



# Formação Inteligência Artificial



# Introdução à Inteligência Artificial





# Agentes Inteligentes e Sistemas de Busca



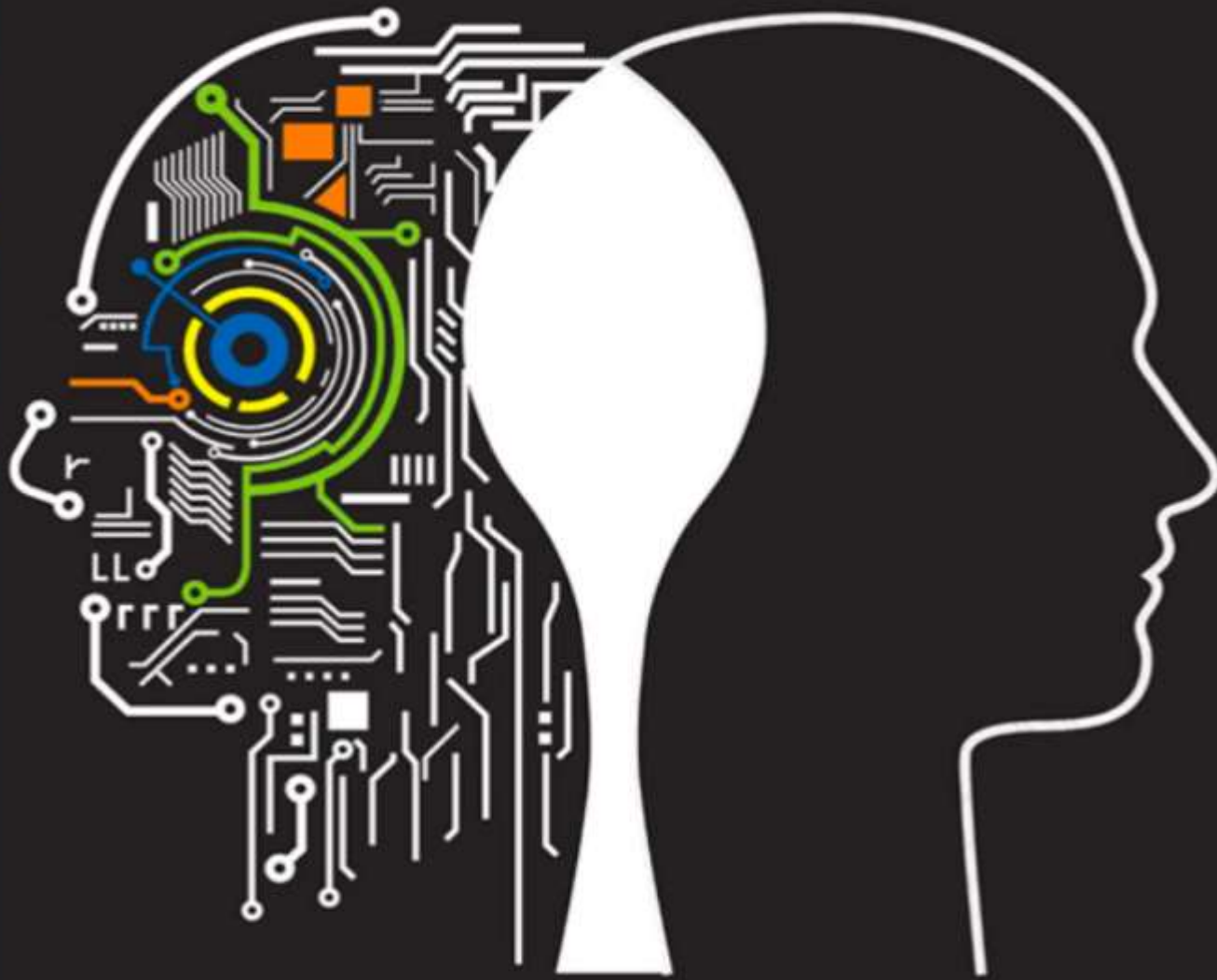
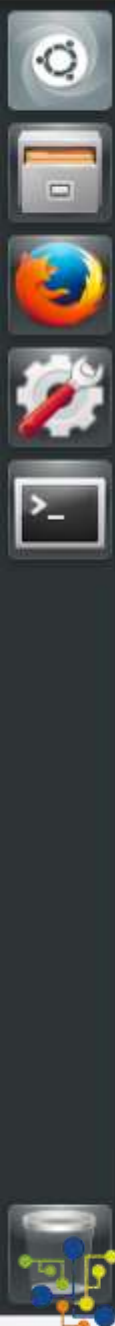


# Agentes Inteligentes e Sistemas de Busca

---

- O que são Agentes Inteligentes?
- Agentes e Ambientes
- Estrutura e Tipos de Agentes
- Agentes de Resolução de Problemas
- Algoritmos de Busca
- Estratégias de Busca Com e Sem Informação
- Funções Heurísticas
- Busca Online, Local e Não-Determinística





Data Science Academy



# O Que são Agentes?





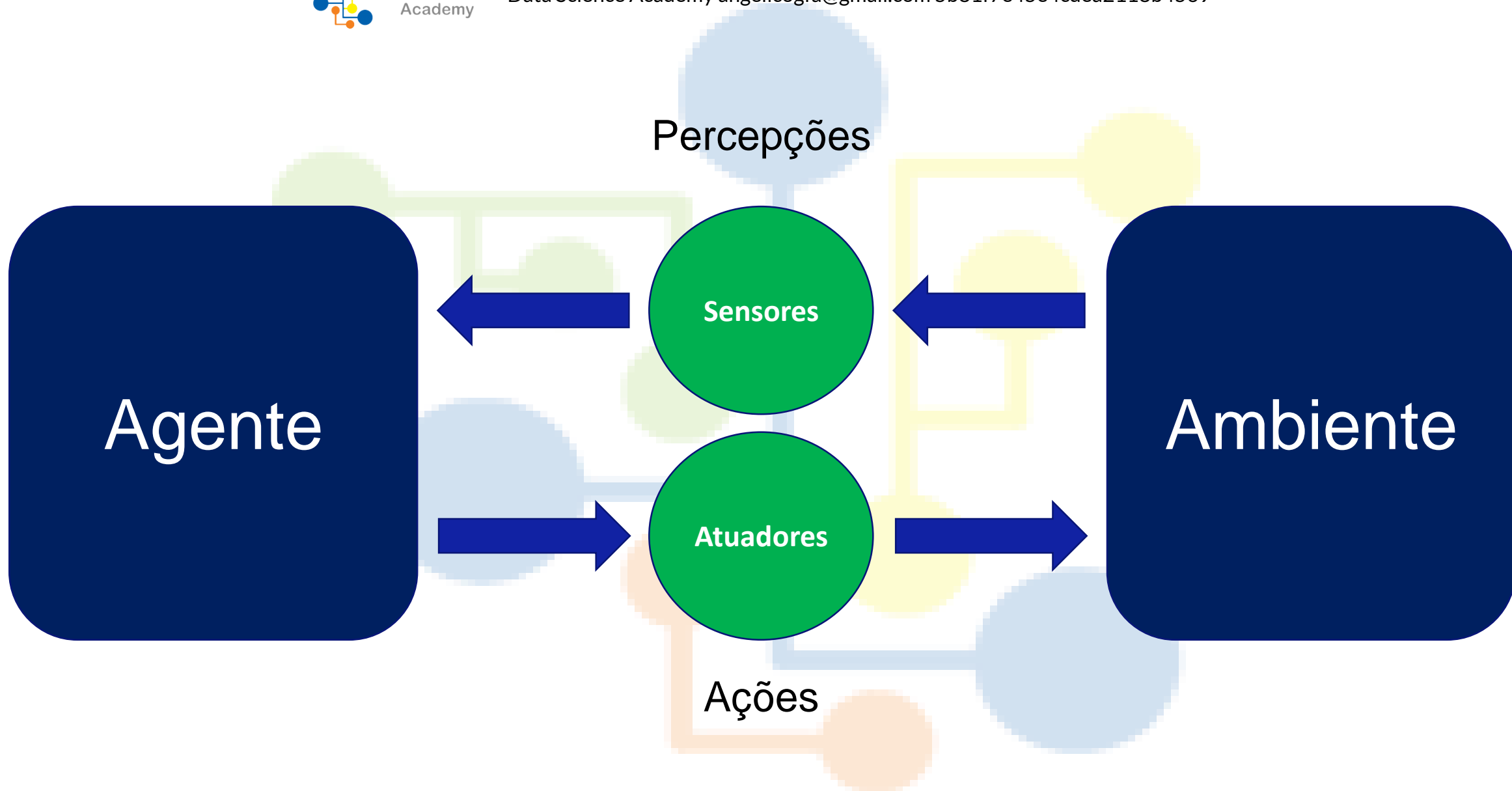


Inteligência

## Agentes Inteligentes

Entidades autónomas, dotadas de uma base de conhecimento e capazes de interagir com o meio em que estão, tomando assim, decisões que irão auxiliar ou até mesmo substituir o trabalho de um humano.









# Exemplos de Agentes

## Agente Humano

- Sensores: olhos, ouvidos, outros órgãos
- Atuadores: mãos, pernas, bocas, outras partes do corpo

## Agente de Software

- Sensores: entrada do teclado, conteúdo de arquivos, pacotes de rede
- Atuadores: monitor, disco rígido, envio de pacotes pela rede

## Agente Robótico

- Sensores: câmeras, sensores
- Atuadores: motores, partes mecânicas

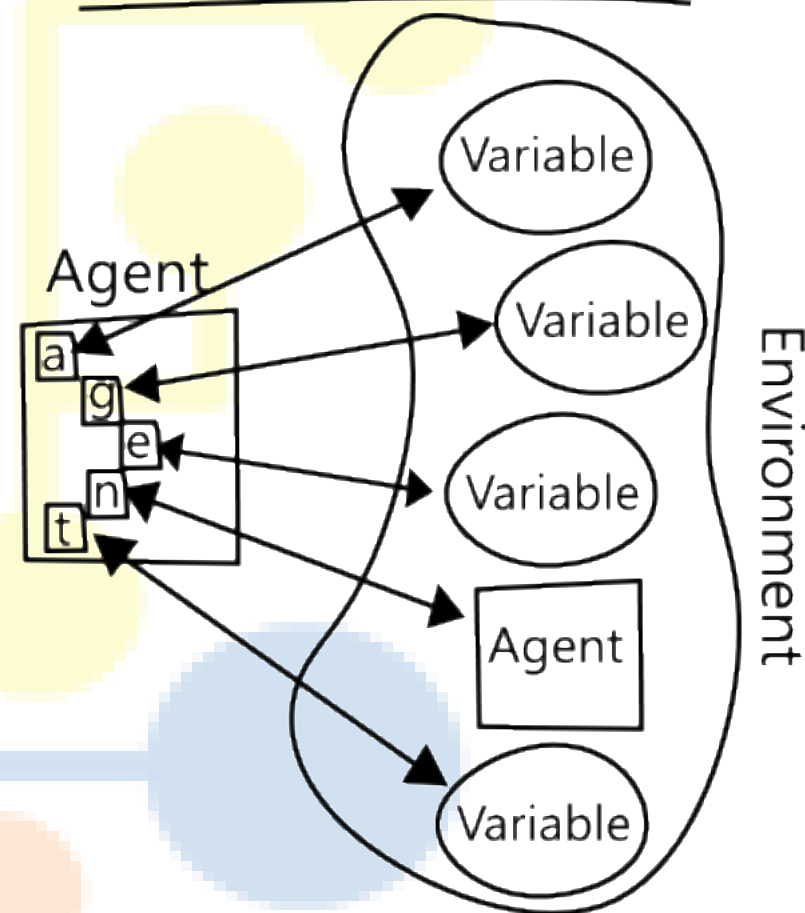




# Agentes Inteligentes

Um agente inteligente pode ser um software desenvolvido para automatizar e executar uma tarefa em uma rede para o usuário

## Intelligent Agent





# Funcionamento do Agente Inteligente

Três características principais  
do agente inteligente:



Inteligência



Interatividade



Autonomia





# Comportamento do Agente Inteligente


$$f = P^* \rightarrow A$$

$P^*$  = uma sequência de percepções e  
 $A$  = uma ação





# Comportamento do Agente Inteligente

- Percepções: local e conteúdo
- Ações: esquerda, direita, aspirar, standby





# Comportamento do Agente Inteligente

- Percepções: local e conteúdo  
Ex: Local: A ou B, Conteúdo: limpo ou sujo
- Ações: esquerda, direita, aspirar, standby

| Sequência de Percepções            | Ação     |
|------------------------------------|----------|
| [A, Limpo]                         | Direita  |
| [A, Sujo]                          | Aspirar  |
| [B, Limpo]                         | Esquerda |
| [B, Sujo]                          | Aspirar  |
| [A, Limpo], [A, Limpo]             | Direita  |
| [A, Limpo], [A, Sujo]              | Aspirar  |
| ...                                | ...      |
| [A, Limpo], [A, Limpo], [A, Limpo] | Direita  |
| [A, Limpo], [A, Limpo], [A, Sujo]  | Aspirar  |
| ...                                | ...      |







# Medida de Desempenho do Agente

Exemplos de medidas de desempenho do agente aspirador de pó:

- quantidade de sujeira aspirada
- gasto de energia
- gasto de tempo
- quantidade de barulho gerado

| Sequência de Percepções            | Ação     |
|------------------------------------|----------|
| [A, Limpo]                         | Direita  |
| [A, Sujo]                          | Aspirar  |
| [B, Limpo]                         | Esquerda |
| [B, Sujo]                          | Aspirar  |
| [A, Limpo], [A, Limpo]             | Direita  |
| [A, Limpo], [A, Sujo]              | Aspirar  |
| ...                                | ...      |
| [A, Limpo], [A, Limpo], [A, Limpo] | Direita  |
| [A, Limpo], [A, Limpo], [A, Sujo]  | Aspirar  |
| ...                                | ...      |





**amazon** echo





# Agentes e Ambiente

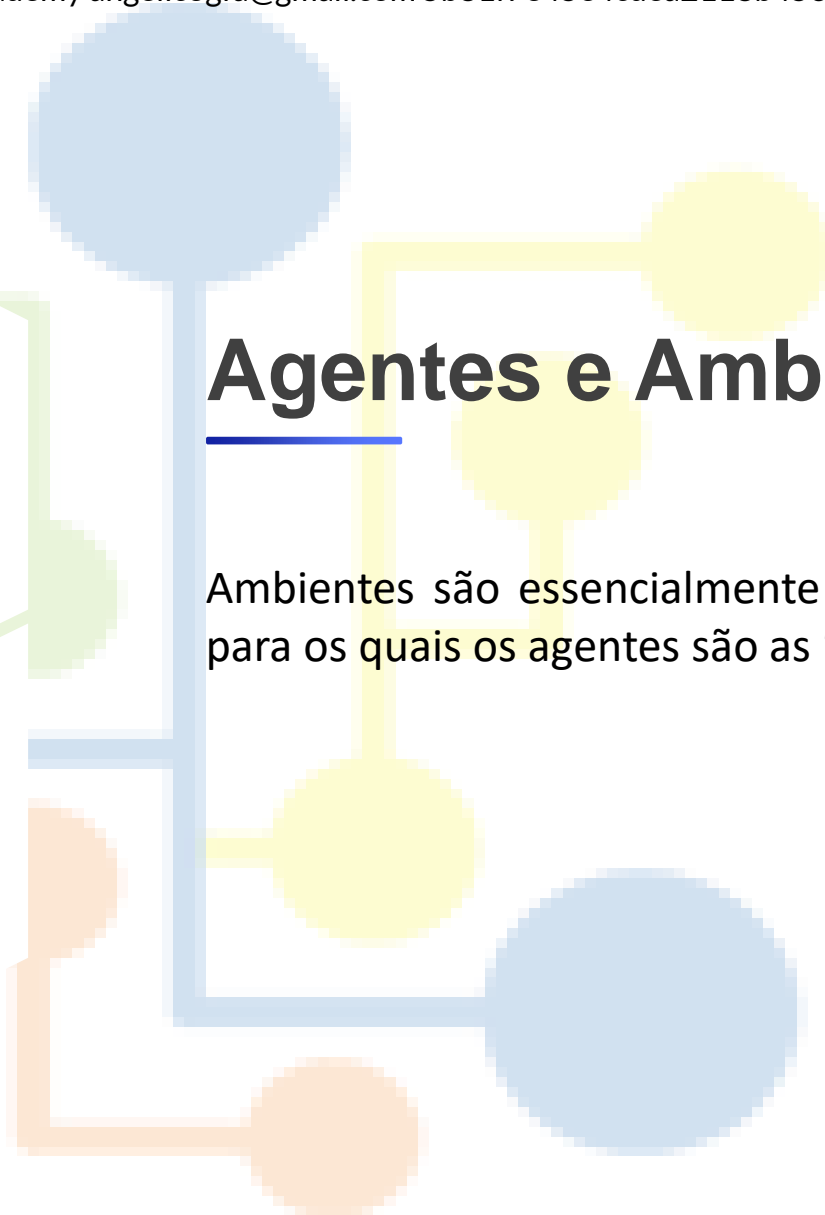




Inteligência

## Agentes e Ambientes

Ambientes são essencialmente os “problemas” para os quais os agentes são as “soluções”.

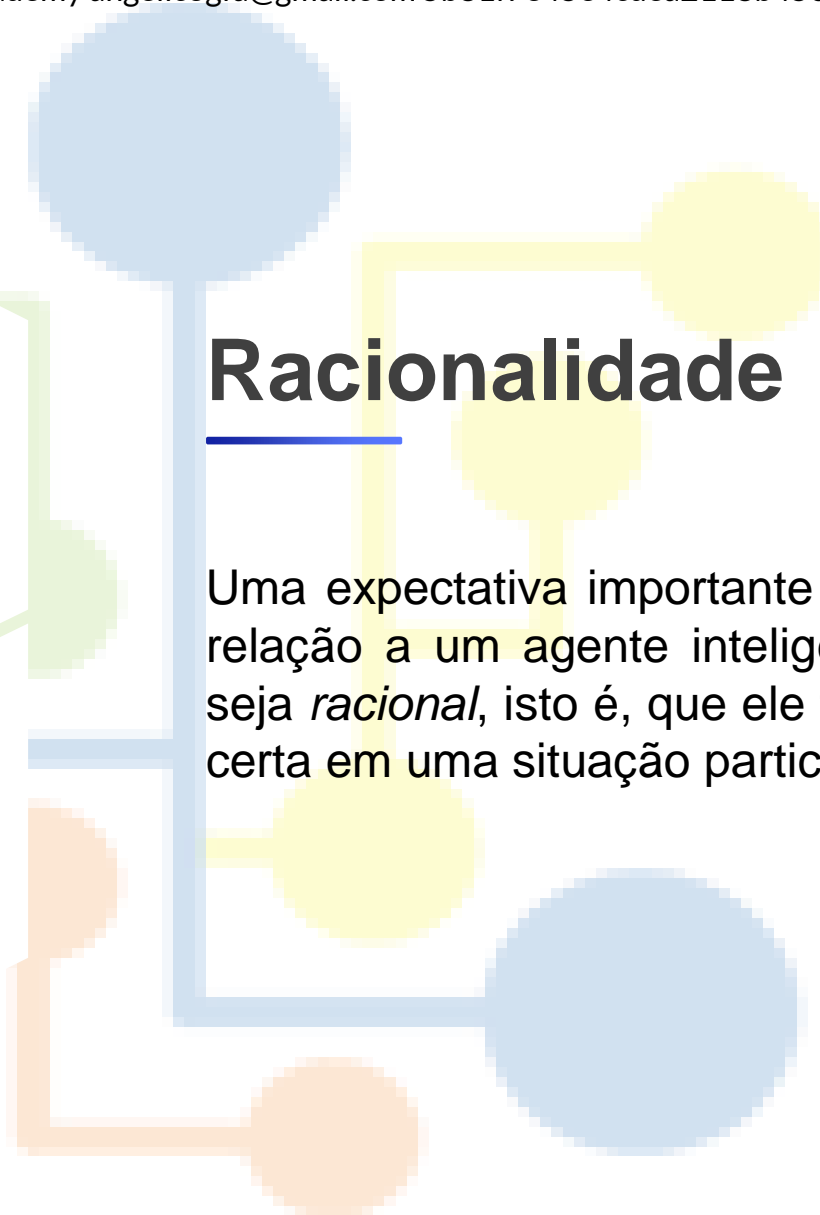




Inteligência

## Racionalidade

Uma expectativa importante que temos em relação a um agente inteligente é que ele seja *racional*, isto é, que ele tome a decisão certa em uma situação particular.





## Racionalidade

Outro aspecto importante a notar é que o agente racional não pode sempre selecionar a ação que seria objetivamente a melhor, pois isso exigiria um conhecimento total sobre o mundo, o que é impossível.

