1				
2				
3				
4				










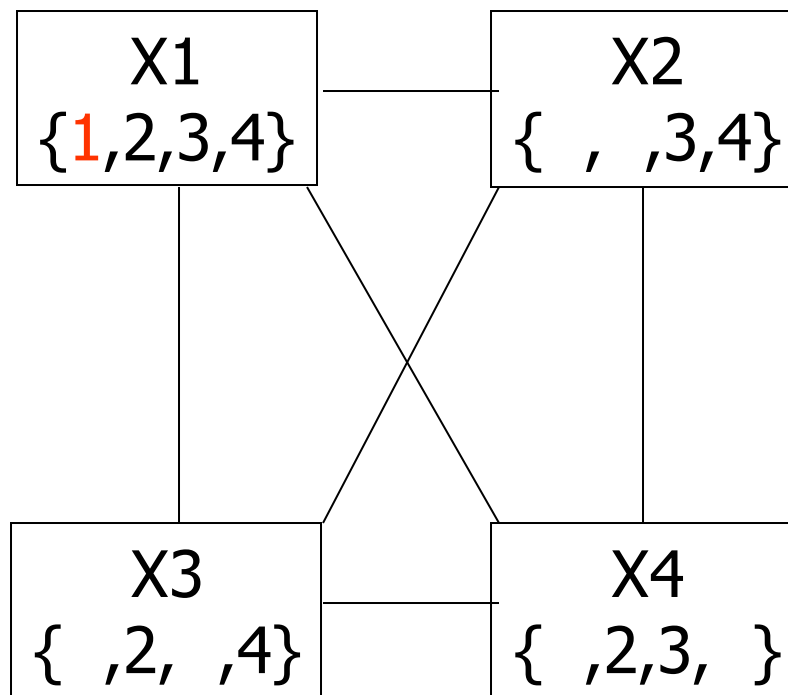
# Exemplo: Problema das 4-Rainhas

	1	2	3	4
1				
2				
3				
4				



# Exemplo: Problema das 4-Rainhas

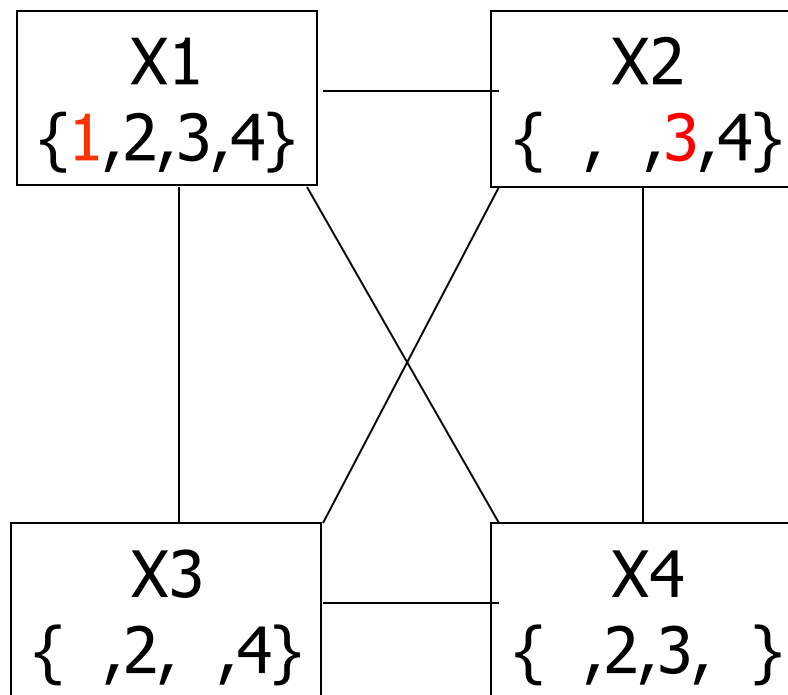
	1	2	3	4
1				
2				
3				
4				





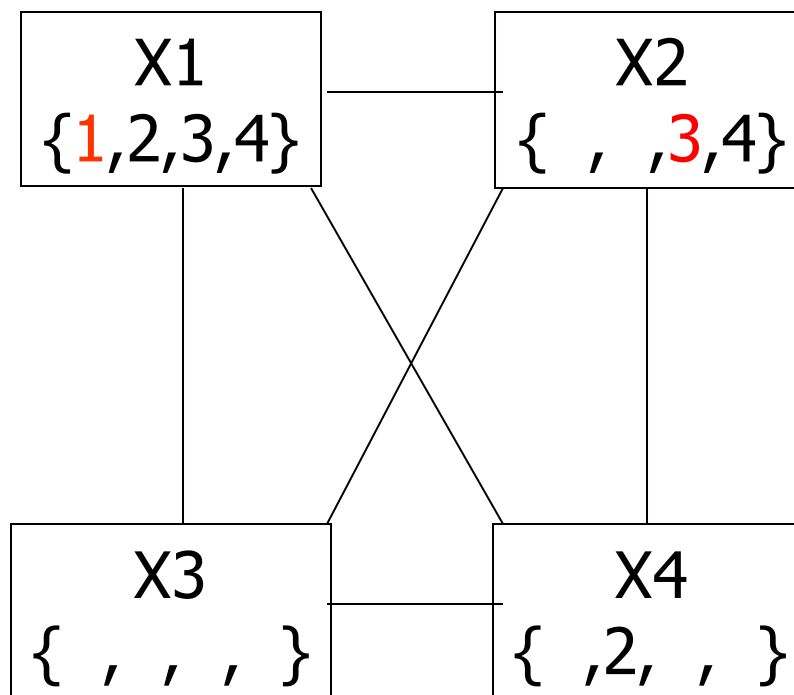
# Exemplo: Problema das 4-Rainhas

	1	2	3	4
1	★	●	●	●
2		●	●	
3		★	●	●
4			●	●



