

Primeiro trabalho de Inteligência Computacional

Avaliação Experimental de Agentes Inteligentes

Estudante: Luis Gustavo Barbosa Benevides, luis.benevides@aluno.uece.br

Professor: Gustavo Campos

1 Introdução

Primeiro, notemos que o ambiente é **determinístico**, pois dado o atual estado do ambiente e a ação do agente, o próximo estado do ambiente é completamente determinado. Ele também é **parcialmente observável**, pois os sensores do agente não são capazes de, a cada momento, lhe dar acesso ao estado completo do ambiente (mais especificamente, o agente só consegue perceber se o quadrado em que ele se encontra está limpo, e se tem obstáculos nos quadrados vizinhos). O ambiente é também de um único agente, estático, sequencial, discreto e conhecido.

O [diretório do projeto](#) é composto principalmente por 3 arquivos:

- **ambiente.py:** Possui a classe Ambiente, onde podemos configurar a probabilidade de encontrar sujeira ou obstáculo em um quadrado, gerar o ambiente e adicionar um (e somente um) agente ao ambiente. Nela também estão implementados os percepts (no método percept) gerados pelo ambiente, assim como as mudanças de estado sofridas por ele (método sofrer_acao). Para poder parar a simulação, foi implementado também um método para verificar que o ambiente já foi limpo (já que o agente não tem essa informação).
- **agentes.py:** Possui as classes Agente (superclasse), AgenteReativoSimples e AgenteBaseadoEmModelo.
- **simulacao.py:** Onde é implementada uma interface para mostrar visualmente o agente (baseado em modelos) em ação, e realizada algumas simulações a fim de comparar o desempenho dos dois agentes.

Os dois tipos de agentes abordados não conhecem o padrão inicial de sujeira, nem a geografia (extensão, limites e obstáculos) do ambiente, já que o ambiente é parcialmente observável. Para deixar isso mais evidente na simulação, os quadrados cinzas representam os quadrados não visitados (i.e. os quadrados que o agente ainda não sabe que existem).

2 Ambiente

Agora, comentaremos sobre o arquivo ambiente.py. Temos uma matriz retangular, e no construtor, podemos escolher as taxas de sujeira e obstáculo em cada quadrado, que são probabilidades. Como cada quadrado pode estar limpo, sujo ou com obstáculo, temos que a taxa de quadrados limpos é $1 - (taxa_sujeira + taxa_obstaculo)$, onde $0 \leq taxa_obstaculo$, $taxa_sujeira \leq 1$.

Assim, quando vamos gerar o ambiente, para cada quadrado, sorteamos um valor v entre 0 e 1 e verificamos se ele está no intervalo $[0, taxa_sujeira)$, $[taxa_sujeira, taxa_sujeira + taxa_obstaculo)$ ou $[taxa_sujeira + taxa_obstaculo, 1]$. É claro, a probabilidade de v cair nesses intervalos é, respectivamente, $taxa_sujeira$, $taxa_sujeira$ e $1 - (taxa_sujeira + taxa_obstaculo)$.

```

1  import random
2  import matplotlib.pyplot as plt
3
4
5  class Ambiente:
6      def __init__(self, largura, altura, taxa_sujeira=0.3, taxa_obstaculo=0.1):
7          self.largura = largura
8          self.altura = altura
9          self.matriz = {} #a matriz é um dicionário com tuplas (x, y) como chaves
10             #e valores "LIMPO", "SUJO" ou "OBSTACULO"
11          self.agentes = []
12
13          for x in range(largura):
14              for y in range(altura):
15                  # v
16                  v = random.random()
17                  if v < taxa_sujeira:
18                      self.matriz[(x, y)] = "SUJO"
19                  elif v < taxa_sujeira + taxa_obstaculo:
20                      self.matriz[(x, y)] = "OBSTACULO"
21                  else:
22                      self.matriz[(x, y)] = "LIMPO"
23
24
25      def adicionar_agente(self, agente, posicao):
26          if self.matriz.get(posicao) != "OBSTACULO":
27              self.agentes.append(agente)
28              agente.posicao = posicao
29              agente.ambiente = self
30
31
32      def percept(self, posicao):
33          estado_local = self.matriz.get(posicao, None)
34          vizinhos = {}
35          movimentos = {"CIMA": (0, -1), "BAIXO": (0, 1),
36             "ESQUERDA": (-1, 0), "DIREITA": (1, 0)}
37          for direcao, (dx, dy) in movimentos.items():
38              viz = (posicao[0] + dx, posicao[1] + dy)
39
40              #se o quadrado vizinho está dentro dos limites do ambiente
41              if 0 <= viz[0] < self.largura and 0 <= viz[1] < self.altura:
42                  if self.matriz.get(viz) == "OBSTACULO":
43                      vizinhos[direcao] = "OBSTACULO"
44                  else:
45                      vizinhos[direcao] = "LIVRE"
46              else:
47                  vizinhos[direcao] = "PAREDE"
48          return estado_local, vizinhos

