Algoritmo Genético Para Processo de Decisão de Markov

Vinicius Meirelles de Siqueira

04.08.2022 © 2021.2 UFRB

- *Vinicius Meirelles de Siqueira, Universidade Federal do Recôncavo da Bahia, aluno. E-mail: vmeirelles.eng@gmail.com*
- *Camila Bezerra da Silva, Universidade Federal do Recôncavo da Bahia, professora orientadora. E-mail: camilabezerra@ufrb.edu.br*

1 Contextualização

Os últimos anos têm sido de grande avanço para o campo da inteligência artificial, o desenvolvimento da capacidade de processamento dos computadores tem permitido que cada vez mais problemas complexos possam ser solucionados. Sistemas em inteligência artificial permitem que agentes construídos por software determinem um comportamento ideal de um agente em um cenário específico.

A criação de um quadro matemático para modelar a tomada de decisões no mundo tem sido muito facilitada por esse tipo de algoritmo. Diversos modelos como os PDM permitem o estudo de problemas reais assim como problemas de otimização dos algoritmos.

2 Objetivos

Criar algoritmo genético capaz de resolver problemas de decisão de Markov.

3 Fundamentação Teórica

3.1 Agentes e Atuadores

Os agentes são mecanismos para se chegar a metas definidas, eles percebem o mundo através de seus sensores e realizam ações através de seus atuadores.

3.2 Problemas de Decisão de Markov

Um problema de Decisão de Markov é um problema que segue um esquema proposto pelo matemático Andrey Markov. Nesse problema, um agente deve decidir a melhor ação a ser tomada analisando seu estado atual. Existe nesse problema: um conjunto de possíveis estados, um conjunto de ações possíveis, o(s) modelos e recompensas a serem dadas em uma determinada situação.

3.3 Algoritmos Genéticos

Algoritmos genéticos são técnicas de otimização baseada em busca, que se baseiam na teoria de Seleção Natural.

Assim como no modelo de Darwin, o modelo possui uma população - um conjunto de possíveis soluções para o problema, cromossomos - as possíveis soluções do problema, gene - um elemento da solução, alelo - o valor que um gene pode assumir.

O modelo segue a seguinte estrutura:

1. Inicialização de uma população inicial.
2. Avaliação da População
3. Reprodução
4. Mutação
5. Seleção
6. Repetição dos passos 2. a 5.
7. Retornar melhor elemento

4. MÉTODOS

4.1 Criação do Ambiente Virtual:

Com finalidade de realizar a criação de um modelo virtual para que um agente trabalhe foi utilizado o pacote de ferramentas *OpenIA gym* para a linguagem de programação python.

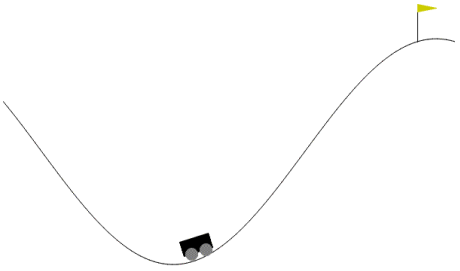

Fig. 1. Mountain Car DMP.

O Mountain Car DMP é um PDM determinístico que consiste em um carro colocado no fundo de um vale, com as únicas ações possíveis sendo as acelerações que podem ser aplicadas ao carro em qualquer direção. O objetivo do PDM é acelerar estrategicamente o carro para alcançar o estado de meta no topo da colina direita.

Observações possíveis: velocidade do carro, posição no eixo x. Ações: acelerar para esquerda, não acelerar, acelerar para a direita. O carro obedece as seguintes equações:

$$velocity(1)=velocity(t)+(action-1)*force-cos(3*position(t))*gravity)$$
$$position(t+1) = position(t) + velocity(t+1)$$

4.2 Criação do Algoritmo Genético

Definiu-se então, que a sequência de ações possíveis para o atuador seria um indivíduo na população. Exemplo de indivíduo: aceleração a esquerda, não acelerar, acelerar a direita, não acelerar e assim por diante.

```python
#Gerar uma ação (GENE)
def gerarAcao():
    return random.choice(range(0,qAcoes))

#Gera um conjunto de ações (CROMOSSOMO)
def gerarIndividuo():
    lista = []
    for i in range (0, qMovimentos):
        lista.append(gerarAcao())
    return lista

#Gera um conjunto de indivíduos (POPULAÇÃO)
def gerarPopulacao():
    lista = []
    for i in range (0, qIndividuos):
        lista.append(gerarIndividuo())
    return lista
```

Fig. 3 - Funções de criação da população.

