
**UNIVERSIDADE ESTADUAL DO CEARÁ**

**BRUNO BEZERRA CHAVES**

# MÉTODOS COMBINATORIAIS PARA PROBLEMAS EM REDES DINÂMICAS: ALGORITMOS DE CAMINHO MÍNIMO

**FORTALEZA - CEARÁ**

**2015**

**BRUNO BEZERRA CHAVES**

# MÉTODOS COMBINATORIAIS PARA PROBLEMAS EM REDES DINÂMICAS: ALGORITMOS DE CAMINHO MÍNIMO

Monografia apresentada no Curso de Graduação em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Estadual do Ceará, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Marcos José Negreiros Gomes

**FORTALEZA - CEARÁ**

**2015**

**Dados Internacionais de Catalogação na Publicação**
**Universidade Estadual do Ceará**
**Biblioteca Central Prof. Antônio Martins Filho**
**Bibliotecário(a) Responsável - AA - CRB-0/000**

| | |
|---|---|
| O00a | Chaves, Bruno Bezerra.<br>Métodos Combinatoriais Para Problemas Em Redes Dinâmicas: Algoritmos De Caminho Mínimo / Bruno Bezerra Chaves. – Fortaleza, 2015.<br>61 p.;il.<br>Orientador: Prof. Dr. Prof. Dr. Marcos José Negreiros Gomes<br>Monografia (Graduação em Ciência da Computação) - Universidade Estadual do Ceará, Centro de Ciências e Tecnologia.<br>1. Caminho Mínimo 2. Grafos Dinâmicos I. Universidade Estadual do Ceará, Centro de Ciências e Tecnologia.<br>CDD:000.0 |

**BRUNO BEZERRA CHAVES**

**MÉTODOS COMBINATORIAIS PARA PROBLEMAS EM REDES DINÂMICAS: ALGORITMOS DE CAMINHO MÍNIMO**

Monografia apresentada no Curso de Graduação em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Estadual do Ceará, como requisito parcial para obtenção do grau de Bacharel.

Aprovada em: 13/01/2015

**BANCA EXAMINADORA**

---

Prof. Dr. Marcos José Negreiros Gomes
Universidade Estadual do Ceará – UECE
Orientador

---

Prof. Dr. Albert Einstein Fernandes Muritiba
Universidade Federal do Ceará – UFC

---

MSc. Anderson Bezerra Calixto
Empresa Brasileira de Serviços Hospitalares – EBSERH

## AGRADECIMENTOS

À minha família, pelo incentivo e apoio em todos os momentos.

À minha namorada Ismaela, que sempre me ajudou na pesquisa com paciência e incentivo.

Aos amigos que conheci na UECE, pela amizade e ajuda. Em especial, aos amigos Hedley Luna, Ivo Coelho, João Amílcar e Luiz Prudêncio.

Ao professor Marcos Negreiros pela orientação durante todo o curso e pelas oportunidades de iniciação à pesquisa.

A todas as pessoas que passaram pela minha vida e contribuíram para a construção de quem sou hoje.

*"A satisfação está no esforço e não apenas na realização final."*
***Mahatma Gandhi***

## RESUMO

Este trabalho trata do problema do caminho mínimo para gráficos dinâmicos, onde o custo da mudança de arcos, a mudança de topologia ou ambas mudam ao longo de um horizonte de tempo. Estas situações aparecem no tráfego dinâmico e/ou planejamento de rotas para as redes de transporte. Mostramos os resultados de uma adaptação do método Radix-Heap Dijkstra para lidar com as diferentes variações de redes dinâmicas, e mostramos nossos resultados para uma série de grafos dinâmicos selecionados. Usamos o software DYNAGRAPH como um ambiente computacional para avaliar nossos métodos. Extendemos o software DYNAGRAPH criando um Editor de Características, que permite alterar os atributos visuais dos vértices de um grafo dinâmico.



**Palavras-Chave**: Caminho Mínimo. Grafos Dinâmicos

**ABSTRACT**

This work deals with the dynamic shortest path problem for dynamic graphs where the cost of the arcs change, the topology change or both things change along a time horizon. These situations appears in dynamic traffic, flows and/or route planning for transportation networks. We show the results of an adaptation of the Radix-Heap Dijkstra algorithm to deal with the different kinds of dynamic networks. We show our results for a number of selected dynamic graphs. We use the DYNAGRAPH software as a computational environment to evaluate our methods. We extend the DYNAGRAPH software creating a Features Editor, which allows change the visual attributes of the vertices of a dynamic graph.

**Keywords**: Shortest Path. Dynamic Graph

## LISTA DE FIGURAS

## LISTA DE TABELAS

## SUMÁRIO

# 1 INTRODUÇÃO

O problema de caminho mínimo é um dos problemas fundamentais da computação assim como é um problema clássico em otimização combinatória. Ele é intensamente estudado e utilizado em diversas áreas como Engenharia de Transportes, Pesquisa Operacional, Ciência da Computação e Inteligência Artificial. Isso acontece porque tem potencial de aplicação a inúmeros problemas que ocorrem em transportes, logística, redes de computadores de telecomunicações, etc (PEER; SHARMA, 2007).

O roteamento de veículos em um sistema de transporte é atualmente uma das aplicações mais comuns. Neste contexto, vértices representam cruzamento de ruas, os arcos representam as vias e os pesos representam medida de custo, tempo ou distância. O caminho mínimo entre dois cruzamentos é dado pelo conjunto de arcos que resulta no custo mínimo do percurso. Este custo não necessariamente é a menor distância a percorrer, o conceito é mais genérico, considerando algum atributo quantificável, como, por exemplo, distância, tempo, risco, etc (NETTO, 1996), (CORMEN et al., 2001), (ZIVIANI, 2004). Através do crescente desenvolvimento dos computadores pessoais ou portáteis, como sistemas de navegação de carros e sistemas embarcados GPS, esse tipo de problema de caminho mínimo tem se tornado cada vez mais usado.

Diante o grande aumento do tráfego de veículos, se torna indispensável o uso de Sistemas Inteligentes de Transporte (ITS - Intelligent Transportation System), que realizam o controle e gerenciamento desse fluxo de veículos, permitindo assim o aumento do poder de decisão para o planejamento de ações de maneira mais inteligente e eficiente. Sistemas de previsão de tráfego são como forma de auxiliar o controle de tráfego de veículos. Através de análises de dados históricos e de tempo real sobre os dados lidos de fluxo de veículos, informações futuras geradas através de modelos estatísticos e computacionais podem prever o comportamento do tráfego em determinadas vias (LEONARD, 2013).

## 1.1 Motivação

Um sistema de seleção da melhor rota em condições de tráfego intenso poderia direcionar os condutores a rotas alternativas, para que se evite transitar em rotas já saturadas, mitigando os gargalos de tráfego nessas regiões. Todas essas informações seriam geradas em tempo real, e estariam disponíveis em um sistema web para os condutores através de aplicações para acesso público. Essas aplicações podem fazer uso de Sistemas Inteligentes de Transporte (do inglês Intelligent Transportation Systems ou ITS), como já dito anteriormente, definidos genericamente como sistemas de transporte que usam tecnologias de informação e de telecomunicações para assegurar a sua melhor utilização e operação, buscando controlar e reduzir congestionamentos

e filas. Entre as inúmeras aplicações, incluem-se sistemas de apoio à navegação em tempo real, cuja finalidade é auxiliar motoristas a encontrar os melhores caminhos ou rotas para atingirem seus locais de destino.

Inúmeras aplicações baseadas em SIG's (GIS - Geographic Information Systems), que permitem tratamento computacional a dados geográficos ou geo-referenciados, vêm sendo disponibilizadas na internet. São ferramentas em forma de mapa que facilitam o usuário localizar um endereço, encontrar o menor caminho entre dois lugares, ou até mesmo o caminho mais rápido para chegar ao destino.

O uso de sistemas "on-line" que sugerem a melhor rota de um ponto de origem a um ponto de destino conhecido, obtém dados em tempo real, e com isso realizam a previsão de rotas otimizadas, levando em conta o tráfego dentro das diversas faixas de horário, os usuários, etc. Os usuários que tiverem acesso ao sistema poderiam saber onde há congestionamentos ou qualquer tipo de obstrução das vias. Isso acontece porque o sistema recebe informações dos veículos, logo ele é considerado um sistema dinâmico, pois ele se adapta de acordo com os acontecimentos ao longo do tempo. Para isso, é necessário um algoritmo de caminho mínimo otimizado que forneça uma resposta para o usuário compatível com o trajeto que irá realizar.

## 1.2 Objetivos

### 1.2.1 Objetivo Geral

Criar uma ferramenta que propõe rotas otimizadas entre dois pontos conhecidos ao longo do tempo numa rede de topologia dinâmica, utilizando o software Dynagraph (CALIXTO; NEGREIROS, 2013).

### 1.2.2 Objetivos Específicos

- Seguir o modelo computacional Dynagraph (CALIXTO; NEGREIROS, 2013) para redes dinâmicas para criacão de um modelo composto que aborde redes de topologia estática e dinâmica;
- Desenvolver uma ferramenta capaz de sugerir um trajeto com menor tempo de percurso entre dois pontos conhecidos, baseada no tempo médio de percurso em trechos intermediários, numa rede que pode ser alterada ao longo do tempo.

## 1.3 Metodologia de Desenvolvimento

Para o desenvolvimento dessa solução, os grafos utilizados neste trabalho são hipotéticos, pois a pesquisa se concentra na modelagem do algoritmo de Dijkstra com Radix Heap aplicado a Grafos Dinâmicos. O trabalho foi dividido em 3 etapas, que são descritas à seguir:

- Analisar modelos de Grafos dinâmicos: determinar dentre os modelos existentes o que melhor se adapta ao problema proposto;
- Empregar modelos de caminhos mínimos: adaptar a aplicação para sistemas que usam previsão dos tempos de percurso;
- Efetuar testes: elaborar relatórios através de testes que possam ser analisados pelo software de visualização e edição Dynagraph.

## 1.4 Organização do Trabalho

Este trabalho está organizado em cinco capítulos: O Capítulo 1 apresenta soluções tecnológicas para resolução de problemas de caminho mínimo, além das motivações e metodologia para o desenvolvimento da pesquisa. O Capítulo 2 apresenta uma revisão bibliográfica sobre redes de Topologia Estática, redes Dinâmicas e sua geração e manutenção, descrevendo algumas soluções. O Capítulo 3 apresenta a fundamentação teórica para a compreensão do trabalho desenvolvido, abordando caminhos em grafos estáticos e dinâmicos, e descrevendo os algoritmos desenvolvidos nesta pesquisa. O Capítulo 4 apresenta os resultados obtidos pelos testes realizados no Dynagraph. No Capítulo 5 são apresentadas as conclusões deste trabalho e propostas para trabalhos futuros.

# 2 GRAFOS ESTÁTICOS E DINÂMICOS

Segundo (NEGREIROS; MACULAN, 2014), um grafo é formado por três conjuntos:
- Vértice ou Nodos: representam os pontos (N ou V);
- Arestas ou Elos: representam ligações não orientadas entre os Nodos (E);
- Arcos: representam ligações orientadas entre os vértices (A).

O Grafo pode ser representado da seguinte forma: $G(V,E,A)$.

Grafo orientado ou assimétrico - $G(V,A)$, $E = \emptyset$: uma aresta $(u,v)$ é dita orientada de $u$ para $v$ se o par $(u,v)$ for ordenado, com $u$ precedendo $v$.

Numa abordagem computacional, um programa orientado a objetos pode ser associado a um grafo cujos vértices representam as classes definidas no programa, e cujas arestas indicam a herança entre as classes. Existe uma aresta de um vértice $v$ a um vértice $u$ se a classe para $v$ estender a classe de $u$. Logo o grafo é assimétrico, pois essas arestas são dirigidas ou orientadas porque a relação de herança só existe em uma direção. Outro exemplo segue na figura 1.

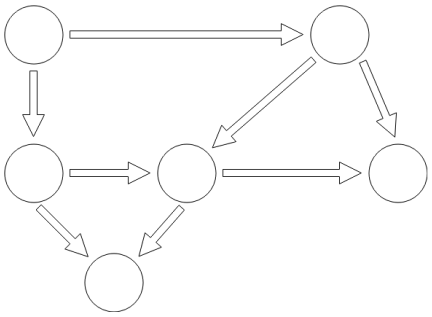

Figura 1: Grafo Orientado ou Assimétrico

Grafo não orientado ou simétrico - $G(V,E)$, $A = \emptyset$: uma aresta $(u,v)$ é dita não-orientada se o par $(u,v)$ não for ordenado. As aresta não-orientadas são por vezes denotadas como conjunto $u,v$, mas, para simplificar, é utilizado a notação de pares ordenados $(u,v)$, notando que no caso não-orientados $(u,v)$ é o mesmo que $(v,u)$ (GOODRICH; TAMASSIA, 2007), figura 2.

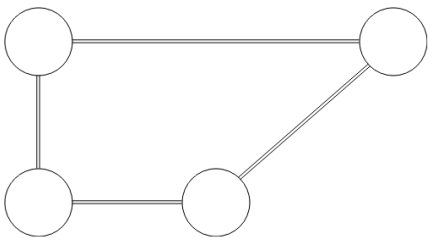

Figura 2: Grafo não Orientado ou Simétrico - $G(V,E)$

Também é possível visualizar colaborações entre pesquisadores de certa área construindo um grafo cujos vértices são associados aos pesquisadores e cujas arestas conectam pa-

res de vértices associados aos pesquisadores que escreveram juntos um artigo ou livro (Figura 3). Tais arestas são não-orientadas porque a co-autoria é uma relação simétrica, ou seja, se A é co-autor de B, então necessariamente B é co-autor de A.

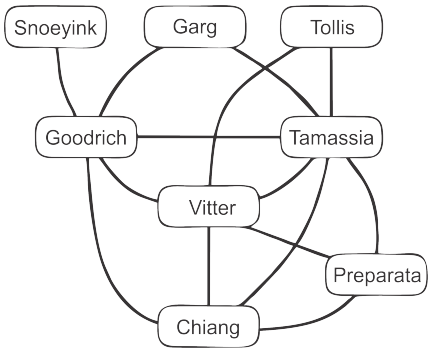


Figura 3: Grafo de co-autoria de alguns autores
Fonte: Elaboração própria, baseada em (GOODRICH; TAMASSIA, 2007)

(GOODRICH; TAMASSIA, 2007) diz que se todas as arestas em um grafo forem não-dirigidas, então diz-se que o grafo é um grafo não-dirigido. De forma similar, um grafo dirigido, ou dígrafo, é um grafo em que todas as arestas são dirigidas. Um grafo que tem arestas dirigidas e não-dirigidas é chamado de grafo misto, como mostra a figura 4. Um mapa viário de uma cidade pode ser modelado como um grafo cujos vértices são cruzamentos ou finais de ruas, e cujas arestas podem ser trechos de ruas sem cruzamentos. Este grafo tem arestas não-dirigidas, representando ruas de dois sentidos, e arestas dirigidas, correspondendo a trechos de um único sentido. Assim, um grafo que representa as ruas de uma cidade é um grafo misto.

