



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. Métodos Combinatoriais Para Problemas Em Redes Dinâmicas: Algoritmos De Caminho Mínimo / Bruno Bezerra Chaves. – Fortaleza, 2015. 74 p.;il. Orientador: Prof. Dr. Prof. Dr. Marcos José Negreiros Gomes Monografia (Graduação em Ciência da Computação) - Universidade Estadual do Ceará, Centro de Ciências e Tecnologia. 1. Caminho Mínimo 2. Grafos Dinâmicos I. Universidade Estadual do Ceará, Centro de Ciências e Tecnologia.
	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

Figura 1	Grafo Orientado ou Assimétrico	19
Figura 2	Grafo não Orientado ou Simétrico - $G(V,E)$	19
Figura 3	Grafo de co-autoria de alguns autores	20
Figura 4	Grafo Misto - $G(V,E,A)$	20
Figura 5	Matriz de adjacências de grafo não orientado	21
Figura 6	Matriz de adjacência de grafo orientado	21
Figura 7	Representação em Listas Encadeadas de um grafo	22
Figura 8	Representação de um grafo em Listas Duplamente Encadeadas	22
Figura 9	Grafo com Topologia Estática e Atributos Dinâmicos	24
Figura 10	Grafo com Topologia Dinâmica e Atributos Estáticos	24
Figura 11	Grafo com Topologia e Atributos Dinâmicos	25
Figura 12	Comparação entre a representação de grafos agregados(esquerda) e representação de sequência temporal(direita)	26
Figura 13	Captura de tela do software Gephi	27
Figura 14	Captura de tela do software Gephi	28
Figura 15	Estrutura JSON usada pelo Dynagraph	30
Figura 16	Capturas de tela do software Dynagraph no modo de edição de um Grafo Dinâmico	31

Figura 17	Representação das propriedades do princípio do algoritmo de caminho mí-	
	nimo	33
Figura 18	Grafo - Caminho Mínimo	36
Figura 19	Grafo exemplo para determinação de caminho mínimo	41
Figura 20	Representação do intervalo de previsão em uma ligação	41
Figura 21	Processo de determinação de caminho mínimo - Parte 1	42
Figura 22	Processo de determinação de caminho mínimo - Parte 2	43
Figura 23	Processo de determinação de caminho mínimo - Parte 3	43
Figura 24	Processo de determinação de caminho mínimo - Parte 4	44
Figura 25	Processo de determinação de caminho mínimo - Parte 5	45
Figura 26	Processo de determinação de caminho mínimo - Parte 6	45
Figura 27	Processo de determinação de caminho mínimo - Parte 7	46
Figura 28	Estrutura JSON usada pelo vetor de custos	47
Figura 29	Disposição dos vértices nos buckets	48
Figura 30	Disposição dos vértices nos buckets - Parte 1	48
Figura 31	Disposição dos vértices nos buckets - Parte 2	48
Figura 32	Radix Heap - Parte 1	49
Figura 33	Disposição dos vértices nos buckets - Parte 3	49

Figura 34 Radix Heap - Parte 2	49
Figura 35 Disposição dos vértices nos buckets - Parte 4	50
Figura 36 Radix Heap - Parte 3	50
Figura 37 Disposição dos vértices nos buckets - Parte 5	50
Figura 38 Disposição dos vértices nos buckets - Parte 6	51
Figura 39 Radix Heap - Parte 4	51
Figura 40 Disposição dos vértices nos buckets - Parte 7	51
Figura 41 Radix Heap - Parte 5	52
Figura 42 Caminho Mínimo - Topologia Dinâmica e Atributos Estáticos	52
Figura 43 Estrutura JSON do tempo de existência dos vértices e arestas	54
Figura 44 Dynagraph - Captura de tela	55
Figura 45 Radix Heap Dinâmico - Parte 1	56
Figura 46 Radix Heap Dinâmico - Parte 2 e 3	56
Figura 47 Radix Heap Dinâmico - Parte 4 e 5	57
Figura 48 Radix Heap Dinâmico - Parte 6 e 7	57
Figura 49 Simulador de Caminho Mínimo - Topologia Dinâmica e Atributos Dinâmicos	58
Figura 50 Simulador de Caminho Mínimo - Topologia Estática e Atributos Dinâmicos: sem permissão adiante	59

Figura 51	Simulador de Caminho Mínimo - Topologia Estática e Atributos Dinâmicos: com permissão adiante	59
Figura 52	Estrutura JSON - Exemplo 1	60
Figura 53	Simulador de Caminho Mínimo - Exemplo 1	61
Figura 54	Caminho Mínimo no Dynagraph - Exemplo 1	62
Figura 55	Simulador de Caminho Mínimo: Estrutura JSON - Exemplo 1	63
Figura 56	Simulador de Caminho Mínimo - Exemplo 2	63
Figura 57	Caminho Mínimo no Dynagraph - Exemplo 2	64
Figura 58	Simulador de Caminho Mínimo: Estrutura JSON - Exemplo 2	65
Figura 59	Simulador de Caminho Mínimo - Exemplo 3	66
Figura 60	Caminho Mínimo no Dynagraph - Exemplo 3	66
Figura 61	Simulador de Caminho Mínimo: Estrutura JSON - Exemplo 3	67
Figura 62	Simulador de Caminho Mínimo - Exemplo 4	67
Figura 63	Caminho Mínimo no Dynagraph - Exemplo 4	68
Figura 64	Simulador de Caminho Mínimo: Estrutura JSON - Exemplo 4	69
Figura 65	Simulador de Caminho Mínimo - Exemplo 5	69
Figura 66	Caminho Mínimo no Dynagraph - Exemplo 5	70
Figura 67	Simulador de Caminho Mínimo: Estrutura JSON - Exemplo 5	71

LISTA DE TABELAS

Tabela 1	Exemplo de contatos (arestas) em uma rede dinâmica	26
Tabela 2	Radix Heap inicial	36
Tabela 3	Radix Heap - final da iteração 1	37
Tabela 4	Processo de determinação de caminho mínimo - Parte 1	42
Tabela 5	Processo de determinação de caminho mínimo - Parte 2	43
Tabela 6	Processo de determinação de caminho mínimo - Parte 3	44
Tabela 7	Processo de determinação de caminho mínimo - Parte 4	44
Tabela 8	Processo de determinação de caminho mínimo - Parte 5	45
Tabela 9	Processo de determinação de caminho mínimo - Parte 6	46

LISTA DE ALGORITMOS

1	Dijkstra	33
2	Radix Heap	39
3	Com Permissão Adiante	41
4	Sem Permissão Adiante	47
5	Radix Heap - Topologia Dinâmica e Atributos Dinâmicos	53

SUMÁRIO

1 INTRODUÇÃO	16
1.1 Motivação	16
1.2 Objetivos	17
1.2.1 Objetivo Geral	17
1.2.2 Objetivos Específicos	17
1.3 Metodologia de Desenvolvimento	17
1.4 Organização do Trabalho	18
2 GRAFOS ESTÁTICOS E DINÂMICOS	19
2.1 Rede de Topologia Estática	20
2.2 Redes Dinâmicas	23
2.2.1 Topologia Estática e Atributos Dinâmicos	24
2.2.2 Topologia Dinâmica e Atributos Estáticos	24
2.2.3 Topologia e Atributos Dinâmicos	24
2.3 Geração e manutenção de Redes Dinâmicas	25
2.3.1 O modelo de Kim e Anderson	25
2.3.2 Gephi	26
2.3.3 O modelo Dynagraph	28
3 CAMINHOS EM GRAFOS	32
3.1 Caminhos em Redes Estáticas	32
3.1.1 Problema de Caminho Mínimo	32
3.1.2 Algoritmo de Dijkstra	33
3.1.3 Algoritmo Radix Heap	34
3.1.3.1 Operações sobre o Radix Heap	35
3.1.3.2 Funcionamento do Algoritmo de Dijkstra com Radix Heap	36
3.1.3.3 Complexidade do Algoritmo de Dijkstra com Radix Heap	38
3.2 Caminhos em Redes Dinâmicas	40
3.2.1 Algoritmos de Caminho Mínimo Dinâmico	40
3.2.1.1 Topologia Estática e Atributos Dinâmicos	40

3.2.1.2 Topologia Dinâmica e Atributos Estáticos	52
3.2.1.3 Topologia Dinâmica e Atributos Dinâmicos.....	53
4 VALIDAÇÃO.....	58
4.1 Ferramenta de cálculo do caminho mínimo.....	58
5 CONCLUSÃO E TRABALHOS FUTUROS	72
BIBLIOGRAFIA	73

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 que selecione a melhor rota para os condutores ajudaria a diminuir o tráfego intenso em determinados locais. Todas essas informações seriam geradas em tempo real e poderia usar Sistemas Inteligentes de Transporte, definidos como sistemas de transporte que usam tecnologias de informação e de telecomunicações buscando 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 (LEONARD, 2013).

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 criação 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 estático 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.

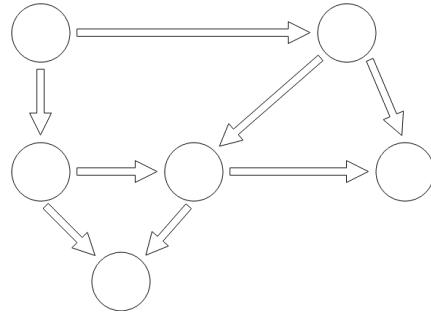


Figura 1: Grafo Orientado ou Assimétrico

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 arestas 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.

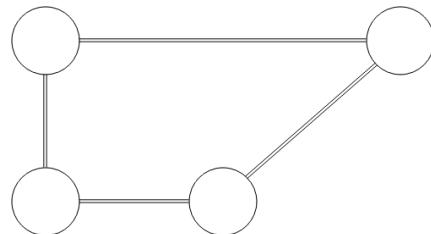


Figura 2: Grafo não Orientado ou Simétrico - $G(V, E)$

Também é possível visualizar colaborações entre pesquisadores de certa área cons-

truindo um grafo cujos vértices são associados aos pesquisadores e cujas arestas conectam pares 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.

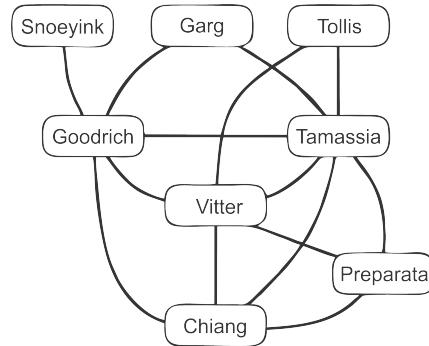


Figura 3: Grafo de co-autoria de alguns autores

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 digrafo, 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.

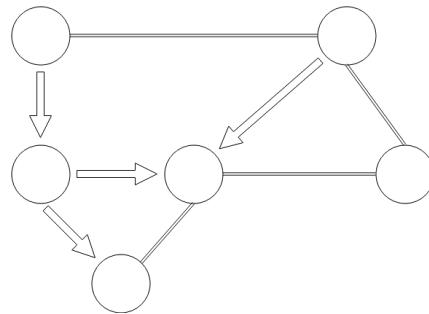


Figura 4: Grafo Misto - $G(V, E, A)$

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:

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

$$a_{ij} = 0 \text{ em caso contrário.}$$

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

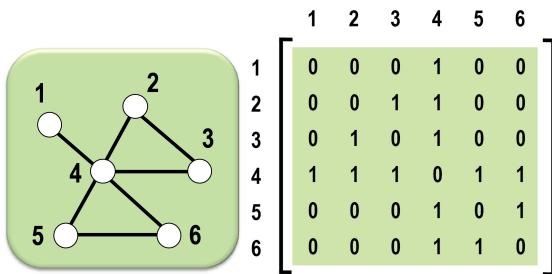


Figura 5: Matriz de adjacências de grafo não orientado
Fonte: (GOLDBARG; GOLDBARG, 2012)

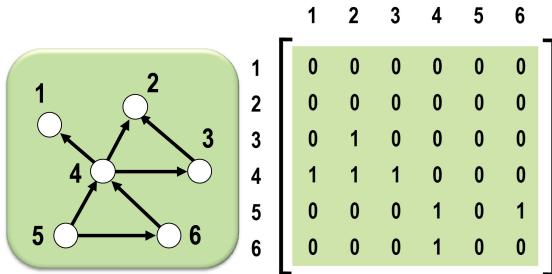


Figura 6: Matriz de adjacência de grafo orientado
Fonte: (GOLDBARG; GOLDBARG, 2012)

Em Listas Encadeadas, segundo (NEGREIROS; MACULAN, 2014) e (CORMEN et al., 2001), quando se deseja armazenar um grafo pouco denso, ou seja, $D_{max} \leq \frac{|V|}{2}$ 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;

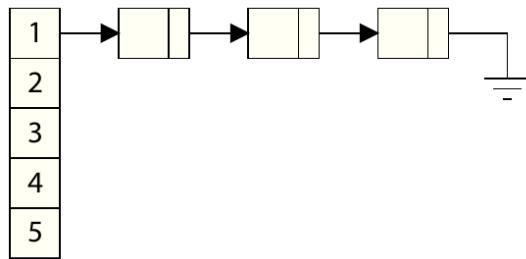


Figura 7: Representação em Listas Encadeadas de um grafo
 Fonte: (CALIXTO; NEGREIROS, 2013) apud (NEGREIROS; MACULAN, 2014)

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;

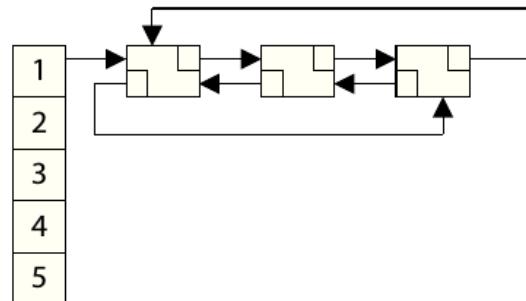


Figura 8: Representação de um grafo em Listas Duplamente Encadeadas
 Fonte: (CALIXTO; NEGREIROS, 2013) apud (NEGREIROS; MACULAN, 2014)

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, L, f) , onde V é um conjunto de vértices, L é um conjunto de ligações $\{u, v\}$, 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, L, 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), L (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 digrafo 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 digrafo com elos ou arcos dinâmicos, o conjunto L varia com o tempo. Assim, as arestas ou arcos podem ser adicionados ou removidos a partir do grafo ou digrafo;
- Em um grafo ou digrafo dinâmico de nodos ponderados, a função f varia com o tempo. Assim, os pesos nos nós também variam;
- Em um grafo ou digrafo dinâmico de elos ou arcos ponderados, a função g varia com o tempo;
- Em um grafo ou digrafo 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.

