E-fólio A | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Introdução à Inteligência Artificial

CÓDIGO: 21071

DOCENTE: José Coelho

A preencher pelo estudante

NOME: Bruno Ricardo de Sá Ferreira

N.º DE ESTUDANTE: 2201529

CURSO: Engenharia Informática

DATA DE ENTREGA: 15 de Abril de 2024

Critérios	Auto-avaliação:
Análise do Problema (1 valor)	1
Identificação de Algoritmos (1 valor)	0.8
Resultados (2 valores)	1.7

Auto-avaliação de acordo com o avaliador: +0.1 na nota do e-fólio

Critérios de correção no enunciado.

Relatório de Introdução a Inteligência Artificial

Temos como objetivo, proteger o número máximo de famílias em determinadas zonas contra atividade criminal.

Para isso temos uma verba disponível para cada conjunto de mapas,(territórios), que são divididos em MXN partes iguais.

As verbas disponibilizadas permitem-nos adquirir delegacias, custando elas 4 euros, e deputados, custando cada 1 euro.

Temos 5 valores de verbas disponíveis para os determinados territórios que são:

Moedas de Ouro	Delegacias	Deputados	resto
4	1	0	0
8	2	0	0
8	1	1	3
12	2	2	2
12	3	0	0
12	1	5	3
16	4	4	0
16	2	6	2
16	3	3	5
20	6	0	3
20	1	13	3
20	2	10	2
20	4	4	20

Então, temos o número possíveis de combinações que podemos fazer com as verbas disponíveis, sabendo que o raio de proteção de cada delegacia aumenta consoante o número de deputados.

Deputados	Raio de proteção
0 (apenas o xerife)	1 (protege 3x3 zonas)
1	2 (protege 5x5 zonas)
5	3 (protege 7x7 zonas)
13	4 (protege 9x9 zonas)

O nosso objetivo para o projeto é então, utilizar um algoritmo de procura cega que nos permita, proteger o número máximo de famílias possíveis, dentro da nossa matriz(território), avaliando quais das possíveis combinações com os nossos budgets são as mais eficazes.

Para resolver o problema, utilizei a linguagem Python e estruturei o código da seguinte forma:

Dicionário de verbas com as combinações possíveis de acordo com os budgets, tendo em conta os raios.

```
verbas = {
    4: {
        1: ([(1,1)], [1])
    },
    8: {
        1: ([(1,1),(1,5) ], [1,1]),
        2: ([(2,4)], [2])
```

Inicializei as instancias com chaves para as verbas, e objetivos para que possamos combina-los com as combinações.

```
territorio = {
    1: {
        'matriz': [[0,7,0,0,4], [0,0,0,4,0], [1,0,0,0,0], [4,4,1,0,0], [6,0,3,4,4]],
        'verba': 4,
        'objetivo': [19, 20],
```

Estabeleci limites de dimensão do mapa, para que as delegacias não cheguem a uma zona em que a sua área exceda as dimensões da matriz, e também tive em conta os casos de sobreposição para que a mesma delegacia não calculasse a mesma família duas vezes.

```
# Movimentos possíveis: esquerda, baixo, direita, cima
movimentos = [(0, -1), (1, 0), (0, 1), (-1, 0)]

# Verifica se a posição está dentro dos limites do território
def dentro_do_mapa(x, y, matriz):
    return 0 <= x < len(matriz) and 0 <= y < len(matriz[0])

# Calcula o número de famílias protegidas para todas as delegacias
def calcular_todas_familias(delegacias, raios, territorio):
    todas_familias = 0
    protegidas = set()  # Conjunto para armazenar zonas já protegidas
    for (dx, dy), raio in zip(delegacias, raios):
        for x in range(dx - raio, dx + raio + 1):
            for y in range(dy - raio, dy + raio + 1):
                if dentro_do_mapa(x, y, territorio) and (x, y) not in protegidas:
                    todas_familias += territorio[x][y]
                    protegidas.add((x, y))
    return todas_familias
```

Para este exercício, testei dois algoritmos, busca em largura e busca em profundidade, ambas com limite, aos invés das outros estudados até agora, dado a natureza do problema e com base nas resoluções do professor. Testando os dois percebi que uma busca sem limite gera um loop infinito que consome bastante memória do computador, esgotando todos os recursos da máquina e que acaba por nem apresentar uma solução. E fiz questão de deixar, a busca em largura com limite para evidenciar o facto de que a nível de desempenho foi muito inferior a busca em profundidade com limite.