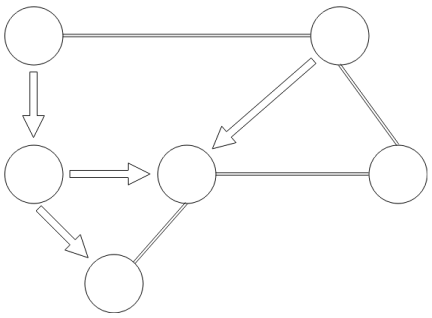

Figura 4: Grafo Misto - $G(V,E,A)$

## 2.1 Rede de Topologia Estática

Várias estruturas representam esse tipo de rede, como: estrutura matricial, estrutura de listas encadeadas, estrutura de listas duplamente encadeadas, dentre outras (NEGREIROS, 1996). Segundo (CORMEN et al., 2001), existem duas maneiras para representar um grafo $G = (V,E)$: como uma coleção de listas de adjacências ou como uma matriz de adjacências. A representação de lista de adjacências em geral é preferida, porque ela fornece um modo

compacto para representar grafos esparsos, onde para os quais $|E|$ é muito menor que $|V|^2$. Contudo, uma representação de matriz de adjacências pode ser preferível, quando o grafo é denso, onde $|E|$ está próximo de $|V|^2$, ou quando é preciso ter a possibilidade de saber com rapidez se existe uma aresta conectando dois vértices dados.

Segundo (GOLDBARG; GOLDBARG, 2012), uma matriz $A = |a_{ij}|$ quadrada de ordem $n$ é denominada matriz de adjacência de $G = (V, E)$ quando:

$a_{ij} = 1$, se $\exists (i, j) \in E$

$a_{ij} = 0$ em caso contrário.

As figuras 5 e 6 apresentam exemplos de matrizes de adjacências para grafos não direcionados e direcionados, respectivamente.

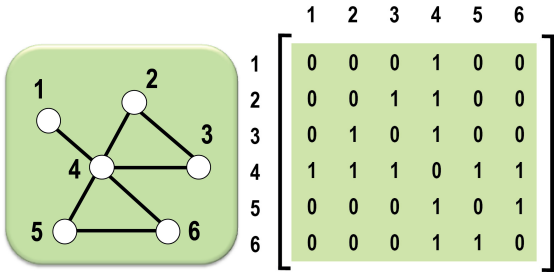


Figura 5: Matriz de adjacências de grafo não direcionado
Fonte: (GOLDBARG; GOLDBARG, 2012)

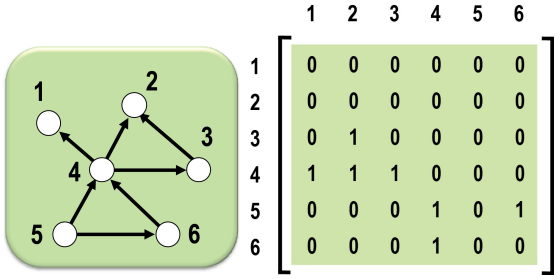


Figura 6: Matriz de adjacência de grafo direcionado
Fonte: (GOLDBARG; GOLDBARG, 2012)

Em Listas Encadeadas, segundo (NEGREIROS; MACULAN, 2014) e (CORMEN et al., 2001), quando se deseja armazenar um grafo pouco denso, ou seja, $D_{max} \leqslant \frac{|V|}{2}$ é mais vantajoso usar estruturas mais eficientes de armazenamento, assim como as listas e vetores de listas. Neste caso, os vértices estão posicionados no vetor principal, onde a própria célula do vetor guarda o rótulo do vértice que entrou primeiro e assim sucessivamente, como numa pilha. As pilhas, que derivam de cada célula do vetor, podem ser construídas levando-se em conta as ligações de arcos/elos ao elemento vértice da célula que o gera. No índice de cada célula de lista, mantém-se pelo menos uma informação contendo o vértice de ligação, figura 7.

*Type*
Lista: ^ Lta;

*Lta* = Record
*v* : word; { Arco/elo ligado a VV_i }
*prx* : Lista;
end;
*VV* = Array[1..n] of Lista;

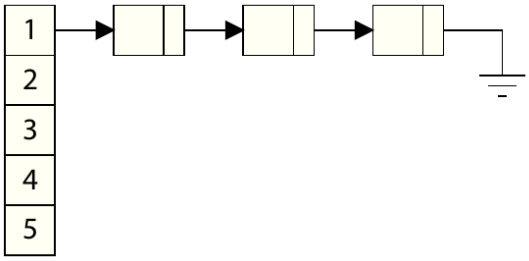


Figura 7: Representação em Listas Encadeadas de um grafo
Fonte: (CALIXTO; NEGREIROS, 2013) apud (NEGREIROS; MACULAN, 2014)

Estruturas de Listas Duplamente Encadeadas são da forma, representada na figura 8, segundo (NEGREIROS; MACULAN, 2014):

*Type*
Lista: ^ Lta;
*Lta* = Record
*v* : word; { Arco/elo ligado a VV_i }
*prx* : Lista;
*ant* : Lista;
end;
*VV* = Array[1..n] of Lista;

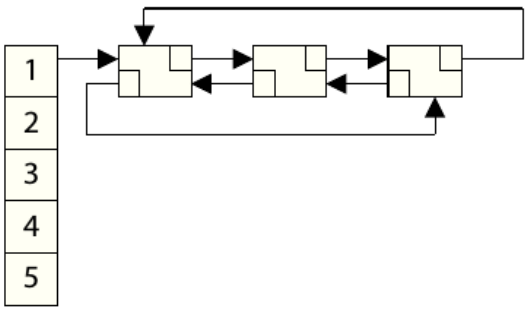


Figura 8: Representação de um grafo em Listas Duplamente Encadeadas
Fonte: (CALIXTO; NEGREIROS, 2013) apud (NEGREIROS; MACULAN, 2014)

## 2.2 Redes Dinâmicas

Em (HARARY; GUPTA, 1997), são definidas três tipos de redes: rede de nodos, rede de elos e rede ponderada.

- Uma rede de nodos (ou grafo de nodos ponderados) é uma tripla $(V,E,f)$, onde $V$ é um conjunto de vértices, $E$ é um conjunto de arestas $\{u,v\}$, e $f$ é uma função, $f : V \rightarrow N$ onde $N$ é um sistema numérico, atribuição de um valor ou um peso.
- Uma rede de elos (ou grafo de arestas ponderadas) é uma tripla $(V,E,g)$, definida de forma semelhante.
- Uma rede ponderada (ou grafo totalmente ponderado) tem pesos atribuídos a ambos nós e arestas.

Outro tipo de rede é chamada de rede genérica, que contém atributos e características.

Grafo Dinâmico $G^t(V^t,L^t)$ é todo grafo que modifica seus vértices ($V^t$) e/ou ligações ($L^t$) ao longo de um período de tempo ($H \in [T_i,T_k]$). Ou seja, as entidades $V$ (um conjunto de nodos), $E$ (um conjunto de elos), $f$ (mapeamento de vértices para números) e $g$ (mapeamento de arestas para números) podem se modificar dentro do intervalo H. Logo, existem cinco tipos básicos de Grafos Dinâmicos.

- Em um grafo ou dígrafo com nós dinâmicos, o conjunto $V$ varia com o tempo. Assim, alguns nós podem ser adicionados ou removidos. Quando os nós são removidos, as arestas ligadas a eles também são removidas;
- Em um grafo ou dígrafo com elos dinâmicos, o conjunto $E$ varia com o tempo. Assim, os arcos podem ser adicionados ou removidos a partir do grafo ou dígrafo;
- Em um grafo dinâmico de nodos ponderados, a função $f$ varia com o tempo. Assim, os pesos nos nós também variam;
- Em um grafo dinâmico de elos ponderados, a função $g$ varia com o tempo;
- Em um grafo dinâmico totalmente ponderado (ou grafo com topologia e atributos dinâmicos), ambas as funções $f$ e $g$ podem variar com o tempo.

### 2.2.1 Topologia Estática e Atributos Dinâmicos

Nesta rede os vértices e as arestas são constantes ao longo do tempo, mas seus atributos podem ser alterados. O grafo é definido como $G(V^t,L^t)$, onde $V^t$ e $L^t$ são constantes no horizonte $H \in [T_i,T_k]$. A figura 9 exemplifica este grafo.

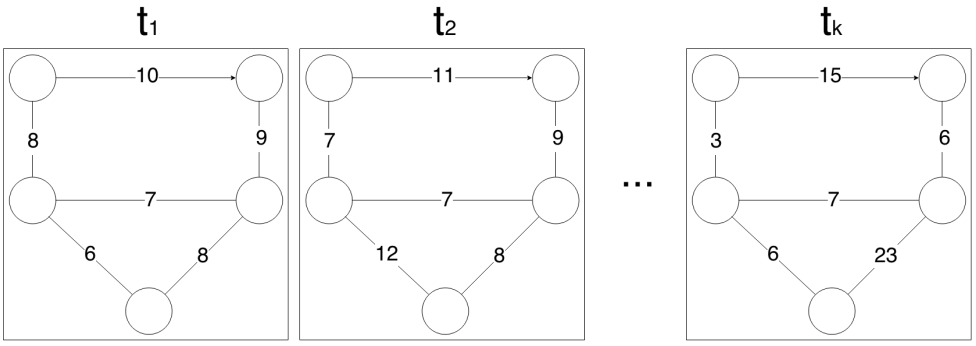


Figura 9: Grafo com Topologia Estática e Atributos Dinâmicos

### 2.2.2 Topologia Dinâmica e Atributos Estáticos

Nesta rede, ao longo do tempo, os vértices e as arestas podem ser removidos, adicionados e até mesmo modificados para outra posição, mas suas características como espessura, cor e tamanho são fixas. O grafo é definido como $G(V^t, L^t)$, onde $V^t$ e $L^t$ mudam no horizonte $H \in [T_i, T_k]$, porém os atributos sempre serão os mesmos. A figura 10 mostra um exemplo desse grafo.

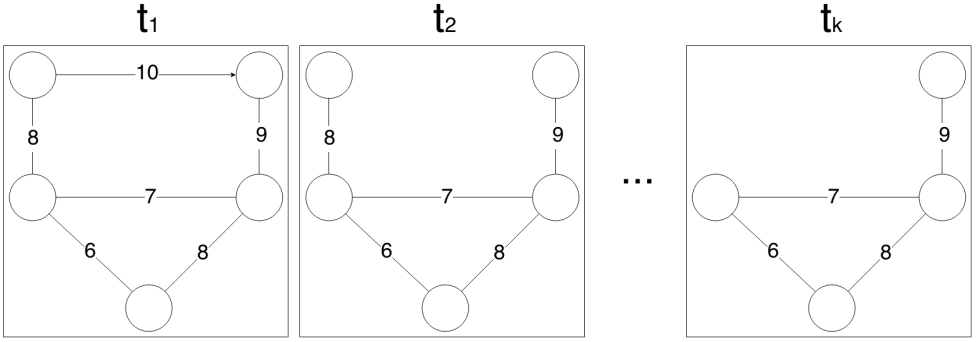


Figura 10: Grafo com Topologia Dinâmica e Atributos Estáticos

### 2.2.3 Topologia e Atributos Dinâmicos

Nesta rede, o grafo é definido como $G(V^t, L^t)$, onde $V^t$ e $L^t$ mudam no horizonte $H \in [T_i, T_k]$, como visto na figura 11. Estas redes são mais complexas (CALIXTO; NEGREIROS, 2013), pois lidam com grande volume de dados comparadas as redes anteriores descritas.

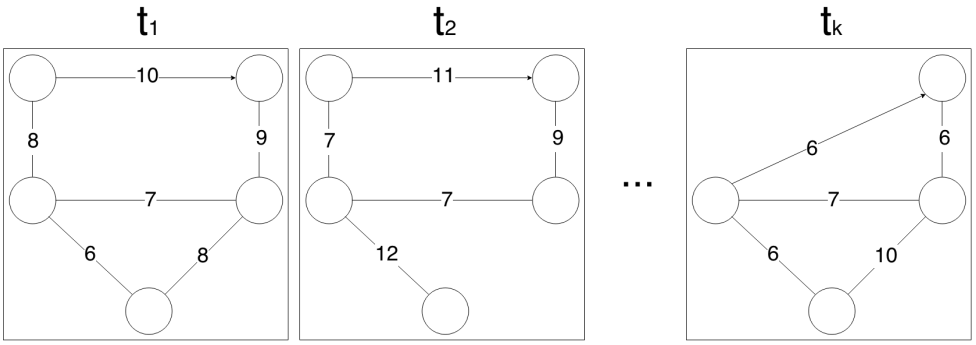


Figura 11: Grafo com Topologia e Atributos Dinâmicos

## 2.3 Geração e manutenção de Redes Dinâmicas

Os seguintes trabalhos abordam grafos dinâmicos e são utilizados como base deste trabalho:

- Modelo de Kim e Anderson, (KIM; ANDERSON, 2012);
- Gephi, (BASTIAN; HEYMANN; JACOMY, 2009);
- Dynagraph, (CALIXTO; NEGREIROS, 2013).

### 2.3.1 O modelo de Kim e Anderson

A ideia central de (KIM; ANDERSON, 2012) é modelar uma rede dinâmica como digrafos orientados ao tempo (time-ordered graph), que é gerada através da ligação de instantes temporais com arestas direcionadas que unem cada nó ao seu sucessor no tempo. Com isso, transformar uma rede dinâmica em um grafo maior, mas facilmente analisável. Isto permite não só a utilização dos algoritmos desenvolvidos para gráficos estáticos, mas também para melhor definir métricas para gráficos dinâmicos.

Segundo (KIM; ANDERSON, 2012) um sistema de grafos dinâmicos é um objeto de representação visual que pode descrever melhor o comportamento dinâmico de objetos relacionados a eventos dinâmicos e introduzir novas formas de enxergar ou descrever a evolução de eventos dinâmicos na natureza.

Assumindo que a duração de um período observado é finito, de um tempo inicial $t_{start}$ até o tempo final $t_{end}$, sem perda de generalidade, é dado $t_{start} = 0$ e $t_{end} = T$. Uma rede dinâmica $G^D_{0,T} = (V, E_{0,T})$ consiste em um conjunto de vértices e arestas temporais existentes no intervalo de tempo $[0, T]$, onde os vértices $V$ e conjunto de arestas temporais $E_{0,T}$, onde uma aresta $(u, v)_{i,j} \in E_{0,T}$ existe entre vértices $u$ e $v$ em um intervalo de tempo $[i, j]$, tal que $i \leqslant T$ e $j \geqslant 0$ (KIM; ANDERSON, 2012).