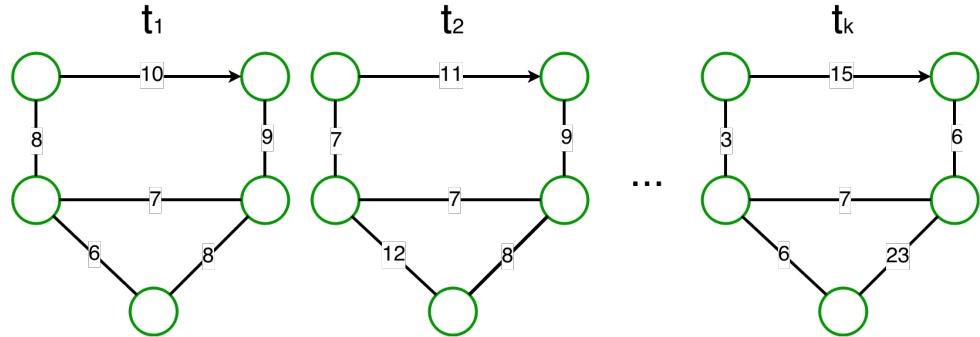


Figura 9: Grafo com Topologia Estática e Atributos Dinâmicos

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.

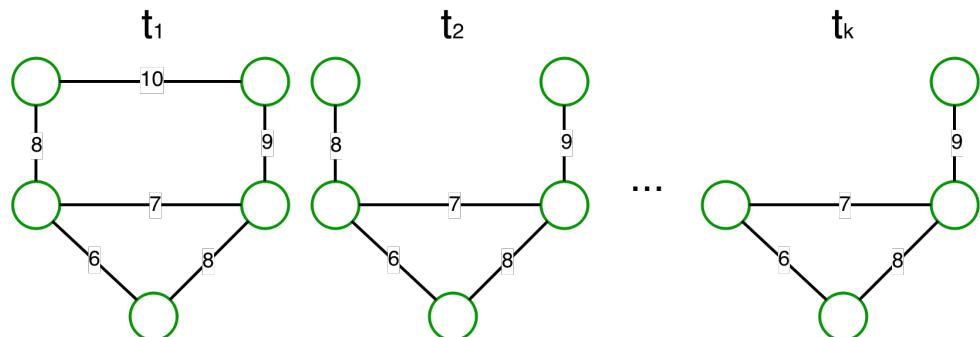


Figura 10: Grafo com Topologia Dinâmica e Atributos Estáticos

2.2.3 Topologia e Atributos Dinâmicos

Nesta rede os atributos podem ser alterados ao longo do tempo, e 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 às redes anteriormente descritas.

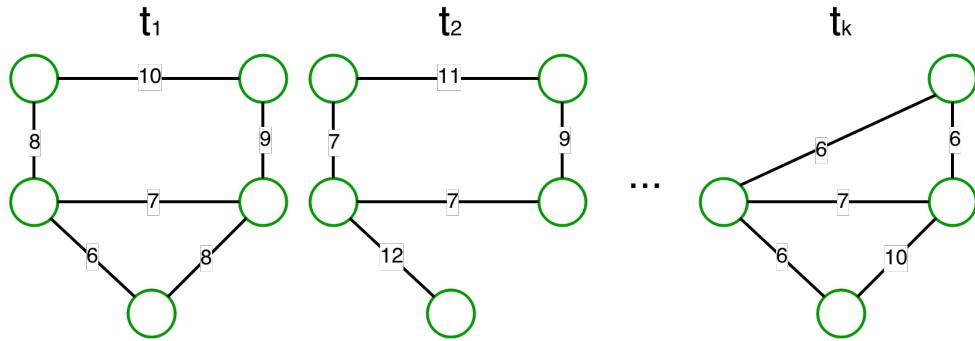


Figura 11: Grafo com Topologia e Atributos Dinâmicos

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 grafos estáticos, mas também para melhor definir métricas para grafos 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 $G_{0,T}^D = (V, E_{0,T})$ 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 $i \leq T$ e $j \geq 0$ (KIM; ANDERSON, 2012).

Uma das principais características desse modelo é que o conjunto de vértices V não muda, enquanto o conjunto de arestas 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). Uma rede dinâmica pode ser representada como uma série de grafos estáticos G_1, G_2, \dots, G_N . A notação $G_t (1 \leq t \leq n)$ 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 $i \leq wt$ e $j > w(t-1)$. G_t é o t -ésimo *snapshot* temporal de uma rede dinâmica $G_{0,T}^D$ durante a t -ésima janela de tempo (KIM; ANDERSON, 2012).

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 agregada.

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

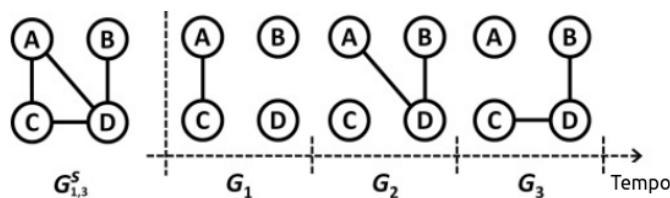


Figura 12: Comparação entre a representação de grafos agregados(esquerda) e representação de sequência temporal(direita)

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). Um adendo importante, é que vértices e/ou ligações não podem sumir e resurgir no intervalo, limitando a generalidade do Grafo Dinâmico a ser construído.

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

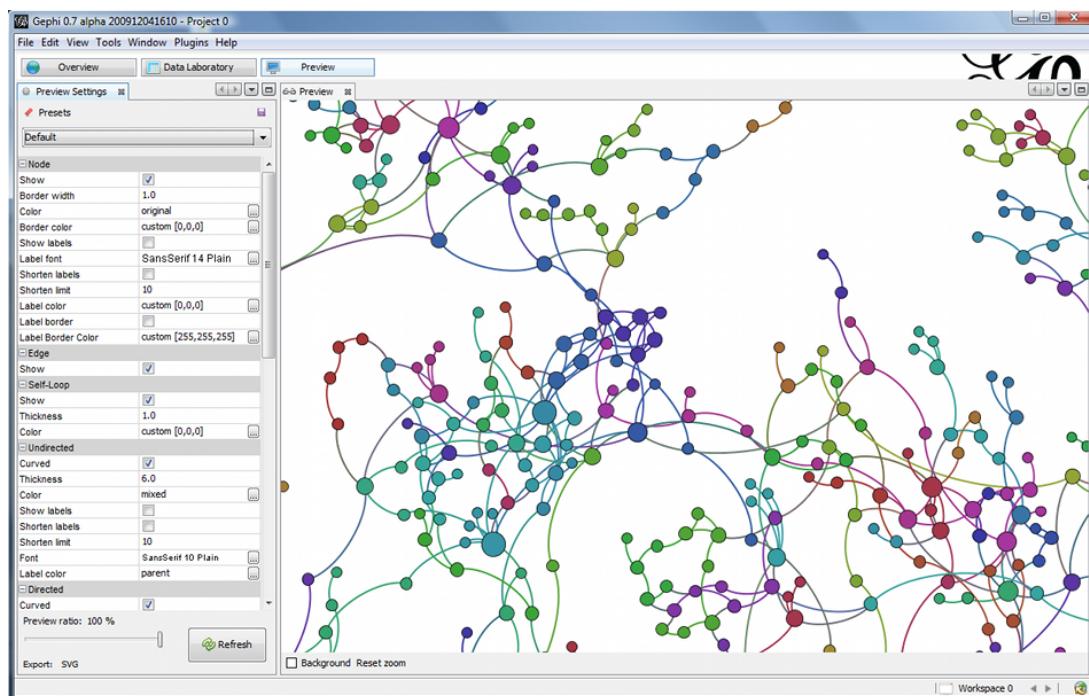


Figura 13: Captura de tela do software Gephi
Fonte: gephi.github.io

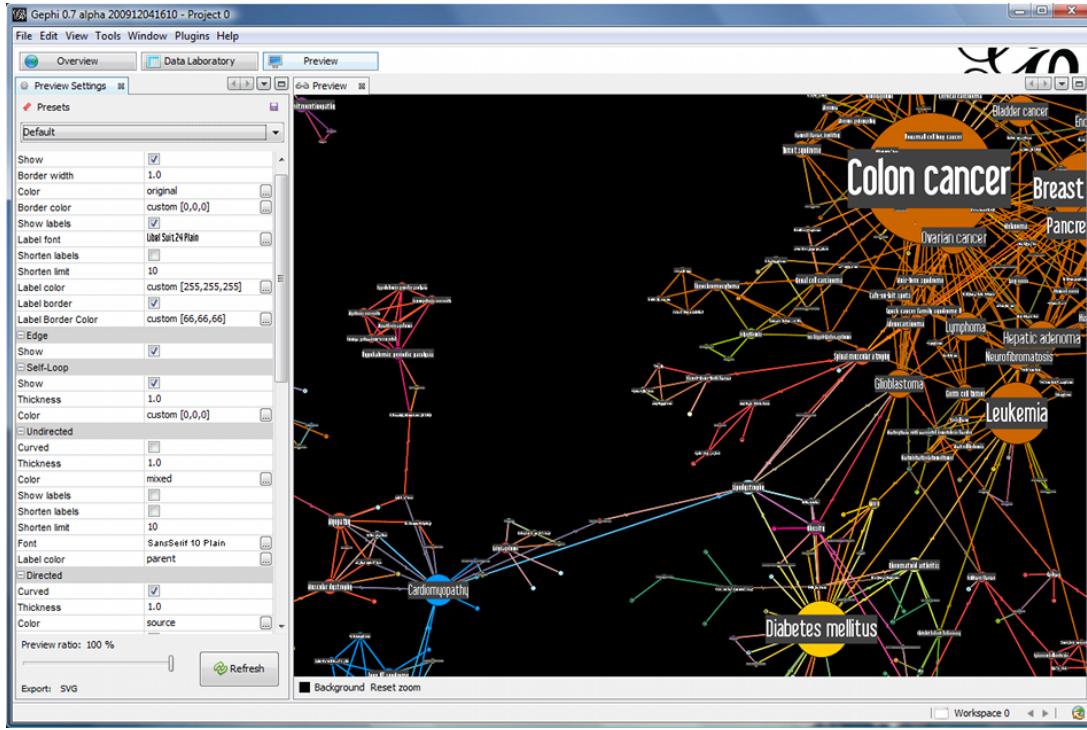


Figura 14: Captura de tela do software Gephi
Fonte: gephi.github.io

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 $O^{t_v} = \{O_{0_v}, O_{1_v}, \dots, O_{i_v}\}$ e $O^{t_e} = \{O_{0_e}, O_{1_e}, \dots, O_{i_e}\}$ são os conjuntos indexados de objetos, e $F^{t_v} = \{F_{0_v}^{t_v}, F_{1_v}^{t_v}, \dots, F_{w_v}^{t_v}\}$ e $F^{t_e} = \{F_{0_e}^{t_e}, F_{1_e}^{t_e}, \dots, F_{w_e}^{t_e}\}$ são os conjuntos indexados de características modificáveis de seus respectivos 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 15 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.

Na figura 15, em “binding” são definidas as características dos vértices e arestas. Segundo o exemplo da mesma figura, “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 à do “vertex”, porém com algumas particularidades como repetição de um símbolo ao longo da aresta, e uma customização no campo “path” seguindo a notação SVG¹.

