**Portifólio 3**

# Inteligência Artificial - Problemas de Satisfação de Condições (CSPs)

---

**Autor:** Wildemberg Sales da Silva Junior

**Matrícula:** 202017503

**Data:** 30/12/2024

**Instituição/Universidade:** Universidade de Brasília(UnB)

**Disciplina:** Inteligência Artificial - FGA0221

---

## Resumo

Este artigo aborda a resolução de problemas por meio da satisfação de condições, que é um modelo diferente da busca tradicional, onde os agentes deixam de se importar somente com o estado objetivo do resultado, e agora observam os conjuntos de variáveis que possuem valores específicos. Durante a exploração deste problemas de satisfação de condições, também foram aprofundados assuntos realcionados ao entendimento da estruturas de problemas, analisamos os tipos de condições que um problema pode possuir e os modelos de consistência que garante que haja uma solução para o problema. Também foram explorados alguns algoritmos que podem ser utilizados para encontrar a solução de problemas deste porte, apresentando exemplos concretos em que eles podem ser utilizados.

## Visão Geral

O artigo tratou de temas relevantes e interessantes que trouxeram uma visão diferente de como agentes podem solucionar problemas mais complexos além da busca tradicional. Temas como tipo de condições aumentou ainda mais o entendimento sobre as estruturas que um problema pode ter, e a introdução sobre consistência mostrou como é possível garantir que um problema possua solução baseado em suas condições.

## Introdução

Neste artigo abordaremos uma versão diferentes de resolução de problemas por agentes inteligentes, que será a resolução de problemas por meio da satisfação de condições (CSPs).

Sabemos que existe a resolução por meio de buscas que os agentes realizam, onde seu objetivo é avaliar os estados em que "trafegam" para encontrar o estado objetivo, esses agentes utilizam avaliações heurísiticas específicas de domínios para descobrirem se estão ou não no estado objetivo. Durante essas buscas, os agentes inteligentes consideram cada estado como atômico, não se importando com as características do estado. Neste novo modelo que iremos explorar, essa "visão" do agente muda em relação aos estados.

No modelo em que os agentes exploram a resolução de problemas por satisfação de condição, eles deixam de considerar cada estado atômico, e começam a avaliar as variáveis de cada estado, com isso, eles começam a entender que um problema foi resolvido, quando cada variável tiver um valor que satisfaça todas as condições impostas do problema.

## Representação Atômica vs. Fatorada

Como dito anteriormente, os agentes inteligentes quando utilizam a resolução de problemas por satisfação de condição, ele deixam de considerar a representação dos estados de forma atômica, e começam a avaliar uma representação fatorada dos estados, onde é considerado as variáveis do estado, e o problema só é resolvido quando todas as variáveis possuem os valores que atendam as condições (RUSSEL; NORVIG, 2010).

## Definindo Problemas de Satisfação de Condições

Agora que entendemos como os agentes agem quando se trata de problemas de satisfação de condições, agora devemos entender o que são esses problemas e suas principais características.

De acordo com Russel e Norvig, 2010, um problema de satisfação de condição consiste em três componentes, X, D, e C, onde cada um representa:

- X: conjunto de variáveis, {X1, ..., Xn};
- D: conjunto de domínio de cada variável (valores que pode obter), {D1, ..., Dn};
- C: conjunto de restrições que especificam as possíveis combinações de valores possíveis. Cada restrição é uma par de escopo e rel, onde escopo é a tupla de variáveis que participam da combinação, e rel é a relação que define os valores que as variáveis podem assumir, onde cada rel pode ser representado por uma lista de tuplas válidas ou abstratas por meio de condições;

Portanto, para resolver um problema de satisfação de condição é necessários definirmos um conjunto de estados e a noção de solução. Onde cada estado é definido por uma **atribuição** de valores a alguma ou todas as variáveis (RUSSEL; NORVIG, 2010). As atribuições podem ter diferentes formas:

- Atribuição consistente: é uma atribuição que não viola qualquer condição;
- Atribuição completa: quando todas as variáveis recebem uma atribuição;
- Atribuição parcial: quando somente algumas variáveis recebem valores;

Baseado nas atribuições chegamos a uma **solução**, que pode ser descrita como uma atribuição consistente e completa.

