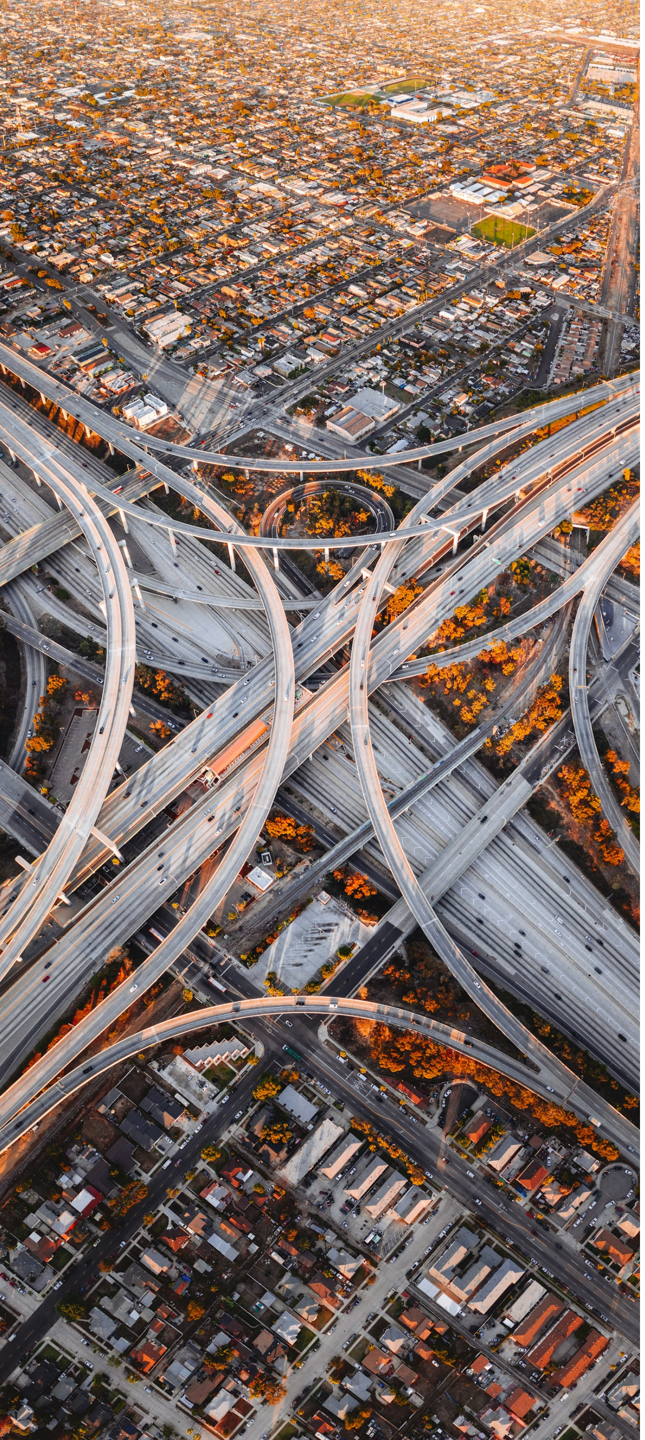


Busca com Satisfação de Restrições

Inteligência Artificial

Pontifícia Universidade Católica de Campinas

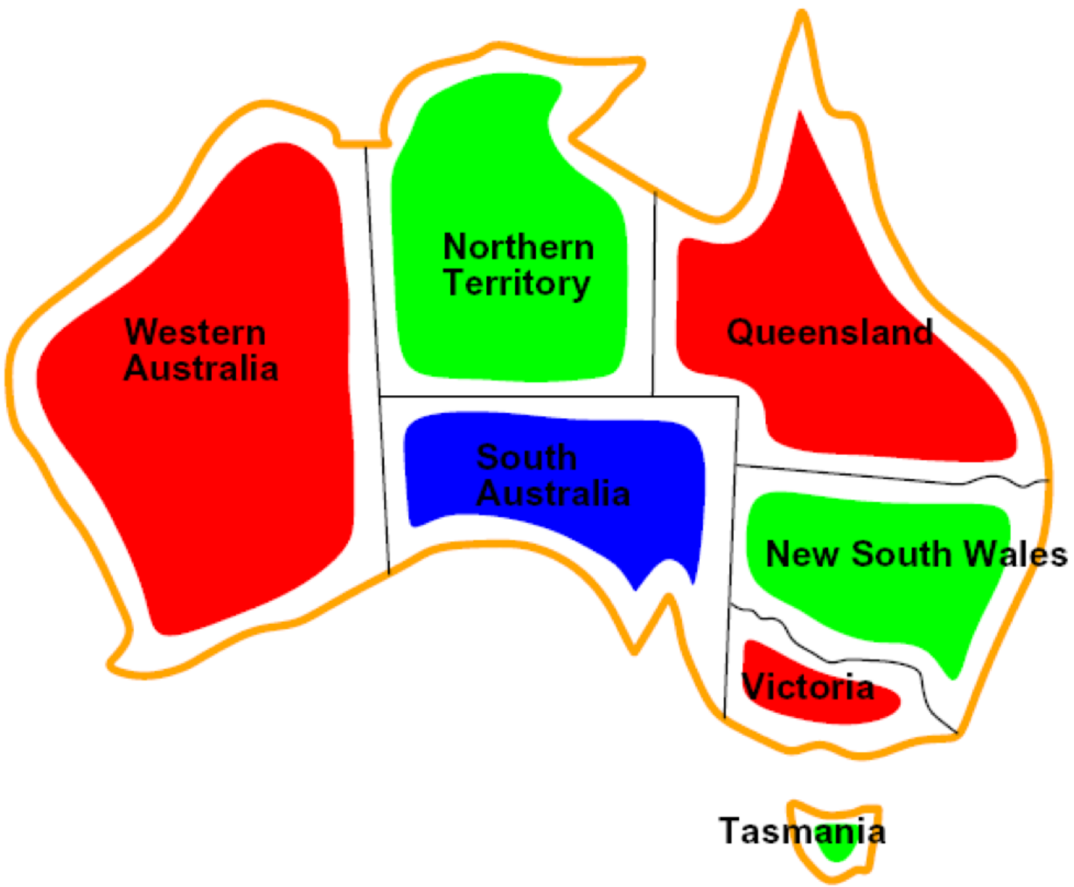
Prof. Dr. Denis M. L. Martins

Objetivos de Aprendizagem

- Formalizar e modelar CSPs: Identificar variáveis, domínios e restrições em problemas.
- Compreender técnicas de busca em CSP (força bruta, backtracking, etc.)
- Aplicar heurísticas em CSP paraotimizar a ordem de escolha das variáveis e valores.