Em outras palavras, para o agente ser racional, não temos a expectativa dele ser *onisciente*.



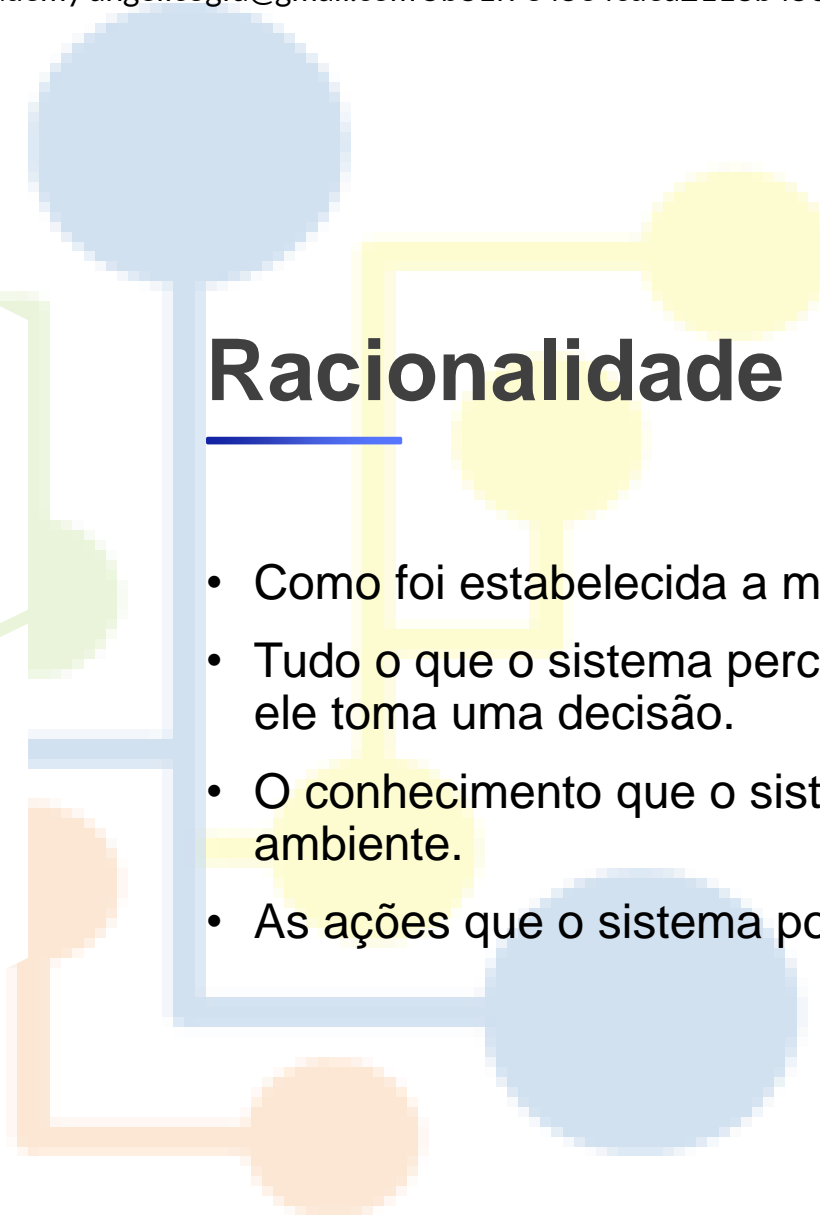






## Racionalidade

- Como foi estabelecida a medida de performance.
- Tudo o que o sistema percebeu até o momento que ele toma uma decisão.
- O conhecimento que o sistema tem sobre o ambiente.
- As ações que o sistema pode efetuar.





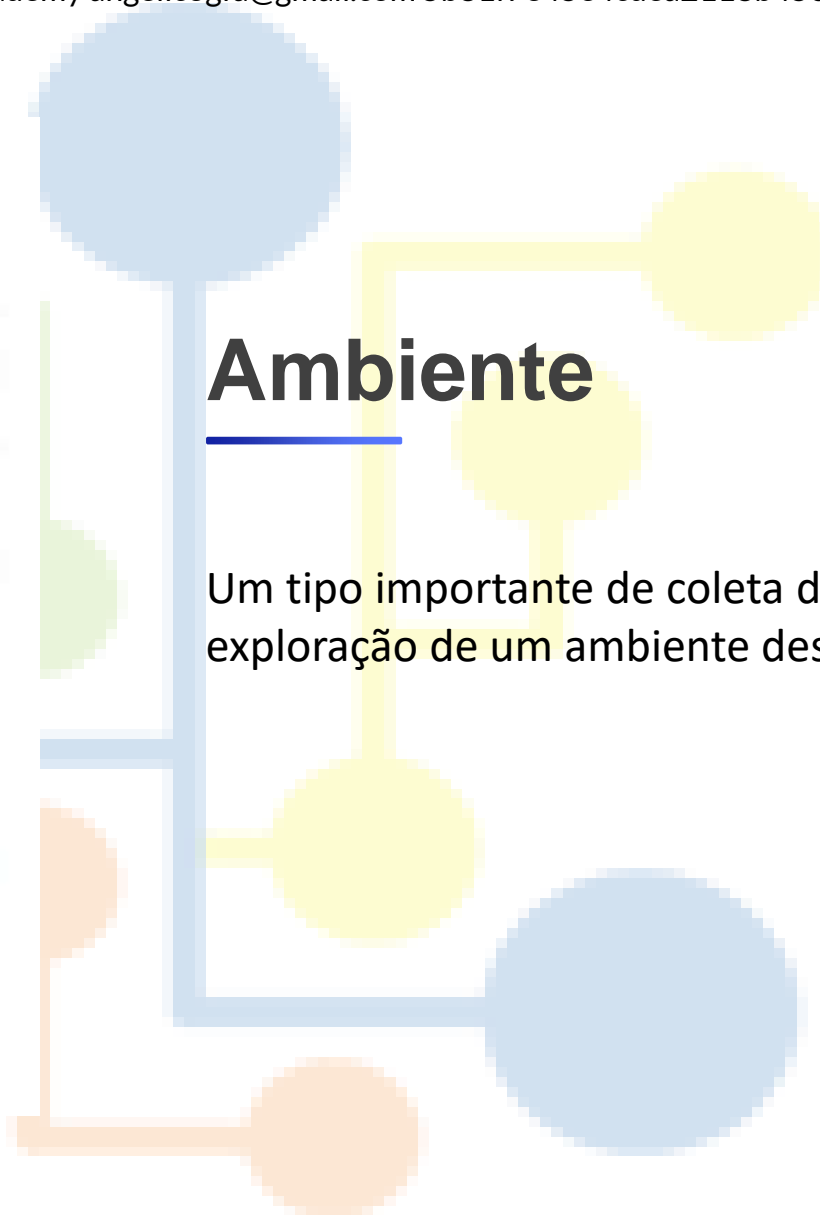
*Um agente racional ideal faz as ações que maximizam a medida de performance, considerando o que ele pode deduzir das informações e que ele recebe do ambiente onde ele atua e o tipo de conhecimento que ele tem.*

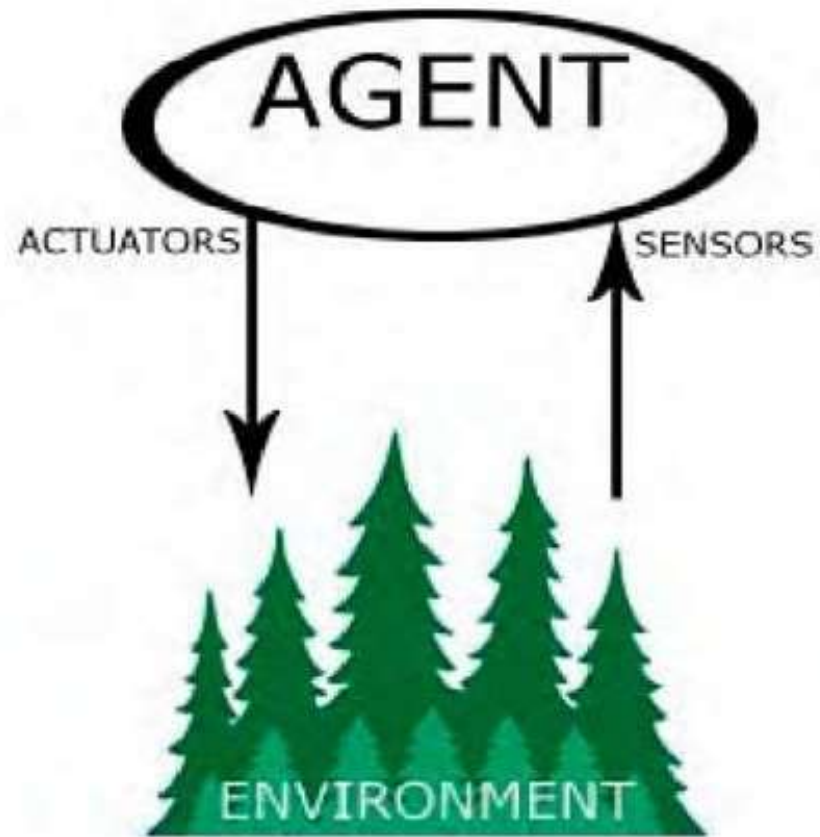




## Ambiente

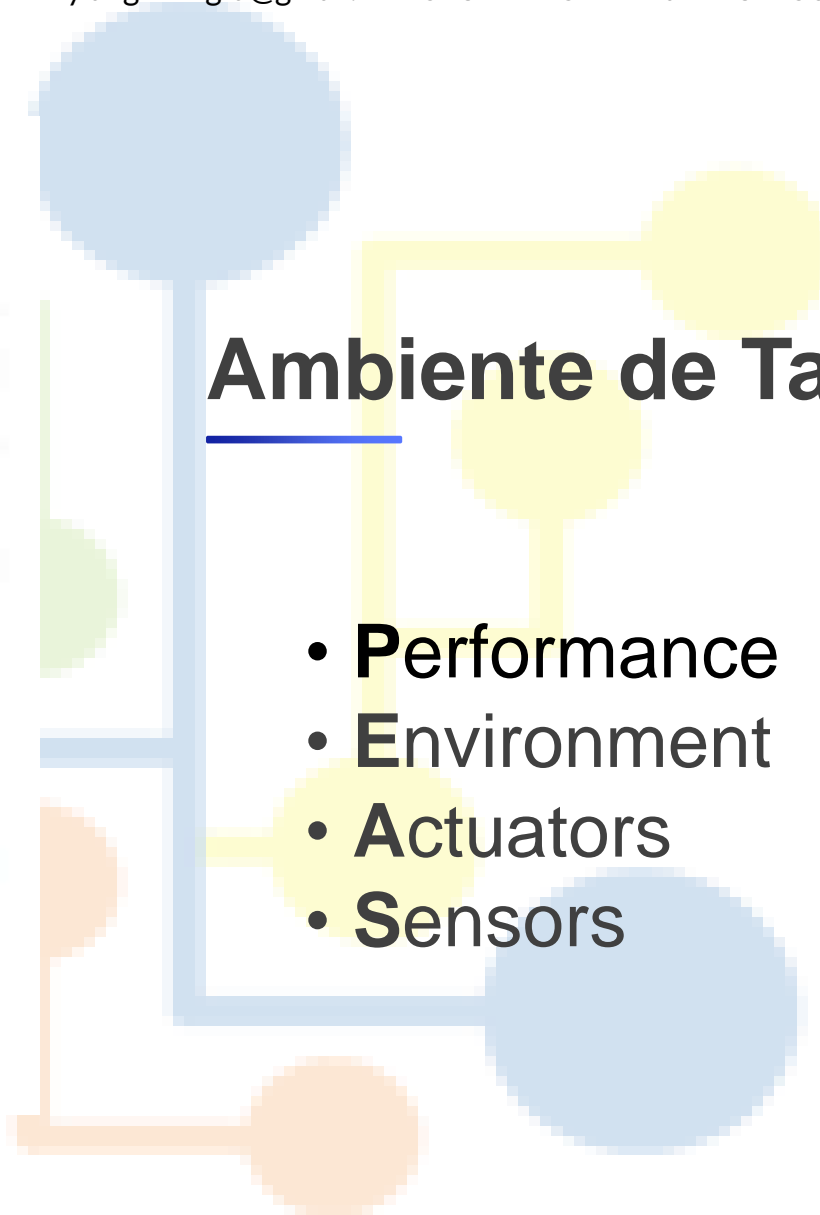
Um tipo importante de coleta de informação é a exploração de um ambiente desconhecido.





## Ambiente de Tarefas - PEAS

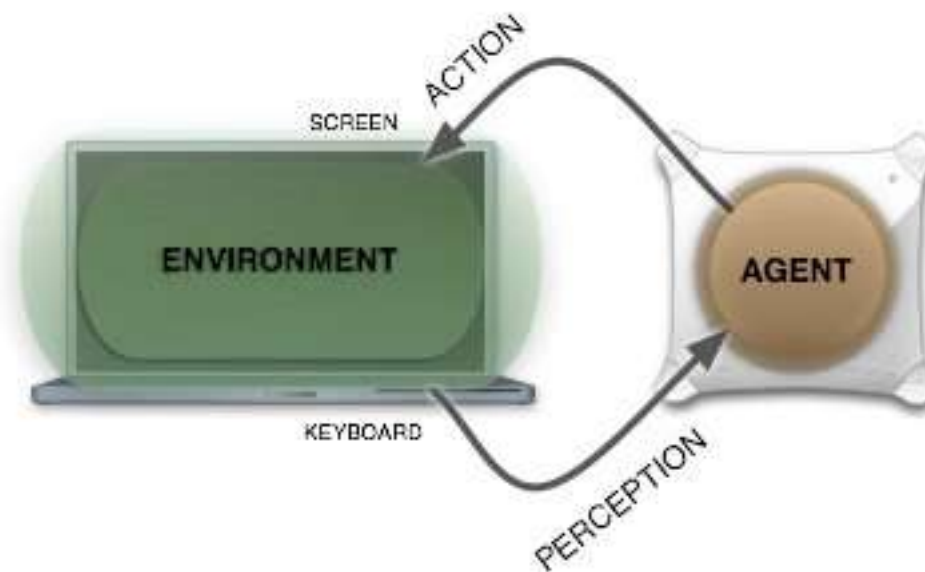
- **Performance**
- **Environment**
- **Actuators**
- **Sensors**





# Ambiente

| Sequência de Percepções            | Ação     |
|------------------------------------|----------|
| [A, Limpo]                         | Direita  |
| [A, Sujo]                          | Aspirar  |
| [B, Limpo]                         | Esquerda |
| [B, Sujo]                          | Aspirar  |
| [A, Limpo], [A, Limpo]             | Direita  |
| [A, Limpo], [A, Sujo]              | Aspirar  |
| ...                                | ...      |
| [A, Limpo], [A, Limpo], [A, Limpo] | Direita  |
| [A, Limpo], [A, Limpo], [A, Sujo]  | Aspirar  |
| ...                                | ...      |







# Agentes Inteligentes

|   |  |
|---|--|
| Sistema que pensa<br>como o ser humano<br><i>(modelo cognitivo)</i> | Sistema que pensa<br>racionalmente<br><i>(lógica)</i>            |
| Sistema que age como o<br>ser humano<br><i>(teste de Turing)</i>    | Sistema que age<br>racionalmente<br><i>(agentes inteligente)</i> |



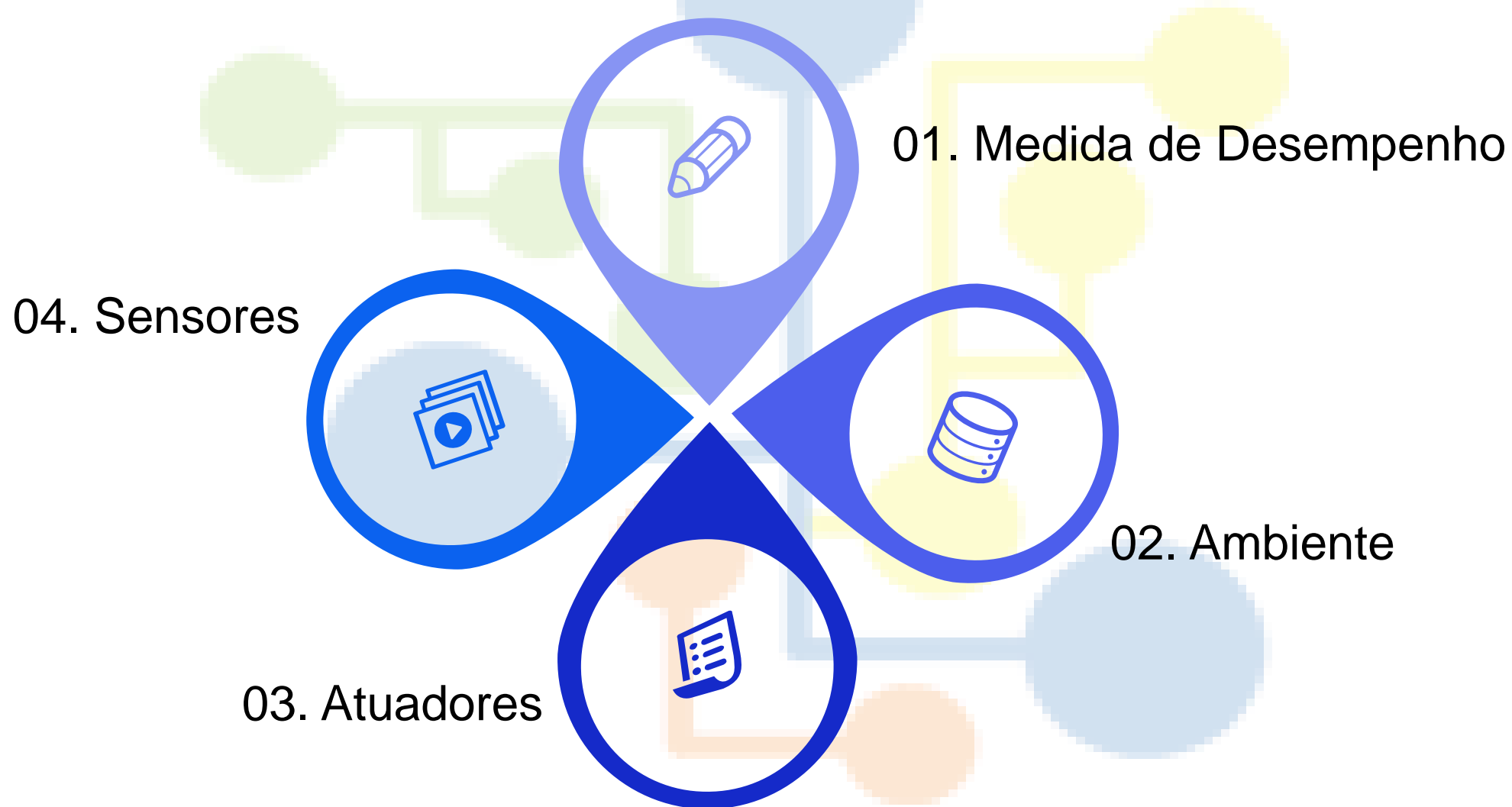


# Ambiente e Modelagem do Agente





# Modelagem do Agente





# Agente - Motorista de Táxi Automatizado

| Tipo de agente    | Medida de desempenho  | Ambiente   | Atuadores  | Sensores  |
|-------------------|---|--|--|---|
| Motorista de táxi | Viagem segura, rápida, dentro da lei, confortável, maximizar lucros | Estradas, outros tipos de tráfego, pedestres, clientes | Direção, acelerador, freio, sinal, buzina, visor | Câmeras, sonar, velocímetro, GPS, hodômetro, acelerômetro, sensores do motor, teclado |





# Outros Exemplos

| <b>Tipo de agente</b>                     | <b>Medida de desempenho</b>               | <b>Ambiente</b>                            | <b>Atuadores</b>  | <b>Sensores</b>  |
|---|---|--|---|--|
| Sistema de diagnóstico médico             | Paciente saudável, minimizar custos       | Paciente, hospital, equipe                 | Exibir perguntas, testes, diagnósticos, tratamentos, indicações | Entrada pelo teclado para sintomas, descobertas, respostas do paciente |
| Sistema de análise de imagens de satélite | Definição correta da categoria da imagem  | Link de transmissão de satélite em órbita  | Exibir a categorização da cena                                  | Arrays de pixels em cores  |
| Robô de seleção de peças                  | Porcentagem de peças em bandejas corretas | Correia transportadora com peças; bandejas | Braço e mão articulados   | Câmera, sensores angulares articulados                                 |



# Propriedades do Ambiente

---

## Completamente Observável

Se os sensores de um agente permitem acesso ao estado completo do ambiente em cada instante, dizemos que o ambiente é completamente observável

## Parcialmente Observável

Um ambiente poderia ser parcialmente observável devido ao ruído e a sensores imprecisos ou porque partes do estado estão simplesmente ausentes nos dados do sensor



# Propriedades do Ambiente

---

## Determinístico

O próximo estado do ambiente é completamente determinado pelo estado atual e pela ação executada pelo agente.