Neste modelo, em uma rede dinâmica, o conjunto de vértices $V$ é sempre o mesmo, enquanto o conjunto de arestas existentes muda ao longo do tempo.

A letra $w$ representa a duração de cada *snapshot* (ou janela de tempo) e expressa em alguma unidade de tempo (como segundos ou horas).

Conforme (KIM; ANDERSON, 2012), uma rede dinâmica pode ser representada como uma série de grafos estáticos $G_1, G_2, ..., G_N$. A notação $G_t(1 \leqslant t \leqslant n)$ representa o grafo agregado que consiste de um conjunto de vértices $V$ e um conjunto de arestas $E_t$, onde uma aresta $(u,v) \in E_t$ existe somente se uma aresta temporal $(u,v)_{i,j} \in E_{0,T}$ existe entre os vértices $v$ e $u$ no intervalo de tempo $[i,j]$, tal que $i \leqslant wt$ e $j > w(t-1)$. $G_t$ é o t-ésimo *snapshot* temporal de uma rede dinâmica $G^D_{0,T}$ durante a t-ésima janela de tempo.

A tabela 1 mostra uma relação de arestas e seus intervalos de existência. A figura 12 mostra os mesmos dados desta tabela, em uma série de grafos estáticos e representação agredada.

| Aresta | Intervalo de tempo |
|---|---|
| (A,C) | [1,1] |
| (A,D) | [2,2] |
| (B,D) | [2,3] |
| (C,D) | [3,3] |

Tabela 1: Exemplo de contatos (arestas) em uma rede dinâmica

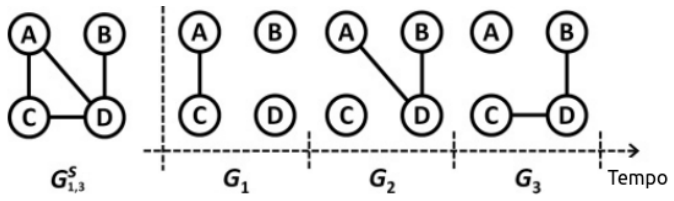


Figura 12: Comparação entre a representação de grafos agregados(esquerda) e representação de sequência temporal(direita)
Fonte: (KIM; ANDERSON, 2012)

### 2.3.2 Gephi

Gephi é um software de código aberto para análise e manipulação de redes. Ele usa um motor de renderização 3D para exibir grandes redes em tempo real e para acelerar a exploração.

Módulos desenvolvidos podem importar, visualizar, espacializar, filtrar, manipular e exportar todos os tipos de redes (BASTIAN; HEYMANN; JACOMY, 2009).

Segundo (CALIXTO; NEGREIROS, 2013) o Gephi a princípio, seria um aplicativo para grafos estáticos, porém, posteriormente foi incorporada uma característica temporal à sua estrutura. Seus dados são baseados em uma espécie de matriz para vértices e uma outra para arestas. Cada coluna representa uma informação. Para os vértices, há colunas como identificador, rótulo, posição, tamanho, cor e intervalo de tempo. Para as arestas, há colunas para o identificador, origem, destino, o tipo (dirigido ou não), rótulo, peso e cor. No modelo proposto, é possível adicionar novas colunas para vértices ou arestas, e apenas nestas é possível definir informações que mudam no tempo.

Como o Gephi não permite que alguns tipos de dados estruturados sejam modificados ao longo do tempo, os vértices não podem mudar de posição no tempo, tampouco suas características visuais durante a sua existência. O mesmo acontece com as mudanças de características visuais das arestas, que se mantêm constantes. Apesar disto, os atributos dos vértices e arestas podem modificar no tempo (CALIXTO; NEGREIROS, 2013).

As figuras 13 e 14 mostram a interface gráfica do Gephi.

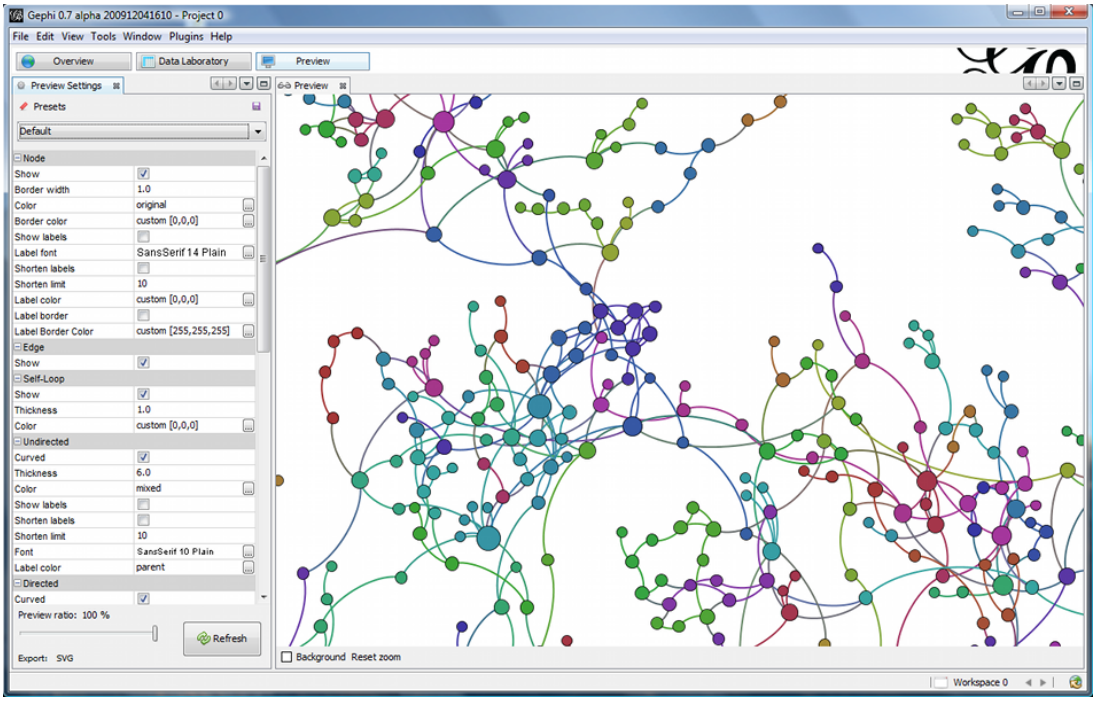


Figura 13: Captura de tela do software Gephi
Fonte: gephi.github.io

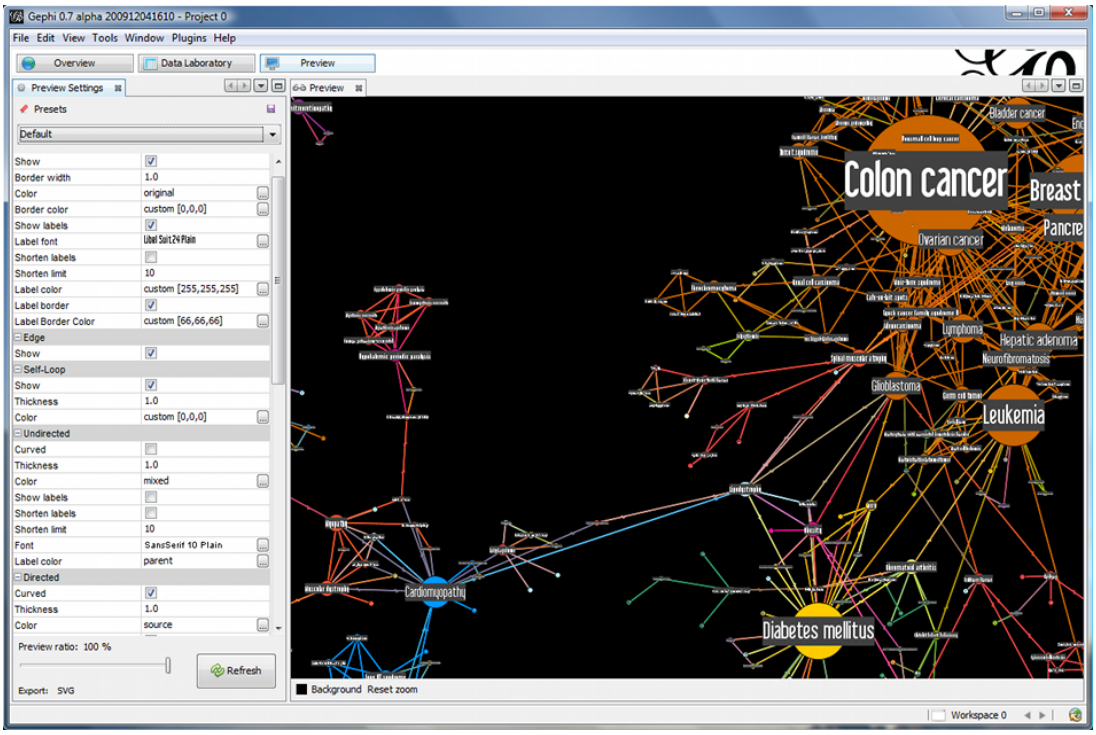


Figura 14: Captura de tela do software Gephi
Fonte: gephi.github.io

### 2.3.3 O modelo Dynagraph

O modelo Dynagraph (CALIXTO; NEGREIROS, 2013) é baseado na primeira proposta em (CALIXTO; NEGREIROS, 2012), que por sua vez é baseado no modelo de (KIM; ANDERSON, 2012), porém o Dynagraph usa sequências temporais para vértices, arestas, características modificáveis dos vértices e arestas e o relacionamento entre suas características. Com isso, é formado um grafo com as informações necessárias para qualquer instante no tempo. O Dynagraph é capaz de visualizar o comportamento do grafo ao longo de um período de tempo, e editá-lo. A ferramenta construída permite visualizar previsões e processos dinâmicos em vários contextos e realizar simulações preditivas sobre estes eventos (CALIXTO; NEGREIROS, 2013).

Um grafo dinâmico é definido por $G^t$, que ocorre no intervalo $t = [t_s, t_f]$, onde $t_s$ é o tempo inicial e $t_f$ é o tempo final. $G^t = (V^{t_v}, E^{t_e})$, onde $t_v \subseteq t$ e $t_e \subseteq t$, e $V^{t_v}$ e $E^{t_e}$ são funções que geram vértices e arestas respectivamente, em função do intervalo $t = t_v \cup t_e$.

As estruturas $V^{t_v} = \{O^{t_v}, F^{t_v}\}$ para vértices e $E^{t_e} = \{O^{t_e}, F^{t_e}\}$ para arestas são semelhantes, onde $O^{t_v} = \{O_{0_v}, O_{1_v}, ..., O_{i_v}\}$ e $O^{t_e} = \{O_{0_e}, O_{1_e}, ..., O_{i_e}\}$ os objetos indexados de objetos, e $F^{t_v} = \{F^{t_v}_{0_v}, F^{t_v}_{1_v}, ..., F^{t_v}_{w_v}\}$ e $F^{t_e} = \{F^{t_e}_{0_e}, F^{t_e}_{1_e}, ..., F^{t_e}_{w_e}\}$ os conjuntos indexados de características modificáveis destes objetos (CALIXTO; NEGREIROS, 2013).

A estrutura de dados usada no Dynagraph segue a Notação de Objeto Javascript (JSON),

que é um formato de texto de intercâmbio de dados (CROCKFORD, 2008). A figura 56 mostra como é essa estrutura seguindo três objetos principais: “metadata”, “binding” e “data”.

Em “metadata” são definidos os campos para utilização de qualquer identificador, por exemplo é possível utilizar “ini” e “fim”, que representam o tempo inicial e final de um elemento, no lugar de “start” e “end” respectivamente.

Em “binding” são definidas as características dos vértices e arestas. Seguindo o exemplo da figura 56, “vertex” poderá ser do tipo “v1”, “v2” e “v3”, onde cada tipo contém informações da forma do vértice. Essa forma pode ser uma imagem no formato “png” ou “jpg” ou customizada com as seguintes características:

- path: círculo ou seta;
- fillColor: cor do preenchimento;
- strokeColor: cor da borda;
- fillOpacity: opacidade;
- scale: tamanho;
- strokeWeight: espessura da borda.

A aresta, ou “polyline”, segue uma estrutura semelhante a do “vertex”, porém com algumas particularidades como repetição de um símbolo ao longo da aresta, e uma customização no campo “path” seguindo a notação SVG[1].

Em “data” são definidos os elementos do grafo, os tempos de início e fim de cada elemento ou tempo de existência. No caso dos vértices, a posição de cada elemento pode ser escrita no formato UTM ou latitude e longitude. O tipo de cada elemento, descrito em “binding”, e no caso das arestas, são definidos os pontos de origem e destino.

Na figura 56 vemos a estrutura de construção de um grafo dinâmico. Os campos “metadata” e “binding” foram omitidos para evidenciar o objeto “data”.

---

```
{
  "metadata": {...},
  "binding": {...},
  "data": {
    "vertex": {
      "elements": {
        "1": {"ini": "2014-01-01 11:00:13", "fim": null},
```

[1] Scalable Vector Graphics - é uma forma de descrever de forma vetorial desenhos e gráficos bidimensionais

```
        "2": {"ini": "2014-01-02 15:17:53", "fim": null},
        "3": {"ini": "2014-01-03 03:48:20", "fim": null}
      },
      "temporalfields": {
        "posicao": {
          "1": [{
            "time": "2014-01-01 11:00:13",
            "data": {"x": 552713, "y": 9583920, "zona": 24, "hemisferioSul": true}
          }],
          "2": [{
            "time": "2014-01-02 15:17:53",
            "data": {"x": 552931, "y": 9584080, "zona": 24, "hemisferioSul": true}
          }],
          "3": [{
            "time": "2014-01-03 03:48:20",
            "data": {"x": 552939, "y": 9583780, "zona": 24, "hemisferioSul": true}
          },
          {
            "time": "2014-01-04 03:48:20",
            "data": {"x": 553058, "y": 9583937, "zona": 24, "hemisferioSul": true}
          }
          ]
        },
        "tipo": {
          "1": [
            {"time": "2014-01-01 11:00:13", "data": "v1"},
            {"time": "2014-01-03 13:09:02", "data": "v2"}
          ],
          "2": [
            {"time": "2014-01-02 15:17:53", "data": "v2"},
            {"time": "2014-01-03 13:09:02", "data": "v1"},
            {"time": "2014-01-04 03:48:20", "data": "v2"}
          ],
          "3": [
            {"time": "2014-01-03 03:48:20", "data": "v3"}
          ]
        }
      }
    },
    "edge": {
      "elements": {
        "1": {"ini": "2014-01-02 19:00:07", "fim": null},
        "2": {"ini": "2014-01-03 03:48:20", "fim": null}
      },
      "temporalfields": {
        "origemdestino": {
          "1": [{"time": "2014-01-02 19:00:07","data": {"origem": 1,"destino": 2}}],
          "2": [{"time": "2014-01-04 03:48:20","data": {"origem": 2,"destino": 3}}]
        }
      }
    }
  }
}
```

Figura 15: Estrutura JSON usada pelo Dynagraph

A figura 16 demonstra os dados do código da figura 56 sendo utilizados no software Dynagraph com variações temporais no grafo e nas características de seus vértices e arestas.

