PROBLEMAS DE SATISFAÇÃO DE RESTRIÇÕES CAP 6

Parcialmente adaptado de
http://aima.eecs.berkeley.edu

Resumo

- Exemplos de CSP (Constraint Satisfaction Problems)
- Procura com retrocesso (backtracking) para CSPs
- Estrutura e decomposição de problemas
- Procura local para CSPs

Problemas de Satisfação de Restrições (CSPs)

- Problema usual de procura:
  - o estado é uma “caixa negra” – estrutura de dados arbitrária suportando métodos para teste objectivo, funções de avaliação e sucessor
- CSP:
  - estado é definido com variáveis X_i que tomam valores num domínio D_i
  - teste objectivo é um conjunto de restrições especficando as combinações permitidas para subconjuntos de variáveis
- Exemplo simples de uma linguagem de representação formal
- Permite recorrer a algoritmos genéricos melhores do que os algoritmos de procura estudados anteriormente

Problemas de Satisfação de Restrições (CSPs)

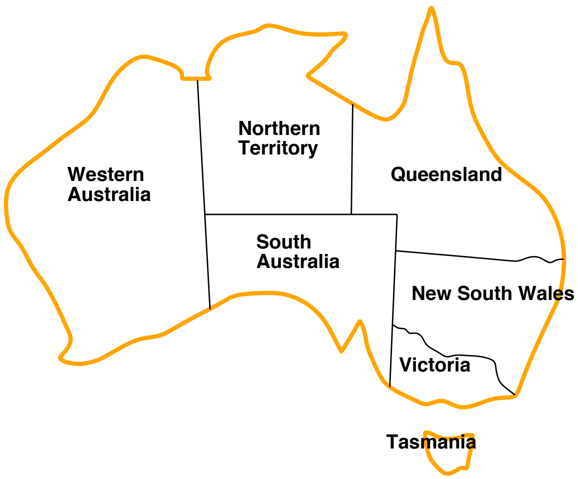
Formulação de um CSP:

- Conjunto finito de variáveis $V_1, \ldots, V_n$
- Domínios não vazios dos valores possíveis de cada variável $D_1,\ldots,D_n$
- Conjunto finito de restrições $C_1, \ldots, C_m$
- Cada restrição C_i limita os valores que cada variável pode tomar:
  - Consiste num par <*escopo,rel*>, onde
    - *escopo* é um tuplo de variáveis
    - *rel* uma relação que define os valores que essas variáveis podem tomar.
  - A relação rel pode ser representada por uma lista explícita de todos os tuplos possíveis ou através de uma definição implícita.
  - Por exemplo, se V_1 e V_2 tomam valores no domínio {A,B}, uma restrição indicando que as variáveis têm que ter valores diferentes pode ser representada como
    - $<(V_1,V_2), [(A,B),(B,A)]>$ ou
    - $<(V_1,V_2), V_1 \neq V_2>$.
    - Por vezes, deixa-se o escopo implicito e indica-se apenas $V_1 \neq V_2$

Problemas de Satisfação de Restrições (CSPs)

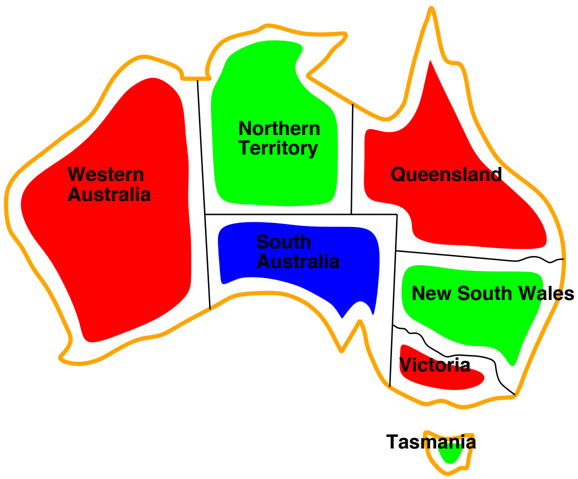
- Um estado é definido como uma atribuição de valores a uma ou mais variáveis
- Atribuição consistente: uma atribuíção que não viola as restrições
- Atribuição completa: uma atribuíção que contempla todas as variáveis
- Solução: uma atribuição completa e consistente
- Atribuição parcial: uma atribuíção que contempla apenas uma parte das variáveis

Exemplo de CSP: Coloração de Mapas



- Variáveis: WA, NT, Q, NSW, V , SA, T
- Domínios: D_i = {red; green; blue}
- Restrições: regiões adjacentes devem ter cores diferentes
  - e.g., WA ≠ NT (se a linguagem o permitir), ou
  - (WA,NT) ∈ {(red, green),(red, blue),(green, red),(green, blue),…}

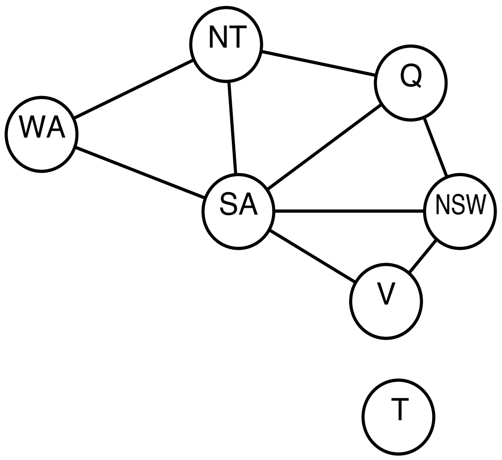
Exemplo de CSP: Coloração de Mapas



- Soluções são atribuições de valores a variáveis que satisfazem todas as restrições, e.g.,
- {WA=red, NT=green, Q=red, NSW=green, V =red, SA=blue, T =green}

Grafo de Restrições

- CSP binário: cada restrição relaciona no máximo duas variáveis
- Grafo de restrições: nós são variáveis e arcos correspondem a restrições



- Algoritmos genéricos para CSPs fazem uso da estrutura do grafo para tornar a procura mais efciente.
  - E.g., Tasmânia é um subproblema independente!
  - E.g., se SA=azul, podemos reduzir o domínio dos seus 5 vizinhos – que não poderão ter a cor azul – reduzindo o número de possíveis atribuição de 3^5=243 para 2^5=32.

Tipos de CSPs

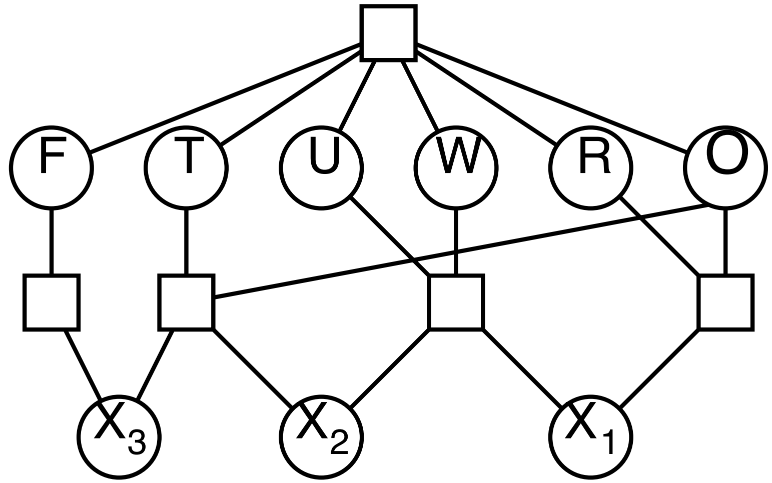
- Variáveis discretas
  - Domínios finitos; cardinalidade d $\Rightarrow$ $O(d^n)$ atribuições completas
    - e.g., CSPs Booleanos, incl. satisfatibilidade Booleana (NP-completo)
  - Domínios infinitos (inteiros, cadeias de caracteres, etc.)
    - escalonamento: variáveis representam início/fim das tarefas
    - utilizam linguagem de restrições: $StartJob_1 + 5 \leq StartJob_3$
    - restrições lineares são solúveis, não-lineares são indecidíveis
- Variáveis contínuas
  - e.g., tempos de início/fim das observações do Telescópio Hubble
  - restrições lineares resolúveis em tempo polinomial por métodos de programação linear

Tipos de Restrições

- Unárias: restrições envolvendo apenas uma variável,
  - e.g., SA ≠ green
- Binárias: restrições envolvendo pares de variáveis,
  - e.g., SA ≠ WA
- Ordem superior: restrições envolvendo 3 ou mais variáveis,
  - e.g., restrições das colunas em problemas cripto-aritméticos
  - E.g., alldiff
- Preferências (restrições suaves),
  - e.g., red é melhor do que green
  - habitualmente representado atribuindo um custo a cada atribuição de variáveis
  - problemas de optimização com restrições

Exemplo de CSP: Criptoaritmética

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



- Variáveis: F T U W R O X_1 X_2 X_3
- Domínios: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- Restrições:
  - alldiff(F, T, U, W, R, O)
  - $O + O = R + 10 \cdot X_1$, etc.

CSPs reais

- Problemas de alocação
  - e.g., quem é o professor de determinada disciplina?
- Problemas de horários
  - e.g., quando é que uma disciplina é oferecida e onde?
- Configuração de Hardware
- Folhas de cálculo
- Escalonamento de Transportes
- Planeamento/Escalonamento de Produção
- Planeamento Espacial
- Notar que muitos problemas reais requerem variáveis contínuas

Formulação como Problema de Procura

- Os estados são definidos pelos valores atribuídos até ao momento às variáveis
  - Estado inicial: a atribuição vazia, { }
  - Função sucessor: atribuir um valor a uma variável livre que não entre em conflito com a atribuição actual.
    - falha se não existirem atribuições possíveis (irreparável!)
  - Teste objectivo: a atribuição não tem variáveis livres

1. É o mesmo para todos os CSPs
2. Toda a solução ocorre à profundidade n com n variáveis
   - utilização de procura em profundidade primeiro
3. Caminho é irrelevante, pode-se usar formulação de estado completo
4. $b = (n - l)d$ à profundidade l, logo $n!d^n$ folhas (apesar de haver apenas d^n atribuições completas)

Procura com retrocesso (backtrack)

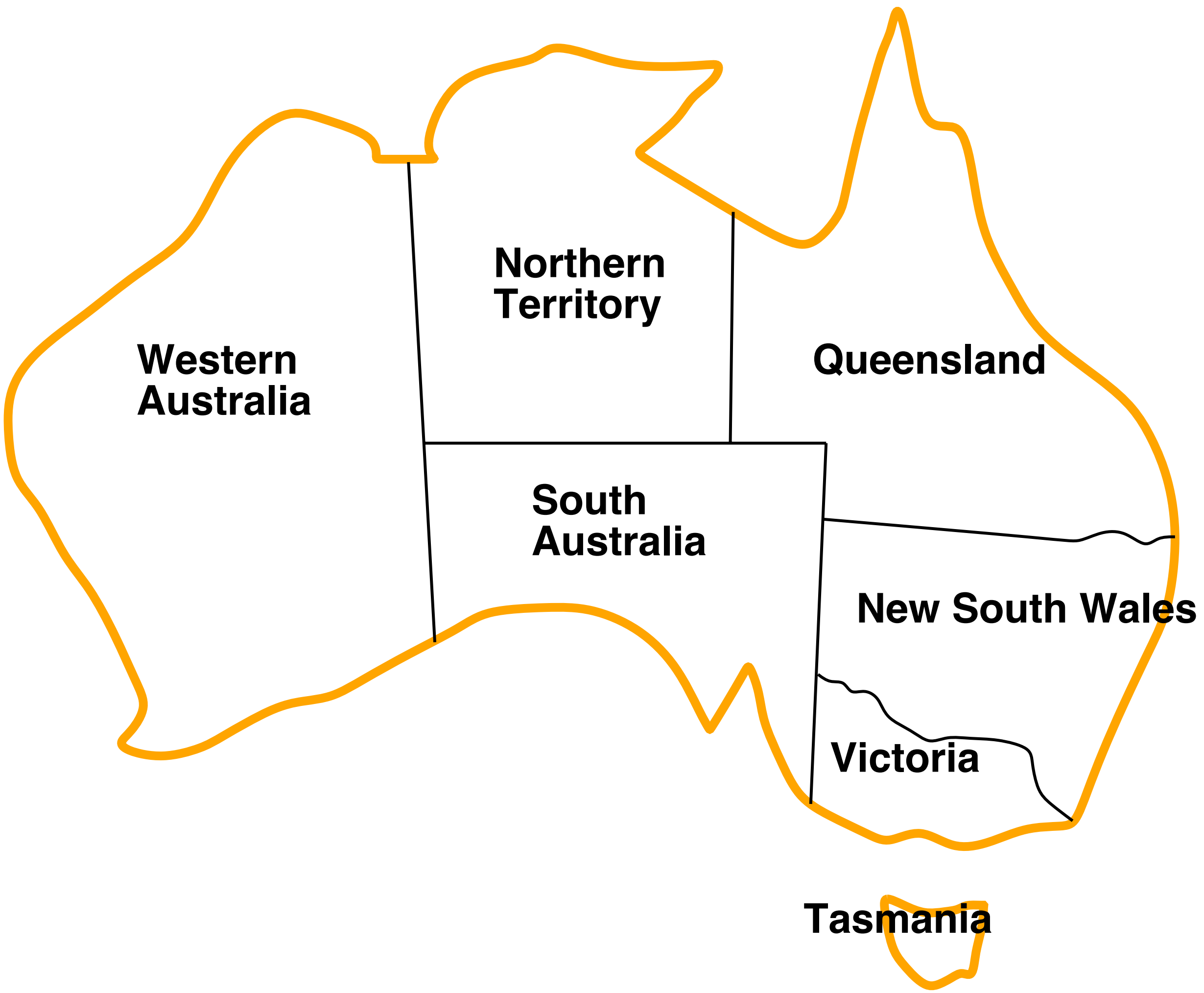
- As atribuições de variáveis são comutativas, i.e.,
  - [WA=red depois NT=blue] mesmo que [NT=blue depois WA=red]
- Só é preciso considerar atribuições a uma única variável em cada nó
  - b = d e existem d^n folhas
- Procura em profundidade primeiro para CSPs com atribuições a uma única variável é designada por procura com retrocesso
- Procura com retrocesso é o algoritmo básico cego para CSPs
  - Consegue resolver n-rainhas para n≈25

