Atividade Avaliativa I
Inteligência Artificial - PUCRS

Martin W. D. Ferreira

2021/2

Resumo

Este trabalho objetivou apresentar a solução de um labirinto representado em uma estrutura de matriz duas dimensões utilizando algoritmo genético para encontrar o caminho desde o ponto inicial até o ponto final.

Palavras-chave: linguagem. programação.

1 Definição do Problema

O problema se consiste em um labirinto em formato de uma matriz de duas dimensões, que no caso sempre terá o mesmo número de colunas e linhas. Um exemplo de matriz válida:

```
12
E 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 1 1 1
1 0 0 0 0 1 1 1 0 1 1 0
1 0 1 1 1 1 1 1 0 0 0 0
0 0 0 1 0 0 0 0 1 0 1 1
1 1 0 0 0 1 0 1 0 0 1 1
1 1 1 0 1 1 0 0 0 1 1 0
0 0 1 0 0 1 0 1 0 1 1 0
0 0 0 0 1 1 0 0 0 1 1 0
1 1 1 0 1 0 0 1 1 1 1 0
1 1 1 0 1 0 0 0 0 1 1 1
1 1 1 0 0 0 0 1 0 0 0 S
```

Essa matriz sempre virá em um arquivo texto no formato apresentado, no caso a primeira linha sempre será o inteiro indicando o número de linha e colunas, o exemplo mostra uma matriz de 12x12.A letra 'E' na matriz representa o ponto inicial do labirinto, já a letra

'S' representa o ponto final. Os algorismos presentes na matriz representam os caminhos possíveis de seguir, o número 0 indica um caminho livre e o número 1 indica uma parede no labirinto, ou seja, a ideia do problema é encontrar o caminho do ponto inicial até o final, de forma que não se saia do universo do labirinto e que o jogador não passe por nenhuma parede ao longo dele.

O problema exige que se ache algum caminho entre os pontos sem necessariamente ser o melhor possível, sendo que os possíveis movimentos dentro do universo da matriz, são que a partir a de cada célula é é possível se movimentar para uma das quatro direções: para cima, para baixo, para direita e para a esquerda.Outro ponto importante para a resolução desse problema é que o algoritmo que deve ser implementado para solucionar esse labirinto deve ser ou algoritmo genético ou *simulated annealing*.

2 Algoritmo Genético

A escolha do algoritmo a ser utilizado para resolução do problema era de grande importância, pois toda a modelagem de como resolveríamos o desafio se basearia nele, foi escolhido o algoritmo genético (WIKPEDIA, 2019), por se tratar de um algoritmo conhecido por conseguir tratar de problemas muito mais complexos e por ser provado tanto na biologia quanto na computação por ser assertivo. Algoritmos Genéticos são inspirados no princípio Darwiniano da evolução das espécies e na genética. São algoritmos probabilísticos que fornecem um mecanismo de busca paralela e adaptativa baseado no princípio de sobrevivência dos mais aptos e na reprodução.

O algoritmo começa por criar uma população inicial completamente aleatória, onde essa população, no caso do problema que estamos implementando a solução, será basicamente vários jogadores que estarão tentando chegar ao final do labirinto, e cada um desses jogadores possui uma série de movimentos aleatórios no labirinto, dessa maneira espera-se que esses movimento venham a ser mais assertivos ao longo da execução do algoritmo de forma que em algum momento encontre uma solução possível de caminho até o final do labirinto.

O primeiro passo é montar essa população de jogadores com movimentos aleatórios, o que temos que fazer em seguida é conseguir calcular o quão bom são os movimentos de cada um dos jogadores, para isso devemos ter uma função heurística que realize esse cálculo, para o nosso problema os movimentos são considerados bons se deixam o jogador mais próximos do ponto de saída do labirinto e se o jogador não atravessou nenhuma parede no meio do caminho.

O terceiro passo do algoritmo após termos como saber quais são os melhores indivíduos da população é conseguirmos realizar uma especie de reprodução entre os mesmos, ou seja, para irmos melhorando a nossa população devemos ir misturando os genes entre os indivíduos de forma que conseguimos resultados melhores do que na última população, e para garantir que teremos uma nova população mais apta que a anterior, é feito uma ordenação na população onde só reproduzimos os indivíduos mais aptos, dessa maneira conseguimos resultados melhores na próxima geração. Outro ponto importante também de levarmos em conta, é que devemos realizar mutações para os indivíduos na nova geração para que assim conseguirmos ter mais diversidade na nossa população e ter surpresas positivas com mutações que nos ajudem a obter indivíduos mais apto.

Por fim o algoritmo continua melhorando a sua população realizando reproduções e mutações nela, sempre checando utilizando a função de aptidão para conferir se ainda

não foi encontrado uma possível solução par o problema, que seria a condição de parada do algoritmo, pois o problema estaria resolvido. Nas próximas seções do artigo, será apresentado em detalhes como foi implementado cada uma dessas partes no código.