```

Listing 1: Recorte de ambiente.py

No método percept, a partir da posição atual do agente, obtemos o estado do quadrado em que ele se encontra, e também se temos obstáculos nos quadrados vizinhos à posição atual. Note que o agente só vai saber se um quadrado livre está limpo ou sujo, quando ele estiver nesse quadrado.

```

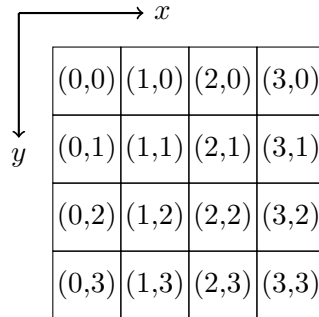
1  def sofrer_acao(self, agente, acao):
2      if acao == "LIMPAR":
3          if self.matriz[agente.posicao] == "SUJO":
4              self.matriz[agente.posicao] = "LIMPO"
5              return "LIMPOU"
6          else:
7              return "JA_LIMPO"      #já que o agente não estará na mesma posição que um obstáculo
8      elif acao in ["CIMA", "BAIXO", "ESQUERDA", "DIREITA"]:
9          dx, dy = {
10              "CIMA": (0, -1),
11              "BAIXO": (0, 1),
12              "ESQUERDA": (-1, 0),
13              "DIREITA": (1, 0)
14          }[acao]
15          nova_pos = (agente.posicao[0] + dx, agente.posicao[1] + dy)
16          if (0 <= nova_pos[0] < self.largura and
17              0 <= nova_pos[1] < self.altura and
18                  self.matriz[nova_pos] != "OBSTACULO"):
19              agente.posicao = nova_pos
20              return "MOVEU"
21          else:
22              return "BATEU"
23      return "INUTIL"
24
25  def esta_limpo(self):
26      return all(estado != "SUJO" for estado in self.matriz.values())
27

```

Listing 2: Recorte ambiente.py (continuação)

No método `sofrer_acao`, com a ação LIMPAR, o quadrado da posição que o agente está muda seu estado para LIMPO (ou continua nesse estado, se já estava limpo). Se a ação for CIMA, BAIXO, DIREITA ou ESQUERDA, a posição do agente é atualizada de acordo, contanto que nova posição não exista um obstáculo ou parede (ou seja, a posição ultrapassa os limites da matriz).

Observação: A ação BAIXO aumenta o y da posição em 1, ao invés de aumentar, pois a posição (0,0) é o quadrado superior esquerdo, e daí a x cresce para a direita (como esperado), mas o y cresce para baixo. É por isso que a ação CIMA diminui o y em 1. Um exemplo com as posições correspondentes a cada quadrado numa matriz de largura = altura = 4.



(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1)	(2,1)	(3,1)
(0,2)	(1,2)	(2,2)	(3,2)
(0,3)	(1,3)	(2,3)	(3,3)

Figure 1: Coordenadas quando a matriz é 4×4 .

3 Agentes

```
1  from ambiente import *
2
3  class Agente:
4      """Superclasse para todos os agentes"""
5
6      def __init__(self):
7          self.posicao = None
8          self.ambiente = None
9          self.pontuacao = 0
10
11     def agir(self):
12         raise NotImplementedError("Subclasses devem implementar este método.")
13
14
15 class AgenteReativoSimples(Agente):
16     """Agente que reage apenas ao estado atual do ambiente."""
17
18     def agir(self):
19         #tem uma percepção do ambiente a partir da posição atual
20         percept_atual, vizinhos = self.ambiente.percept(self.posicao)
21         if percept_atual == "SUJO":
22             if self.ambiente.sofrer_acao(self, "LIMPAR") == "LIMPOU":
23                 self.pontuacao += 1 # Se limpou, ganha ponto
24         else:
25             #vizinhos é um dicionário com direções ("CIMA", "BAIXO", "ESQUERDA", "DIREITA") como chaves
26             # e estados ("LIVRE", "OBSTACULO", "PAREDE") como valores
27             acoes_possiveis = [direcao for direcao, estado in vizinhos.items() if estado == "LIVRE"]
28             if acoes_possiveis:
29                 direcao = random.choice(acoes_possiveis) # escolhe uma ação au hasard
30                 self.ambiente.sofrer_acao(self, direcao)
31                 self.pontuacao -= 1 # Movimento custa ponto
32
```

Listing 3: Recorte agentes.py

A nível de código, cada classe de agente herda da superclasse 'Agente', sobrescrevendo o método 'agir', de acordo com o tipo de agente.