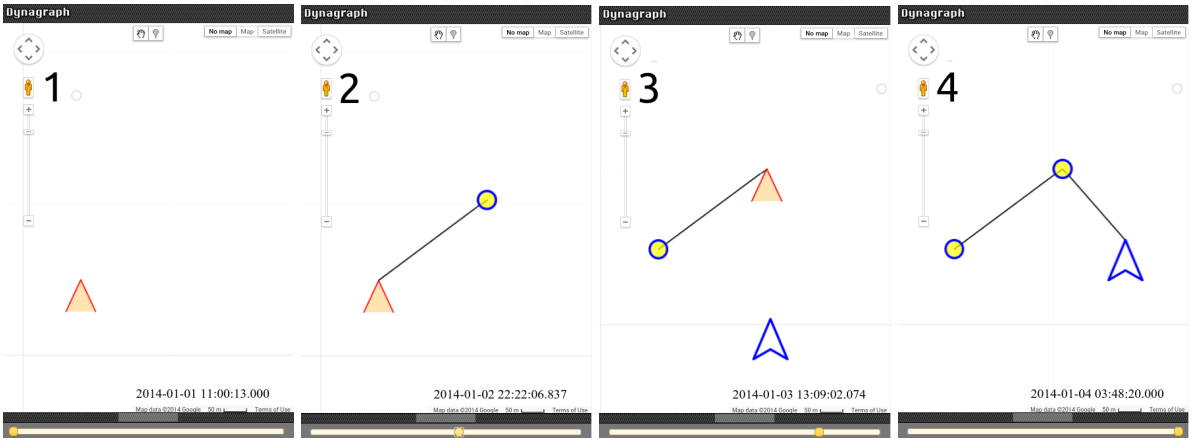


Figura 16: Capturas de tela do software Dynagraph

# 3 CAMINHOS EM GRAFOS

## 3.1 Caminhos em Redes Estáticas

Dentre os diversos problemas que surgem em grafos, este é o mais fundamental de todos e aquele que ao longo dos anos foi dos mais estudados. Várias técnicas surgiram desde os meados de 1950 com o objetivo de tratar eficientemente o problema de caminhos mínimos em grafos. A principal delas considera o fato de se promover uma arborescência em um grafo onde a medida que os vértices explorados são atingidos, tem-se uma proximidade da solução do problema (NEGREIROS; MACULAN, 2014).

### 3.1.1 Problema de Caminho Mínimo

Uma rede de transporte (que pode ser uma malha viária, rodoviária, etc.) pode ser representada por uma grafo $G = (N, A)$, onde $N$ é o conjunto de nós e A é o conjunto de arcos os quais interligam estes nós. Considerado o número de nós $|N| = n$; e o número de arcos $|A| = m$; para cada arco $(i, j) \in A$ está associado um custo unitário $c_{ij}$. O caminho entre um nó origem $(s)$ e um nó destino $(t)$ é definido por uma sequência de arcos: $(s,i),...(k,l),...(j,t)$. O Problema de Caminho Mínimo (PCM) consiste em determinar um caminho entre $s$ e $t$ tal que a somatória dos custos unitários dos arcos (ou alguma outra medida de impedância) que compõem este caminho seja o mínimo (ATZINGEN et al., 2007).

Dada uma rede $G$ com $m$ nós e $n$ arcos, associando a cada arco $(i, j)$ o custo $c_{ij}$, o Problema de Caminho Mínimo é encontrar o menor caminho entre o nó 1 para o nó $m$ em $G$ (caminho mínimo de menor valor). O custo do caminho é a soma dos custos sobre os arcos do caminho encontrado. Uma formatação genérica do problema de caminho mínimo é dada via grafos, quando deseja-se encontrar o percurso de custo mínimo entre dois vértices $i$ e $j$ de um grafo $G(V,L)$, onde, se $\exists\ \Gamma(i, j)$, $i,\ j \in V$, então $w_{ij}$ é tomado como o custo mínimo de um caminho direto entre os vértices $i$ e $j$, e todo $i_1$, ..., $i_k \in V$, distintos de $i$, $j$, são ditos serem vértices do caminho, onde $i$ precede $i_1$ e assim por diante, nesta ordem (NEGREIROS; MACULAN, 2014).

O princípio de qualquer algoritmo de caminho mínimo está associado ao seguinte teorema demonstrado em (NEGREIROS; MACULAN, 2014), como mostra a figura 17:

$C(\Gamma(s,t)) = C(\Gamma(s,k)) + C(\Gamma(k,t))$ é mínimo $\Longleftrightarrow \Gamma(s,k)$ é mínimo e $\Gamma(k,t)$ é mínimo.

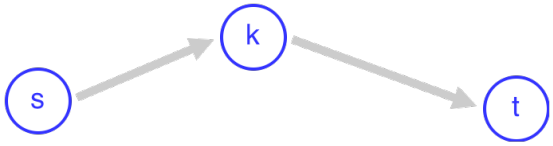


Figura 17: Teorema - Princípio do algoritmo de caminho mínimo

### 3.1.2 Algoritmo de Dijkstra

O algoritmo de Dijkstra foi proposto em 1959 e permite determinar a solução ótima através da adição de vértices à árvore de caminho mínimo pelo processo de relaxamento de uma aresta, que consiste em verificar se há a possibilidade de melhorar o caminho obtido até o momento (NETTO, 1996). Este algoritmo considera basicamente um processo de rotulação de vértices à medida que o mínimo caminho é encontrado passo a passo, interativamente, em vértices intermediários. O algoritmo requer que nenhum peso no grafo seja negativo. Abaixo é apresentado o pseudocódigo do algoritmo de Dijkstra.

```
AlgoritmoDijkstra
    inicialize a distância para todos os nós em G = infinito
    inicialize o predecessor de todos os nós em G = vazio
    Enquanto H não estiver vazio faça
        u = o nó com menor rótulo extraído de H
        Para cada v adjacente a u:
            Se rótulo[v] > rótulo[u] + distância[u, v]:
                rótulo[v] = rótulo[u] + distância[u, v];
                predecessor[v] = u;
                atualiza a posição de v em H
            Fim Se
        Fim Para
    Fim Enquanto
Fim Dijkstra
```

Figura 18: Algoritmo de Dijkstra
Fonte: (CORMEN et al., 2001)

### 3.1.3 Algoritmo Radix Heap

O algoritmo Radix Heap é utilizado numa variação do algoritmo de Dijkstra e foi proposto inicialmente por Ahuja, Mehlhorn, Orlin e Tarjan em (AHUJA et al., 1990). Ele é considerado ainda no meio científico como um dos algoritmos mais eficientes para resolver o problema do caminho mínimo. A implementação do Radix Heap é um híbrido da implementação primitiva $O(n^2)$ e implementação de Dial($O(m+nC)$)). Estas duas implementações representam dois

extremos no que diz respeito à quantidade dos buckets utilizados. A implementação primitiva considera todos os vértices rotulados temporariamente juntos, em um bucket grande, e procura por um vértice com o menor rótulo. Já o algoritmo de Dial usa um grande número de buckets e separa os vértices, armazenando dois vértices quaisquer com rótulos diferentes em diferentes segmentos (AHUJA; MAGNANTI; ORLIN, 1993). A implementação Radix Heap melhora esses dois métodos através de uma solução intermediária: Ele armazena vários, mas não todos os vértices em um mesmo bucket. Por exemplo, em vez de armazenar apenas os vértices com $d[v] = k$ em um bucket k, como na implementação do Dial, pode-se armazenar todos os vértices com $d[v]$ dentro do intervalo [100$k$ para 100$k$ + 99] no bucket $k$ (AHUJA; MAGNANTI; ORLIN, 1993).

Para o bucket[k] é definido um intervalo de valores denotado por intervalo(k). O número de inteiros no intervalo é chamado de largura do intervalo e denotado por largura(k). No exemplo anterior, o intervalo do bucket $k$ é [100$k$, 100$k$ + 99] e sua largura é 100.

Usar larguras de tamanho $k$ permite reduzir o número de buckets necessários por um fator de $k$. Mas para encontrar o rótulo de menor distância, é preciso procurar todos os elementos no bucket não vazio de menor índice. Para superar isso, o algoritmo radix heap considera usar larguras variáveis e altera os intervalos de forma dinâmica. O Radix Heap segue as propriedades:

1. As larguras dos buckets são 1, 1, 2, 4, 8, 16, ..., de modo que o número de buckets necessários é somente $O(log(NC))$, onde N é o número de vértice e C o custo da maior aresta.

2. Os intervalos dos buckets são modificados dinamicamente e são realocados os vértices de menor rótulo temporário para um único bucket, cuja largura é 1.

A Propriedade 1 nos permite manter apenas $O(log(NC))$ buckets e, assim, supera a desvantagem do algoritmo de Dial, que usa muitos buckets. A Propriedade 2 nos permite, como no algoritmo Dial, evitar a necessidade de pesquisar todo o bucket para encontrar um vértice de menor rótulo temporário. Quando implementado deste modo, esta versão do algoritmo radix heap tem complexidade $O(m + nlog(nC))$ (AHUJA; MAGNANTI; ORLIN, 1993).

Para um dado problema de caminho mínimo, o radix heap consiste em $1 + [log(NC)]$ buckets. Os buckets são numeradas de 0 até $K = [log(nC)]$. O algoritmo irá alterar os intervalos dos buckets de forma dinâmica, e cada vez que muda os intervalos, redistribui os vértices nos buckets. Inicialmente, os buckets têm os seguintes intervalos:
intervalo(0) = [0];
intervalo(1) = [1];
intervalo(2) = [2, 3];
intervalo(3) = [4, 7];
intervalo(4) = [8, 15];

...
intervalo(K) = [$2^{K-1}$, $2^K - 1$].

Esses intervalos mudam à medida que o algoritmo prossegue. No entanto, a largura dos buckets nunca aumenta para além das suas larguras iniciais (AHUJA; MAGNANTI; ORLIN, 1993).

#### 3.1.3.1 Operações sobre o Radix Heap

Determinar qual intervalo contém um dado valor pode ser feito percorrendo o vetor de buckets com complexidade $O(lognC)$. Logo, inserir um vértice na estrutura, tem complexidade $O(K)$, pois $O(K) = O(lognC)$

Para descrever a operação de retirar o item da fila de prioridades que contém o menor valor chave considere o seguinte exemplo: Supondo que o rótulo temporário de um vértice de conteudo(4) é 9, cujo intervalo é [8, 15]. O algoritmo irá examinar cada vértice em conteudo(4) para identificar um vértice com o rótulo de menor distância. Segundo (AHUJA; MAGNANTI; ORLIN, 1993), os rótulos de distância que o algoritmo de Dijkstra designa como permanente não são decrescentes, isso implica que nenhum rótulo temporário de distância jamais voltará a ser inferior a 9 e, consequentemente, não precisará mais dos buckets de 0 a 3.

Em vez de deixar estes buckets inativos, o algoritmo redistribui o intervalo[9, 15] para os buckets anteriores, resultando nos intervalos intervalo(0) = [9], intervalo(1) = [10], intervalo(2) = [11, 12], intervalo(3) = [13,15] e intervalo(4) = $\emptyset$ . Uma vez que o intervalo(4) está vazio agora, o algoritmo redistribui os vértices que estavam em conteudo(4) para os buckets adequados (0, 1, 2, e 3). Assim, cada um dos vértices no bucket 4 move-se para um bucket de menor índice e todos vértices com o rótulo de menor distância são movidos para o bucket 0, que tem largura 1 (AHUJA; MAGNANTI; ORLIN, 1993).

#### 3.1.3.2 Funcionamento do Algoritmo de Dijkstra com Radix Heap

Sempre que o algoritmo encontra vértices com o rótulo de menor distância em um bucket com largura maior que 1, ele verifica todos os vértices no bucket para identificar um vértice com rótulo de menor distância. Em seguida, o algoritmo redistribui o intervalo dos buckets e muda cada vértice no bucket para o bucket de menor índice. Uma vez que o radix heap contém $k$ buckets, um vértice pode mudar na maioria das $k$ vezes, e consequentemente, o algoritmo irá verificar qualquer vértice na maioria das $k$ vezes. Por isso, o número total de verificações de vértices é $O(NK)$, o qual não é "muito grande"(AHUJA; MAGNANTI; ORLIN, 1993).

Para demonstrado o funcionamento do algoritmo de Dijkstra com Radix Heap serão

utilizadas figuras do livro (AHUJA; MAGNANTI; ORLIN, 1993). O grafo na figura 19 exemplifica o radix heap, e o número ao lado de cada arco indica o seu comprimento.

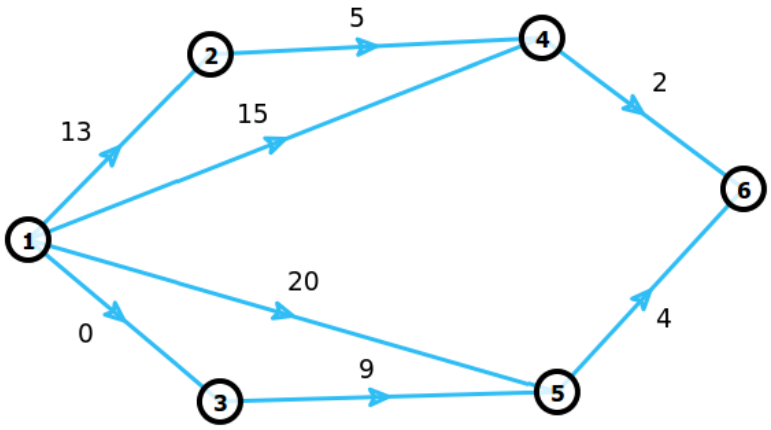


Figura 19: Grafo - Caminho Mínimo

O vértice origem é $s = 1$. O peso da maior aresta é $C = 20$, logo $K = [log(nC))] = [log(120)] = 7$. A Tabela 2 especifica os rótulos de distância determinada pelo algoritmo de Dijkstra após análise do vértice 1.

| vértice $i$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| rótulo d[$i$] | 0 | 13 | 0 | 15 | 20 | $\infty$ |

| bucket $k$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| intervalo($k$) | [0] | [1] | [2, 3] | [4, 7] | [8, 15] | [16, 31] | [32, 63] | [64, 127] |
| conteudo($k$) | {3} | $\emptyset$ | $\emptyset$ | $\emptyset$ | {2, 4} | {5} | $\emptyset$ | |

Tabela 2: Radix Heap inicial

Para selecionar o vértice com o rótulo de menor distância, os buckets 0, 1, 2, ..., K são percorridos para encontrar o primeiro bucket não vazio. No exemplo, o bucket 0 é não vazio. Uma vez que o bucket 0 tem largura 1, cada vértice neste bucket tem o mesmo (no mínimo) rótulo de distância. Assim, o algoritmo determina o vértice 3 como permanente, exclui o vértice 3 do radix heap, e verifica o arco (3, 5) para alterar o rótulo de distância do vértice 5 de 20 para 9. Em seguida, é verificado se o novo rótulo de distância do vértice 5 está contido no intervalo de seu bucket presente, que é o bucket 5. Como o seu rótulo de distância diminuiu, o vértice 5 deve se mover para um bucket de menor índice. Assim, é concluída a análise sequencial dos bucket da direita para a esquerda, a partir do bucket 5, para identificar o primeiro bucket cujo intervalo contém o número 9, que é o bucket 4. O vértice 5 é movido do bucket 5 para bucket 4. A tabela 3 mostra o novo radix heap.

| vértice $i$ | 2 | 4 | 5 | 6 |
|---|---|---|---|---|
| rótulo d[$i$] | 13 | 15 | 9 | $\infty$ |

| bucket $k$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| intervalo($k$) | [0] | [1] | [2, 3] | [4, 7] | [8, 15] | [16, 31] | [32, 63] | [64, 127] |
| conteudo($k$) | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | {2, 4, 5} | $\emptyset$ | $\emptyset$ | $\emptyset$ |

Tabela 3: Radix Heap - final da iteração 1

Varrendo os buckets sequencialmente, é observado que o bucket $k = 4$ é o primeiro bucket não vazio. Uma vez que o intervalo deste bucket contém mais de um número inteiro, o primeiro vértice no bucket não precisa ter o rótulo de menor distância. Tem-se que intervalo(4) é [8, 15], mas como seu menor rótulo temporário neste bucket é 9, o novo intervalo a ser redistribuído é intervalo[9, 15] da seguinte maneira:
intervalo(0) = [9],
intervalo(1) = [10],
intervalo(2) = [11, 12],
intervalo(3) = [13, 15],
intervalo(4) = $\emptyset$.

