UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Christian Zirke Osta

ANÁLISE COMPARATIVA DE ALGORITMOS DE CAMINHO DE CUSTO MÍNIMO APLICADO EM REDE DE FIBRA ÓPTICA

Florianópolis

2019

RESUMO

Um número crescente de operadoras de telecomunicações estão percebendo a necessidade de migrar para redes ópticas, seja pela qualidade do sinal, trazendo estabilidade frente aos intempéries, seja entregando uma internet com maior velocidade aos clientes que descobriram a alta qualidade de vídeos ou serviços completamente em nuvem ou pela maior segurança no tráfego dos dados se comparado aos canais wireless de longa distancia (rádio). Entretanto existe uma dificuldade no momento de fazer o planejamento de um novo trecho de cabeamento, visto que é necessário utilizar a rede de posteamento elétrico já existente. Este trabalho tem como objetivo realizar uma análise comparativa entre dois algoritmos utilizados para busca de caminho de custo mínimo, sendo eles o algoritmo de Dijkstra e A*, de forma a encontrar uma rota viável para o novo cabeamento, em tempo hábil e de com menor custo computacional, reduzindo ao máximo o custo das empresas em relação à possíveis rotas de cabos, e avaliando casos em que deve ser dada preferência a um algoritmo específico.

Palavras-chave: Dijkstra. A*. Floyd-Warshall. Problema de menor caminho. Rede de fibra óptica

ABSTRACT

A growing number of telecom operators are realizing the need to migrate to optical networks, either by signal quality, stability in the face of the weather, or delivering a faster internet to customers who have discovered the high quality of videos or services completely in the cloud or greater security in data traffic compared to wireless long distance (Radio) channels. However, there is a difficulty when planning a new section of cabling, since it is necessary to use the existing electric posting network. This work aims to perform a comparative analysis between two algorithms used to search for minimum cost path, being the algorithm of Dijkstra and A *, in order to find a viable route for the new cabling, in a timely manner and with less computational cost, reducing to a maximum the cost of the companies in relation to the possible routes of cables, evaluating cases in which preference should be given to a specific algorithm.

Keywords: Dijkstra. A*. Floyd-Warshall. Shortest Path Problem. Optical fiber network

SUMÁRIO

1 INTRODUÇÃO

As redes de dados vêm se aprimorando para funcionar da forma mais rápida e confiável possível. E, no universo, não há nada mais rápido que a luz, que viaja a 300 milhões de quilômetros por segundo. É natural, portanto, que essa velocidade seja empregada para transmissões de dados. A fibra óptica é uma tecnologia que tem conquistado o mundo, sendo muito utilizada principalmente na área de telecomunicações. As redes de fibra óptica já são uma realidade no Brasil, sendo utilizadas para levar dados de forma rápida, porém o seu alto custo de implantação e manutenção tem tornado a sua disseminação um plano futuro das operadoras de telecomunicações. Um dos problemas enfrentados é após uma rede inicialmente pronta, ocorre a necessidade de expansão da rede. Mas como expandir de forma a ter o menor custo possível com os novos elementos? Este trabalho surge justamente para auxiliar na manutenção e expansão da rede, reduzindo o custo de implantação de novos cabos, ao encontrar uma rota de cabeamento mínimo dado duas caixas com posições já definidas, utilizando a teoria de grafos e aplicando algoritmos para busca de caminho de custo mínimo. Serão apresentados e estudados dois algoritmos que calculam o caminho de custo mínimo em um grafo, sendo eles o algoritmo de Dijkstra e A*, para posteriormente realizar uma comparação sobre os resultados dos algoritmos aplicados em uma rede real.

1.1 CONTEXTUALIZAÇÃO

Uma rede de fibra óptica é composta principalmente por três elementos: postes, caixas e cabos. Cada qual será definido a seguir:

- **Postes:** É comum que empresas prestadoras de serviço de telecomunicação (por exemplo: telefonia fixa, TV a cabo, etc..) entrem em acordo com empresas locais do ramo de distribuição de energia para liberação de uso do posteamento existente, sendo de propriedade da concessora de energia. O poste utilizado para conter tanto as caixas quanto o cabo é de propriedade da empresa de energia, dificultando a manipulação de posição dos postes de forma a favorecer a empresa.

- **Caixas:** As caixas em uma rede de fibra têm uma função bem simples, mas importante, de permitir a abertura do cabo e a

manipulação da fibra em um ambiente seguro. Sendo a fibra óptica um objeto de extrema sensibilidade, a sua exposição ao tempo aberto seria um grande problema, certamente danificando a fibra de forma permanente em pouco tempo, gerando custo para a empresa e desconforto para um possível cliente. É dentro da caixa que ocorre o redirecionamento de uma fibra para um (ou vários) clientes.

- **Cabos:** O cabo tem como finalidade transportar a fibra entre dois pontos da rede, seja entra duas caixas ou a conexão entre uma caixa e um cliente. É possível passar um cabo de um poste a outro somente se já existir um cabeamento de energia entre esses dois postes, dificultando o lançamento de novos cabos devido ao trajeto limitado.

Pelo fato da empresa de telecomunicação não ter controle sobre o posicionamento dos postes, resta a eles somente o posicionamento de caixas e lançamento de cabos. Este trabalho surge como um estudo para auxílio das empresas de telecomunicação, estudando e apresentando algoritmos que permitam descobrir caminhos de menor custo e com um bom desempenho, seja o resultado utilizado para um caminho de fibra de uma caixa até um imóvel de um cliente, ou o possível trajeto de um novo cabo entre duas caixas.

1.2 OBJETIVOS

Os objetivos geral e específicos deste trabalho são assim definidos:

1.2.1 Objetivo Geral

Este trabalho tem como objetivo geral a realização de uma análise comparativa sobre o desempenho dos principais algoritmos para busca de caminho de custo mínimo em um grafo representado por uma rede de posteamento real.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

Objetivo 01: Compreender a conceitualização de grafos a fim de realizar a modelagem do trabalho como um problema em grafos.

Objetivo 02: Investigar quais são os algoritmos de busca de caminho mínimo adequados ao problema

Objetivo 03: Implementar os algoritmos, utilizando como entrada de dados o grafo representando a rede real.

1.3 MÉTODOS DE PESQUISA

O desenvolvimento do trabalho é realizado através de três fases vinculadas aos objetivos propostos, são estas: estudo da fundamentação teórica, implementação dos algoritmos propostos e por último, a aplicação de cada algoritmo em uma rede real para avaliação dos resultados.

Fase 01. Estudo da fundamentação teórica: Nesta fase é realizada o estudo referente a fundamentação teórica sobre os algoritmos que serão utilizados. Deve ser estudado o funcionamento de cada um deles, possíveis cenários de aplicação, vantagens e desvantagens de cada um.

Fase 02. Implementação dos algoritmos: Nesta fase serão implementados os algoritmos de Dijkstra, descrito na seção 2.3.1, e A*, descrito na seção 2.3.2. Será feita um estudo para escolher uma linguagem adequada para implementação, assim como análise de ambiente que os testes serão executados.

Fase 03. Aplicação e avaliação dos resultados: Esta etapa consiste em realizar a aplicação dos algoritmos implementados na fase 02, gerando uma quantidade de dados suficiente para uma análise significativa, e então realizar uma comparação e avaliação dos resultados apresentados por cada um dos algoritmos.