O agente reativo simples leva em conta apenas o percept atual e, através de regras condição-ação, faz a conexão entre o percept recebido e a ação que irá realizar. Nesse caso, o agente limpa o quadrado onde se encontra se ele estiver sujo. Ou, se o quadrado estiver limpo, o agente move para um dos vizinhos que estiver livre. Para tentar evitar o loop infinito, o agente escolhe randomicamente a direção em que vai se movimentar, dentre as possíveis opções.

Agora, no agente reativo baseado em modelo, o modelo do mundo assume que quando o agente limpa um quadrado sujo, este passa a ficar limpo, e ainda mais, não volta a ficar sujo. Ele assume também que o ambiente é estático, ou seja, o ambiente não sofre alterações enquanto o agente está deliberando a próxima ação. Ele considera também que os obstáculos são fixos.

Assim, na implementação feita, o agente memoriza os lugares que já visitou, para tomar a decisão de ir para um dos quadrados vizinhos livres e não visitados (sempre que houver essa opção). Quando, em um momento, todos os vizinhos estiverem sido visitados, o agente escolhe um ao acaso, assim como o agente reativo simples.

```

1  class AgenteBaseadoEmModelo(Agente):
2
3      def __init__(self):
4          super().__init__()
5          self.mapa = {}
6          self.visitados = set()
7          self.posicao_interna = (0,0)
8
9      def agir(self):
10         percept_atual, vizinhos = self.ambiente.percept(self.posicao)
11
12         self.atualizar_estado_interno(vizinhos)
13         self.visitados.add(self.posicao_interna)
14         self.mapa[self.posicao_interna] = percept_atual
15
16         if percept_atual == "SUJO":
17             if self.ambiente.sofrer_acao(self, "LIMPAR") == "LIMPOU":
18                 self.pontuacao += 1 # Se limpou, ganha ponto
19                 self.mapa[self.posicao_interna] = "LIMPO"
20             return
21
22         # Procurar vizinho não visitado
23         for direcao, estado in vizinhos.items():
24             viz = self.coordenadas_vizinho(direcao)
25             if estado == "LIVRE" and viz not in self.visitados:
26                 if self.ambiente.sofrer_acao(self, direcao) == "MOVEU":
27                     self.posicao_interna = viz
28                     self.pontuacao -= 1
29                     print(direcao)
30                     return
31
32         # Senão, mover para qualquer vizinho livre
33         livres = [d for d, estado in vizinhos.items() if estado == "LIVRE"]
34         if livres:
35             direcao = random.choice(livres)
36             self.ambiente.sofrer_acao(self, direcao)
37             nova_pos = self.coordenadas_vizinho(direcao)
38             self.posicao_interna = nova_pos
39             self.pontuacao -= 1
40
41         def atualizar_estado_interno(self, vizinhos):
42             for direcao, estado in vizinhos.items():
43                 viz = self.coordenadas_vizinho(direcao)
44                 if estado == "LIVRE":
45                     self.mapa[viz] = "LIVRE"
46                 elif estado == "OBSTACULO":
47                     self.mapa[viz] = "OBSTACULO"
48                 else:
49                     self.mapa[viz] = "PAREDE"
50
51         def coordenadas_vizinho(self, direcao):
52             dx, dy = {
53                 "CIMA": (0, -1),
54                 "BAIXO": (0, 1),
55                 "ESQUERDA": (-1, 0),
56                 "DIREITA": (1, 0)
57             }[direcao]
58             return (self.posicao_interna[0] + dx, self.posicao_interna[1] + dy)

```

Listing 4: Recorte agentes.py (continuação)

Como o agente não tem conhecimento sobre a geografia do ambiente, ele constrói um mapa interno do ambiente, a partir da posição inicial, que ele guarda como posição (0,0), e vai adicionando os quadrados que ele vai visitando, a medida que os movimentos vão sendo realizados. Por exemplo, se ele estava inicialmente na posição (2,1), que para ele é a (0,0) e foi para a direita, chegando na posição (3,1), que para ele é a (1,0), nesse momento o agente sabe da existência dos quadrados (0,0) e (1,0), e também que já foram visitados.

Na verdade ele também sabe dos quadrados na borda desse mapa que foi descrito, pois quando o agente está em um quadrado, ele "vê" os quadrados adjacentes (menos a eventual sujeira neles), sem precisar visitá-los.