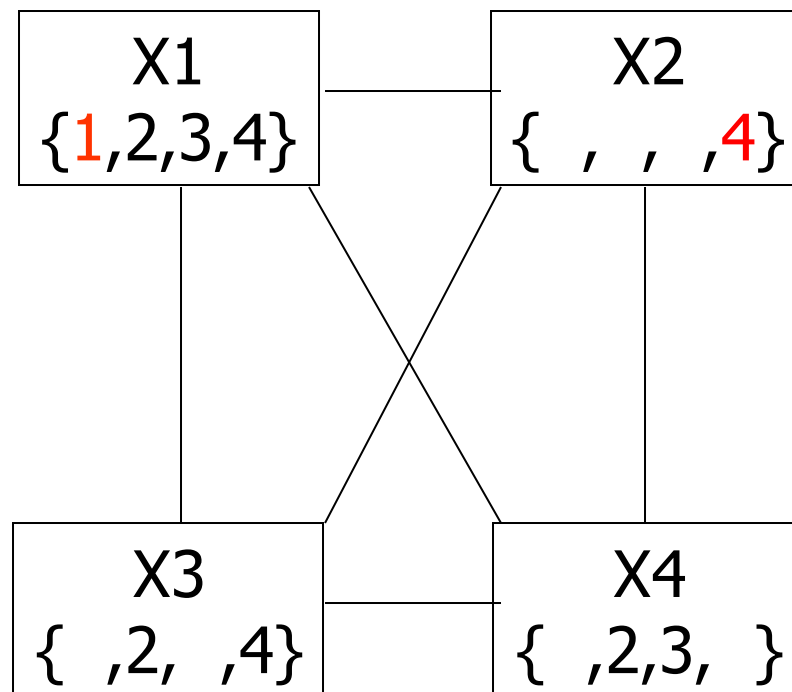
# Exemplo: Problema das 4-Rainhas

	1	2	3	4
1	★	●	●	●
2		●	●	
3		★	●	●
4			●	●














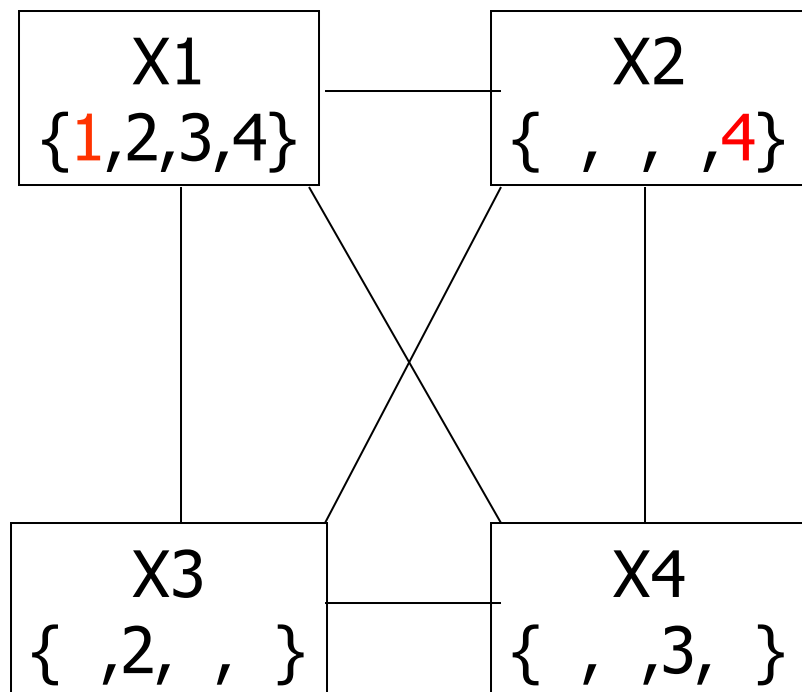
# Exemplo: Problema das 4-Rainhas

	1	2	3	4
1	★	●	●	●
2		●		●
3		●	●	
4		★	●	●



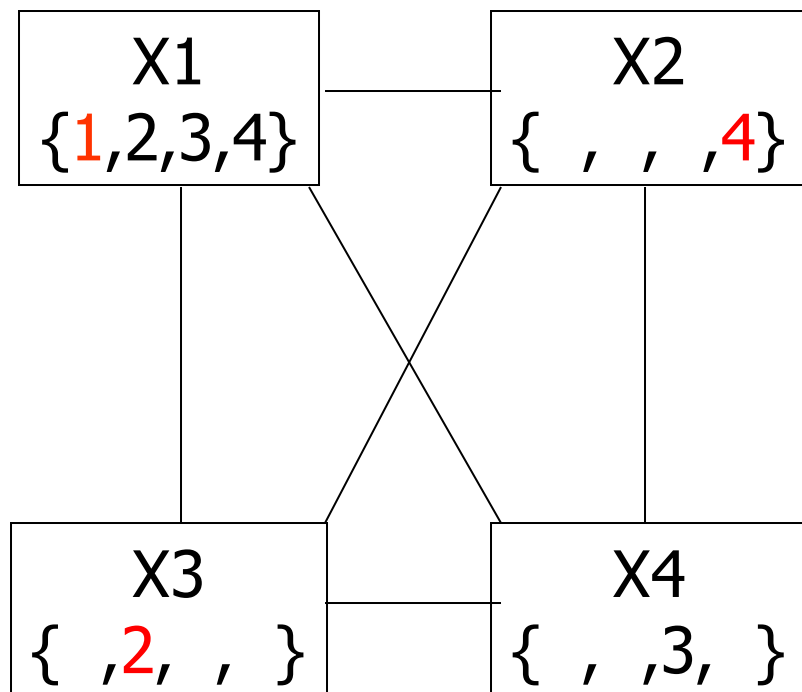
# Exemplo: Problema das 4-Rainhas

	1	2	3	4
1				
2				
3				
4				