## Não-Determinístico

O próximo estado do ambiente é desconhecido. Não se tem certeza do que pode acontecer com o ambiente ao executar uma ação.



# Propriedades do Ambiente

---

## Estático

O ambiente não muda enquanto o agente pensa.

## Dinâmico

O ambiente pode mudar enquanto o agente pensa ou está executando uma ação.



# Propriedades do Ambiente

---

## Discreto

Um número limitado e claramente definido de percepções, ações e estados.

## Contínuo

Um número possivelmente infinito de percepções, ações e estados.



# Propriedades do Ambiente

---

## Agente Único

Um único agente.

## Multi-Agente

Vários agentes interagindo no ambiente. Esses agentes podem ser cooperativos ou competitivos.



# Propriedades do Ambiente

|                          | Xadrez | Taxista Automático | Poker | Diagnostico Medico |
|--------------------------|--------|--------------------|-------|--------------------|
| Completamente observável | Sim    | Não                | Não   | Não                |
| Determinístico           | Sim    | Não                | Não   | Não                |
| Estático                 | Sim    | Não                | Sim   | Não                |
| Discreto                 | Sim    | Não                | Sim   | Não                |
| Agente único             | Não    | Não                | Não   | Sim                |







# A Estrutura dos Agentes





Inteligência

## Arquitetura

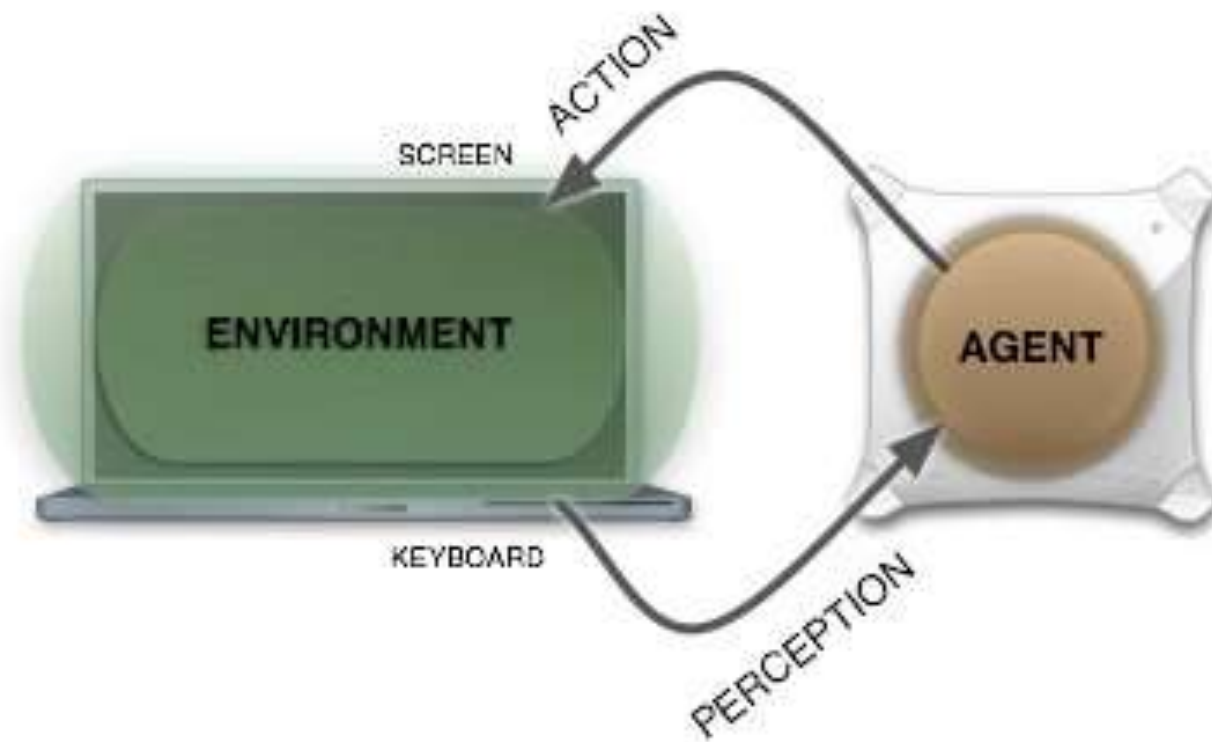
agente = arquitetura + programa







# Arquitetura





# Arquitetura

**função** AGENTE-DIRIGIDO-POR-TABELA(*percepção*) **retorna** uma ação  
**variáveis estáticas:** *percepções*, uma sequência, inicialmente vazia  
*tabela*, uma tabela de ações, indexada por sequências de percepções,  
inicialmente completamente especificada  
  
anexar *percepção* ao fim de *percepções*  
*ação*  $\leftarrow$  ACESSAR(*percepções*, *tabela*)  
**retornar** *ação*

A entrada visual de uma única câmera chega à velocidade de aproximadamente 27 megabytes por segundo (30 quadros por segundo, 640 × 480 pixels com 24 bits de informações de cores). Isso nos dá uma tabela de pesquisa com mais de 10.250.000.000.000 entradas para uma hora de direção





# Tipos de Agentes Inteligentes





# Tipos de Agentes



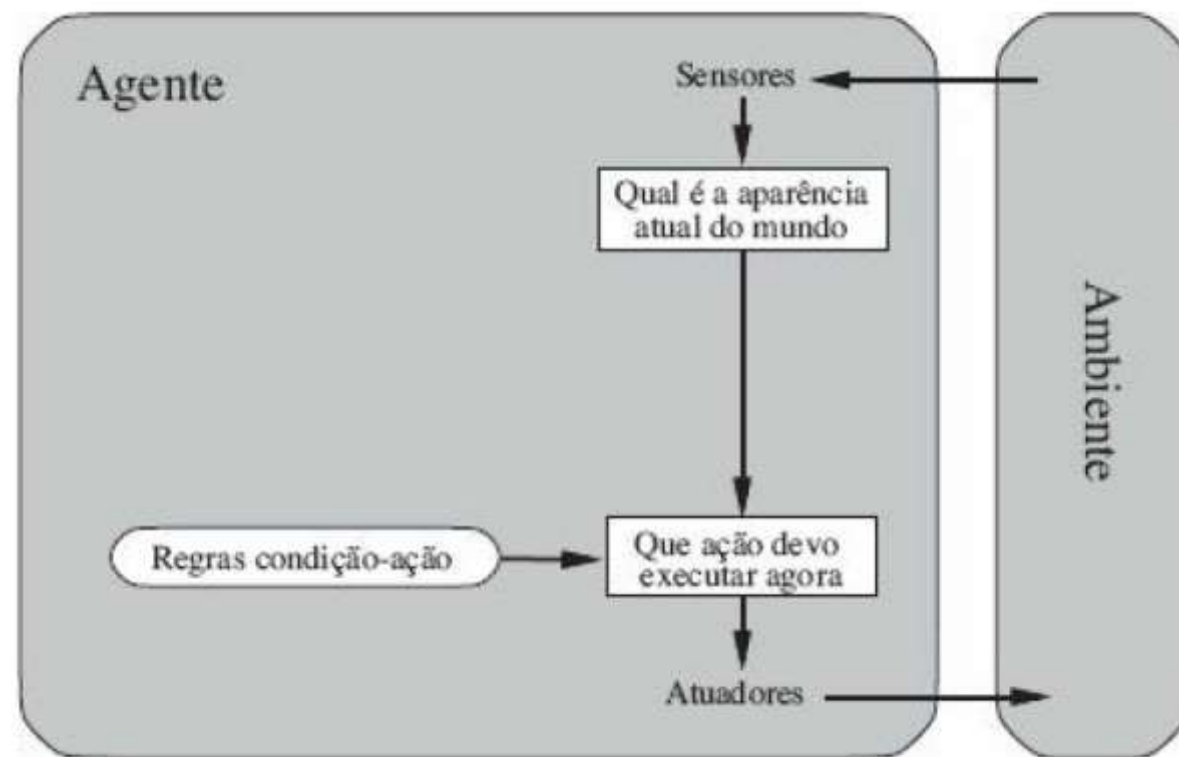


# Tipos de Agentes



## Agentes Reativos Simples

- O funcionamento do agente reativo é baseado em regras de condição-ação:  
*if condição then ação.*
- São simples e limitados:
  - Funcionará somente se a decisão correta puder ser tomada com base apenas na percepção atual.
  - A tabela de regras condição-ação pode se tornar muito grande em problemas complexos.
  - Ambiente completamente observável.





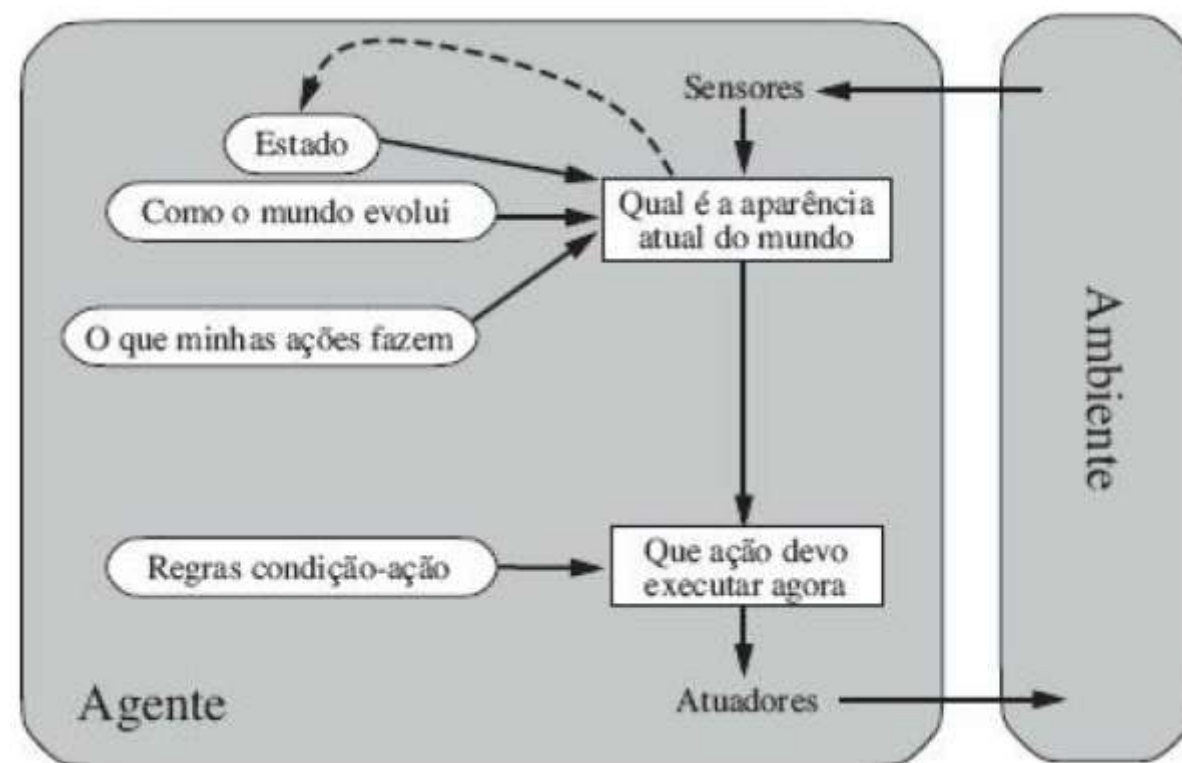


# Tipos de Agentes



## Agentes Reativos Baseados em Modelos

- Um agente reativo baseado em modelo pode lidar com ambientes parcialmente observáveis, mas o agente deve controlar as partes do mundo que ele não pode ver.
- O agente deve manter um estado interno que dependa do histórico de percepções e reflita os aspectos não observados no estado atual.
- Agente baseado em modelo é um agente que usa um modelo de mundo.





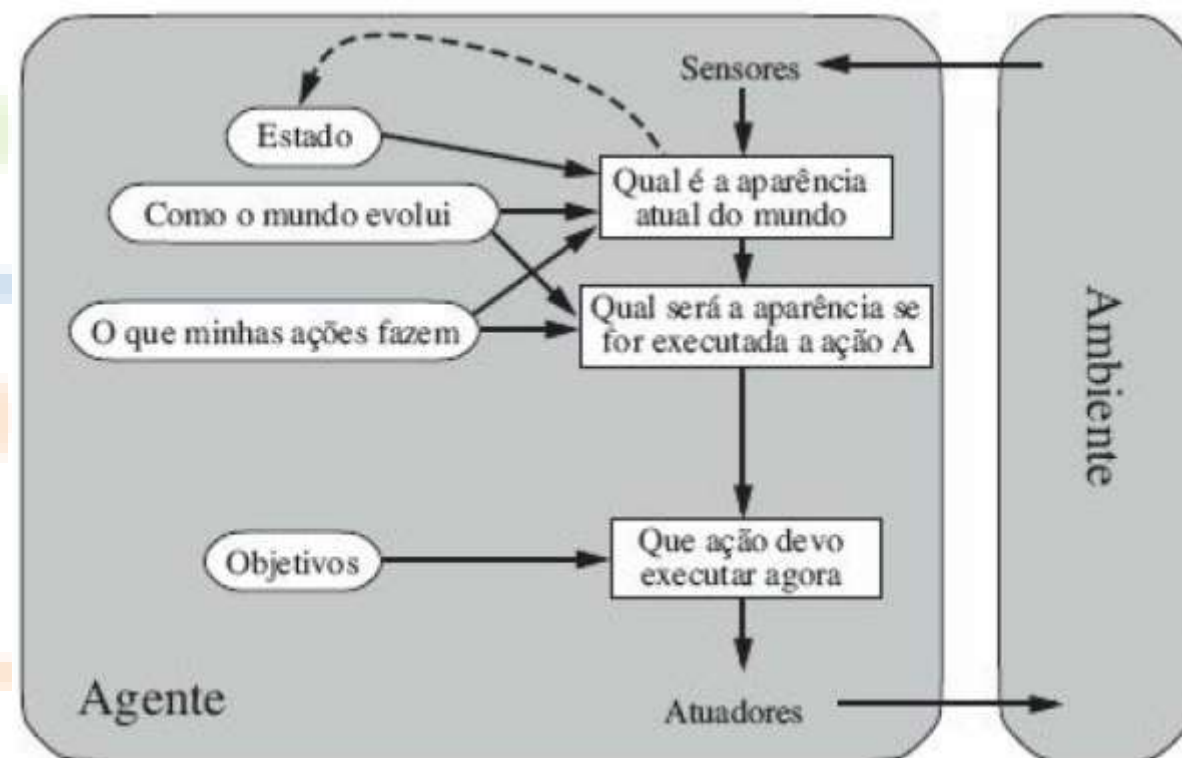


# Tipos de Agentes



## Agentes Baseados em Objetivos

- Agentes baseados em objetivos expandem as capacidades dos agentes baseados em modelos através de um “objetivo”.
- O objetivo descreve situações desejáveis. Exemplo: estar no destino
- A seleção da ação baseada em objetivo pode ser:
  - Direta: quando o resultado de uma única ação atinge o objetivo.
  - Mais complexa: quando será necessário longas sequências de ações para atingir o objetivo.



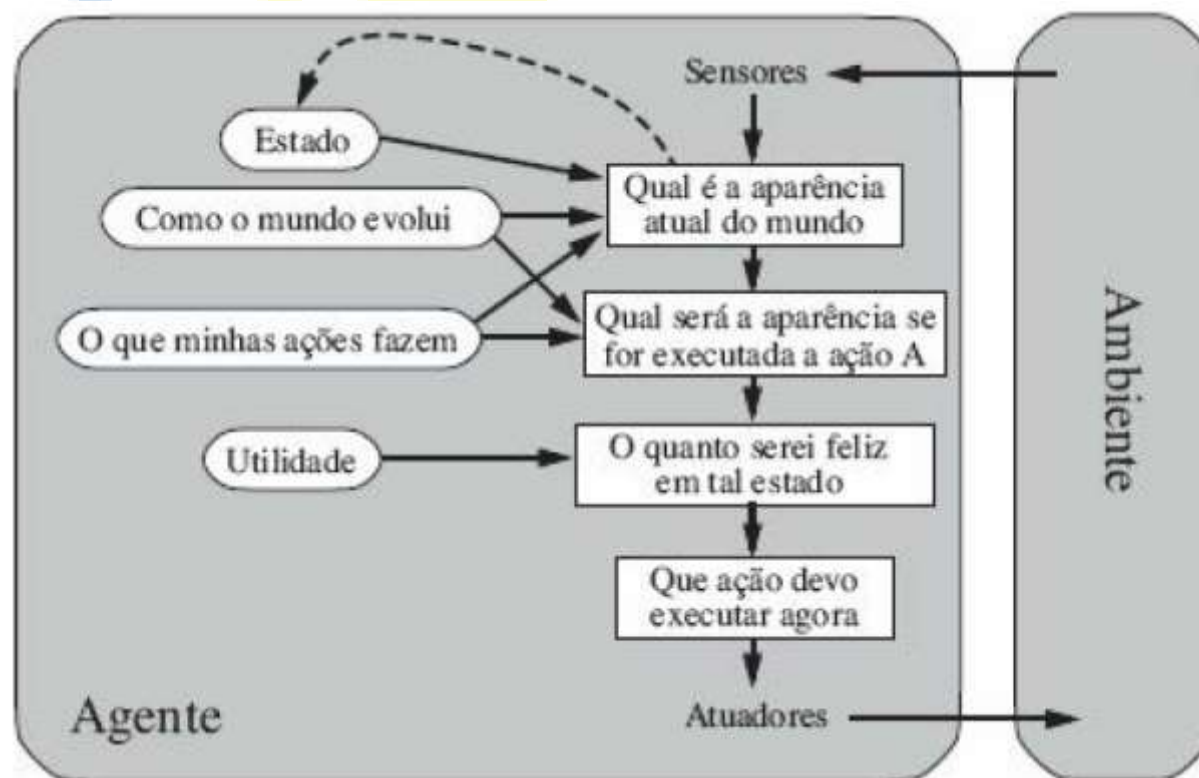


# Tipos de Agentes



## Agentes Baseados na Utilidade

- Agentes baseados na utilidade buscam definir um grau de satisfação com os estados. O quanto o agente está “feliz” com aquele estado.
- Se um estado do mundo é mais desejável que outro, então ele terá maior utilidade para o agente.
- Utilidade é uma função que mapeia um estado para um número real que representa o grau de satisfação com este estado.



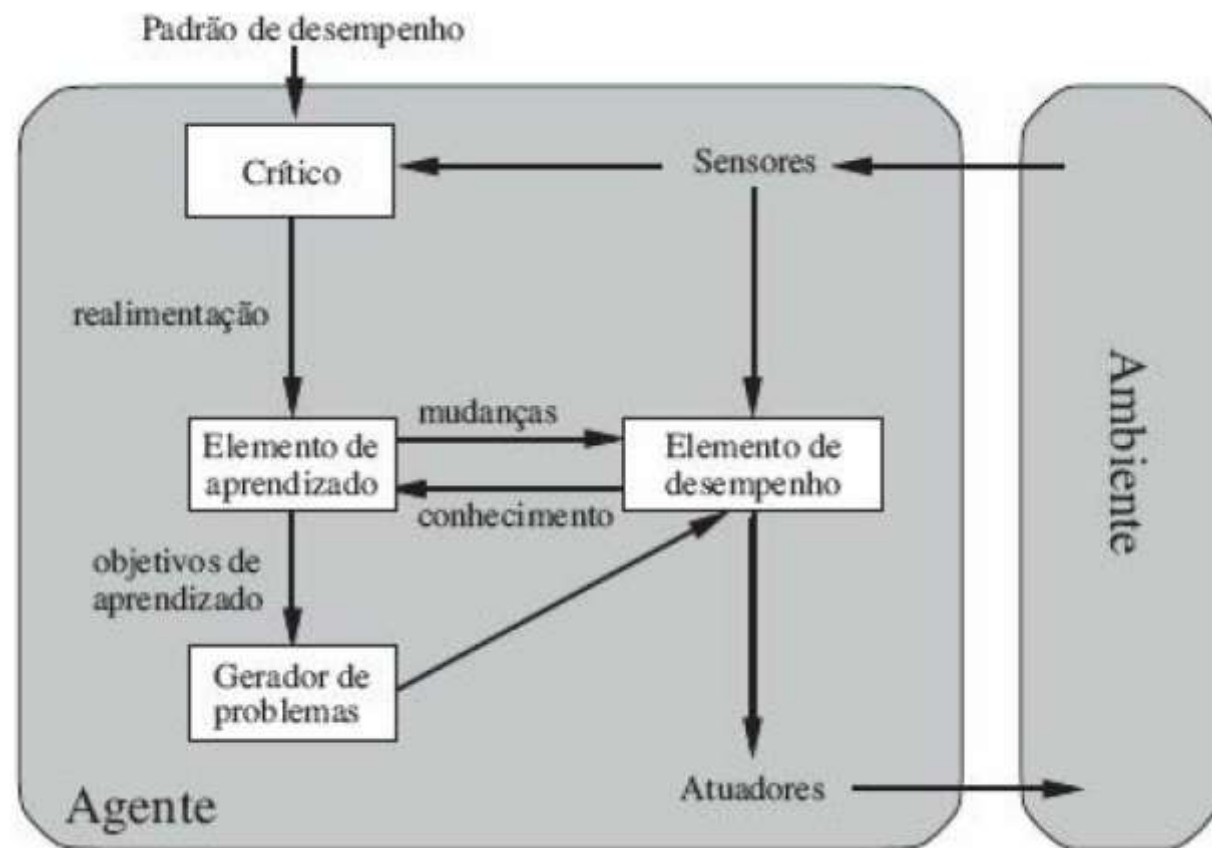


# Tipos de Agentes



## Agentes com Aprendizagem

- Agentes com aprendizado podem atuar em ambientes totalmente desconhecidos e se tornar mais eficientes do que o seu conhecimento inicial poderia permitir.
- Em agentes sem aprendizagem, tudo o que o agente sabe foi colocado nele pelo projetista.