1.4 APRESENTAÇÃO DO TRABALHO

No capítulo 2 será apresentada a fundamentação teórica do trabalho, contando com uma breve introdução sobre conceitos básicos de grafos, descrição dos algoritimos escolhidos e uma descrição mais detalhada do problema. No capítulo 4 será visto detalhes sobre o desenvolvimento do trabalho, implementação dos algorimtos, heurísticas utilizadas e metodologia para extração de resultados. O capítulo 5 contém a apresentação e análise dos resultados, seguido pelo capítulo 6, realizando a conclusão do trabalho em relação aos objetivos definidos previamente.

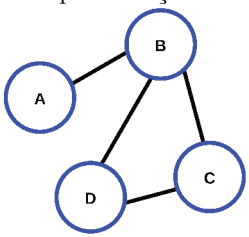
2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados os algoritmos utilizados neste trabalho. Para isto, inicialmente definem-se alguns conceitos da área de grafos, visto que a rede de posteamento pode ser modelada com um grafo, e também apresentado um problema clássico na área de grafos: o problema do caminho de custo mínimo. Por fim, é apresentado o problema e como ele pode ser modelado como um grafo. (CORMEN, 2009).

2.1 GRAFOS

A teoria dos grafos é um ramo da matemática que estuda as relações entre os objetos de um determinado conjunto. Para tal são empregadas estruturas chamadas de grafos, representados pela expressão $G(V, E)$, onde G representa o grafo, composto por um conjunto não vazio V de vértices, também conhecidos como nodos, e um segundo conjunto E, composto pelas arestas do grafo, onde $E = V \times V$, que podem ser um par ordenado (v, w), no caso de um grafo dirigido, ou um conjunto $\{v, w\}$, no caso de um grafo não dirigido, ambos conceitos definidos posteriormente, tal que v e $w \in V$. Os grafos são geralmente representados graficamente da seguinte maneira: é desenhado um círculo para cada vértice, e para cada aresta é desenhado um arco conectando suas extremidades, conforme vemos na figura 1.

Figura 1 Representação de um grafo.



No exemplo, temos um grafo não dirigido com quatro nodos, representados pelos círculos e quatro arestas, representadas pelas ligações entre os vértices. Uma aplicação real para um grafo seria, por exemplo, modelar um mapa, onde cada nodo seria uma cidade, e suas arestas seriam possíveis rotas entre as cidades. Uma definição de um grafo

$G(V, E)$ para o problema citado se daria como no exemplo 1 abaixo:

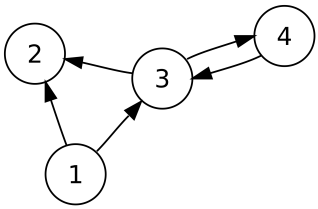
Exemplo 1
Considerando o conjunto de cidades V, composto por c_1, c_2, c_3, ..., c_n como sendo os nodos de um grafo G, o conjunto de arestas de G seria, por exemplo, o conjunto E, onde cada elemento de E é a representação de uma estrada que faça a ligação entre duas cidades contidas em V.

2.1.1 Conceitos sobre grafos

Existem diversos conceitos que caracterizam um grafo e suas propriedades. Para o contexto deste trabalho, os mais importantes são:

- **Ordem:** A ordem de um grafo é dada pelo tamanho de seu conjunto de vértices. Exemplo o grafo da figura 1 é um grafo de ordem 4.
- **Adjacência:** Em um grafo simples, por exemplo o grafo da figura 1, dois nodos A e B são considerados adjacentes se existe uma aresta $a \in E$ que conecta os dois vértices. Outros exemplos de adjacência no mesmo grafo são os nodos B e D, B e C, e D e C.
- **Grafo dirigido:** Um grafo dirigido é um grafo onde a relação entre os nodos pode não ser simétrica. A figura 2 é um exemplo de um grafo dirigido, pois suas arestas possuem um sentido, como na relação entre o nodo 1 e o nodo 2, onde a aresta implica que a relação é $1 \rightarrow 2$

Figura 2 – Representação de um grafo dirigido.



- **Grafo não dirigido:** Em um grafo, a relação entre os nodos é simétrica, pois suas arestas não possuem um sentido. O grafo representado na figura 1 é um exemplo de um grafo não dirigido,

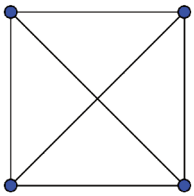
pois suas arestas não possuem sentido, como pode-se observar na relação entre A e B por exemplo.

- **Cadeia:** Uma cadeia em um grafo é uma sequência qualquer de arestas adjacentes que ligam dois vértices. Na figura 1 temos diversos exemplos de cadeia, uma delas é representada pela sequência de vértices A, B, C e D.

- **Ciclo:** Um ciclo é uma cadeia em que o vértice inicial e final são iguais, e não ocorre a repetição de nenhuma aresta. Um ciclo presente no grafo da figura 1 é a cadeia representada pelos vértices B, C e D.

- **Grafo completo:** Um grafo é considerado um grafo completo quando existe uma aresta que conecta todos os pares de vértices do grafo. O grafo da figura 3 é um exemplo de um grafo com quatro nodos, considerado um grafo completo.

Figura 3 Representação de um grafo completo.

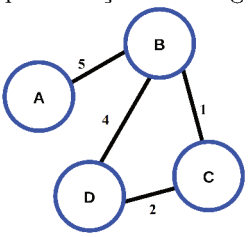


- **Grafo esparso:** Grafos em que a quantidade média de arestas por vértice é consideravelmente menor que a quantidade em um grafo completo de mesma ordem são chamados de grafos esparsos. Por exemplo um grafo com 1000 nodos, e cada nodo com uma média de 4 arestas.

- **Grafo valorado:** Um grafo é chamado de grafo valorado quando existe uma função que atribui um valor a seus vértices ou arestas. O grafo da figura 1 por exempo não é um grafo valorado, pois não existe nenhum valor associado a elas. Agora se for considerado o grafo da figura 4, com a seguinte definição:
  $\boldsymbol{V = \{c \mid c \text{ é uma cidade }\}}$
  $\boldsymbol{E = \{(v, w, d) \mid \text{Existe uma rota que liga a cidade } v \text{ com a cidade } w \text{ de comprimento } d\}}$

pode-se ver que existem valores associados a cada uma das arestas. O significado valor depende da definição do grafo, podendo ser tanto uma distância entre uma cidade e outra, quanto o tempo de trajeto, dependendo do problema que se quer resolver com a modelagem.

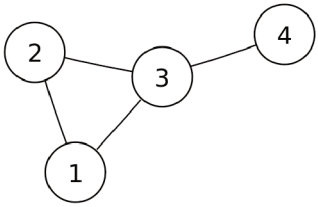
Figura 4 Representação de um grafo valorado.



2.1.2 Representando um grafo

Existem diversas maneiras de se representar um grafo, cada um com suas vantagens e desvantagens. A melhor forma de representação varia sempre de acordo com o problema que se quer resolver. Serão utilizados três critérios na análise: A quantidade de memória ocupada, o tempo levado para descobrir se existe adjacência entre dois vértices, e quanto tempo é levado para encontrar todos os adjacentes de um nodo. Para melhor exemplificação, de cada forma de estrutura será utilizado o grafo exemplo da figura 5 abaixo.