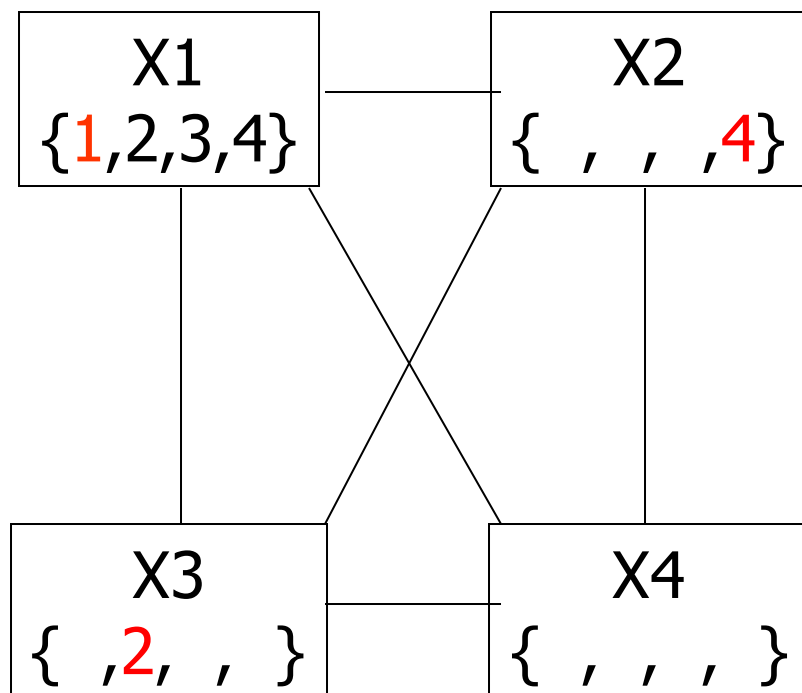
# Exemplo: Problema das 4-Rainhas

	1	2	3	4
1	★	●	●	●
2	■	●	★	●
3	■	●	●	●
4	■	★	●	●



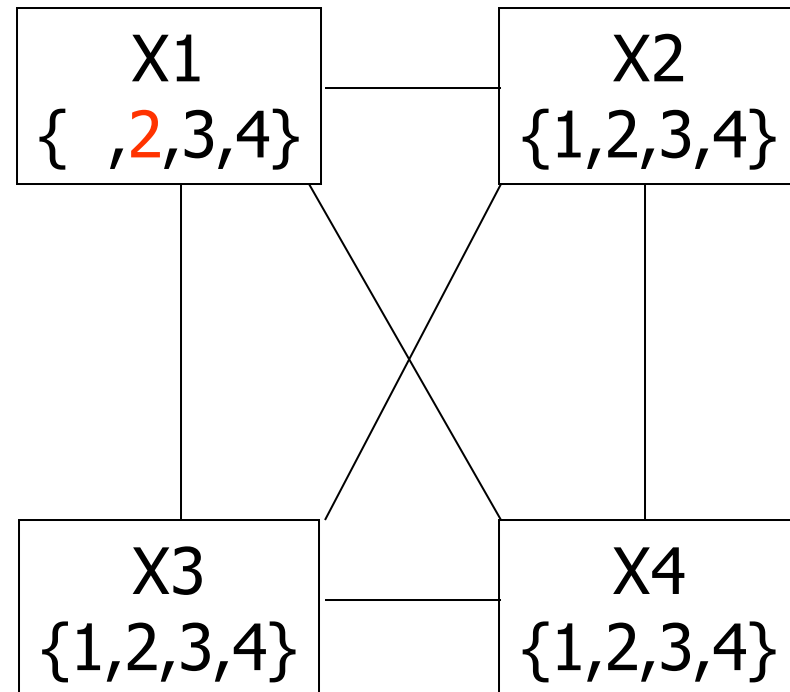
# Exemplo: Problema das 4-Rainhas

	1	2	3	4
1	★	●	●	●
2		●	★	●
3		●	●	●
4		★	●	●



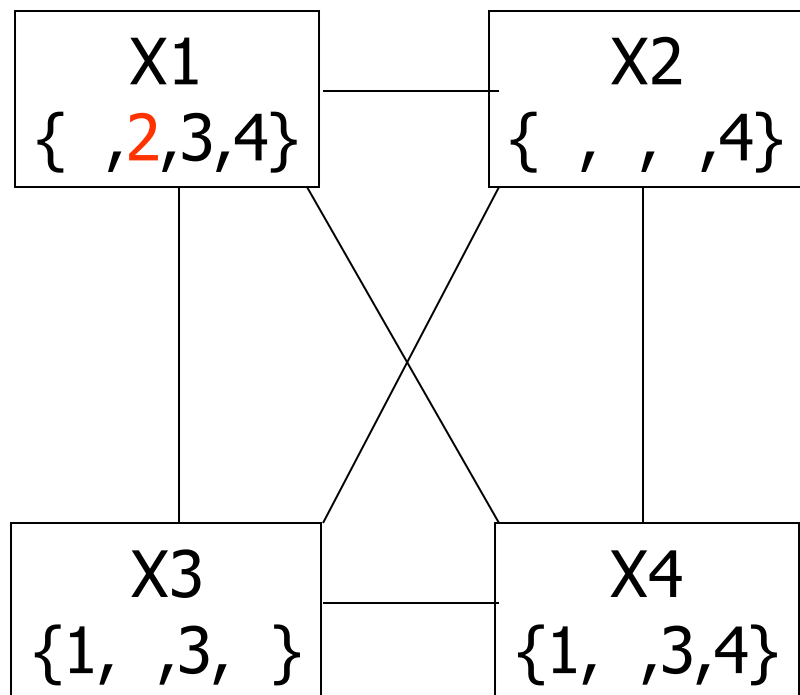
# Exemplo: Problema das 4-Rainhas

	1	2	3	4
1		●		
2	★	●	●	●
3		●		
4			●	