Procura com retrocesso

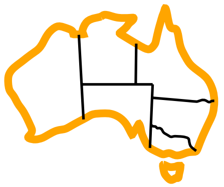
function BACKTRACKING-SEARCH(*csp*) **return** a solution or failure
  return RECURSIVE-BACKTRACKING([] *, csp*)

function RECURSIVE-BACKTRACKING(*assigned, csp*) **return** a solution or failure
  if *assigned* is complete **then return** *assigned*
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*], *assigned*, *csp*)
  for each *value* **in** ORDER-DOMAIN-VALUES(*var, assigned, csp*) **do**
    if *value* is consistent with *assigned* according to CONSTRAINTS[*csp*] **then**
      result ← RECURSIVE-BACTRACKING([*var=value* | *assigned*]*, csp*)
      if *result* ≠ *failure* **then return** *result*
  end
  return *failure*

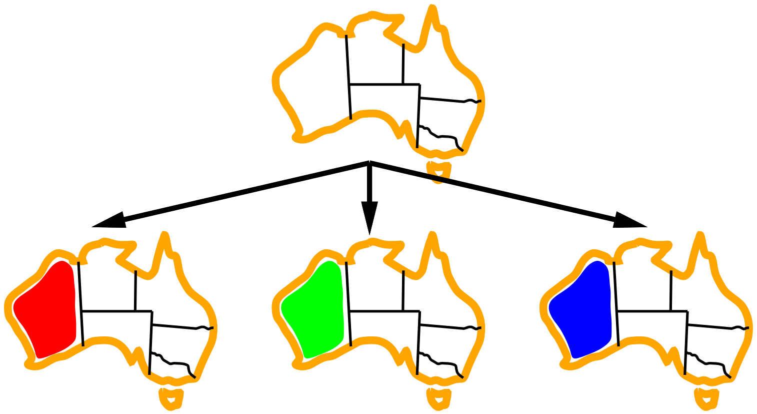
Exemplo de Procura com Retrocesso



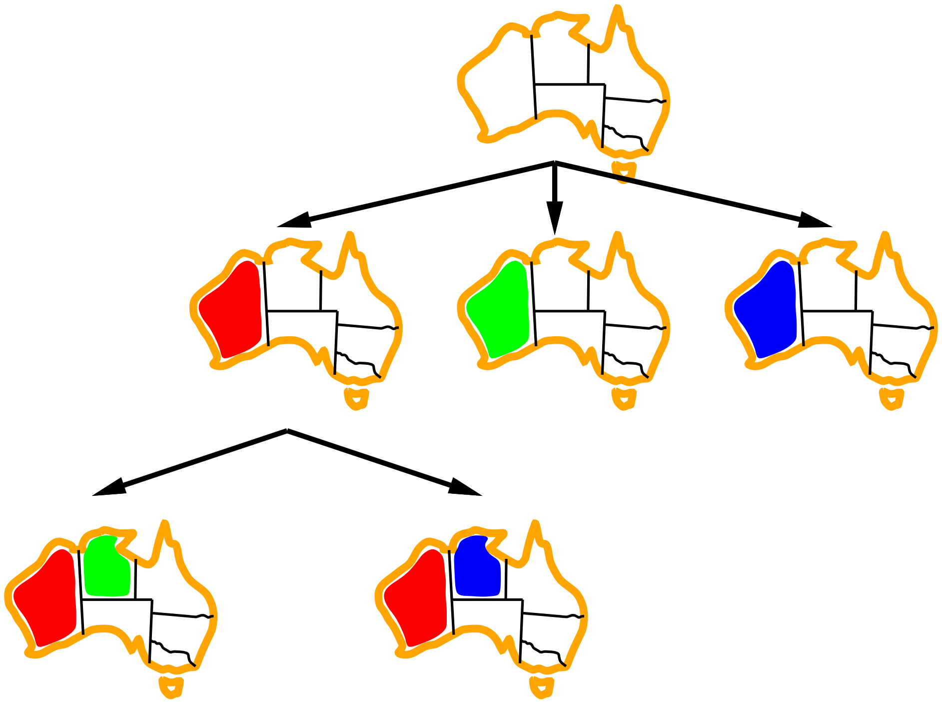
Exemplo de Procura com Retrocesso



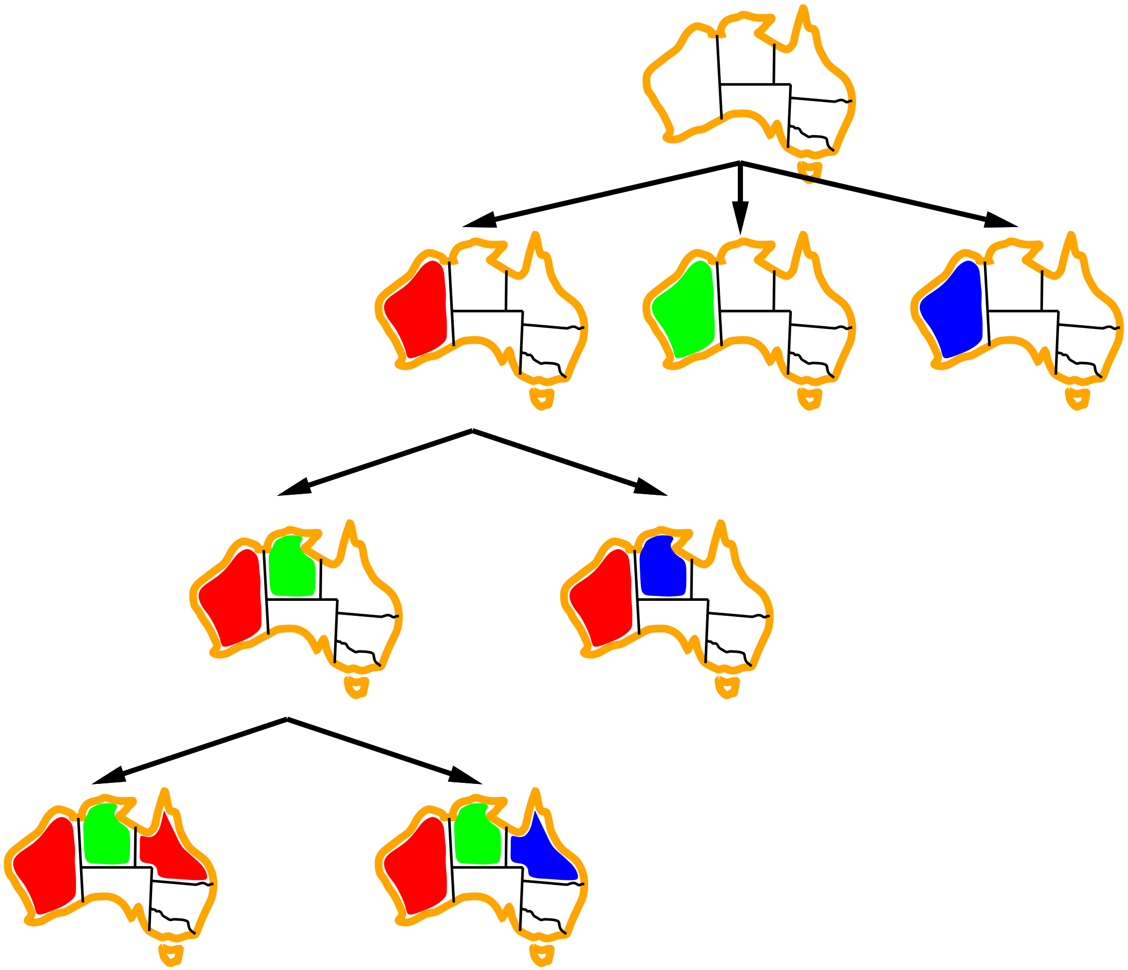
Exemplo de Procura com Retrocesso



Exemplo de Procura com Retrocesso



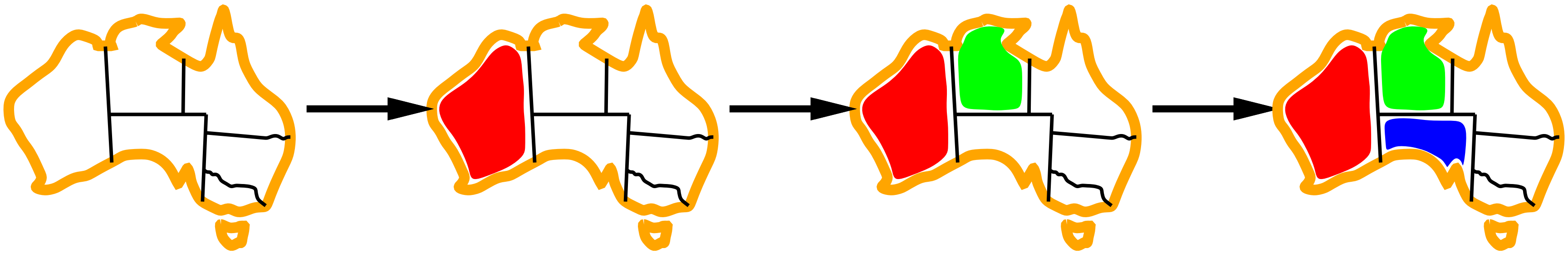
Exemplo de Procura com Retrocesso



Melhorando a eficiência do Retrocesso

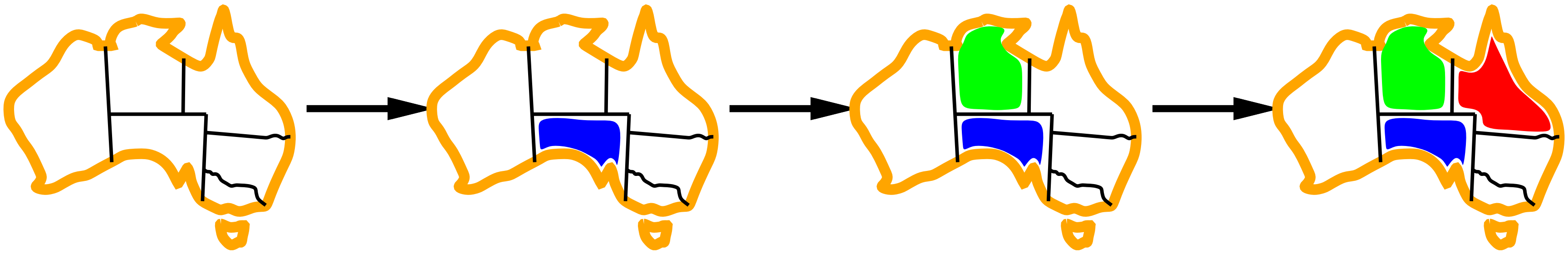
- Métodos genéricos podem resultar em ganhos substanciais de eficiência
  - 1. Qual a variável a atribuir?
  - 2. Por que ordem devem ser tentados os seus valores?
  - 3. Podem-se detectar falhas inevitáveis mais cedo?
  - 4. Pode-se utilizar vantajosamente a estrutura do problema?