Figura 5 Grafo com nodos com representação numérica.



- **Lista de arestas:** A primeira forma de representação é a mais simples das três estruturas apresentadas. É simplesmente uma

lista onde cada elemento da lista representa uma aresta, com o conjunto $\{v, w\}$ representando quais nodos são ligados pela aresta. A representação do grafo da figura 5 ficaria da seguinte forma: ({1,2}, {1,3}, {2,3}, {2,4}). Por ser simplesmente uma lista com as arestas, o espaço ocupado por essa representação é proporcional ao número de arestas no grafo, $O(E)$. Tanto para descobrir uma adjacência em específico quanto para descobrir todos os nodos adjacentes de um vértice (considerando que a lista não tem nenhum tipo de ordenação), é necessário um percorrimento completo da lista, sendo assim, o tempo de execução de ambas as operações também é proporcional ao conjunto das arestas, portanto o tempo de execução é $O(E)$.

- **Matriz de adjacência:** Em um grafo com n vértices, a matriz de adjacência é uma matriz $n \times n$, preenchida com 1s e 0s, onde um valor presente na linha i e coluna j recebe 1 para caso exista uma aresta de par (i, j), e recebe 0 para caso não exista a aresta. A matriz de adjacência do grafo da figura 5 está representada abaixo.

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

  A matriz de adjacência, por ser de tamanho $n \times n$, ocupa um espaço proporcional ao quadrado do número de vértices, logo a complexidade espacial é $O(n^2)$. Entretanto, ocorre uma melhora em relação ao tempo para buscar as adjacências e descobrir se uma aresta está presente. Para buscar todos os adjacentes de um vértice, basta acessar a linha relativa a ele na matriz, portanto a complexidade desse procedimento é constante, $O(1)$. E para verificar a existência de uma aresta (i, j), basta observar o valor presente na posição da matriz, também em tempo constante, assim o tempo de execução também é $O(1)$. A desvantagem da matriz de adjacência é o fato de sempre ocupar V^2 de espaço, mesmo para um grafo esparso, onde a maioria das entradas vai ser 0. (WILSON, 1970)

- **Lista de adjacência:** É uma combinação entre a lista de arestas e a matriz de adjacência. A ideia é juntar o acesso constante ao valor de um nodo, presente na matriz, com a complexidade espacial

reduzida da lista de arestas. Portanto, tem-se uma primeira lista, cada posição representando um nodo do grafo, e com tempo de acesso constante, $O(1)$, resolvendo assim o problema para buscar todos os adjacentes. Cada posição i da lista possui uma segunda lista, que detêm todos os nodos que por sua vez têm uma aresta adjacente ao nodo i. Para verificar a existência de uma aresta específica, é necessário porém uma iteração completa sobre essa segunda lista, que no pior caso terá tamanho igual ao número de vértice, logo temos complexidade do procedimento proporcional ao número de nodos do grafo, $O(V)$. Como complexidade espacial, têm-se uma primeira lista de tamanho V, e a soma total das listas secundários é igual ao número de arestas do grafo, V. A complexidade espacial é $O(V + E)$, ou simplesmente $O(E)$, visto que o valor de E é no mínimo igual ao valor de $V - 1$ caso todos os nodos tenham pelo menos uma aresta. A lista representando o grafo 5 é a seguinte: ({2, 3}, {1, 3}, {1, 2, 4}, {3})

2.2 PROBLEMA DE CAMINHO DE CUSTO MÍNIMO

Considere a necessidade de realizar o trajeto Florianópolis - Chapecó, mas utilizando a rota mais curta possível. Como seria possível determinar tal rota?

Uma forma seria enumerar todas as rotas possíveis, com suas distâncias, e escolher a menor rota. Essa seria a forma mais simples, porém teria como resultado uma quantidade grande de rotas, das quais a grande maioria é certamente uma escolha ruim, por exemplo uma rota que saia do estado, passa por Curitiba e depois retorna para Chapecó, é uma rota possível, mas certamente não é uma das melhores.

Por sorte esse é um problema conhecido e já muito estudado, conhecido como problema de caminho de custo mínimo (*Shortest Path Problem*), e muitos algoritmos foram desenvolvidos com o objetivo de resolver esse tipo de problema. Ele pode ser dividido em três subproblemas:

- ***Single-Source (Single-Destination):*** Esse tipo de problema consiste em descobrir todas as rotas mínimas a partir de (ou até) um determinado nodo. No exemplo citado, seria como calcular a menor rota entre Florianópolis e todas as outras cidades brasileiras, ou de todas as cidades até Florianópolis. Dois algoritmos conhecidos que resolvem esse tipo de problema são o Algoritmo de Dijsktra, que será melhor explicado na seção 2.3.1 deste tra-

balho, e o Algoritmo de Bellman-Ford. Ambos funcionam de forma similar, a partir de um nodo fonte, encontram o caminho de custo mínimo desse nodo para todos os outros do grafo. A grande diferença entre eles é que Dijkstra não pode ser aplicado em grafos com arestas de peso negativo, fato possível com o algoritmo de Bellman-ford, porém abrindo mão de desempenho, sendo executado em $O(V.E)$, enquanto Dijkstra é executado em $O(E + V. \log V)$.(CORMEN, 2009)

- ***All-Pairs:*** A estratégia para resolução desse problema é calcular as rotas de caminho mínimo entre todos os nodos, ou seja, para cada nodo, calcula-se a menor rota entre este e todos os outros. Seguindo o exemplo das cidades, seria como calcular a menor rota entre todas as cidades brasileiras. Um exemplo de algoritmo que resolve esse tipo de problema é o algoritmo de Floyd-Warshall, que não será abordado neste trabalho. Floyd-Warshall encontra o caminho de custo mínimo dentre todos os pares de nodos do grafo, utilizando caminhos parciais para encontrar o de menor custo e é executado com complexidade $O(n^3)$.(CORMEN, 2009)

- ***Single-Pair:*** Consiste em encontrar a menor rota entre dois pontos apenas. Seria como calcular apenas a menor rota entre Florianópolis e Chapecó. Um algoritmo que resolve essa classe de problema é o A*, que será melhor detalhado na seção 2.3.2. A* é um algoritmo que utiliza de uma heurística para determinar qual é o nodo seguinte no caminho de custo mínimo, encontrando por fim um caminho de custo mínimo entre um par de vértices no grafo. A complexidade e otimização do resultado do algoritmo dependem da heurística que será escolhida.(CORMEN, 2009)

2.3 ALGORITMOS EM GRAFOS

Nesta seção serão apresentados os algoritmos de Dijkstra, descrito na seção 2.3.1, por ser considerado por muitos autores como o mais eficiente em busca de caminho de custo mínimo, e o algoritmo de busca A*, descrito na seção 2.3.2, por ser um algoritmo que resolve a classe de problema *Single-Pair*, classe em que o problema estudado melhor se encaixa.