- Avaliar desempenho: Medir tempo, backtracks e profundidade.

Exemplo de Motivação

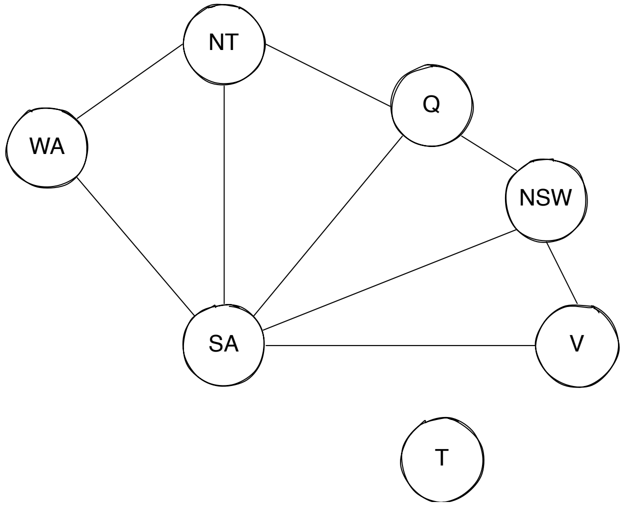
- Como colorir o mapa ao lado?
- Quais os componentes do problema?
  - Variáveis: Regiões
- O que podemos fazer com os componentes?
  - Colorir $\longrightarrow$ Domínio: Cores
- Quais regras precisamos seguir?
  - Restrição: Regiões adjacentes não podem ter a mesma cor.
  - Você consegue pensar em mais alguma restrição?
- Como modelar este problema?
  - Usando grafos (exemplo na lousa)



Fonte da Imagem: Sam Griesemer.

