Projeto 5: Problemas de Satisfação de Restrições (CSP)

Autor: Marcos Antonio Teles de Castilhos

Disciplina: FGA0221 - Inteligência Artificial

Professor: Fabiano Araujo Soares, Dr.

1. Introdução

Este projeto explora a resolução de **Problemas de Satisfação de Restrições (CSPs)**, uma técnica fundamental em Inteligência Artificial para encontrar soluções que atendam a um conjunto de regras ou limitações. Diferente das buscas por caminho ou otimização local, o foco do CSP é encontrar uma **atribuição completa e consistente** de valores para um conjunto de variáveis, respeitando as restrições definidas.

O projeto consiste em duas implementações que demonstram diferentes facetas da modelagem e resolução de CSPs:

1. **projeto_5_csp_mapa_visual.py:** Resolve o problema clássico da **Coloração de Mapas** (Austrália) com uma visualização gráfica do algoritmo de Backtracking, MRV e Forward Checking.
2. **projeto_5_csp_timetabling_terminal.py:** Resolve um problema mais complexo de **Alocação de Aulas (Timetabling)** via terminal, introduzindo restrições de maior ordem, a distinção entre restrições duras e suaves (preferências), e a busca pela *melhor* solução (otimização). Inclui um laboratório de testes para analisar diferentes cenários.

2. Conceitos Fundamentais de CSP

Conforme a definição formal, um CSP é composto por:

- **Variáveis (X):** Entidades que precisam receber um valor (ex: estados de um mapa, disciplinas a serem alocadas).

- **Domínios (D):** Conjunto de valores possíveis para cada variável (ex: cores disponíveis, possíveis combinações de professor/sala/horário).
- **Restrições (C):** Regras que especificam quais combinações de valores são permitidas (ex: estados vizinhos não podem ter a mesma cor, um professor não pode estar em dois lugares ao mesmo tempo).

A solução é uma atribuição de um valor do seu domínio a cada variável, de forma que todas as restrições sejam satisfeitas simultaneamente.

3. Algoritmo de Resolução: Backtracking Otimizado

Ambos os scripts utilizam o **Backtracking Search** como algoritmo base. Sua estratégia é atribuir valores às variáveis sequencialmente e retroceder (backtrack) quando uma atribuição viola uma restrição ou leva a um beco sem saída.

Para aumentar a eficiência, foram implementadas as seguintes técnicas [4]:

- **Heurística MRV (Minimum Remaining Values):** Ao escolher a próxima variável a ser atribuída, seleciona-se aquela com o menor número de valores ainda disponíveis em seu domínio. Isso tende a detectar falhas mais cedo. (Implementado em **selecionar_variavel_mrv** no script do mapa).
- **Forward Checking:** Após atribuir um valor a uma variável, remove-se esse valor dos domínios das variáveis vizinhas (ou relacionadas pela restrição). Se o domínio de algum vizinho ficar vazio, sabe-se que a atribuição atual levará a uma falha, permitindo o backtrack imediato. (Implementado em **forward_check** no script do mapa e implicitamente na lógica de consistência do timetabling).

4. Implementação 1: **csp_mapa_visual.py**

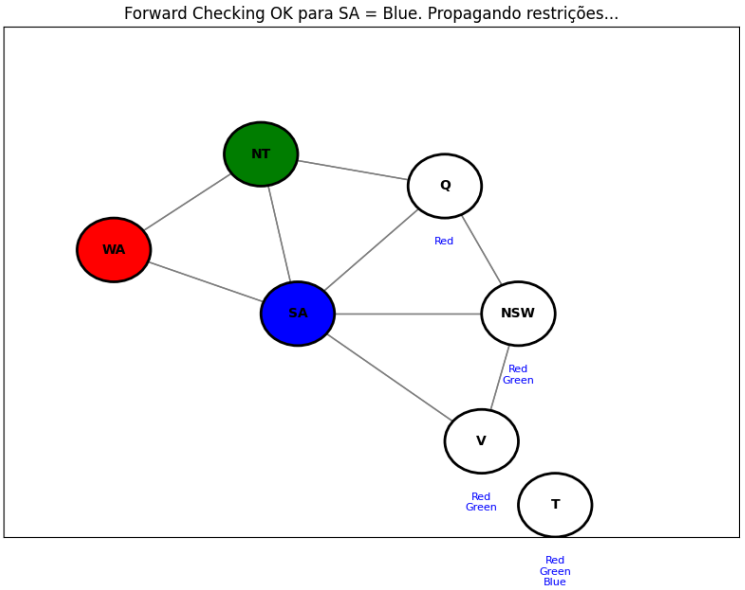
- **Problema:** Colorir o mapa da Austrália com 3 cores (Red, Green, Blue) de forma que regiões adjacentes tenham cores diferentes.
- **Modelagem:**
    - Variáveis: WA, NT, SA, Q, NSW, V, T.
    - Domínio: {'Red', 'Green', 'Blue'} para cada variável.
    - Restrições: Binárias (!=) entre estados vizinhos (ex: WA != NT).