2.3.1 Dijkstra

O algoritmo de Dijkstra resolve o problema da classe *Single-Source* em grafos valorados e que possuam todas as arestas com peso ≥ 0. Para o caso de haverem arestas de peso negativo, uma alternativa pode ser a utilização do algoritmo de Bellman-Ford. Ambos os algoritmos utilizam de uma técnica conhecida como "relaxamento", que consiste em manter dois atributos, d e π, presentes em todos os nodos, para representar a distância mínima estimada entre o vértice inicial e o nodo em questão, e o nodo anterior nesse caminho, respectivamente. Para utilização dessa técnica, é necessário um pré processamento, de complexidade temporal (V), INITIALIZE-SINGLE-SOURCE, conforme o algoritmo 2.1.

Algoritimo 2.1 – Pré processamento de *Single Source*

```
1 INITIALIZE-SINGLE-SOURCE(G, s)
2     for each vertex v ∈ G.V
3         v.d = ∞
4         v.π = NIL
5     s.d = 0
```

O procedimento "*INITIALIZE-SINGLE-SOURCE*"tem como parâmetro um grafo G e um vértice inicial s, utilizado em procedimentos para busca de caminho de custo mínimo para soluções de problemas na classe "*Single-Source*"e consiste em marcar todos os vértices do grafo com caminho de custo mínimo nulo (*NIL*) e distância mínima como sendo ∞, exceto pelo vértice inicial s, que possui distância mínima igual a 0.

Algoritimo 2.2 – Processo de "relaxamento"

```
1 RELAX(u, v, w)
2     if v.d > u.d + w(u,v)
3         v.d = u.d + w(u,v)
4         v.π = u
```

O processo de "relaxamento", com complexidade $O(1)$, consiste em testar e decidir se a distância mínima encontrada até o momento pode ser melhorada, e se for possível, atualizar os valores de d e π. O algoritmo de relaxamento, detalhado em 2.2, recebe três parâmetros, sendo eles um nodo anterior u, o nodo atual v, e uma função para determinar o peso das arestas (distâncias) w. É verificado se a distância

estimada até o momento entre $s \rightarrow v$, presente em $v.d$, é maior do que a distância estimada entre $s \rightarrow u$ incrementado da distância entre u e v encontrado usando a função $w(u, v)$. Em caso positivo, então foi encontrado uma nova rota mínima $s \rightarrow t$, e os valores são atualizados.

Algoritimo 2.3 – O algoritmo de Dijkstra

```
1 DIJKSTRA(G, w, s)
2     INITIALIZE-SINGLE-SOURCE(G, s)
3     S = ∅
4     Q = G.V
5     while Q ≠ ∅
6         u = EXTRACT-MIN(Q)
7         S = S ∪ {u}
8         for each vertex v ∈ G.Adj[u]
9             RELAX(u,v,w)
```

O algoritmo de Dijkstra, descrito no algoritmo 2.3, recebe três parâmetros, sendo eles o grafo G, a função retornando a distância entre cada par de nodos w, e um vértice inicial (fonte) s, que será a base para cálculo das distâncias. Os primeiros procedimentos servem como pré processamento, inicializando as distâncias e antecessores com a invocação do procedimento "*INITIALIZE-SINGLE-SOURCE*", conforme explicado anteriormente, e a inicialização de duas variáveis: um conjunto S, inicialmente vazio, e uma priority queue Q, contendo os vértices (utilizando como valor de comparação o atributo de distância d). Após isso, ocorre uma iteração sobre Q, removendo um nodo u, sendo u o nodo de menor distância até s presente em Q, e adiciona u em S. Por último, é percorrido todos os nodos adjacentes a u, e para cada nodo é invocado o procedimento "*RELAX*", passando como parâmetro o nodo u, o adjacente v e a função de distância w, fazendo a atualização das distâncias mínimas estimadas. Ao final da execução, se tem como resultado a distância mínima, partindo da fonte s, até todos os outros nodos, com cada distância presente no atributo d.

Algoritimo 2.4 – Caminho de custo mínimo com Dijkstra

```
1 PRINT-PATH(G, s, v)
2     if v == s
3         print s
4     elseif v.π == NIL
5         print "Não existe um caminho entre v e s"
6     else
7         PRINT-PATH(G, s, v.π)
```

```
8         print v
```

É possível também obter o caminho de custo mínimo, realizando uma busca recursiva sobre o atributo π, conforme apresentado no algoritmo 2.4, por este conter o elemento anterior no caminho. É feita uma verificação se v é igual a s, e em caso positivo é interrompida a execução do procedimento, pois a recursão chegou ao limite, sendo s e v iguais o nodo fonte. Caso negativo, é verificada a existência de um nodo anterior a v, no atributo v.π, e caso v.π seja *NIL* significa que não existe um caminho ligando os dois nodos. Caso as duas verificações se provem falsas, é executado novamente o procedimento "*PRINT-PATH*", com os parâmetros G, s e $v.\pi$ para continuar a recursão, até que s seja igual a v. A complexidade do algoritmo gira principalmente em torno da estrutura de dados utilizada para implementação de Q, alcançando $O(E + V. \log V)$ em um grafo esparso quando utilizada uma "*Fibonacci heap*". (CORMEN, 2009)

2.3.2 A*

O A* é um algoritmo de busca iterativo e ordenado, que mantém um conjunto de nodos abertos e inexplorados em uma tentativa de alcançar o objetivo. Ao início de cada iteração, A* utiliza de uma função $f(n)$ para encontrar um nodo n dentre os abertos cujo $f(n)$ possua o menor valor. $f(n)$ é o resultado da soma de $f(n) = g(n)+h(n)$, onde:

- $g(n)$ guarda o valor do menor caminho, desde a fonte até o nodo inicial.

- $h(n)$ é o valor estimado do nodo atual até o nodo destino.

A função $f(n)$ estima o caminho de menor custo desde o nodo inicial até o nodo final, passando por n. O algoritmo verifica se o nodo é o final após remover o nodo do conjunto de nodos abertos, garantindo que o caminho é o de menor custo. Um nodo ter um $f(n)$ baixo, significa que ele está mais perto ao nodo final. O principal componente de $f(n)$ é o cálculo de $h(n)$, pelo fato de que $g(n)$ é facilmente calculado ao percorrer os nodos, porém sem uma boa heurística para indicar um nodo próximo, A* não funciona melhor que uma busca cega pelo grafo. Logo, dependendo da admissibilidade da heurística, o algoritmo pode ter um resultado ótimo ou não.

No algoritmo 2.5 têm-se um exemplo genérico de A*, recebendo quatro parâmetros: o vértice inicial, o vértice final, uma função de peso das arestas e uma função de cálculo de heurística. Ele inicia com uma *priority queue* com apenas o nodo inicial, chamada *open*, e um conjunto de nodos visitados chamado *closed*. Inicia então uma iteração em cima de *open*, removendo sempre o nodo com maior prioridade (menor distância estimada), n. Caso n seja o nodo destino, representa o fim do algoritmo. Caso contrário, é percorrido os vértices adjacentes *adj* de n que não pertencem a *closed*, verificando se a distância até o momento é a mais curta até esse nodo. Em caso positivo, é removida a instância de *adj* em *open*, se existir, alterado o nodo antecessor, presente em Π, e alterando o atributo de distância, presente em d. Por último, o nodo *adj* é inserido em *open* com uma prioridade igual à menor distância da fonte até ele encontrada até o momento somado do valor do valor da heurística sobre o vértices, $h(adj)$.