Exemplo de Motivação (cont.)

- Variáveis: $V = \{WA, NT, Q, SA, NSW, V, T\}$
- Domínio: $D = \{R, G, B\}$
- Restrições: $C = \{WA \neq NT, WA \neq SA, \ldots, T \neq R\}$
- Solução: $\{WA = R,\ NT = G,\ Q = R,\ SA = B,\ NSW = G,\ V = R,\ T = G\}$
    - Uma solução possível



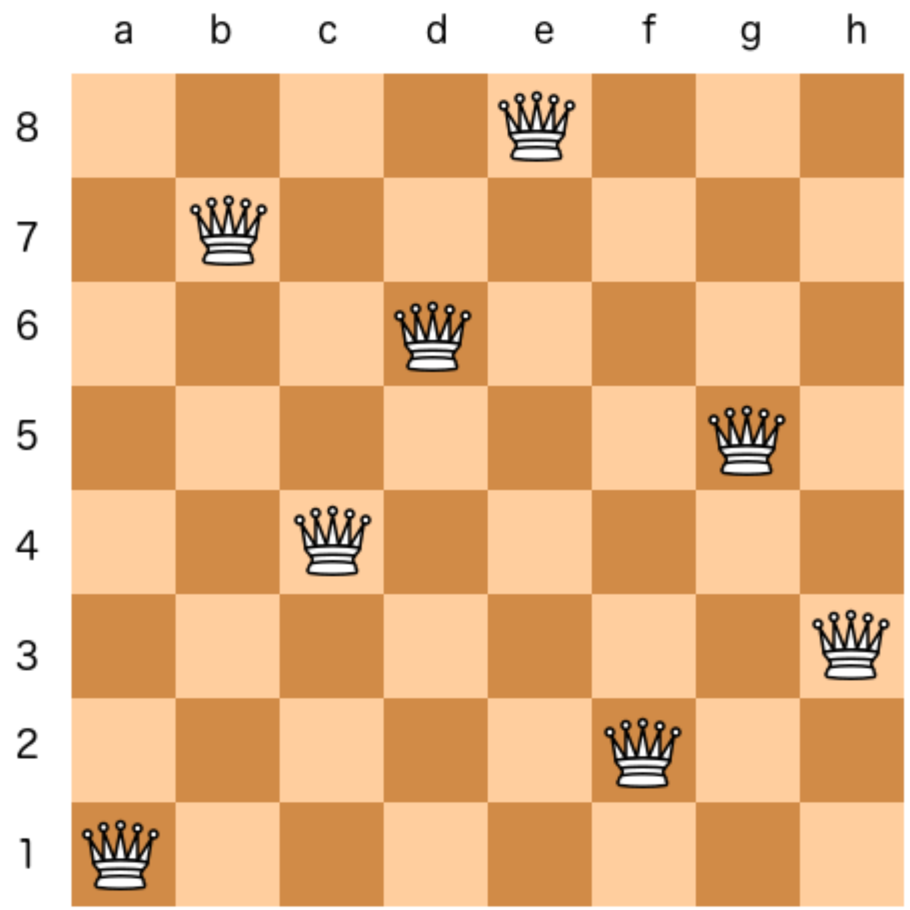
Fonte da Imagem: Resumos LEIC-A.

Busca com Satisfação de Restrições (CSP)

- CSP (Constraint Satisfaction Problem): Problema onde se busca uma atribuição de valores a variáveis que satisfaça todas as restrições.
- **Componentes principais**:
  - **Estado** $V = \{X_1, X_2, \ldots, X_n\}$: conjunto de variáveis
  - **Domínio** D_i : domínio discreto da variável X_i
  - **Restrições** $C = \{C_1, C_2, \ldots, C_m\}$: conjunto de restrições
    - Unárias: sobre uma única variável
    - Binárias: sobre duas variáveis
    - N-árias: sobre várias variáveis
- **Solução**: atribuição $\langle X_1 = v_1, \ldots, X_n = v_n \rangle$ tal que todas as C_k são satisfeitas.