Outros intervalos não mudam. Os vértices do bucket 4 foram redistribuídos nos buckets 0 a 3. Os buckets resultantes têm os seguintes conteúdo:
conteúdo(0) = 5,
conteúdo(1) = $\emptyset$,
conteúdo(2) = $\emptyset$,
conteúdo(3) = 2, 4,
conteúdo(4) = $\emptyset$.

Esta redistribuição esvazia necessariamente o bucket 4 e move o vértice com o rótulo de menor distância para o bucket 0.

#### 3.1.3.3 Complexidade do Algoritmo de Dijkstra com Radix Heap

Cada operação de inserção do vértice no bucket consome tempo $O(K)$. Como um vértice só pode ser movido $K$ vezes no máximo, então $O(nK)$ é um limite para o número total de movimentos de vértices. O termo $m$ significa o número de distâncias atualizadas, logo o tempo total gasto em atualizar os rótulos temporários é $O(m+nK)$. A operação de selecionar um vértice começa verificando os buckets da esquerda para a direita para encontrar o primeiro bucket não vazio $k$ no radix heap. Esta operação requer tempo $O(K)$ por iteração e $O(nk)$ no

total. A redistribuição do intervalo segue atribuindo o primeiro inteiro para bucket 0, o próximo inteiro para bucket 1, os próximos dois inteiros para bucket 2, nos próximos quatro inteiros para bucket 3, e assim por diante. Uma vez que o bucket $k$ tem uma largura inferior a $2^{k-1}$, e uma vez que as larguras dos primeiros $k$ buckets pode ser tão grande como $1,1,2,...,2^{k-2}$ até uma largura total de potencial $2^{k-1}$, o intervalo útil do bucket k sobre os buckets $0,1,...,k-1$ é redistribuído. Esta redistribuição dos intevalos e re-inserções subsequentes de vértices esvazia o bucket k e move os vértices com os rótulos de menor distância para o bucket 0 (AHUJA; MAGNANTI; ORLIN, 1993). Portanto, como $k = [log(nC)]$, o tempo de execução do algoritmo de Dijkstra com Radix Heap é $O(m+nk) = O(m+nlog(nC))$. Usando a estrutura de dados Fibonacci heap com a implementação do radix heap, é possível reduzir ainda mais a complexidade para $O(m+n\sqrt{logC})$, o que dá uma execução mais rápida do algoritmo em tempo polinomial para resolver o problema do caminho mínimo com comprimentos de arcos não negativos (AHUJA et al., 1990). A Figura 20 apresenta o pseudocódigo do algoritmo Radix Heap.

```
AlgoritmoRadixHeap
    buckets = [];
    distancias = [];
    Inicializa o rótulo de distância dos vértices
    Para i = 1 até i < vertices.length
        distancias[i] = MAXINT
    Fim Para
    Inicializa os buckets e seus intervalos
    Para i = 0 até i < buckets.length
        iniBucket = 2^(k-1)
        fimBucket = 2^k - 1
    Fim Para
    Insere os vértices dentro dos buckets correspondentes
    Enquanto todos os vértices não forem rotulados permanentemente
        menorBucket = menor bucket não vazio
        Se (o menorBucket tem largura = 1 || menorBucket.size = 1)
            verticeSelecionado = vértice do menorBucket
            Para i = 0 até i < número de arcos do verticeSelecionado
                Atualiza o rótulo de distância dos vértices
                Insere o vértice no bucket que contenha a sua faixa de valores
            Fim Para
            Remove do heap o verticeSelecionado
            Marca o rótulo verticeSelecionado como permanentemente
        Fim Se
        Senão
            Recalcula o intervalo dos buckets
            Redistribui os vertices
    Fim Enquanto
Fim RadixHeap
```

Figura 20: Radix Heap - Pseudocódigo do algoritmo

## 3.2 Caminhos em Redes Dinâmicas

Embora o Problema de Caminho Mínimo seja um dos problemas de otimização combinatória mais bem estudados na literatura (AHUJA; MAGNANTI; ORLIN, 1993), Caminhos em Redes Dinâmicas tem recebido muito menos atenção ao longo dos anos. (NANNICINI; LIBERTI, 2008) aborda duas categorias de Problema de Caminho Mínimo em Grafos Dinâmicos. O primeiro é chamado geralmente na literatura de variante dependente do tempo: o custo de um arco é o tempo de viagem, que é dado por uma função pré-determinada de tempo, que significa que o custo de um arco $(u,v)$ sobre um caminho depende do tempo a partir do caminho e do tempo já gasto para alcançar $u$. O segundo ainda não tem um nome comum na literatura: são grafos onde a função de custo muda ou é atualizada depois de um certo intervalo de tempo, mas o grafo é estático entre duas alterações da função custo.

### 3.2.1 Algoritmos de Caminho Mínimo Dinâmico

Essa seção aborda 3 tipos de estruturas:

- Topologia Estática e Atributos Dinâmicos;
- Topologia Dinâmica e Atributos Estáticos;
- Topologia Dinâmica e Atributos Dinâmicos.

#### 3.2.1.1 Topologia Estática e Atributos Dinâmicos

(LEONARD, 2013) trata o Problema de Caminho Mínimo com previsão de tempo utilizando um vetor de custos e através da aplicação do algoritmo Dijkstra modificado. O vetor de custos é composto por dados das passagens dos veículos na via como instante da passagem, velocidade, tipo e placa do veículo, que são periodicamente calculados a cada 10 minutos para cada ligação ou aresta. O algoritmo de Dijkstra é modificado para que o mesmo atualize os custos de suas arestas à medida que os tempos de percurso se modificam, pois o trânsito dos veículos nas vias descreve um comportamento dinâmico ao longo do tempo.

Por exemplo, se o tempo parcial até um determinado ponto for de 7 minutos, então é utilizado para o próximo cálculo o tempo de previsão no intervalo $t+1$, mas se o tempo parcial for de 15 minutos, logo o período de previsão é ultrapassado, e com isso utiliza-se $t+2$ para o cálculo do próximo trajeto. Se o tempo parcial estiver entre 30 e 40 minutos utiliza-se o intervalo $t+4$, entre 40 e 50 minutos $t+5$, e assim por diante. Essa abordagem só altera o

intervalo de previsão quando o valor do custo do vértice for superior a 10. A figura 24 apresenta o pseudocódigo do algoritmo Dijkstra Modificado.

**Algoritmo Dijkstra Modificado**

```
Início
    inicialize a distância para todos os nós em G = infinito
    inicialize o predecessor de todos os nós em G = vazio
    Enquanto H não estiver vazio faça
        u = o nó com menor rótulo extraído de H
        Para cada v adjacente a u:
            Se rótulo[v] > rótulo[u] + distância[u, v][t]:
                rótulo[v] = rótulo[u] + distância[u, v][t];
                predecessor[v] = u;
                atualiza a posição de v em H
            Fim Se
        Fim Para
    Fim Enquanto
Fim Dijkstra
```

Figura 21: Dijkstra Modificado - Pseudocódigo do algoritmo

A seguir, é apresentado o funcionamento do algoritmo. O exemplo na figura 22 utiliza 6 pontos representados através do grafo G(N={P1, P2, P3, P4, P5, P6}, A).

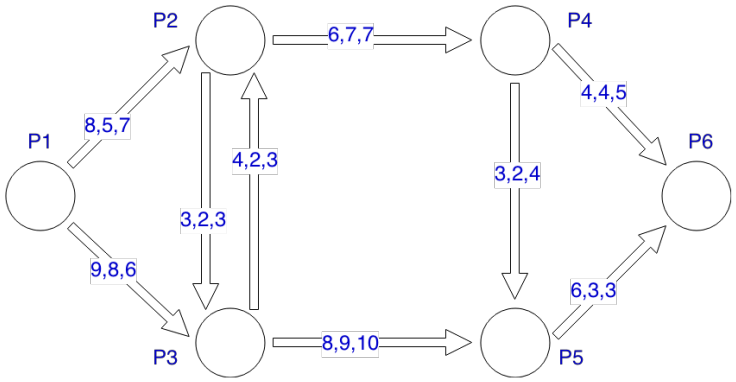


Figura 22: Grafo exemplo para determinação de caminho mínimo
Fonte: Elaboração própria, baseada em (LEONARD, 2013)

As arestas possuem vários pesos que são os tempos de percursos previstos entre os pontos para os tempos futuros $t+1$, $t+2$ e $t+3$. Esses custos são armazenados no algoritmo através de um vetor de atributos. Para determinar o caminho mínimo, utiliza-se um conjunto chamado PERM, que inicialmente contém o vértice fonte P1. A qualquer momento PERM contém todos os vértices para os quais já foram determinados os menores caminhos usando apenas vértices em PERM, a partir de P1. Para cada vértice $s$ fora de PERM mantém-se a menor

distância dentro do seu respectivo intervalo de previsão dist[$s$] de P1 a $s$ usando caminhos onde o único vértice que não está em PERM seja $s$ (LEONARD, 2013).

Outra característica do algoritmo é a necessidade de armazenar o vértice adjacente a $s$ neste caminho em path[$s$]. Selecionando o vértice com menor distância, entre todos os que ainda não pertencem a PERM, adiciona ele a PERM, chamando-o de *current*, e recalcula-se as distâncias (dist) para todos os vértices adjacentes a ele que não estejam em PERM, pois pode haver um caminho menor a partir de P1, passando por *current*, do que aquele que havia antes de *current* ser agregado a PERM. É preciso atualizar path[$s$] se houver um caminho mais curto, e com isso indicar que *current* é o vértice adjacente a $s$ pelo novo caminho mínimo (LEONARD, 2013).

Para determinar o caminho mínimo é preciso definir o ponto de origem ou nó raiz. A partir disso, o algoritmo aplica custo de valor tendendo ao infinito a todos os vértices exceto P1, que possui custo 0. A figura 23 exibe o vértice P1 como ponto de saída .

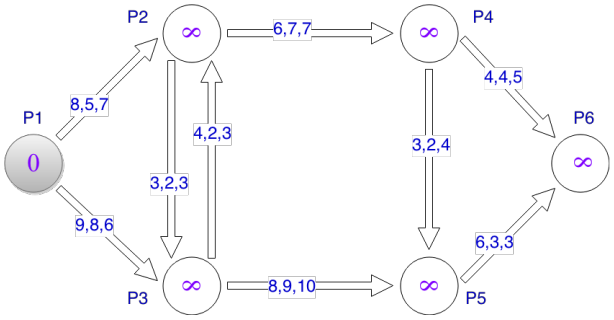


Figura 23: Processo de determinação de caminho mínimo - Parte 1

| Vértice | PERM | Distância | Predecessor |
|---|---|---|---|
| P1 | Sim | 0 | - |
| P2 | Não | ∞ | - |
| P3 | Não | ∞ | - |
| P4 | Não | ∞ | - |
| P5 | Não | ∞ | - |
| P6 | Não | ∞ | - |

Tabela 4: Processo de determinação de caminho mínimo - Parte 1

A partir de P1 consulta-se os vértices adjacentes a ele, que são P2 e P3. Para todos os vértices adjacentes denonimados $s$, calcula-se o pseudocódigo da figura 24.

```
Se dist[s] > dist[d] + peso(d,s)
    dist[s] = dist[d] + peso(d,s)
    path[s] = s
Fim Se
```

Figura 24: Dijkstra Modificado - Trecho aplicado a vértices adjacentes

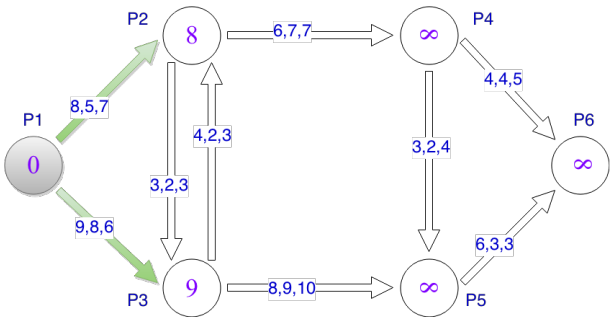


Figura 25: Processo de determinação de caminho mínimo - Parte 2

| Vértice | PERM | Distância | Predecessor |
|---|---|---|---|
| P1 | Sim | 0 | - |
| P2 | Não | 8 | P1 |
| P3 | Não | 9 | P1 |
| P4 | Não | ∞ | - |
| P5 | Não | ∞ | - |
| P6 | Não | ∞ | - |

Tabela 5: Processo de determinação de caminho mínimo - Parte 2

P2 é selecionado de PERM, pois possui a menor distância dist[x] = 8. Então inclui-se x em PERM e consulta-se seus vértices adjacentes, que no caso são P3 e P4.