# Tipos de Agentes

An abstract diagram in the background consists of several colored circles (blue, green, yellow, orange) connected by lines of the same color, forming a network-like structure. A central white rectangular box with a thin black border contains the text 'Em Resumo...'.

*Em Resumo...*

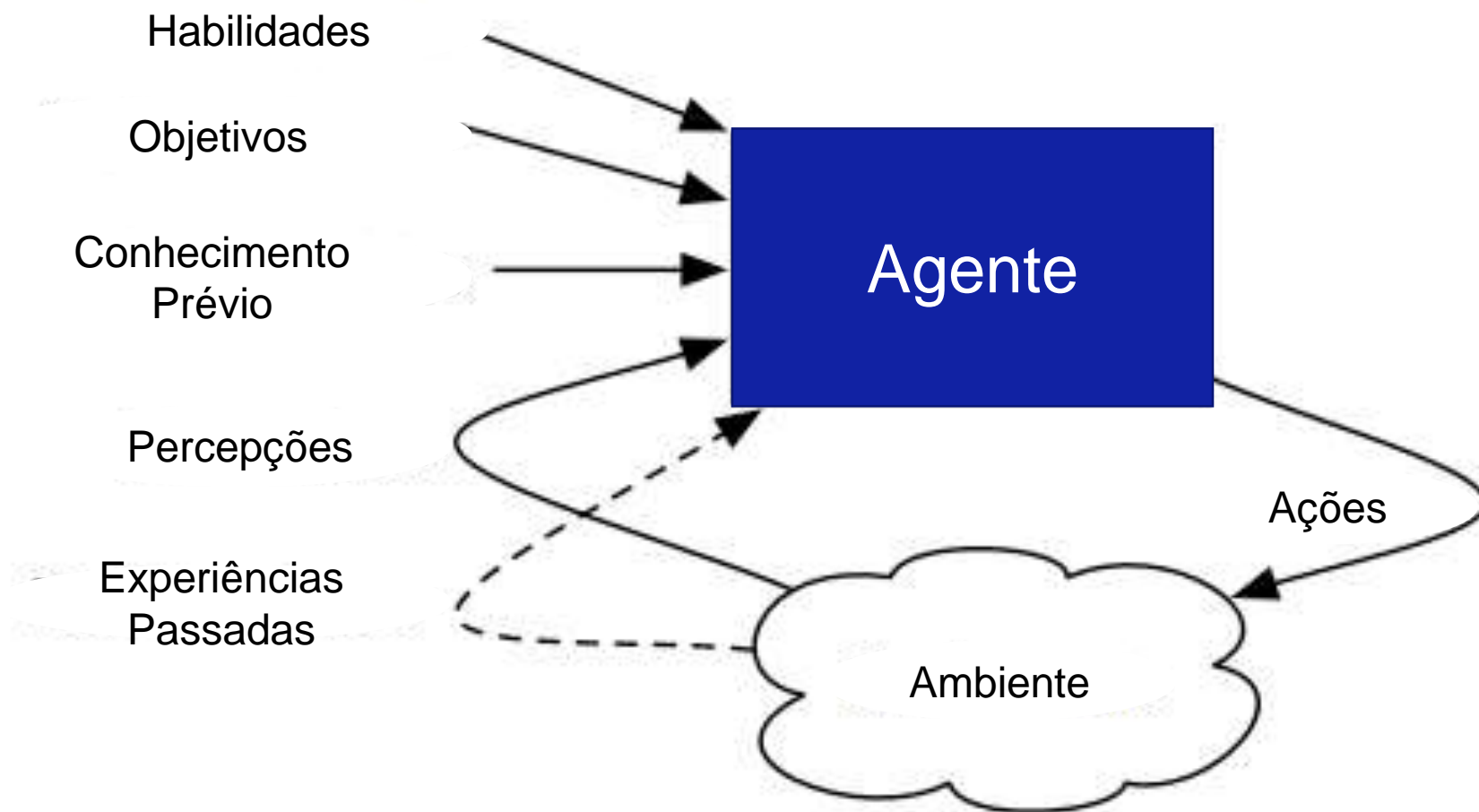


# Agente de Resolução de Problemas



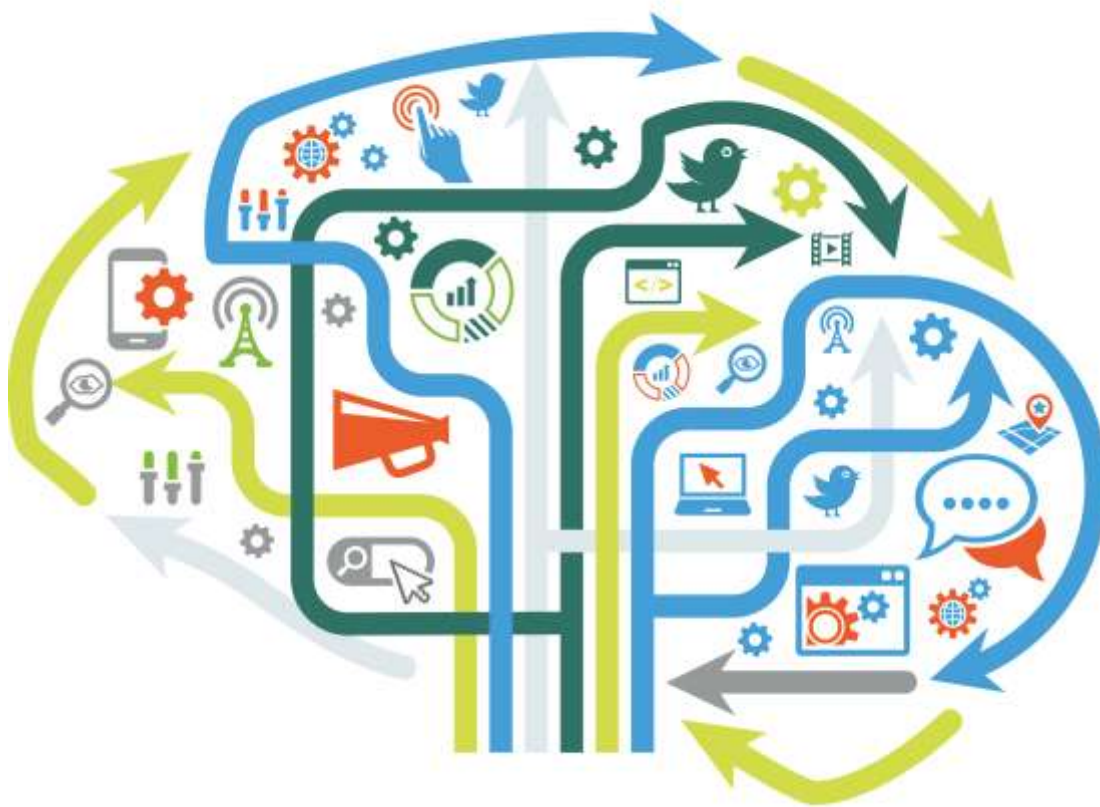


# Agentes





# Agentes de Resolução de Problemas



Os agentes de resolução de problemas utilizam representações **atômicas**, ou seja, os estados do mundo são considerados como um todo, sem estrutura interna visível para os algoritmos de resolução de problemas.



# Agentes de Resolução de Problemas





# Agentes de Resolução de Problemas



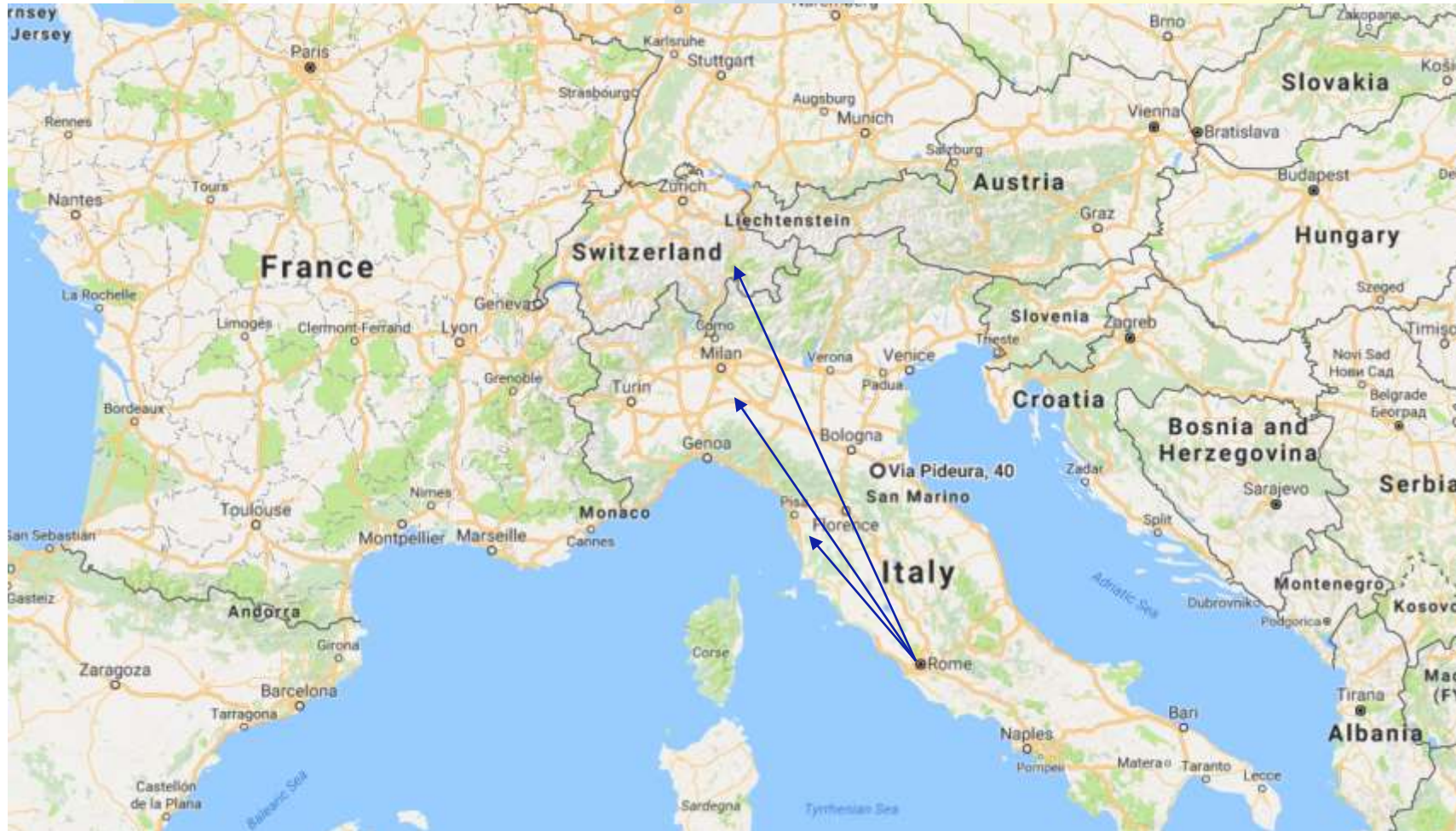
**A formulação de problemas** é o processo de decidir que ações e estados devem ser considerados, dado um objetivo.





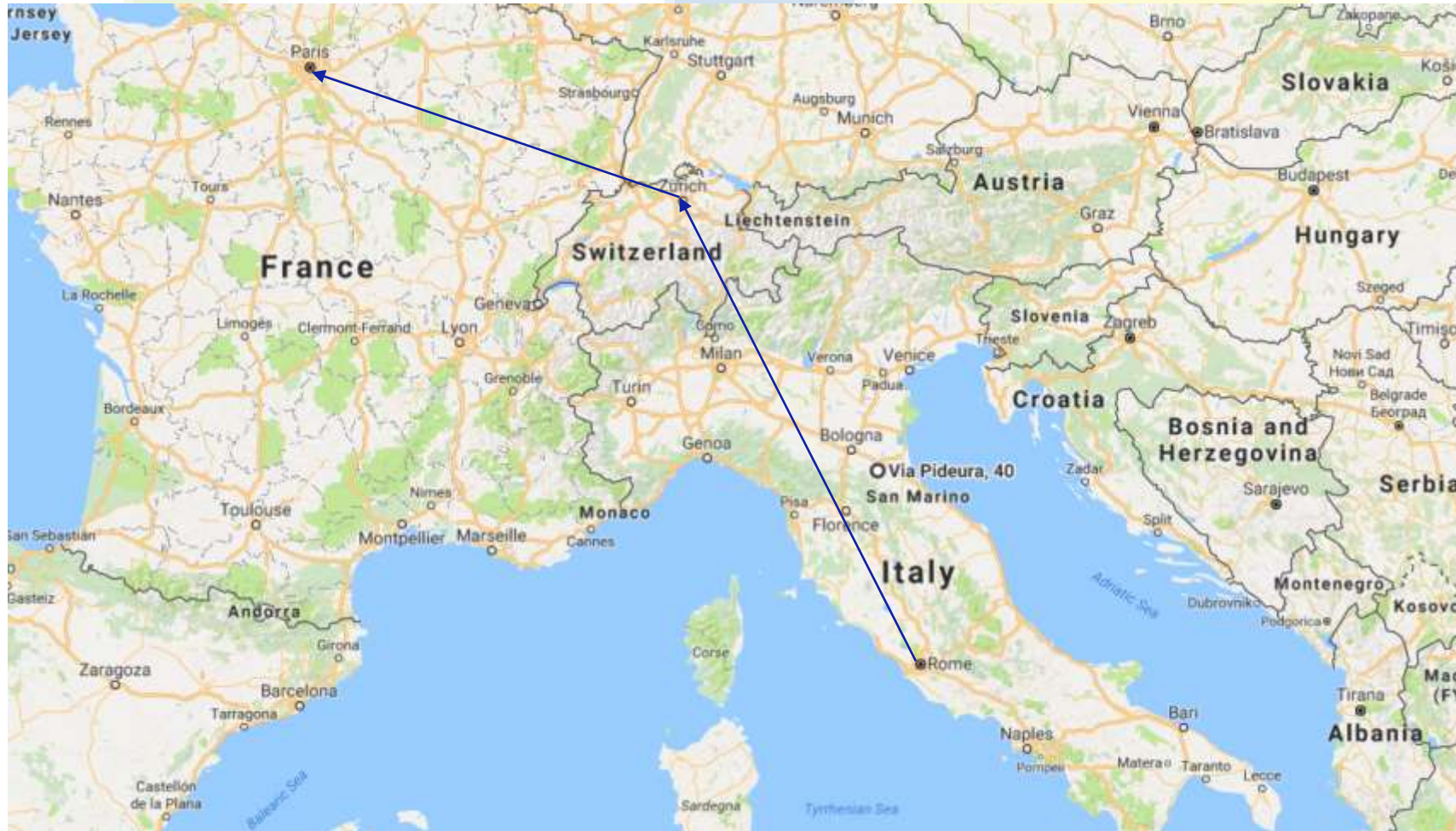


# Agentes de Resolução de Problemas





# Agentes de Resolução de Problemas







# Agentes de Resolução de Problemas



- Observável
- Discreto
- Conhecido
- Determinístico

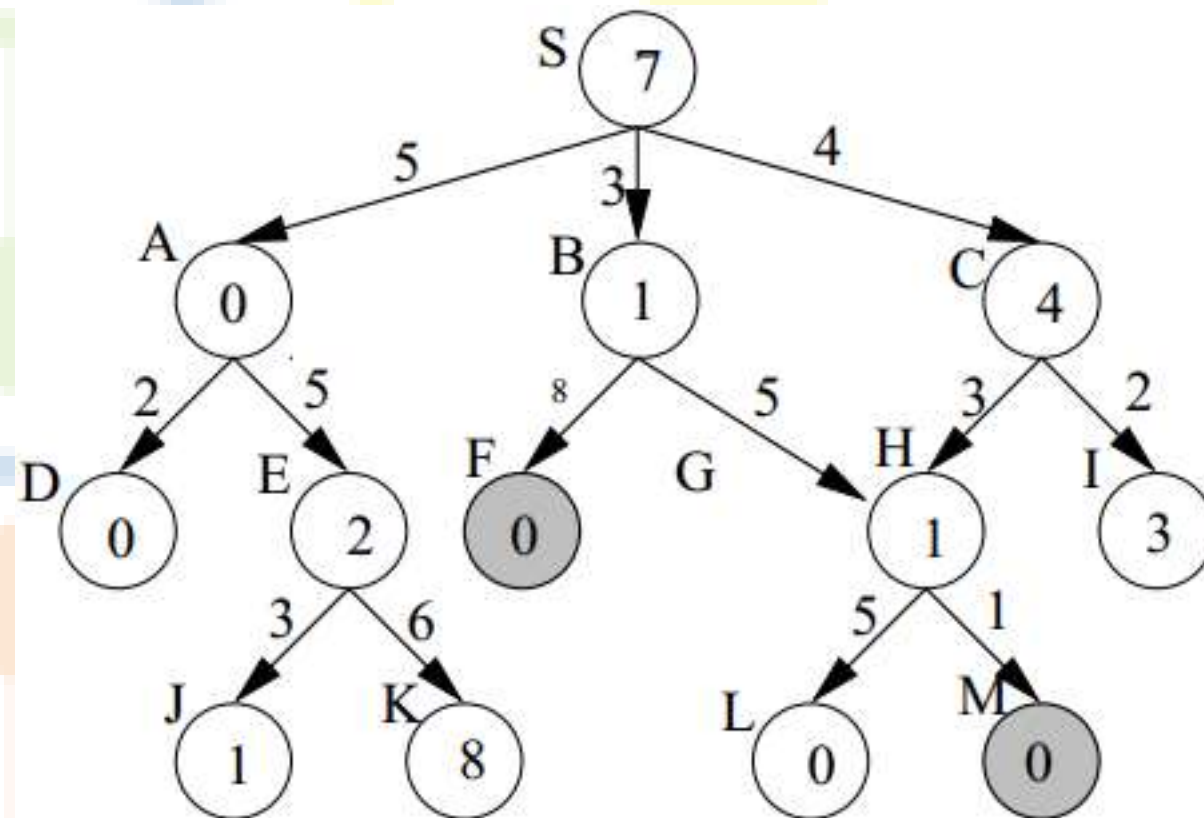






# Agentes de Resolução de Problemas

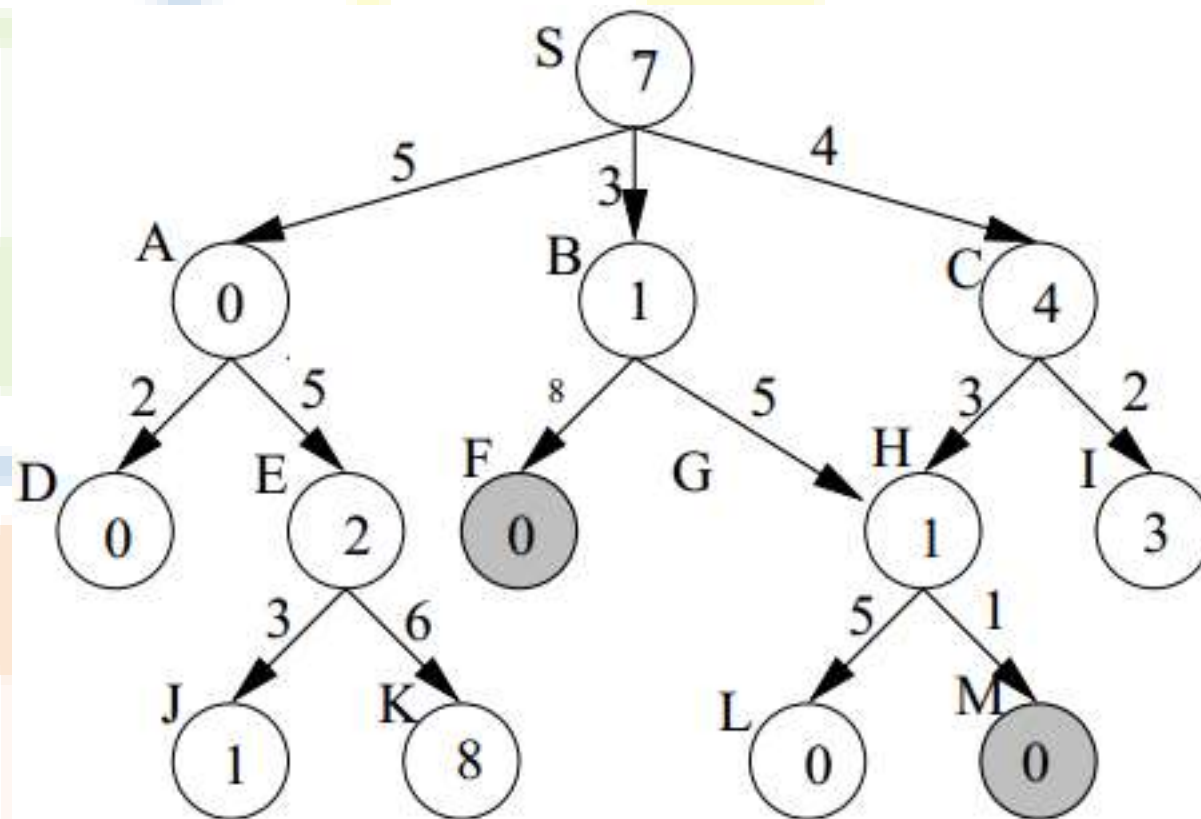
*Sob essas premissas, a  
solução para qualquer  
problema é uma sequência  
fixa de ações*