Para a função de avaliação optou-se por manter o valor de distanciamento do veículo para gerar um valor de comparação entre os veículos que chegaram mais longe ou não. A função avaliar coloca os veículos de acordo sua ordem de avaliação, para que a função seleção possa selecionar apenas quais irão sobreviver.

```python
#Gera um valor de avaliação para cada indivíduo (FITNESS)
def fitness(list):
    fitness = []
    env = gym.make(envName, new_step_api=True)
    for j in range (0,len(list)):
        env.reset()
        for i in range (0, qMovimentos):
            state, reward, done, info, data = env.step((list[j][i]))
            if(i == qMovimentos - 1):
                posFinal = state[0]
        fitness.append(posFinal)
    return fitness

#Ordena os indivíduos de acordo melhor fitness (AVALIAÇÃO)
def avaliar(list):
    avalicaoes = fitness(list)
    print(avalicaoes)
    print("Media da geração: ", sum(avalicaoes)/len(avalicaoes))
    lista = [x for (y,x) in sorted (zip(avalicaoes,list))]
    print("Melhor da geração: ", lista[len(lista)-1])
    return lista

#Seleciona os indivíduos de melhor fitness apenas (SELEÇÃO)
def selecionar(list):
    lista = avaliar(list)
    return lista[(len(lista)//2):] #segunda metade
```

Fig. 3 - Funções de avaliação e Seleção

Para a criação das próximas gerações, para cada indivíduo pai avaliado em uma posição n de posicionamento, ele terá $n^2/(\text{sum i } (n[i]^2))$ chances de ser chamado para reprodução dos próximos indivíduos. A função crossover é realizar esse sorteio de pais e criar os filhos. Os filhos por sua vez são criados pela função criar filho, que junta os genes por 1 ponto de inserção ao meio do cromossomo, chamando também a função mutação para que esse novos filhos tenham chances de ter algum dos movimentos em seus genes trocados ao acaso.

```python
#Algoritmo que gere um gene aleatório em novas crianças (MUTAÇÃO)
def mutation(item):
    probability = random.randint(0, 100)
    if probability <= iMutacao:
        mutated_allele = random.randint(0, qMovimentos-1)
        item[mutated_allele] = random.randint(0, qAcoes-1)
    return item

#Algoritmo que junta os genes dos pais (REPRODUÇÃO II)
def criarFilho(parent1, parent2):
    qMov = qMovimentos//2
    return mutation(parent1[:qMov] + parent2[qMov:])

#Sorteia 2 entre os mais aptos para procriar (REPRODUÇÃO I)
def crossover(parents):
    kids = []
    cont = 0
    for i in range(0, qIndividuos-len(parents),2):
        pai = random.choice(indices)
        mae = random.choice(indices)
        sun1 = criarFilho(parents[pai],parents[mae])
        sun2 = criarFilho(parents[mae],parents[pai])
        kids.append(sun1)
        kids.append(sun2)
    return kids
```

Fig. 4 - Funções de reprodução

Por último, trabalhou-se com os seguintes parâmetros para o problema do veículo:

Gerações: 1000.
Indivíduos: 80.
Aplicações de Aceleração no Veículo: 200.
Chance de mutação 5%.

5 Resultados

5.1 Resultados Mountain Car DMP

Com a execução do algoritmo por mil gerações se obteve o seguinte resultado:
População 250:
[0.06824787, 0.06684995, 0.072425835, 0.059548777, 0.075333945, 0.03427038, -0.026277006, 0.056347672, 0.05770506, 0.10203482, 0.08484185, -0.016882205, 0.10371246, 0.079610914, -0.0029459328, 0.016757676, 0.05693549, 0.05228452, 0.07728642, 0.07446639, 0.042988557, 0.06789196, 0.033128653, 0.026130224, 0.109660536, 0.0107468, 0.04217976, 0.02580197, 0.06392785, -0.0037294142, 0.07483074, 0.021250099, 0.08631746, 0.075022295, 0.020790672, 0.0637133, -0.014249439, 0.10679312, 0.07316174, 0.10258565, 0.019490339, -0.007437454, -0.0065053054, -0.009291416, 0.089404084, 0.052202344, 0.0025002963, 0.028027644, 0.10067557, 0.029015932, 0.010719772, -0.0062055164, 0.04134287, 0.09693522, 0.045968115, -0.01913772, 0.053368755, 0.043080654, 0.09367376, 0.09496886, -0.0126827285, 0.105505966, 0.07019397, -0.03538796, -0.013404782, -0.014725446, 0.03336464, 0.087517686, 0.051327534, 0.06531235, 0.06363128, 0.0896472, 0.07468427, 0.008816344, 0.07586104, 0.07029299, 0.05695923, 0.09855116, 0.055362605, 0.076369815]