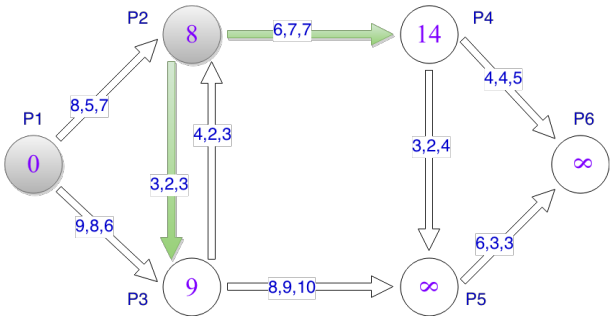


Figura 26: Processo de determinação de caminho mínimo - Parte 3

| Vértice | PERM | Distância | Predecessor |
|---|---|---|---|
| P1 | Sim | 0 | - |
| P2 | Sim | 8 | P1 |
| P3 | Não | 9 | P1 |
| P4 | Não | 14 | P2 |
| P5 | Não | ∞ | - |
| P6 | Não | ∞ | - |

Tabela 6: Processo de determinação de caminho mínimo - Parte 3

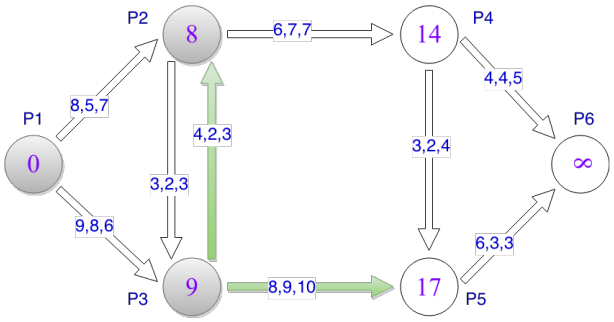


Figura 27: Processo de determinação de caminho mínimo - Parte 4

| Vértice | PERM | Distância | Predecessor |
|---|---|---|---|
| P1 | Sim | 0 | - |
| P2 | Sim | 8 | P1 |
| P3 | Sim | 9 | P1 |
| P4 | Não | 14 | P2 |
| P5 | Não | 17 | P3 |
| P6 | Não | ∞ | - |

Tabela 7: Processo de determinação de caminho mínimo - Parte 4

Ao selecionar o vértice P4, não pertencente a PERM, o intervalo de previsão é ultrapassado de 10 minutos. Logo, é utilizada a segunda posição do vetor de custos $t+2$. O custo do trajeto até o vértice P5 pelo P4 é menor do que o custo atual por P3, logo atualiza-se seu peso e predecessor para 16 e P4 respectivamente.

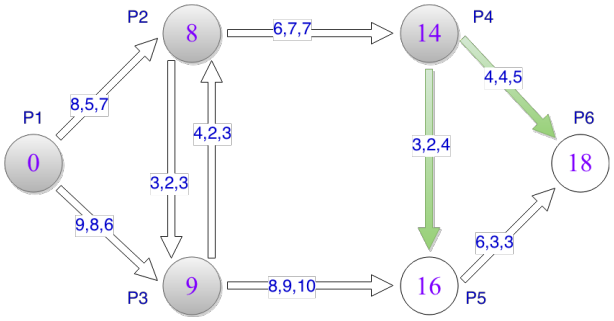


Figura 28: Processo de determinação de caminho mínimo - Parte 5

| Vértice | PERM | Distância | Predecessor |
|---|---|---|---|
| P1 | Sim | 0 | - |
| P2 | Sim | 8 | P1 |
| P3 | Sim | 9 | P1 |
| P4 | Sim | 14 | P2 |
| P5 | Não | 16 | P4 |
| P6 | Não | 18 | P4 |

Tabela 8: Processo de determinação de caminho mínimo - Parte 5

P5 é adicionado a PERM e seu único vértice adjacente é P6. Logo temos:

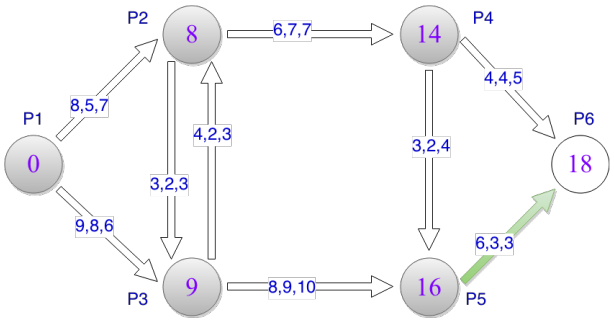


Figura 29: Processo de determinação de caminho mínimo - Parte 6

| Vértice | PERM | Distância | Predecessor |
|---|---|---|---|
| P1 | Sim | 0 | - |
| P2 | Sim | 8 | P1 |
| P3 | Sim | 9 | P1 |
| P4 | Sim | 14 | P2 |
| P5 | Sim | 16 | P4 |
| P6 | Sim | 18 | P4 |

Tabela 9: Processo de determinação de caminho mínimo - Parte 6

E finalmente temos a figura 30, pois P6 não possui vértice adjacente. Logo, o tempo para percorrer o caminho mínimo passando por P1, P2, P4 e P6 é 18 minutos.

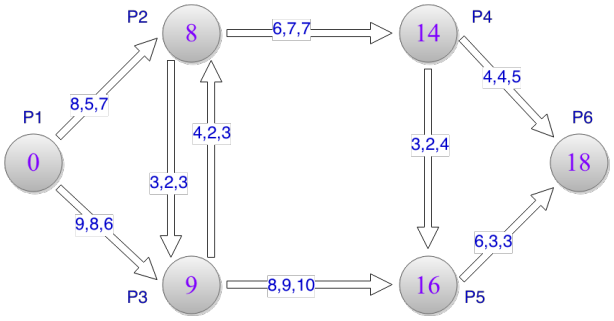


Figura 30: Processo de determinação de caminho mínimo - Parte 7

A presente pesquisa resolve o mesmo problema utilizando o Algoritmo de Dijkstra com Radix Heap para Grafos Dinâmicos. Além disso, o algoritmo indica um intervalo de limite inferior e superior. Ele utiliza um vetor de custos composto pelo tempo médio das passagens dos veículos, em intervalos de 10 minutos ao longo do dia, numa determinada ligação ou aresta. Por exemplo, entre 12 horas e 12 horas e 10 minutos um veículo demora em média 2 minutos para percorrer a aresta, entre 12 horas e 10 minutos e 12h e 20 minutos ele demora 3 minutos para percorrer a mesma aresta, e assim por diante. Os dados do vetor de custo são armazenados em um arquivo no formato JSON como mostra a figura 31. Cada aresta possui um vetor de tamanho 24 para representar as horas, e cada elemento do vetor possui um vetor de tamanho 6 para representar os intervalos de 10 minutos ao longo de 1 hora.

```
{
  "aresta 1": [
    [// 0h
      13,2,1,4,3,2
    ],
    [// 1h
      0,3,4,7,3,2
    ],
    ...
    [// 23h
      5,6,6,3,4,2
    ]
  ],
  "aresta 2": [
    ...
  ],
  "aresta 3": [
    ...
  ]
}
```

Figura 31: Estrutura JSON usada pelo vetor de custos

Nesse algoritmo, o intervalo de previsão é alterado quando o valor do custo do vértice acrescido do valor do custo da aresta adjacente for superior a 10. Por exemplo, se o tempo parcial até um determinado ponto somado ao custo de travessia da aresta adjacente for de 8 minutos, então é utilizado para o próximo cálculo o tempo de previsão no intervalo $t+1$, mas se a soma for de 13 minutos, logo o período de previsão é ultrapassado, e com isso utiliza-se $t+2$ para o cálculo do próximo trajeto. Se a soma estiver entre 20 e 30 minutos utiliza-se o intervalo $t+3$, entre 30 e 40 minutos $t+4$, e assim por diante. A figura 32 apresenta o pseudocódigo do algoritmo em questão.

```
Dijkstra com Radix Heap em Grafos Dinâmicos
    buckets = [];
    Inicializa o rótulo de distância dos vértices
    Para i = 1 até i < vertices.length
        vertices[i].rotulo = MAXINT
    Fim Para
    Inicializa os buckets e seus intervalos
    Para i = 0 até i < buckets.length
        iniBucket = 2^(k-1)
        fimBucket = 2^k − 1
        adiciona Bucket(iniBucket, fimBucket) a buckets
    Fim Para
    Insere os vértices dentro dos buckets correspondentes
    Enquanto todos os vértices não forem rotulados permanentemente
        menorBucket = menor bucket não vazio
        Se (o menorBucket tem largura = 1 || menorBucket.size = 1)
            verticeSelecionado = vértice do menorBucket
            Se verticeSelecionado não foi rotulado permanentemente
                Para i = 0 até i < verticeSelecionado.getArcos().size
                    Atualiza o rótulo do vérticeDestino
                    Insere o vértice no bucket correspondente
                Fim Para
            Fim Se
            Remove do bucket o verticeSelecionado
            Marca o rótulo verticeSelecionado permanentemente
        Fim Se
        Senão
            Recalcula o intervalo dos buckets
            Redistribui os vertices
    Fim Enquanto
Fim RadixHeap
```

Figura 32: Dijkstra com Radix Heap

A seguir, é apresentado o funcionamento do algoritmo utilizando o mesmo grafo da figura 22 e o mesmo ponto de origem. Para determinar o caminho mínimo, cada vértice armazena o identificador do vértice anterior. O algoritmo utiliza “buckets” para armazenar os vértices de acordo com seus rótulos. Inicialmente, os rótulos de todos os vértices recebem um custo de valor tendendo ao infinito exceto P1, que possui custo 0, da mesma forma que a figura 23. A figura 33 exibe a disposição dos vértices em seus respectivos “buckets”.

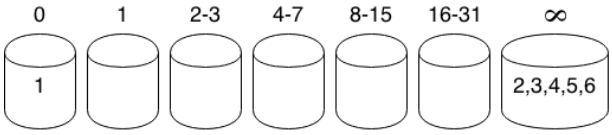


Figura 33: Disposição dos vértices nos buckets

Após consultar os vértices adjacentes a P1, que são P2 e P3, o pseudocódigo da figura 34 é executado e o mesmo resultado da figura 25 é encontrado. Em seguida, os "buckets" são atualizados como mostra a figura 35.

```
custoAresta = horário até o momento
 Se custoAresta + custoVerticeSelecionado > janelaTemporalAtual
    custoAresta = tempo de previsão do próximo intervalo
 Fim Se
 Se custoAresta + custoVerticeSelecionado < rótulo do verticeDestino
    custoAresta = custoAresta + custoVerticeSelecionado
    identificador do verticeDestino recebe identificador vértice anterior
 Fim Se
 Senão
    custoAresta = rótulo do verticeDestino
 rótulo do verticeDestino = custoAresta
```

Figura 34: Radix Heap - Pseudocódigo para rotular os vértices adjacentes

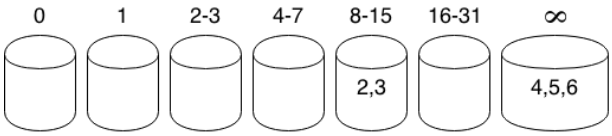


Figura 35: Disposição dos vértices nos buckets - Parte 1

Como o menor "bucket" não tem largura igual a 1 e possui mais de 1 elemento, o intervalo dos "buckets" é atualizado:

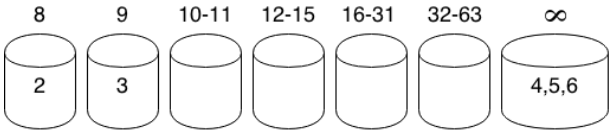


Figura 36: Disposição dos vértices nos buckets - Parte 2

O vértice P2 é selecionado do menor “bucket” e em seguida consulta-se seus vértices adjacentes, que são P3 e P4. O intervalo de previsão é ultrapassado ao somar o rótulo de P2 com o custo da aresta, que segue até P3, no intervalo $t+1$. Logo, é utilizado a próxima posição do vetor de custos, que é $t+2$. As figuras 37 e 38 exibem o resultado.

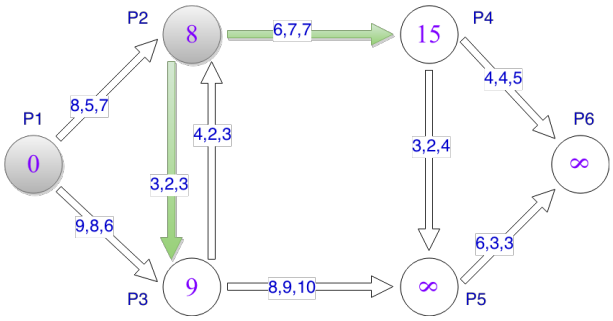


Figura 37: Radix Heap - Parte 1

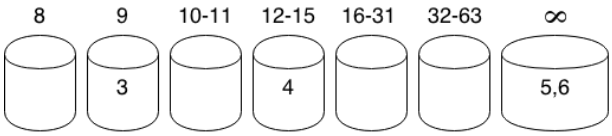


Figura 38: Disposição dos vértices nos buckets - Parte 3

Em seguida, são exibidos os próximos passos até chegar no resultado final.

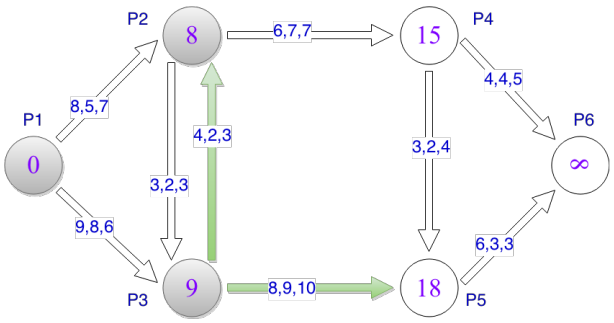


Figura 39: Radix Heap - Parte 2

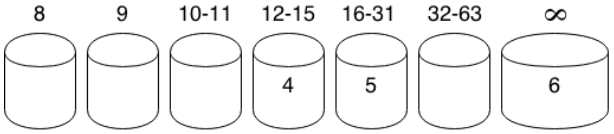


Figura 40: Disposição dos vértices nos buckets - Parte 4

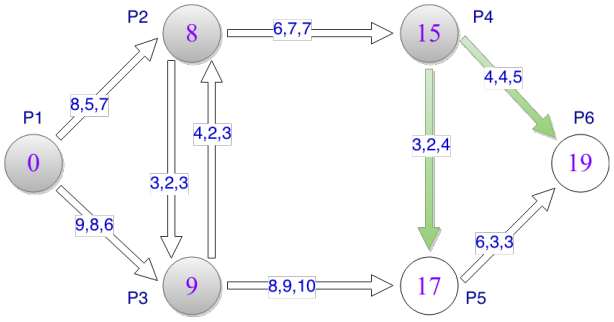


Figura 41: Radix Heap - Parte 3

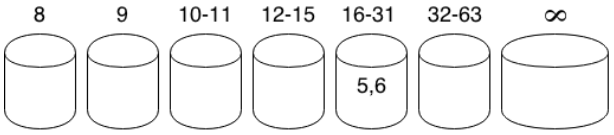


Figura 42: Disposição dos vértices nos buckets - Parte 5

Novamente o intervalo dos “buckets” é atualizado e P5 é selecionado.

