CTC-17 Inteligência Artificial
Problema de Satisfação de Restrições

Prof. Paulo André Castro
pauloac@ita.br
www.comp.ita.br/~pauloac
Sala 110,
IEC-ITA

Sumário

- Conceituação
- Aplicando busca genérica a PSRs
- Melhorando desempenho com o Algortimo de Backtracking
  - Verificação Forward por Valor restantes mínimo
- Heurísticas para PSRs
- Otimização

Exemplos de PSRs no mundo real

- Problemas de indicação
  - Exemplo: quem ensina que curso, que tripulação faz qual vôo
- Problemas de organização
  - Qual curso é oferecido quando e onde?
  - Configuração de hardware
- Problemas de fluxo de transporte
- Planificação em fábricas
- Alocação de freqüências em áreas, etc.
- Observe que muitos problemas reais envolvem variáveis reais

Problema de Satisfação de Restrições

Problema padrão:

Um estado é um "caixa preta"—qualquer estrutura de dados que suporte testes de objetivo, avaliação, sucessor

PSR:

estado é definido por variáveis V_i assumindo *valores* do domínio D_i

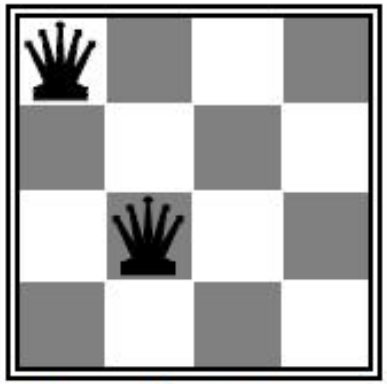
o teste de objetivo é um conjunto de restrições especificando combinações permissíveis de valores para subconjuntos de variáveis

Exemplo simples de uma *linguagem de representação formal*

Permite o uso de algoritmos de *propósito geral* mais poderosos que algoritmos de busca usuais

Exemplo: 4 Rainhas como um PSR

Assuma uma rainha em cada coluna. Em qual fila entra cada uma?



$Q_1 = 1 \quad Q_2 = 3$

Variáveis Q_1, Q_2, Q_3, Q_4

Domínios $D_i = \{1, 2, 3, 4\}$

Restrições

$Q_i \neq Q_j$ (ñ pode estar na mesma coluna)

$|Q_i - Q_j| \neq |i - j|$ (ou na mesma diagonal)

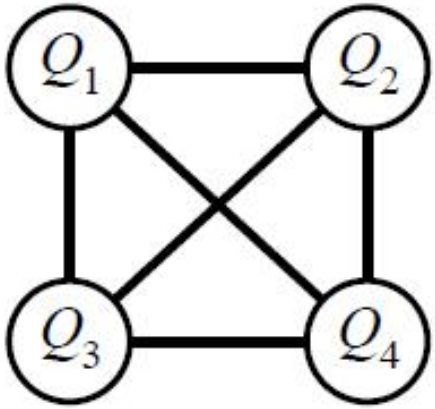
Traduza cada restrição como um conjunto de valores permissíveis para suas variáveis

Por exemplo, valores para (Q_1, Q_2) são $(1,3)$ $(1,4)$ $(2,4)$ $(3,1)$ $(4,1)$ $(4,2)$

Grafo de restrições

PSR Binário: cada restrição relaciona duas variáveis no máximo

Grafo de restrições: nós são variáveis, arcos mostram as restrições



Exemplo 2: Coloração de Mapa

Colorir um mapa de forma que países adjacentes não tenham a mesma cor

Variaveis

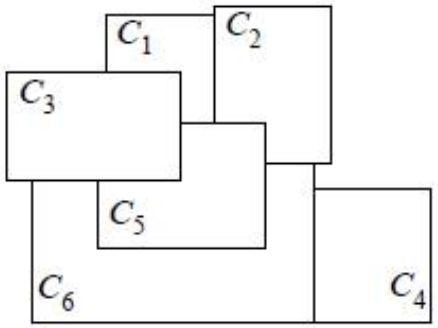
Países C_i

Domínios

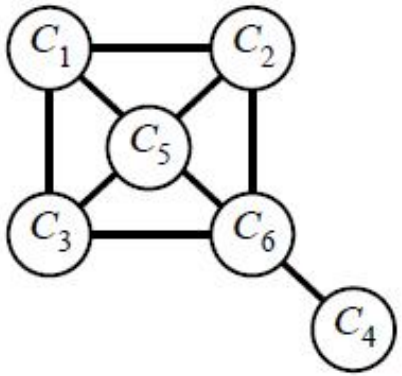
$\{Vermelho, Azul, Verde\}$

Restrições

$C_1 \neq C_2$, $C_1 \neq C_5$, etc.