Para ficar mais claro a ideia de problemas que podem ser resolvidos baseado na satisfação de condições, podemos ver um exemplo baseado no ambiente acadêmico:

**Exemplo:** Dentro de um ambiente acadêmico como a Universidade de Brasília (UnB), os alunos necessitam montar sua grade de horário e matérias todo o semestre, com isso podemos colocar essa problematica para um agente resolver, modelando o problema da seguinte forma:

- **Problema**: Criar uma grade horária de matérias para o semestre:
- **Condições (C)**: Não pode ter duas matérias no mesmo horário e o aluno tem que ter permissão para se matricular nessa matéria;
- **Variáveis (X)**: Cada Matéria;
- **Domínios (D)**: Horários disponíveis de cada matéria, e permissões do aluno;

Desta forma o agente poderia agir sobre essa estrutura para se chegar a uma solução.

## Tipos de Condições

Além de entendermos as variáveis que compõem um problema, também é possível explorar os tipos de condições que fazem parte do problema. A seguir temos os tipos de condições e aplicações baseadas no exemplo anterior, de cada um tipo para aprofundamos o conhecimento:

- **Unitária**: Quando restringe o valor de somente uma variável. No exemplo anterior o agente teria que escolher turmas para horários diferentes, se aplicarmos uma regra onde ele não pode colocar no horário de 14h, seria uma restrição unitária;
- **Binária**: As condições binárias relacionam duas variáveis. No exemplo anterior o agente não poderia colocar duas matérias no mesmo horário, ou seja, Horário_Tuma_A != Horário_Turma_B;
- **Ternária**: A condição ternária de dá quando relacionamos três variáveis, ou seja, a Turma_C deve estar entra a Turma_A e a Turma_B;
- **Global**: Essa condição se dá quando se envolve um número arbitrário de variáveis, não necessariamente todas, portanto, aplicando no exemplo

anterior, podemos dizer que as matérias devem ter professores diferentes para cada;

Os algoritmos de solução dos CSPs, pode realizar duas ações, eles podem buscar onde ele vai escolher uma nova atribuição para a variável de várias possibilidades, ou ele pode fazer uma inferência chamada **propagação de condições**, onde ele reduz o número de valores possíveis para a variável atual, e como consequência, ele reduz sucessivamente os possíveis valores das variáveis seguintes, garantindo uma **consistência local**.

## Consistência

Como dito anteriormente é possível gerar uma consistência dentro dos problemas que abordamos, com isso, iremos explorar os tipos de consistências possíveis que pode ser gerado.

- **Nó**: Para a consistência de nó (nó-consistente), é necessário que todos os valores de seu domínio respeitem a condição unária da variável, com isso, é eliminado todos os domínios que não atendem a condição;
- **Arco**: Para a consistência de arco (arco-consistente), é necessários que os valores do domínio satisfaçam as restrições binárias da variável;
- **Trajeto**: A consistência de trajeto é aplicada quando fixa as restrições binárias para garantir a consistência em um trajeto de três variáveis;
- **K**: A consistência em K é definida quando em um conjunto de K-1 variáveis, e para qualquer atribuição a essas variáveis, um valor consistente puder ser atribuido a qualquer uma das variáveis do conjunto de K;
- **Globais**: Na consistência global, ela deve ser K-consistênte para quase todas ou todas as K variáveis que fazem parte do conjunto, onde por mais que seja global, não implica que necessariamente serão todas.

## Algoritmos

Dentro do campo de implementação dos agentes que realizam a solução de problemas por satisfação de busca, podemos começar a citar alguns algoritmos e suas aplicações. A seguir iremos explorar alguns algoritmos comumente utilizados.

### AC-3

O algoritmo AC-3 examina os arcos existentes através de pares de variáveis (X e Y), com isso ele vai removendo os domínios de X que não seguem as restrições impostas pro par. O algoritmo vai mantendo em memória as coleções de arcos que ainda não foram verificados, então quando um domínio de uma variável é retirado, todos os arcos que tem relação de restrições com o domínio são adicionados a coleção, menos aquele que está na restrição atual (WIKIPEDIA, 2024).