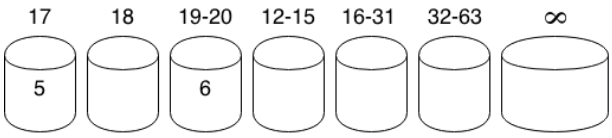


Figura 43: Disposição dos vértices nos buckets - Parte 6

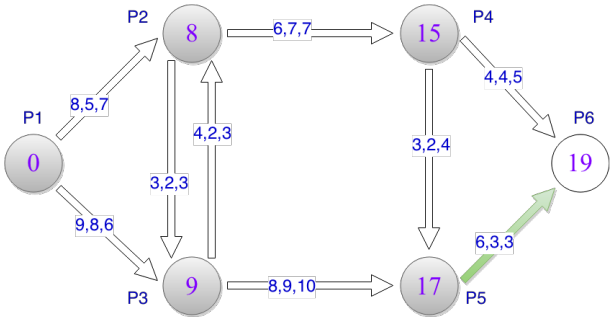


Figura 44: Radix Heap - Parte 4

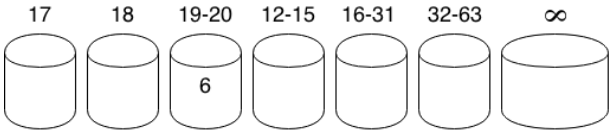


Figura 45: Disposição dos vértices nos buckets - Parte 7

Finalmente, P6 é selecionado, mas como não possui vértices adjcentes, o caminho mínimo é definido passando pelos vértices P1, P2, P4 e P6 com tempo de percurso de 19 minutos, como mosta a figura 46.

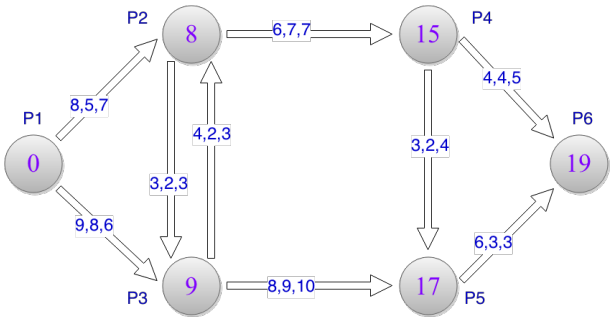


Figura 46: Radix Heap - Parte 5

#### 3.2.1.2 Topologia Dinâmica e Atributos Estáticos

Esse algoritmo não foi implementado e segue a estrutura apresentada na seção 2.2.2, onde vértices e arestas podem mudar ao longo do tempo e seus atributos são constantes. A figura 47 apresenta essa estrutura:

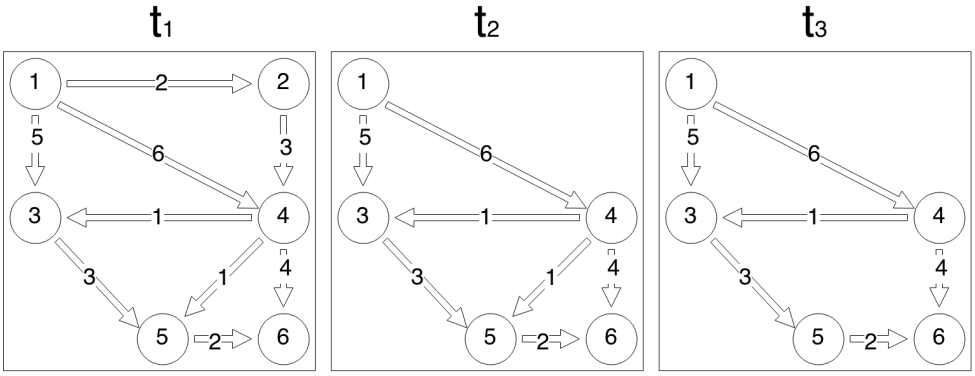


Figura 47: Caminho Mínimo - Topologia Dinâmica e Atributos Estáticos

O exemplo utiliza pontos representados através do grafo G(N={P1, P2, P3, P4, P5, P6}, A), onde o ponto de partida é P1 e o destino o ponto P6. No instante $t_1$ é possível observar vários caminhos até chegar ao objetivo. Já no instante $t_2$ o vértice P2 foi removido, e em $t_3$ a aresta que começa em P4 e vai até P5 é removida, e com isso reduzindo as possibilidades de chegar a P6. Diante disso, se faz necessário conhecer o tempo inicial e final de cada vértice e aresta para chegar a P6.

#### 3.2.1.3 Topologia Dinâmica e Atributos Dinâmicos

Essa seção resolve o problema apresentado na seção 3.2.1.1 utilizando o Algoritmo de Dijkstra com Radix Heap seguindo a estrutura da seção 2.2.3.

Ao executar o exemplo da figura 22, com o mesmo vetor de custo e ponto de origem, tem-se os vértices adjacentes P2 e P3. Ao atualizar o rótulo dos vértices adjacentes, é necessário

executar o mesmo pseudocódigo apresentado na figura 34. Além disso, é levado em consideração o tempo inicial e final dos vértices e arestas para determinar o caminho mínimo. Para isso, verifica-se as seguintes questões:

- Se a aresta já existe, ou seja, se o tempo inicial dela é menor que o horário atual;
- Se a aresta existirá durante o tempo de travessia, que correspondente ao custo dela no intervalo de previsão;
- Se os vértices origem e destino existirão durante durante a travessia.

A figura abaixo exibe o tempo de existência de cada elemento do grafo:

```
{
  "data": {
    "vertex": {
      "elements": {
        "P1": { "ini": "2014-12-16 00:00:00", "fim": null },
        "P2": { "ini": "2014-12-16 00:00:00", "fim": null },
        "P3": { "ini": "2014-12-16 00:00:00", "fim": null },
        "P4": { "ini": "2014-12-16 00:00:00", "fim": null },
        "P5": { "ini": "2014-12-16 00:00:00", "fim": "2014-12-16 00:10:00" },
        "P6": { "ini": "2014-12-16 00:00:00", "fim": null }
      }
    },
    "edge": {
      "elements": {
        "1": { "ini": "2014-12-16 00:01:00", "fim": null },
        "2": { "ini": "2014-12-16 00:00:00", "fim": null },
        "3": { "ini": "2014-12-16 00:00:00", "fim": null },
        "4": { "ini": "2014-12-16 00:00:00", "fim": null },
        "5": { "ini": "2014-12-16 00:00:00", "fim": null },
        "6": { "ini": "2014-12-16 00:00:00", "fim": null },
        "7": { "ini": "2014-12-16 00:17:00", "fim": null },
        "8": { "ini": "2014-12-16 00:18:00", "fim": null },
        "9": { "ini": "2014-12-16 00:00:00", "fim": null }
      }
    }
  }
}
```

Figura 48: Estrutura JSON do tempo de existência dos vértices e arestas

Tomando os vértices adjacentes a P1, observa-se o grafo da figura 49, onde as arestas que ligam P1 a P2 e P4 a P5 ainda não existem.

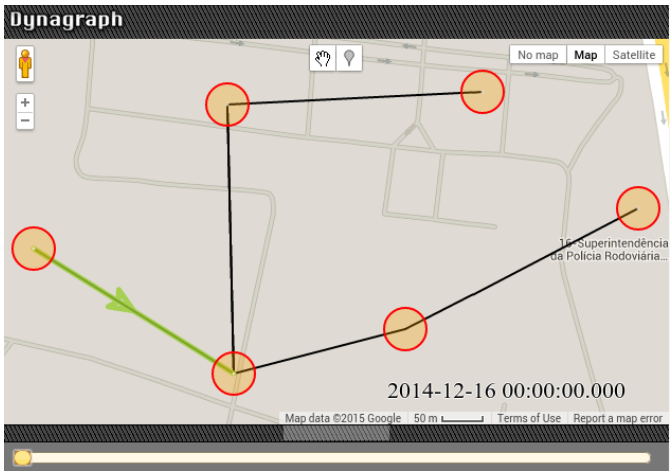


Figura 49: Radix Heap Dinâmico - Parte 1

A seguir, é apresentada a sequência de instantes utilizando o dynagraph após resolução do algoritmo.

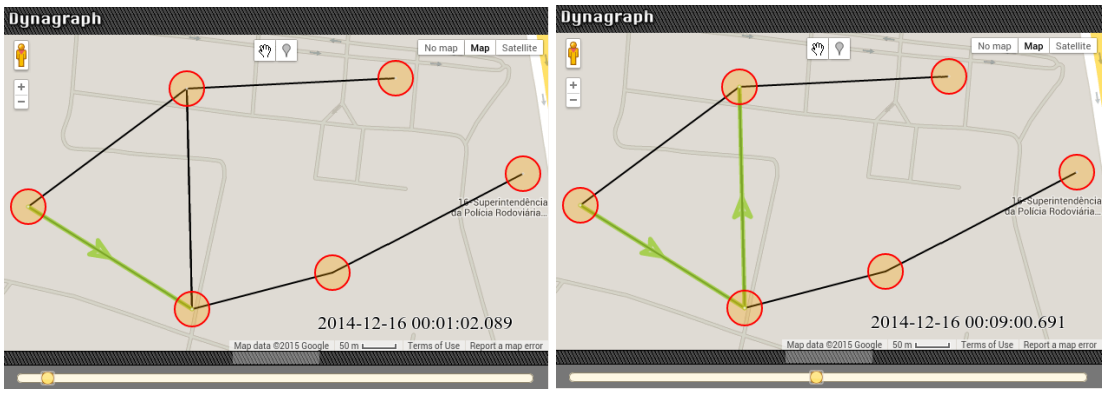


Figura 50: Radix Heap Dinâmico - Parte 2 e 3

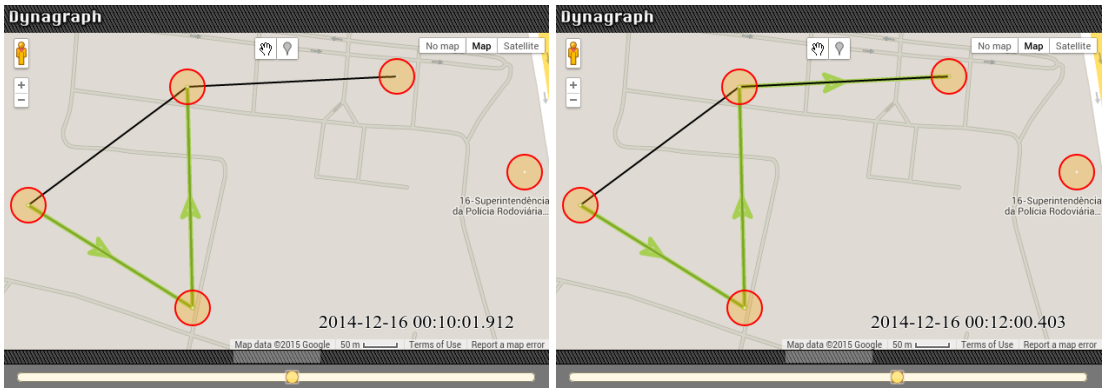


Figura 51: Radix Heap Dinâmico - Parte 4 e 5

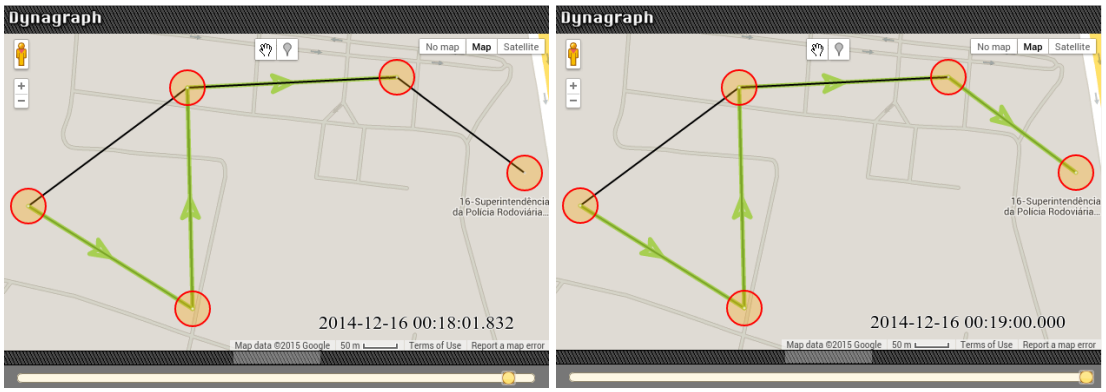


Figura 52: Radix Heap Dinâmico - Parte 6 e 7

# 4 VALIDAÇÃO

Para validação dos algoritmos foram gerados alguns grafos hipotéticos com cenários distintos. Muitos deles utilizaram o mesmo vetor de custo, como dito na seção 3.2, que representam o tempo médio das passagens dos veículos por uma aresta ao longo do dia, em intervalos de 10 minutos.