N-Queens: Exemplo clássico de CSP

- **Objetivo**: Queremos colocar exatamente uma rainha em cada linha do tabuleiro, sem que nenhuma delas ataque outra.
    - **Coluna**: duas rainhas não podem ficar na mesma coluna.
    - **Diagonal**: duas rainhas não podem estar na mesma diagonal (tanto a diagonal que vai de canto superior esquerdo a inferior direito quanto a que vai de canto superior direito a inferior esquerdo).
- **Complexidade**: O número de soluções cresce super-exponencialmente (para $n = 8$ há 92 soluções em **4.426.165.368 possibilidades**).
- **NP-completo**: possui $\mathcal{O}(n!)$ possibilidades.



Fonte da Imagem: Paul Brown.

Vamos jogar o N-Queens Puzzle:

https://www.n-queens.com/

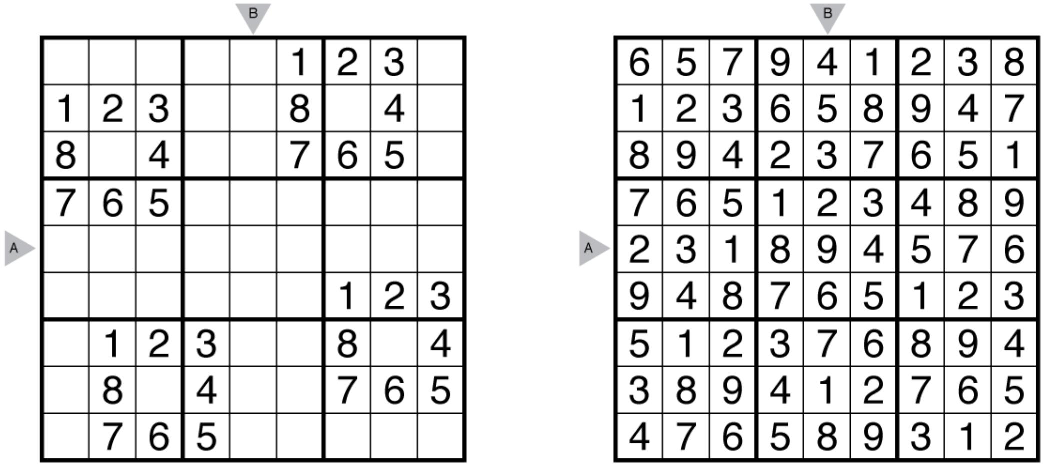
CSP no Cotidiano: Aplicações Práticas

Área	Descrição
Planejamento de Rotas (GPS)	Variáveis = pontos de controle; restrições de tempo e distância.
Agendamento de Cursos	Horários, salas, professores → CSP com domínios discretos.
Design de Circuitos Digitais	Distribuição de componentes na placa → restrição de espaço.
Computação em Nuvem	Alocação de recursos, distribuição de dados
Data Marketplaces	Distribuição de ganhos em mercados de dados

Sudoku: Mais um exemplo de CSP

- Tabuleiro 9×9 com nove blocos 3×3.
- Preencher **todas** as células com números $1 \dots 9$.

Restrições
Cada linha contenha cada número exatamente uma vez.
Cada coluna contenha cada número exatamente uma vez.
Cada bloco contenga cada número exatamente uma vez.



Fonte da Imagem: GM Puzzles.

Sudoku: Exemplo

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Fonte da Imagem: Wikipedia.