A partir desse mapa que ele constrói, e tendo guardado também quais posições já foram visitados, o agente baseado em modelos implementado, quando está num quadrado já limpo, move-se de preferência para um quadrado não-visitado (quando houver).

4 Interface

Implementamos uma interface para facilitar a visualização do agente baseado em modelos, mas os detalhes são irrelevantes. Porém, comentaremos o que acontece na simulação. Após o plot do gráfico (que comentaremos na próxima seção), aparece uma janela simulando um mundo aleatório do aspirador de pó.

O esquema de cores é o seguinte:

- Cinza: Representa os quadrados (livres) pelos quais o agente ainda não passou.
- Branco: Os quadrados (livres) pelos quais o agente já passou.
- Preto: Representa os obstáculos.
- Marrom: Representa os quadrados que contém sujeira.

Após a simulação, outra janela aparece mostrando o mapa que o agente construiu internamente. Cujo esquema de cores é:

- Verde: Representa os quadrados livres dos quais o agente tem conhecimento. Ele é composto pelos quadrados que ficaram brancos na simulação mais os quadrados que estão na "borda" desse trajeto. Pois o agente, quando visita um quadrado, consegue saber se os vizinhos estão livres ou se têm obstáculos, sem precisar visitá-los.
- Preto: Representa os obstáculos. Eles são identificados quando o agente visita um quadrado vizinho a eles, já que o agente não pode ir para um quadrado visitado.
- Laranja: Representa os quadrados do ambiente original que o agente não sabe que existem. A rigor, eles não fazem parte do mapa do agente, só existem para fazermos uma comparação (novamente, o agente não sabe que está num grid $n \times n$).

A figura a seguir exemplifica o mundo após o fim de uma simulação (esquerda) e o mapa interno do agente (direita). Note que os quadrados não visitados, mas que estão na borda dos quadrados visitados, também estão para o mapa (pois o agente os percebe). Note também que os quadrados laranjas podem ser tanto obstáculos, como quadrados livres: O agente não sabe que esses quadrados existem, quem dirá saber o que tem nesses quadrados.

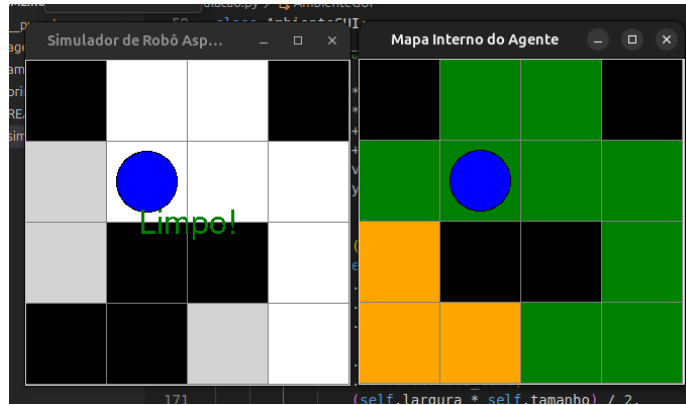


Figure 2: Simulação e mapa interno

5 Resultados

Executando `simulação.py`, primeiro simulamos os dois tipos de agentes, em 30 ambientes 4×4 gerados aleatoriamente, com número de passos no máximo 100, e produzimos um boxplot dos resultados. Também printamos a média dos desempenhos no terminal.

Em uma das execuções do programa, plotamos o seguinte gráfico (como os ambientes gerados são aleatórios, teremos um boxplot diferente a cada vez):

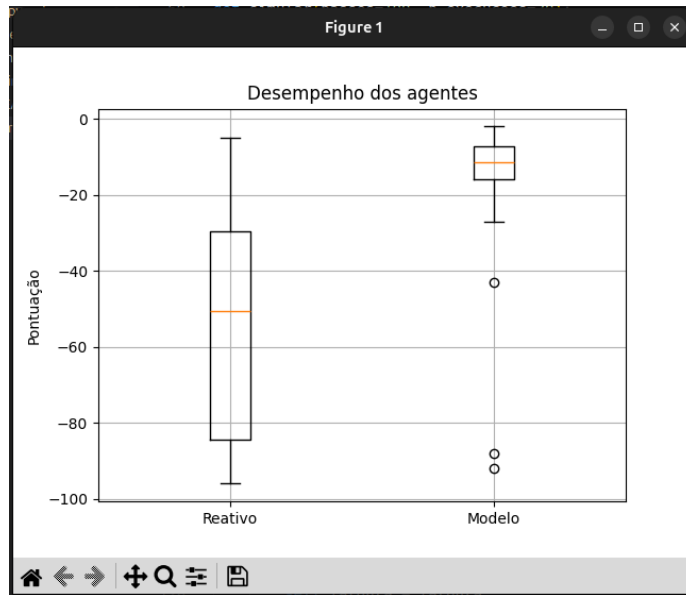


Figure 3: Boxplot das pontuações de cada agente em 30 ambientes

Podemos observar algumas coisas:

1. O agente reativo simples tem um desempenho mais variado, possivelmente por conta da aleatoriedade na sua tomada de decisão.
2. Os outliers provavelmente representam os casos em que o agente começa muito longe da sujeira, ou ele precisa contornar um obstáculo, ou mesmo nos casos em que o ambiente é gerado

de tal forma que o agente não consegue concluir sua missão, e daí ele fica se movimentando em vão, até atingir os 100 passos máximos.

3. Como esperado, o agente baseado em modelos tem um desempenho melhor, em média, do que o do agente reativo simples.

E esse é o print das pontuações médias, para a execução em questão:

```
(root) (base) C:\Users\gustavo\Documents\Inteligência Computacional\Primeiro_Trabalho_IntelComp\simulacao.py"
Pontuação média Reativo: -54.17
Pontuação média Modelo: -17.60
(MSR) (base) C:\Users\luís.gustavo@luís.gustavo-Lenovo-IdeaPad-S145-15ITL> (Documents\Inteligência
```

Figure 4: Pontuações médias de cada agente

6 Pontos de melhoria

Entre os vários aspectos que podem ser melhorados no projeto de ambos os agentes, um deles é evidenciado pelo seguinte problema: Os agentes foram projetados de tal maneira que quando eles atingem seus objetivos (de não deixar quadrados sujos), quando isso é possível, eles não têm a consciência de que o fizeram.

Ou seja, eles também não conseguem perceber quando um problema não tem solução, pois ficam agindo, à espera de que alguém os avise quando parar. Por exemplo, o ambiente, que é gerado aleatoriamente, pode impedir espacialmente (através de obstáculos) que o agente acesse um quadrado sujo, a fim de limpá-lo. Daí, se não paramos a execução dele, teoricamente ele vai agir sem nunca parar, perdendo pontos em vão e diminuindo sua performance. As figuras a seguir mostram dois exemplos de um ambientes assim.

A seguir, proponho 2 possíveis soluções para esse problema. A primeira consiste em avisarmos ao agente que ele não conseguirá limpar todos os quadrados sujos. Podemos verificar isso considerando que cada quadrado livre (sujo ou não) é um vértice de um grafo, e as arestas representam que dois quadrados são adjacentes, ou seja, é possível sair de um deles e chegar no outro, a partir de um dos 4 movimentos possíveis (cima, baixo, direita e esquerda). A partir daí, podemos verificar se todos os vértices representando quadrados sujos estão na mesma componente conexa que contém o vértice representativo da posição inicial do agente. Em outras palavras, verificamos se todos os quadrados sujos são alcançáveis.

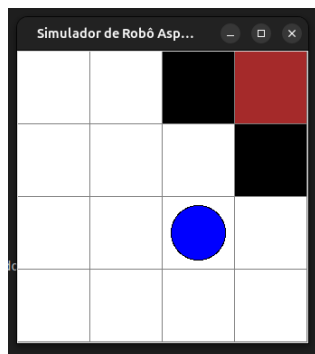


Figure 5: Primeiro exemplo



Figure 6: Segundo exemplo

Lembre-se que nesse ambiente, um quadrado limpo não se tornará sujo. Ou seja, o agente não precisa se preocupar com os quadrados limpos que ele não consegue alcançar, pois eles continuarão limpos.

A segunda abordagem consiste em o agente manter na memória uma lista de estados que ele vai descobrindo que existem, mas ainda não visitou, pois quando ele está num quadrado, ele percebe a existência dos quadrados adjacentes (se tem obstáculo ou não), e eventualmente ao visitar esses vizinhos, ele pode reconhecer a existência de mais quadrados.

A partir do momento que essa lista esvazia, o agente já visitou todos os estados alcançáveis para ele e, mesmo quando ele não é dito para parar (ou seja, ainda existem quadrados sujos), ele escolhe literalmente parar, pois ele chega à conclusão de que o que ele poderia fazer, já foi feito, já todos os quadrados estão limpos (e continuarão assim), e ele não consegue visitar novos quadrados. É como se ele considerasse que todo o ambiente são os quadrados que ele consegue alcançar, e então ele atingiu a meta naquele ambiente.

Referências

- [1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd Edition, Pearson, 2010.
- [2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th Edition, Pearson, 2021.