# Exemplo: Problema das 4-Rainhas

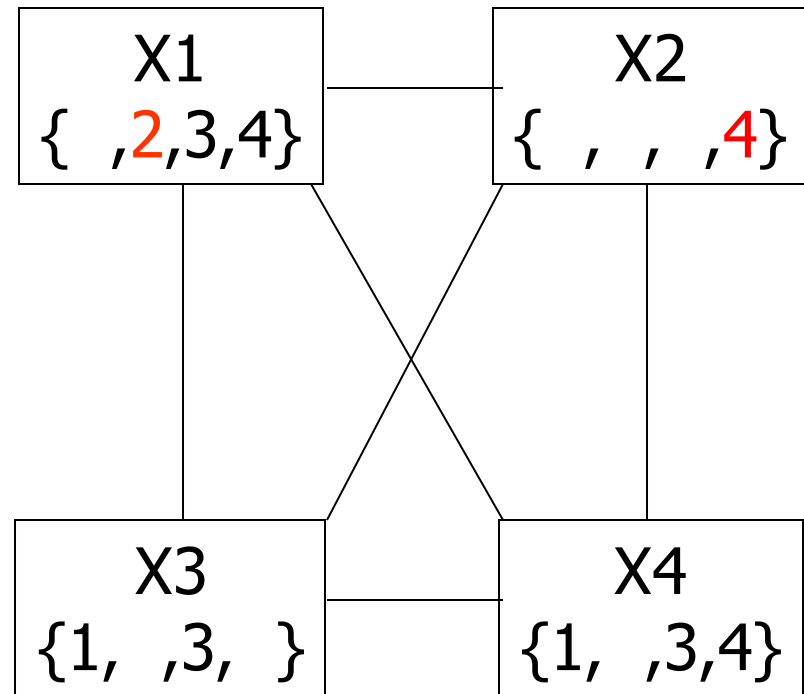
	1	2	3	4
1		●		
2	★	●	●	●
3		●		
4			●	





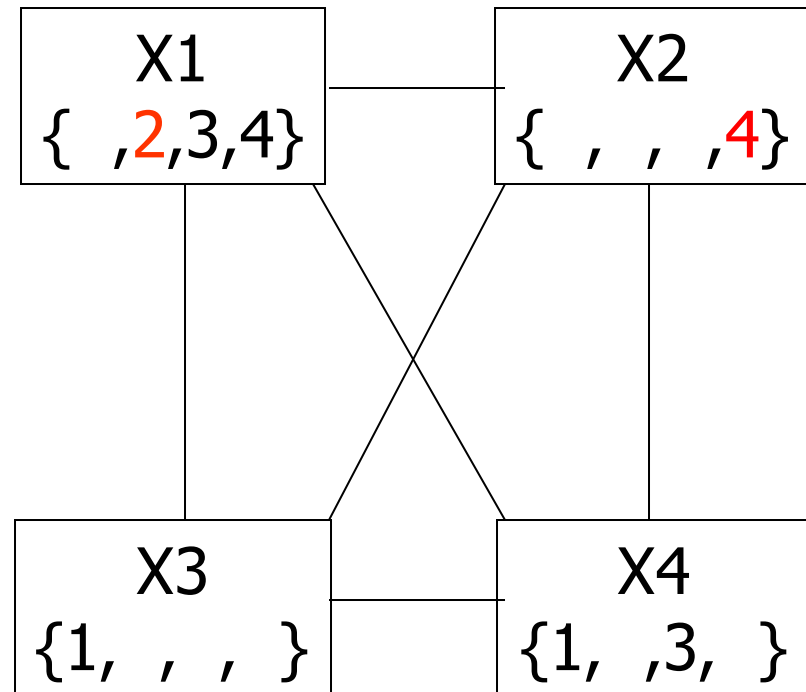
# Exemplo: Problema das 4-Rainhas

	1	2	3	4
1		●		
2	★	●	●	●
3		●	●	
4		★	●	●



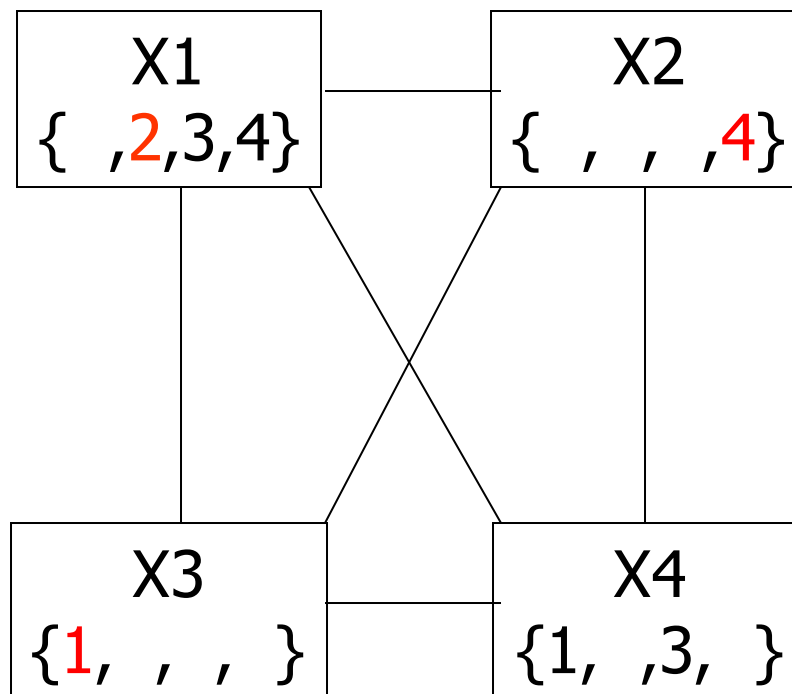
# Exemplo: Problema das 4-Rainhas

	1	2	3	4
1		●		
2	★	●	●	●
3		●	●	
4		★	●	●



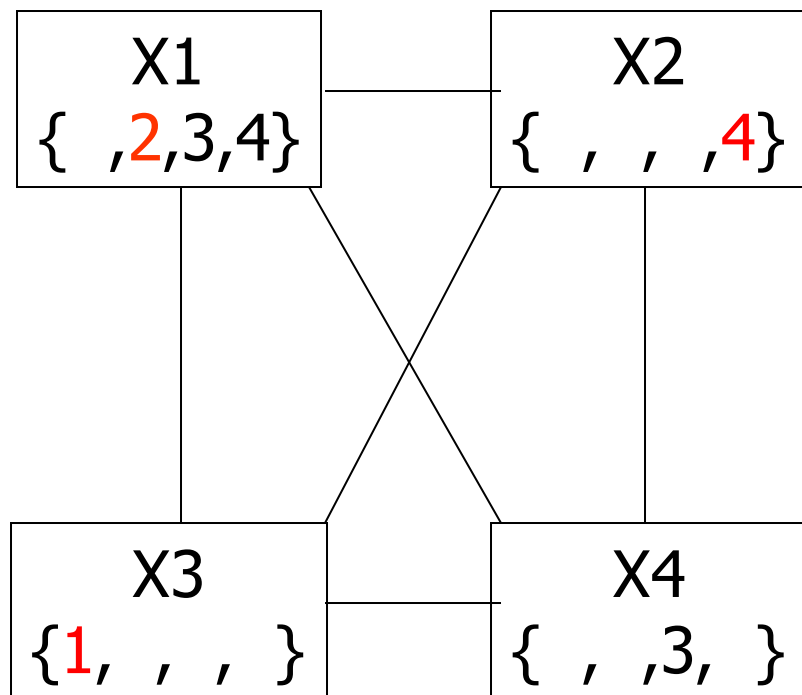