Grafo de restrições:



Aplicação da Busca Genérica

- Iniciaremos com um método bastante simples (ingênuo) e depois aperfeiçoaremos
- Estados são definidos pelos valores nomeados até o momento
- Estado inicial: nenhuma variável nomeada
- Operadores: indicar um valor a uma variável não nomeada
- Teste de objetivos: todas as variáveis nomeadas, nenhuma restrição violada

- Observe que isto ocorre para todos os PSRs.

Implementação

O estado em um PSR mantém rastro de que variáveis tiveram valores nomeados até o momento

Cada variável tem um domínio e um valor atual

datatype CSP-STATE
  components: UNASSIGNED, a list of variables not yet assigned
  ASSIGNED, a list of variables that have values

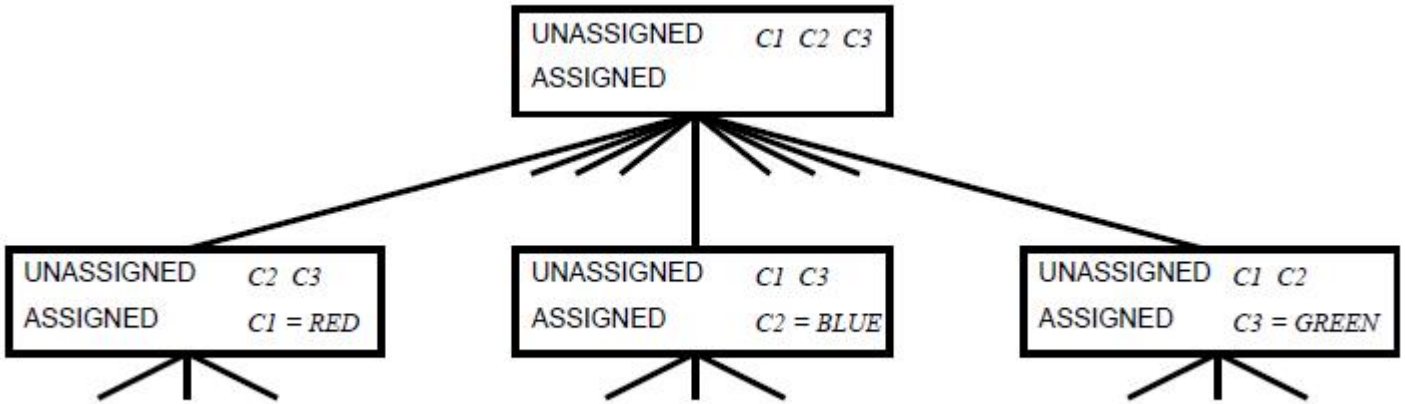
datatype CSP-VAR
  components: NAME, for i/o purposes
  DOMAIN, a list of possible values
  VALUE, current value (if any)

As restrições podem ser representadas

explicitamente como jogos de valores permissíveis, ou

implicitamente por uma função que testa a satisfação da restrição

Busca genérica no coloramento de gráficos



Avaliação da Solução Ingênua ?

Profundidade max. do espaço m??

Profundidade do espaço de soluções d??

Algoritmo de busca a utilizar??

Fator de ramificação b??