A seguir, no **código 1**, podemos ver a implementação do pseudo-código do AC-3:

```
AC-3(CSP):
  1. Inicialize a fila Q com todos os arcos (Xi, Xj) das restrições do CSP.
  2. Enquanto Q não estiver vazia:
      3. Remova um arco (Xi, Xj) de Q.
      4. Se REMOVER_VALORES_INCONSISTENTES(Xi, Xj):
          5. Para cada variável Xk adjacente a Xi (exceto Xj):
              Adicione (Xk, Xi) à fila Q.
  6. Retorne True (todos os arcos são consistentes) ou False (algum domínio ficou vazio).

REMOVER_VALORES_INCONSISTENTES(Xi, Xj):
  1. Inicialize alterado = False.
  2. Para cada valor vi em Di:
      3. Se não existe valor vj em Dj que satisfaça a restrição entre Xi e Xj:
          4. Remova vi de Di.
          5. alterado = True.
  6. Retorne alterado.
```

Código 1: Implementação em pseudo-código do algoritmo AC-3, Fonte: OPENAI. Assistente Virtual ChatGPT, 2024

Podemos analisar um exemplo para esse algoritmo:

**Exemplo**: Um professor irá aplicar uma prova para sua turma, ele conhece muito bem seus alunos e quer alocá-los em suas cadeiras, ele entende que alguns alunos não podem ficar juntos pois pode haver cola durante a prova, cada cadeira só pode ter um aluno, e alguns alunos tem a necessidade de sentar em lugares específicos. Se aplicassemos o algoritmo para ajudar com esse problema, iriamos obter um um conjunto de domínios que atendem as condições.

### Backtracking

O Backtracking é um modelo de busca de profundidade que escolhe valores para todas as váriáveis do problema, uma variável de cada vez, e que realiza o retrocesso quando não há valores válidos restantes para serem atribuidos a uma variável (RUSSEL; NORVIG, 2010).

A seguir no *código 2*, podemos ver a implementação do algoritmo em pseudo-código:

```
BACKTRACKING(CSP, atribuição):
  1. Se todas as variáveis estão atribuídas:
      Retorne atribuição.
  2. Escolha uma variável não atribuída X.
  3. Para cada valor v em Domínio(X):
      4. Se v é consistente com atribuição:
          5. Atribua X = v.
          6. Result = BACKTRACKING(CSP, atribuição).
          7. Se Result  falha:
```

```
            Retorne Result.
        8. Remova a atribuição X = v.
  9. Retorne falha.
```

Código 2: Implementação em pseudo-código do algoritmo Backtracking, Fonte: OPENAI. Assistente Virtual ChatGPT, 2024

Podemos ver um exemplo para esclarecer melhor o a utilização do backtracking:

**Exemplo**: Baseado no problema da mochila, pertencente ao "Os 21 problemas NP-completos", que foram introduzidos por Richard Karp em 1972, onde a ideia é ter uma mochila com um limite de peso e quantidade de items, e temos uma quantidade Y de items. A utilização do algoritmo backtracking iria nos fornecer uma série de combinações possíveis para alocarmos nossos itens na mochila. E se colocassemos uma restrição como a necessidade de levar obrigatoriamente uma lanterna e um tablet, o algoritmo iria analisar as combinações e iria evitar todas que não seguem essas condições.

### Busca local

Esse algoritmo de busca local é muito eficaz para resolver CSPs, ele utiliza a formulação de estados completos, onde o estado inicial atribui um valor a cada variável, e no decorrer do processo de busca ele altera o valor de uma variável por vez até chegar na solução.