Algoritimo 2.5 – Algoritmo de A*

```
1  A*(s, v, w, h)
2      s.d = 0
3      s.Π = s;
4      open = PRIORITY-QUEUE
5      closed = ∅
6      INSERT(open, s, 0)
7      while open ≠ ∅
8          n = EXTRACT-MIN(open)
9          closed = closed ∪ {n}
10         if (n == v) return true
11         for each vertex adj ∈ n.adjacents
12             if (adj ∈ closed) continue
13             if not adj.d or adj.d > n.d + w(n, adj)
14                 if adj ∈ open
```

```
15                  REMOVE(open, adj)
16              adj.Π = n
17              adj.d = n.d + w(n, adj)
18              INSERT(open, adj, adj.d + h(adj))
19      return false
```

Para recuperação do caminho de custo mínimo pode ser executado o algoritmo 2.6, semelhante aos já apresentados em outras sessões.(HEINEMAN; POLLICE; SELKOW, 2016)

Algoritimo 2.6 – Caminho de custo mínimo com A*

```
1  PRINT-PATH(s, v)
2      if v == s
3          print s
4      elseif v.Π == NIL
5          print "Não existe um caminho entre v e s"
6      else
7          PRINT-PATH(s, v)
8          print v
```

2.4 O PROBLEMA DE CABEAMENTO EM REDES DE FIBRA

A partir do problema descrito na seção 1.1, é possível modelar a rede de fibra como dois modelos de grafos:

- **Grafo usando os postes:** A primeira possibilidade é um modelo de grafo não dirigido, que utiliza os postes como vértices e a capacidade de se passar um cabo entre eles como critério de adjacência, conforme o modelo representado no exemplo 2 abaixo.

  Exemplo 2
  $G\ (V, E)$
  $V = \{p \mid p\ \ \textit{é um poste disponível para utilização.}\}$
  $E = \{(p_1, p_2, d) \mid \textit{Existe um cabo entre } p_1 \textit{ e } p_2\textit{, e a distância entre os dois postes é } d\}$

- **Grafo usando caixas e cabos:** O segundo modelo proposto é de um grafo dirigido, utilizando as caixas como vértices, e o critério de adjacência é a existência de um cabo ligando duas caixas, partindo de uma caixa c_1 até uma caixa c_2, conforme o modelo representado no exemplo 3.

Exemplo 3
$G\ (V, E)$
$V = \{c \mid c$ *é uma caixa existente.*$\}$
$E = \{(c_1, c_2, d) \mid$ *Existe um cabeamento óptico entre* c_1 *e* c_2*, de comprimento* $d\}$

Ambos os modelos são válidos, porém para o contexto deste trabalho o foco será o primeiro modelo proposto, que envolve a utilização de postes como vértices, visto que resolve melhor o problema em questão, por utilizar somente elementos da rede que já possuem posicionamento definido, e permitir a execução de algoritmos de caminho de custo mínimo para encontrar uma melhor rota para o cabeamento.

3 TRABALHOS RELACIONADOS

Em 2011, foi publicado um artigo (SACHENBACHER et al., 2011), que aborda o problema de caminho de custo mínimo envolvendo rotas para veículos elétricos. É explicado conceitos sobre consumo energético, e pontos a serem considerados para cálculo de caminho de menor custo energético. O grafo foi modelado considerando nodos como pontos no mapa e as arestas como o trecho da via que liga os pontos. O peso de cada aresta deve ser calculado durante a execução, levando em consideração a elevação, tamanho e limite de velocidade do trecho. Para solução do problema, é utilizada uma implementação de A* (*Energy-A**), utilizando uma heurística com base no consumo energético estimado entre dois pontos. O resultado do trabalho apresenta uma comparação do desempenho superior da implementação de A* quando comparado com a solução resolvida utilizando algumas das outras estratégias disponíveis na literatura (Dijkstra e Pallotino).

Em 2008, foi um publicado um artigo (PANAHI; DELAVAR, 2008), introduzindo um sistema de sugestão de rota em tempo real para veículo de emergência. O objetivo do trabalho era desenvolver o sistema, de forma que ele calculasse o menor caminho utilizando dados históricos, e atualizando o resultado baseado em dados de trânsito em tempo real. O conceito de menor caminho nesse contexto significando menor tempo de trajeto. Como resultado, foi identificado que o sistema de roteamento dinâmico é muito mais eficiente quando comparado com roteamentos estáticos, dando ainda mais impacto ao resultado quando considerado incidentes nas rodovias.

4 PROPOSTA

Este trabalho tem como objetivo a implementação dos algoritmos descritos na seção 2.3, Dijkstra e A*, visando auxiliar na resolução do problema descrito na seção 1.1.

Para isso, foi realizada uma modelagem do problema utilizando a teoria de grafos. O grafo em questão é a representação de uma rede de posteamento da cidade de Toledo, no estado do Paraná, com um total de 18781 postes, distribuídos numa área de 1197 km^2. Cada poste foi modelado como um nodo do grafo, e as arestas representam a possibilidade de se passar um cabo entre dois postes.

Como mencionado na seção 2.3.2, o algoritmo A* faz uso de uma heurística para melhor obtenção de resultados. Para o resolver problema descrito, foi escolhido distância em linha reta como heurística para a implementação, dando preferência aos nodos que estão geograficamente mais próximos ao nodo em destino.

4.1 ESTRUTURAÇÃO DOS DADOS

A entrada de dados se dará por um conjunto pré processado, armazenado em um arquivo no formato JSON, cada nodo indexado por um identificador único, e o nodo contendo latitude, longitude e uma lista de identificadores de seus adjacentes, conforme exemplificado:

```
{
  "1": {
    "id": 1,
    "lat": -24.71513,
    "lng": -53.74197,
    "adjacents": [
      2,
      3,
      4
    ]
  }
}
```

Além do conjunto total, será também utilizado dois subconjuntos, um contendo 12 mil nodos, e um segundo contendo 6 mil, para

melhor resultado sobre a escalabilidade de cada algoritmo. Devido ao alto grau de esparsidade do grafo, será utilizado lista de adjacência para sua representação, visto que o grafo possui um total de 20417 adjacências, com média de 1.08 adjacências por nodo. Para os algoritmos de Dijkstra e A*, que necessitam de ordenação de lista em tempo real, será utilizado uma *Priority Queue* previamente implementada.

4.2 DESCRIÇÃO DO AMBIENTE

- **Configurações da máquina:**
  - **Sistema Operacional:** Windows 10 Education 64 bits.
  - **Memória RAM:** 16 GB.
  - **Processador:** Intel i7 4ª Geração.
- Linguagem de implementação: Javascript
- Interpretador: Node.js v10.15.3

4.3 PROCEDIMENTOS

Para extração dos resultados, foi implementado um procedimento que recebe como entrada um conjunto de dados conforme exemplificado na seção 4.1. Para cada um dos 3 conjuntos de dados, serão realizadas 1000 simulações, onde em cada uma serão escolhidos dois nodos de forma aleatória dentro do grafo e aplicada a busca de caminho utilizando os algoritmos avaliados.

5 ANÁLISE DOS RESULTADOS

Com as simulações executadas, é apresentado em forma de tabelas e gráficos as comparações e análises feitas com relação ao tempo médio de processamento e uso de memória.

