AGRESTE TOUR

Kaic de Oliveira Barros < kaicbarros@gmail.com>
Layla Joana Santos <layla.joana@gmail.com
Marcos Neto Santos <marcos_nto@hotmail.com>
Paulo Vitor Dos Santos Felipe<Paulo.vitor_18@hotmail.com>

Universidade Federal de Sergipe (UFS) - Curso de Sistemas de Informação – Campus Itabaiana Av. Vereador Olímpio Grande, S/N – Bairro Centro – CEP 495000-000 – Itabaiana - SE

RESUMO

Este artigo tem como principal objetivo demostrar uma ferramenta capaz de representar em aspectos práticos o método de busca A*(estrela) da área de Inteligência artificial, com base nesse algoritmo, tem-se como resultado um software chamado Agreste Tour que tem como objetivo mostrar ao usuário o melhor caminho possível para ele trafegar, levando em consideração o seu contexto que no caso do agreste tour é a região agreste do estado de Sergipe, que contem 14 cidades, as quais o usuário poderá escolher a cidade de onde irá sair e a cidade onde quer chegar e a ferramenta se encarregará de mostra-lo o melhor caminho.

ABSTRACT

Title: *"Agreste tour".*

*This article has as main objective to demonstrate a tool able to represent in practical aspects the method of search a * (star) of the area of Artificial Inteligência, based on this algorithm, has as a result a software called Agreste Tour that aims to show To the user the best possible way for him to travel, taking into account its context that in the case of the agreste tour is the rough region of the state of Sergipe, which contains 14 cities, which the user can choose the city from where he will leave and the City where you want to go and the tool will show you the best way.*

1 INTRODUÇÃO

A Inteligência artificial é um ramo da ciência que estuda o conjunto de paradigmas que pretendem justificar como um comportamento inteligente pode emergir de implementações artificiais, em computadores. Com a aprendizagem de IA várias opções imediatas pode decidir o que fazer comparando diferentes sequencias de ações possíveis, esse processo de procurar pela melhor sequência e chamando de busca.

Entre os objetivos da disciplina de Inteligência Artificial está à Busca com informação, sendo estudada em sala os seguintes algoritmos para tal (Busca gulosa pela melhor escolha e o A*(estrela), neste artigo iremos tratar do algoritmo A*(estrela), para melhor abstração do formalismo, conceitos e mecanismo de funcionamento, tem-se como auxilio didático um software que implementa o método A*.

Primeiramente, irá ser demonstrado os aspectos teóricos do algoritmo de busca A*. E demostra o desenvolvimento da ferramenta toda sua implementação caracterizando a estrutura do software em relação às classes de objetos utilizados e implementação de algoritmo. Na próxima seção será abordada uma breve introdução ao conceito de Busca A*.

2 A*(ESTRELA)

Na ciência da computação, A***** (pronunciado como "A estrela") é um algoritmo computacional que é amplamente utilizado na localização e cruzamento de gráficos, o processo de traçar um caminho eficientemente direcionado entre múltiplos pontos, chamados de nós. Goza de um uso generalizado devido ao seu desempenho e precisão. No entanto, em sistemas práticos de roteamento de viagens, geralmente é superado por algoritmos que podem pré-processar o gráfico para obter um melhor desempenho, embora outros trabalhos tenham encontrado A * superior a outras abordagens.

É um algoritmo de busca heurística, ou seja, leva em conta o objetivo para decidir qual caminho escolher e o conhecimento extra sobre o problema é utilizado para guiar o processo de busca. A função heurística (h), estima o custo do caminho mais barato (mínimo) do estado atual até o final.

Ex: Encontrar a rota mais curta entre duas cidades, o h (n) pode ser a distância em linha reta entre o nó n e o nó final.

Estratégia: Combina o custo do caminho g (n) com o valor da heurística h (n).

a) **F(n) = g(n) + h(n)**
   Onde:
   G(n) = custo do caminho do nó inicial até o nó n.
   H(n) = custo do caminho estimado do nó n até n nó final.
   F(n) = custo do caminho total estimado.

b) **Busca A* + heurística admissível**
   H(n) e admissível se para cada nó n, h(n) <= h*(n), onde h*(n) e custo verdadeiro de alcançar o estado objetivo a partir de n.