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

```

1  {
2      "metadata": {...},
3      "binding": {...},
4      "data": {
5          "vertex": {
6              "elements": {
7                  "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

```

8         "2": {"ini": "2014-01-02 15:17:53", "fim": null},
9         "3": {"ini": "2014-01-03 03:48:20", "fim": null}
10    },
11    "temporalfields": {
12      "posicao": {
13        "1": [
14          {"time": "2014-01-01 11:00:13",
15           "data": {"x": 552713, "y": 9583920, "zona": 24, "hemisferioSul": true}
16        ],
17        "2": [
18          {"time": "2014-01-02 15:17:53",
19           "data": {"x": 552931, "y": 9584080, "zona": 24, "hemisferioSul": true}
20        ],
21        "3": [
22          {"time": "2014-01-03 03:48:20",
23           "data": {"x": 552939, "y": 9583780, "zona": 24, "hemisferioSul": true}
24        ],
25        {
26          "time": "2014-01-04 03:48:20",
27           "data": {"x": 553058, "y": 9583937, "zona": 24, "hemisferioSul": true}
28        }
29      ]
30    },
31    "tipo": {
32      "1": [
33        {"time": "2014-01-01 11:00:13", "data": "v1"},
34        {"time": "2014-01-03 13:09:02", "data": "v2"}
35      ],
36      "2": [
37        {"time": "2014-01-02 15:17:53", "data": "v2"},
38        {"time": "2014-01-03 13:09:02", "data": "v1"},
39        {"time": "2014-01-04 03:48:20", "data": "v2"}
40      ],
41      "3": [
42        {"time": "2014-01-03 03:48:20", "data": "v3"}
43      ]
44    }
45  }
46 },
47 "edge": {
48   "elements": {
49     "1": {"ini": "2014-01-02 19:00:07", "fim": null},
50     "2": {"ini": "2014-01-03 03:48:20", "fim": null}
51   },
52   "temporalfields": {
53     "origemdestino": {
54       "1": [{"time": "2014-01-02 19:00:07", "data": {"origem": 1, "destino": 2}}],
55       "2": [{"time": "2014-01-04 03:48:20", "data": {"origem": 2, "destino": 3}}]
56     }
57   }
58 }
59 }
60 }

```

Figura 15: Estrutura JSON usada pelo Dynagraph

A figura 16 monstra os dados do código da figura 15 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 no modo de edição de um Grafo Dinâmico

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 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 à 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 $G = (V, A)$, onde V é o conjunto de nós e A é o conjunto de arcos os quais interligam estes nós. Considerado o número de nós $|V| = n$; 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: $(s, i), \dots (k, l), \dots (j, t) = \Gamma(s, t)$. O Problema de Caminho Mínimo (PCM) consiste em determinar um caminho entre s e t tal que a somatória dos custos unitários dos arcos 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 e o nó m em G (caminho mínimo de menor valor). O custo do caminho é dado pela 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 $i_1, \dots, i_k \in V$, 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 contexto:

1. Seja $G(V, A)$ um grafo dirigido e ponderado, onde $c_{i,j} \geq 0, \forall (i, j) \in A$ e $c_{i,j} \in \mathbb{Z}^+$;
2. G contém um caminho direto entre todo nó $s \in V$ a todo nó $t \in V$;

Propriedade 1: Se o caminho $s = i_1 \rightarrow i_2 \rightarrow \dots \rightarrow i_h = k$ é o caminho mínimo de s a

k ($\Gamma(s, k)$), então para todo nó $q = 2, 3, \dots, (t - 1)$ o subcaminho $s = i_1 \rightarrow i_2 \rightarrow \dots \rightarrow i_q$ é o menor caminho do nó fonte ao nó i_q .

Propriedade 2: Seja d o vetor que representa as distâncias mais curtas de um vértice s a qualquer vértice t de G . Então o caminho direto P de um nó fonte a um nó K é um caminho mínimo se e somente se $d(j) = d(i) + c_{ij}$ para todo arco $(i, j) \in P$.

Prova das propriedades pode ser encontrada em (AHUJA; MAGNANTI; ORLIN, 1993). As propriedades 1 e 2 conduzem ao seguinte teorema representado na figura 17:

Teorema 3.1. *Seja $C(\Gamma(s, t))$ o custo do caminho de s a t , então é certo dizer: $C(\Gamma(s, t)) = C(\Gamma(s, k)) + C(\Gamma(k, t))$ é mínimo $\iff \Gamma(s, k)$ é mínimo e $\Gamma(k, t)$ é mínimo .*

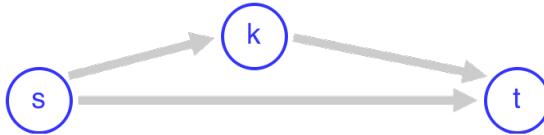


Figura 17: Representação das propriedades do princípio do algoritmo de caminho mínimo

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. Esse processo 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 menor 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.

Algorithm 1 Dijkstra

```

1: initialize a distância para todos os nós em  $G = \text{infinito}$ 
2: initialize o predecessor de todos os nós em  $G = \text{vazio}$ 
3: enquanto  $H$  não estiver vazio faça
4:    $u =$  o nó com menor rótulo extraído de  $H$ 
5:   para  $v \leftarrow 1$  até número de adjacentes a  $u$  faça
6:     se  $\text{rotulo}[v] > \text{rotulo}[u] + \text{distancia}[u, v]$  então
7:        $\text{rotulo}[v] = \text{rotulo}[u] + \text{distância}[u, v]$ 
8:       predecessor[v] = u
9:       atualiza a posição de  $v$  em  $H$ 
10:    fim se
11:  fim para
12: fim enquanto

```

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 $[100k \text{ para } 100k + 99]$ 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 $[100k, 100k + 99]$ 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:

Propriedade 3: 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.

Propriedade 4: 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 3 nos permite manter apenas $O(\log(NC))$ *buckets* e, assim, supera a desvantagem do algoritmo de Dial, que usa muitos *buckets*. A Propriedade 4 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 $O(m + n\log(nC))$ (AHUJA; MAGNANTI; ORLIN, 1993).

Para um dado problema de caminho mínimo, o radix heap consiste em formar 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) = [2K-1, 2K - 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 $O(\log nC)$. Logo, inserir um vértice na estrutura, tem complexidade $O(K)$, pois $O(K) = O(\log nC)$

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 $\text{intervalo}(0) = [9]$, $\text{intervalo}(1) = [10]$, $\text{intervalo}(2) = [11, 12]$, $\text{intervalo}(3) = [13, 15]$ e $\text{intervalo}(4) = \emptyset$. 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 demonstrar o funcionamento do algoritmo de Dijkstra com Radix Heap serão utilizadas figuras do livro (AHUJA; MAGNANTI; ORLIN, 1993). O grafo na figura 18 exemplifica o radix heap, e o número ao lado de cada arco indica o seu comprimento.

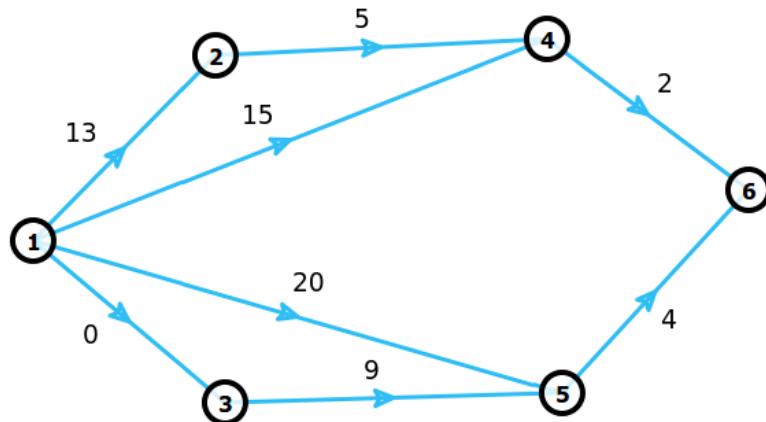


Figura 18: Grafo - Caminho Mínimo

O vértice origem é $s = 1$. O peso da maior aresta é $C = 20$, logo $K = \lceil \log(nC) \rceil = \lceil \log(120) \rceil = 7$. 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	∞		
bucket k	0	1	2	3	4	5	6	
intervalo(k)	[0]	[1]	[2, 3]	[4, 7]	[8, 15]	[16, 31]	[32, 63]	[64, 127]
conteúdo(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	∞			
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]
conteúdo(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 $\text{intervalo}(4)$ é [8, 15], mas como seu menor rótulo temporário neste *bucket* é 9, o novo intervalo a ser redistribuído é o intervalo [9, 15] da seguinte maneira:

$$\text{intervalo}(0) = [9],$$

$$\text{intervalo}(1) = [10],$$

$$\text{intervalo}(2) = [11, 12],$$

$$\text{intervalo}(3) = [13, 15],$$

$$\text{intervalo}(4) = \emptyset.$$

Outros intervalos não mudam. Os vértices do *bucket* 4 foram redistribuídos nos *buckets* 0 a 3. Os *buckets* resultantes têm os seguintes conteúdo:

$$\text{conteúdo}(0) = 5,$$

$$\text{conteúdo}(1) = \emptyset,$$

$$\text{conteúdo}(2) = \emptyset,$$

$$\text{conteúdo}(3) = 2, 4,$$

$$\text{conteúdo}(4) = \emptyset.$$

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 $1, 1, 2, \dots, 2^{k-2}$ até uma largura total de potencial $2k - 1$, o intervalo útil do *bucket* k sobre os *buckets* $0, 1, \dots, k - 1$ é redistribuído. Esta redistribuição dos intervalos 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 $k = \lceil \log(nC) \rceil$, o tempo de execução do algoritmo de Dijkstra com Radix Heap é $O(m + nk) = O(m + n\log(nC))$. Usando a estrutura de dados Fibonacci heap com a implementação do radix heap, é possível reduzir ainda mais a complexidade para $O(m + n\sqrt{\log C})$, 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). O algoritmo 2 apresenta o pseudocódigo do Radix Heap.

Algorithm 2 Radix Heap

```

1: buckets = []
2: distancias = []
3: Inicializa o rótulo de distância dos vértices
4: para i  $\leftarrow$  1 até número de vértices faça
5:   distancias[i]  $\leftarrow$  MAXINT
6: fim para
7: Inicializa os buckets e seus intervalos
8: para i  $\leftarrow$  0 até número de buckets faça
9:   iniBucket  $\leftarrow$   $2^{k-1}$ 
10:  fimBucket  $\leftarrow$   $2^k - 1$ 
11: fim para
12: Insere os vértices dentro dos buckets correspondentes
13: enquanto todos os vértices não forem rotulados permanentemente faça
14:   menorBucket  $\leftarrow$  menor bucket não vazio
15:   se o menorBucket tem largura = 1 || número de elementos no menorBucket = 1 então
16:     verticeSelecionado  $\leftarrow$  vértice do menorBucket
17:     para v  $\leftarrow$  0 até número de arcos do verticeSelecionado faça
18:       Atualiza o rótulo de distância dos vértices
19:       Insere o vértice no bucket que contenha a sua faixa de valores
20:     fim para
21:     Remove do heap o verticeSelecionado
22:     Marca o rótulo verticeSelecionado como permanentemente
23:   senão
24:     Recalcula o intervalo dos buckets
25:     Redistribui os vertices
26:   fim se
27: fim enquanto
  
```

3.2 Caminhos em Redes Dinâmicas

Embora o Problema de Caminho Mínimo seja um dos problemas de otimização combinatoria 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”: nele, 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

Esta seção aborda 3 diferentes tipos de grafos, onde se pretende calcular o caminho mais rápido entre pares de vértices de uma rede dinâmica.

- $G(V^t, L^t)$ com Topologia Estática e Atributos Dinâmicos;
- $G(V^t, L^t)$ com Topologia Dinâmica e Atributos Estáticos;
- $G(V^t, L^t)$ com 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 com Radix Heap 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 é chamada de Algoritmo com Permissão Adiante, pois só altera o intervalo de previsão quando o valor do

custo do vértice for superior a 10. O algoritmo 2 representa essa abordagem, e o algoritmo 3 mostra a modificação do método na linha 18 do algoritmo 2.

Algorithm 3 Com Permissão Adiante