Variável mais constrangida



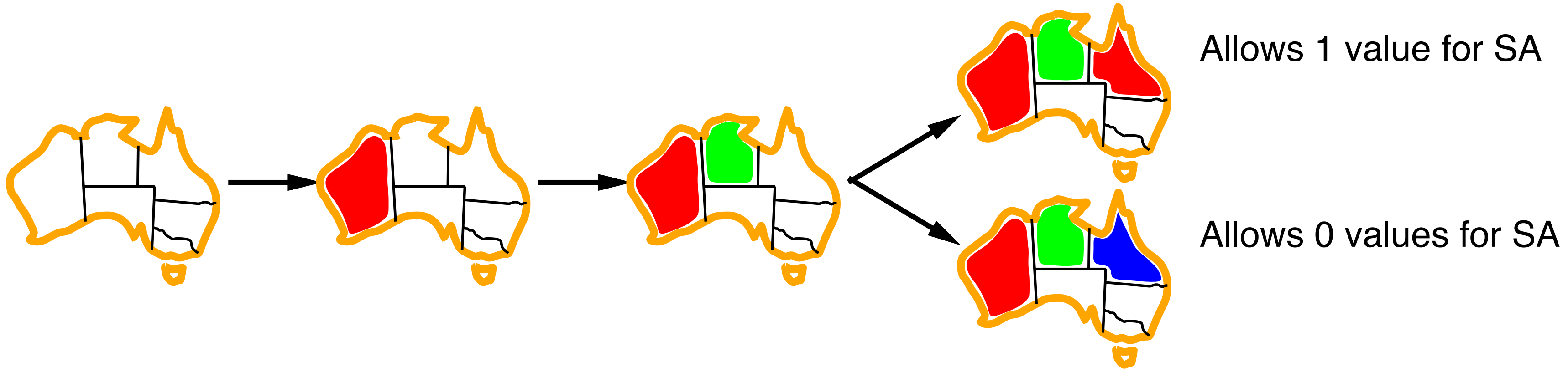
- Variável mais constrangida (MRV – minimum remaining values):
  - seleccionar a variável com menos valores possíveis

Variável mais constrangedora



- Desempate entre variáveis mais constrangidas
- De entre as variáveis mais constrangidas:
  - escolher a variável com maior número de restrições nas restantes variáveis livres

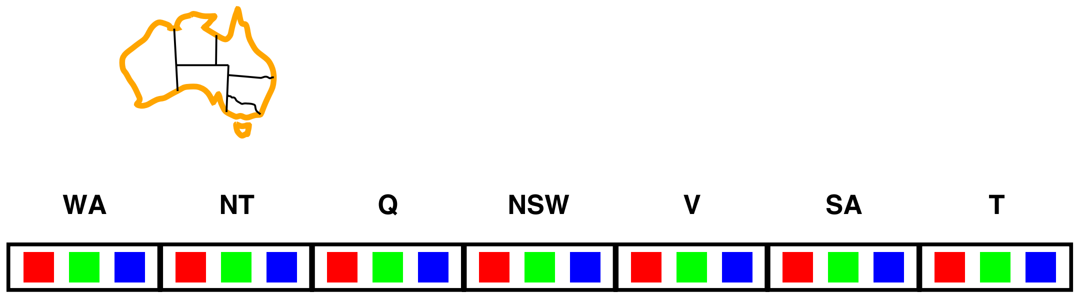
Valor menos restritivo



- Dado uma variável, escolher o valor menos restritivo:
  - aquele que eliminar menos valores nas restantes variáveis

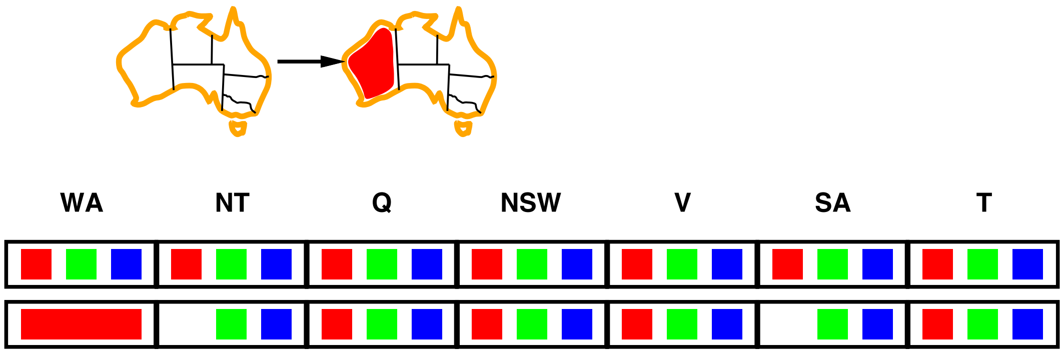
- Importante quando só estamos interessados numa solucão. Irrelevante para quando pretendemos obter todas as soluções, ou quando o problema não tem soluções.
- A combinação destas heursticas permite a resolução de problemas com 1000 rainhas.

Verificação para a frente



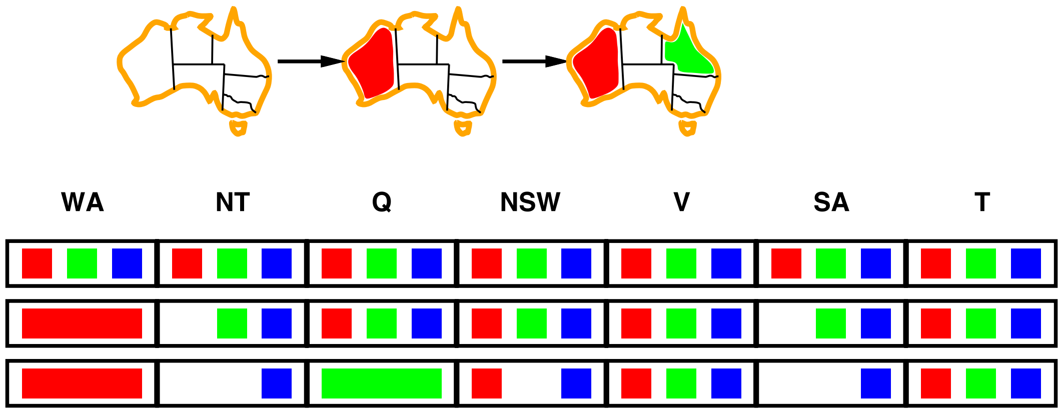
- Ideia: Vigiar os valores possíveis das variáveis por atribuir
- Termina a procura quando uma variável não possui valores possíveis

Verificação para a frente



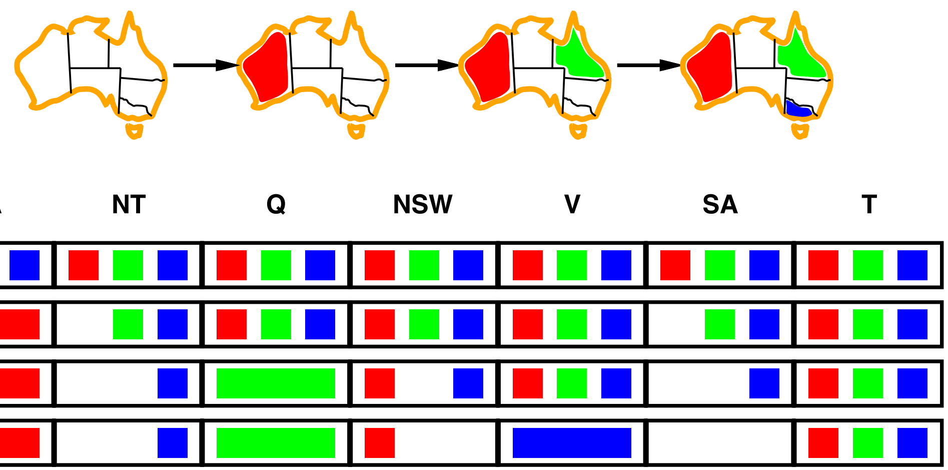
- Atribuír {WA=red}
- Efeitos nas variáveis ligadas a WA através de restrições:
  - NT deixa de poder ser red
  - SA deixa de poder ser red

Verificação para a frente



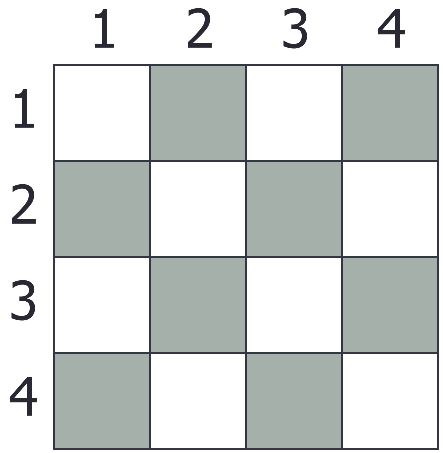
- Atribuír {Q=green}
- Efeitos nas variáveis ligadas a Q através de restrições:
  - NT deixa de poder ser green
  - NSW deixa de poder ser green
  - SA deixa de poder ser green
- A meta-heurística MRV (minimum remaining values) iria seleccionar NT ou SA a seguir. Porquê?

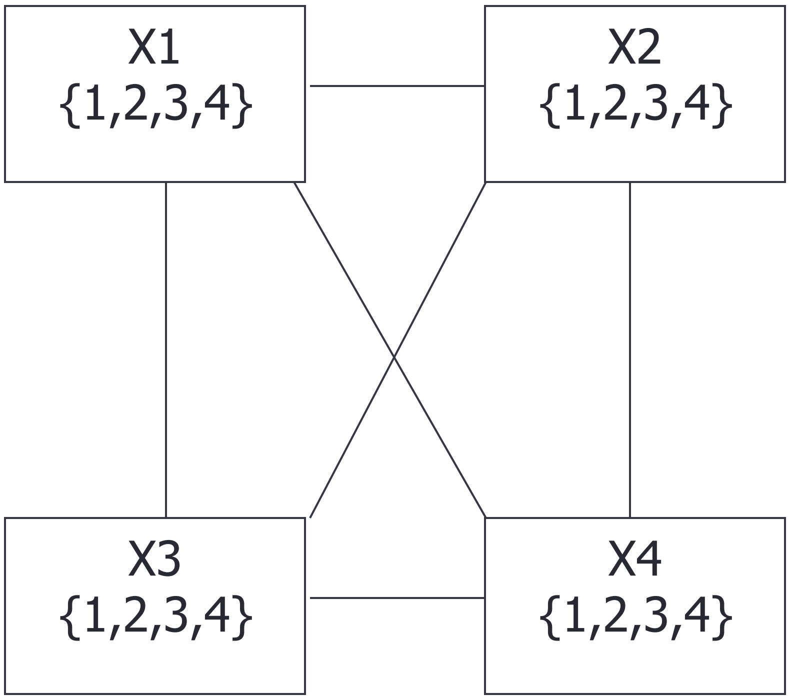
Verificação para a frente



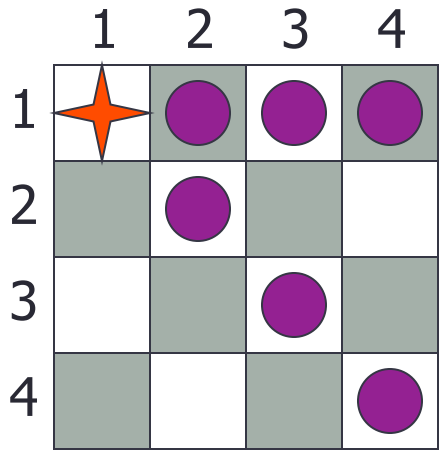
- Atribuír {V=blue}
- Efeitos nas variáveis ligadas a V através de restrições:
  - NSW deixa de poder ser blue
  - SA deixa de poder ser qualquer cor
- FC detecta que a atribuição parcial é inconsistente com as restrições, e o retrocesso (backtracking) ocorre.