Por exemplo, na instancia 7, mesmo com um limite inferior a busca em largura gera mais estados e faz mais expansões que a busca em profundidade com limite. Essa ramificação tem influencia direta na performance do algoritmo que ficou clara através do tempo de execução.

Desta forma, baseei o meu trabalho em torno do algoritmo de profundidade com limite, e deixei a largura com limite precisamente para podermos fazer certas comparações e percebermos melhor
(Neste caso eu) como estes tipos de algoritmos funcionam

```
def busca_em_largura_limitada_simultanea(inicio, raio, matriz, objetivo, delegacias, raios,

def busca_em_profundidade_limitada_simultanea(inicio, raio, matriz, objetivo, delegacias, raios,
    estados_gerados, id_territorio, i, limite):
```

Em seguida implementei uma função de busca para os algoritmos em caso, um dicionário valores que cria uma relação entre o raio e moedas a serem gastam para calcularmos quanto sobrou e um relógio para calcular o tempo de execução.

```
def busca(estrategia, inicio, raio, matriz, objetivo, delegacias, raios, estados_gerados,
    id_territorio, i, limite=None):
     valores = {1: 4, 2: 5, 3: 9, 4: 17}
start_time = time.time()
```

De volta a analise do problema, a ramificação vai aumentando de acordo com o nível de profundidade que implementamos e também com o aumento das nossas instancias matrizes, isto porque, é-nos permitido movimentar todas as delegacias ao mesmo tempo, o que se reflete na criação de vários estados novos a partir do estado atual. Por exemplo, visto que temos 4 movimentos possíveis e que podemos ter até 5 delegacias numa instância. Dependendo da localização em que elas estejam iniciadas, podem ser criados 20 estados novos a partir de um só estado, então temos um problema em que de acordo com a instância pode ter uma ramificação bem alta, e em contra partida bem baixa também, quando temos instancias menores.
Também concluí que o nível de profundidade mais alto nem sempre traz a melhor solução, como se houvesse uma espécie de overfitting, quando a árvore é demasiado profunda pode deixar escapar algumas possibilidades. Nos casos da instancia 7, 8 e 10, a troca de profundidade foi essencial para alcançar os melhores resultados.

O algoritmo de procura em profundidade com limite de nível, é uma variação do algoritmo em profundidade que vai adicionar um limite para que não haja uma exploração infinita. O algoritmo começa com um nó inicial e um limite de profundidade, expande um nó não visitado, gerando todos os seus nós filhos, ou seja, todos os estados que podem ser alcançados a partir do estado atual através de uma ação válida, se qualquer um dos nós filhos é um nó objetivo então o algoritmo termina e retorna o caminho para esse nó objetivo. Se nenhum dos nós filhos é um nó objetivo, o algoritmo seleciona um dos nós filhos e repete o processo de expansão, Se o algoritmo atinge um nó sem nós filhos não visitados, ou todos os nós filhos excedem o limite de profundidade, ele faz um backtracking, ou seja, retorna ao nó pai e repete o processo com o próximo nó filho. O algoritmo continua esse processo até encontrar uma solução ou até que todos os nós dentro do limite de profundidade tenham sido visitados.

A Busca em Largura com Limite de Nível começa com um nó inicial e um limite de profundidade.O algoritmo expande um nó não visitado, gerando todos os seus nós filhos, ou seja, todos os estados que podem ser alcançados a partir do estado atual através de uma ação válida. Se qualquer um dos nós filhos é um nó objetivo então o algoritmo termina e retorna o caminho para esse nó objetivo. Se nenhum dos nós filhos é um nó objetivo, o algoritmo adiciona os nós filhos à fila de nós a serem explorados, desde que a profundidade do nó filho não exceda o limite de profundidade. O algoritmo então passa para o próximo nó na fila e repete o processo. O algoritmo continua esse processo até encontrar uma solução ou até que todos os nós dentro do limite de profundidade tenham sido visitados.