Avaliação da Solução Ingênua

Profundidade max. do espaço m?? n (número de variáveis)

Profundidade do espaço de soluções d?? n (todas as vars. nomeadas)

Algoritmo de busca a utilizar?? *depth-first*

Fator de ramificação b?? $\sum_i |D_i|$ (no topo da árvore)

Isto pode ser melhorado dramaticamente observando-se o seguinte:

1) a ordem das nomeações é irrelevante, portanto muitos caminhos são equivalentes
2) as nomeações adicionadas não podem corrigir uma restrição violada

Solução Melhorada: Backtracking

Uso busca *depth-first*, mas

1) fixo a ordem de nomeações, $\Rightarrow b = |D_i|$
   (pode ser feito na função SUCESSORS)
2) verifico violações de restrições

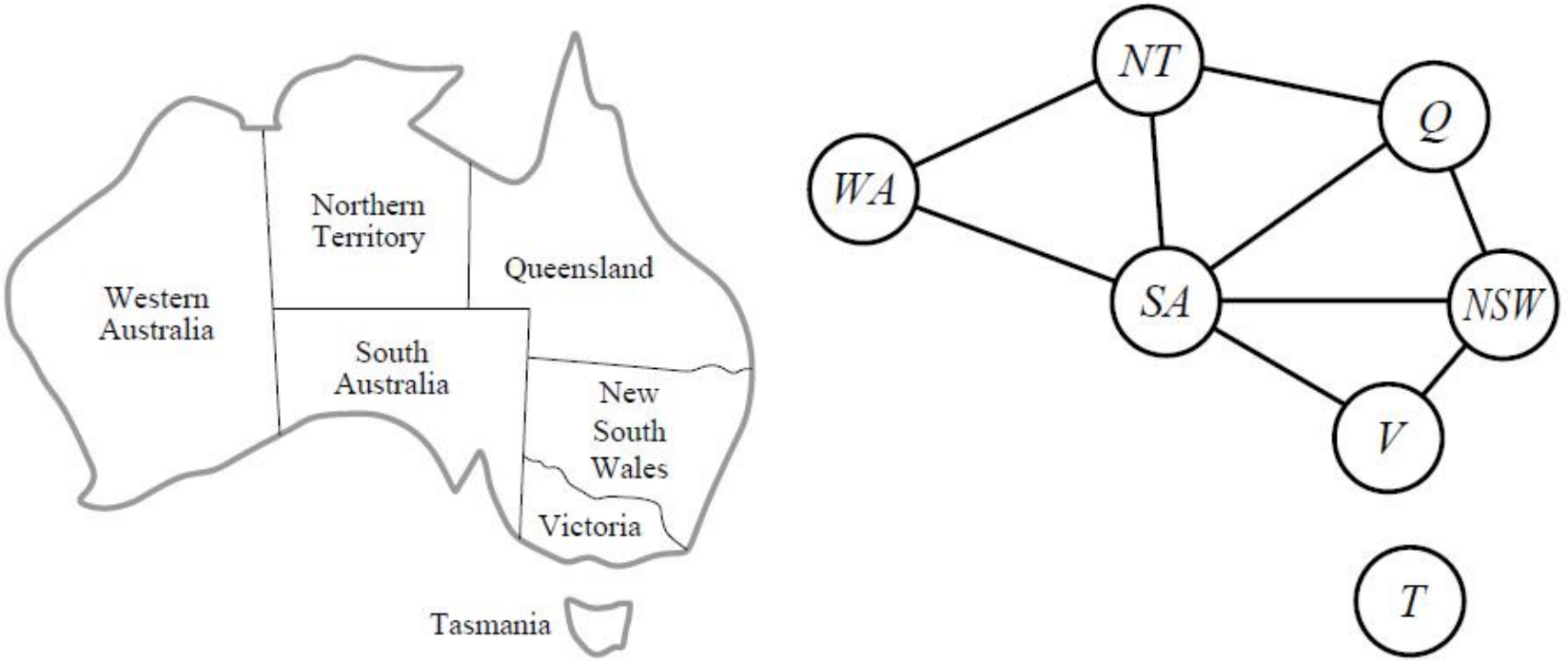
A verificação de violação de restrições pode ser implementada de duas maneiras:

1) modifico SUCESSORS para só nomear valores permitidos, dados os valores já nomeados ou
2) verifico se restrições são satisfeitas antes de expandir um estado