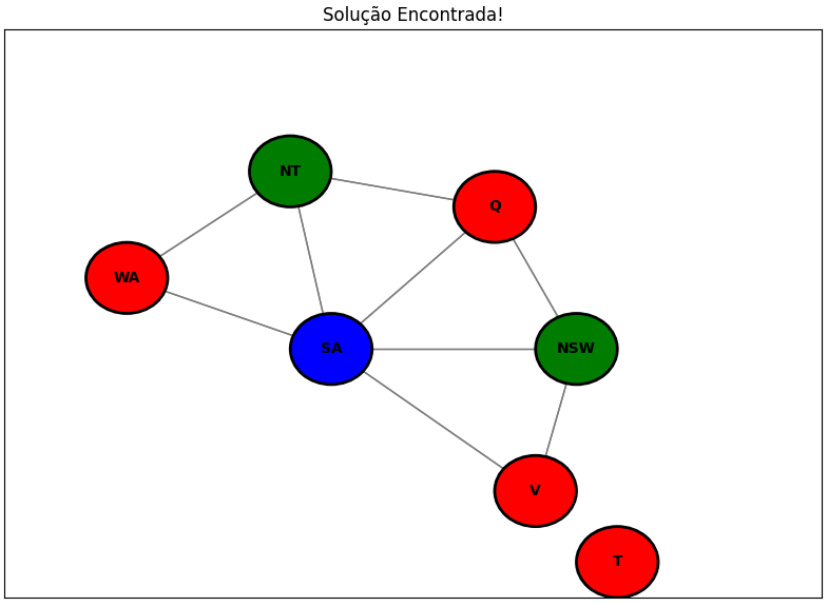
- **Visualização (matplotlib):**
    - Representa o problema como um **grafo de restrições** (nós = estados, arestas = vizinhança).
    - Anima o processo de Backtracking:
        - Nós brancos: Não atribuídos (mostra domínio restante abaixo).
        - Nós coloridos: Atribuídos com a cor correspondente.
        - **Forward Checking Visível:** Ao colorir um nó, a cor desaparece dos domínios dos vizinhos.
        - **Backtracking Visível:** Ao retroceder, o nó volta a ficar branco e os domínios dos vizinhos são restaurados.
- **Como Usar:**

    pip install matplotlib

    python csp_mapa_visual.py
- Observe a animação passo a passo. A velocidade pode ser ajustada alterando o valor em **plt.pause().**

4.1 Imagens






```
--- Problema de Coloração de Mapas (Austrália) ---

--- Solução Final ---
- NSW: Green
- NT: Green
- Q: Red
- SA: Blue
- T: Red
- V: Red
- WA: Red

Feche a janela gráfica para terminar.
```