3 Estrutura da Solução

Foi escolhida a linguagem de programação Python para o desenvolvimento da solução, onde o código fonte está separado em 3 arquivos distintos: *App.py* que é o arquivo que deve ser considerado como o executável, pois nele onde que ocorrem as chamadas para as outras partes do programa que possuem as funções de execução do algoritmo, também temos o arquivo *Player.py* que possui a estrutura de um indivíduo da população que será criada, e por fim temos o arquivo *Policy.py*,onde ficam de fato as funções do algoritmo que comentamos na seção anterior.

4 Desenvolvimento do algoritmo

Nessa seção do artigo, será apresentado como foi modelado o algoritmo genético (TRUONG, 2015) para que consiga resolver o problema proposto, assim como justificativas e análises o do porque foram tomadas decisões chaves ao longo do desenvolvimento do código.

4.1 Leitura do arquivo de texto

Primeiramente era necessário armazenar o arquivo texto para alguma estrutura dentro do código, para isso foi decidido que o ideal seria criar um arreio de duas dimensões, onde esse arreio poderia armazenar a estrutura do arquivo de forma que os elementos pudessem ser acessados pelos seus índices tanto de linha quanto da sua coluna, então criamos um arreio de duas dimensões chamado de *maze* que nele era possível acessar qualquer algarismo presente da matriz simplesmente fornecendo os índices deles, por exemplo, para acessar o primeiro elemento da matriz, basta utilizar a estrutura de forma que ela acesse o elemento que está na linha 0 e coluna 0, então quando chamamos o arreio *maze[0][0]*, é retornado o resultado esperado.

4.2 Criação da população inicial

Após termos a estrutura da matriz armazenada, foi desenvolvido a primeira parte do algoritmo, que basicamente consistia em criar uma população de jogadores com valores de movimentação aleatória, para seguir com essa ideia, primeiramente foi criado uma classe a parte no arquivo chamado *Player.py*, onde nela ficava a estrutura de cada indivíduo, cada indivíduo possui variáveis para representar em qual ponto que eles se encontram no labirinto, então foi implementado uma variável x para representar em qual linha o jogador se encontra e outra variável y que representa a coluna do jogador, cada jogador também possui uma variável chamada de *penalty*, que possui o intuito de representar se aquele jogador passou por algum caminho que não devia no labirinto ou se saiu dos limites do mesmo, por fim cada jogador também possui um arreio de inteiros que representam os movimentos que no labirinto, os movimentos são associados da seguinte maneira:

- 0: o jogador anda para a célula acima
- 1: o jogador anda para a célula abaixo

- 2: o jogador anda para a célula à esquerda
- 3: o jogador anda para a célula à direita

Após ter essa estrutura pronta, gerar a população inicial era uma tarefa simples, foram escolhido dois valores fixos, um para o tamanho a população que gostaríamos de ter e outro para quantos movimentos cada jogador teria. Para o valor inicial da população foi decidido que seria 1000 e 30 para o de movimentos, esses valores inicialmente foram apenas inciais, pois é normal ocorrer ajustes nesses valores ao longo da implementação do programa. Para gerar a população de fato, basicamente foi implementado um *loop* que rodava o número de interações de acordo com o tamanho da população, e a cada interação era criado um novo objeto do tipo jogador, que são os indivíduos da população e para cada um dos jogadores era passado como paramêtro um arreio do tamanho do número de movimentos, com cada item do arreio sendo um número de 0 a 4, sendo assim quando o *loop* fosse finalizado teríamos a população inicial criada com valores de movimento aleatórios para cada um dos jogadores.

4.3 Função Heurística

A função heurística ou função de aptidão no algoritmo, é a função que avalia o quanto aquele indivíduo é apto, ou seja, o quanto mais ele está perto da solução final que buscamos. Para esse algoritmo o mais importante é que o jogador consiga chegar no ponto de saída do labirinto através de seus movimentos, mas também é preciso que o jogador não passe por cima de nenhuma parede no meio do seu caminho, tendo em vista esses dois pontos, foi construída a função onde ela mede a distância do jogador, a partir de sua posição atual, até o ponto de saída do labirinto, essa distância pode ser calculada utilizando a função d=|x2-x1| + |y2-y1|, onde d x2 e y2 são os valores de linha e coluna do ponto final, já x1 e y1 são do ponto inicial, d é o resultado que nada mais é o valor da distância.

Sendo assim buscamos que o valor de d seja zero, pois isso indica que o jogador está no ponto final. Além do valor da distância, também é somado o valor das penalidades de cada um dos jogadores, as penalidades são adicionadas quando um jogador bate na parede ou sai do labirinto, para ser considerada uma solução ideal o resultado final dessa soma da distância e das penalidades seja zero.