5.1 ANÁLISE DE TEMPO

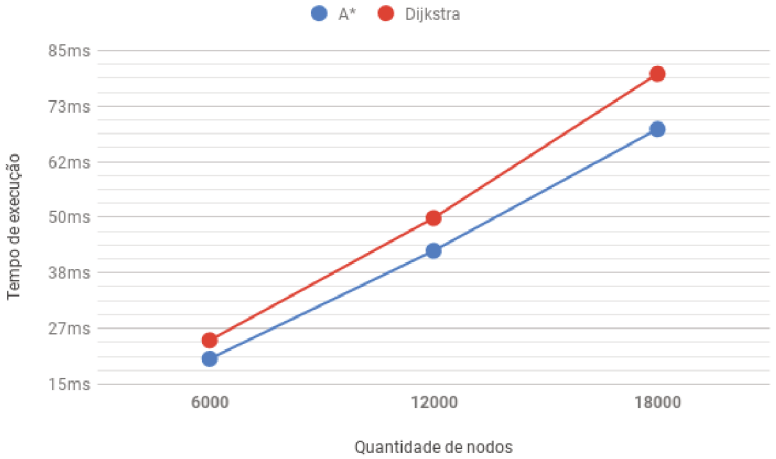
Na tabela abaixo, seguem os tempos de execução após as simulações sobre os conjuntos de dados.

Tabela 1 Análise de tempo de execução, em ms

	Dijkstra			A*		
Conjunto de dados	6000	12000	18000	6000	12000	18000
Mediana	22	45	74	19	41	64
Média	24.19	49.85	80.10	20.31	42.97	68.50
Desvio Padrão	7.44	15.30	13.65	4.82	8.57	14.60

No gráfico abaixo. segue uma comparação de escalabilidade em relação ao tempo de execução.

Figura 6 Tempo de execução.



Conforme visto na tabela 1 e no gráfico 6, A* apresentou um melhor desempenho, conseguindo uma média menor e valores mais constantes do que se comparado ao algoritmo de Dijkstra, chegando a apresentar um desempenho médio 17% mais rápido quando utilizado o conjunto total de dados.

5.2 ANÁLISE DE ESPAÇO

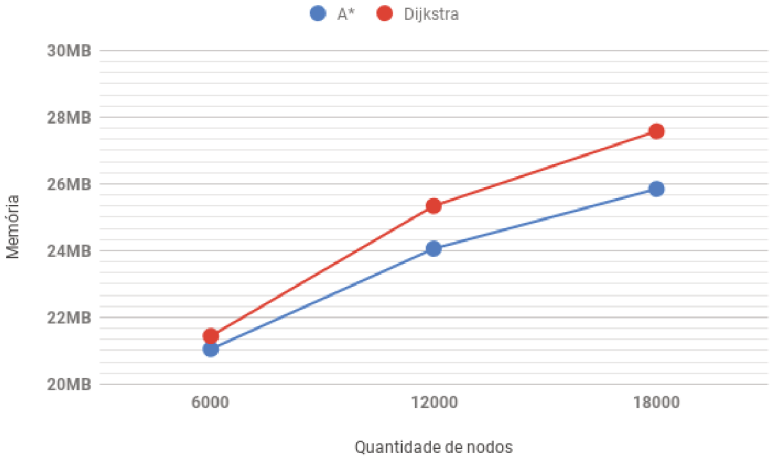
Na tabela abaixo, seguem o uso de memória durante as execuções sobre os conjuntos de dados.

Tabela 2 Análise de uso de memória, em MB

	Dijkstra			A*		
Conjunto de dados	6000	12000	18000	6000	12000	18000
Mediana	21	25	28	21	24	26
Média	21.44	25.34	27.58	21.05	24.06	25.85
Desvio Padrão	6.20	7.00	7.75	6.10	6.9	7.57

No gráfico abaixo. segue uma comparação de escalabilidade em relação ao uso de memória.

Figura 7 Consumo de memória



Conforme os valores apresentados na tabela 2 e no gráfico 7 é possível novamente identificar uma superioridade do algoritmo A* sobre Dijkstra nos 3 conjuntos. A* também demonstrou uma melhor escalabilidade, com aumento de 22% de consumo de memória considerando os conjuntos de 6 e 18 mil, e 28% de aumento pelo algoritmo de Dijkstra.

6 CONCLUSÃO

Neste trabalho é apresentado o problema de trajeto de cabeamento em uma rede de fibra óptica, onde existe o desejo de uma nova conexão entre dois pontos de uma rede e a necessidade de um trajeto válido entre os pontos.

O trabalho tem como objetivo a modelagem de uma rede utilizando a teoria de grafos, realização de um estudo e implementação de algoritmos clássicos que resolvam o problema de caminho de custo mínimo, e uma análise sobre o desempenho de execução dos algoritmos sobre uma rede real.

Dentre os algoritmos estudados, foram escolhidos Dijkstra e A* para implementação e análise. Cada algoritmo foi executado sobre sub conjuntos de uma rede de 18 mil postes da cidade de Toledo, no estado do Paraná, para retirada e análise dos dados.

A partir das análises descritas no capítulo 5, os algoritmos obtiveram um desempenho próximo, com vantagem no tempo de execução e consumo de memória para o algoritmo de A*. Se comparados ao algoritmo originalmente utilizado para solução do problema, Floyd-Warshall, o resultado é significativamente melhor, com a aplicação do algoritmo sobre o conjunto de teste podendo levar horas para sua conclusão. Isso se justifica por Floyd-Warshall possuir complexidade proporcional ao número de nodos, sendo recomendando para grafos densos, e Dijkstra e A* apresentam melhor desempenho em grafos esparsos, como o do problema.

A escolha do algoritmo deve levar em consideração a quantidade de aplicações. Casos em que existe a necessidade de busca de pelo menos 2 caminhos envolvendo o mesmo nodo, deve ser dada prioridade para o algoritmo de Dijkstra, visto que ele retorna todos os caminhos de um nodo em sua aplicação. Para casos de um único caminho, deve ser escolhido o algoritmo de A*.

6.1 SUGESTÕES PARA TRABALHOS FUTUROS

Trabalhos futuros podem ser realizados tendo como base o problema citado, aplicando diferentes adaptações dos algoritmos aplicados, incluindo variações de A* que consideram modificações no grafo para melhor desempenho. Outra possibilidade é a extensão do problema para outros setores, resolvendo, por exemplo, problemas envolvendo

eficiência energética.

REFERÊNCIAS

BONDY, J. A.; MURTY, U. S. R. et al. **Graph theory with applications**. [S.l.]: Citeseer, 1976.

CORMEN, T. H. **Introduction to algorithms**. [S.l.]: MIT press, 2009.

FUCHS, F. On preprocessing the alt-algorithm. **Student thesis, Faculty of Computer Science, Institut for Theoretical Informatics (ITI), Karlsruhe Institute of Technology (KIT)**, 2010.

GOLDBERG, A. V. Shortest path algorithms: Engineering aspects. In: SPRINGER. **International Symposium on Algorithms and Computation**. [S.l.], 2001. p. 502–513.

GOLDBERG, A. V.; HARRELSON, C. Computing the shortest path: A* search meets graph theory. In: SOCIETY FOR INDUSTRIAL AND APPLIED MATHEMATICS. **Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms**. [S.l.], 2005. p. 156–165.