```

1: custoAresta ← horário até o momento
2: se custoVerticeSelecionado > janelaTemporalAtual então
3:     custoAresta ← tempo de previsão do próximo intervalo
4: fim se
5: se custoAresta + custoVerticeSelecionado < custoVerticeDestino então
6:     custoVerticeDestino ← custoAresta + custoVerticeSelecionado
7:     predecessor verticeDestino ← verticeSelecionado           ▷ predecessor[v] ← u
8: fim se

```

A seguir, é apresentado o funcionamento do algoritmo. O exemplo na figura 19 utiliza 6 pontos representados através do grafo $G(N=\{P1, P2, P3, P4, P5, P6\}, A)$.

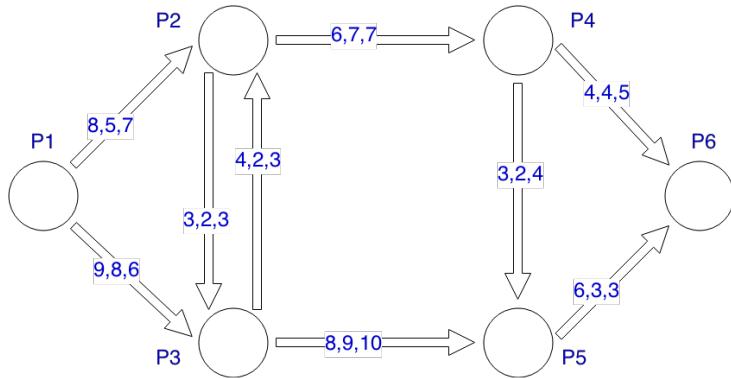


Figura 19: Grafo exemplo para determinação de caminho mínimo

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 $t + 3$, como mostra a figura 20. 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).



Figura 20: Representação do intervalo de previsão em uma ligação

Outra característica do algoritmo é a necessidade de armazenar o vértice adjacente a s neste caminho em $\text{path}[s]$. Selecionando o vértice com menor distância, entre todos os que ainda não pertencem a PERM, o adiciona 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 $\text{path}[s]$ 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 21 exibe o vértice P1 como ponto de saída. Se o caminho no vértice P1 iniciou no primeiro intervalo (0-10], e vai até P2, o percurso leva 8 minutos.

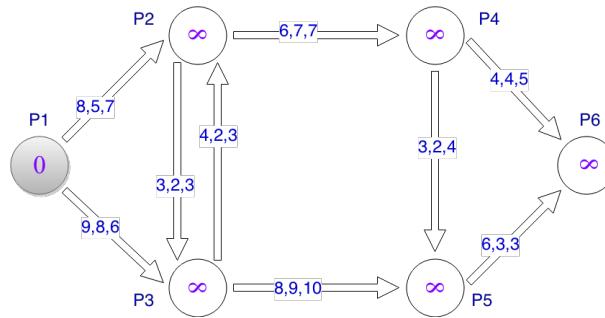


Figura 21: Processo de determinação de caminho mínimo - Parte 1

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 denominados s , calcula-se o pseudocódigo 3.

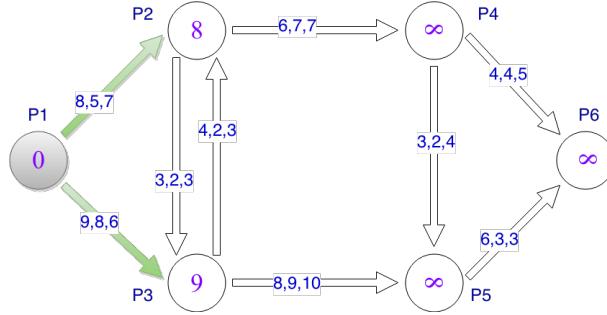


Figura 22: Processo de determinação de caminho mínimo - Parte 2

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 $\text{dist}[x] = 8$. Então inclui-se x em PERM e consulta-se seus vértices adjacentes, que no caso são P3 e P4.

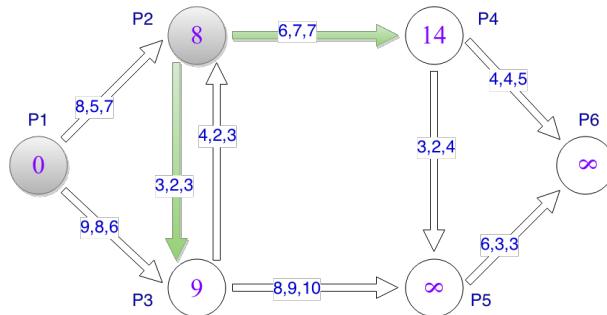


Figura 23: Processo de determinação de caminho mínimo - Parte 3

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

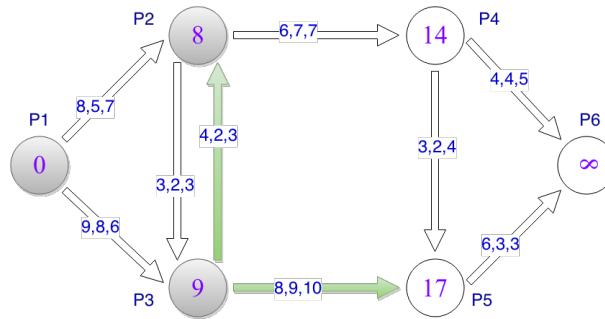


Figura 24: Processo de determinação de caminho mínimo - Parte 4

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.

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

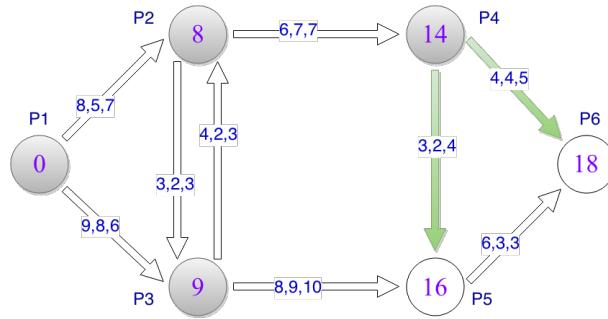


Figura 25: Processo de determinação de caminho mínimo - Parte 5

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

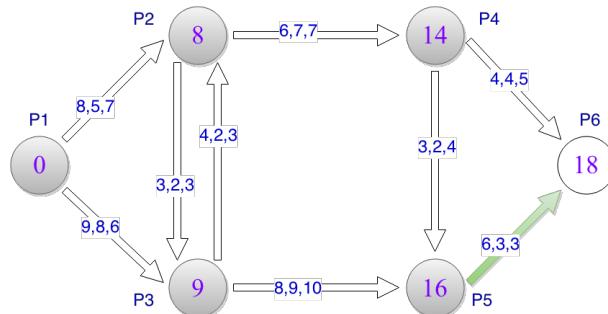


Figura 26: Processo de determinação de caminho mínimo - Parte 6

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 27, pois P6 não possui vértice adjacente. Logo, o tempo para percorrer o caminho mínimo passando por P1, P2, P4 e P6 é 18 minutos.

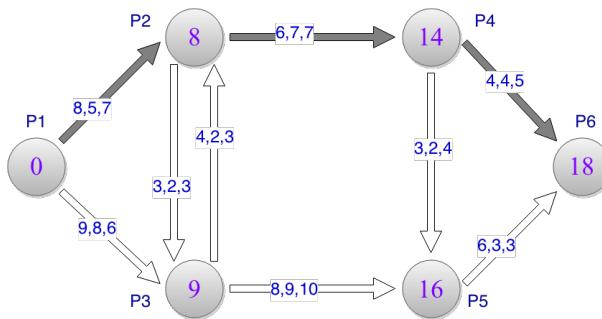


Figura 27: Processo de determinação de caminho mínimo - Parte 7

A presente pesquisa resolve o mesmo problema utilizando o Algoritmo de Dijkstra com Radix Heap para Grafos Dinâmicos. Foram implementados dois algoritmos nesse contexto: o primeiro segue a mesma abordagem do Dijkstra Modificado com Radix Heap de (LEONARD, 2013), em relação a mudança do tempo de previsão ao exceder 10 minutos, chamado Algoritmo com Permissão Adiante; e o segundo altera o intervalo de previsão quando o valor do custo do vértice acrescido do valor do custo da aresta adjacente for superior a 10, chamado Algoritmo sem Permissão Adiante. Os dois usados em conjunto indicam um intervalo de limite inferior e superior para determinação do caminho mínimo.

Os dois algoritmos utilizam 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 28. 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.

```

1   {
2     "aresta 1": [
3       [ // 0h
4         13, 2, 1, 4, 3, 2
5       ],
6       [ // 1h
7         0, 3, 4, 7, 3, 2
8       ],
9       ...
10      [ // 23h
11        5, 6, 6, 3, 4, 2
12      ]
13    ],
14    "aresta 2": [
15      ...
16    ],
17    "aresta 3": [
18      ...
19    ]
20  }

```

Figura 28: Estrutura JSON usada pelo vetor de custos

A mudança de intervalo de previsão do segundo algoritmo segue o seguinte 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. O Algoritmo sem Permissão Adiante é representado pelo algoritmo 2, e o algoritmo 4 apresenta o pseudocódigo dessa abordagem, onde mostra a modificação do método na linha 18 do algoritmo 2.

Algorithm 4 Sem Permissão Adiante

- 1: $\text{custoAresta} \leftarrow$ horário até o momento
 - 2: **se** $\text{custoAresta} + \text{custoVerticeSelecionado} > \text{janelaTemporalAtual}$ **então**
 - 3: $\text{custoAresta} \leftarrow$ tempo de previsão do próximo intervalo
 - 4: **fim se**
 - 5: **se** $\text{custoAresta} + \text{custoVerticeSelecionado} < \text{custoVerticeDestino}$ **então**
 - 6: $\text{custoVerticeDestino} \leftarrow \text{custoAresta} + \text{custoVerticeSelecionado}$
 - 7: $\text{predecessor verticeDestino} \leftarrow \text{verticeSelecionado}$ $\triangleright \text{predecessor}[v] \leftarrow u$
 - 8: **fim se**
-

A seguir, é apresentado o funcionamento do algoritmo utilizando o mesmo grafo da figura 19 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 21. A figura 29 exibe a disposição dos vértices em seus respectivos *buckets*.

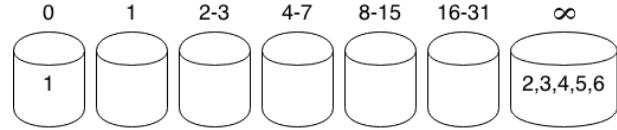


Figura 29: Disposição dos vértices nos buckets

Após consultar os vértices adjacentes a P1, que são P2 e P3, o pseudocódigo 4 é executado e o mesmo resultado da figura 22 é encontrado. Em seguida, os *buckets* são atualizados como mostra a figura 30.

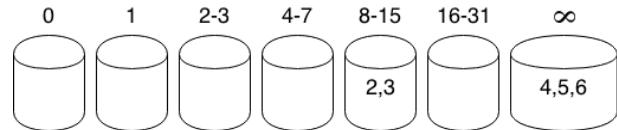


Figura 30: Disposição dos vértices nos buckets - Parte 1

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

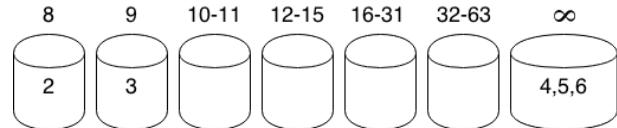


Figura 31: Disposição dos vértices nos buckets - Parte 2

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 32 e 33 exibem o resultado.

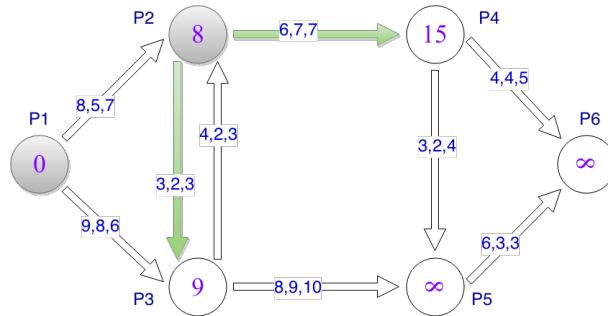


Figura 32: Radix Heap - Parte 1

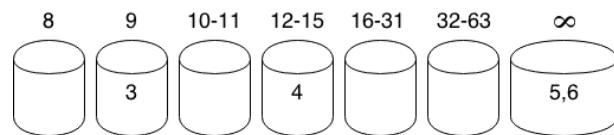


Figura 33: Disposição dos vértices nos buckets - Parte 3

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

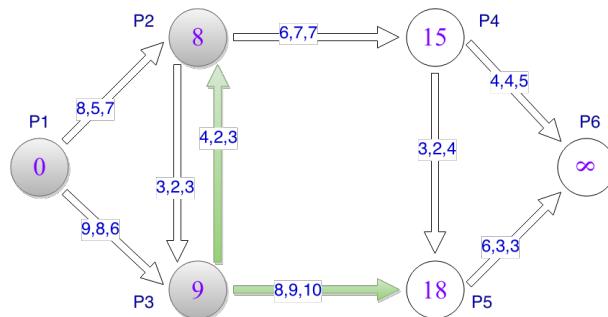


Figura 34: Radix Heap - Parte 2

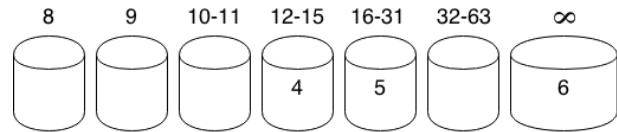


Figura 35: Disposição dos vértices nos buckets - Parte 4

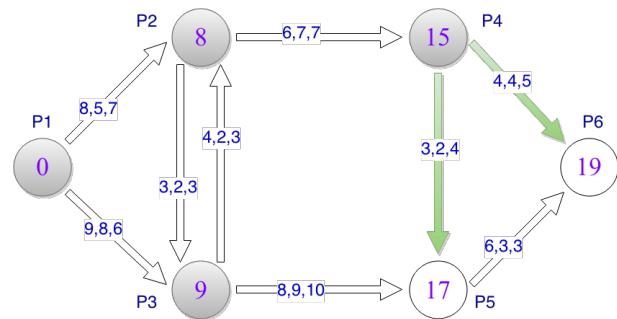


Figura 36: Radix Heap - Parte 3

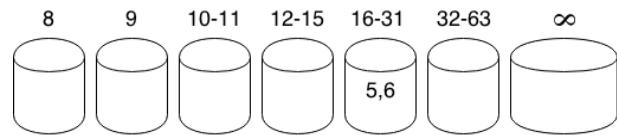


Figura 37: Disposição dos vértices nos buckets - Parte 5

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

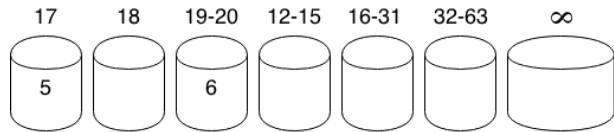


Figura 38: Disposição dos vértices nos buckets - Parte 6

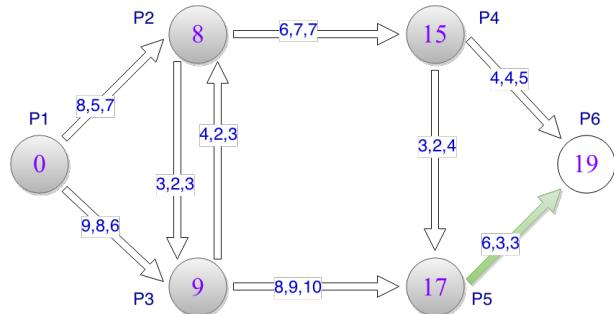


Figura 39: Radix Heap - Parte 4

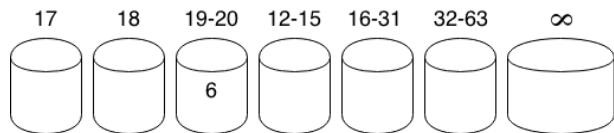


Figura 40: Disposição dos vértices nos buckets - Parte 7

Finalmente, P6 é selecionado, mas como não possui vértices adjacentes, o caminho mínimo é definido passando pelos vértices P1, P2, P4 e P6 com tempo de percurso de 19 minutos, como mostra a figura 41.

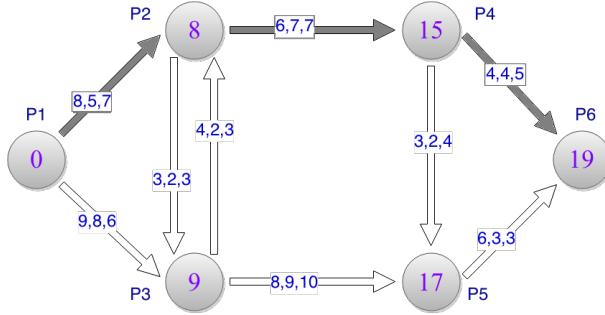


Figura 41: Radix Heap - Parte 5

Tomando-se o grafo da figura 27 e o grafo da figura 41, vimos que os percursos foram os mesmos $P1 \rightarrow P2 \rightarrow P4 \rightarrow P6$ porém os custos (tempos) foram distintos. No primeiro algoritmo, obtivemos o tempo mínimo de percurso de 18 minutos, enquanto no segundo obtivemos 19 minutos. Nota-se aqui que a visão adiante ou robusta afeta diretamente o cálculo do caminho mínimo (tempo) no grafo dinâmico, apesar de não ter afetado a descrição do percurso. Porém deve-se tomar cuidado, pois os percursos para cada método podem ser distintos.

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 42 apresenta essa estrutura:

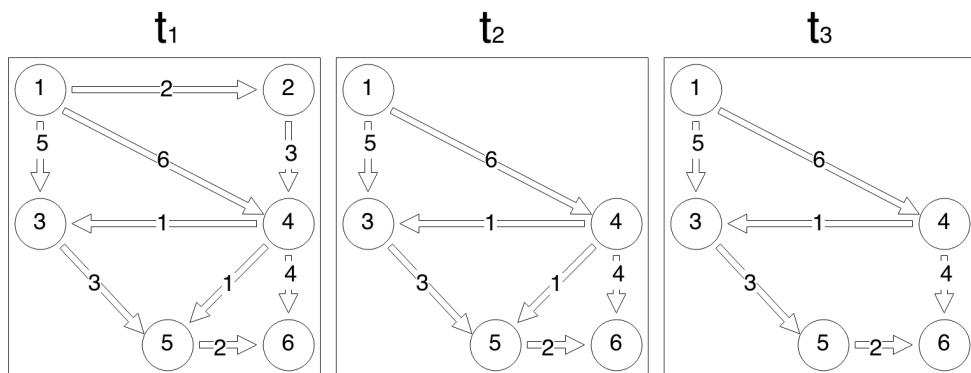


Figura 42: Caminho Mínimo - Topologia Dinâmica e Atributos Estáticos

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 19, 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 4. 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 de referência;
- Se a aresta existirá durante o tempo de travessia, que correspondente ao seu custo no intervalo de previsão;
- Se os vértices origem e destino existirão durante a travessia.

O algoritmo, fica, pois do seguinte modo:

Algorithm 5 Radix Heap - Topologia Dinâmica e Atributos Dinâmicos