Após ser executada a função de aptidão, logo em seguida é chamada a função de ordenação da população. A função de ordenação basicamente ordena a população de jogadores, baseados nos resultados que foram gerados na função de aptidão, realizamos essa operação para que quando formos realizar a reprodução para gerar uma nova população, só ser utilizados os indivíduos mais aptos da população anterior. Logo abaixo temos o código utilizado para a função heurística chamada de *fitness*:

```
1   while i < len(self.population):
2         for index,item in enumerate(self.population[i].moves):
3             if(item == 0):
4                 self.population[i].moveUp()
5             elif(item == 1):
6                 self.population[i].moveDown()
7             elif(item == 2):
8                 self.population[i].moveLeft()
9             elif(item == 3):
10                self.population[i].moveRight()
```

```
            # add penalty if player hit a wall
            if(self.data[self.population[i].x][self.population[i].y]
            == '1'):
                self.population[i].addPenalty()

        # fitness = | (x2-x1) | + | (y2-y1) | + accumulated penalties
        score = ((self.finalX - self.population[i].x)**2 +
                 (self.finalY - self.population[i].y)**2) +
                 self.population[i].penalty
        self.population[i].addScore(score)
        i+=1
```

E a função de ordenação da população:

```
    fiteness = sorted(self.population, key=lambda population: population.s
    self.population = fiteness
```

4.4 Reprodução

A reprodução da população ocorre de forma que serão utilizados os indivíduos mais aptos para gerar a nova geração dos mesmos. Para realizar essa operação, são selecionados aleatoriamente dois indivíduos da população anterior que sempre estarão entre a metade mais apta da população, eles são chamados de pais que irão gerar mais dois novos indivíduos para a população subsequente, esses novos herdam os valores dos movimentos de seus pais, então cada um dos novos indivíduos recebe uma parte de cada arreio que armazena os movimentos de um dos seus pais, o ponto de corte que é escolhido para dividir os arreios de movimentos dos pais é um número aleatório, dessa forma o primeiro filho gerado receberá as partes de corte, primeiro do indivíduo pai número 1 e depois a parte que restou do pai número 2, já o segundo filho gerado receberá o contrario, primeiro a parte do segunda pai escolhido e depois a do primeiro.

Uma melhoria que foi implementada ao longo do desenvolvimento do código foi a de armazenar o valor do melhor indivíduo da população anterior, pois era perdido muito valor na hora da execução do código porque o melhor resultado sempre era excluído, então com essa melhoria nessa função não é substituído o jogador mais apto da geração passada. Abaixo é apresentado o código da função que realiza essa operação:

```
    halfPopulationSize = int(len(self.population)/2)
    movesSize = int(len(self.population[0].moves)  )
    i=1
    newPopulation = []
    while i < halfPopulationSize:
        number = randint(movesSize)
        firstParent = self.population[randint(len(self.population)/2)]
        .moves[0:number]
        secondParent = self.population[randint(len(self.population)/2)]
        .moves[number:movesSize]
        child1= np.concatenate((firstParent, secondParent))
        child2= np.concatenate((secondParent, firstParent))
        newPopulation.append(Player.Player(self.data, child1))
        newPopulation.append(Player.Player(self.data, child2))
```

```
15          i+=1
16      self.population[0].resetPlayer()
17      newPopulation.insert(0, self.population[0])
18      self.population=newPopulation
```

4.5 Mutação

A mutação é uma parte muito importante do algoritmo, pois ela muitas vezes que faz uma solução possível aparecer, essa função alterar um dos genes de indivíduos da população de acordo com a probabilidade definida para isso ocorrer. No caso do nosso problema cada gene seria um dos movimentos que os jogadores executam e a chance de um desses movimento ser alterado é de 60%, o gene que será alterado é escolhido de forma aleatória dentro do arreio de movimentos e o único indivíduo que nunca terá seu gene alterado é sempre o mais apto da geração mais recente, pois ele armazena a melhor solução encontrada até o momento, então não faria sentido perdemos o melhor valor que possuímos até aquele momento.

4.6 Encontrar solução Final

Por fim temos a função que encontrará a solução final para o algoritmo, ela basicamente junta todas as outras funções criadas em um *loop*, de forma que ele ficará executando até que seja encontrado um indivíduo que seja considerado a solução possível para o problema.

Referências

TRUONG, T. *Solving A 2D Maze Game Using a Genetic Algorithm and A* Search*, S.I, 2015. Disponível em: <https://tonytruong.net/solving-a-2d-maze-game-using-a-genetic-algorithm-and-a-search-part-2/>. Acesso em: 5 out.2021. Citado na página 3.

WIKPEDIA. *Algoritmo genético*, S.I, 2019. Disponível em: <https://pt.wikipedia.org/wiki/Algoritmo_gen%C3%A9tico>. Acesso em: 5 out.2021. Citado na página 2.