Sudoku: Formalização

- **Variáveis**: $X_{r,c}$ para $r = 1 \ldots 9,\ c = 1 \ldots 9$. Total de 81 variáveis.
- **Domínios**: $\mathcal{D}_{r,c} = \{1, \ldots, 9\}$ (ou o valor já dado no enunciado).
- **Restrições Binárias**: $C_{ij} = \{(v_i, v_j) \in D_i \times D_j \mid f_{ij}(v_i, v_j) = 0\}$
  - Linha: $X_{r,c_1} \neq X_{r,c_2}$ para todos $c_1 \neq c_2$.
  - Coluna: $X_{r_1,c} \neq X_{r_2,c}$ para todos $r_1 \neq r_2$.
  - Bloco 3×3: $X_{r_1,c_1} \neq X_{r_2,c_2}$ se ambos pertencem ao mesmo bloco.
- **Restrições N-Árias**: $C_{i_1 \ldots i_k} = \{(v_{i_1}, \ldots, v_{i_k}) \mid g(v_{i_1}, \ldots, v_{i_k}) = 0\}$
  - Equivalentemente, cada conjunto (linha/coluna/bloco) deve conter valores distintos – pode ser expressa como restrição "todos diferentes".
- **Problema de Busca**:
  $\text{Solve}(V, D, C) = \bigcup_{\substack{a_i \in D_i \\ i=1\ldots n}} \mathbf{1}_C(a_1, \ldots, a_n)$

Sukodu: Modelagem

- **Domínio**:

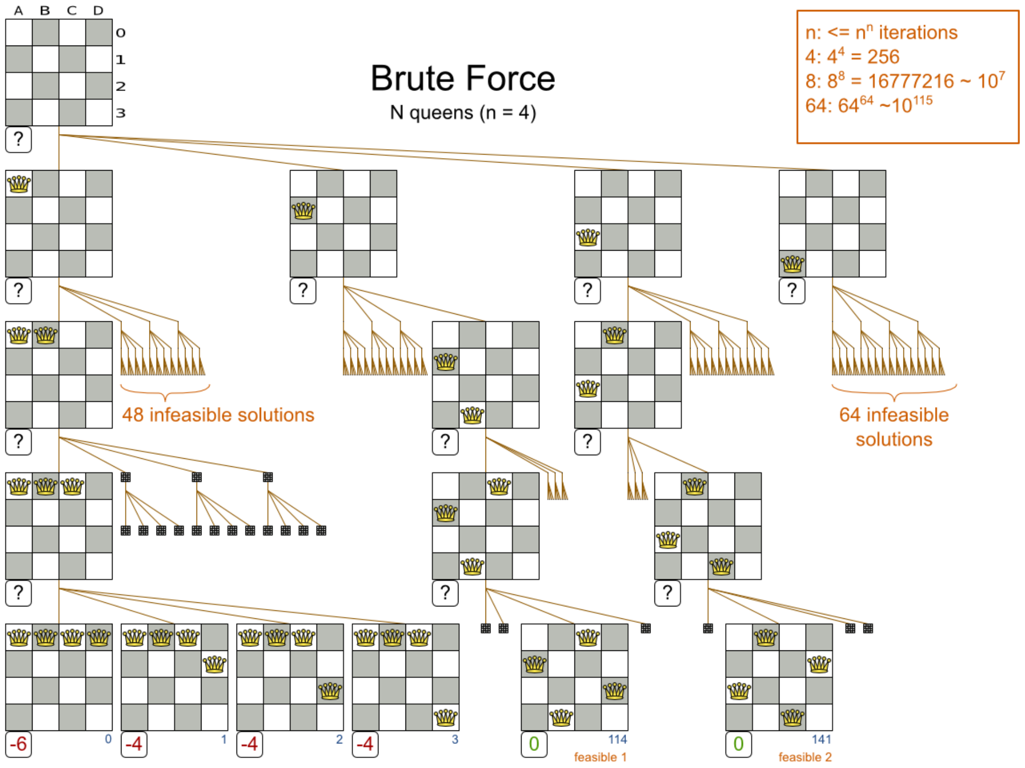
$$D_{r,c} = \begin{cases} \{k\} & \text{se a célula já contém } k \\ \{1, \ldots, 9\} & \text{caso contrário} \end{cases}$$

- **Restrição de Linha**: $C_r^L = \{(v_{c_1}, \ldots, v_{c_9}) \mid v_{c_i} \neq v_{c_j}\ \forall i \neq j\}$
- **Restrição de Coluna**: $C_c^C = \{(v_{r_1}, \ldots, v_{r_9}) \mid v_{r_i} \neq v_{r_j}\ \forall i \neq j\}$
- **Restrição de Bloco**: Para bloco $b = (b_r, b_c)$ com $b_r, b_c \in \{0, 1, 2\}$:
  $C_b^B = \{(v_{r,c}) \mid v_{r,c} \neq v_{r',c'}\ \forall (r,c), (r',c') \in B_b,\ (r,c) \neq (r',c')\}$
- **Solução**:

$$\text{Solution} = \bigcap_{r=1}^{9} C_r^L \cap \bigcap_{c=1}^{9} C_c^C \cap \bigcap_{b=0}^{2} \bigcap_{b'=0}^{2} C_{(b,b')}^B$$

