UNIVERSIDADE
FEDERAL DO CEARÁ

Discentes: Iago Oliveira Lima
Robert de Almeida Cabral
Docente: Profa. Dra. Maria Viviane de Menezes
Curso: Engenharia de Computação
6º Semestre

Inteligência Artificial

Laboratório 02 - Resolução de Problemas por Meio de Busca

16 de outubro de 2017

Introdução

Agentes reativos não são implementáveis em ambientes onde o número de regras condição-ação é grande demais pra se armazenar, nestes casos podemos implementar agentes baseado em objetivos, também chamados de agente de resolução de problemas. Um agente com várias opções imediatas que pode decidir o que fazer comparando diversas sequências de ações possíveis, este processo de procurar pela melhor sequência é chamado de agente de resolução de problemas por meio de busca.

Três tipos de buscas são usadas nesse relatório são elas: busca em largura (BFS), busca em profundidade (DFS) e busca por custo uniforme (UCS). Todos aplicados a grafos para que não ocorra ciclos e possivelmente ingressasse em loop infinito. O algoritmo de busca em largura (BFS) começa por um vértice raiz r especificado pelo usuário, o algoritmo expande r, depois expande os filhos de r e assim por diante até alcançar seu objetivo, este algoritmo numera os vértice em sequência, na ordem em que eles são descobertos. O algoritmo de busca em profundidade (DFS) explora todos os vértices e todos os arcos de um grafo dado e atribui um número a cada vértice. Já o algoritmo de busca por custo uniforme (UCS) leva em consideração o custo de sair de um nó até chegar a outro e sempre irá expandir o nó que obtiver menor custo.

Temos por finalidade mostrar qual algoritmo tem o melhor desempenho dado o problema do mapa rodoviário da Romênia. O problema consiste em dado um grafo e dois estados (uma origem e um destino) e deve retornar qual caminho percorrer para chegar ao objetivo.

O presente relatório esta estruturado da seguinte forma: uma descrição da formulação do problema, onde iremos formular o problema para podermos aplicar os algoritmos. Uma descrição de como os algoritmos de busca foram implementados. Uma descrição dos experimentos realizados com a coleta de 20 pontos aleatórios no grafo, também será descrito o hardware e o software onde os experimentos foram realizados. E uma conclusão que aponta qual algoritmo obteve desempenho superior no experimento.

Formulação do problema

Tomaremos como exemplo para a formulação o problema de sair do estado de Arad e chegar ao estado de Bucharest.

Estado Inicial

No estado inicial especificamos a configuração inicial do problema, no nosso caso "estar em Arad".

Ações

O nosso conjunto de ações é composto por somente uma função, denominada S(x,y) que quer dizer "sair de x e chegar y"

Modelo de transição

O modelo de transição está exemplificado na Figura 1.