# Exemplo: Problema das 4-Rainhas

	1	2	3	4
1		●	★	●
2	★	●	●	●
3		●	●	
4		★	●	●



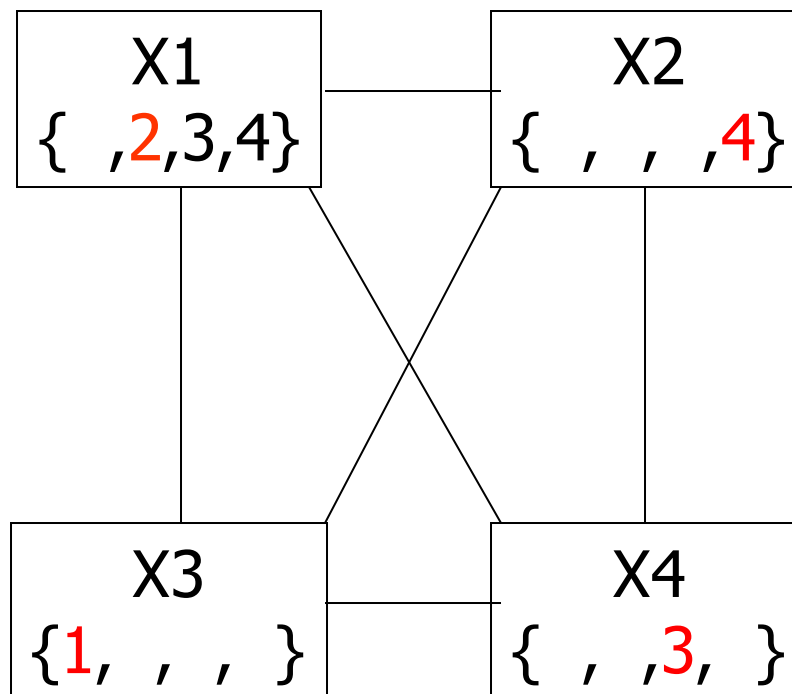
# Exemplo: Problema das 4-Rainhas

	1	2	3	4
1		●	★	●
2	★	●	●	●
3		●	●	
4		★	●	●



# Exemplo: Problema das 4-Rainhas

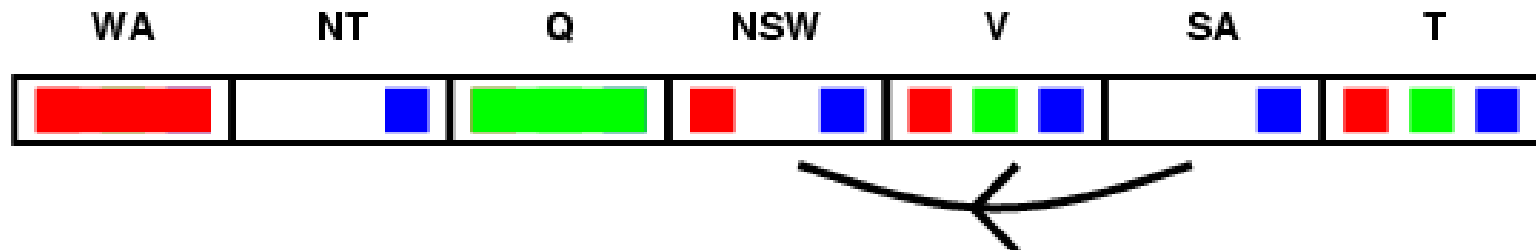
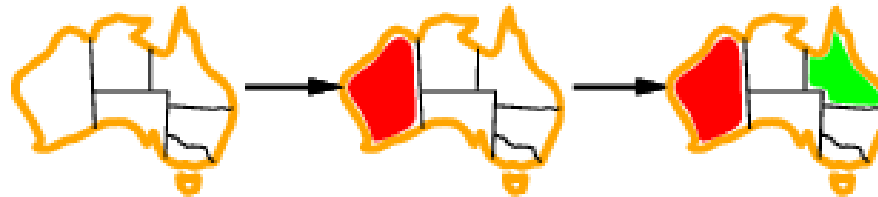
	1	2	3	4
1		●	★	●
2	★	●	●	●
3		●	●	★
4	●	★	●	●