HEINEMAN, G. T.; POLLICE, G.; SELKOW, S. **Algorithms in a nutshell: A practical guide**. [S.l.]: "O'Reilly Media, Inc.", 2016.

IWAZAKI, C. H. et al. Análise comparativa entre dois algoritmos que determinam um caminho de mínimo custo em grafos com custos não-negativos. 1987.

OHSHIMA, T.; NAGAMOCHI, H.; ZHAO, L. A landmark algorithm for the time-dependent shortest path problem. **Master's thesis, Graduate School of Informatics, Kyoto University**, 2008.

PANAHI, S.; DELAVAR, M. A gis-based dynamic shortest path determination in emergency vehicles. **World applied sciences journal**, Citeseer, v. 3, n. 1, p. 88–94, 2008.

RIOS, L. H. O.; CHAIMOWICZ, L. A survey and classification of a* based best-first heuristic search algorithms. In: SPRINGER. **Brazilian Symposium on Artificial Intelligence**. [S.l.], 2010. p. 253–262.

SACHENBACHER, M. et al. Efficient energy-optimal routing for electric vehicles. In: **Twenty-fifth AAAI conference on artificial intelligence**. [S.l.: s.n.], 2011.

SOUZA, A. L. et al. Teoria dos grafos e aplicações. Universidade Federal do Amazonas, 2013.

WEST, D. B. et al. **Introduction to graph theory**. [S.l.]: Prentice hall Upper Saddle River, 2001.

WILSON, R. J. **An introduction to graph theory**. [S.l.]: Pearson Education India, 1970.

APÊNDICE A – Artigo monografia

Comparativo da aplicação de algoritmos de caminho de custo mínimo em uma rede de posteamento.

Christian Zirke Osta[1]

[1]Curso de Sistema de Informação - Departamento de Informática e Estatística - Universidade Federal de Santa Catarina (UFSC) Florianópolis - SC - Brasil

christian.zirke@grad.ufsc.br

Abstract. *With the popularization of fiber optics, many internet providers begin to adopt this technology for better distribution and quality of the internet. There is, however, the difficulty of network planning. This work brings a study to help identify good solution to the problem of new stretch of cabling in a fiber network. For this, a real pole network, modeled as a graph, is used and a sequence of simulations is applied using two algorithms well-known in the literature for solving the problem of minimum cost path, Dijkstra and A *, followed by the analysis of the results, concluding the best approach to the problem.*

Resumo*. Com a popularização da fibra óptica, muitos provedores de internet começam a aderir essa tecnologia para melhor distribuição e qualidade da internet. Existe porém a dificuldade para planejamento da rede. Este trabalho traz um estudo para auxiliar a identificação de boa solução para o problema de novo trecho de cabeamento em uma rede de fibra. Para isso, é utilizada uma rede de posteamento real, modelada como um grafo, e aplicado uma sequência de simulações usando dois algoritmos famosos na literatura por resolverem o problema de caminho de custo mínimo, Dijkstra e A*, seguida da análise dos resultados, concluindo na melhor abordagem para o problema.*

1. Introdução

Cada dia mais empresas de telecomunicação estão aderindo a tecnologia utilizando fibra óptica, porém o alto custo de implantação e expansão da rede ainda é uma barreira que atrapalha o desenvolvimento e aceitação. Surge então a necessidade de realizar o projeto da rede com o menor custo de implantação possível. Esse trabalho surge como auxílio para uma solução do problema, realizando uma modelagem do problema como um problema de grafos, caminho de custo mínimo, e aplicando soluções clássicas buscando um bom desempenho de solução e resultado. Neste trabalho são estudados e analisados dois algoritmos, Dijkstra e A*, para identificar qual se adequa mais para o modelo de grafo proposto. O trabalho segue a seguinte estrutura: primeiro uma breve introdução aos conceitos que serão apresentados, uma seção sobre trabalhos que apresentam problemas e/ou soluções parecidas, seguido pela apresentação da estratégia de execução, os resultados obtidos e suas análise. O trabalho se encerra com uma conclusão sobre as análise e sugestões sobre trabalhos futuros.

2. Conceitos

Nesta seção será feita uma breve introdução a alguns conceitos de forma a ajudar na compreensão do artigo.

2.1 Problema de caminho de custo mínimo

Problema de caminho de custo mínimo é um problema clássico na área de grafos, que visa encontrar uma ou mais rotas entre nodos de um grafo de forma que seja um trajeto de custo mínimo. O problema pode ser dividido em três sub-problemas:

- **Single Source:** Problema consiste em encontrar, dado um nodo inicial, um caminho partindo desse nodo até todos os outros nodos do grafo. Exemplos de algoritmos que resolve essa classe de problema: Bellman-Ford, Dijkstra.
- **All-Pairs:** Essa classe representa problemas em que é necessária a rota de menor custo com origem e destino sendo todos os nodos do grafo. Exemplo de algoritmo que resolve essa classe de problema: Floyd-Warshall.
- **Single-Pair:** Classe que encontra um caminho de menor custo entre dois nodos do grafo. Exemplo de algoritmo que resolve essa classe de problema: A*.

2.2 Algoritmos

Para solução do problema, foram escolhidos dois algoritmos para comparação: Dijkstra, por ser um algoritmo clássico da literatura, e A*, algoritmo clássico de IA que em conjunto com uma heurística confiável apresenta resultado ótimo.

2.2.1 Dijkstra

O algoritmo de Dijkstra é um algoritmo clássico para resolução do problema de caminho de custo mínimo. Em sua implementação original, o algoritmo de Dijkstra resolve problemas das classe *Single Source* em um grafo que não possua arestas de peso negativo com complexidade temporal $O(n^2)$. Dijkstra funciona com uma iteração sobre as arestas do nodo inicial, e escolhendo como nodo seguinte o alcançável por menor custo. O nodo escolhido é marcado como fechado. O algoritmo repete o procedimento até não haver um nodo alcançável que não tenha sido fechado.

2.2.2 A*

O algoritmo A* é um algoritmo clássico na área de IA que resolve a classe de problemas *Single Pair*. A* faz uso de uma heurística, uma função que estima o custo de um nodo até o nodo destino. O algoritmo funciona de forma iterativa, mantendo um conjunto com nodos abertos e a cada iteração, o nodo de menor custo estimado é fechado e utilizado pelo algoritmo, percorrendo suas arestas. O custo estimado de um nodo é dado pela soma do custo das arestas do caminho atual e a aplicação da heurística sobre o nodo. O algoritmo repete o procedimento até que o nodo de menor custo estimado seja o nodo destino.

3. Trabalhos correlatos

Em 2011, foi publicado um artigo [Sachenbacher 2011], que aborda o problema de caminho de custo mínimo envolvendo rotas para veículos elétricos. É explicado conceitos sobre consumo energético, e pontos a serem considerados para cálculo de caminho de menor custo energético. O grafo foi modelado considerando nodos como pontos no mapa e as arestas como o trecho da via que liga os pontos. O peso de cada aresta deve ser calculado durante a execução, levando em consideração a elevação, tamanho e limite de velocidade do trecho. Para solução do problema, é utilizada uma implementação de A* (*Energy-A**), utilizando uma heurística com base no consumo energético estimado entre dois pontos. O resultado do trabalho apresenta uma comparação do desempenho superior da implementação de A* quando comparado com a solução resolvida utilizando algumas das outras estratégias disponíveis na literatura (Dijkstra e Pallotino).

