Problema do Taxi

Antonio Fuziy

Fabrício Barth

Reinforcement Learning

Taxi Driver Problem - Reinforcement Learning

Aluno: Antonio Fuziy

Para a resolução do problema do taxi, foi necessário implementar uma classe `TaxiDriver` que instancia a classe `State` da biblioteca `Graph`, para inicializar-la utilizou-se alguns parâmetros como `operator`, representando o movimento realizado pelo taxi ao longo do problema, `taxi_position`, representando a posição do taxi no momento atual, `got_passenger`, sendo uma booleana que define se o passageiro está ou não dentro do taxi, `passenget_position`, representando a posição onde o passageiro se encontra esperando o taxi, `goal_location`, define a posição do destino, `start_position` define a posição inicial do taxi, `obstacles` representa um array de todos os obstáculos dentro do percurso e por fim, `size_x` e `size_y` definindo o tamanho do percurso a ser tratado.

```
class TaxiDriver(State):
  def __init__(self, operator, taxi_position, got_passenger, passenger_position, goal_location, start_position,
obstacles, size_x, size_y):
    self.operator = operator
    self.taxi_position = taxi_position
    self.got_passenger = got_passenger
    self.passenger_position = passenger_position
    self.goal_location = goal_location
    self.start_position = start_position
    self.obstacles = obstacles
    self.size_x = size_x
    self.size_y = size_y
```

Para geração dos sucessores deveriam ser observados alguns pontos, como verificar se o taxi pode se movimentar para algum dos lados, ou se ele pode pegar o passageiro, dessa forma, utilizou-se um método `can_move_taxi()` que observa a possibilidade de movimentação do taxi e o método `can_get_passenger()` o qual verifica se é possível que o passageiro entre no taxi. Ambos esses métodos foram representados abaixo:

```
def can_get_passenger(self):
  if (self.taxi_position == self.passenger_position) and (not self.got_passenger):
    return True
  return False

def can_move_taxi(self, next_position):
  for obstacle in self.obstacles:
    if next_position == obstacle:
    return False
  return True
```

O método de geração dos sucessores foi representado abaixo:

```
def sucessors(self):
  sucessors = []

  up = (self.taxi_position[0] - 1, self.taxi_position[1])
  down = (self.taxi_position[0] + 1, self.taxi_position[1])
  left = (self.taxi_position[0], self.taxi_position[1] - 1)
  right = (self.taxi_position[0], self.taxi_position[1] + 1)

  if self.can_move_taxi(up) and (up[0] ⩾ 0):
    sucessors.append(TaxiDriver("UP", up, self.got_passenger, self.passenger_position, self.goal_location,
self.start_position, self.obstacles, self.size_x, self.size_y))
  if self.can_move_taxi(down) and (down[0] < self.size_y):
    sucessors.append(TaxiDriver("DOWN", down, self.got_passenger, self.passenger_position, self.goal_location,
self.start_position, self.obstacles, self.size_x, self.size_y))
  if self.can_move_taxi(right) and (right[1] < self.size_x):
    sucessors.append(TaxiDriver("RIGHT", right, self.got_passenger, self.passenger_position, self.goal_location,
self.start_position, self.obstacles, self.size_x, self.size_y))
  if self.can_move_taxi(left) and (left[1] ⩾ 0):
    sucessors.append(TaxiDriver("LEFT", left, self.got_passenger, self.passenger_position, self.goal_location,
self.start_position, self.obstacles, self.size_x, self.size_y))

  if self.can_get_passenger():
    sucessors.append(TaxiDriver("GET PASSENGER", self.taxi_position, True, self.passenger_position, self.goal_location,
self.start_position, self.obstacles, self.size_x, self.size_y))

  return sucessors
```

Por fim, para realizar a heurística como forma de otimização do problema do taxi, utilizou-se a verificação da distância euclidiana do taxi até o passageiro e a mesma distância até o destino, sempre verificando se o passageiro encontra-se dentro do taxi para observar uma ou outra distância. Esse método foi representado abaixo:

```
def h(self):
  if self.got_passenger:
    return abs(self.taxi_position[0]-self.goal_location[0]) + abs(self.taxi_position[1]-self.goal_location[1])

  return abs(self.taxi_position[0]-self.passenger_position[0]) + abs(self.taxi_position[1]-self.passenger_position[1])
```