# Propagação de restrições

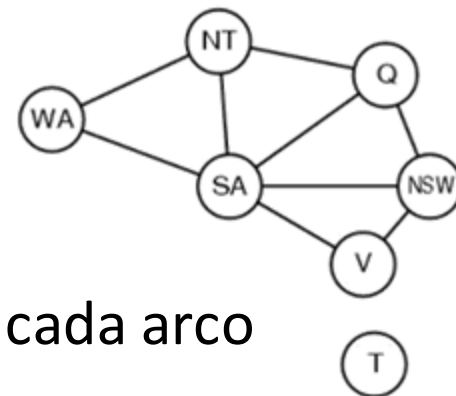


- A verificação prévia não detecta todas inconsistências
- A forma mais simples de propagação torna cada arco **consistente**
- Arco  $X \rightarrow Y$  (ligação em um grafo de restrições) é consistente **see** Para **cada** valor  $x_i$  de  $X$  existe **algum** valor permitido  $y_j$  em  $Y$

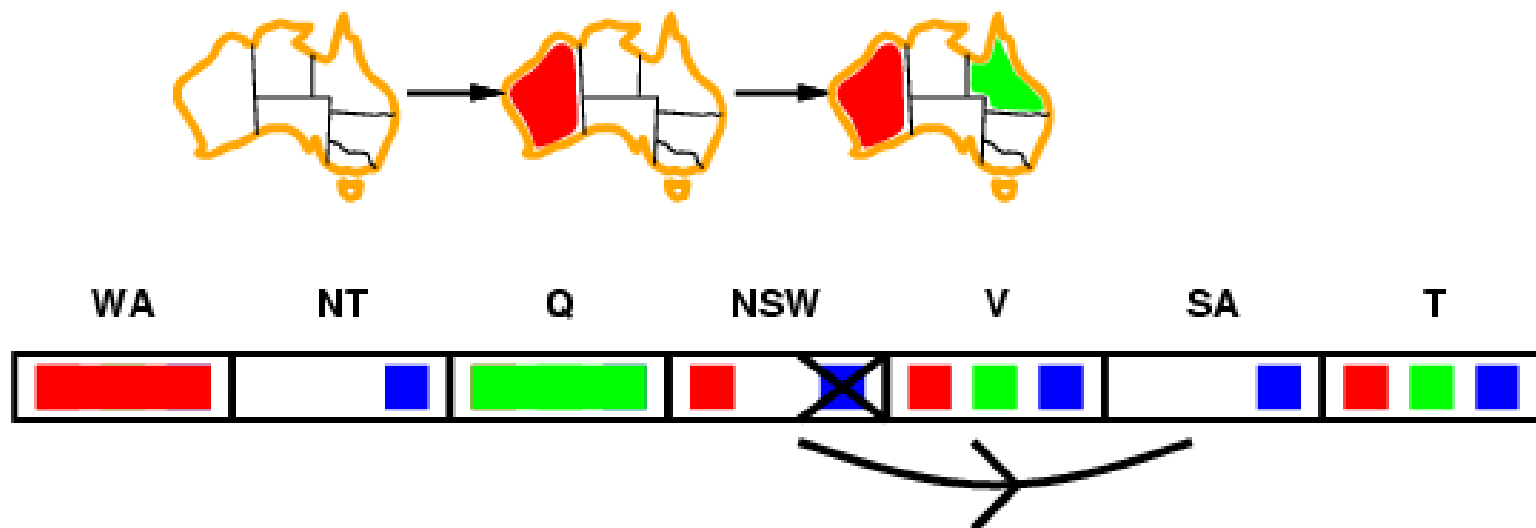


- No caso de SA= azul, existe uma atribuição consistente para NSW=vermelho. Nesse caso o arco de SA até NSW é consistente