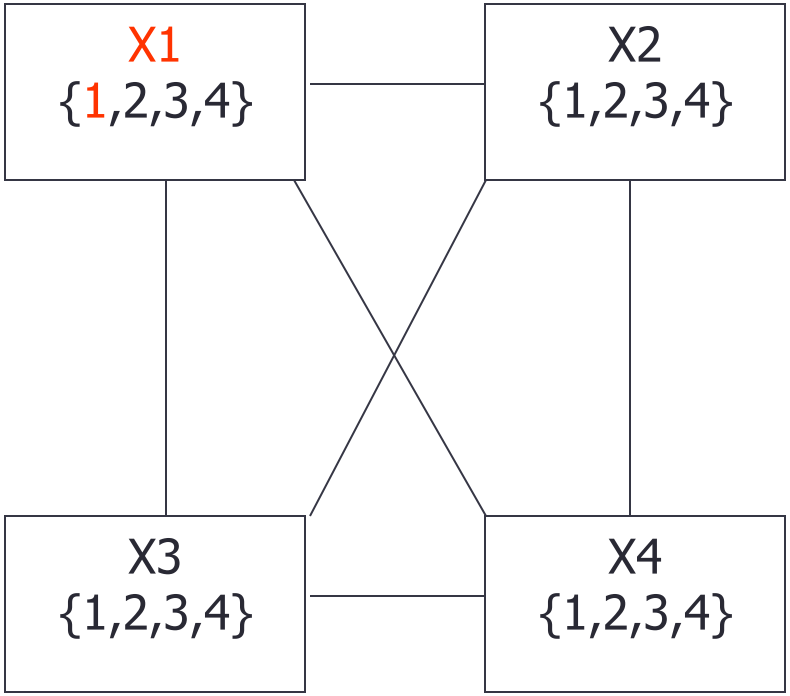
Exemplo: Problema das 4-rainhas



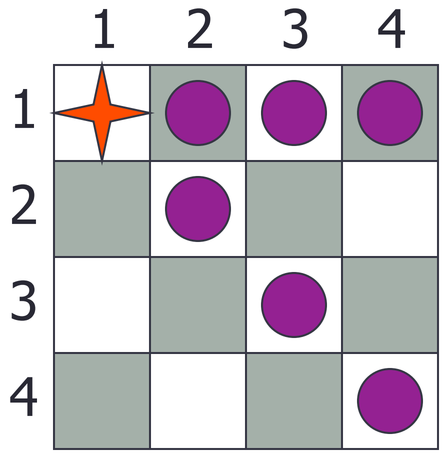


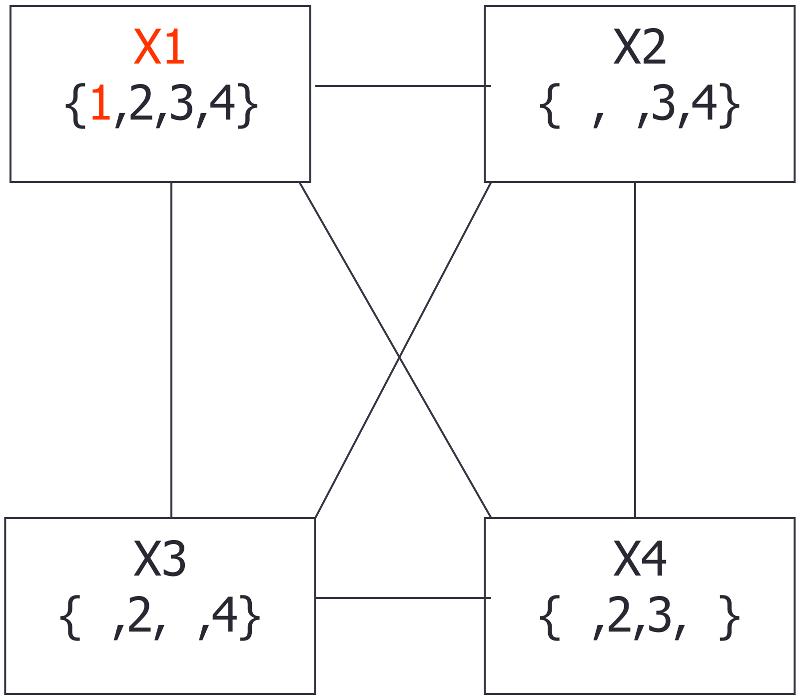
Exemplo: Problema das 4-rainhas



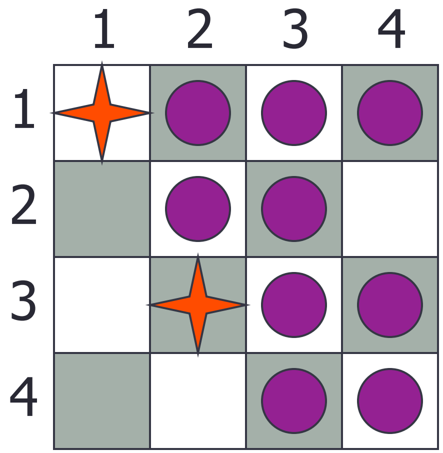


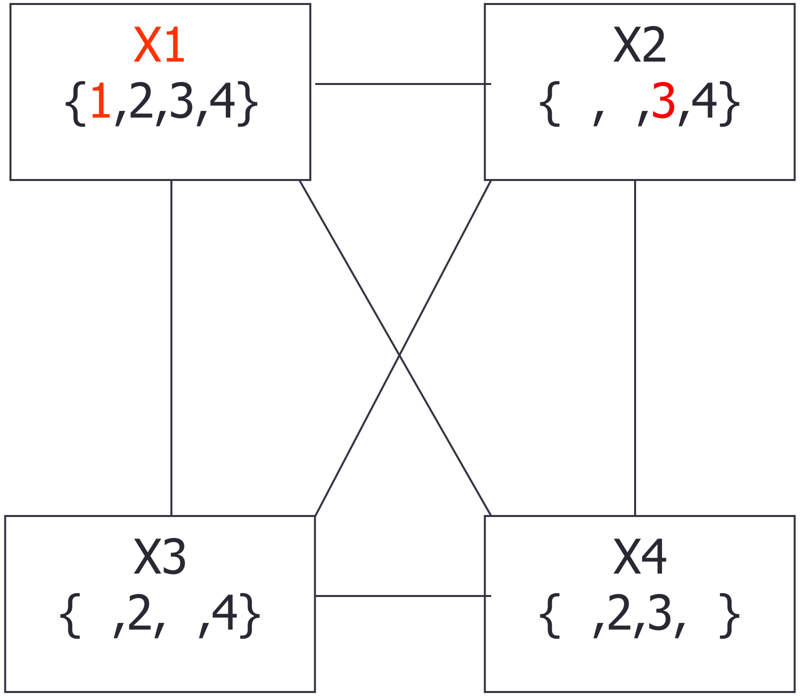
Exemplo: Problema das 4-rainhas



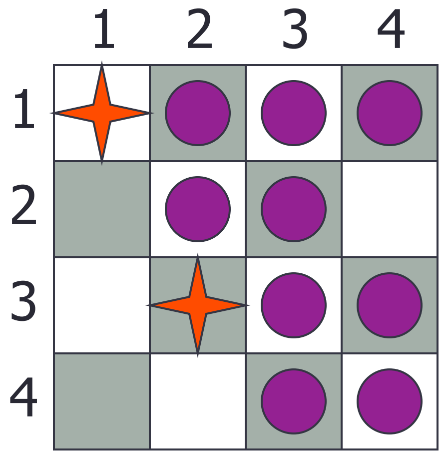


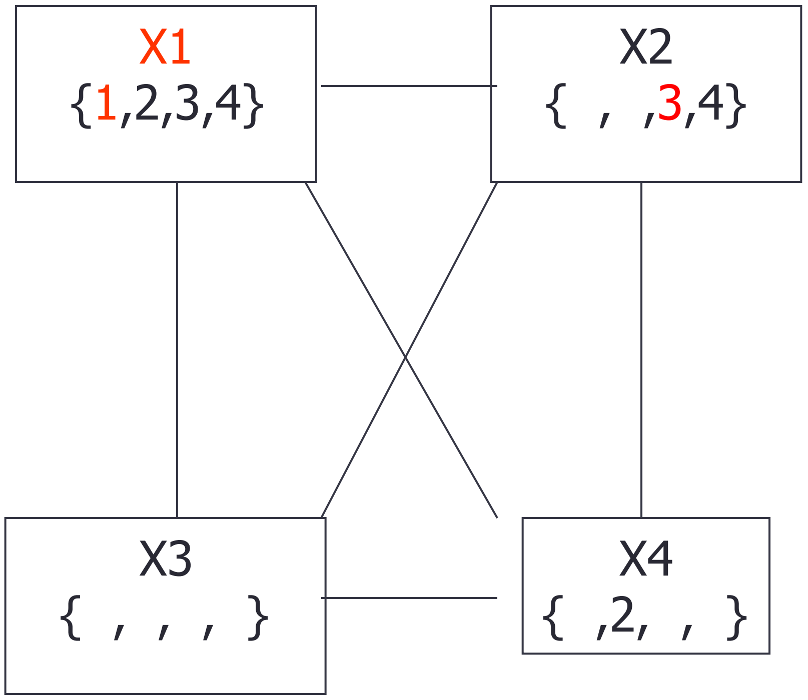
Exemplo: Problema das 4-rainhas



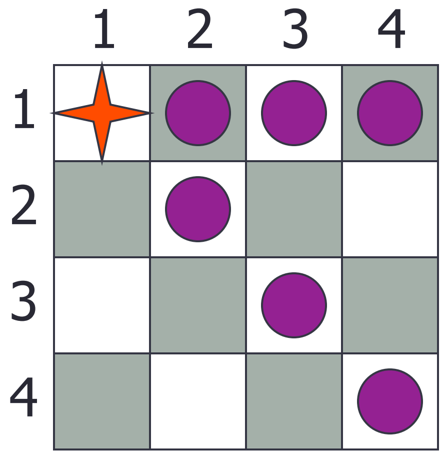


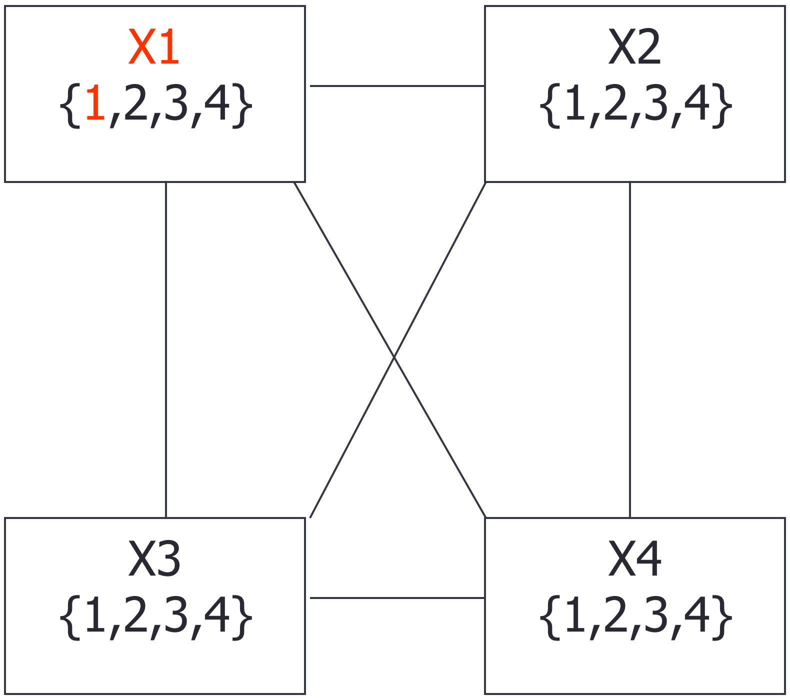
Exemplo: Problema das 4-rainhas



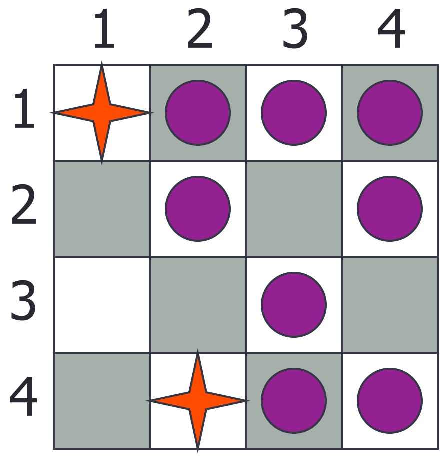


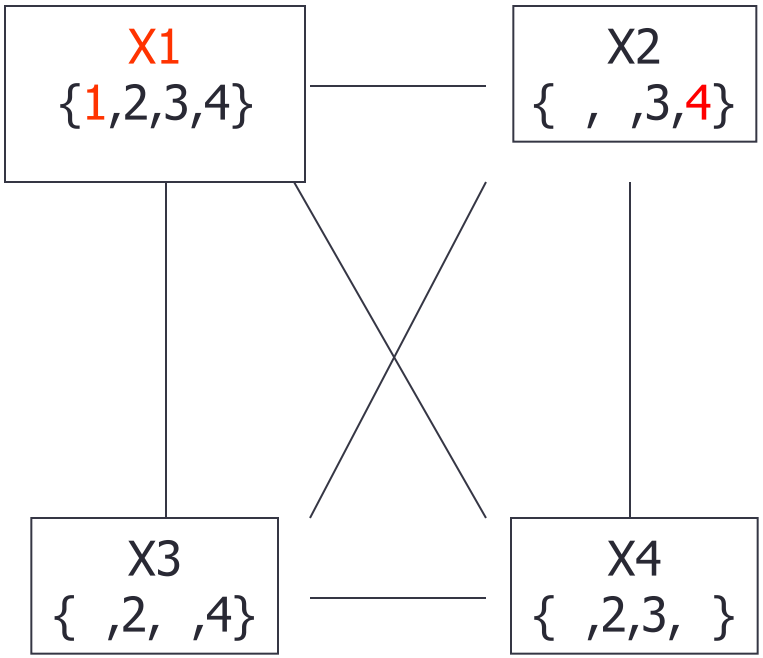
Exemplo: Problema das 4-rainhas



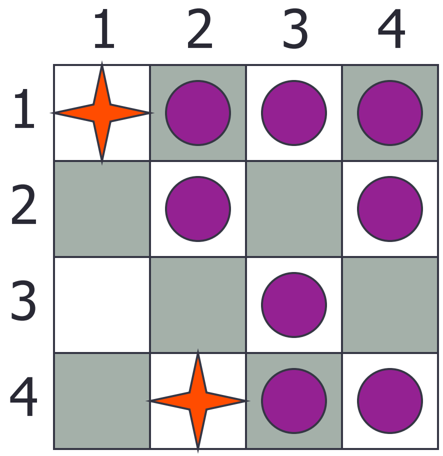


