Russia 2018 World Cup Route Using The Uniform Cost Method

Clovis Oliveira Mangueira[1], **Fabrícia de Jesus Santos**[1],
Paulo Victor dos Santos[1], **Weslley Henrique Santos**[1]

[1]Departamento de Sistemas de Informação
Universidade Federal de Sergipe (UFS) – Itabaiana, SE – Brasil

{clovis_jack, paulo.victor_18}@hotmail.com

fabriciacooper@gmail.com, weslley_campos@outlook.com

1. Introdução

No Brasil, o futebol é a paixão da nação e muitas vezes sendo considerado o esporte mais importante. Devido a essa paixão, a torcida brasileira é fiel ao seu time, desse modo, não seria diferente na copa do mundo. A copa do mundo está sendo realizada na Rússia, o qual é considerado o maior país de extensão territorial.

O jogos são realizados em províncias (estados) diferentes com grande distância, o que consequentemente, leva muito tempo para viagem. Para facilitar o deslocamento e poder acompanhar melhor os jogos, o projeto tem como objetivo apresentar o melhor caminho para os amantes de futebol. Utilizando o método de custo uniforme, os critérios escolhidos para apresentar o melhor caminho, foram: distância, tempo e número de pedágio de um percusso.

2. Busca de Custo Uniforme

A estratégia de busca uniforme é uma pequena modificação da estratégia de busca em largura. Na busca em largura primeiro expande-se o nó raiz, depois todos os nós gerados por esse, e assim por diante até que se chegue ao estado meta. Ou seja, todos os nós que estão numa profundidade d da árvore serão expandidos e visitados antes que os nós que estão numa profundidade d+1.

A estratégia de busca uniforme é basicamente a mesma coisa. Mas ao invés de pegar o primeiro nó expandido que está na lista aguardando processamento, o nó que possui o menor custo (g(N)) será escolhido para ser expandido. Se certas condições sempre forem cumpridas, garante-se que a primeira solução encontrada será a mais barata. Uma condição é a seguinte: o custo do caminho nunca deve ir diminuindo conforme avançamos por ele, em outras palavras, é importante que:

Listing 1. Pseudo-código para calcular a função g(n)

```
1 g(Sucessor)>=g(N)
2 em todos os nos N, g(N) e o custo conhecido de ir-se da raiz ate o nodu
```

3. Aplicação

A aplicação se baseou com dados fornecido por uma empresa russa Astra Log. A mesma é uma empresa que oferece serviços de transporte de mercadoria para vários países, principalmente a Rússia. Logo, foi possível gerar o grafo para ajudar no desenvolvimento do

projeto. Pode-se acompanhar o grafo na 1, onde cada ponto representa as cidades que acontecerão os jogos para a copa do mundo. Dentro dos dados fornecidos pela empresa Astra Log, estava a distancia de uma cidade para outra. Porém, para obter o tempo e o número de pedágio, realizou-se uma busca no Google Maps. Contudo, a aplicação tem