A seguir no código 3, podemos ver a implementação em pseudo-código do algoritmo:

```
BUSCA_LOCAL(CSP):
  1. Inicialize uma atribuição completa (possivelmente aleatória).
  2. Enquanto não atingir o limite de iterações:
      3. Se a atribuição for uma solução (todas as restrições satisfeitas):
          Retorne atribuição.
      4. Selecione uma variável e altere seu valor para melhorar
      o número de restrições satisfeitas.
  5. Retorne falha.
```

Código 3: Implementação em pseudo-código do algoritmo de busca local, Fonte: OPENAI. Assistente Virtual ChatGPT, 2024

Podemos ver um exemplo onde a utilização deste algoritmo seria idela para encontrar a solução:

**Exemplo**: Se pensarmos em uma problemática dentro de uma empresa, onde devemos distribuir da melhor forma possível tempo, dinheiro e materias, para benefícios totais, respeitando as condições de limites de gastos e alocação de tempos. O algoritmo de busca iria conseguir encontrar a melhor solução para este problema.

**Backtracking Recursivo**

Baseado no algoritmo backtrancking, mas com uma leve alteração, durante o seu processo ele acumula um conjunto de conflitos, e ao mesmo tempo realiza a verifica a atribuição de valores válidos, e se não for encontrado nenhum valor válido, ele retorna o elemento mais recente do conjunto de conflito e o indicador de falha (RUSSEL; NORVIG, 2010).

A seguir no código 4, podemos ver a implementação do algoritmo no pseudo-código:

```
função retrocessoRecursivo(estado):
    se éSolução(estado):  // Verifica se o estado atual é uma solução
        retornar estado  // Se for uma solução, retorna o estado

    para cada escolha em opçõesPossíveis(estado):  // Para cada escolha possível
        novoEstado = aplicarEscolha(estado, escolha)  // Aplica a escolha e gera um novo est

        se éViável(novoEstado):  // Verifica se o novo estado é viável
            resultado = retrocessoRecursivo(novoEstado)  // Chama recursivamente para o pró

            se resultado != falha:  // Se encontrou uma solução, retorna o resultado
                retornar resultado

    retornar falha  // Se nenhuma escolha levou a uma solução, retorna falha
```

Código 4: Implementação em pseudo-código do algoritmo de Backtracking Recursivo, Fonte: OPENAI. Assistente Virtual ChatGPT, 2024

## Estrutura de Problemas

Ao se analisar os problemas de satisfação de condições, podemos ter diferentes visões para suas soluções, ao olhar as condições de um problema, podemos simular eles em um grafo de condições que pode ser utilizado para acelerar a busca por soluções.

Baseado nos grafos de condições, é possível aplicarmos algums métodos que facilitam a compreensão e entendimento para a resolução do problema, a seguir temos os seguintes métodos:

- **Decomposição em subproblemas independentes:** Se o grafo puder ser dividido em componentes desconectados, então é possível tratar o problema de forma separada;
- **Grafos em árvore:** Se for possível tranformar o grafo em uma árvore, fica mais fácil de se aplicar uma abordagem eficiente para encontrar a solução;
- **Redução de Grafos de Condições a Árvores:** No caso onde o grafo não é uma árvore, pode tentar eliminar variáveis que façam o grafo restante se tornar uma árvore;

- **Decomposição em Árvore de Grafos de Condição:** O grafo pode ser decomposto em problemas menores, e ao final, as soluções são combinadas.

Ao se utilizar destes métodos, se torna possível obter soluções de forma mais fácil dentro de um problema.

## Conclusão

Neste artigo, foi possível analisarmos um novo modelo de resolução de problemas realizado pelos agentes. Esse modelo se baseia em uma série de condições que tornam um problema mais complexo para um agente inteligente, mas, entendo o problema de forma aprofundada, os tipos de condições que ele possui, e outras características que foram discutidas, é possível chegarmos a uma solução aplicando os algoritmos corretos.

## Referências


[1] RUSSELL, Stuart; NORVIG, Peter. *Inteligência Artificial: Uma Abordagem Moderna* – 3ª edição.

[2] OPENAI. Assistente Virtual ChatGPT. Respostas geradas com base em inteligência artificial. Disponível em: https://openai.com. Acesso em: 24 dez. 2024.

[3] WIKIPEDIA. **AC-3 algorithm**. Disponível em: https://en.wikipedia.org/wiki/AC-3_algorithm. Acesso em: 29 dez. 2024.