Melhor da geração: [2, 1, 1, 0, 0, 2, 2, 0, 1, 0, 0, 2, 1, 0, 2, 0, 0, 2, 0, 2, 1, 1, 2, 0, 0, 1, 1, 2, 2, 2, 2, 2, 1, 1, 2, 0, 1, 2, 2, 1, 2, 1, 0, 1, 0, 1, 2, 2, 1, 0, 2, 0, 2, 2, 0, 0, 2, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 2, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 2, 1, 0, 2, 1, 2, 1, 0, 0, 1, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 1, 2, 2, 0, 2, 1, 2, 0, 1, 2, 1, 2, 1, 2, 2, 1, 2, 0, 0, 0, 2, 2, 1, 1, 2, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 2, 0, 1, 1, 2, 1, 0, 1, 1, 2, 0, 0, 0, 1, 2, 2, 2, 1, 1,
2, 0, 2, 0, 0, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1, 2, 2, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1]

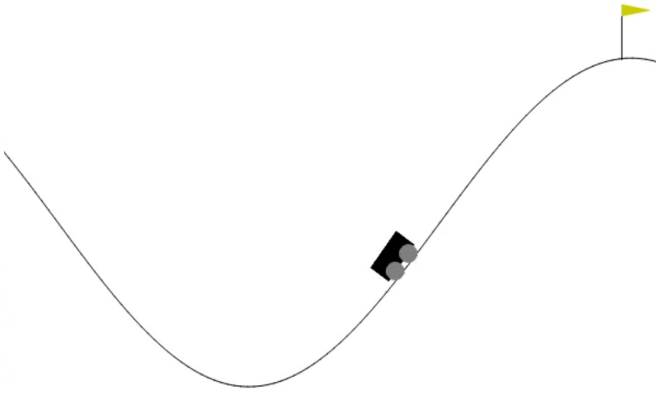

Fig. 5 - Imagem para o resultado 250.

População 500:
[0.15722759, 0.26403522, 0.16690078, 0.25936615, 0.26382086, 0.20975573, 0.24263966, 0.24176404, 0.26337108, 0.1814435, 0.25746992, 0.27336976, 0.23539694, 0.2691169, 0.22041154, 0.24638058, 0.20705815, 0.26294535, 0.22972074, 0.27230874, 0.25004813, 0.27145845, 0.27038884, 0.22242686, 0.20641047, 0.21338566, 0.19382219, 0.19448408, 0.17600583, 0.26241964, 0.2675457, 0.22404176, 0.26610336, 0.23137276, 0.24490266, 0.22275154, 0.27181414, 0.195949, 0.26143178, 0.16537689, 0.24822399, 0.26354983, 0.2202985, 0.20154154, 0.20727505, 0.25585797, 0.26353619, 0.2361251, 0.16847925, 0.27346623, 0.17237402, 0.21708551, 0.18213832, 0.2711701, 0.17452335, 0.17227317, 0.19655679, 0.27155122, 0.26702628, 0.24377127, 0.2640201, 0.15569025, 0.2667831, 0.26101032, 0.15272571, 0.26824552, 0.23789646, 0.22926271, 0.23653768, 0.17246209, 0.18931739, 0.25368112, 0.27350745, 0.20589317, 0.21341151, 0.2720406, 0.18227307, 0.18163402, 0.17744711, 0.1910979]
Melhor da geração: [2, 1, 1, 0, 0, 2, 2, 0, 1, 1, 1, 2, 1, 0, 2, 0, 0, 2, 2, 2, 2, 1, 2, 0, 0, 1, 1, 2, 2, 2, 2, 2, 1, 1, 2, 0, 1, 2, 2, 1, 2, 1, 0, 1, 0, 1, 0, 2, 1, 0, 2, 0, 2, 2, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 2, 2, 1, 2, 2, 1, 2, 1, 0, 0, 1, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 1, 2, 2, 0, 1, 1, 2, 1, 1, 2, 1, 2, 1, 2, 2, 1, 1, 0, 0, 0, 2, 0, 1, 1, 2, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 2, 0, 0, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1, 2, 2, 2, 1, 2,
2, 0, 2, 0, 0, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1]