# Agentes de Resolução de Problemas

Esse processo de procurar por tal sequência de ações que alcançam o objetivo é chamado de **busca**





# Componentes de um Problema

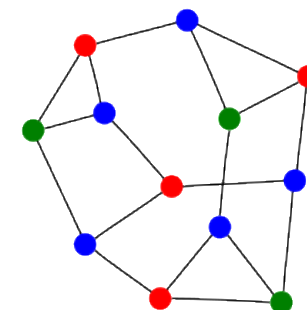




# Componentes de um Problema



Espaço de  
estados do  
problema



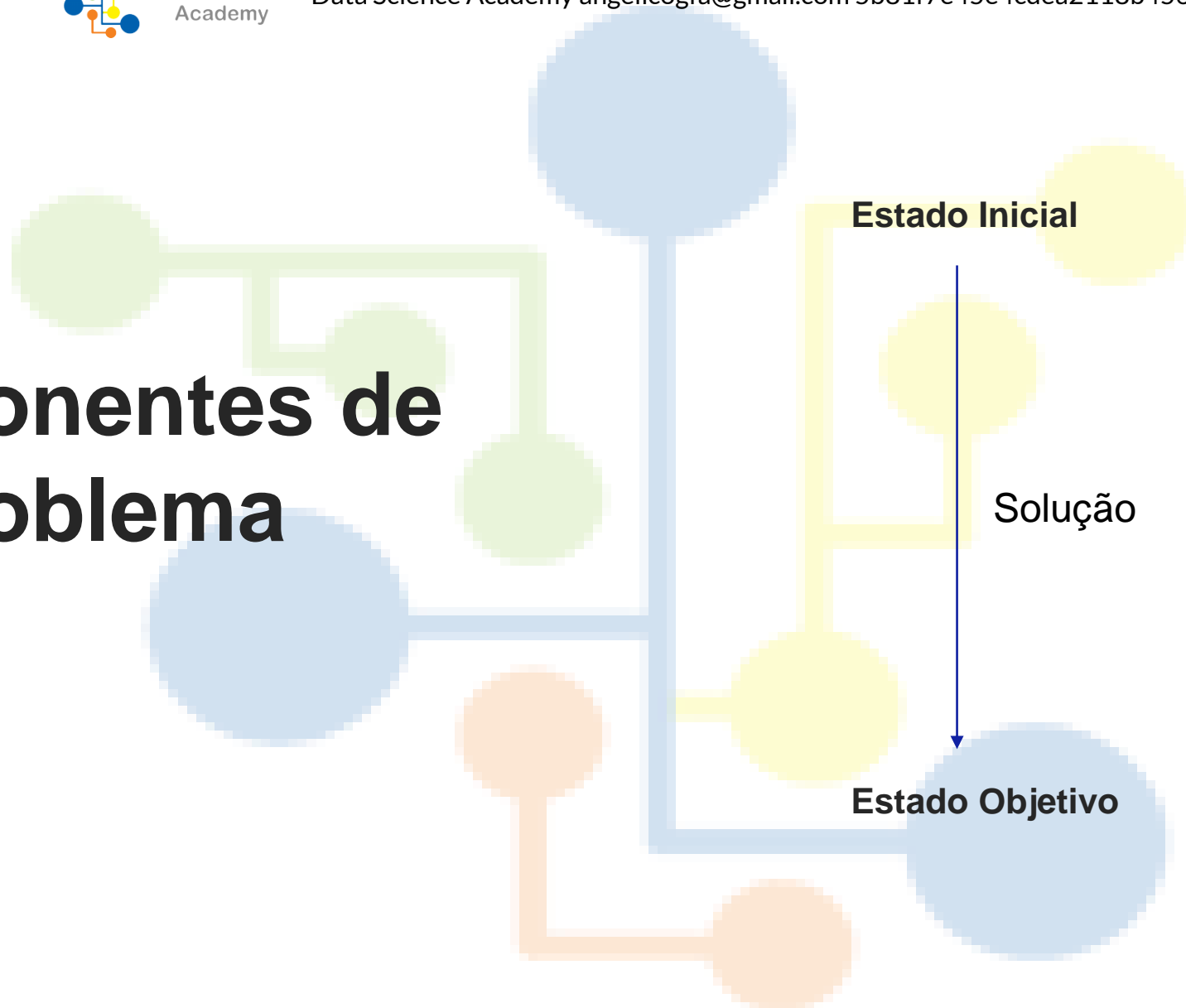


# Componentes de um Problema





# Componentes de um Problema







Mas isso foi exatamente o que estudamos no curso de Machine Learning?

Sim, exatamente. Acabamos de descrever como funciona um algoritmo de Machine Learning, que é uma forma de Inteligência Artificial. Mas no curso de Machine Learning fizemos isso usando linguagens R e Python!

Ao longo de toda a Formação IA criaremos muitos modelos usando esta mesma representação e no próximo capítulo, aqui mesmo deste curso, teremos uma atividade prática usando linguagem Python!



**Estado Inicial**



**Ações**



**Modelo de Transição**



**Teste de Objetivo**



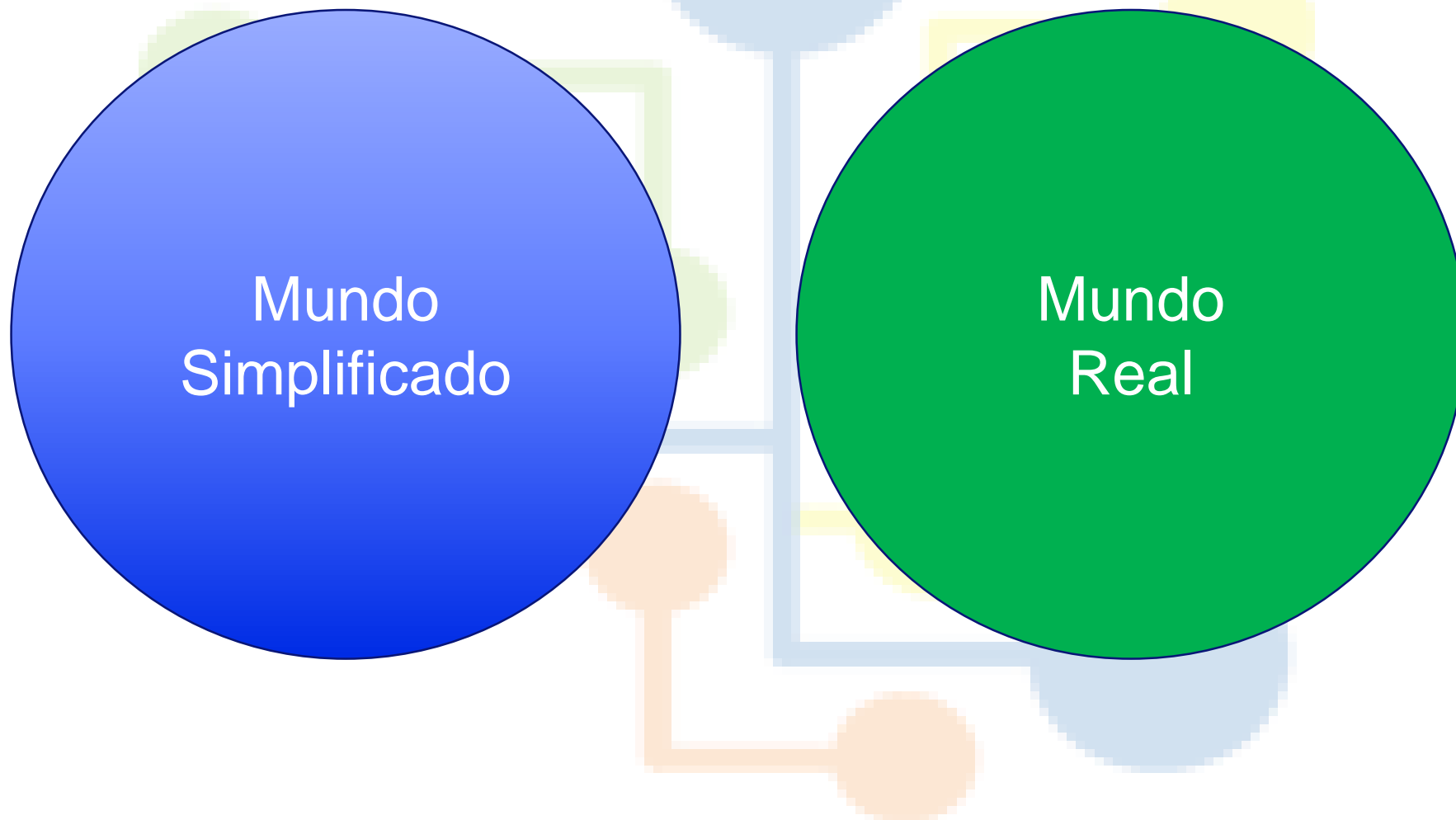
**Custo do Caminho**





# Formulação de Problemas

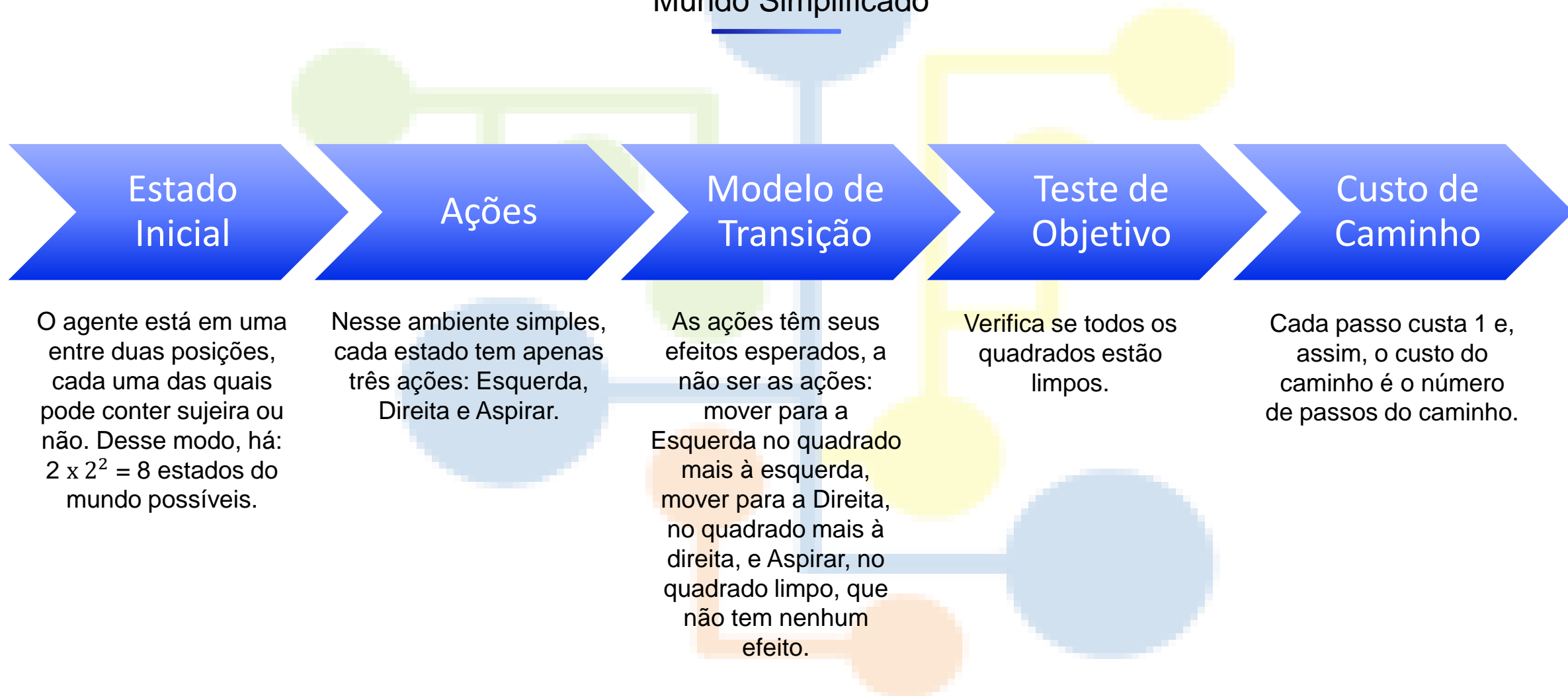






# Agente – Aspirador de Pó

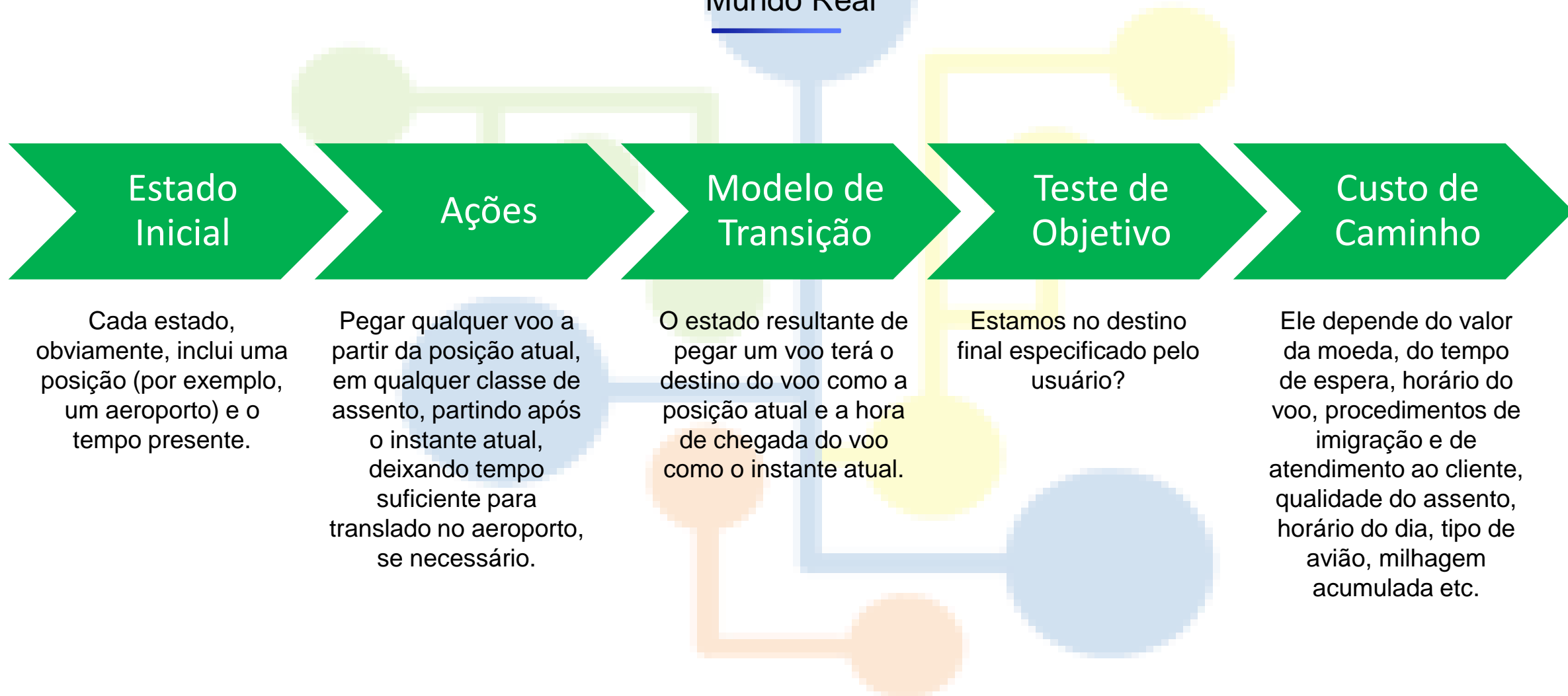
Mundo Simplificado





# Agente – Sistema de Informações para Viagens

Mundo Real





# Formulação de Problemas

- Problema do Caixeiro Viajante (melhor rota a ser escolhida)
- Layout de Circuitos Eletrônicos
- Navegação de Robôs
- Sequência Automática de Montagem

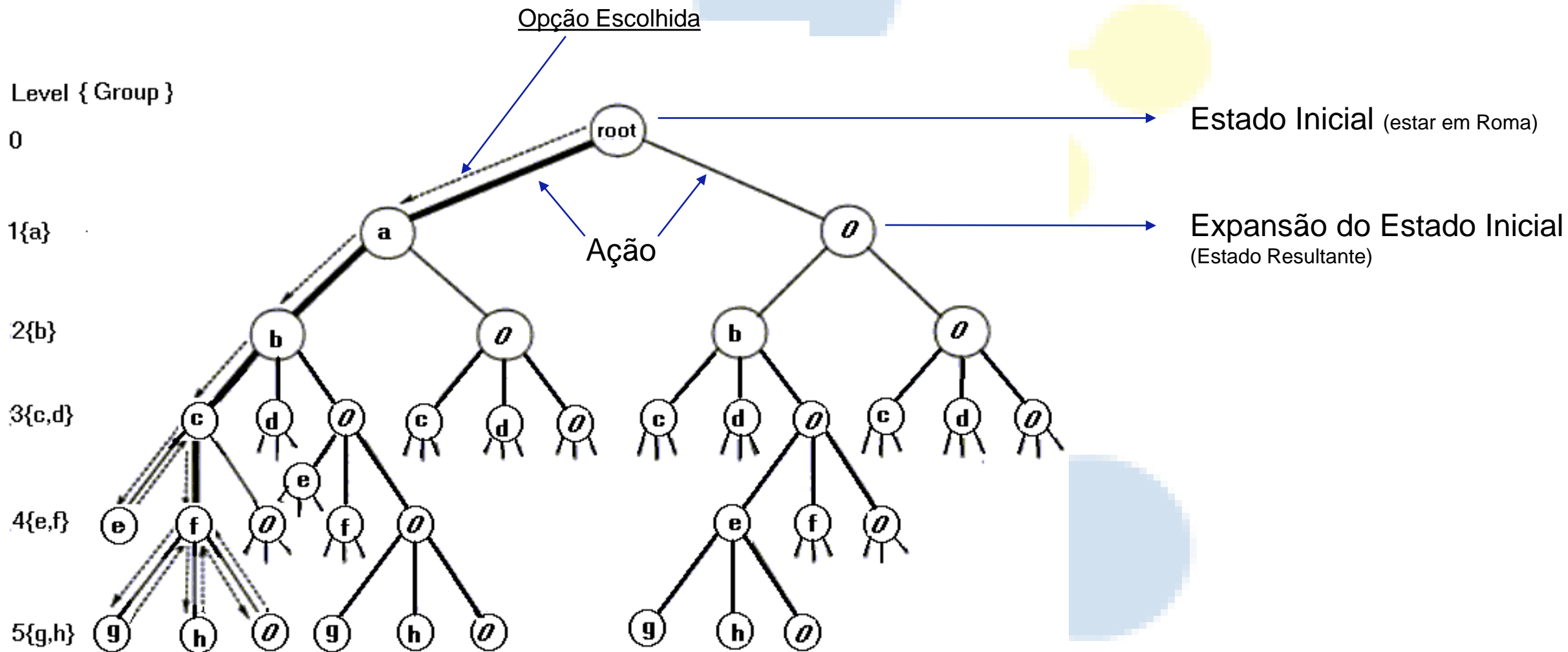






# Algoritmos de Busca







# Árvore de Busca

**função BUSCA-EM-ÁRVORE(*problema*) retorna** uma solução ou falha  
inicializar a borda utilizando o estado inicial do *problema*  
**repita**  
    **se** *borda vazia* **então retornar** falha  
    escolher um nó folha e o remover da borda  
    **se** o nó contém um estado objetivo **então retornar** a solução correspondente  
    expandir o nó escolhido, adicionando os nós resultantes à borda

