MAC0425/5910 Inteligência Artificial

Bruno Nunes Leal Faria[1]

[1]Instituto de Matemática e estatística (IME) – Universidade de São Paulo (USP)
Rua do Matão 1010 – 05508-090 – São Paulo – SP – Brasil

bnlf@ime.usp.br

Resumo. *O objetivo deste exercício-programa é implementar algumas técnicas de busca vistas em aula e ver sua aplicabilidade em casos mais gerais. Foi feito uma implementação de um ambiente de mina de ouro onde o objetivo era aplicar algoritmos de busca de forma a encontrar os ouros nas minas. Três algoritmos foram propostos, mas apenas dois foram finalizados nesta implementação.*

1. Introdução

Este programa foi desenvolvido em Python e dedica-se a resolver o problema da mina de ouro proposto para o Ep1 da disciplina. O programa encontra-se incompleto, podendo haver erros no método de busca DFS ou BFS. O algoritmo do A* não chegou a ser implementado, porém a descrição da heurística pode ser encontrada na seção 3.

Ambiente de desenvolvimento: Python 2.7.3, Windows 8 x64

Código fonte: /src/

Arquivos de teste: /src/data/

Método de execução:

$ cd ./src

$ python main.py $< arquivoteste >< tipobusca >$

Exemplo:

$ python main.py data/input.txt L

Tipos de busca:

- L: busca em largura
- P: busca em profundidade limitada
- A: busca A* (não implementado)

2. Descrição

O programa está dividido em vários módulos, conforme descrição abaixo:

Main.py: Ponto inicial do programa. Faz a checkagem dos parametros se estão corretos e leitura de entrada. A execução é sequencial e utiliza os parametros de entrada para prosseguir se validados com sucesso. Esta é a primeira parte da preparação do ambiente.

Environment.py: Descreve o ambiente em si. Possui o mapa da mina, locais onde estão os ouros, tamanho do ambiente, quantidade de ouro disponível e o tipo de busca que

será executado. Também possui métodos para traduzir o arquivo de input em uma matrix e posteriormente em um grafo de ações válidas.

Agent.py: Define parâmetros do agente que vai percorrer a mina. Guarda o caminho percorrido e o estado atual.

Search.py: Implementação propriamente dos mecanismos de buscas. Inicialmente estavam agregados no modulo do agente, mas para facilitar a manutenção foram colocados em um módulo próprio. Contém implementações para BFS, iDFS e a função que calcula o custo para a heurística criada para A*

3. Heurística A*

Embora o algoritmo não esteja implementado, a heurística foi preparada e encontra-se no arquivo Search.py. O método consiste em verificar a distância absoluta entre o estado atual menos o estado objetivo, no caso, a próxima pepita de ouro. Tal que:

$$num\_passos = abs(posicao\_do\_ouro - posical\_atual)$$

$$custo\_total = (custo\_passo * num\_passos) + recompensa\_ouro$$

Desta forma, tendo como referência o custo em linha reta até o objetivo, o agente pode tomar a melhor decisão de qual caminho tomar para que se tenha o menor gasto no final.

4. Análise de desempenho

Como a implementação não foi concluída, não foi possível fazer um paralelo entre a performance dos três algoritmos, mas algumas observações podem ser feitas a respeito do iDFS:

Quando limitado a valores muito baixos, pode não chegar no estado objetivo. Se não for estabelecido um limite, a recursão pode consumir muito CPU para ser executada e, para certos casos, demorar bem mais que a BFS. Testes onde não há muitos *muros* também podem fazer com que a execução deste algorítmo leve mais tempo que o da BFS para encontrar o estado objetivo, isto porque o número de ações possíveis no ambiente acaba sendo muito elevado.