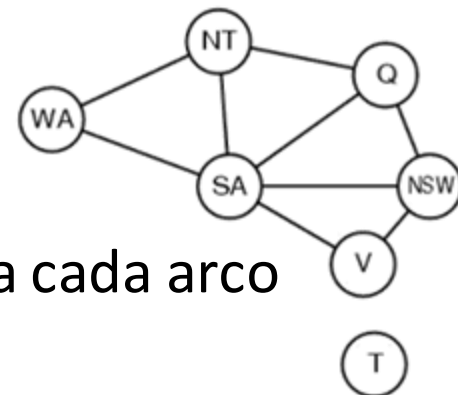
# Consistência de arco



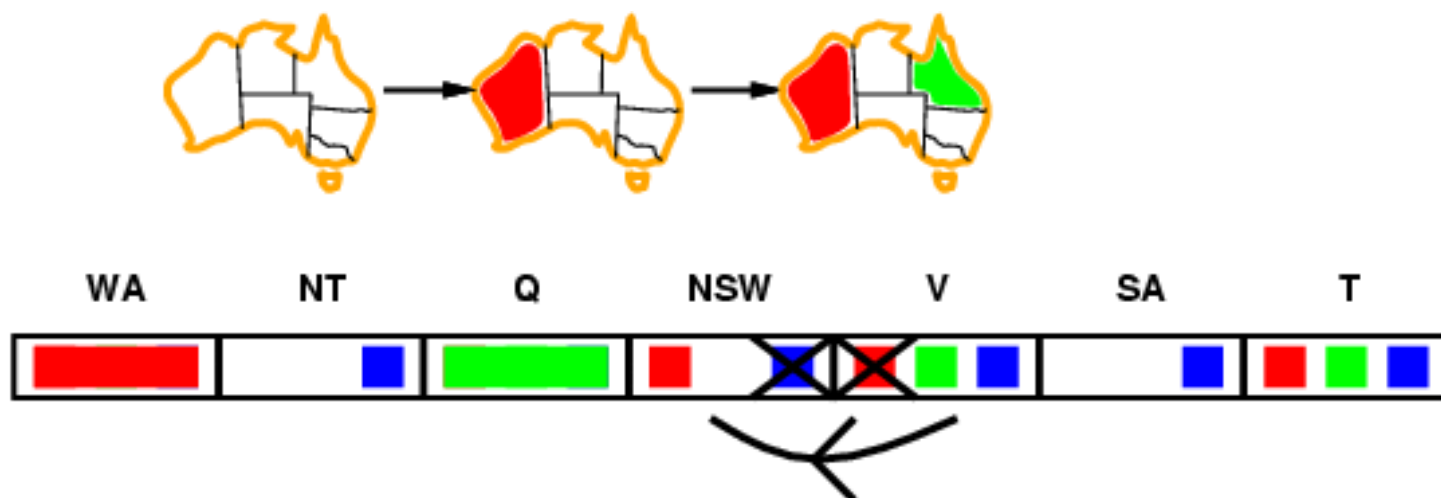
- A forma mais simples de propagação torna cada arco **consistente**
- Arco  $X \rightarrow Y$  (ligação em um grafo de restrições) é consistente **see** para **cada** valor  $x$  de  $X$  existe **algum** valor permitido  $y$  em  $Y$



# Consistência de arco

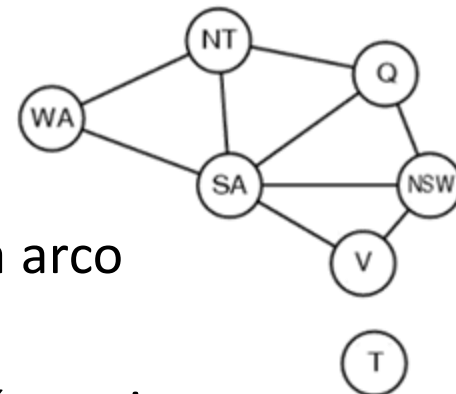


- A forma mais simples de propagação torna cada arco **consistente**
- Arco  $X \rightarrow Y$  (ligação em um grafo de restrições) é consistente **see** para **cada** valor  $x$  de  $X$  existe **algum** valor permitido  $y$  em  $Y$
- Se  $X$  perde um valor para remover uma inconsistência de arco, pode surgir uma nova inconsistência - então os vizinhos de  $X$  devem ser re-verificados

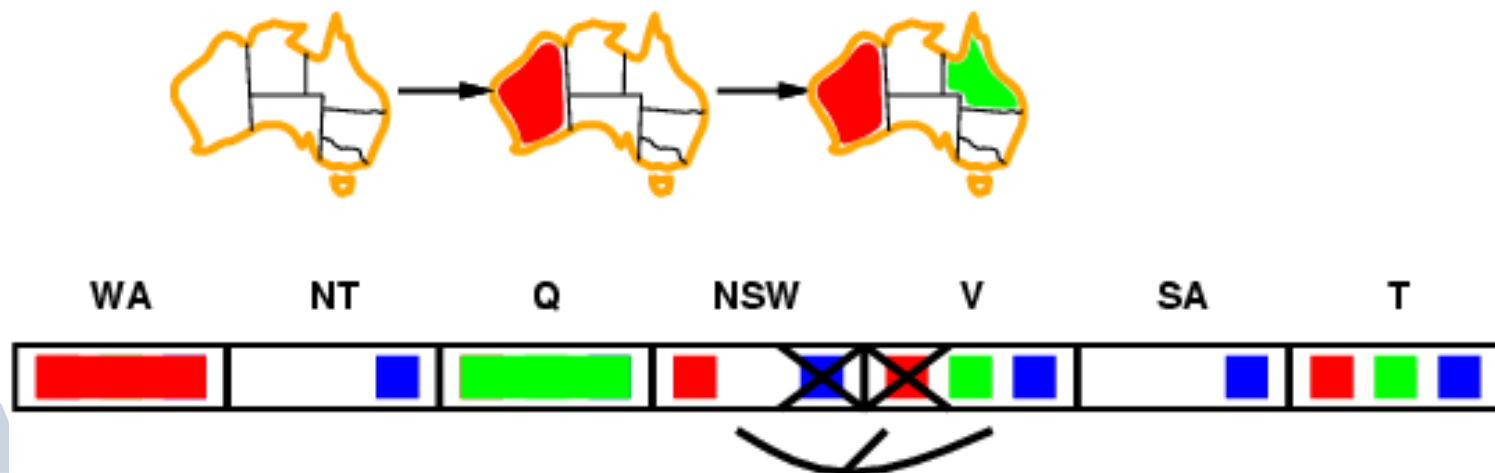




# Consistência de arco



- A forma mais simples de propagação torna cada arco **consistente**;
- Arco  $X \rightarrow Y$  (ligação em um grafo de restrições) é consistente **See** para **cada** valor  $x$  de  $X$  existe **algum** valor permitido  $y$  em  $Y$
- Se  $X$  perde um valor, então os vizinhos de  $X$  devem ser re-verificados;
- **Consistência de arco detecta falhas antes da verificação prévia;**
- **Pode ser executado como uma etapa de pré-processamento.**