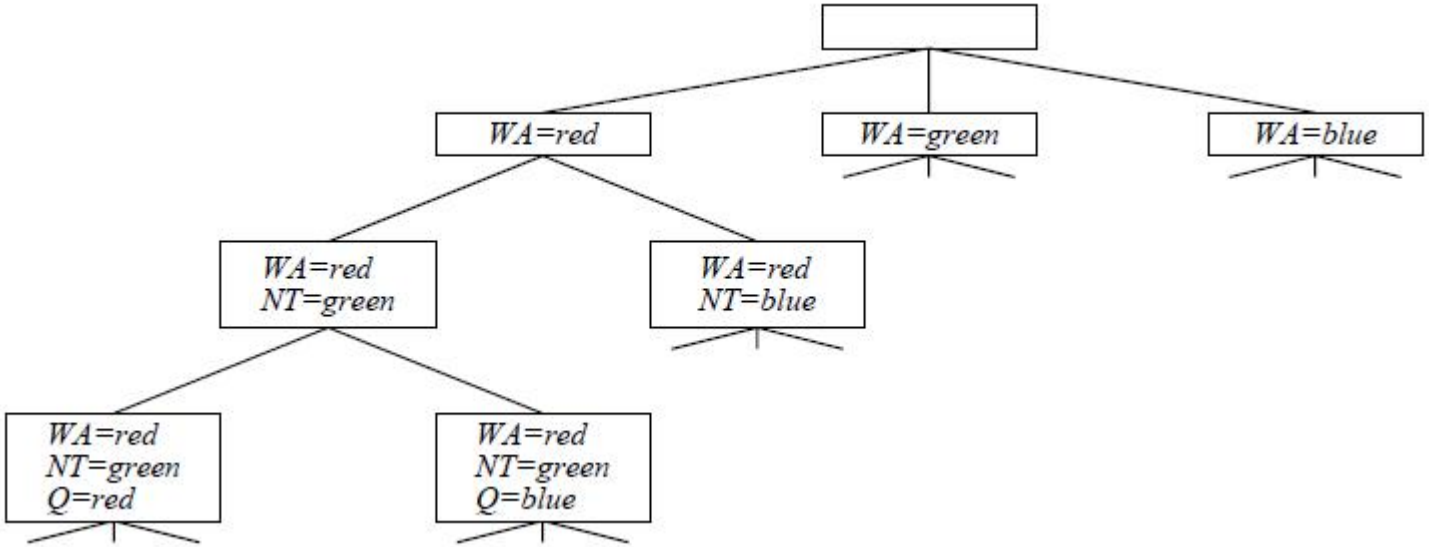
Backtracking é o algoritmo desinformado básico para PSRs

Pode resolver o n-rainhas para $n \approx 15$

Coloração de Mapas : Austrália



Backtracking no Problema de Coloração de Mapas



Algoritmo 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 ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

Idéias para Melhorar o Backtracking

- **Verificação Prévia**
  - Atualizar a lista de possíveis opções para as variáveis, dada a atribuição corrente e restrições

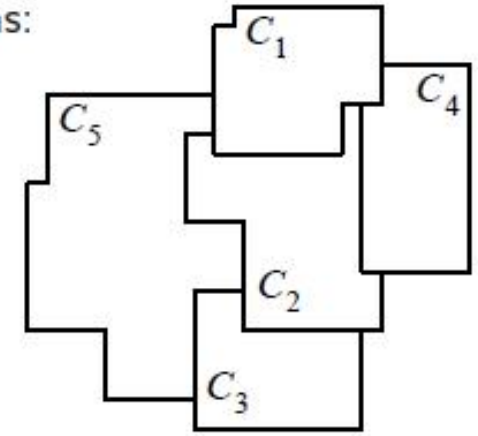
- Qual variável deve ser escolhida primeiro para atribuição?
  - Selecionar varíavel com maior número de restrições: **Heurística de Grau**
  - Selecionar variável com menor número de valores restantes no domínio: **VRM(Valor Restante Mínimo)**

Verificação Forward (Verificação Prévia)

Idéia: Mantenha rastro dos valores legais restantes para variáveis não nomeadas
Terminar a busca quando qualquer variável não tiver nenhum valor legal

Exemplo simplificado de coloramento de mapas:

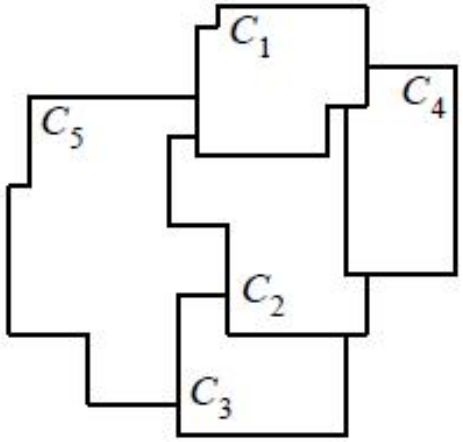
	VERMELHO	AZUL	VERDE
C_1			
C_2			
C_3			
C_4			
C_5			