Algoritmo por Força Bruta

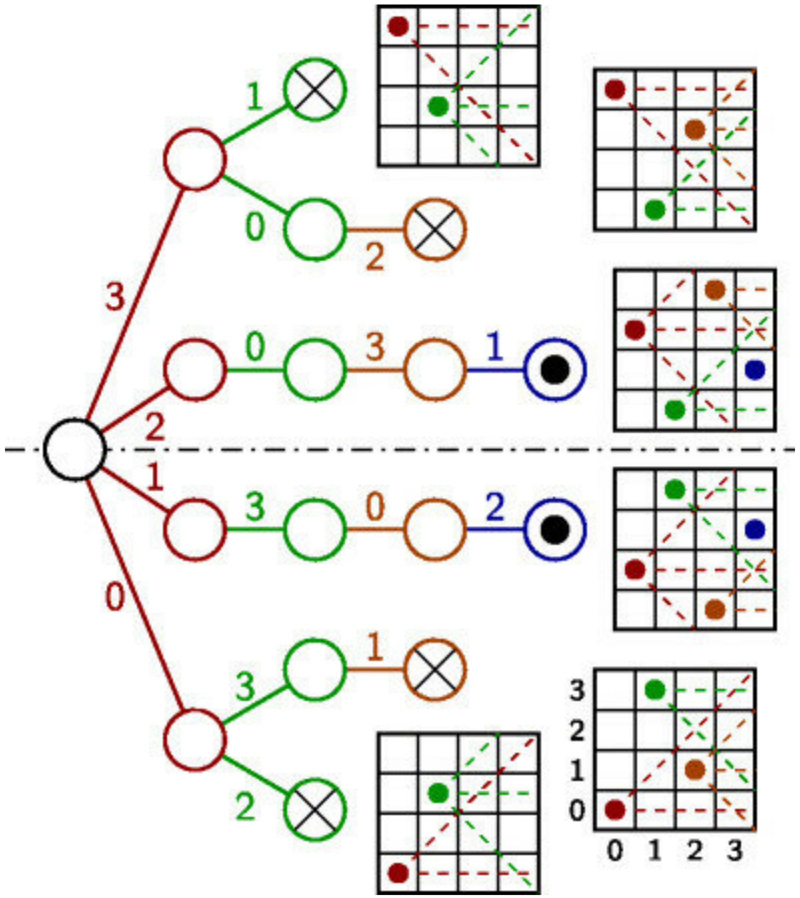
- Explora todas as combinações de valores.
- Útil apenas para problemas pequenos (ex.: Sudoku, 9^2 variáveis).
- **Complexidade**: $\mathcal{O}(\prod_{i=1}^{n} |D_i|)$



Backtracking

- DFS não considera informações sobre restrições: continua percorrendo o grafo de busca mesmo que uma restrição já tenha sido violada.
  - Exploração inútil
  - Pode ser aplicado a CSPs sem modificações, porém a taxa de sucesso depende fortemente da ordem de atribuição.
- Backtracking: Variante especializada do DFS que incorpora **checagem de consistência** no momento da atribuição.
  - Ao detectar violação de restrição, revira imediatamente para o nível anterior (**backtrack**), evitando a exploração de sub-árvores inviáveis.
  - Integra heurísticas de escolha de variável e valor (MRV, LCV) que reduzem drasticamente o espaço de busca em CSPs típicos.
  - Exige manutenção adicional de estruturas de dados (domínios restritos, histórico de escolhas), aumentando a sobrecarga de memória.

Backtracking N-Queens



Fonte da Imagem: ResearchGate.