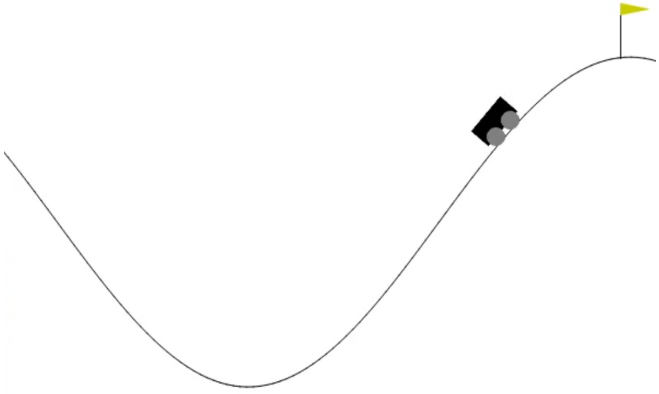

Fig. 6 - Imagem para o resultado 500.

População 750:
[0.35292178, 0.36054114, 0.35114795, 0.35819468, 0.34493375, 0.3702055, 0.25369462, 0.35763583, 0.2912992, 0.36099422, 0.22284107, 0.35762686, 0.29953712, 0.36519653, 0.34095004, 0.35583034, 0.35977858, 0.36105457, 0.3609602, 0.32484865, 0.2495673, 0.25287792, 0.34506643, 0.23087996, 0.3307791, 0.22632024, 0.25363955, 0.28469396, 0.3327717, 0.3423068, 0.3528612, 0.22664048, 0.29564342, 0.3493723, 0.2883238, 0.3553929, 0.3433104, 0.21843617, 0.3265532, 0.26954073, 0.36067688, 0.24751098, 0.3570644, 0.23305386, 0.35376924, 0.23516846, 0.23437114, 0.33932987, 0.30609792, 0.33900577, 0.34486425, 0.29079875, 0.3524418, 0.357029, 0.3610713, 0.3600027,

0.35184234, 0.3322662, 0.3500094, 0.30253518, 0.35338745, 0.33541775, 0.22516088, 0.3135535, 0.24934997, 0.2895808, 0.34977713, 0.35702062, 0.34848318, 0.35907826, 0.3500343, 0.29631153, 0.30037302, 0.22540875, 0.28470615, 0.3502427, 0.29070586, 0.23639898, 0.35346198, 0.25189194]
Melhor da geração: [2, 1, 1, 0, 2, 2, 2, 0, 1, 1, 1, 2, 1, 0, 2, 0, 2, 2, 2, 2, 2, 1, 2, 0, 0, 1, 1, 2, 2, 2, 2, 2, 1, 1, 2, 0, 1, 2, 2, 1, 2, 1, 0, 1, 0, 1, 0, 2, 0, 0, 2, 0, 2, 2, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 1, 2, 1, 2, 0, 1, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 1, 2, 2, 0, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2, 2, 1, 1, 0, 0, 0, 2, 0, 0, 1, 2, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 2, 0, 0, 1, 0, 1, 0, 2, 1, 2, 0, 0, 0, 1, 2, 2, 2, 1, 2,
2, 0, 2, 0, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1]

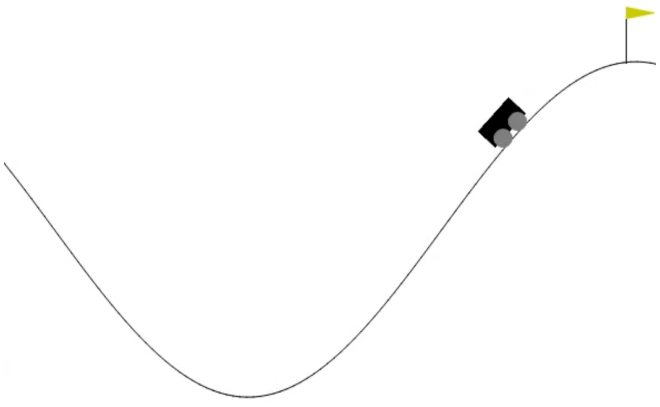

Fig. 4 - Imagem para o resultado 750.