Pode resolver o n-rainhas até $n \approx 30$

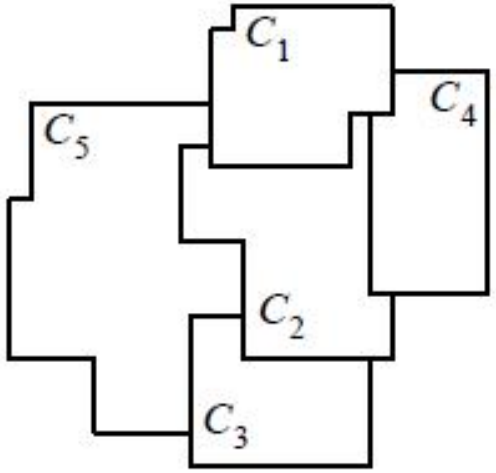
Simulação Verificação Prévia (VP) - 1

	VERMELHO	AZUL	VERDE
C_1	$\surd$		
C_2	$\times$		
C_3			
C_4	$\times$		
C_5	$\times$		



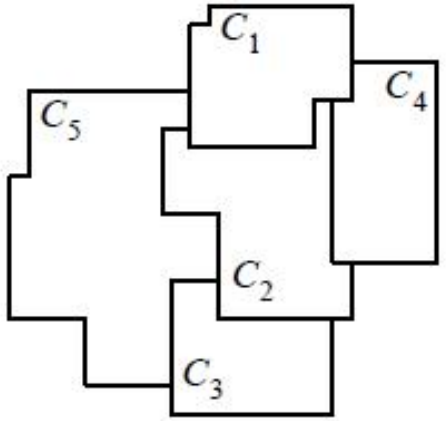
Simulação Verificação Previa - 2

	VERMELHO	AZUL	VERDE
C_1	$\surd$		
C_2	$\times$	$\surd$	
C_3		$\times$	
C_4	$\times$	$\times$	
C_5	$\times$	$\times$	



Simulação Verificação Forward - 3

	VERMELHO	AZUL	VERDE
C_1	√		
C_2	×	√	
C_3		×	√
C_4	×	×	
C_5	×	×	×



Verificação Prévia e Heurísticas

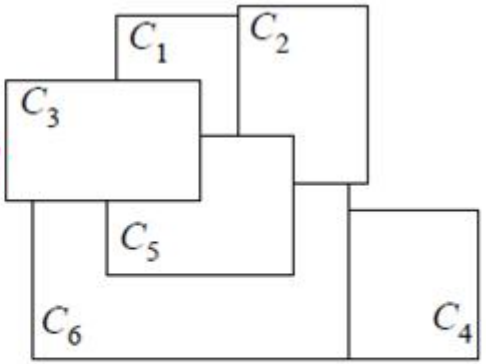
- Heurísticas como heurísticas de grau ou VRM associada a Verificação Prévia podem trazer ganhos de desempenho significativos
- Heurísticas podem ser usadas também para selecionar o valor a atribuir para cada variável, além de selecionar a ordem de atribuição das variáveis

Dados $C_1 = Vermelho$, $C_2 = Verde$, escolha $C_3 = ??$

$C_3 = Verde$: *valor menos restritivo*

Dados $C_1 = Vermelho$, $C_2 Verde$, que fazer??

