QXD0037 - Inteligência Artificial

Laboratório 03 - Em Busca de uma Solução

Profa. Dra. Viviane Menezes

Data: 12.09.2018

1 Objetivos

Os objetivos desta atividade prática são implementar uma das três estratégias de busca (*busca em largura*, *busca de custo uniforme* ou *busca em profundidade*) para solucionar o problema do mapa rodoviário da Romênia.

2 Regras

- A atividade deve ser feita em dupla.
- Cada dupla deve entregar um único arquivo compactado (formato zip), contendo a implementação e a tabela com o resultado dos experimentos.

3 Em busca de uma solução

Depois de formular o problema, é preciso resolvê-lo. Uma solução é uma sequência de ações possíveis que começam a partir do estado inicial formam uma **árvore de busca**, enraizada no estado inicial, com nós correspondendo aos estados no espaço de estados do problema e os ramos correspondendo às ações do problema. Um nó na árvore de busca é uma estrutura composta pelos seguintes elementos: o estado no espaço de estados a que o nó corresponde; o pai na árvore de busca; a ação que foi aplicada para geração deste nó e; o custo de caminho que é o custo de sair do estado inicial e chegar até o nó.

A partir de cada nó, considera-se todas as possíveis ações aplicáveis a este nó e é feita uma **expansão** gerando nós filhos. O conjunto de todos os nós folhas disponíveis para a expansão é chamado de **borda**. O processo de expansão dos nós na borda continua até que uma solução seja encontrada ou não esxistam mais estados a expandir.

Todos os algoritmos de buscam compartilham essa estrutura básica. Eles variam na escolha do próximo nó a ser expandido.

3.0.1 Busca em Largura

O pseudocódigo do algoritmo `Busca-Em-Largura` é apresentado na Figura 3.0.3. Ele recebe como entrada um problema e retorna como saída a sequência de ações que leva o agente do estado inicial a um estado objetivo. A busca em largura implementa a `borda` como uma fila FIFO (*First In, First Out*)

```
01. BUSCA-EM-LARGURA(problema){
02.       nó.estado ← estado inicial do problema
03.       nó.custoDeCaminho ← 0
04.       se nó.estado é um estado objetivo então
05.            retorne solução
06.       borda ← uma fila FIFO contendo apenas nó
07.       explorados ← {}
08.       repita
09.            se borda está vazia então
10.                 retorne falha
11.            nó ← remover(borda)
12.            adicionar nó.estado a explorados
13.            para cada ação aplicável
14.                 filho ← criarNó(problema, nó, ação)
15.                 se filho.estado não está em explorados então
16.                      se filho.estado é objetivo então
17.                           retorne solução
18.                      inserir(filho, borda)
19. }
```

Figura 1: Pseudocódigo da busca em largura, adaptado de [Russell and Norvig, 2010].

3.0.2 Busca de Custo Uniforme

O pseudocódigo do algoritmo `BUSCA-DE-CUSTO-UNIFORME` é apresentado na Figura 3.0.2. Ele recebe como entrada um problema e retorna como saída a sequência de ações que leva o agente do estado inicial a um estado objetivo. A busca de custo uniforme implementa a `borda` como uma fila de prioridades, ordenada pelo custo de caminho.

3.0.3 Busca em Profundidade

O pseudocódigo do algoritmo `BUSCA-EM-PROFUNDIDADE` é apresentado na Figura **??**. Ele recebe como entrada um problema e retorna como saída a sequência de ações que leva o agente do estado inicial a um estado objetivo. A busca em profundidade implementa a `borda` como uma fila LIFO (*Last In, Last Out*), também conhecida como pilha.

```
01. BUSCA-DE-CUSTO-UNIFORME(problema){
02.       nó.estado ← estado inicial do problema
03.       nó.custoDeCaminho ← 0
04.       borda ← uma fila de prioridades, contendo apenas nó
05.       explorados ← {}
06.       repita
07.            se borda está vazia então
08.                 retorne falha
09.            nó ← remover(borda)
10.            se no.estado é objetivo então
11.                 retorne solução
12.            adicionar nó.estado a explorados
13.            para cada ação aplicável em nó
14.                 filho ← criarNó(problema, nó, ação)
15.                 se filho.estado não está na borda ou em explorados
16.                      inserir(filho, borda)
17.                 senão se filho.estado está na borda com maior custo
18.                      substituir nó borda por filho
19. }
```

Figura 2: Busca de custo uniforme, adaptado de [Russell and Norvig, 2010].

4 Implementação

Você deve implementar um dos algoritmos de `BUSCA-EM-LARGURA`, `BUSCA-DE-CUSTO-UNIFORME` e `BUSCA-EM-PROFUNDIDADE` para resolver o problema do mapa rodoviário da Romênia. Inclua no projeto anterior, pelo menos, a seguinte classe:

- `Node`: que descreve um nó na árvore de busca.

Seu programa deve receber a formulação do problema e devolver a sequência de ações necessárias para resolver o problema e o custo de caminho para chegar ao objetivo.

5 Experimentos

Você deve realizar experimentos considerando pelo menos 10 problemas do mapa rodoviário da Romênia com diferentes origens e destinos. Para cada problema, você deve anotar o tempo de execução para a busca escolhida.

5.1 Medindo o Tempo de Execução

Para medir o tempo de execução, use o comando `time` (do Sistema Operacional Linux) que executa o programa e depois disso mostra o(s) tempo(s) consumido(s). Por exemplo, o comando

```
01. BUSCA-EM-PROFUNDIDADE(problema){
02.       nó.estado ← estado inicial do problema
03.       nó.custoDeCaminho ← 0
04.       se nó.estado é um estado objetivo então
05.            retorne solução
06.       borda ← uma fila LIFO contendo apenas nó
07.       explorados ← {}
08.       repita
09.            se borda está vazia então
10.                 retorne falha
11.            nó ← remover(borda)
12.            adicionar nó.estado a explorados
13.            para cada ação aplicável
14.                 filho ← criarNó(problema, nó, ação)
15.                 se filho.estado não está em explorados então
16.                      se filho.estado é objetivo então
17.                           retorne solução
18.                      inserir(filho, borda)
19. }
```

Figura 3: Busca em profundidade em um grafo.

```
% time ./RomaniaMap Arad Bucarest

real    0m0.032s
user    0m0.030s
sys     0m0.010s
```

executa o programa `RomaniaMap` que fará uma busca para encontrar a sequência de ações necessárias para sair de Arad para Bucarest. No final, o programa time mostra que o programa levou 32ms para executar. Destes, apenas 30ms fora usados pelo programa, (em operação do usuário). O resto foi gasto em ciclos do sistema. O tempo que você deve considerar é o `user time`.

5.2 Construindo a tabela

A tabela deve informar a origem e destino de cada problema, o tempo gasto para resolver o problema e o custo total da solução.

Referências

[Russell and Norvig, 2010] Russell, S. and Norvig, P. (2010). *Artificial Intelligence*. Elsevier, 3a edition.