## 4.1 Ferramenta de simulação do caminho mínimo

Para determinar o caminho mínimo entre dois pontos conhecidos em grafos dinâmicos, foi criado uma extensão ao Dynagraph, que simula o caminho mínimo. Para isso, segue a sequência:

- Ler a estrutura de dados JSON;
- Exibir o grafo com todos os vértices e arestas;
- Selecionar um dos 3 algoritmos para cálculo e exibição do caminho mínimo;
- Baixar o arquivo JSON ou ir para a aplicação Dynagraph, que contém os mesmos dados do grafo mais as arestas do caminho mínimo;
- Selecionar o dynagraph e abrir o arquivo baixado ();

As figuras 53, 54 e 55 mostram captura de telas da ferramenta exibindo exemplos de caminho mínimo dos 3 algoritmos e os seus dados:

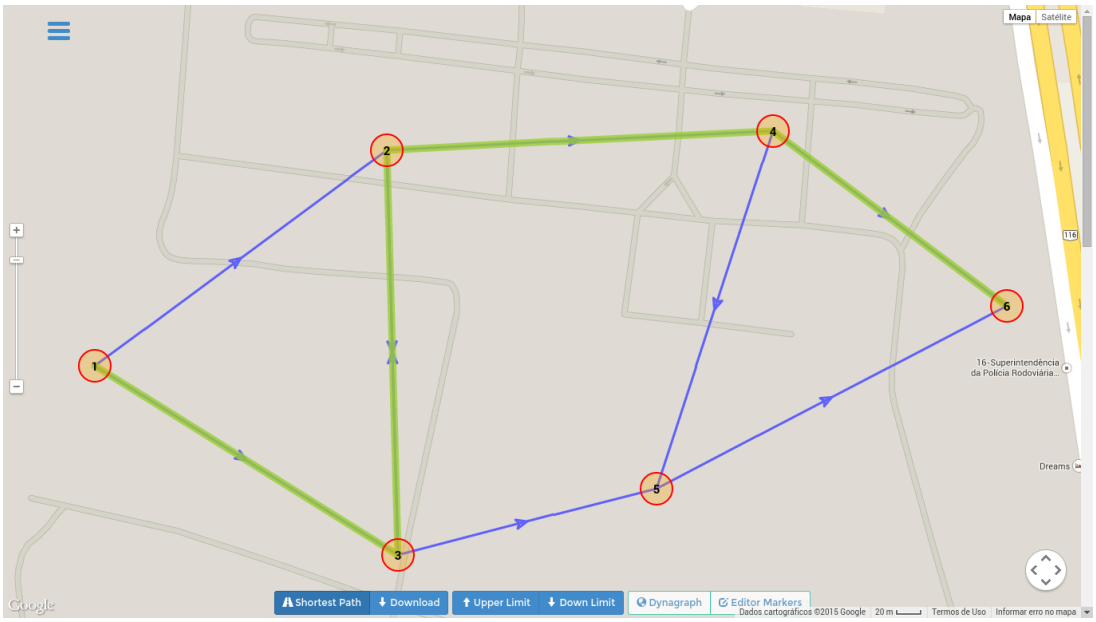


Figura 53: Simulador de Caminho Mínimo - Topologia Dinâmica e Atributos Dinâmicos

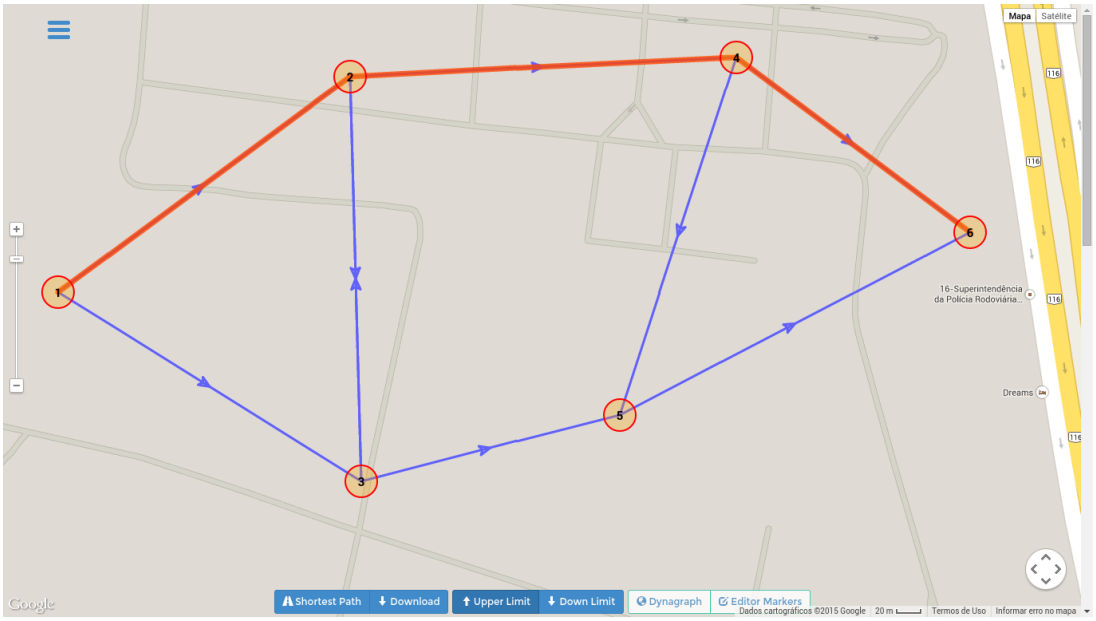


Figura 54: Simulador de Caminho Mínimo - Topologia Estática e Atributos Dinâmicos

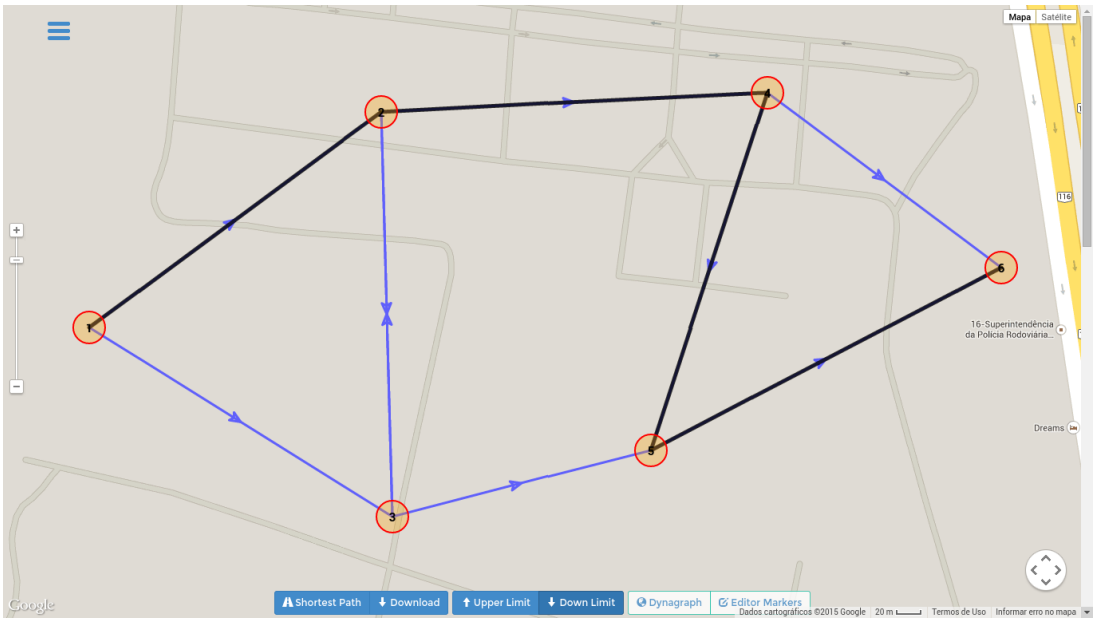


Figura 55: Simulador de Caminho Mínimo - Topologia Estática e Atributos Dinâmicos: mesma abordagem em (LEONARD, 2013)

---


```
{
  "data": {
    "vertex": {
      "elements": {
        "1": {"ini": "00:00:00", "fim": null},
        "2": {"ini": "00:00:00", "fim": null},
        "3": {"ini": "00:00:00", "fim": null},
        "4": {"ini": "00:00:00", "fim": null},
        "5": {"ini": "00:00:00", "fim": "00:10:00"},
        "6": {"ini": "00:00:00", "fim": null}
      }
    },
    "edge": {
      "elements": {
        "1": {"ini": "00:01:00", "fim": null},
        "2": {"ini": "00:00:00", "fim": null},
        "3": {"ini": "00:00:00", "fim": null},
        "4": {"ini": "00:00:00", "fim": null},
        "5": {"ini": "00:00:00", "fim": null},
        "6": {"ini": "00:00:00", "fim": null},
        "7": {"ini": "00:17:00", "fim": null},
        "8": {"ini": "00:18:00", "fim": null},
        "9": {"ini": "00:00:00", "fim": null}
      },
      "temporalfields": {
        "origemdestino": {
          "1": [{"time": "00:00:00", "data": {"origem": 1, "destino": 2 } } ],
          "2": [{"time": "00:00:00", "data": {"origem": 1, "destino": 3 } } ],
          "3": [{"time": "00:00:00", "data": {"origem": 2, "destino": 3 } } ],
          "4": [{"time": "00:00:00", "data": {"origem": 2, "destino": 4 } } ],
          "5": [{"time": "00:00:00", "data": {"origem": 3, "destino": 2 } } ],
          "6": [{"time": "00:00:00", "data": {"origem": 3, "destino": 5 } } ],
```

```
            "7": [{"time": "00:00:00", "data": {"origem": 4, "destino": 5 } } ],
            "8": [{"time": "00:00:00", "data": {"origem": 4, "destino": 6 } } ],
            "9": [{"time": "00:00:00", "data": {"origem": 5, "destino": 6 } } ]
          }
        }
      }
    }
}

//vetor de custos as 0h
{
  "1": [[8,5,7,4,3,2 ],...],
  "2": [[9,8,6,4,3,5 ],...],
  "3": [[4,2,3,4,3,2 ],...],
  "4": [[6,7,7,7,5,8 ],...],
  "5": [[3,2,3,3,2,7 ],...],
  "6": [[8,9,9,3,4,2 ],...],
  "7": [[3,2,4,2,3,5 ],...],
  "8": [[4,6,5,4,3,2 ],...],
  "9": [[6,3,3,4,3,2 ],...]
}
```

Figura 56: Estrutura JSON - Exemplo 1

A seguir, são apresentados os grafos gerados pela ferramenta de simulação com seus respectivos dados.

# 5 CONCLUSÃO E TRABALHOS FUTUROS

O presente trabalho utilizou algoritmos de caminho mínimo em grafos dinâmicos para determinação de percusos com tempo mínimo em grafos hipotéticos. A aplicação destes algoritmos resultou na formulação de um modelo computacional em grafos de topologia dinâmica e atributos dinâmicos. Também foi construído uma ferramenta (*software*) extendida do Dynagraph, que exibe o caminho mínimo entre dois pontos conhecidos.

Inicialmente foram apresentados os algoritmos de topologia estática e atributos dinâmicos com limite inferior e superior. Dentre os algoritmos implementados, o algoritmo que altera o custo de previsão antes de ultrapassar o tempo de previsão foi utilizado no algoritmo de topologia dinâmica e atributos dinâmicos.

Pode-se concluir que o algoritmo utilizado em grafos de topologia dinâmica e atributos dinâmicos é eficiente ao ponto de garantir a resolução do problema dentro dos objetivos propostos, pois ele garante um intervalo de previsão para travessia de cada aresta e intervalo para o caminho mínimo. Mesmo sabendo que o modelo não é o mais eficiente, pois o mesmo é implementado usando o algoritmo de Dijkstra com Radix Heap, e poderia usar a estrutura de dados Fibonacci heap com a implementação do radix heap, e com isso reduzir ainda mais a complexidade.

Conclui-se também, a integração com a ferramenta Dynagraph enviando o grafo com o caminho mínimo diretamente, evitando baixar o arquivo JSON. Caso o usuário deseje baixar, ele terá a opção.

Para trabalhos futuros, pretende-se resolver alguns detalhes para garantir o funcionamento da ferramenta. A aplicação ainda se limita em determinar o caminho mínimo atráves do primeiro ponto até o último ponto do grafo, pois não é possivel selecionar os vértices origem e destino. Outra limitação é a seleção de um grafo em arquivo externo, pois o mesmo seleciona um grafo na implementação. Pretende-se disponibilizar uma opção na ferramenta que indique o horário inicial de partida do caminho mínimo. Outro ponto é aperfeiçoar o Editor de Características extendendo a edição para arestas.

# BIBLIOGRAFIA

AHUJA, R.K.; MAGNANTI, Thomas L.; ORLIN, James B. *Network flows: theory, algorithms, and applications*. [S.l.]: Prentice-Hall, Inc. Upper Saddle River, NJ, 1993.

AHUJA, R.K. et al. Faster algorithms for the shortest path problem. *Journal of the Association for Computing Machinery*, v. 37, n. 2, p. 213–223, 1990.

ATZINGEN, J. V. et al. Análise Comparativa de Algoritmos Eficientes para o Problema de Caminho Mínimo. In: *XXI ANPET: Congresso de Pesquisa e Ensino em Transportes, Panorama Nacional da Pesquisa em Transportes. Associação Nacional de Ensino e Pesquisa em Transportes*. [S.l.]: Rio de Janeiro, Brasil, 2007. p. 1–12.

BASTIAN, Mathieu; HEYMANN, Sebastien; JACOMY, Mathieu. Gephi: An Open Source Software for Exploring and Manipulating Networks. In: *Internacional AAAI Conference on Weblogs and Social Media*. [S.l.]: São José, Califórnia, Estados Unidos, 2009.

CALIXTO, Anderson; NEGREIROS, Marcos. DYNAGRAPH: Um Modelo de Edição e Representação de Grafos Dinâmicos. 1 ed. CLAIO/SBPO, p. 8, 2012.

CALIXTO, Anderson; NEGREIROS, Marcos. *DYNAGRAPH: Um Modelo de Edição e Representação de Grafos Dinâmicos*. Dissertação (Mestrado) — Mestrado Acadêmico em Ciência da Computação (MACC), Universidade Estadual do Ceará, 2013.

CORMEN, Thomas H. et al. *Introduction to Algorithms*. [S.l.]: Cambridge and NewYork: The MIT Press and McGraw-Hill Book Company, 2nd. Edition, 2001.

CROCKFORD, Douglas. *JavaScript: The Good Parts*. [S.l.]: (O'Reilly Media), 2008.

GOLDBARG, Marco; GOLDBARG, Elizabeth. *Grafos: Conceitos, algoritmos e aplicações*. [S.l.]: 1a Edição, Editora Elservier, 2012.

GOODRICH, Michael T.; TAMASSIA, Roberto. *Estrutura de Dados e Algoritmos em JAVA*. [S.l.]: (4a Edição, Editora Bookman), 2007.

HARARY, F.; GUPTA, G. Dynamic Graph Models. *Mathl. Comput. Modelling*, v. 25, n. 7, p. 79–87, 1997.

KIM, Hyoungshick; ANDERSON, Ross. Temporal node centrality in complex networks. 1 ed. PHYSICAL REVIEW, p. 8, 2012.

LEONARD, Augusto C.S. *Uma Metodologia para Previsão On-line de Tempos de Viagem e Rotas Ótimas em Redes Viárias Urbanas*. Dissertação (Mestrado) — Mestrado Acadêmico em Ciência da Computação (MACC), Universidade Estadual do Ceará, 2013.

NANNICINI, G.; LIBERTI, L. Shortest paths on dynamic graphs. *International Transactions in Operational Research*, v. 15, n. 5, p. 551–563, 2008.

NEGREIROS, Marcos José. *Contribuições para otimização em grafos e problemas de percurso de Veículos: Sistema SisGRAFO*. Tese (Doutorado) — COPPE/UFRJ - Eng Sistemas - DSc Thesis, 1996.

NEGREIROS, Marcos José; MACULAN, Nelson. *Otimização em Grafos*. [S.l.]: (Não Publicado), 2014.

NETTO, P. O. Boaventura. *Grafos - Teoria, Modelos, Algoritmos*. [S.l.]: 2a edição, Editora Edgard Blücher Ltda, 1996.

PEER, S.K.; SHARMA, D.K. Finding the shortest path in stochastic networks. *Computers & Mathematics with Applications*, v. 53, n. 5, p. 729–740, 2007.

ZIVIANI, N. *Projeto de algoritmos com implementação em pascal e C*. [S.l.]: 2a Edição, Editora Pioneira Thomson Learning, 2004.