**função BUSCA-EM-GRAFO(*problema*) retorna** uma solução ou falha  
inicializar a borda utilizando o estado inicial do *problema*  
**inicializar o conjunto explorado tornando-o vazio**  
**repita**  
    **se** *borda vazia* **então retornar** falha  
    escolher um nó folha e o remover da borda  
    **se** o nó contém um estado objetivo **então retornar** a solução correspondente  
    **adicionar o nó ao conjunto explorado**  
    expandir o nó escolhido, adicionando os nós resultantes à borda  
    **apenas se não estiver na borda ou no conjunto explorado**





# Árvore de Busca

- Nó ESTADO: o estado no espaço de estados a que o nó corresponde
- Nó PAI: o nó na árvore de busca que gerou esse nó
- AÇÃO: a ação que foi aplicada ao pai para gerar o nó
- CUSTO-DO-CAMINHO: o custo do caminho do estado inicial até o nó

**função** NÓ-FILHO(*problema*, *pai*, *ação*) **retorna** um nó

**retorna** um nó com

$ESTADO = problema.RESULTADO(pai.ESTADO, ação),$

$PAI = pai, AÇÃO = ação,$

$CUSTO-DE-CAMINHO = pai.CUSTO-DE-CAMINHO + problema.CUSTO-DO-PASSO(pai.ESTADO, ação)$







# Árvore de Busca

**função** NÓ-FILHO(*problema*, *pai*, *ação*) **retorna** um nó

**retorna** um nó com

$ESTADO = problema.RESULTADO(pai.ESTADO, ação),$

$PAI = pai, AÇÃO = ação,$

$CUSTO-DE-CAMINHO = pai.CUSTO-DE-CAMINHO + problema.CUSTO-DO-PASSO(pai.ESTADO, ação)$





# Árvore de Busca

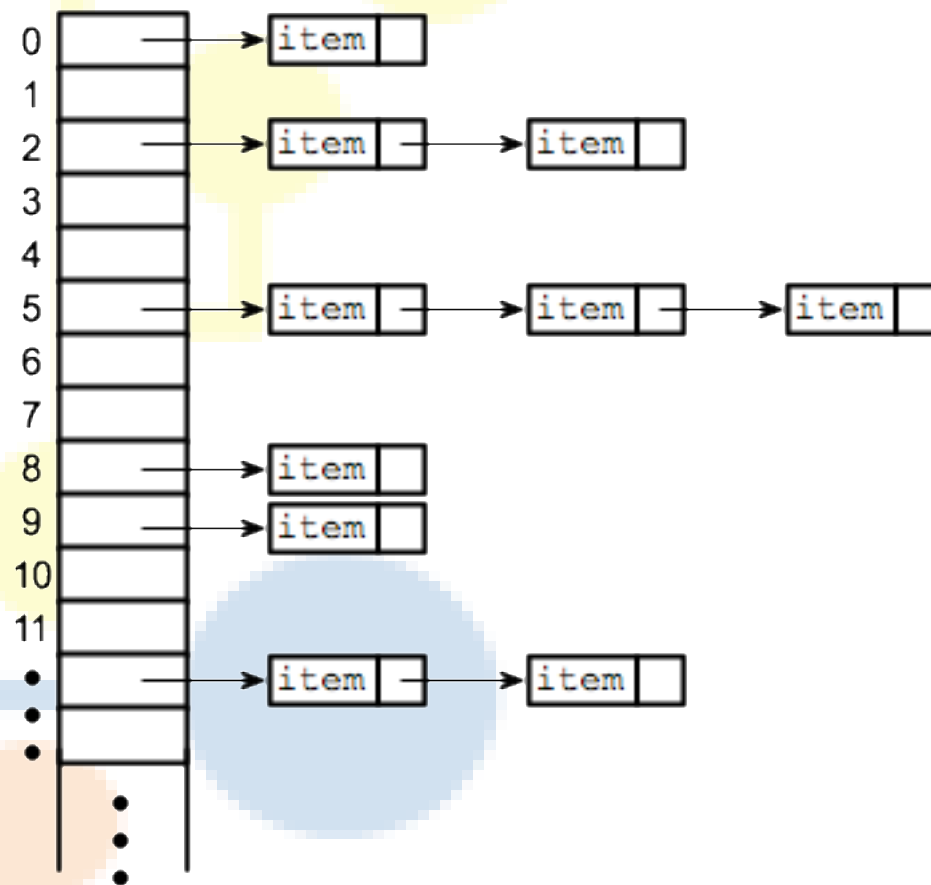




# Árvore de Busca

Deve-se ter o cuidado de implementar a tabela hash com a correta noção de igualdade entre estados.

Por exemplo, no problema do caixeiro-viajante, é necessário que a tabela hash saiba que o conjunto de cidades visitadas {Roma, Zurique, Paris} é o mesmo que {Paris, Zurique, Roma}.





# Estratégias de Busca Sem Informação

---







# Estratégias de Busca Sem Informação

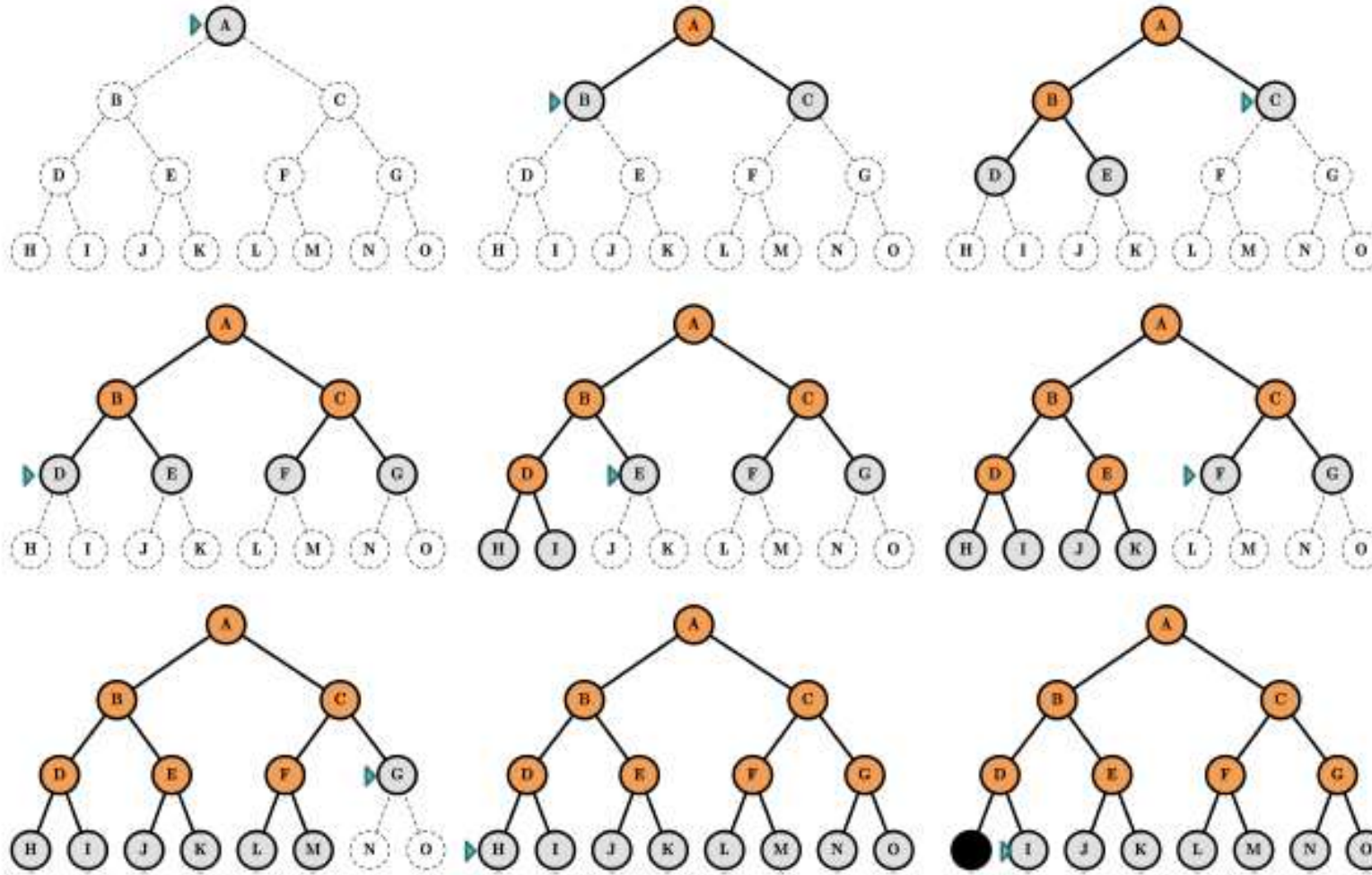
---

1. **Busca em Largura** - Breadth-First search (BFS)
2. **Busca em Profundidade** - Depth-First search (DFS)
3. **Busca em Profundidade Limitada** - Depth-limited search (DLS)
4. **Busca de Aprofundamento Iterativo** - Iterative-deepening search (IDS)
5. **Busca de Custo Uniforme** - Uniform-cost search (UCS)





# Busca em Largura (Breadth-First search (BFS))





# Busca em Largura (Breadth-First search (BFS))

```
function BREADTH-FIRST-SEARCH(initialState, goalTest)  
  returns SUCCESS or FAILURE :
```

```
  frontier = Queue.new(initialState)  
  explored = Set.new()
```

```
  while not frontier.isEmpty():  
    state = frontier.dequeue()  
    explored.add(state)
```

```
    if goalTest(state):  
      return SUCCESS(state)
```

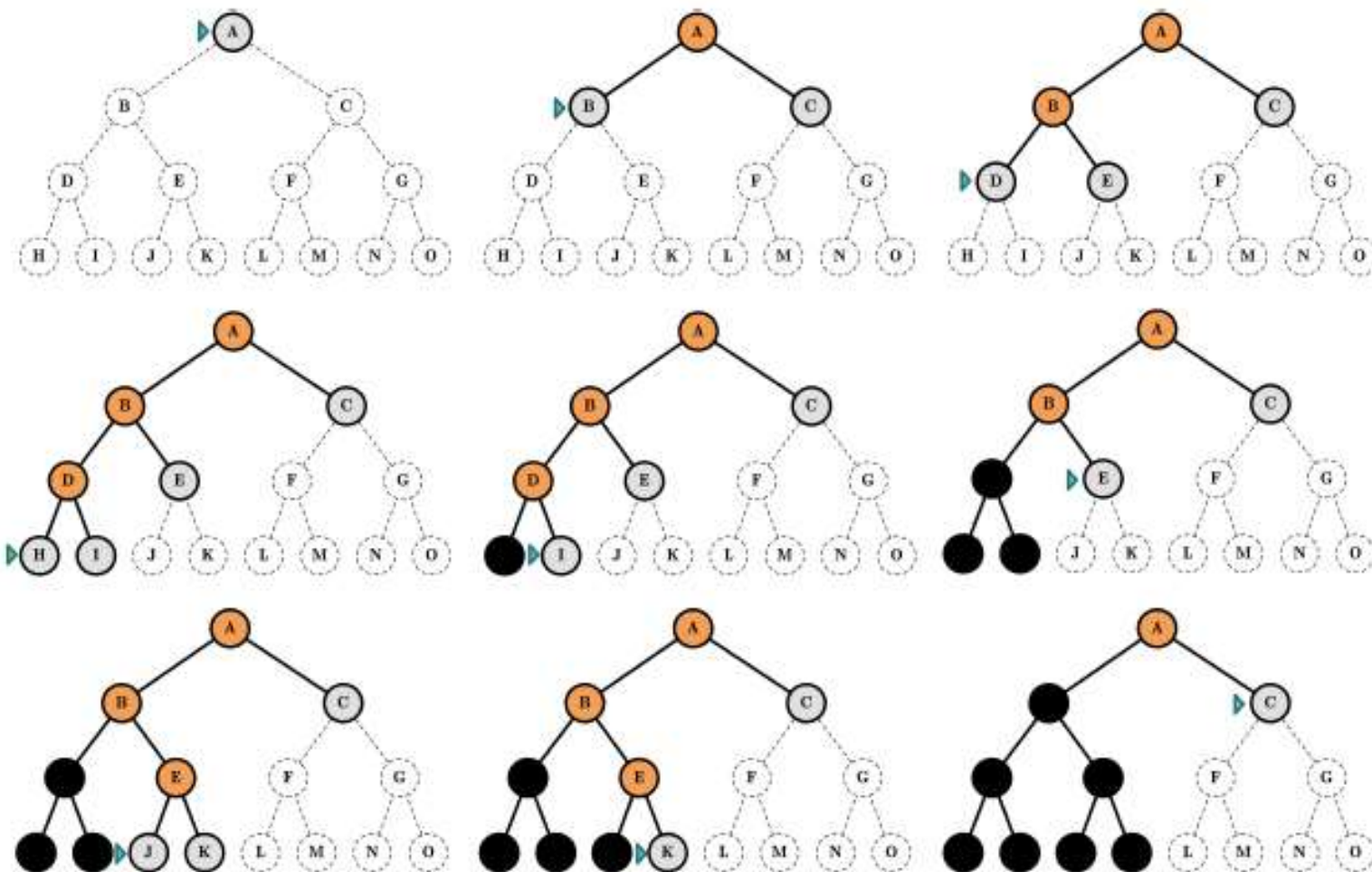
```
    for neighbor in state.neighbors():  
      if neighbor not in frontier  $\cup$  explored:  
        frontier.enqueue(neighbor)
```

```
  return FAILURE
```





# Busca em Profundidade (Depth-First search (DFS))







# Busca em Profundidade (Depth-First search (DFS))

```
function DEPTH-FIRST-SEARCH(initialState, goalTest)  
    returns SUCCESS or FAILURE :
```

```
    frontier = Stack.new(initialState)  
    explored = Set.new()
```

```
    while not frontier.isEmpty():  
        state = frontier.pop()  
        explored.add(state)
```

```
        if goalTest(state):  
            return SUCCESS(state)
```

```
        for neighbor in state.neighbors():  
            if neighbor not in frontier  $\cup$  explored:  
                frontier.push(neighbor)
```

```
    return FAILURE
```





## Busca em Profundidade Limitada (Depth-limited search (DLS))

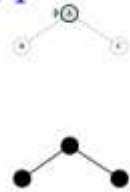
A premissa é: se tivermos algum conhecimento sobre o problema, não precisamos ir tão fundo na busca!



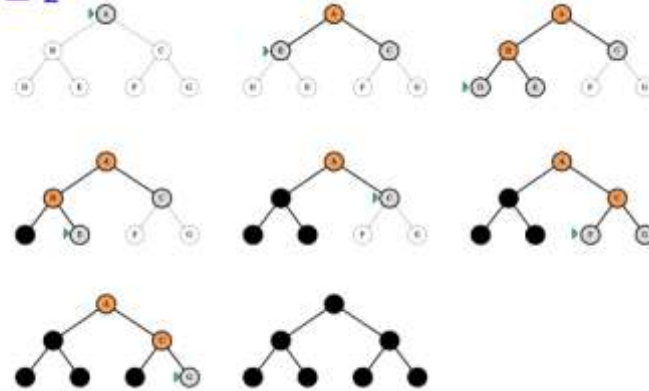


# Busca de Aprofundamento Iterativo (Iterative-deepening search (IDS))

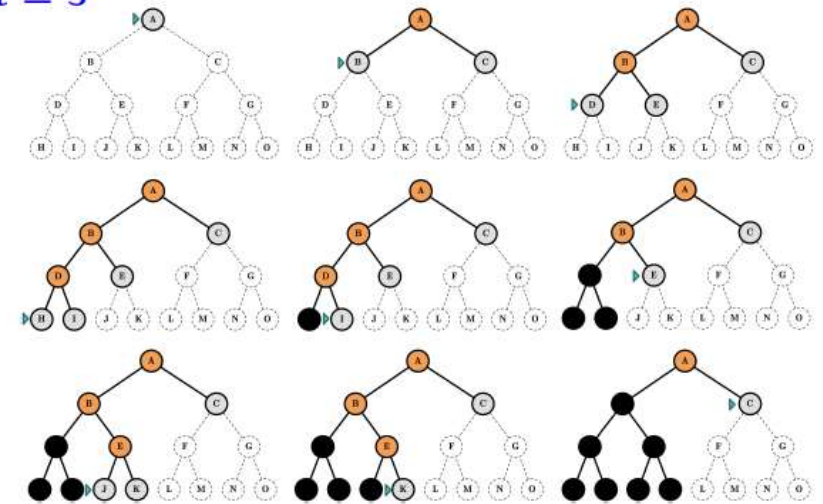
Limit = 1



Limit = 2



Limit = 3





# Busca de Aprofundamento Iterativo (Iterative-deepening search (IDS))

**função** BUSCA-DE-APROFUNDAMENTO-ITERATIVO(*problema*) **retorna** uma solução ou falha  
**para** profundidade = 0 até  $\infty$  **faça**  
  *resultado*  $\leftarrow$  BUSCA-EM-PROFUNDIDADE-LIMITADA(*problema*, *profundidade*)  
**se** *resultado*  $\neq$  corte **então retornar** *resultado*







# Busca de de Custo Uniforme (Uniform-cost search (UCS))

```
function UNIFORM-COST-SEARCH(initialState, goalTest)
  returns SUCCESS or FAILURE : /* Cost  $f(n) = g(n)$  */

  frontier = Heap.new(initialState)
  explored = Set.new()

  while not frontier.isEmpty():
    state = frontier.deleteMin()
    explored.add(state)

    if goalTest(state):
      return SUCCESS(state)

    for neighbor in state.neighbors():
      if neighbor not in frontier  $\cup$  explored:
        frontier.insert(neighbor)
      else if neighbor in frontier:
        frontier.decreaseKey(neighbor)

  return FAILURE
```





# Estratégias de Busca Informada

---





# Estratégias de Busca Informada

---

Uma estratégia de **busca informada** é a que utiliza conhecimento de um problema específico, além da definição do problema em si, e pode encontrar soluções de forma mais eficiente do que uma estratégia de busca sem informação





# Estratégias de Busca Informada



**função BUSCA-EM-ÁRVORE(*problema*) retorna** uma solução ou falha  
inicializar a borda utilizando o estado inicial do *problema*  
**repita**  
    **se** *borda vazia* **então retornar** falha  
    escolher um nó folha e o remover da borda  
    **se** o nó contém um estado objetivo **então retornar** a solução correspondente  
    expandir o nó escolhido, adicionando os nós resultantes à borda

**função BUSCA-EM-GRAFO(*problema*) retorna** uma solução ou falha  
inicializar a borda utilizando o estado inicial do *problema*  
**inicializar o conjunto explorado tornando-o vazio**  
**repita**  
    **se** *borda vazia* **então retornar** falha  
    escolher um nó folha e o remover da borda  
    **se** o nó contiver um estado objetivo **então retornar** a solução correspondente  
    **adicionar o nó ao conjunto explorado**  
    expandir o nó escolhido, adicionando os nós resultantes à borda  
    **apenas se não estiver na borda ou no conjunto explorado**





# Estratégias de Busca Informada

$h(n)$  = custo estimado do caminho de menor custo do estado do nó  $n$  para um estado objetivo.