Na figura 1, está sendo demonstrado o funcionamento do algoritmo A*, tomando como exemplo um caminho possível da nossa ferramenta, onde o estado inicial é a cidade de Itabaiana e o destino é a cidade de Ribeirópolis, primeiramente o algoritimo expande todos os caminhos possíveis a partir do vértice inicial, que são Areia Branca, Moita Bonita, Frei Paulo e Campo do Brito, ou seja, de Itabaiana só tem caminho para esses lugares, aparte disso o algoritmo irá aplicar a função F(x) = G(x) + h(x) em cada um dos vértices, onde o g(x), corresponde ao valor do índice de acidentes normalizados + o valor da distância entre eles e o h(X) corresponde ao valor da heurística do ponto que atual até o destino, com esse cálculo feito verifica-se qual o vértice que se tem o menor valor da função F(x) e expande ele, o processo continua até que chegue no vértice final, chegando no vértice final se ainda tiver vértices sem expandir ainda continua a comparação pra verificar se ainda tem um menor que ele, pois se tiver um valor menor pode ser que tenha um caminho melhor ainda, caso não encontre nenhum menor, então aquele é o resultado é ótimo e o algoritimo finaliza, neste exemplo não expandimos os demais nós pois quando chegamos no final, não havia nenhum menor que ele. Sendo assim o melhor caminho para ir de Itabaiana a Ribeirópolis é : Itabaiana→ Moita Bonita →Ribeirópolis

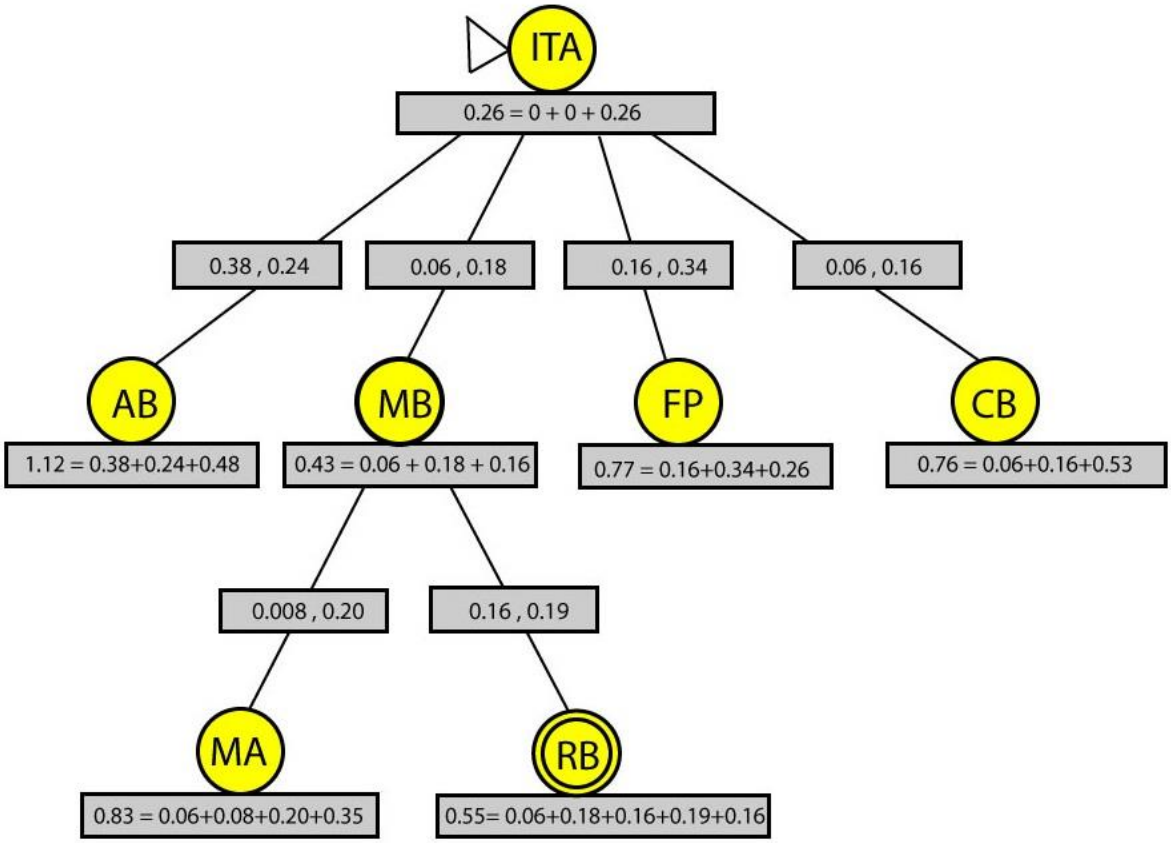

Figura 1– Arvore de Derivação (com valores arredondados) exemplificando o funcionamento do algoritimo A*