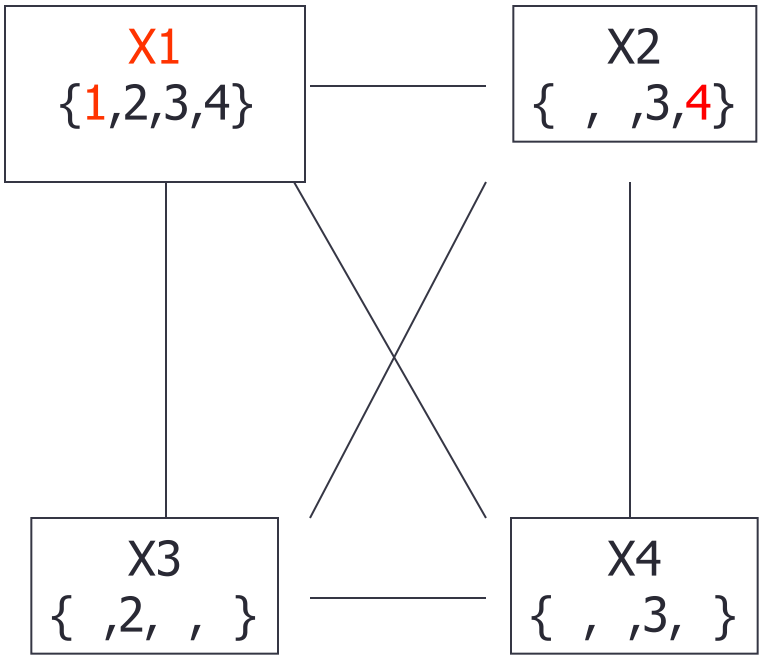
Exemplo: Problema das 4-rainhas



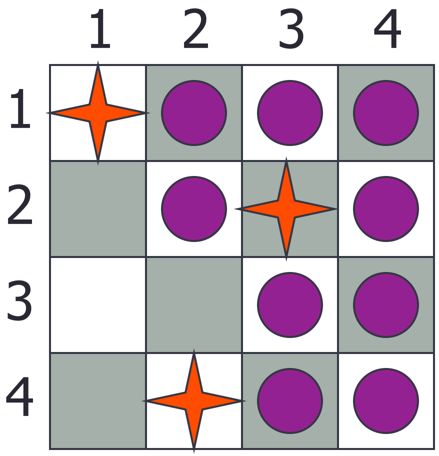


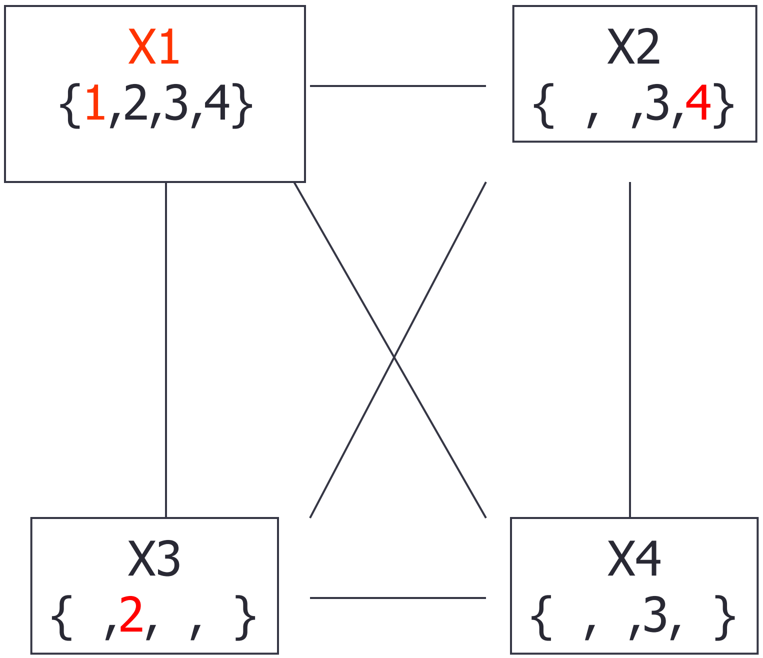
Exemplo: Problema das 4-rainhas



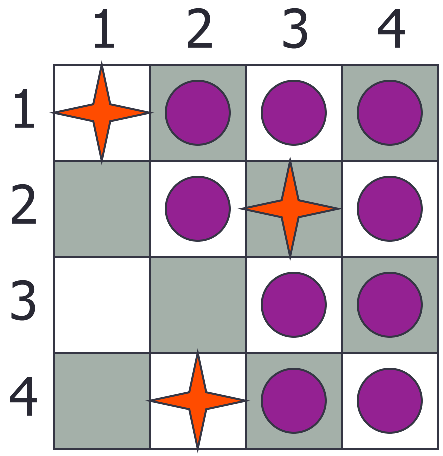


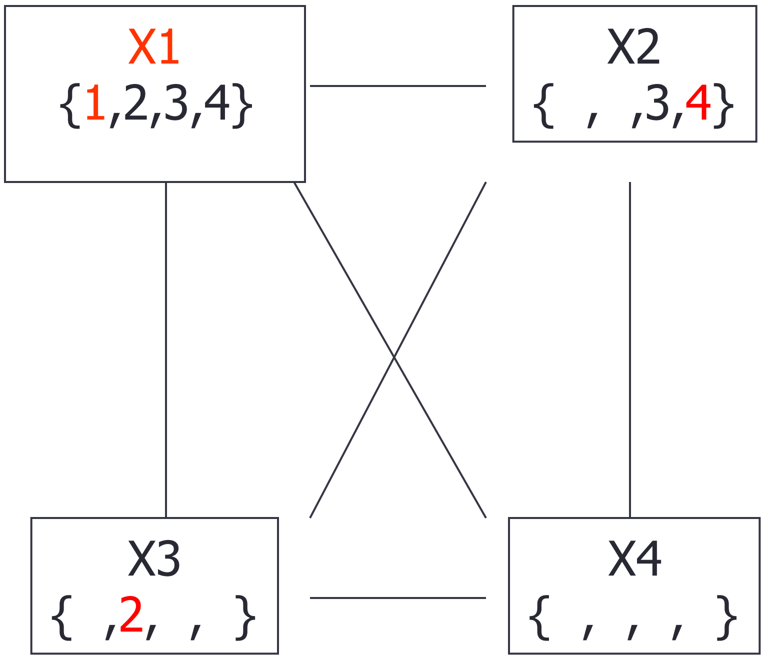
Exemplo: Problema das 4-rainhas



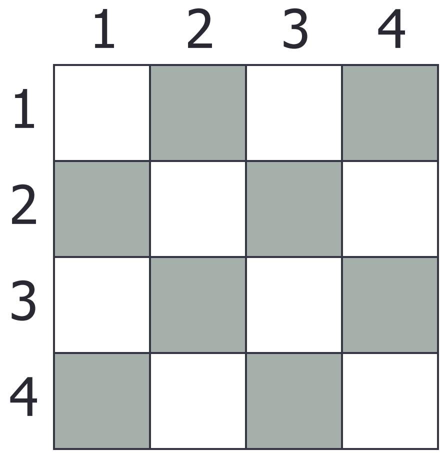


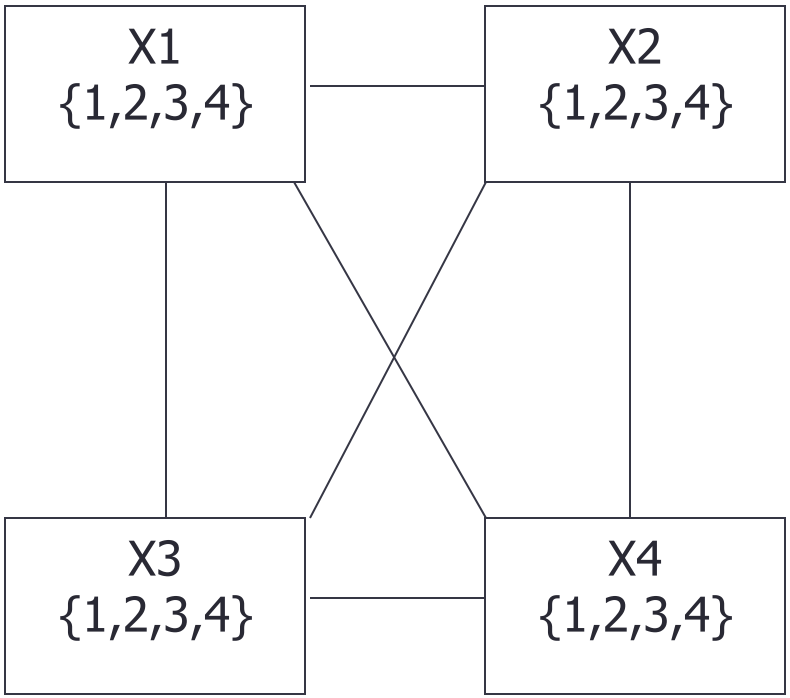
Exemplo: Problema das 4-rainhas



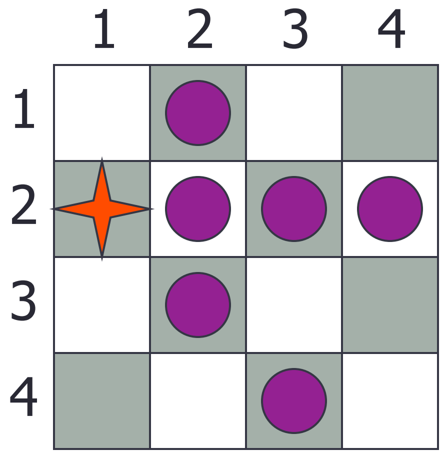


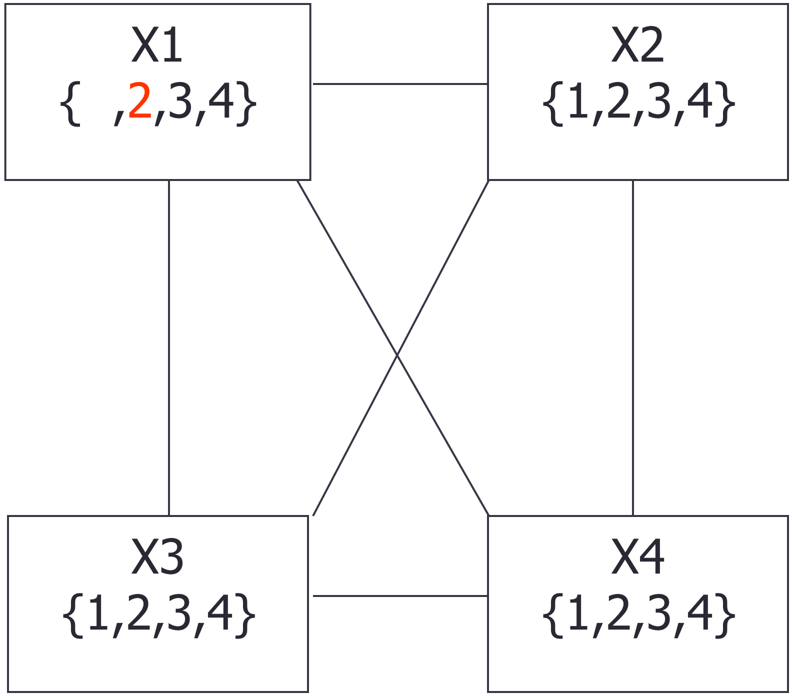
Exemplo: Problema das 4-rainhas



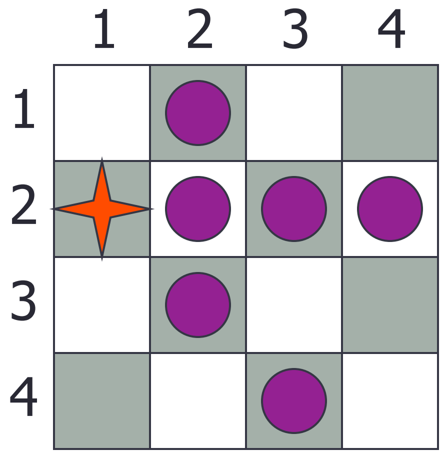


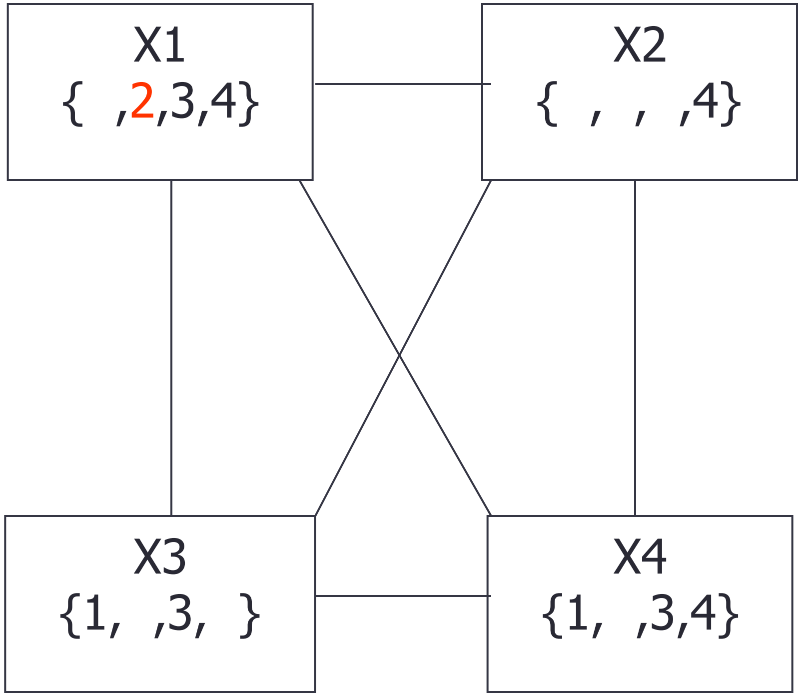
Exemplo: Problema das 4-rainhas



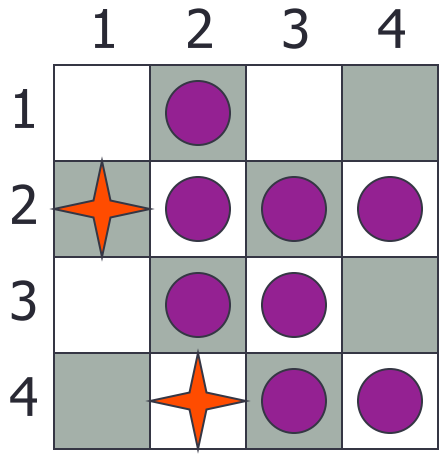