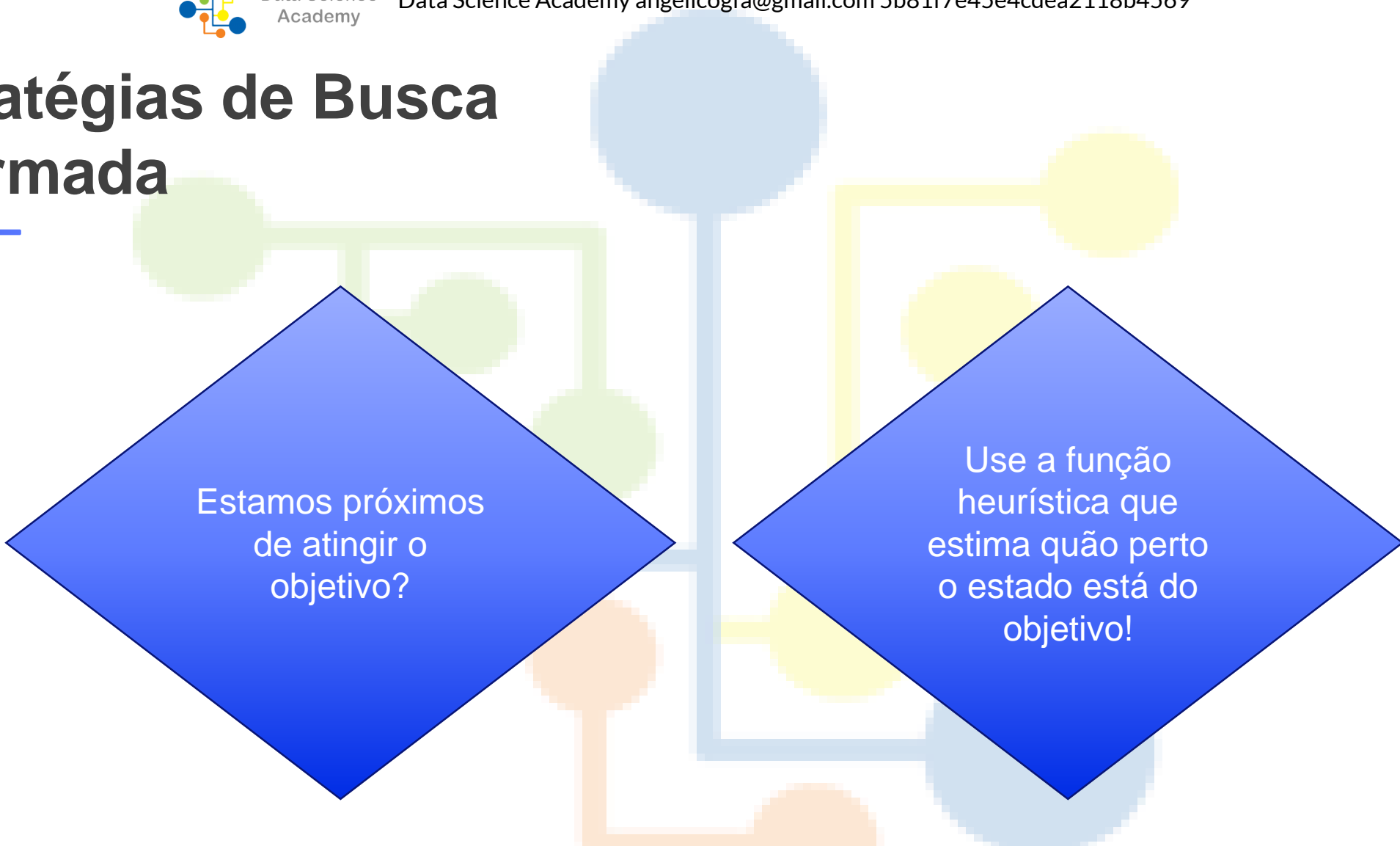
Consideramos as heurísticas como funções arbitrárias, não negativas, de problemas específicos, com uma restrição: se  $n$  for um nó objetivo, então  $h(n) = 0$







# Estratégias de Busca Informada



Estamos próximos  
de atingir o  
objetivo?

Use a função  
heurística que  
estima quão perto  
o estado está do  
objetivo!

A heurística não precisa ser perfeita, apenas uma estimativa!





# Estratégias de Busca Informada

---

- 1- **Busca Gulosa de Melhor Escolha** (Greedy Best-First Search)
- 2- **Busca A\*** (A\* Search)

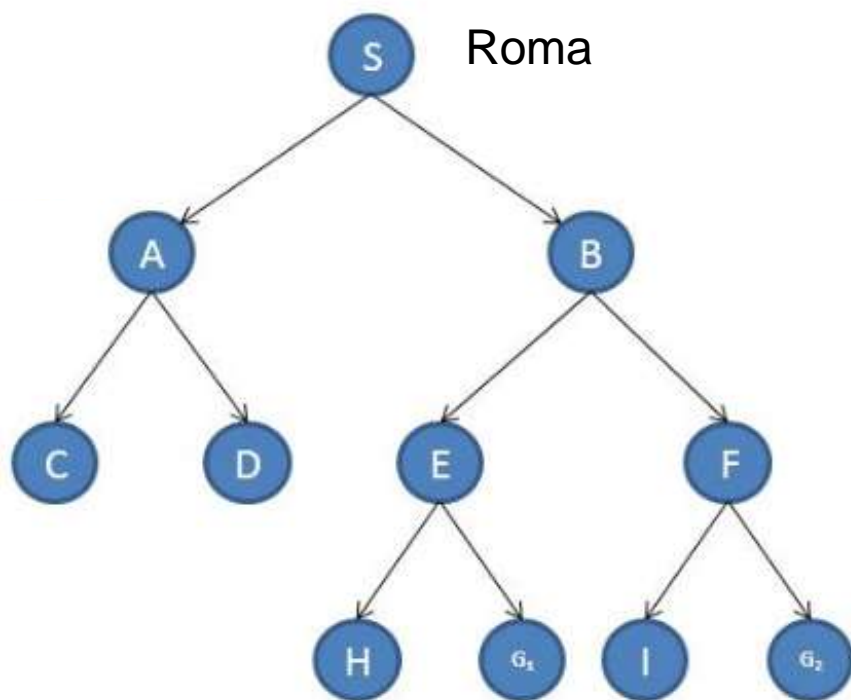




# Busca Gulosa de Melhor Escolha (Greedy Best-First Search)

Greedy Search utiliza uma heurística estimada  $h(n)$

S → Estado inicial  
G1, G2 → Objetivo



| Node | $h(n)$ |
|------|--------|
| A    | 11     |
| B    | 5      |
| C    | 9      |
| D    | 8      |
| E    | 4      |
| F    | 2      |
| H    | 7      |
| i    | 3      |

Neste exemplo,  $h(n)$  pode representar as distâncias entre os nós, ou seja, a distância entre as cidades





# Busca Gulosa de Melhor Escolha (Greedy Best-First Search)

```
function GREEDY-BEST-FIRST-SEARCH(initialState, goalTest)
  returns SUCCESS or FAILURE : /* Cost  $f(n) = h(n)$  */
```

```
  frontier = Heap.new(initialState)
  explored = Set.new()
```

```
  while not frontier.isEmpty():
    state = frontier.deleteMin()
    explored.add(state)
```

```
    if goalTest(state):
      return SUCCESS(state)
```

```
    for neighbor in state.neighbors():
      if neighbor not in frontier  $\cup$  explored:
        frontier.insert(neighbor)
      else if neighbor in frontier:
        frontier.decreaseKey(neighbor)
```

```
  return FAILURE
```



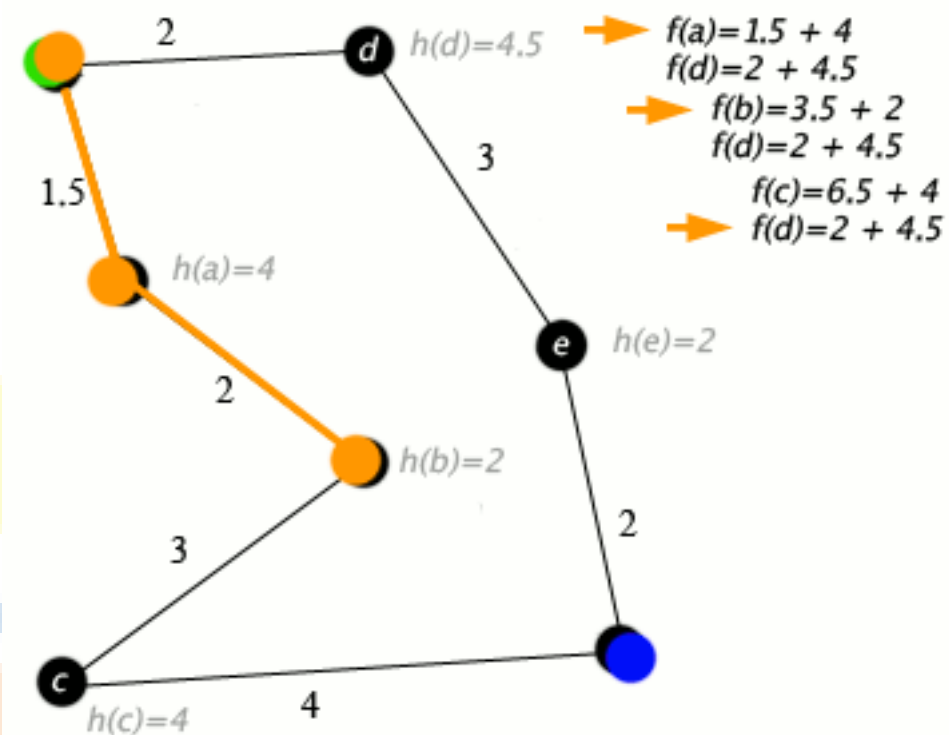


# Busca A\* (A\* Search)

Ela avalia os nós através da combinação de  $g(n)$ , o custo para alcançar o nó, e  $h(n)$ , o custo para ir do nó ao objetivo:  **$f(n) = g(n) + h(n)$** .

Uma vez que  $g(n)$  dá o custo do caminho desde o nó inicial até o nó  $n$  e  $h(n)$  é o custo estimado do caminho de menor custo de  $n$  até o objetivo, teremos:

**$f(n)$  = custo estimado da solução de menor custo através de  $n$ .**







# Busca A\* (A\* Search)

```
function A-STAR-SEARCH(initialState, goalTest)
  returns SUCCESS or FAILURE : /* Cost  $f(n) = g(n) + h(n)$  */

  frontier = Heap.new(initialState)
  explored = Set.new()

  while not frontier.isEmpty():
    state = frontier.deleteMin()
    explored.add(state)

    if goalTest(state):
      return SUCCESS(state)

    for neighbor in state.neighbors():
      if neighbor not in frontier  $\cup$  explored:
        frontier.insert(neighbor)
      else if neighbor in frontier:
        frontier.decreaseKey(neighbor)

  return FAILURE
```





# Heurística

A primeira condição requerida em algoritmos de busca informada, é que  $h(n)$  seja uma heurística admissível. Uma heurística admissível é a que nunca superestima o custo de atingir o objetivo.

Devido à  $g(n)$  ser o custo real para atingir  $n$  ao longo do caminho atual, e  $f(n) = g(n) + h(n)$ , temos como consequência imediata que  $f(n)$  nunca irá superestimar o verdadeiro custo de uma solução ao longo do caminho atual através de  $n$ .





Data Science  
Academy

Data Science Academy angelicogfa@gmail.com 5b81f7e45e4cdea2118b4569

# Aprendizagem Para Melhorar a Busca

---



Data Science Academy

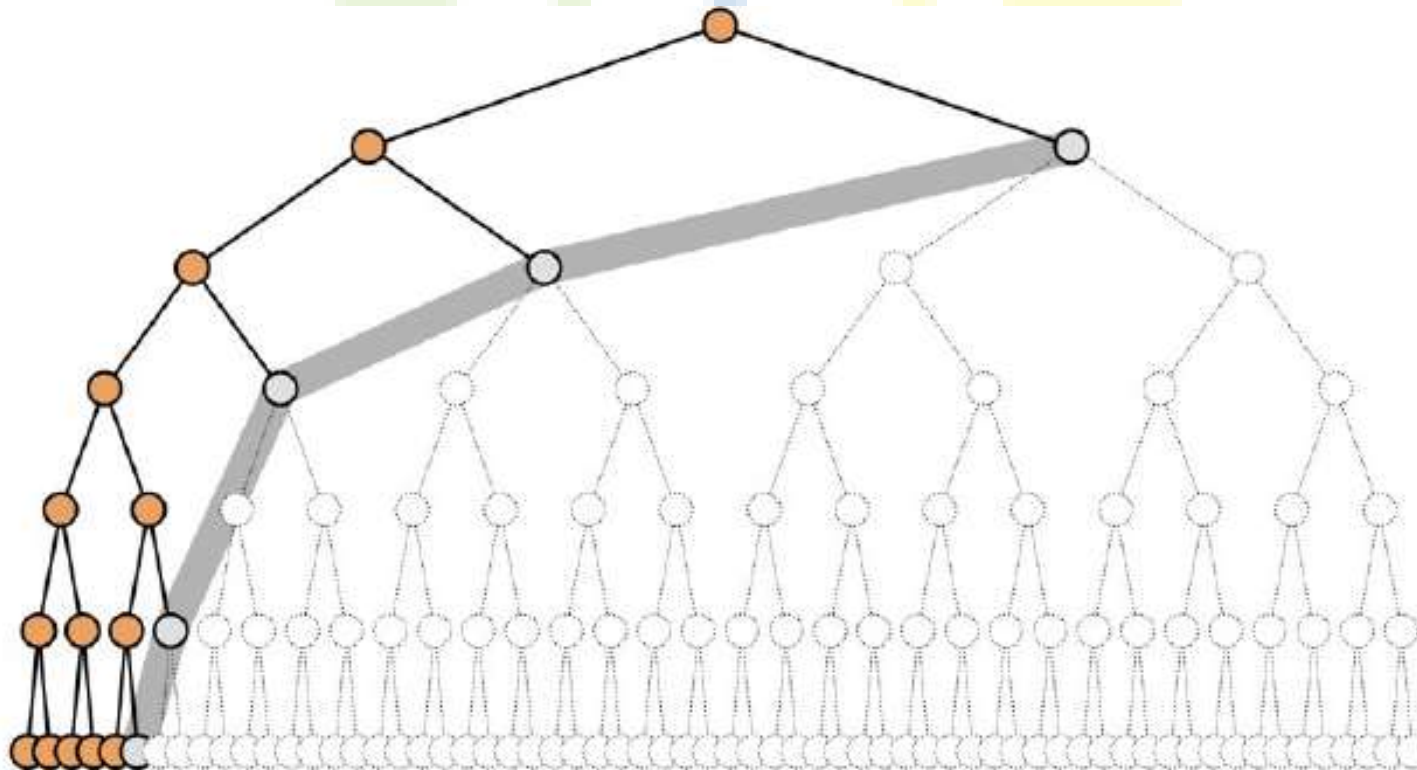


# Pode um agente aprender para melhorar a busca?

**SIM**



# Pode um agente aprender para melhorar a busca?







# Pode um agente aprender para melhorar a busca?

---

O objetivo da aprendizagem é minimizar o custo total da solução do problema, fazendo um compromisso entre o custo computacional e o custo do caminho.





# Busca Local e Problemas de Otimização





# Busca Local e Problemas de Otimização

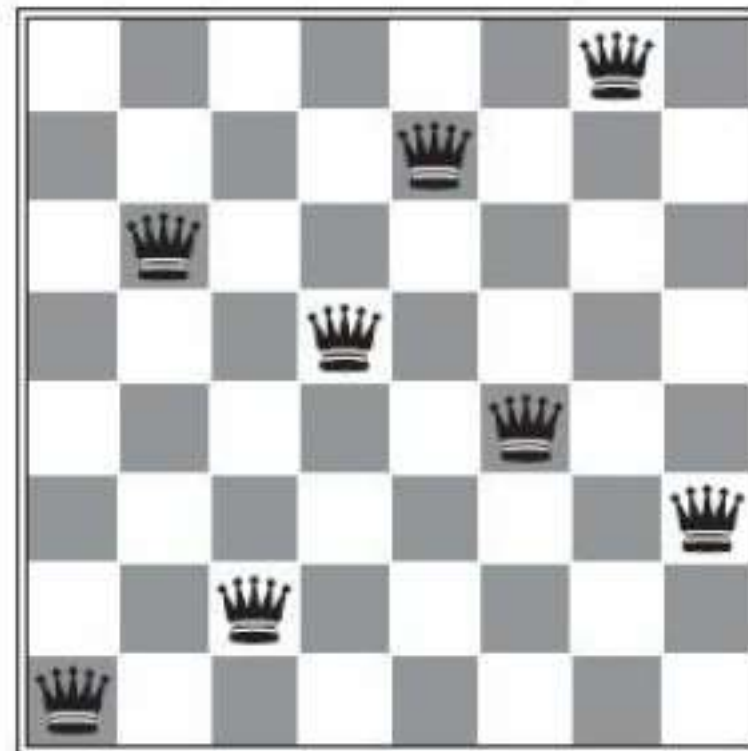
Nos algoritmos que estudamos anteriormente, quando o objetivo é encontrado, o caminho faz parte da solução. Mas dependendo da aplicação, o caminho pode não ser relevante. E quando o caminho não é relevante, os algoritmos de busca sistemática não são indicados, sendo necessário outra categoria de algoritmos.





# Busca Local e Problemas de Otimização

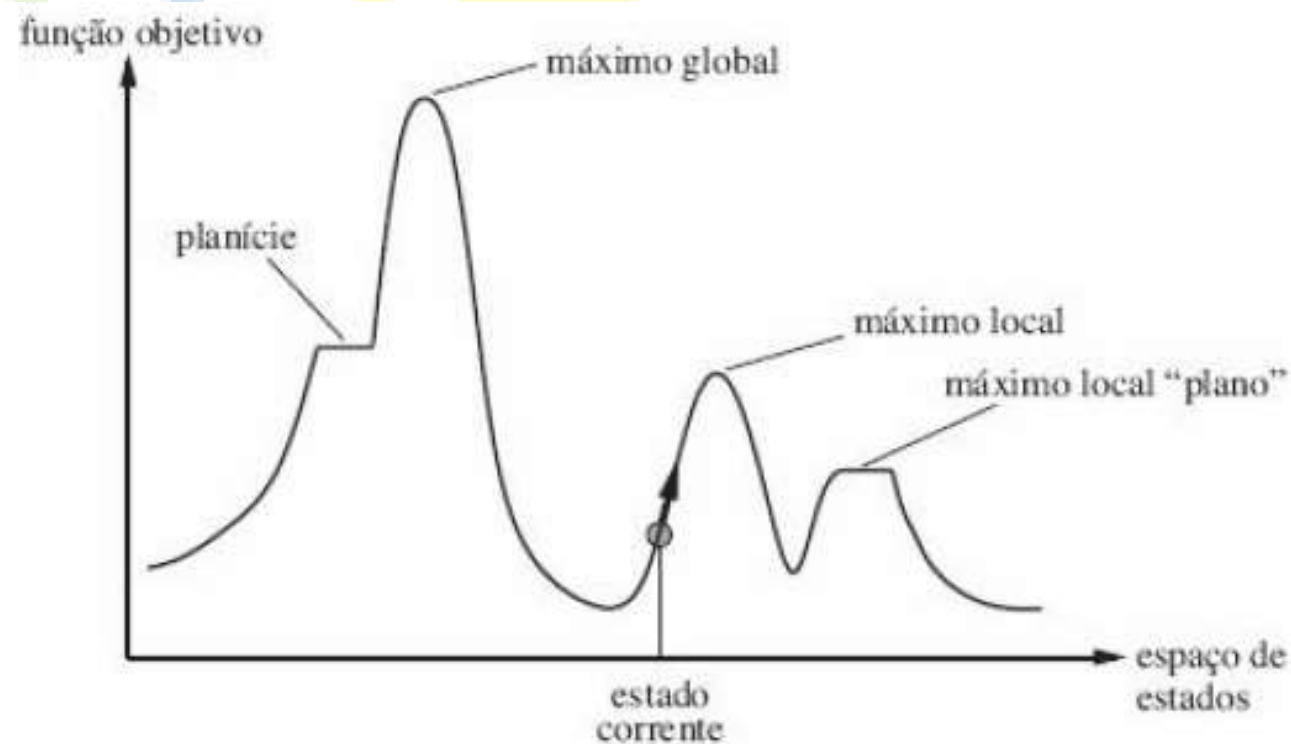
|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 18 | 12 | 14 | 13 | 13 | 12 | 14 | 14 |
| 14 | 16 | 13 | 15 | 12 | 14 | 12 | 16 |
| 14 | 12 | 18 | 13 | 15 | 12 | 14 | 14 |
| 15 | 14 | 14 | ♔  | 13 | 16 | 13 | 16 |
| ♔  | 14 | 17 | 15 | ♔  | 14 | 16 | 16 |
| 17 | ♔  | 16 | 18 | 15 | ♔  | 15 | ♔  |
| 18 | 14 | ♔  | 15 | 15 | 14 | ♔  | 16 |
| 14 | 14 | 13 | 17 | 12 | 14 | 12 | 18 |