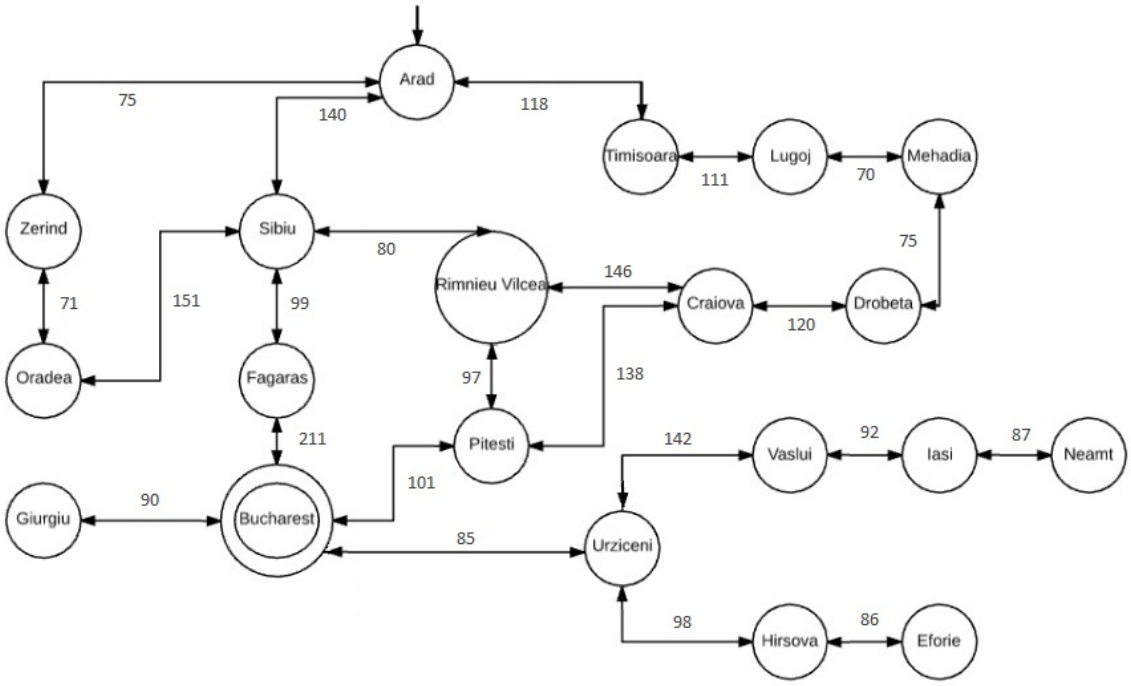

Figura 1: Modelo de transição do problema

Teste de Objetivo

No teste de objetivo, verificamos se o estado que vamos explorar em seguida é o estado objetivo, no nosso caso "Estar em Bucharest"é o nosso objetivo.

Custo de caminho

O custo de caminho que iremos considerar é a distância de um estado a outro, explicitado nas arestas do grafo da Figura 1.

Experimentos

Descrição dos experimentos

A tabela 1 descreve as reais condições de hardware e software onde os experimentos foram executados.

Processador	Intel Core i7-3537U 2.00GHz
Núcleos	4
Memória RAM	8GB
Disco Rigido	1TB
Sistema Operacional	Ubuntu Gnome 17.04

Tabela 1: Plataformas de hardware e software

Para realizar os experimentos, foram escolhidos 10 problemas aleatórios, onde cada problema tem uma origem e um destino. A tabela 2 contém cada experimento com sua respectiva origem e destino.

Problema	Cidade Origem	Cidade Destino
1	Oradea	Neamt
2	Timisoara	Fagaras
3	Drobeta	Iasi
4	Bucharest	Zerind
5	Mehadia	Oradea
6	Sibiu	Eforie
7	Arad	Urziceni
8	Giorgiu	Craiova
9	Vasliu	Lugoj
10	Arad	Bucharest

Tabela 2: Problemas experimentados

Análise dos resultados e Conclusão

Com isso, foram realizados os experimentos e após a compilação dos dados foi gerado um gráfico (Figura 2), onde a cor azul representa os experimentos utilizando o algoritmo de busca em largura, o vermelho de busca em profundidade e o preto de busca por custo uniforme. Pode-se observar que os números na horizontal representa cada problema, e os números na vertical o tempo de execução em milissegundos. Em uma rápida análise do gráfico, podemos concluir que a busca por custo uniforme (UCS) não é muito eficiente para este problema comparado aos outros dois algoritmos. Calculando a média aritmética do tempo de execução de cada algoritmo nos seus 10 experimentos, foi obtido o seguinte resultado:

Algoritmo	Média
BFS	0,200ms
DFS	0,137ms
UCS	0,238ms

Tabela 3: Problemas experimentados

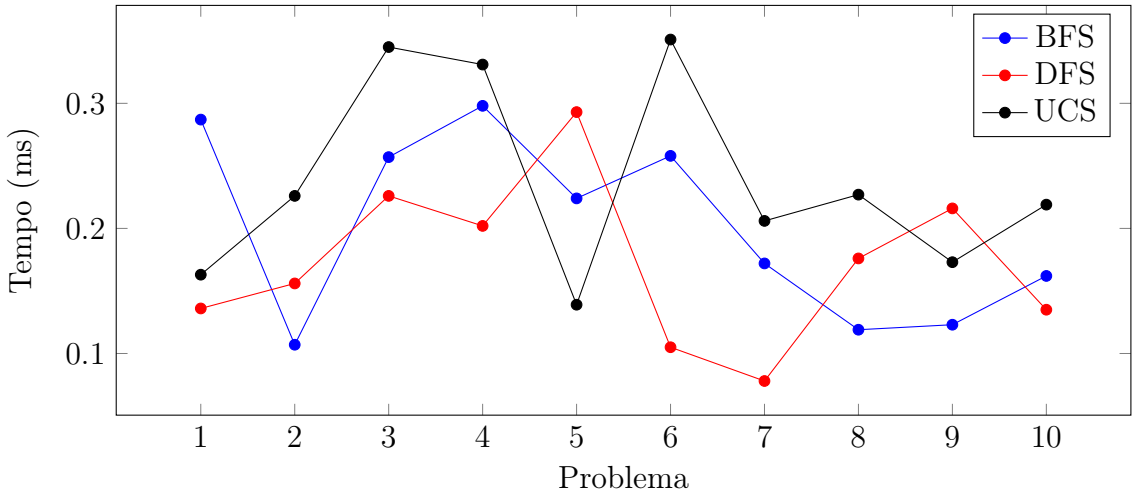

Figura 2: Gráfico comparativo entre os algoritmos

Portanto, dado os resultados observados na tabela 3, podemos concluir que o algoritmo de busca em profundidade (DFS) é o mais eficiente para resolução deste problema, porém ressaltamos que poderá haver variações nos resultados de acordo com a montagem do grafo e de qual nó é explorado inicialmente. Já os algoritmos de busca em largura e de custo uniforme ficaram um pouco menos eficiente.

Descrição da implementação dos algoritmos de busca

Os algoritmos de busca implementados e explicados neste relatório são todos aplicados a grafos, ou seja, algoritmos que não entram em ciclos ou loops infinitos. Para isso, cada um dos algoritmos tem seu vetor de explorados, onde se o nó do grafo é expandido, então ele é inserido no vetor e toda vez que um nó subsequente for expandido esse vetor irá ser percorrido a fim de verificar se o nó em questão já foi explorado antes ou não, caso tenha sido ele não será expandido novamente, desta forma nunca entrará em ciclos. O trabalho foi implementado utilizando a linguagem de programação C++. Todas as buscas tem classes em comum, são elas: Node, Action, State e Graph.

- Node
  - A classe Node tem seu estado, ação, custo de caminho e o Nó pai.
- State
  - Cada estado tem um inteiro que representa a cidade, pois todas as cidades são representadas em um ENUM.
- Action
  - Cada ação tem sua origem e destino.
- Graph
  - A classe Graph é basicamente onde foi implementado o grafo em que todas as cidades tem suas ligações

Busca em largura: Esta busca, como o próprio nome sugere, é uma busca que expande os nós formando uma árvore larga. A implementação dessa busca se deu pela classe Width_Search e o método público BFS, que recebe uma origem, destino e o grafo. Inicialmente o método cria o nó inicial, ou seja, a raiz da árvore, e então cria a fila que será a borda, vetor solução que armazenará cada nó visitado e o vetor de boleano dos explorados.

```
1 queue<Node> edge;
2 vector<Node> solution;
3 bool explored[SIZE_MAP];
```

Logo depois, o algoritmo entra em um loop até retornar a solução. Nesse loop, primeiramente é verificado se a borda está vazia, caso aconteça, o software retorna um erro. Então, é pego o topo da borda e verifica se está no vetor de explorados, caso esteja não é expandido. E por último é verificado cada vértice adjacente, para assim criar os seus filhos. Caso algum desses filhos estejam nos explorados, ele não será adicionado na borda. E caso o vértice seja o objetivo, o método retorna a solução.

Busca de custo uniforme: Esta busca, prioriza os filhos de menor custo. A implementação dessa busca se deu pela classe UniformCostSearch e o método público UCS, que recebe uma origem, um destino e um grafo. Inicialmente o método cria o nó inicial, ou seja, a raiz da árvore, e então cria a fila de prioridade - a difere em relação a busca em largura -, vetor solução que armazenará cada nó visitado, e o vetor de boleanos dos explorados.

```
1 priority_queue<Node> edge;
```

O código é bem similar a busca em largura, tendo apenas a criação de nós filhos. Pois o custo do novo nó é o custo usado anteriormente somado ao custo de caminho de uma certa origem ao destino.

Busca em profundidade: Por fim, a busca em profundidade, que se mostrou a melhor busca para a resolução do problema, pega um nó e expande até chegar ao seu objetivo ou em uma folha, caso chegue em uma folha, pega-se o próximo nó filho da raiz. A implementação é bem similar a busca em largura, com a única diferença que ao invés de uma fila FIFO, é usada uma pilha. E toda vez queque um nó for explorado um novo nó é pego do topo da pilha.