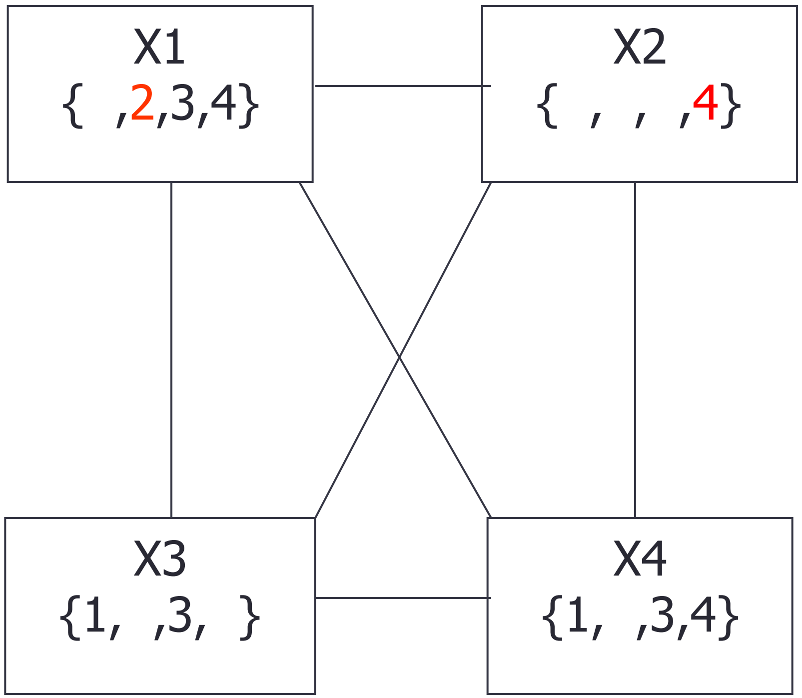
Exemplo: Problema das 4-rainhas



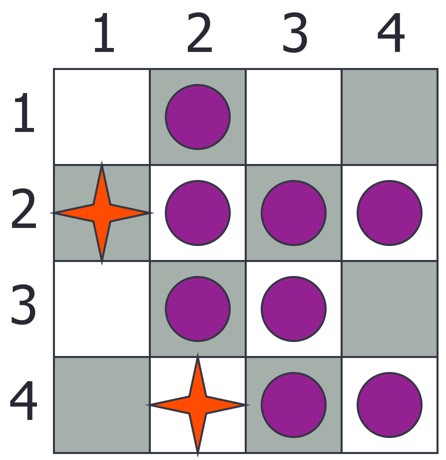


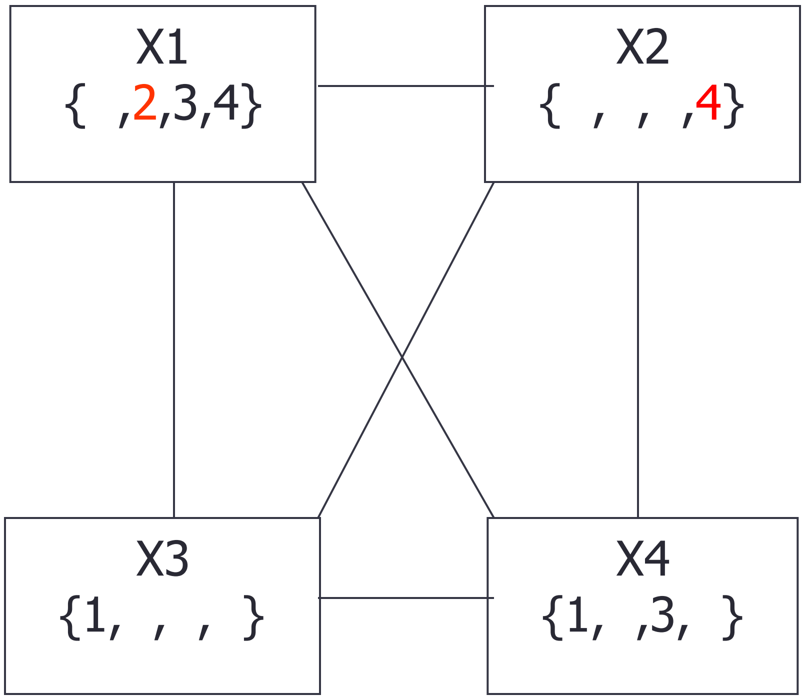
Exemplo: Problema das 4-rainhas



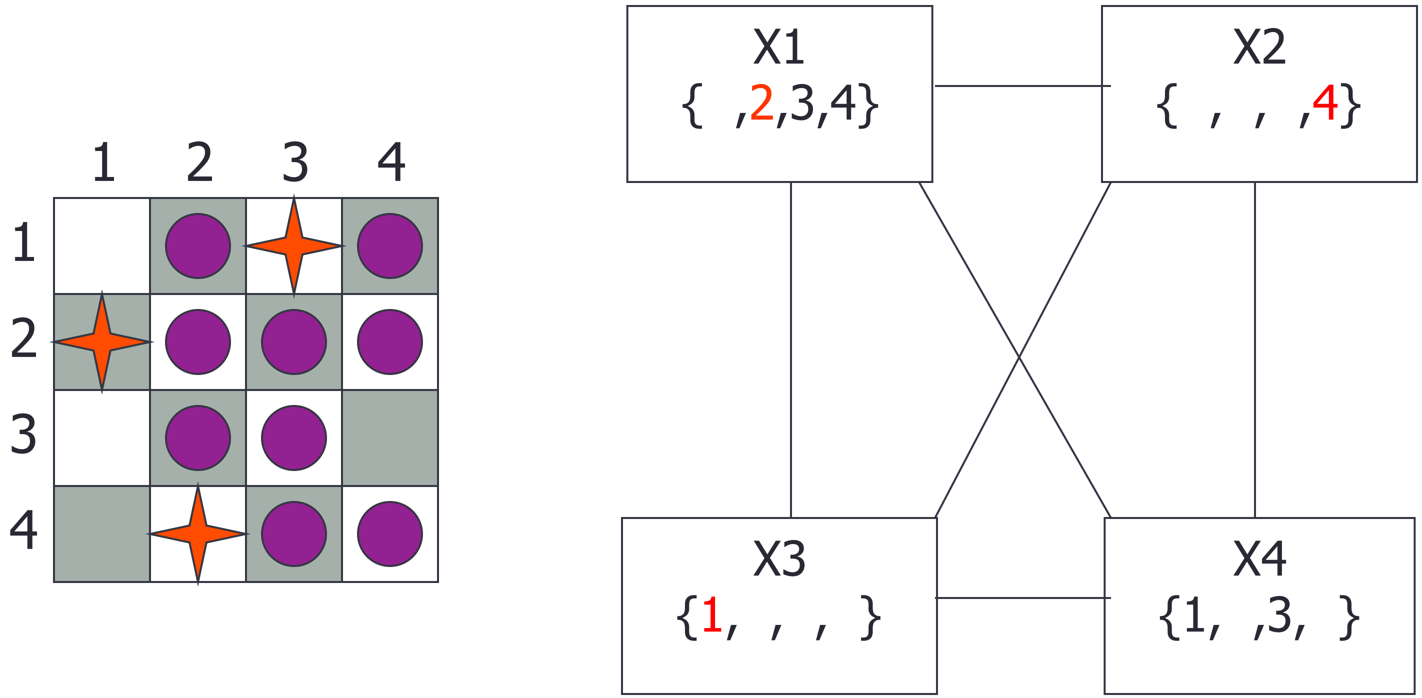


Exemplo: Problema das 4-rainhas

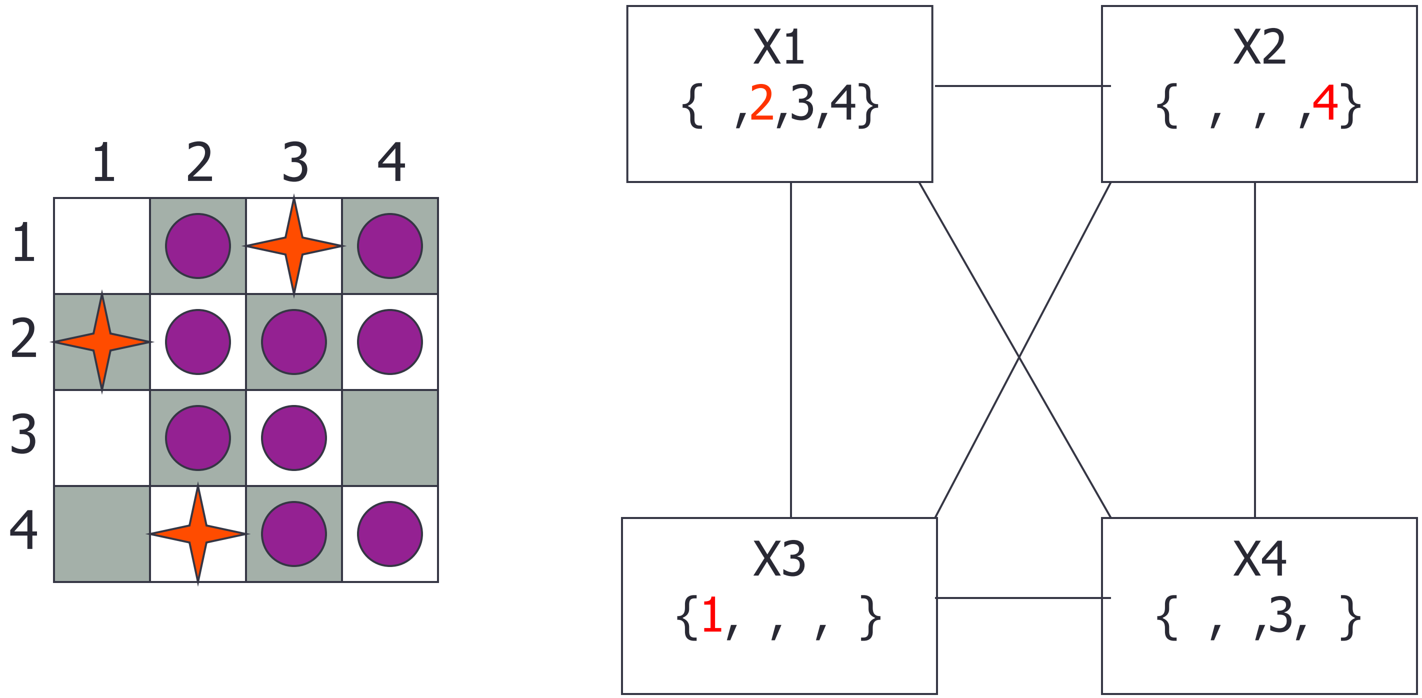




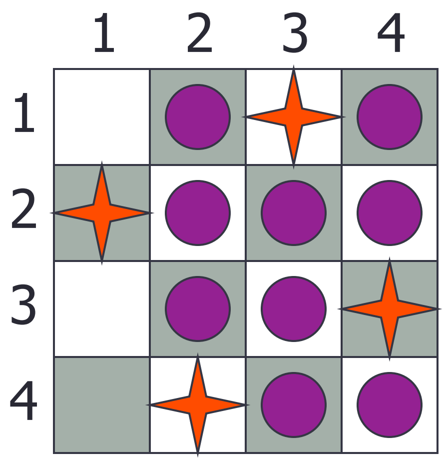
Exemplo: Problema das 4-rainhas

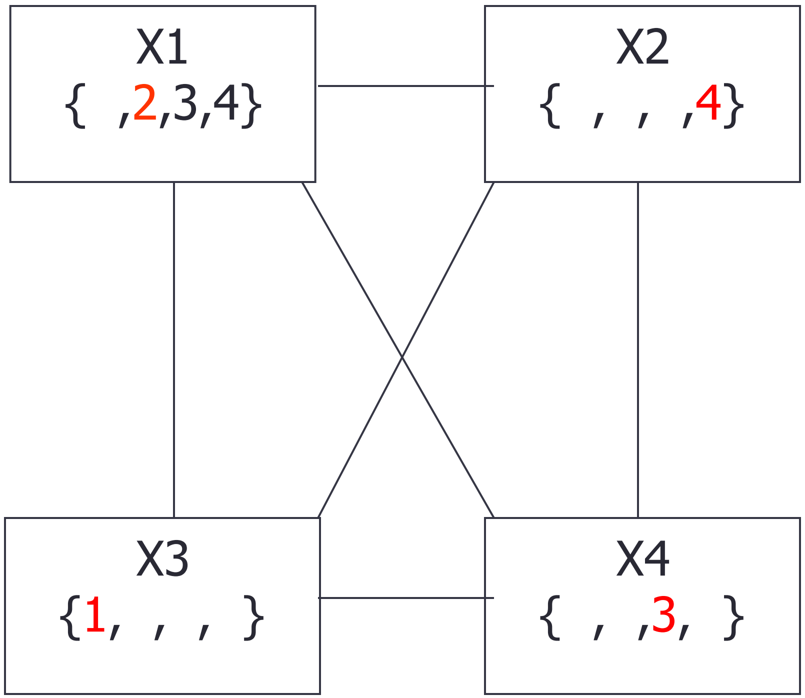


Exemplo: Problema das 4-rainhas



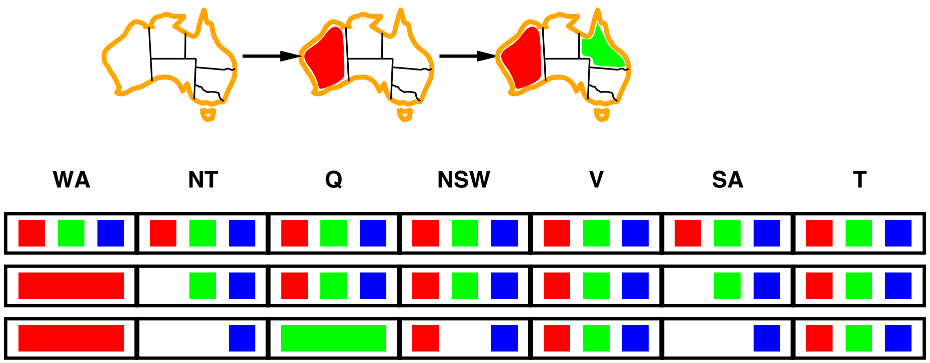
Exemplo: Problema das 4-rainhas





Propagação de Restrições

- A verificação para a frente propaga informação de variáveis atribuídas para variáveis não atribuídas, mas não permite a detecção prematura de todas as falhas:



- NT e SA não podem ser ambos azuis!
- Propagação de restrições obriga repetidamente a satisfação local de restrições

Consistência de Arcos

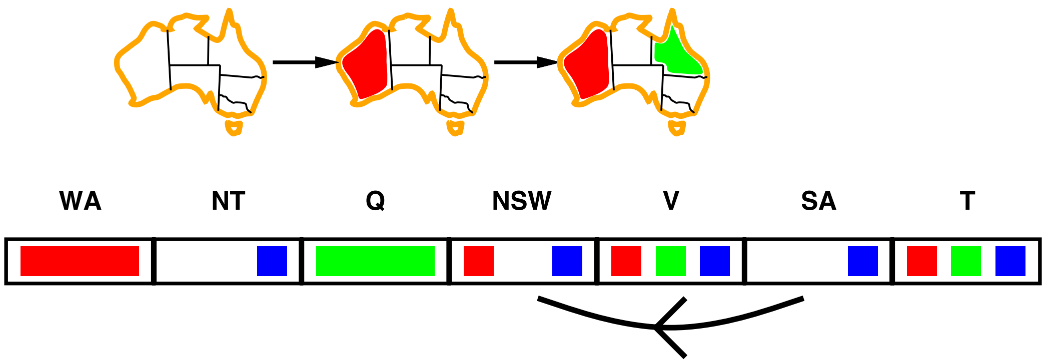
- A forma mais simples de propagação torna cada arco consistente
- $X \longrightarrow Y$ é consistente sse

para todo o valor x de X, então existe um valor permitido para y

Propagação de Restrições

- A forma mais simples de propagação torna cada arco consistente
- X → Y é consistente sse

para todo o valor x de X, então existe um valor permitido para y

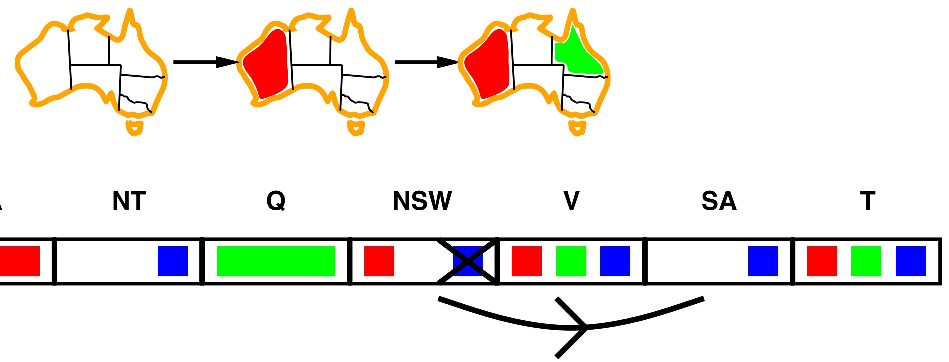


- SA → NSW é consistente pois para SA=blue há um valor permitido para NSW, nomeadamente red.

Propagação de Restrições

- A forma mais simples de propagação torna cada arco consistente
- X → Y é consistente sse

para todo o valor x de X, então existe um valor permitido para y

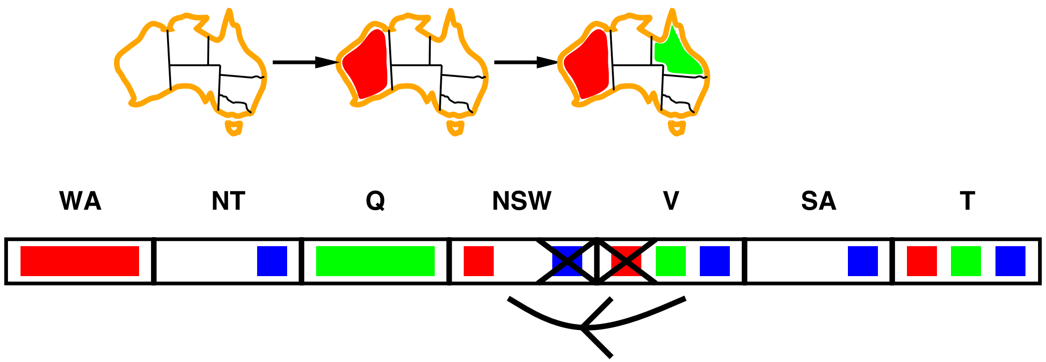


- Se X perde um valor, vizinhos de X precisam de ser reverificados.
- NSW → SA não é consistente pois:
  - para NSW=red há um valor permitido para SA, nomeadamente blue mas
  - para NSW=blue não há um valor permitido para SA.
- Arco pode recuperar consistência eliminado blue de NSW.

Propagação de Restrições

- A forma mais simples de propagação torna cada arco consistente
- $X \rightarrow Y$ é consistente sse

para todo o valor x de X, então existe um valor permitido para y

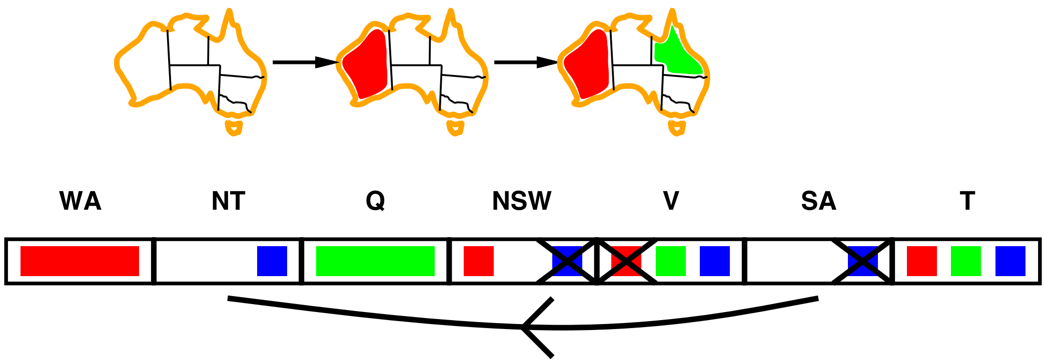


- Ao eliminar blue de NSW, é necessário reverificar os vizinhos:
  - retirar red de V.

Propagação de Restrições

- A forma mais simples de propagação torna cada arco consistente
- X → Y é consistente sse

para todo o valor x de X, então existe um valor permitido para y



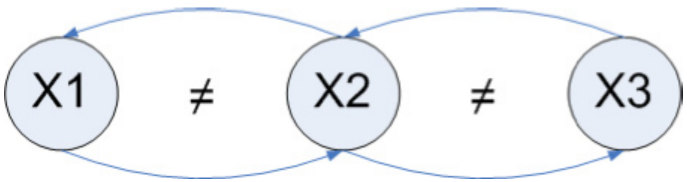
- A consistência de arcos detecta falhas mais cedo do que a verificação para a frente
- Pode ser utilizado como preprocessamento ou após cada atribuição

Algoritmo de Consistência de Arcos

function AC-3(*csp*) **return** the CSP, possibly with reduced domains
  inputs: *csp*, a binary csp with variables $\{X_1, \ldots, X_n\}$
  local variables: *queue,* a queue of arcs initially the arcs in *csp*
  while queue is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if REMOVE-INCONSISTENT-VALUES(csp, X_i, X_j) **then**
      if size of DOMAIN[X_i]=0 **then return** *false*
      for each X_k **in** NEIGHBORS[X_i] **do**
      add (X_k, X_i) to queue

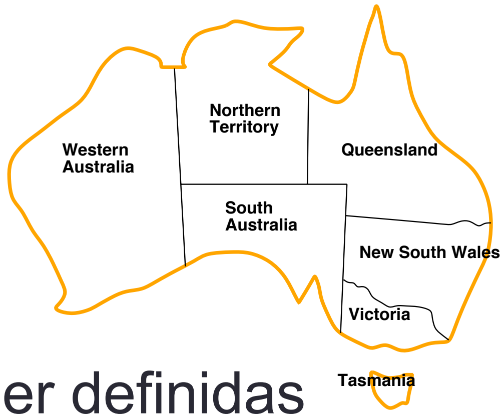
function REMOVE-INCONSISTENT-VALUES(csp X_i, X_j) **return** *true* iff it revises the domain of X_i
  removed $\leftarrow$ *false*
  for each x **in** DOMAIN[X_i] **do**
    if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraints $X_i \leftrightarrow X_j$
    then delete x from DOMAIN[X_i]; *removed* $\leftarrow$ *true*
  return *removed*

Exemplo de utilização do AC-3



X_1	X_2	X_3	Queue
{1}	{1,2}	{1,2}	$X_2 \rightarrow X_3, X_3 \rightarrow X_2, X_1 \rightarrow X_2, X_2 \rightarrow X_1$
{1}	{1,2}	{1,2}	$X_3 \rightarrow X_2, X_1 \rightarrow X_2, X_2 \rightarrow X_1$
{1}	{1,2}	{1,2}	$X_1 \rightarrow X_2, X_2 \rightarrow X_1$
{1}	{1,2}	{1,2}	$X_2 \rightarrow X_1$
{1}	{2}	{1,2}	$X_1 \rightarrow X_2, X_3 \rightarrow X_2$
{1}	{2}	{1,2}	$X_3 \rightarrow X_2$
{1}	{2}	{1}	$X_2 \rightarrow X_3$
{1}	{2}	{1}	

Consistência-k



- Consistência de arco não detecta todas as inconsistências.
  - {WA=red, NSW=red} é inconsistente.
- Formas mais fortes de consistência podem ser definidas usando a noção de consistência-k
- Consistência-k: um CSP é k-consistente se para qualquer conjunto de k-1 variáveis e qualquer atribuição consistente a essas variáveis, existe um valor possível para atribuir consistentemente a qualquer outra variável.
  - E.g. consistência-1 ou consistência de nó
  - E.g. consistência-2 ou consistência de arco
  - E.g. consistência-3 ou consistência de caminho
- Um algoritmo para estabelecer consistência-k toma um tempo exponencial em k, no pior caso.
  - Habitualmente opta-se pela consistência de arco ou consistência de caminho.

Melhorias ao algoritmo de retrocesso

- Vimos o algoritmo com retrocesso cronológico (em caso de falha retorna à última variável atribuída).
- O retrocesso inteligente consiste em retornar à variável anterior responsável pela violação da restrição.
- Tecnicamente isto é conseguido mantendo para cada variável o conjunto de variáveis confituantes, e actualizando-o incrementalmente (e.g. quando RemoveInconsistent retira valores ao domínio).

Algoritmos locais para CSPs

- Trepa-colinas, recristalização simulada funcionam habitualmente com estados "completos", i.e., todas as variáveis atribuídas
- Aplicação a CSPs:
  - permitir estados com restrições por satisfazer
  - os operadores re-atribuem valores a variáveis
- Selecção de Variáveis: escolher aleatoriamente qualquer variável conflituante
- Selecção de Valores: por intermédio da heurística min-conflitos:
  - escolher valor que viola o menor número de restrições
  - i.e., trepa-colinas com h(n) = número total de restrições violadas

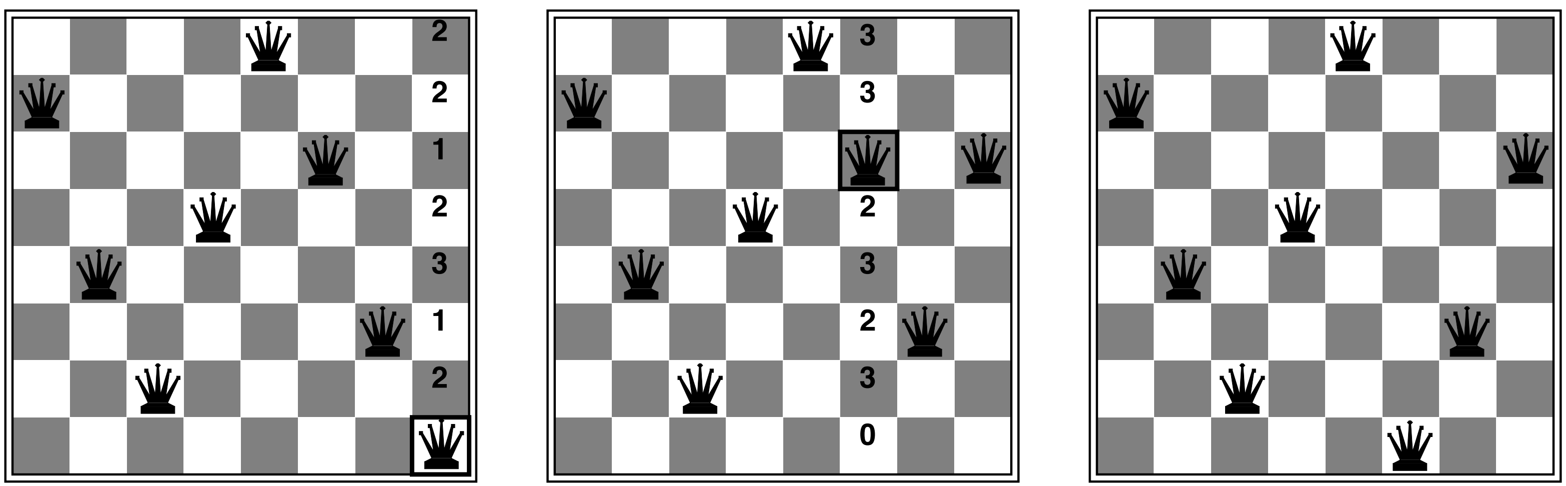
Algoritmo min-conflitos

function MIN-CONFLICTS(*csp, max_steps*) **return** solution or failure
  inputs: *csp*, a constraint satisfaction problem
    max_steps, the number of steps allowed before giving up
  local variables: *current*, a complete assignment
    var, a variable
    value, a value for a variable

  current ← an initial complete assignment for *csp*
  for *i* = 1 to *max_steps* **do**
    if *current* is a solution for *csp* then return *current*
    var ← a randomly chosen, conflicted variable from VARIABLES[*csp*]
    value ← the value *v* for *var* that minimizes CONFLICTS(*var,v,current,csp*)
    set *var* = *value* in *current*
  return *failure*