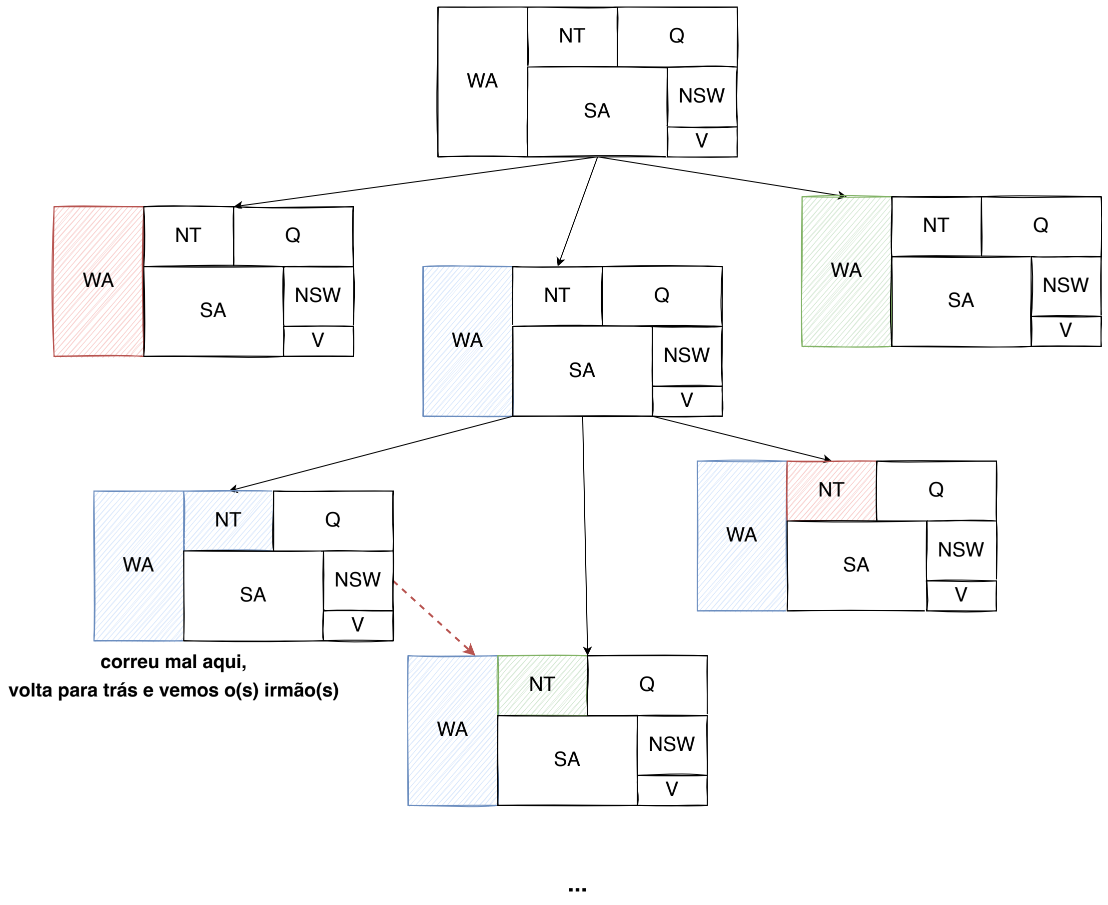
- Coloque cada rainha, uma por uma, em linhas diferentes.
- Ao colocar uma rainha em uma linha, verifique se há conflitos com as rainhas já colocadas.
- Para qualquer coluna, se não houver conflito, marque essa linha e coluna como parte da solução, colocando a rainha.
- Caso não seja encontrada nenhuma célula segura devido a conflitos, volte atrás (ou seja, desfaça a colocação da rainha mais recente).

Backtracking: Sudoku

- Procedimento recursivo: escolhe uma variável, tenta um valor do domínio e verifica restrições locais.
- Se falhar, volta ("backtrack") ao passo anterior.

```
function BACKTRACK_SUDOKU(board):
    if board is complete: return board
    (r,c) = select_unassigned_cell(board)   // MRV + LCV
    for val in order_domain_values(r,c,board): // LCV
        if consistent(r,c,val,board):
            board[r][c] = val
            result = BACKTRACK_SUDOKU(board)
            if result != failure: return result
            board[r][c] = 0          // undo
    return failure
```

Backtracking: Colorir Mapa



Fonte da Imagem: Resumos LEIC-A.