```

1: função ISVALIDPATH(elemento, horarioDestino, horarioOrigem)
2:   ini ← data inicial do elemento
3:   se ini > horarioOrigem então
4:     O elemento ainda não existe devolve falso
5:   fim se
6:   se data final do elemento for diferente de nulo então
7:     fim ← data final do elemento
8:     devolve (fim > horarioDestino && horarioDestino > ini)
9:   fim se
10:  devolve horarioDestino > ini
11: fim função
12: horarioOrigem ← horário de referência
13: horarioDestino ← horário de referência
14:isValidLigacao ← isValidPath(ligacao, horarioDestino, horarioOrigem)
15:isValidVerticeOrigem ← isValidPath(verticeOrigem, horarioDestino)
16:isValidVerticeDestino ← isValidPath(verticeDestino, horarioDestino, horarioOrigem)
17: se !isValidLigacao ||| !isValidVerticeOrigem ||| !isValidVerticeDestino então
18:   custoVerticeDestino ← custoOriginal
19:   predecessor[v] ← predecessor anterior
20: fim se
```

Para ilustrar o método proposto para este tipo de rede, consideramos o seguinte grafo da figura 43, que exibe o tempo de existência de cada elemento do grafo.

```

1   {
2     "data": {
3       "vertex": {
4         "elements": {
5           "P1": { "ini": "2014-12-16 00:00:00", "fim": null },
6           "P2": { "ini": "2014-12-16 00:00:00", "fim": null },
7           "P3": { "ini": "2014-12-16 00:00:00", "fim": null },
8           "P4": { "ini": "2014-12-16 00:00:00", "fim": null },
9           "P5": { "ini": "2014-12-16 00:00:00", "fim": "2014-12-16 00:10:00" },
10          "P6": { "ini": "2014-12-16 00:00:00", "fim": null }
11        }
12      },
13      "edge": {
14        "elements": {
15          "1": { "ini": "2014-12-16 00:01:00", "fim": null },
16          "2": { "ini": "2014-12-16 00:00:00", "fim": null },
17          "3": { "ini": "2014-12-16 00:00:00", "fim": null },
18          "4": { "ini": "2014-12-16 00:00:00", "fim": null },
19          "5": { "ini": "2014-12-16 00:00:00", "fim": null },
20          "6": { "ini": "2014-12-16 00:00:00", "fim": null },
21          "7": { "ini": "2014-12-16 00:17:00", "fim": null },
22          "8": { "ini": "2014-12-16 00:18:00", "fim": null },
23          "9": { "ini": "2014-12-16 00:00:00", "fim": null }
24        }
25      }
26    }
27  }

```

Figura 43: Estrutura JSON do tempo de existência dos vértices e arestas

A seguir, é exibido o grafo dinâmico na figura 43 para acompanhar o exemplo.

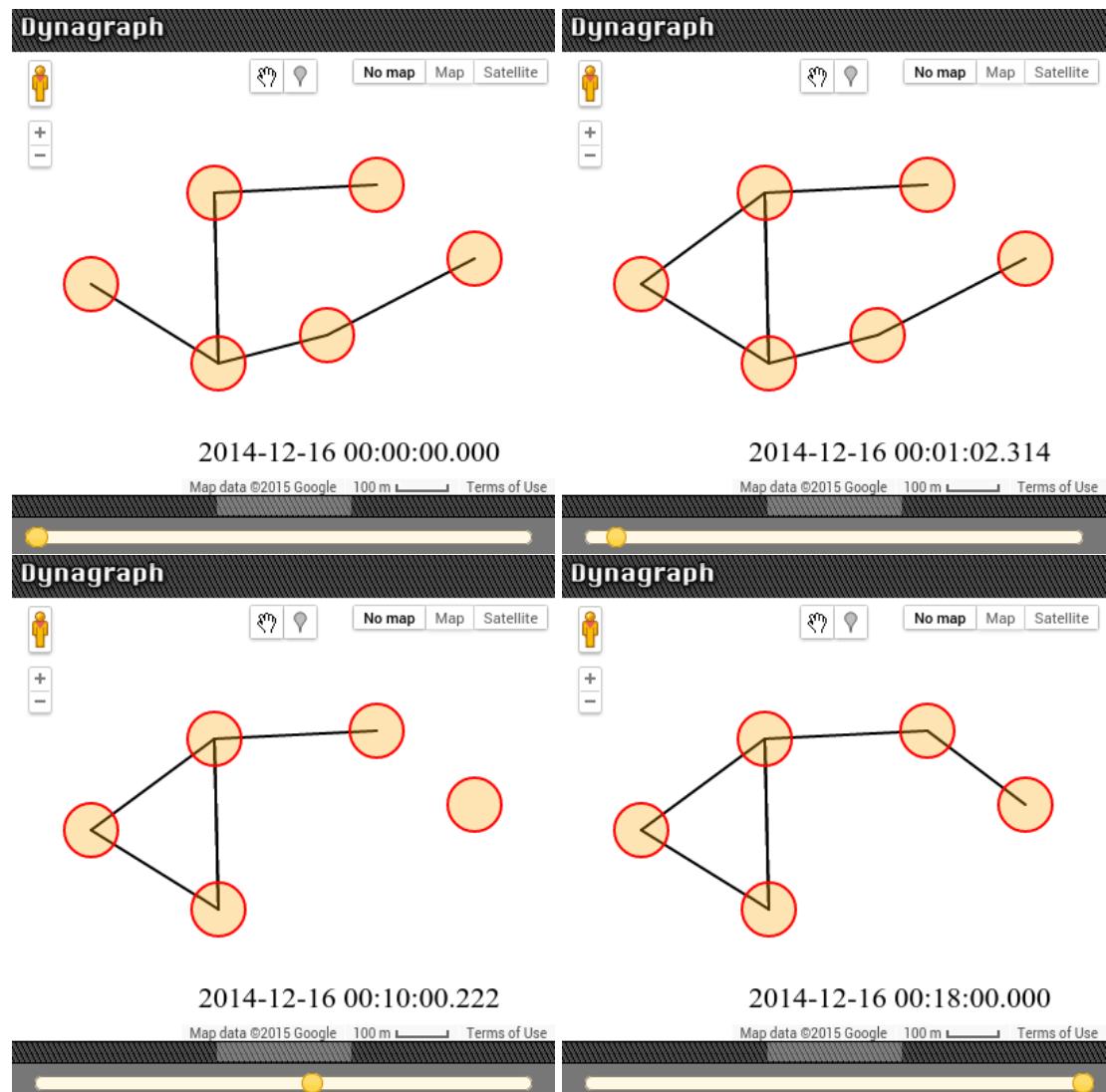


Figura 44: Dynagraph - Captura de tela

A figura 45 exibe o início da solução do caminho sobre o grafo dinâmico. Tomando os vértices adjacentes a P1, observa-se o grafo da figura 45, onde as arestas que ligam P1 a P2 e P4 a P5 ainda não existem.

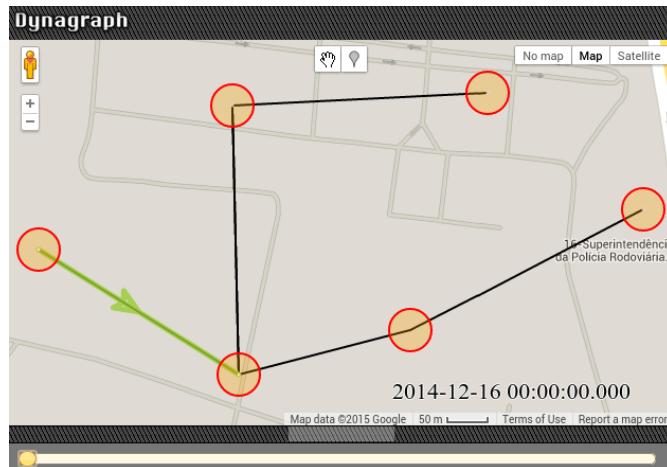


Figura 45: Radix Heap Dinâmico - Parte 1

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

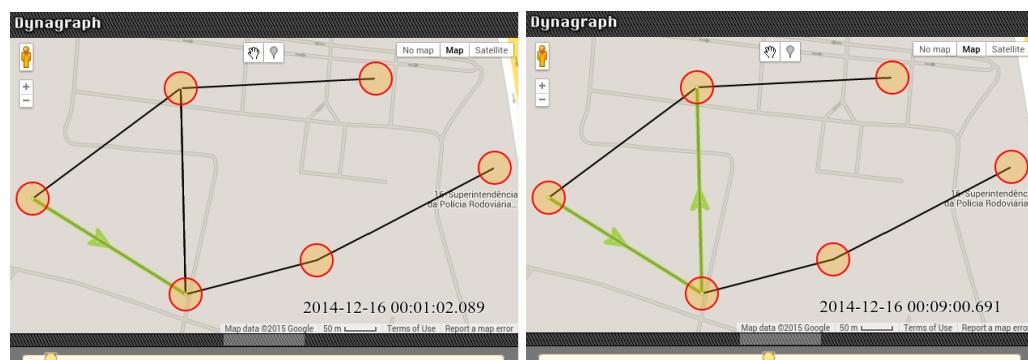


Figura 46: Radix Heap Dinâmico - Parte 2 e 3

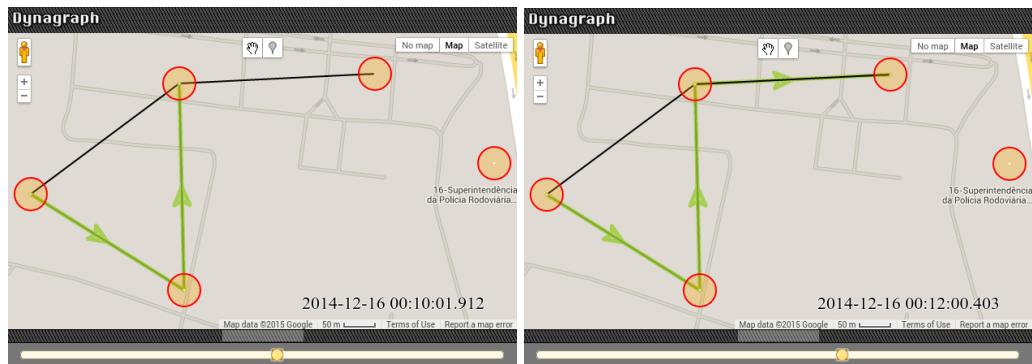


Figura 47: Radix Heap Dinâmico - Parte 4 e 5

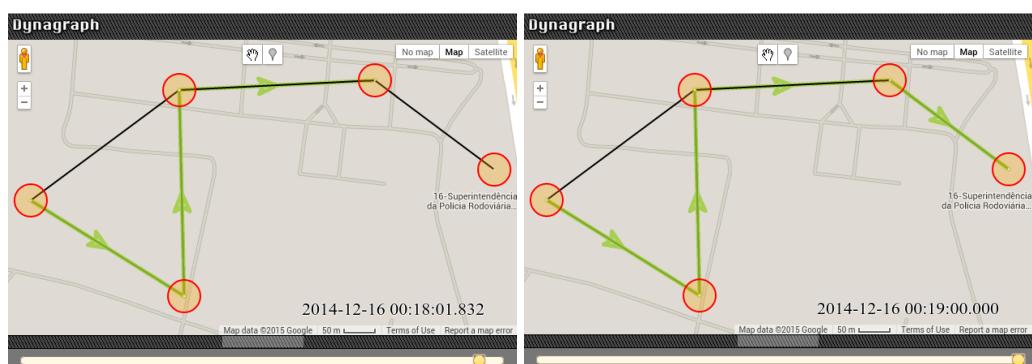


Figura 48: Radix Heap Dinâmico - Parte 6 e 7

4 VALIDAÇÃO

Para validação dos algoritmos foram gerados alguns grafos hipotéticos com cenários distintos. Alguns deles utilizam 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 padrões (10 minutos para nosso caso).

4.1 Ferramenta de cálculo do caminho mínimo

Para determinar o caminho mínimo entre dois vértices conhecidos em grafos dinâmicos, foi construída uma ferramenta (*software*) extendida do Dynagraph, que calcula o caminho mínimo. Para isso, em um grafo dinâmico segue a sequência:

- Ler a estrutura de dados JSON;
- Exibir o grafo com todos os vértices e arestas;
- Selecionar um dos 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;

As figuras 49, 50 e 51 mostram captura de telas da ferramenta exibindo exemplos de caminho mínimo dos algoritmos e os seus dados em um determinado tipo de rede:

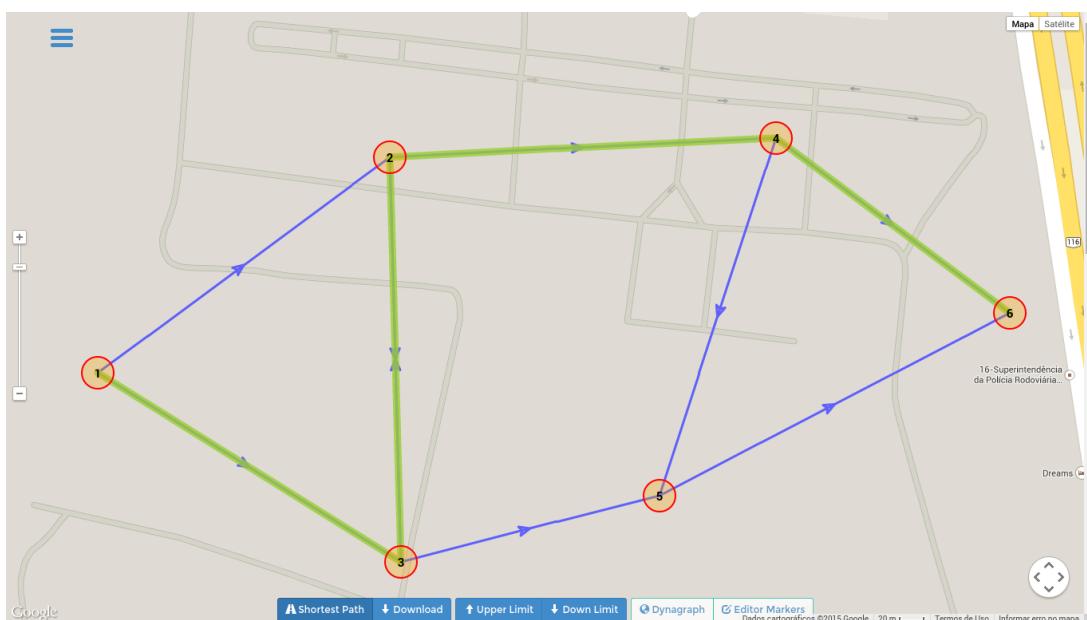


Figura 49: Simulador de Caminho Mínimo - Topologia Dinâmica e Atributos Dinâmicos



Figura 50: Simulador de Caminho Mínimo - Topologia Estática e Atributos Dinâmicos: sem permissão adiante

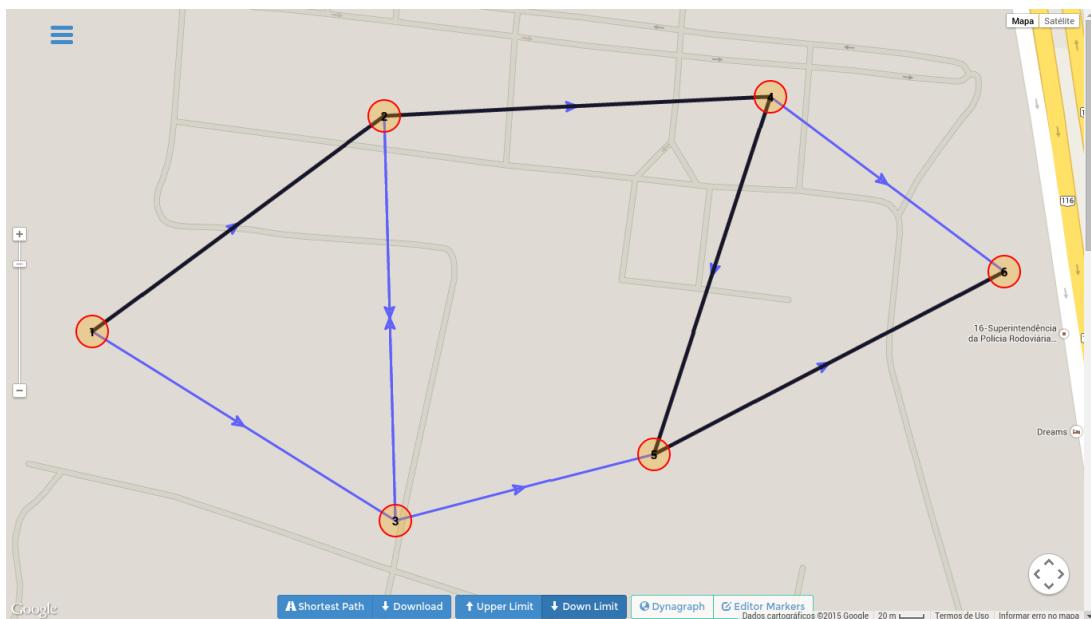


Figura 51: Simulador de Caminho Mínimo - Topologia Estática e Atributos Dinâmicos: com permissão adiante