Exemplo: 8-rainhas

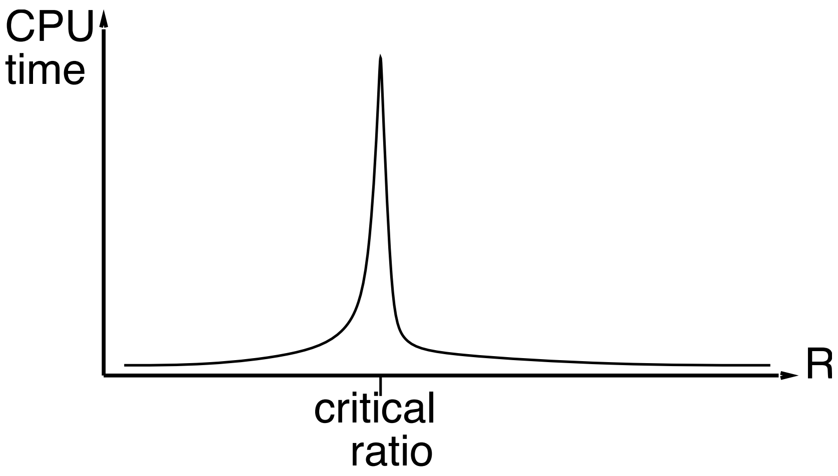
- Estado: 8 rainhas em 8 colunas (8^8 = 16777216 estados)
- Operadores: mover uma rainha na sua coluna
- Teste Objectivo: inexistência de ataques
- Avaliação: h(n) = número de ataques



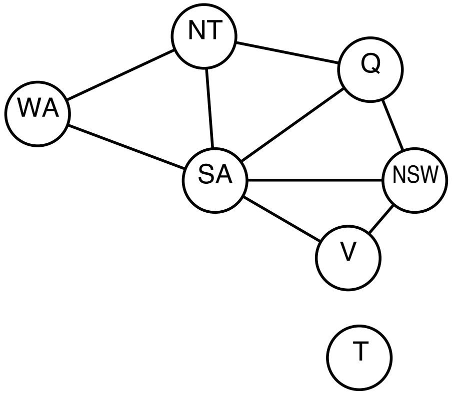
Desempenho de min-conflitos

- Dado um estado inicial aleatório, consegue-se resolver n-rainhas quase em tempo constante para n arbitrário com alta probabilidade (e.g., n = 10,000,000)
- O mesmo se verifica para qualquer CSP gerado aleatoriamente, exceptuando para uma pequena banda do rácio

R = número de restrições / número de variáveis

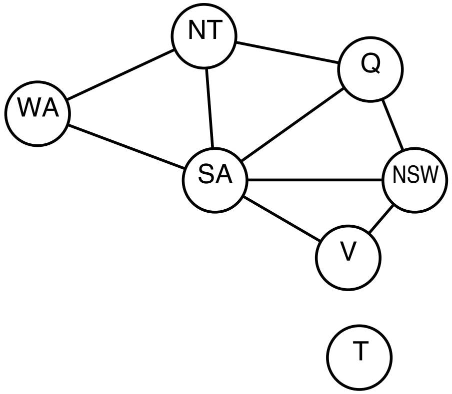


Estrutura dos Problemas



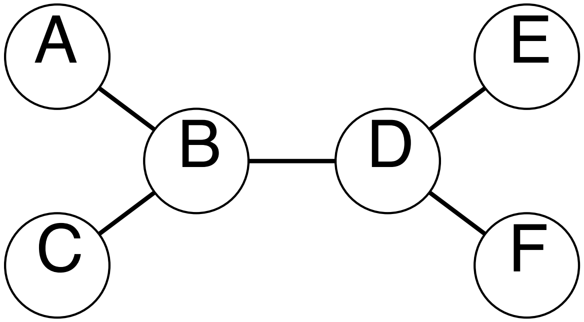
- A Tasmânia e o continente são sub-problemas independentes
- Identificáveis a partir das componentes ligadas do grafo de restrições

Estrutura dos Problemas (cont)



- Suponha-se que cada subproblema tem c variáveis de um total de n
- No pior caso, uma solução tem um custo de $O(n/c \cdot d^c)$ i.e. linear em n
  - Em vez de $O(d^n)$, exponencial em n.
- E.g., n=80, d=2, c=20
  - 2^{80} = 4 mil milhões de anos a 10 milhões de nós/seg
  - $4 \cdot 2^{20}$ = 0.4 segundos a 10 milhões de nós/seg

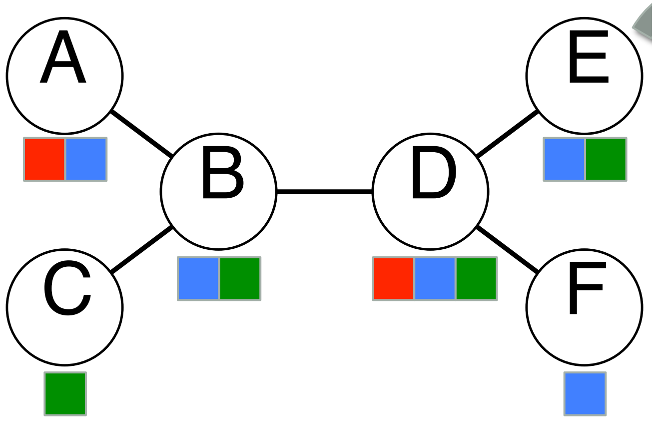
CSPs com estrutura arbórea



- Teorema: se o grafo de restrições não tem ciclos, então o CSP pode ser resolvido em tempo $O(nd^2)$
- Comparar com CSPs genéricos, cujo pior caso temporal é $O(d^n)$
- Esta propriedade também se aplica ao raciocínio lógico e probabilístico:
  - um exemplo importante da relação entre restrições sintácticas e a complexidade do raciocínio.

Algoritmo para CSPs com estrutura arbórea

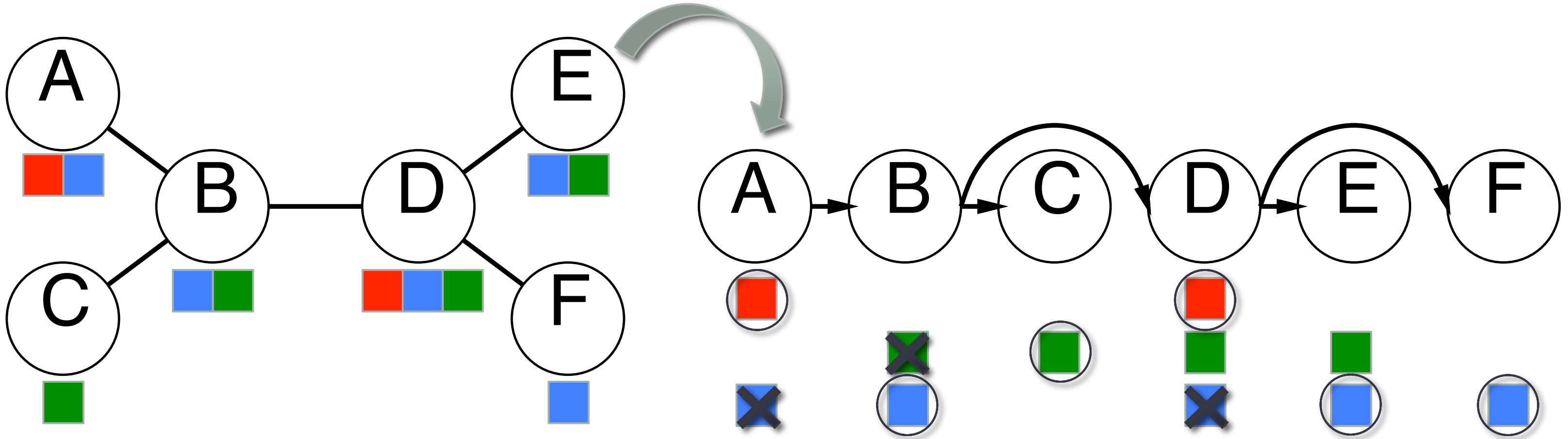
- Escolher uma variável como raiz, ordenar variáveis da raiz para as folhas tal que o pai de um nó aparece primeiro do que todos os seus filhos na ordenação.



- Para j de n até 2, aplicar RemoveInconsistent(Parent(X_j),X_j)
- Para j de 1 até n, atribuir X_j consistentemente com Parent(X_j)

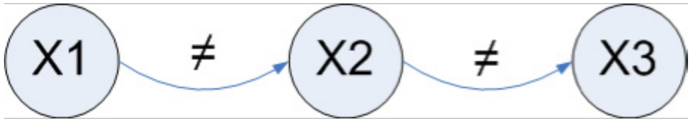
Algoritmo para CSPs com estrutura arbórea

- Escolher uma variável como raiz, ordenar variáveis da raiz para as folhas tal que o pai de um nó aparece primeiro do que todos os seus filhos na ordenação.

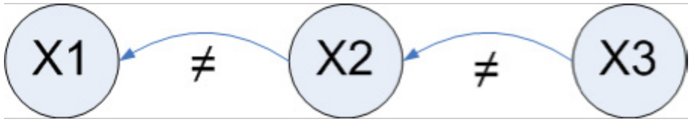


- Para j de n até 2, aplicar RemoveInconsistent(Parent(X_j),X_j)
- Para j de 1 até n, atribuir X_j consistentemente com Parent(X_j)

Aplicação a exemplo



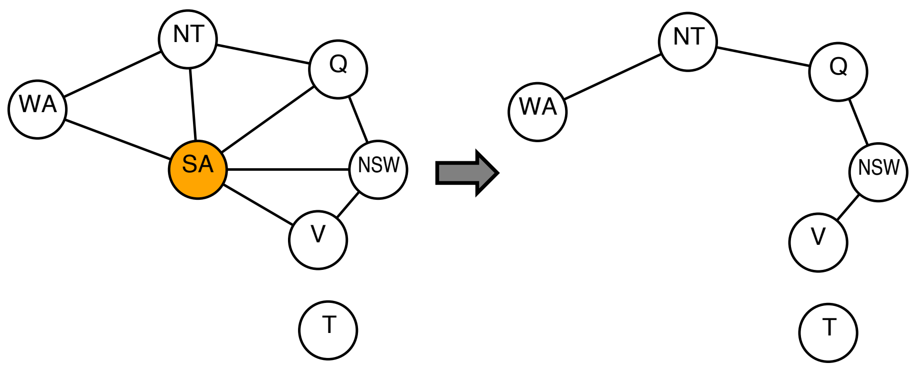
X_1	X_2	X_3	**Acção**
{1}	{1,2}	{1,2}	RemoveInconsistent(X_2,X_3)
{1}	{1,2}	{1,2}	RemoveInconsistent(X_1,X_2)
{1}	{1,2}	{1,2}	X_1 := 1
{1}	{2}	{1,2}	X_2 := 2
{1}	{2}	{1}	X_3 := 1



X_1	X_2	X_3	**Acção**
{1}	{1,2}	{1,2}	RemoveInconsistent(X_2,X_1)
{1}	{2}	{1,2}	RemoveInconsistent(X_3,X_2)
{1}	{2}	{1}	X_1 := 1

CSPs de estrutura quasi-arbórea

- Condicionamento: instanciar uma variável, reduzir os domínios dos nós vizinhos



- Condicionamento por conjunto de corte: instanciar (de todas as maneiras) um conjunto de variáveis tal que o grafo resultante seja uma árvore
- Conjunto de corte de tamanho c ⇒ tempo de execução $O(d^c \cdot (n-c)d^2)$, muito rápido para c pequeno

Sumário

- CSPs são um tipo especial de problema:
  - estados definidos por valores de um conjunto fixo de variáveis
  - teste objectivo definido por restrições nos valores das variáveis
- Retrocesso = procura em profundidade primeiro como uma variável atribuída por nó
- Heurísticas de ordenação de variáveis e selecção de valores ajudam muito
- Verificação para a frente evita atribuições que irão falhar garantidamente
- Propagação de restrições (e.g., consistência de arcos) efectua trabalho adicional para restringir valores e detectar inconsistências mais cedo
- O algoritmo min-conflitos é habitualmente eficaz na prática
- A representação de CSPs permite a análise da estrutura do problema
- CSPs de estrutura arbórea podem ser resolvidos em tempo linear