3 RECURSOS UTILIZADOS

Para o desenvolvimento do software, que se trata de um guia turístico da região agreste de Sergipe, o qual mostra o melhor caminho, ou seja, o caminho mais curto e seguro, para isso utilizamos, além das distâncias entre as cidades, os índices de acidentes das rodovias para garantir que o turista além de trafegar por um caminho mais rápido, esteja também trafegando pelo caminho mais seguro, com base nesse contexto, precisamos das distancias de todas as 14 cidades que compõem o agreste sergipano e os índices de acidentes das rodovias por onde será feita o trafego. Também precisamos das heurísticas que foi utilizada a heurística euclidiana (distancia em linha reta entre dois pontos). Utilizamos o site **distanciaentrecidades.com** para obter os valores das distâncias entre as cidades e a distância em linha reta. Para obtenção dos índices de acidentes das rodovias estaduais, solicitamos à CPRv(Companhia de polícia rodoviária estadual) localizada no município de Aracaju. As tabelas com todos os valores estão disponíveis para download no repositório: **https://github.com/Marcosnto/AgresteTour**

Para utilizar esses valores na aplicação precisamos fazer uma normalização, para manter todos em uma mesma escala, podendo então fazer a soma dos mesmos na função que é utilizada pelo algoritmo. Para realizar essa normalização aplicamos a todos os valores a formula:

$$VN(x) = x - min / max-min$$

Onde o **x**, assume o valor que se deseja normalizar , **min**, assume o valor mínimo da grandeza que se está tratando no momento e **max** o valor máximo.

4 O AGRESTE TOUR

O objetivo do **Agreste tour** é mostrar ao usuário o melhor caminho possível para que ele possa trafegar, para isso ele leva em consideração os índices de acidentes de cada rodovia por onde o usuário poderá passar e a distância em km de um ponto até o outro, e também a sua distância em linha reta (sendo nossa heurística), dessa forma a aplicação promoverá ao usuário o caminho mais curto e mais seguro.

Para implementação do mesmo foram criadas 5 classes, a Classe Vértice, Aresta, Heuristica, GerarGrafo e BuscaAestrela, sendo que o coração do programa está na classe BuscaAestrela, pois nela contém o método principal para a realização da busca que é o método aestrela.

No método *aestrela()* é passado como parâmetro um arrayList de vértices chamado grafo, que nele estará todos os vértices(cidades) do nosso mapa com todos os valores necessários, também é passado o vértice inicial (origem de partida) e o vértice final (destino). A partir deste ponto o método adiciona o vinicial ao conjunto de vértices abertos (ArrayList de vértices que vão ser visitados), guarda o *vInicial* na variável verticePai que é do tipo vértice para poder ter uma referência, e então entra no loop procurando o melhor caminho. A figura 2, mostra o método *aestrela()* completo.

```
public Vertice aestrela(ArrayList<Vertice> grafo, Vertice vinicial, Vertice vfinal) {
    // Adicionando o vertice Incial a lista de vertices abertos
    VerticesAbertos.add(vinicial);
    vinicial.calcular_F(ValorG(vinicial, vinicial), vfinal);
    Vertice verticePai = vinicial;

    while (!verticePai.getNomeCidade().equalsIgnoreCase(vfinal.getNomeCidade())) {
        VerticesAdjecentes = new ArrayList<Vertice>();
        ConjuntoDeVerticesAdjecentes(verticePai);
        for (int i = 0; i < VerticesAdjecentes.size(); i++) {
            VerticesAdjecentes.get(i).gAnt = verticePai.gAnt + ValorG(verticePai, VerticesAdjecentes.get(i));
            VerticesAdjecentes.get(i).calcular_F(VerticesAdjecentes.get(i).gAnt, vfinal);
            if (!VerticesFechados.contains(VerticesAdjecentes.get(i))) {
                VerticesAdjecentes.get(i).setVerticePai(verticePai);
            }
            VerticesAbertos.add(VerticesAdjecentes.get(i));
        }
        VerticesFechados.add(verticePai);
        VerticesAbertos.remove(verticePai);
        verticePai = menorValorF(VerticesAbertos);
    }

    return verticePai;
}
```