```

1 "data": {
2     "vertex": {
3         "elements": {
4             "1": {"ini": "00:00:00", "fim": null},
5             "2": {"ini": "00:00:00", "fim": null},
6             "3": {"ini": "00:00:00", "fim": null},
7             "4": {"ini": "00:00:00", "fim": null},
8             "5": {"ini": "00:00:00", "fim": "00:10:00"},
9             "6": {"ini": "00:00:00", "fim": null}
10        }
11    },
12    "edge": {
13        "elements": {
14            "1": {"ini": "00:01:00", "fim": null},
15            "2": {"ini": "00:00:00", "fim": null},
16            "3": {"ini": "00:00:00", "fim": null},
17            "4": {"ini": "00:00:00", "fim": null},
18            "5": {"ini": "00:00:00", "fim": null},
19            "6": {"ini": "00:00:00", "fim": null},
20            "7": {"ini": "00:17:00", "fim": null},
21            "8": {"ini": "00:18:00", "fim": null},
22            "9": {"ini": "00:00:00", "fim": null}
23        },
24        "temporalfields": {
25            "origemdestino": {
26                "1": [{"time": "00:00:00", "data": {"origem": 1, "destino": 2} } ],
27                "2": [{"time": "00:00:00", "data": {"origem": 1, "destino": 3} } ],
28                "3": [{"time": "00:00:00", "data": {"origem": 2, "destino": 3} } ],
29                "4": [{"time": "00:00:00", "data": {"origem": 2, "destino": 4} } ],
30                "5": [{"time": "00:00:00", "data": {"origem": 3, "destino": 2} } ],
31                "6": [{"time": "00:00:00", "data": {"origem": 3, "destino": 5} } ],
32                "7": [{"time": "00:00:00", "data": {"origem": 4, "destino": 5} } ],
33                "8": [{"time": "00:00:00", "data": {"origem": 4, "destino": 6} } ],
34                "9": [{"time": "00:00:00", "data": {"origem": 5, "destino": 6} } ]
35            }
36        }
37    }
38 }
39 //vetor de custos as Oh
40 {
41     "1": [[8,5,7,4,3,2 ],...],
42     "2": [[9,8,6,4,3,5 ],...],
43     "3": [[4,2,3,4,3,2 ],...],
44     "4": [[6,7,7,7,5,8 ],...],
45     "5": [[3,2,3,3,2,7 ],...],
46     "6": [[8,9,9,3,4,2 ],...],
47     "7": [[3,2,4,2,3,5 ],...],
48     "8": [[4,6,5,4,3,2 ],...],
49     "9": [[6,3,3,4,3,2 ],...]
50 }

```

Figura 52: Estrutura JSON - Exemplo 1

A seguir, são apresentados os grafos gerados pela ferramenta com seus respectivos dados, e a exibição do caminho mínimo no Dynagraph. As figuras seguem a ordem da esquerda para a direita e de cima para baixo.

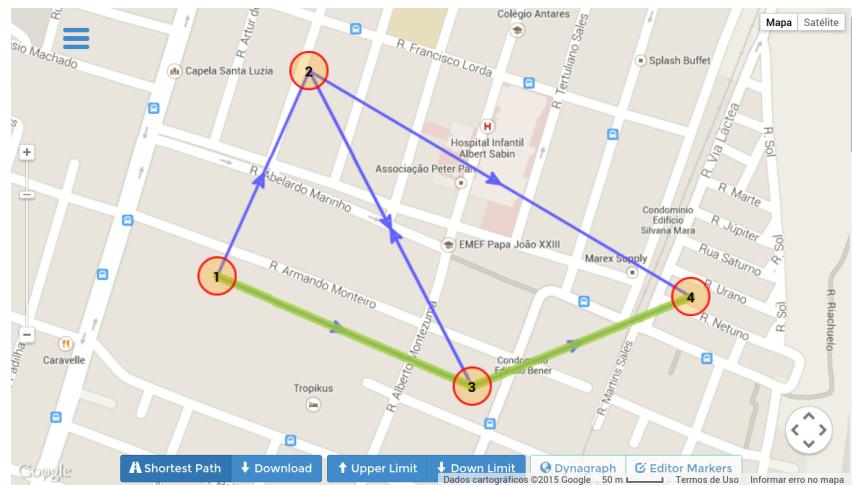


Figura 53: Simulador de Caminho Mínimo - Exemplo 1



Figura 54: Caminho Mínimo no Dynagraph - Exemplo 1

```

1 "data": {
2   "vertex": {
3     "elements": {
4       "1": {"ini": "2014-12-16 00:00:00.000000", "fim": null },
5       "2": {"ini": "2014-12-16 00:00:00.000000", "fim": null },
6       "3": {"ini": "2014-12-16 00:00:00.000000", "fim": null },
7       "4": {"ini": "2014-12-16 00:00:00.000000", "fim": null }
8     }
9   },
10  "edge": {
11    "elements": {
12      "1": {"ini": "2014-12-16 00:00:00.000000", "fim": null },
13      "2": {"ini": "2014-12-16 00:00:00.000000", "fim": null },
14      "3": {"ini": "2014-12-16 00:00:00.000000", "fim": null },
15      "4": {"ini": "2014-12-16 00:00:00.000000", "fim": "2014-12-16 00:12:00.000000"},
16      "5": {"ini": "2014-12-16 00:00:00.000000", "fim": null },
17      "6": {"ini": "2014-12-16 00:05:00.000000", "fim": null }
18    }
19  },
20  "temporalfields": {
21    "origemdestino": {
22      "1": [{"time": "2014-12-16 00:00:00.000000", "data": {"origem": 1, "destino": 2 }},
23      {"2": [{"time": "2014-12-16 00:00:00.000000", "data": {"origem": 1, "destino": 3 }},
24      {"3": [{"time": "2014-12-16 00:00:00.000000", "data": {"origem": 2, "destino": 3 }}]
25    }
26  }
27}

```

```

24     "4": [{"time": "2014-12-16 00:00:00.000000", "data": {"origem": 2, "destino": 4 }},
25         }],
26     "5": [{"time": "2014-12-16 00:00:00.000000", "data": {"origem": 3, "destino": 2 }},
27         }],
28     "6": [{"time": "2014-12-16 00:00:00.000000", "data": {"origem": 3, "destino": 4 }},
29         }],
30   }
31 //vetor de custo as 0h
32 {
33   "1": [[8,5,7,4,3,2 ],...],
34   "2": [[9,8,6,4,3,5 ],...],
35   "3": [[4,2,3,4,3,2 ],...],
36   "4": [[6,7,7,7,5,8 ],...],
37   "5": [[3,2,3,3,2,7 ],...],
38   "6": [[8,9,10,3,4,2 ],...]
39 }

```

Figura 55: Simulador de Caminho Mínimo: Estrutura JSON - Exemplo 1

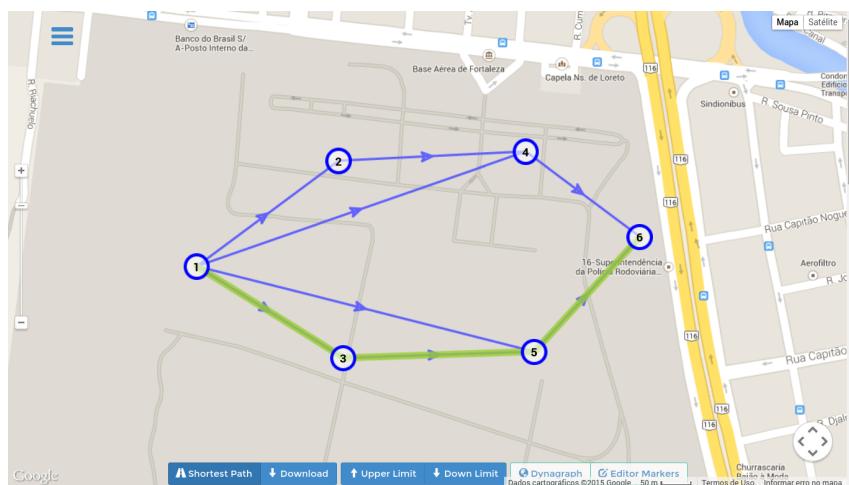


Figura 56: Simulador de Caminho Mínimo - Exemplo 2

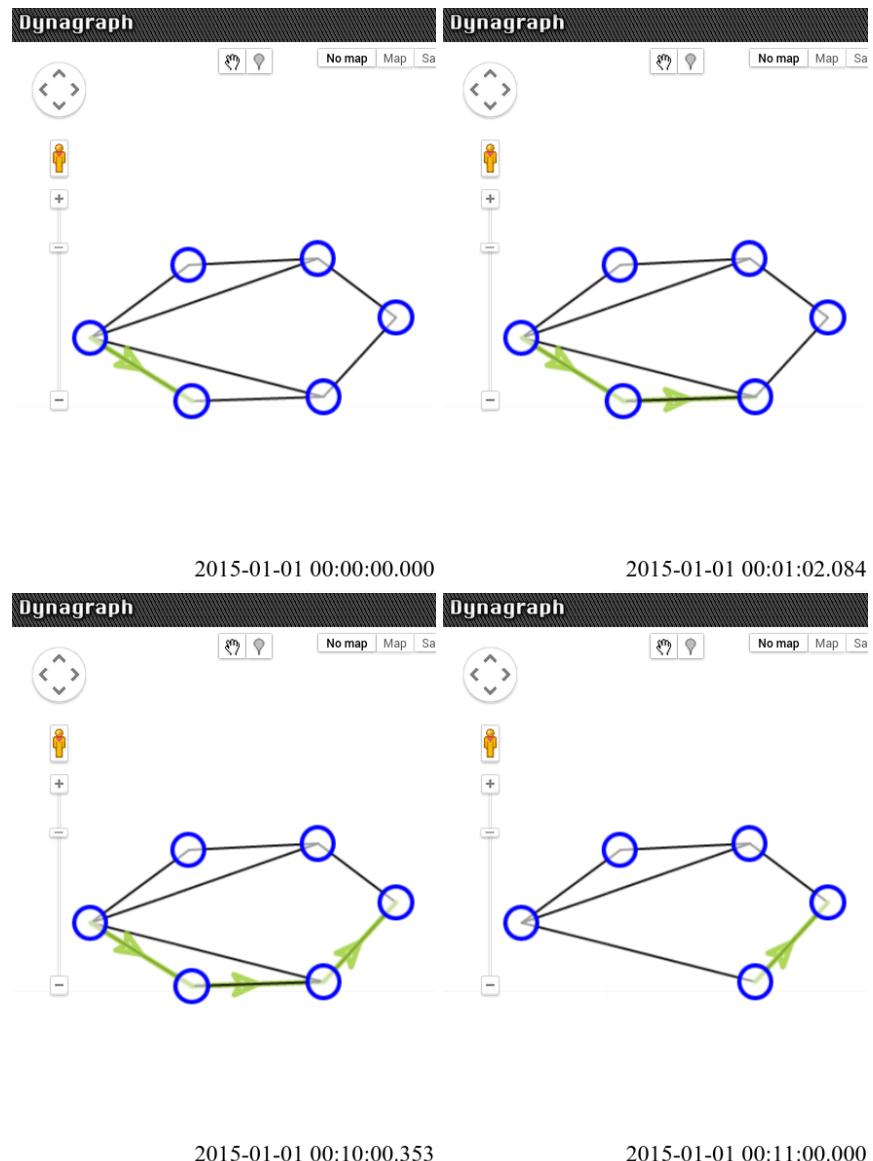


Figura 57: Caminho Mínimo no Dynagraph - Exemplo 2