C_5: *variável mais restritiva*



Pode resolver o n-rainhas para $n \approx 1000$

Algoritmos Iterativos para PSRs

- Idéia: Aceitar estados que não satisfazem a todas as restrições e procurar mudar o valor das variáveis de modo a alcançar estado que satisfaça às restrições

Hill-climbing e *simulated annealing* tipicamente operam com estados "completos" i.e., todas as variáveis nomeadas

Para aplicar a PSRs:

permita estados com restrições não satisfeitas

operadores *renomeiam* valores para variáveis

Seleção de variáveis: aleatoriamente selecione qualquer variável conflitante

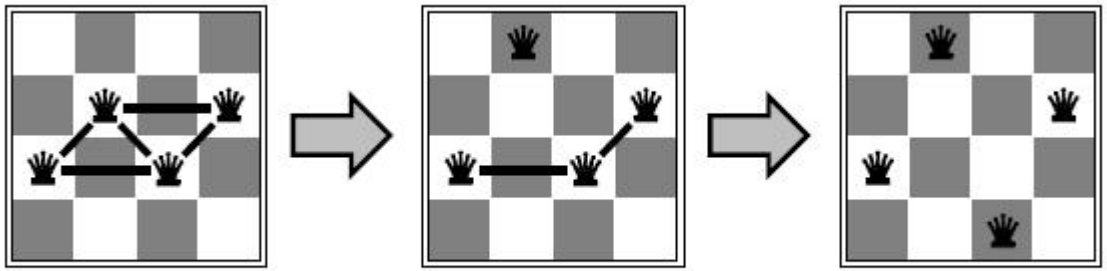
Heurística de mínimos conflitos:

escolha valor que viola menos restrições

i.e., *hillclimb* com $h(n)$ = número total de restrições violadas

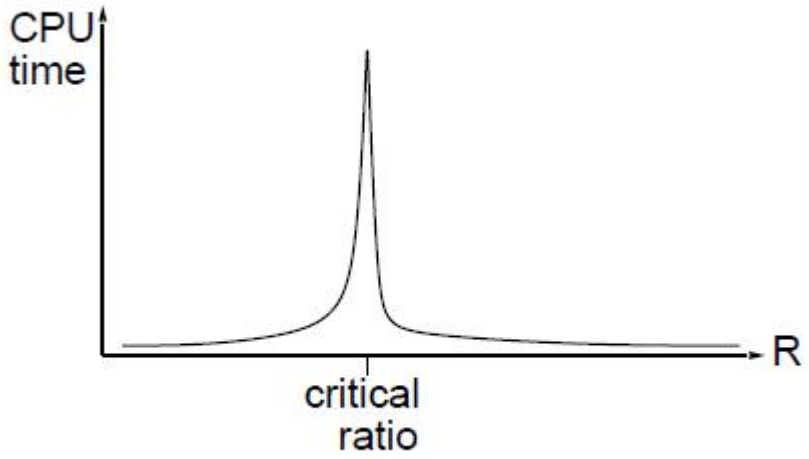
Exemplo: 4 Rainhas

- Estados: 4 rainhas em 4 colunas ($4^4 = 256$ estados)
- Operadores: Mover rainha em coluna
- Teste de objetivo: Nenhum ataque
- Avaliação: h(n) = número de ataques



Heurística de Mínimos Conflitos

- Dado estado inicial aleatório, pode-se resolver o problema das n-rainhas em tempo quase constante para n arbitrário, com alta probabilidade (exemplo, n=10.000.000)
- O mesmo parece ser verdade para qualquer PSR aleatoriamente gerado exceto em uma faixa estreita de relação R = (Número de restrições)/(Número de variáveis)



Algoritmo de Mínimos Conflitos

```
função CONFLITOS-MÍNIMOS(psr, max_etapas) retorna uma solução ou falha
  entradas: psr, um problema de satisfação de restrições
            max_etapas, o número de etapas permitidas antes de desistir

  corrente ← uma atribuição inicial completa para psr
  para i = 1 para max_etapas faça
        se corrente é uma solução para psr então retornar corrente
        var ← uma variável em conflito escolhida ao acaso a partir de VARIÁVEIS[psr]
        valor ← o valor v para var que minimiza CONFLITOS(var, v, corrente, psr)
        definir var = valor em corrente
  retornar falha
```

Resumo

- PSRs constituem um tipo especial de problemas: estados definidos por valores de um conjunto fixo de variáveis. E e teste de objetivo definido por restrições nos valores das variáveis

- Backtracking = busca em profundidade com
  - 1. Ordem fixa de variáveis
  - 2. Apenas sucessores que atendam as restrições

- Verificação Forward previne nomeações que levem a fracasso posterior

- Ordenamento de variáveis e heurísticas de seleção de valores ajudam significativamente

- Conflitos mínimos iterativo é normalmente efetivo na prática