5. Implementação 2: **csp_timetabling.py**

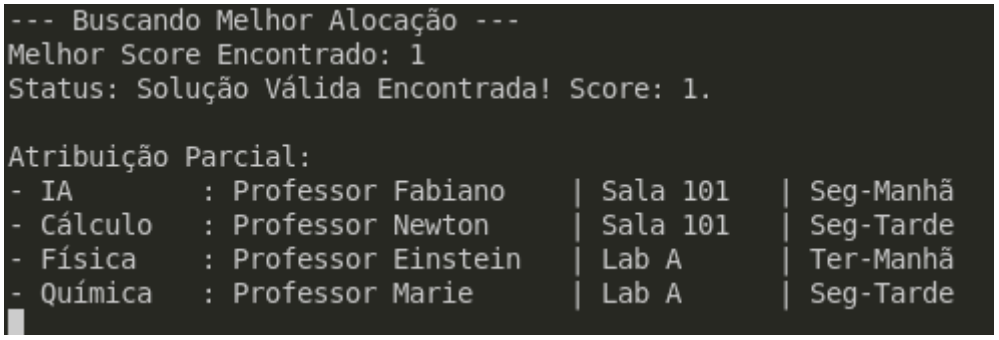
- **Problema:** Alocar 4 disciplinas (IA, Cálculo, Física, Química) a combinações de (Professor, Sala, Horário), respeitando restrições duras e minimizando a violação de preferências (restrições suaves).
- **Modelagem:**
  - Variáveis: IA, Cálculo, Física, Química.
  - Domínio: Produto cartesiano de {Professores} x {Salas} x {Horários}.
  - **Restrições Duras:**
    - Recurso Único (**recurso_unico**): Professor/Sala/Horário não podem ser usados simultaneamente por aulas diferentes (implementado em **is_consistent**). Adiciona a restrição **All-Different** para professores.
    - Requisito de Sala (**requisito_sala**): Certas disciplinas *devem* usar salas específicas (ex: Física no Lab A).

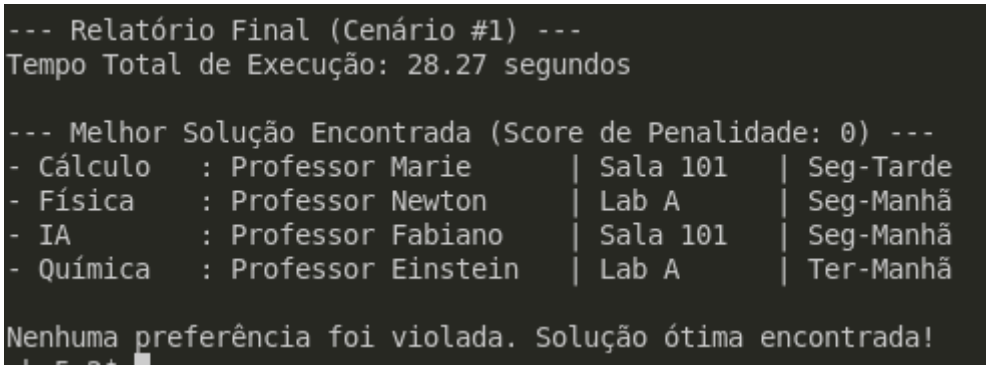
        - Capacitação (**capacitacao_prof**): IA *deve* ser ministrada por Fabiano.
        - (Aplicadas em **aplicar_restricoes_iniciais** e **is_consistent**).
    - **Restrições Suaves (Preferências):**
        - Einstein prefere manhãs.
        - Marie prefere Sala 101.
        - (Cenário 3 adiciona: Física prefere Sala 101 - conflitante).
        - (Avaliadas por **calcular_penalidade**).
- **Algoritmo:** Backtracking modificado para **otimização**. Ele não para na primeira solução válida, mas continua buscando até explorar todo o espaço ou encontrar uma solução com **score de penalidade 0** (ótima), que interrompe a busca (**backtracking_otimizado_terminal**).
- **Laboratório de Testes:** A variável **CENARIO_TESTE** permite testar:
    - Cenário 1: Problema base (espera solução ótima).
    - Cenário 2: Problema insolúvel (menos professores que disciplinas).
    - Cenário 3: Problema com preferências conflitantes (espera solução com penalidade > 0).
- **Visualização (Terminal):**
    - Imprime o estado da atribuição parcial a cada passo.
    - Mostra mensagens de "Tentando...", "Backtrack!", "Solução Válida Encontrada!", "NOVA MELHOR SOLUÇÃO!".
    - Exibe o melhor score encontrado até o momento.
- **Como Usar:**

  python csp_timetabling.py

    - Altere **CENARIO_TESTE** no código para explorar diferentes configurações.
    - Observe a saída no terminal e o relatório final (tempo, melhor solução, score, violações).

5.1 Imagens





6. Conclusão

Este projeto demonstra a flexibilidade e o poder da abordagem CSP. O primeiro script ilustra a resolução de um problema clássico com visualização do Backtracking e suas otimizações (MRV, Forward Checking). O segundo script avança para um problema mais complexo, introduzindo a importante distinção entre restrições duras e suaves e adaptando o backtracking para encontrar a *melhor* solução possível, além de permitir a experimentação com diferentes cenários para analisar a robustez do modelo.