```

1 "data": {
2   "vertex": {
3     "elements": {
4       "1": {"ini": "2015-01-01 00:00:00", "fim": null },
5       "2": {"ini": "2015-01-01 00:00:00", "fim": null },
6       "3": {"ini": "2015-01-01 00:00:00", "fim": "2015-01-01 00:11:00"}, 
7       "4": {"ini": "2015-01-01 00:00:00", "fim": null },
8       "5": {"ini": "2015-01-01 00:00:00", "fim": null },
9       "6": {"ini": "2015-01-01 00:00:00", "fim": null }
10    }
11  },
12  "edge": {
13    "elements": {
14      "1": {"ini": "2015-01-01 00:00:00", "fim": null },
15      "2": {"ini": "2015-01-01 00:00:00", "fim": null },
16      "3": {"ini": "2015-01-01 00:00:00", "fim": null },
17      "4": {"ini": "2015-01-01 00:00:00", "fim": null },
18      "5": {"ini": "2015-01-01 00:00:00", "fim": null },
19      "6": {"ini": "2015-01-01 00:00:00", "fim": null },
20      "7": {"ini": "2015-01-01 00:00:00", "fim": null },
21      "8": {"ini": "2015-01-01 00:00:00", "fim": null }
22    },
23    "temporalfields": {
24      "origemdestino": {
25        "1": [{"time": "2015-01-01 00:00:00", "data": {"origem": 1, "destino": 2 } }],
26        "2": [{"time": "2015-01-01 00:00:00", "data": {"origem": 1, "destino": 3 } }],
27        "3": [{"time": "2015-01-01 00:00:00", "data": {"origem": 1, "destino": 4 } }],
28        "4": [{"time": "2015-01-01 00:00:00", "data": {"origem": 1, "destino": 5 } }],
29        "5": [{"time": "2015-01-01 00:00:00", "data": {"origem": 2, "destino": 4 } }],
30        "6": [{"time": "2015-01-01 00:00:00", "data": {"origem": 3, "destino": 5 } }],
31        "7": [{"time": "2015-01-01 00:00:00", "data": {"origem": 4, "destino": 6 } }],
32        "8": [{"time": "2015-01-01 00:00:00", "data": {"origem": 5, "destino": 6 } }]
33      }
34    }
35  }
36 }
37 //vetor de custo as 0h
38 "1": [[13,2,1,4,3,2 ],...],
39 "2": [[1,2,1,4,3,5 ],...],
40 "3": [[15,7,2,4,3,2 ],...],
41 "4": [[20,2,1,7,5,8 ],...],
42 "5": [[5,5,4,3,2,7 ],...],
43 "6": [[9,6,6,3,4,2 ],...],
44 "7": [[2,2,3,2,3,5 ],...],
45 "8": [[4,2,4,4,3,2 ],...]
46 }

```

Figura 58: Simulador de Caminho Mínimo: Estrutura JSON - Exemplo 2

A seguir, o mesmo exemplo do grafo anterior, mas com o horário de início do percurso diferente. Percebe-se que a hora é ultrapassada, pois o percurso inicia às 23 horas e 50 minutos.

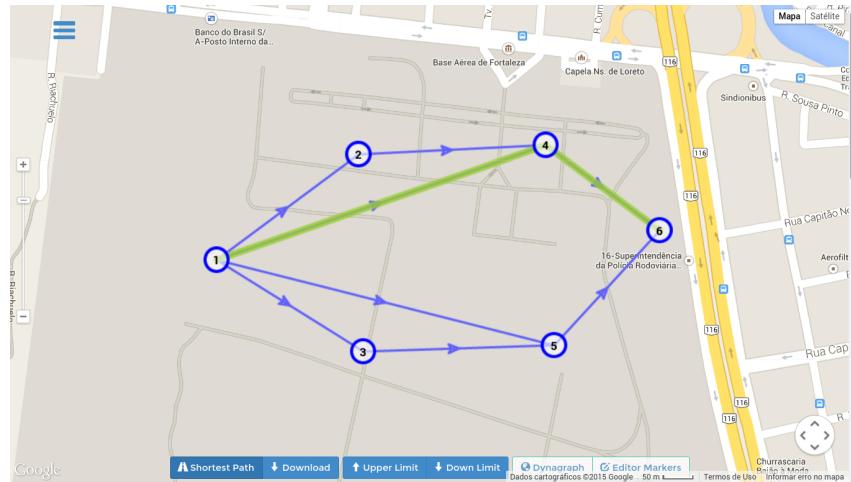


Figura 59: Simulador de Caminho Mínimo - Exemplo 3

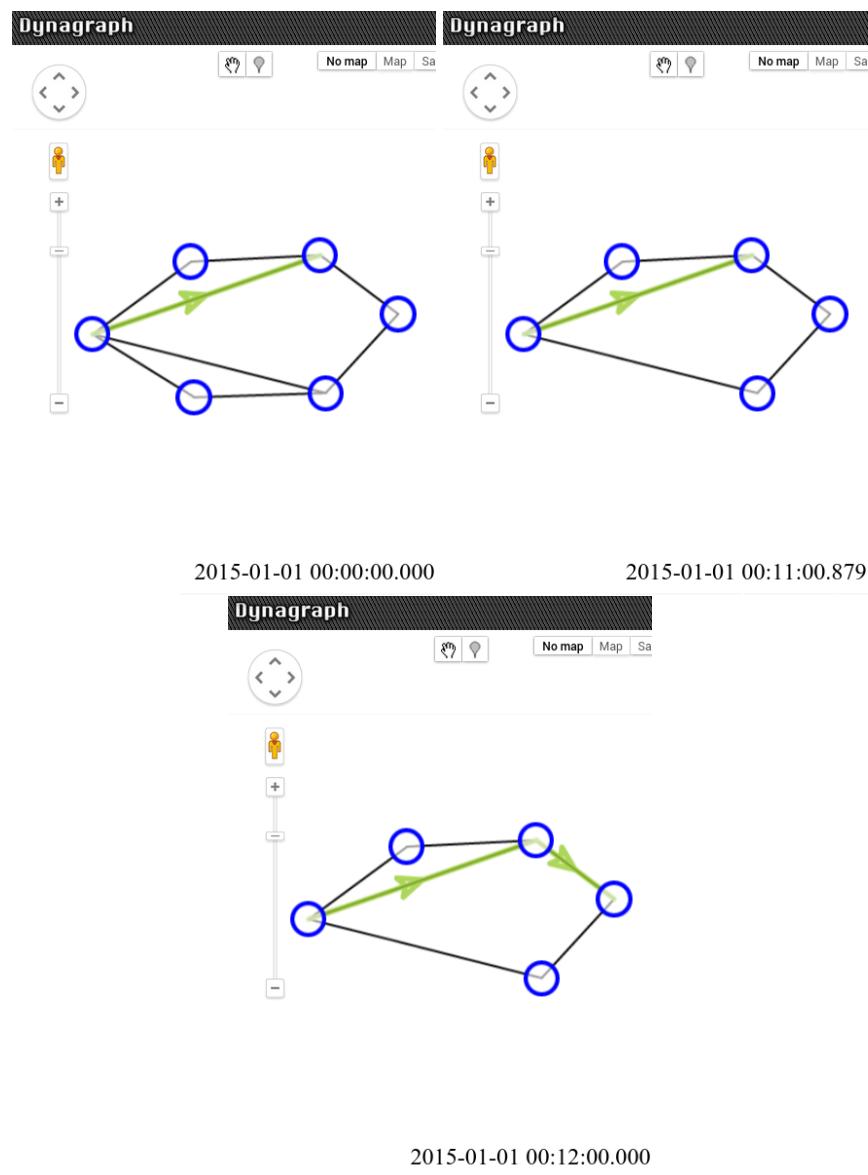


Figura 60: Caminho Mínimo no Dynagraph - Exemplo 3

```

1 //vetor de custo as 0h e 23h
2 {
3   "1": [[13,2,1,4,3,2],... [1,2,1,4,7,11]],
4   "2": [[1,2,1,4,3,5],... [2,3,4,6,6,13]],
5   "3": [[15,7,2,4,3,2],... [5,6,6,3,9,12]],
6   "4": [[20,2,1,7,5,8],... [5,6,6,3,5,18]],
7   "5": [[5,5,4,3,2,7],... [5,6,3,5,8,11]],
8   "6": [[9,6,6,3,4,2],... [5,6,2,3,7,13]],
9   "7": [[2,2,3,2,3,5],... [5,6,6,2,8,12]],
10  "8": [[4,2,4,4,3,2],... [3,4,6,3,9,15]]]
11 }

```

Figura 61: Simulador de Caminho Mínimo: Estrutura JSON - Exemplo 3

Logo abaixo, um exemplo usando o mesmo vetor de custo do exemplo 3.

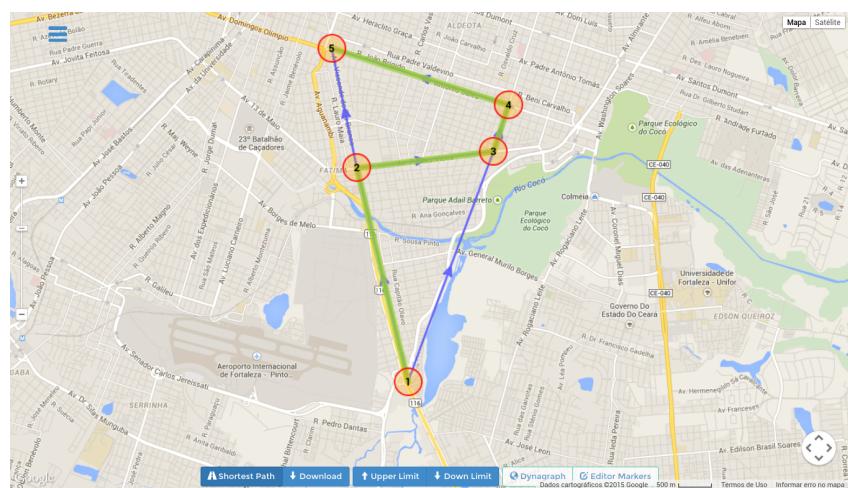


Figura 62: Simulador de Caminho Mínimo - Exemplo 4

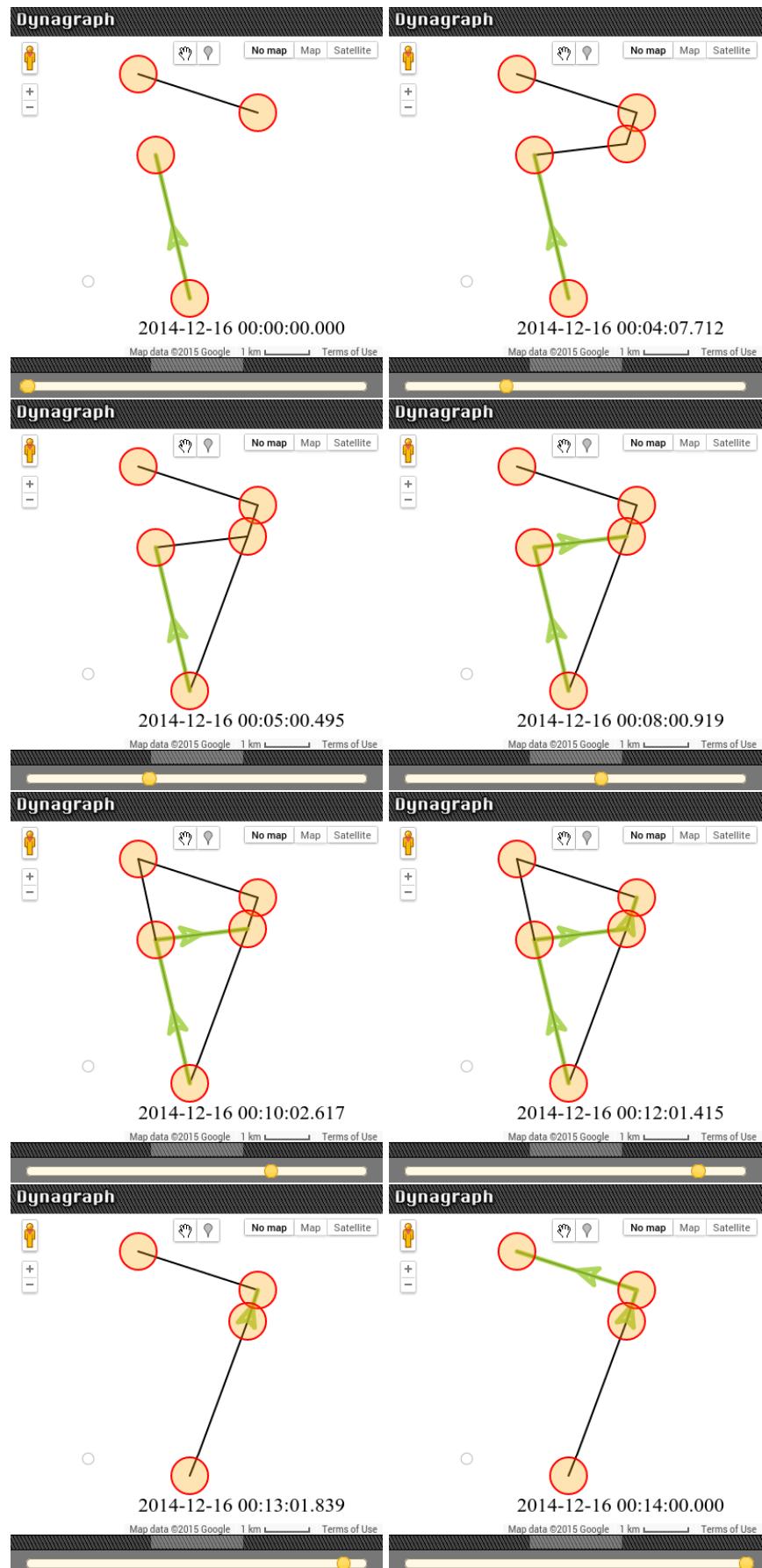


Figura 63: Caminho Mínimo no Dynagraph - Exemplo 4