# Algoritmo de Consistência de Arco CA-3

**função** CA-3(psr) **retorna** o PSR, possivelmente com domínios reduzidos

**entradas:** psr, um PSR binário com variáveis  $\{X_1, X_2, \dots, X_n\}$

**variáveis locais:** fila, uma fila de arcos, inicialmente todos os arcos no psr

**enquanto** fila é não-vazia **faça**

$(X_i, X_j) \leftarrow \text{REMOVE-PRIMEIRO}(\text{fila})$

**se** REMOVER-VALORES-INCONSISTENTES( $X_i, X_j$ ) **então**

**para cada**  $X_k$  em VIZINHOS[ $X_i$ ] **faça**

adicionar  $(X_k, X_i)$  a fila

**função** REMOVER-VALORES-INCONSISTENTES( $X_i, X_j$ ) **retorna** verdadeiro se

removemos um valor

removido  $\leftarrow$  falso

**para cada**  $x$  em DOMINIO[ $X_i$ ] **faça**

**se** nenhum valor  $y$  em DOMINIO[ $X_j$ ] permitir que  $(x, y)$  satisfaça à restrição entre  $X_i$  e  $X_j$

**então** eliminar  $x$  de DOMINIO[ $X_i$ ]; removido  $\leftarrow$  verdadeiro

**retornar** removido

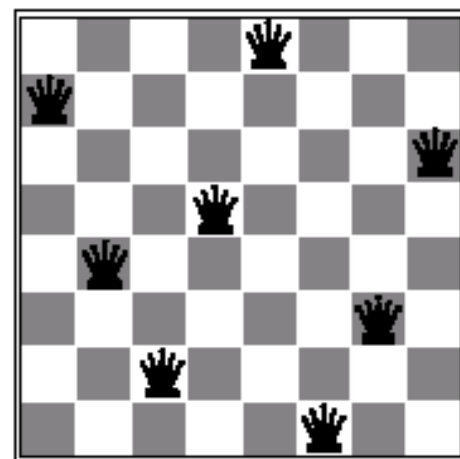
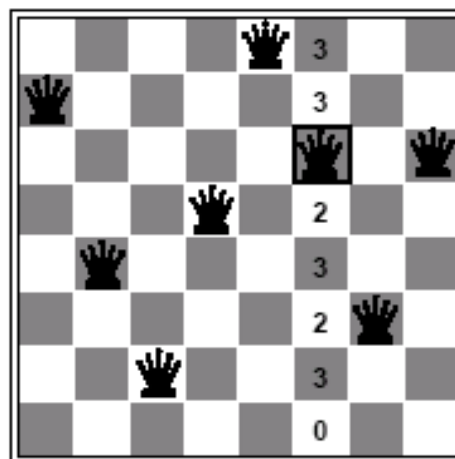
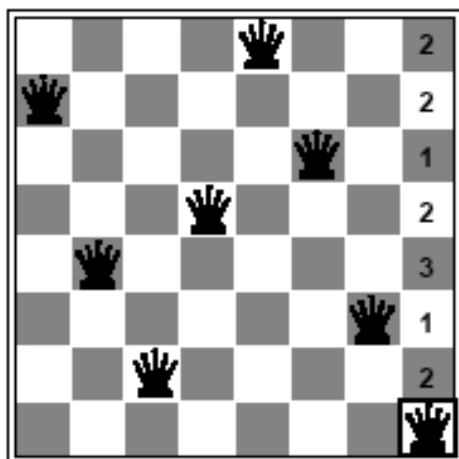
- CA-3 utiliza uma fila para controlar os arcos cuja consistência precisa ser verificada e complexidade de tempo:  $O(n^2 d^3)$

# Busca Local para PSRs

- Métodos de busca local tipicamente trabalham com estados completos, i.e., com todas as variáveis atribuídas
- Para aplicar PSRs:
  - Permite estados com restrições não satisfeitas
  - Operadores podem **reatribuir** valores para variáveis
- Seleção da variável: aleatória para qualquer variável com conflitos

# Exemplo: Heurística para N-rainhas

- Seleção de valor pela heurística **conflitos mínimos**:
  - Escolha um valor que viola o menor número de restrições



- Heurística bastante insensível ao tamanho  $N$  no problema mais geral das  $N$ -rainhas:
  - Resolve para  $n = 1$  milhão em média em 50 passos !!!

# Algoritmo de Conflitos Mínimos

**função** CONFLITOS-MINIMOS(*psr*, *max\_etapas*) **retorna** uma solução ou falha

**entradas:** *psr*, um PSR

*max\_etapas*, o número de etapas permitidas

*corrente* <- uma atribuição inicial completa para *psr*

**para** *i* = 1 até *max\_etapas* **faça**

**se** *corrente* é uma solução para *psr* então **retornar** *corrente*

*var* <- uma variável em conflito escolhida ao acaso a partir de VARIAVEIS[*psr*]

*valor* <- o valor *v* para *var* que minimiza CONFLITOS(*var*, *v*, *corrente*, *psr*)

definir *var* = *valor* em *corrente*

**retornar** *falha*

# Resumo

- PSRs são um tipo especial de problema:
  - Estados são definidos por valores de um conjunto fixo de variáveis
  - Teste objetivo definido pelas restrições nos valores das variáveis
- Backtracking = DFS + Atribuição de variáveis + Verificação de falha
- Ordem variáveis e as heurísticas de seleção de valores ajudam significativamente
- Verificação adiante previne atribuições que garantem uma falha posterior
- Propagação de restrições (p.e. consistência de arco) detecta inconsistência por meio de inferências
- Conflitos mínimos é geralmente efetivo na prática

- Melhorias ao Retrocesso com Métodos de *propósito geral* detectando falhas mais cedo :
  - Para escolha da qual variável deve ser a próxima:
    - Heurística do maior grau
    - Heurística dos mínimos valores remanescentes
  - Ordem dos valores:
    - Heurística do valor menos restritivo
  - Detectando falhas inevitáveis mais cedo:
    - Forward checking (verificação prévia)
    - Propagação de restrições (consistência de arcos)

**Slides baseados em:**

**RUSSEL, S.; NORVIK, P. Artificial  
Intelligence:  
A Modern Approach. Prentice Hall, 1995.**

**Material elaborado por  
Gustavo Batista e Solange Rezende**