# Universidade Federal de Ouro Preto
# BCC 325 - Inteligência Artificial
# Problemas de Satisfação de Restrições

Prof. Rodrigo Silva

## 1 Leitura

- Capítulo 3 do Livro *Artificial Intelligence: Foundations of Computational Agents, 2nd Edition* disponível em *https://artint.info/*
- Capítulo 4 do Livro *Artificial Intelligence: Foundations of Computational Agents, 2nd Edition* disponível em *https://artint.info/*
- Aquele que tudo sabe, tudo vê e nada teme.

## 2 Questões

1. Preencha a seguinte tabela com experiência adquirida no projeto

| Estratégia | Seleção da fronteira | Caminho Encontrado | Custo em Espaço |
|---|---|---|---|
| Busca em Largura | | | |
| Busca em Profundidade | | (g) | |
| Guloso | | | |
| Menor Caminho Primeiro | (b) | | |
| $A^*$ | | | |
| Branch and Bound | | | |

   (a) Menor $h(n)$
   (b) Menor $c(S, n)$
   (c) Menor $h(n) + c(S, n)$
   (d) Primeiro caminho adicionado
   (e) Último caminho adicionado
   (f) Menor número de arcos
   (g) Indefinido
   (h) Menor custo
   (i) Linear
   (j) Exponencial

2. Considere o código a seguir e responda.

   (a) Que tipo de agente este código implementa?
   (b) Qual parte do código corresponde a cada componente de um agente como definido no capítulo 2?
   (c) Este agente fica preso em ciclos?
   (d) Qual alteração devemos fazer para que o agente implemente uma poda de ciclos? Apresente o código da alteração.

```python
class Agent:
    def __init__(self, ambiente) -> None:
        self.ambiente = ambiente
        self.percepcoes = ambiente.percepcoes_iniciais()
        self.F = [[self.percepcoes['posicao']]]
        self.C = []
        self.H = []
        self.C_H = []

    def act(self):

        while self.F:
            path = self.F.pop(0)

            action = {'mover_para':path[-1]}

            self.percepcoes = self.ambiente.muda_estado(action)

            if (self.percepcoes['posicao'] == self.percepcoes['saida']).all():
                return path
            else:
                for vi in self.percepcoes['vizinhos']:
                    self.F.append(path + [vi])
```

Figure 1: agent.py

3. (Seção 4.1.3) Quais são os componentes de um problema de satisfação de restrições (Contraint Satisfaction Problem - CSP)? Apresente definições de cada componente.

4. Apresente aplicações reais que podem ser modeladas como problemas de satisfação de restrições?

5. (Seção 4.2) Explique o funcionamento do algoritmo *Generate-and-test*.

6. Considere o problema a seguir:

$$
\begin{aligned}
X &= \{A, B, C\} \\
D &= \{\{1,2,3,4\}, \{1,2,3,4\}, \{1,2,3,4\}\} \\
C &= \{A < B, B < C\}
\end{aligned}
$$

   (a) Qual o tamanho do espaço de busca? Ou seja, no pior caso, quantas soluções candidatas podem ser geradas?

   (b) Represente este problema como uma rede de restrições?

7. (Seção 4.4) Considere o Generalized Arc Concsistency (GAC) Algorithm apresentado abaixo e responda.

   (a) Demonstre a execução do algoritmo de consistência de arcos, GAC (Generalized Arc Consitency Algorithm) para o problema da questão anterior.

   (b) Quais conclusões pode ser tiradas após a execução do GAC, no geral? O que podemos concluir após a execução do GAC para este problema?

   (c) Após a execução do GAC, qual o tamanho do espaço de busca?

```
GAC(<X,D,C>, {<X,c> | c in C and X in scope(c)}):

def GAC(<X,D,C>, to_do):
    while to_do:
        select and remove <X, c> from to_do
        let {Y1,....,Yk} = scope(c) \ {X}
        new_domain = {x|x in D(X) and exists y1 in D(Y1),...,yk in D(Yk)
                      such that c(X=x,Y1=y1,...Yk=yk)==True}
        if new_domain not equal D(X):
            to_do = to_do union {<Z,_c>| {X,Z} in scope(_c), _c!=c, Z!=X}
            D(X) = new_domain
    return D
```

Figure 2: Algoritmo GAC