Forward Checking (FC)

- Após atribuir $X_i = v$, elimina de cada vizinho X_j valores incompatíveis com v.
- Se algum domínio ficar vazio → falha imediata.

Complexidade amortizada: $\mathcal{O}(n^2 d)$ (onde $d = \max |D_i|$).

Heurísticas de Seleção

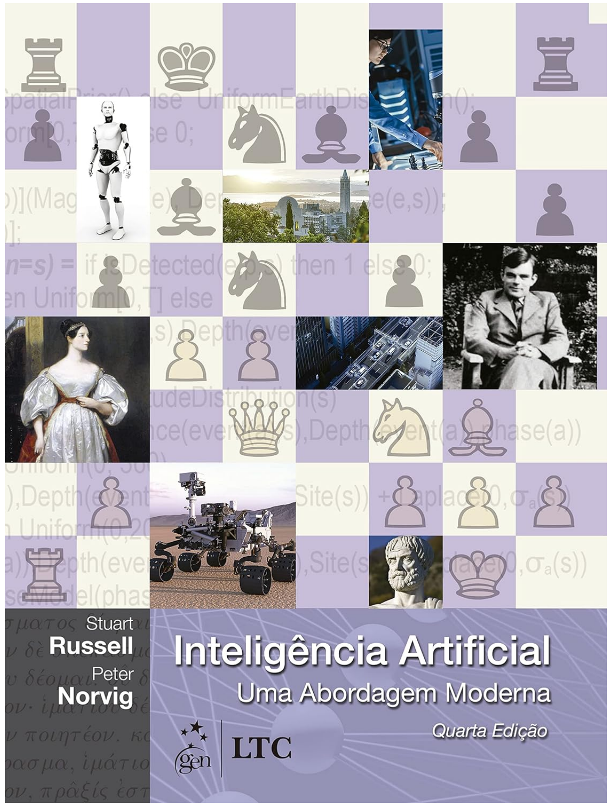
Estratégia	Descrição
MRV (Minimum Remaining Values)	Escolhe variável com menor domínio restante.
Degree	Entre MRV-iguais, escolhe a que tem mais restrições não resolvidas.
LCV (Least Constraining Value)	Ordena valores que restringem menos os vizinhos.

Restrições Leves (Soft Constraints)

- Também é possível definir restrições que podem ser quebradas
- Geralmente associadas a um custo ou preferências
- Ex.: Priorizar o uso das cores na ordem vermelho, verde, azul
- Pode ser resolvido associando custos à cada valor do domínio, podendo ser tratado como problema de otimização
- Também pode ser resolvido ordenando a expansão da busca

Resumo e Próximos Passos

- CSP (Constraint Satisfaction Problem): Variáveis, domínios e restrições definem o problema. Objetivo é encontrar uma atribuição que satisfaça todas as restrições.
- Backtracking (exaustivo + heurísticas).
- Forward Checking – eliminação antecipada de valores impossíveis.
- Heurísticas de escolha: MRV (Minimum Remaining Values) e LCV (Least Constraining Value).
- Aplicações clássicas: Sudoku, Xadrez, otimização logística, etc.
- Próximos passos:
    - Implementar força bruta e backtracking.
    - Algoritmos evolutivos e meta-heurísticas.



Atividade recomendada: Leitura do capítulo 3.

Perguntas e Discussão

- Qual a diferença fundamental entre backtracking puro e forward checking? Em que situações o custo adicional do forward checking vale a pena?
- Como a heurística MRV (Minimum Remaining Values) influencia a profundidade da árvore de busca? Pode haver casos em que escolher a variável com mais valores restantes seja mais vantajoso?
- Como a escolha do modelo de domínio (por exemplo, representar cada linha como uma variável vs. usar variáveis binárias para cada célula) afeta a eficiência das técnicas de CSP? Quais trade-offs surgem na modelagem de problemas com muitos valores distintos por variável?
- Quais são os principais desafios ao adaptar algoritmos de CSP para domínios contínuos ou híbridos (contínuo + discreto)?