População 1000: [0.45820385, 0.4192548, 0.48256272, 0.5339585, 0.6, 0.47483337, 0.48007503, 0.6, 0.5119274, 0.5716169, 0.5210743, 0.4315493, 0.50824755, 0.47404853, 0.6, 0.43402487, 0.6, 0.52447593, 0.5486792, 0.49818206, 0.38484508, 0.4787804, 0.50806123, 0.5195569, 0.5267555, 0.6, 0.47181877, 0.52102846, 0.3739932, 0.47114372, 0.53854007, 0.3964096,
0.55037326, 0.52064186, 0.52714014, 0.4186263, 0.50286704, 0.5465603, 0.4662365, 0.3838178, 0.41549262, 0.47003487, 0.513214, 0.6, 0.45042613, 0.4336257, 0.5418537, 0.5471225, 0.46795797, 0.50984997, 0.46974236, 0.48649937, 0.36517116, 0.46655694, 0.5716169, 0.42739072, 0.6, 0.40665704, 0.47098577, 0.6, 0.53942645, 0.508875, 0.6, 0.54980516, 0.41868484, 0.49038532, 0.4110143, 0.5306873, 0.45546895, 0.6, 0.54673827, 0.38721567, 0.5108279, 0.44837087, 0.54302555, 0.38798174, 0.40751496, 0.6, 0.53878486,
0.3931217]
Melhor da geração: [2, 1, 1, 2, 2, 2, 2, 0, 1, 1, 1, 2, 1, 0, 2, 0, 2, 2, 2, 2, 2, 1, 2, 0, 0, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2, 0, 1, 2, 2, 0, 1, 1, 0, 1, 0, 1, 0, 2, 0, 0, 2, 0, 2, 2, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 2, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 1, 2, 2, 0, 2, 1, 2, 1, 1, 2, 1, 2, 1, 0, 2, 1, 1, 0, 0, 0, 2, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 2, 0, 0, 1, 0, 1, 0, 2, 1, 2, 0, 0, 2, 1, 2, 2, 2, 1, 2,
2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1]

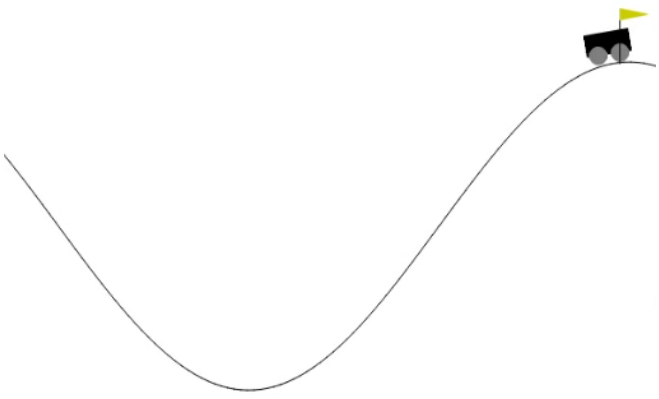

Fig. 4 - Imagem para o resultado 1000. Carro atinge a bandeira.

Realizando o método de precisão para o resultado, iremos medir a quantidade de carrinhos que chegou a uma distância de até 95% do alvo (como o percurso total tem 1.8 metros a tolerância é de 0.09m. Lembrando que o resultado 1000 já contém a nova geração, para fins de análise iremos descartar esses dados, e utilizar apenas a acurácia sobre a estabelecida geração de pais. Temos, então por fim: 36 / 40 = 90%.

7 Conclusão

Foi possível conferir eficiência na técnica de algoritmos genéticos, mostrando alto resultado, para lidar com inteligências artificiais, encontrando as ações para modificar o ambiente de acordo com o desejado, sendo útil para resolver problemas de PDM.

7 Referências

[1] Russell, S. and Novig, P. Artificial Intelligence: a Modern Approach, 3nd Edition, Campus, 2010.

[2] Efficient memory-based learning for robot control Andrew William Moore November 1990

[3] .A Markovian Decision Process, Indiana Univ. Dynamic Programming and Optimal Control 3rd Edition, Volume II by Dimitri P. Bertsekas Massachusetts Institute of Technology 2011