Em 2008, foi um publicado um artigo [Panahi 2008], introduzindo um sistema de sugestão de rota em tempo real para veículo de emergência. O objetivo do trabalho era desenvolver o sistema, de forma que ele calculasse o menor caminho utilizando dados históricos, e atualizando o resultado baseado em dados de trânsito em tempo real. O conceito de menor caminho nesse contexto significando menor tempo de trajeto. Como resultado, foi identificado que o sistema de roteamento dinâmico é muito mais eficiente quando comparado com roteamentos estáticos, dando ainda mais impacto ao resultado quando considerado incidentes nas rodovias.

4. Resultados

Para análise dos resultados, foi utilizado uma modelagem de grafo G(V, E) onde os vértices V representam um poste, e a aresta E representa a possibilidade de passagem entre dois postes.

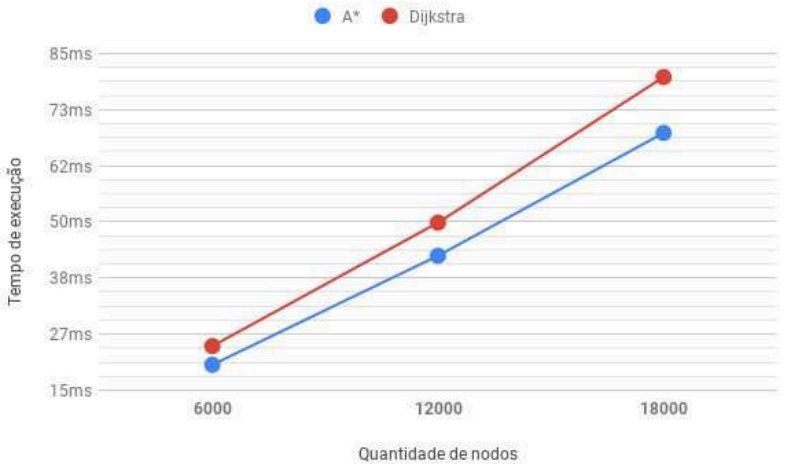
Sobre essa modelagem, foi montado um grafo de entrada contendo aproximadamente 18000, e mais dois subconjuntos desse total, contendo 12000 e 6000, visando testar também a escalabilidade dos algoritmos.

Sobre cada conjunto, foram selecionados de forma aleatória 1000 pares de nodos, podendo ou não existir um caminho válido entre eles. Cada um dos algoritmos foi aplicado sobre os pares de conjuntos.

Os resultados foram analisados considerando tempo de execução e consumo de memória.

Tabela 1 – Análise de tempo de execução, em ms

	Dijkstra			A*		
Conjunto de dados	6000	12000	18000	6000	12000	18000
Mediana	22	45	74	19	41	64
Média	24.19	49.85	80.10	20.31	42.97	68.50
Desvio Padrão	7.44	15.30	13.65	4.82	8.57	14.60



Conforme mostra a tabela e o gráfico, A* apresenta em melhor desempenho de forma geral. Leva uma pequena vantagem nos conjuntos menores, mas a melhor escalabilidade é visível no gráfico, dado o espaçamento entre os tempos de execução dos dois algoritmos. Algoritmos de heurística apresentam de forma geral um desempenho melhor do que algoritmos de complexidade polinomial, visto que possui a heurística para busca dirigida.

Tabela 2 – Análise de uso de memória, em MB

	Dijkstra			A*		
Conjunto de dados	6000	12000	18000	6000	12000	18000
Mediana	21	25	28	21	24	26
Média	21.44	25.34	27.58	21.05	24.06	25.85
Desvio Padrão	6.20	7.00	7.75	6.10	6.9	7.57

Conforme esperado, o A* apresenta novamente um melhor desempenho que o algoritmo de Dijkstra. Com uma pequena vantagem em conjuntos menores, A* apresenta uma melhora significativa ao comparar com conjuntos maiores. Isso se dá pelo fato de Dijkstra abrir um número maior de nodos adjacentes, e A* contar com auxílio da heurística para a busca dirigida, exigindo um menor número de iterações e descobertas para alcançar o resultado.

5. Conclusão

Conforme visto na seção anterior, o algoritmo A* apresentou melhor desempenho, tanto em tempo de execução quanto em consumo de memória, logo, é o algoritmo mais recomendado para aplicação na maior parte dos cenários. Dijkstra leva vantagem em que é necessário o trajeto a partir de um nodo até dois ou mais destinos, evitando re-aplicação do algoritmo.

6. Sugestão de trabalhos futuros

Como sugestão de trabalhos futuros, realizar análises mais profundas sobre os resultados, juntamente com aplicações de derivações especializadas dos algoritmos estudados. Realizar modelagem da estrutura interna da rede óptica. Abordar outras área de relacionadas que possam ser resolvidas de forma semelhante.

Referências

BONDY, J. A.; MURTY, U. S. R. et al. Graph theory with applications. [S.l.]: Citeseer, 1976.

CORMEN, T. H. Introduction to algorithms. [S.l.]: MIT press, 2009.

FUCHS, F. On preprocessing the alt-algorithm. Student thesis, Faculty of Computer Science, Institut for Theoretical Informatics (ITI), Karlsruhe Institute of Technology (KIT), 2010.

GOLDBERG, A. V.; HARRELSON, C. Computing the shortest path: A* search meets graph theory. In: SOCIETY FOR INDUSTRIAL AND APPLIED MATHEMATICS. Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms. [S.l.], 2005. p. 156-165.

HEINEMAN, G. T.; POLLICE, G.; SELKOW, S. Algorithms in a nutshell: A practical guide. [S.l.]: "O'Reilly Media, Inc.", 2016.

IWAZAKI, C. H. et al. Análise comparativa entre dois algoritmos que determinam um caminho de mínimo custo em grafos com custos não-negativos. 1987.

OHSHIMA, T.; NAGAMOCHI, H.; ZHAO, L. A landmark algorithm for the time-dependent shortest path problem. Master's thesis, Graduate School of Informatics, Kyoto University, 2008.

RIOS, L. H. O.; CHAIMOWICZ, L. A survey and classification of a* based best-rst heuristic search algorithms. In: SPRINGER. Brazilian Symposium on Artificial Intelligence. [S.l.], 2010. p. 253-262.

SACHENBACHER, M. et al. Efficient energy-optimal routing for electric vehicles. In: Twenty-fth AAAI conference on artificial intelligence. [S.l.: s.n.], 2011.

SOUZA, A. L. et al. Teoria dos grafos e aplicações. Universidade Federal do Amazonas, 2013.

WEST, D. B. et al. Introduction to graph theory. [S.l.]: Prentice hall Upper Saddle River, 2001.

WILSON, R. J. An introduction to graph theory. [S.l.]: Pearson Education India, 1970.

GOLDBERG, A. V. Shortest path algorithms: Engineering aspects. In: SPRINGER. International Symposium on Algorithms and Computation. [S.l.], 2001. p. 502-513.

PANAHI, S.; DELAVAR, M. A gis-based dynamic shortest path determination in emergency vehicles. World applied sciences
journal, Citeseer, v. 3, n. 1, p. 88{94, 2008.