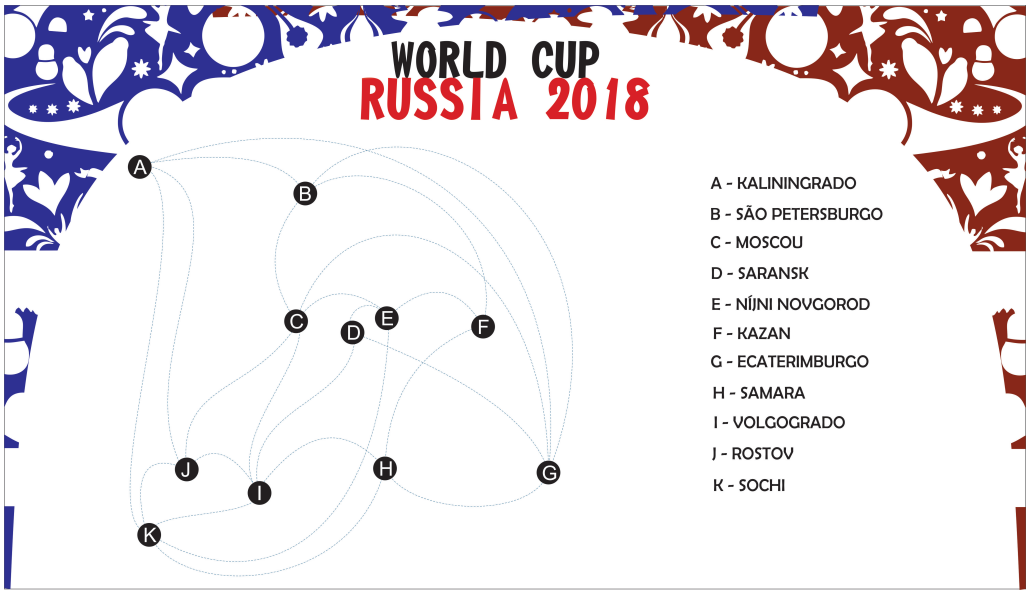

Figura 1. Grafo

por objetivo mostrar para ao usuário o melhor caminho entre as cidades que irão acontecer os jogos, onde dado um estado de partida e um de destino, o programa irá traçar um caminho com base nas informações definidas, que compõe a função G, sendo elas distância, tempo e a número de pedágios dentro da rota.

3.1. Implementação

Com os dados definidos, inicialmente, efetuou uma normalização dos dados para equilibrar os seus valores. Essa normalização se dá pela fórmula:

$$g(X) = \frac{x - min}{max - min}$$

. Onde:

1. x é o valor que deseja normalizar;
2. min representa o valor mínimo dentre os vértices;
3. max representa o valor máximo dentre os vértices;
4. g(X) equivale ao valor normalizado.

Em seguida, temos o método principal do projeto. Basicamente, dada um origem e destino, os nós visitados serão inseridos no array de forma ordenada. Enquanto o array frontier não for vazio, o elem guarda o valor do primeiro nó visitado. Percebe-se que, se a origem e destino tiverem os mesmos valores, o método para e retorna uma lista. Caso não tenha, o valor que foi atribuído para a variável elem, será expandido, fazendo uma busca dos nós adjacentes que foi inserido na matriz “map”. Considere a matriz map um grafo. Observando o método, um array do tipo Node foi criado para receber os nós adjacentes

do nó atual. A partir do *if* que testa a condição se esse array não esta vazio, ele busca por esses nós já ja foram visitados, caso não seja o custo será somado. Vale ressaltar que o método chamado *NodeComparator* realiza uma tarefa importante para a busca de custo uniforme, ela compara o custos com os outros nós para saber se é maior ou não.

Por fim, variáveis como ucsMaxNumOfNodesInMemory e ucsNumOfNodesGenerated foram criados como forma de controle para saber o número de nós que estão em memória, ou seja, nós abertos prontos para ser expandidos e para saber quantos nós foram expandidos.

Listing 2. Método Principal do Projeto

```
public ArrayList<Node> ucs(Node src, Node dest) {

    Comparator<Node> comparator = new NodeComparator();
    PriorityQueue<Node> frontier =
    new PriorityQueue<Node>(MAX_CITIES, comparator);
    frontier.add(src);

    ArrayList<Node> visited = new ArrayList<Node>();

    while (!frontier.isEmpty()) {
        Node elem = frontier.remove();
        visited.add(elem);
        if (elem.city == dest.city)
        break;

        elem.expandNode(findAdjNodes(elem.city),
        findAdjEdges(elem.city));
        ucsMaxNumOfNodesInMemory++;
        ArrayList<Node> childNodes = elem.adjNodes;

        if (!childNodes.isEmpty()) {
            for (int i = 0; i < childNodes.size(); i++) {
                Node child = childNodes.get(i);
                if (!visited.contains(child)
                && !frontier.contains(child)) {
                    child.updateCost(elem.getCost() +
                    map[elem.city][child.city]);
                    System.out.println("Custo do no: " +
                    child.city +
                    " e: " + child.getCost());
                    frontier.add(child);
                }
            }
        }
    }
    ucsNumOfNodesGenerated = visited.size();
    return visited;
```

```
38 }
```

3.2. Interface

Para saber qual melhor caminho, o usuário deve selecionar sua origem e seu destino comboBox, localizado no canto direito da tela. A partir desse momento, ao clicar no botão iniciar, a aplicação irá apresentar o melhor caminho, mostrando os pontos das cidades a serem passadas. Logo a baixo segue a imagem.

4. Conclusão

O presente projeto teve por objetivo aplicar o algoritmo de busca de custo uniforme sobre o problema citado anteriormente. O maior propósito é ganhar conhecimento sobre a área de Inteligência Artificial e ter suporte para aplicar os seus conceitos. Contudo, o método de busca do custo uniforme, não realiza a busca de uma forma ótima, já que ele tem um grande custo computacional, pois é necessário percorrer todos os nós da árvore, uma vez que o menor caminho mostrado por ele, não será de fato o menor.