Figura 2 – Método aestrela()

O método precisa de outros métodos para chegar a solução entre eles está: *calcular_F*() que realizará o cálculo da função do algoritimo A* que é *F(x) = G(x) + H(x)*, nesse mesmo método é pego o valor da heurística.

```
public void calcular_F(float valorG, Vertice vf) {
    for (int i = 0; i < heuristica.size(); i++) {
        if (heuristica.get(i).getNomedestino().getNomeCidade().equalsIgnoreCase(vf.getNomeCidade())) {
            this.funcao = valorG + (heuristica.get(i).getValorHeuristica())*0.4f;
            break;
        }
    }
}
```

Figura 3 – Implementação do Método calcular_F

Também encontra-se os métodos: *ConjuntoDeVerticesAdjecentes(Vertice vi)* que retorna o conjunto de vértices adjacentes ao vértice que está sendo passado como parâmetro, ou seja, todos os caminhos possíveis aparte daquele ponto, *menorValorF()* que retorna o vértice que possui o menor valor da função f do conjunto de vértices que foi passado como parâmetro e o método *ValorG()* que retorna o valor de g entre dois vértices que foi passado como parâmetro, no nosso caso o valor de G(x) será a soma do índice de acidentes daquela rodovia com a distância em km entre eles.

```
public ArrayList<Vertice> ConjuntoDeVerticesAdjecentes(Vertice vi) {
    for (int i = 0; i < vi.arestas.size(); i++) {
        VerticesAdjecentes.add(vi.arestas.get(i).getNomeAresta());
    }
    return VerticesAdjecentes;
}

public Vertice menorValorF(ArrayList<Vertice> grafo) {
    Vertice menorF = grafo.get(0);
    for (int i = 1; i < grafo.size(); i++) {
        if(grafo.get(i).getFuncao()<menorF.getFuncao()){
            menorF = grafo.get(i);
        }
    }
    return menorF;
}
```

Figura 5 – Implementação dos Métodos ConjuntoDeVerticesAdjecentes() e menorValorF

```
public float ValorG(Vertice vi, Vertice vf) {
    float valorGsomado = 0;
    for (int i = 0; i < vi.arestas.size(); i++) {
        if (vi.arestas.get(i).getNomeAresta().getNomeCidade().equals(vf.getNomeCidade())) {
            valorGsomado = (vi.arestas.get(i).getIndice())* 0.5f  + (vi.arestas.get(i).getDistancia())* 0.1f;
        }
    }

    return valorGsomado;
}
```

Figura 5 – Implementação dos Métodos ValorG()

4.1 TELA PRINCIPAL

Na tela principal do **Agreste Tour,** o usuário poderá escolher a cidade de origem(de onde ele irá sair) e a destino(onde quer chegar) nos combos, após escolhidas as cidades clicando no botão Buscar irá aparecer no mapa bolinhas vermelhas nos pontos por onde ele terá que passar indicando que é o melhor caminho ele também poderá visualizar a rota na placa de transito, caso o usuário escolha duas cidades iguais para origem e destino a ferramenta exibirá uma mensagem informando que as cidades são iguais para que o usuário possa corrigir. Clicando no botão Limpar ele limpará a tela para a realização de uma nova consulta. Tem também o botão sobre, que especifica os desenvolvedores da ferramenta e o botão sair que fechará a tela da ferramenta.

Figura 5 – Tela Principal

Figura 6 – Demonstração de busca

Figura 7 – Sobre

5 CONCLUSÃO

Ao fim concluímos que o conhecimento do algoritmo A* na disciplina de IA proporcionou a criação da ferramenta Agreste Tour que irá contribuir para diversos usuários indicando a melhor rota que um turista possa tomar para viajar pelo agreste sergipano. Com rapidez e segurança. Além de percebemos a relevância do conteúdo e a associação do mesmo com a programação, colocando em pratica a teoria a nós apresentada durante o curso, por meio da implementação e reforçando a realização de trabalho em equipe.

Como próximos releases pretendemos aumentar a quantidade de dados utilizados na formula do método e também uma melhoria na interface para proporcionar uma melhor visualização para o usuário.

6 REFERÊNCIAS

http://www.distanciasentrecidades.com/

http://www.detran.se.gov.br/novo_estatistica.asp

Companhia de Policia Rodoviária Estadual – CPRv

Polícia Rodoviária Federal - PRF