# Busca Local e Problemas de Otimização

Os algoritmos de **busca local** operam usando um único **estado atual** (em vez de vários caminhos) e, em geral, se movem apenas para os vizinhos desse estado

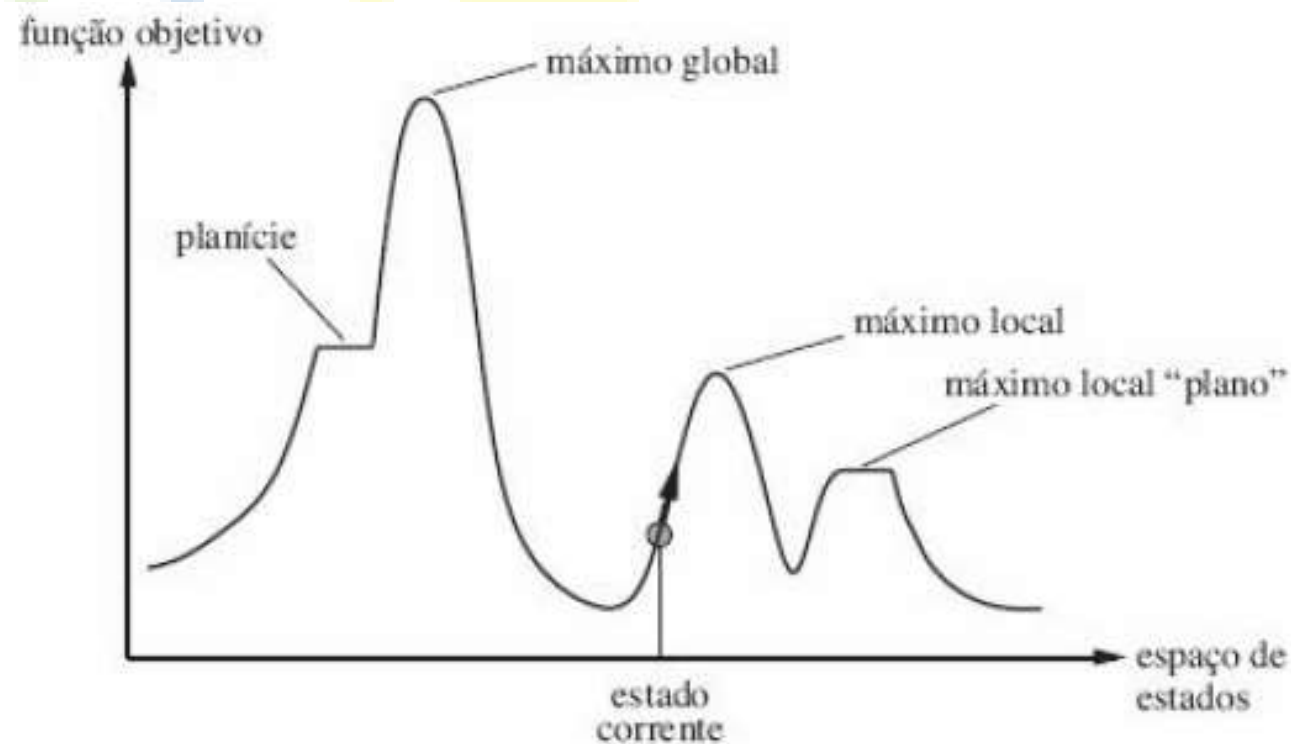






# Busca Local e Problemas de Otimização

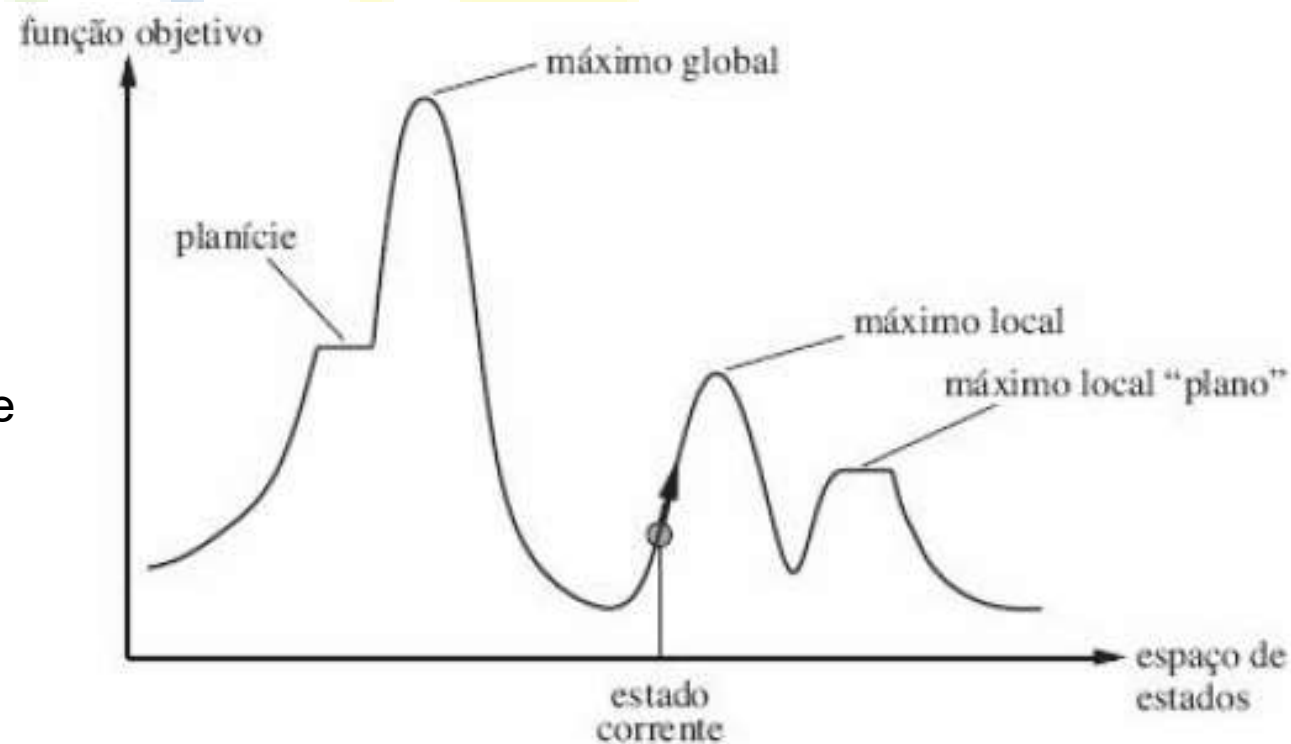
Além de encontrar objetivos, os algoritmos de busca local são úteis para resolver **problemas de otimização**, nos quais o objetivo é encontrar o melhor estado de acordo com uma **função objetivo**





# Busca Local e Problemas de Otimização

Uma topologia tem ao mesmo tempo “posição” (definida pelo estado) e “elevação” (definida pelo valor da função de custo da heurística ou da função objetivo).





# Busca Local e Problemas de Otimização

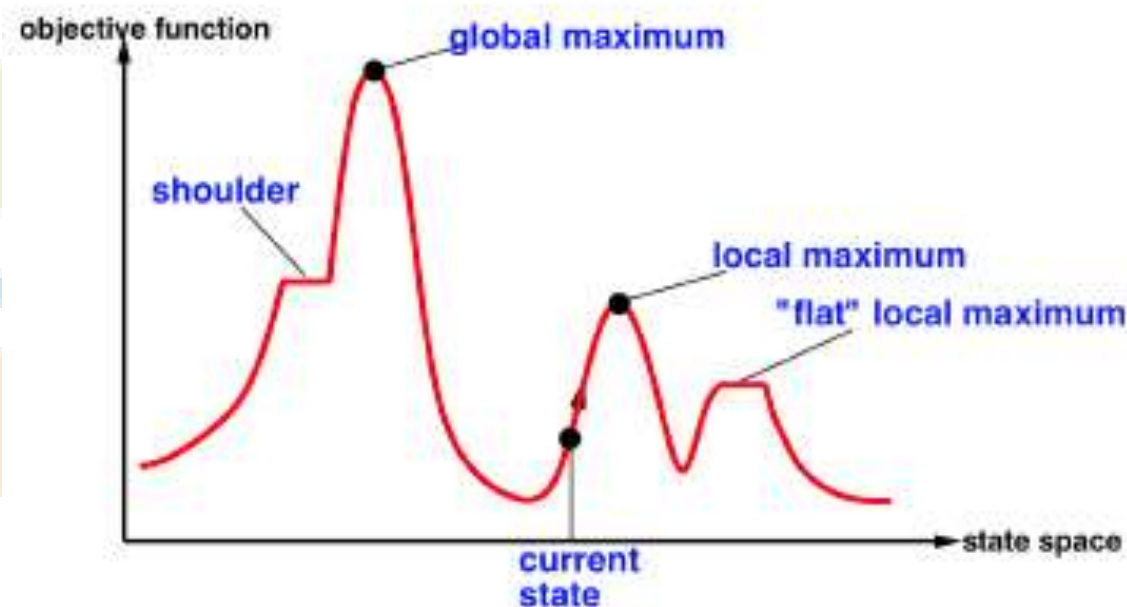
- 1. Busca de Subida de Encosta** (Hill Climbing)
- 2. Têmpera Simulada** (Simulated Annealing)
- 3. Busca em Feixe Local** (Local Beam Search)
- 4. Algoritmos Genéticos** (Genetic Algorithms)





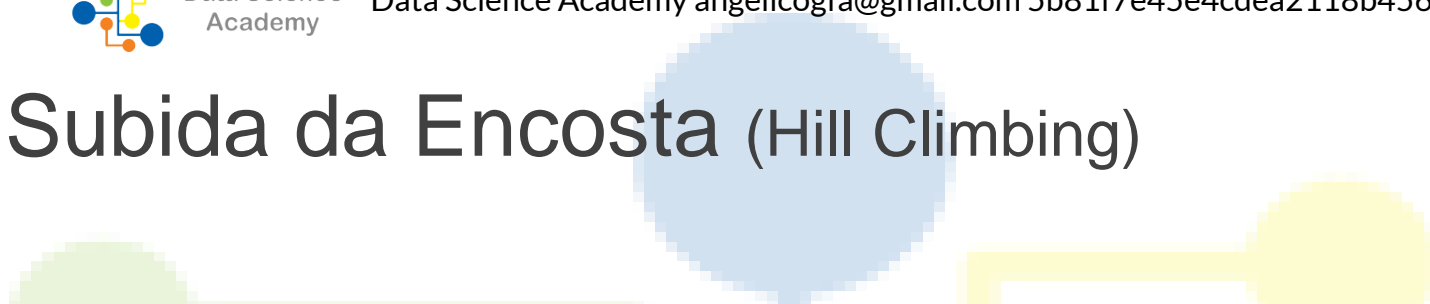
# Busca de Subida da Encosta (Hill Climbing)

O algoritmo termina quando alcança um “pico” em que nenhum vizinho tem valor mais alto





# Busca de Subida da Encosta (Hill Climbing)

A diagram illustrating the Hill Climbing algorithm. It shows a landscape with three hills: a small green hill on the left, a large blue hill in the center, and a yellow hill on the right. A yellow path leads from the base of the blue hill towards the yellow hill, representing the search process.

```
function HILL-CLIMBING(initialState)
    returns State that is a local maximum

    initialize current with initialState

    loop do
        neighbor = a highest-valued successor of current

        if neighbor.value  $\leq$  current.value:
            return current.state

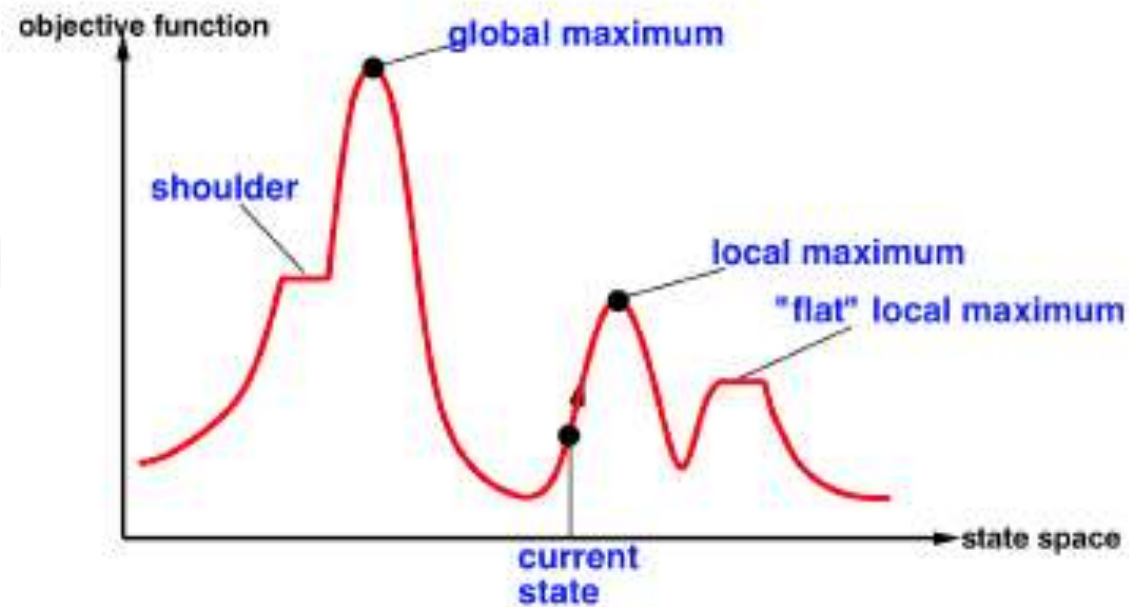
        current = neighbor
```





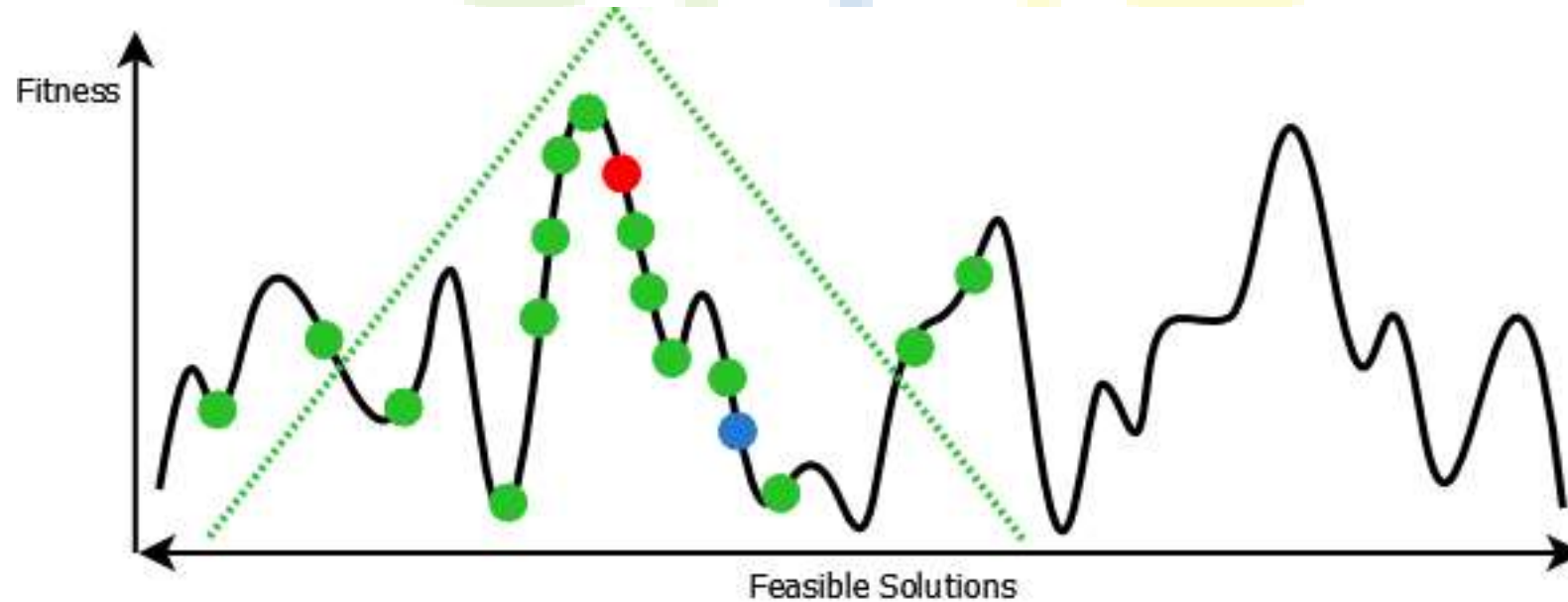


# Têmpera Simulada (Simulated Annealing)





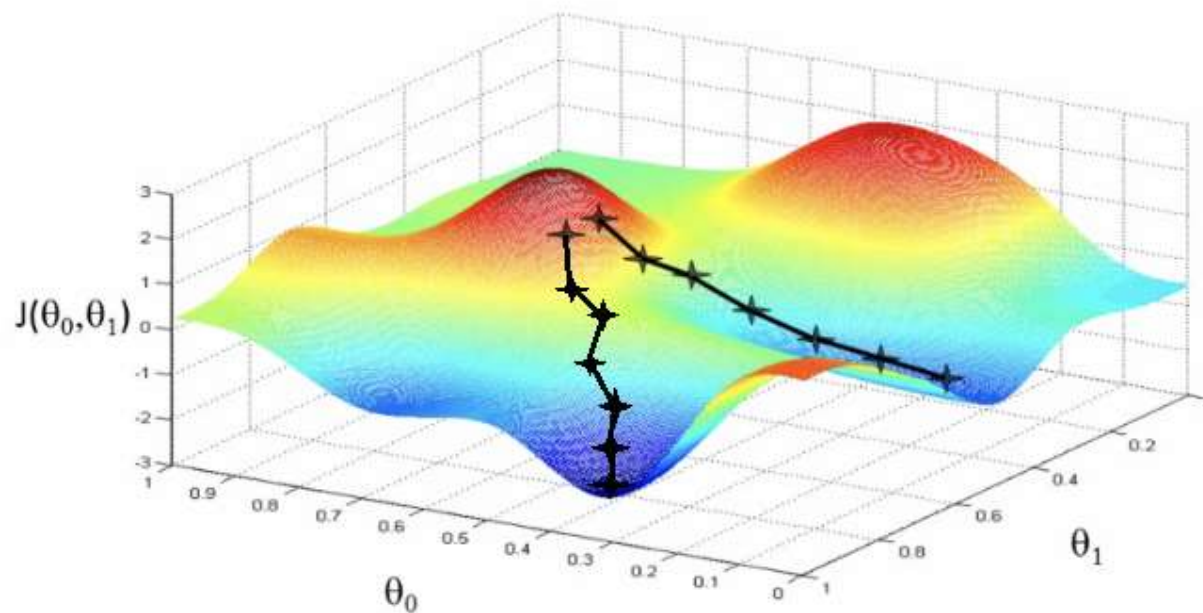
# Têmpera Simulada (Simulated Annealing)





# Têmpera Simulada (Simulated Annealing)

Descida do Gradiente  
(Minimização do Custo)





# Têmpera Simulada (Simulated Annealing)

**função** TÊMPERA-SIMULADA(*problema*, *escalonamento*) **retorna** um estado solução

**entradas:** *problema*, um problema

*escalonamento*, um mapeamento de tempo para “temperatura”

*atual*  $\leftarrow$  CRIAR-NÓ(*problema*.ESTADO-INICIAL)

**para**  $t = 1$  **até**  $\infty$  **faça**

$T \leftarrow$  *escalonamento*[ $t$ ]

**se**  $T = 0$  **então retornar** *corrente*

*próximo*  $\leftarrow$  um sucessor de *atual* selecionado aleatoriamente

$\Delta E \leftarrow$  *próximo*.VALOR – *atual*.VALOR

**se**  $\Delta E > 0$  **então** *atual*  $\leftarrow$  *próximo*

**senão** *atual*  $\leftarrow$  *próximo* somente com probabilidade  $e^{\Delta E/T}$





## Busca em Feixe Local (Local Beam Search)

O algoritmo de **busca em feixe local** mantém o controle de  $k$  estados, em vez de somente um. Ela começa com  $k$  estados gerados aleatoriamente. Em cada passo, são gerados todos os sucessores de todos os  $k$  estados. Se qualquer um deles for um objetivo, o algoritmo irá parar. Caso contrário, ele selecionará os  $k$  melhores sucessores a partir da lista completa e repetirá o procedimento.







Data Science  
Academy

Data Science Academy angelicogfa@gmail.com 5b81f7e45e4cdea2118b4569

# Obrigado



Data Science Academy



Data Science Academy