```

1 "data": {
2   "vertex": {
3     "elements": [
4       "1": {"ini": "2014-12-16 00:00:00", "fim": null },
5       "2": {"ini": "2014-12-16 00:00:00", "fim": "2014-12-16 00:13:00"}, 
6       "3": {"ini": "2014-12-16 00:04:00", "fim": null },
7       "4": {"ini": "2014-12-16 00:00:00", "fim": null },
8       "5": {"ini": "2014-12-16 00:00:00", "fim": null }
9     }
10   },
11   "edge": {
12     "elements": [
13       "1": {"ini": "2014-12-16 00:00:00", "fim": null },
14       "2": {"ini": "2014-12-16 00:05:00", "fim": null },
15       "3": {"ini": "2014-12-16 00:00:00", "fim": null },
16       "4": {"ini": "2014-12-16 00:10:00", "fim": null },
17       "5": {"ini": "2014-12-16 00:00:00", "fim": null },
18       "6": {"ini": "2014-12-16 00:00:00", "fim": null }
19     }
20   },
21   "temporalfields": {
22     "origemdestino": {
23       "1": [{"time": "2014-12-16 00:00:00", "data": {"origem": 1, "destino": 2 } } ],
24       "2": [{"time": "2014-12-16 00:00:00", "data": {"origem": 1, "destino": 3 } } ],
25       "3": [{"time": "2014-12-16 00:00:00", "data": {"origem": 2, "destino": 3 } } ],
26       "4": [{"time": "2014-12-16 00:00:00", "data": {"origem": 2, "destino": 5 } } ],
27       "5": [{"time": "2014-12-16 00:00:00", "data": {"origem": 3, "destino": 4 } } ],
28       "6": [{"time": "2014-12-16 00:00:00", "data": {"origem": 4, "destino": 5 } } ]
29     }
30   }
31 }

```

Figura 64: Simulador de Caminho Mínimo: Estrutura JSON - Exemplo 4

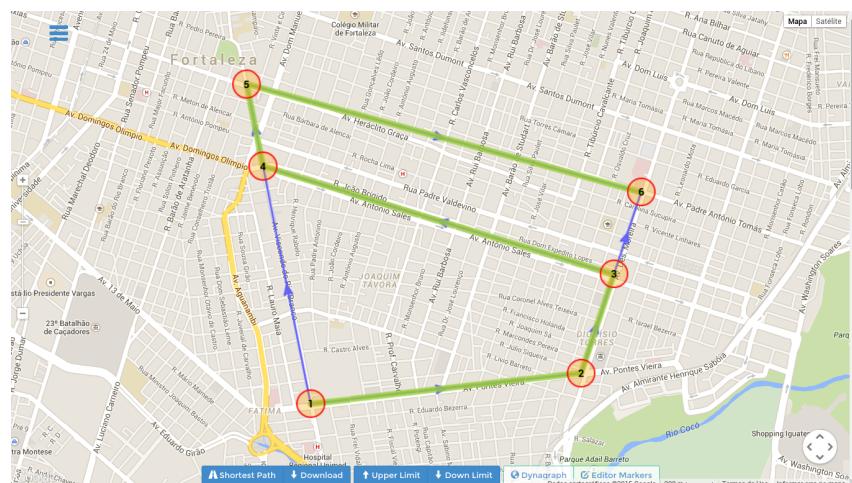


Figura 65: Simulador de Caminho Mínimo - Exemplo 5

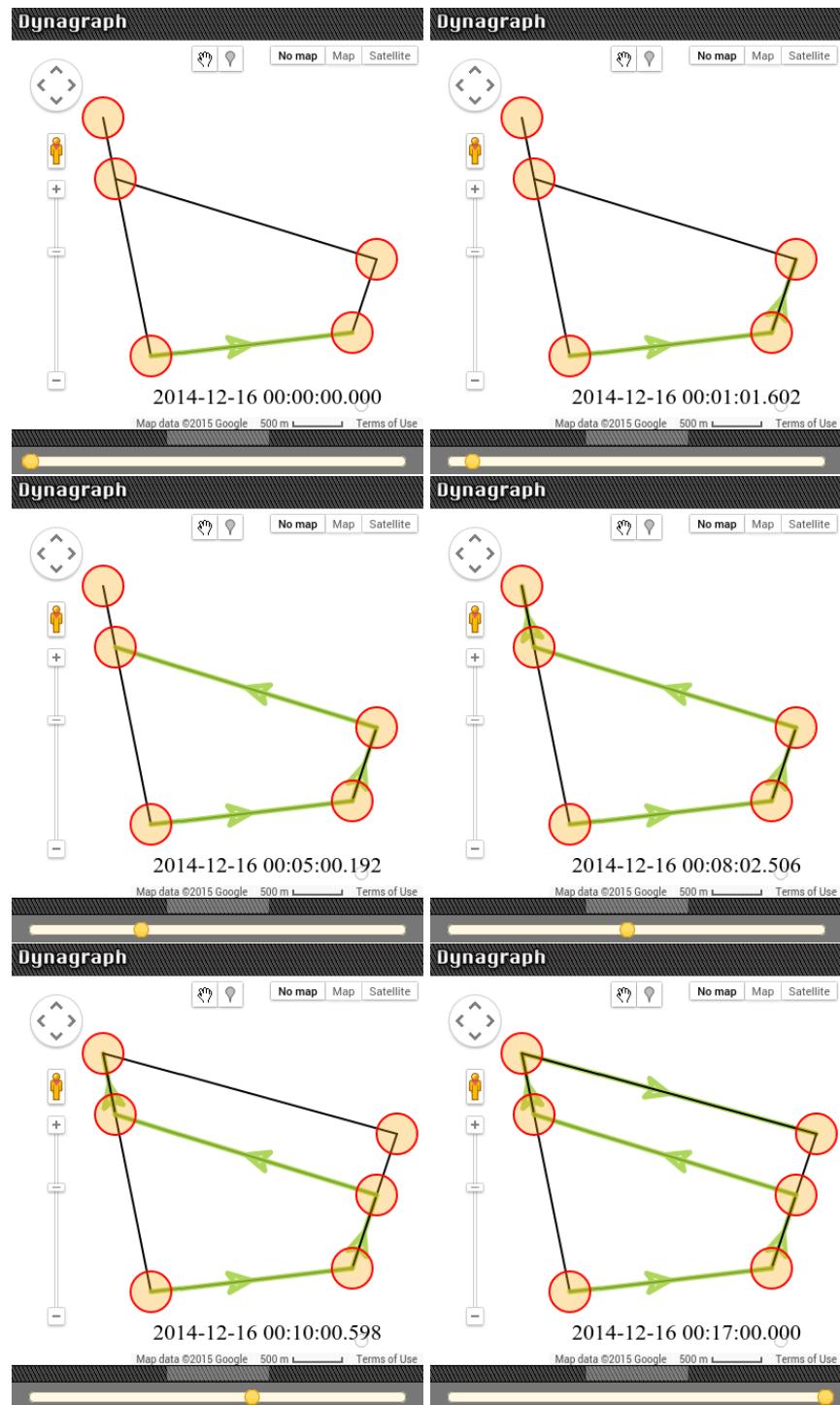


Figura 66: Caminho Mínimo no Dynagraph - Exemplo 5

```

1 "data": {
2   "vertex": {
3     "elements": {
4       "1": {"ini": "2014-12-16 00:00:00", "fim": null },
5       "2": {"ini": "2014-12-16 00:00:00", "fim": null },
6       "3": {"ini": "2014-12-16 00:00:00", "fim": null },
7       "4": {"ini": "2014-12-16 00:00:00", "fim": null },
8       "5": {"ini": "2014-12-16 00:00:00", "fim": null },
9       "6": {"ini": "2014-12-16 00:10:00", "fim": null }
10    }
11  },
12  "edge": {
13    "elements": {
14      "1": {"ini": "2014-12-16 00:00:00", "fim": null },
15      "2": {"ini": "2014-12-16 00:00:00", "fim": null },
16      "3": {"ini": "2014-12-16 00:00:00", "fim": null },
17      "4": {"ini": "2014-12-16 00:00:00", "fim": null },
18      "5": {"ini": "2014-12-16 00:00:00", "fim": null },
19      "6": {"ini": "2014-12-16 00:00:00", "fim": null },
20      "7": {"ini": "2014-12-16 00:00:00", "fim": null }
21    },
22    "temporalfields": {
23      "origemdestino": {
24        "1": [{"time": "2014-12-16 00:00:00", "data": {"origem": 1, "destino": 2 } } ],
25        "2": [{"time": "2014-12-16 00:00:00", "data": {"origem": 1, "destino": 4 } } ],
26        "3": [{"time": "2014-12-16 00:00:00", "data": {"origem": 2, "destino": 3 } } ],
27        "4": [{"time": "2014-12-16 00:00:00", "data": {"origem": 3, "destino": 4 } } ],
28        "5": [{"time": "2014-12-16 00:00:00", "data": {"origem": 3, "destino": 6 } } ],
29        "6": [{"time": "2014-12-16 00:00:00", "data": {"origem": 4, "destino": 5 } } ],
30        "7": [{"time": "2014-12-16 00:00:00", "data": {"origem": 5, "destino": 6 } } ]
31      }
32    }
33  }
34 }
35
36 //vetor de custo as 0h
37 {
38   "1": [[1,5,7,4,3,2 ],...],
39   "2": [[9,8,6,4,3,5 ],...],
40   "3": [[4,2,3,4,3,2 ],...],
41   "4": [[3,7,7,7,5,8 ],...],
42   "5": [[3,2,3,3,2,7 ],...],
43   "6": [[8,9,10,3,4,2 ],...],
44   "7": [[8,9,7,3,5,2 ],...]
45 }

```

Figura 67: Simulador de Caminho Mínimo: Estrutura JSON - Exemplo 5

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 construída uma ferramenta (*software*) extendida do Dynagraph, que calcula e exibe o caminho mínimo entre dois vértices conhecidos.

Inicialmente foram apresentados os algoritmos de topologia estática e atributos dinâmicos com limite inferior e superior. Em seguida, esses mesmos algoritmos foram utilizados 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.

Esta pesquisa também contribuiu com a integração da ferramenta implementada com a ferramenta Dynagraph. Essa integração permite através do Dynagraph solicitar o caminho mínimo do grafo em análise. Para isso, ele faz uma chamada ao *software* desenvolvido. Este, por sua vez, analisa o grafo enviado, calcula o caminho mínimo e envia para o Dynagraph o mesmo grafo acrescido das arestas do caminho mínimo gerado.

Para trabalho futuros, pretende-se provar que o algoritmo de caminho mínimo para grafos dinâmicos implementado garante que o intervalo de previsão é mínimo. Também pretende-se otimizar o algoritmo para garantir o cálculo eficiente, pois a pesquisa somente se concentrou na elaboração do modelo computacional. Para trabalhos futuros na ferramenta, pretende-se resolver alguns detalhes para garantir o funcionamento da ferramenta. A aplicação ainda limita-se ao selecionar 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. E por fim, aperfeiçoar o Editor de Características extendendo a edição para arestas e integrar ao Dynagraph.

BIBLIOGRAFIA

- AHUJA, R.K.; MAGNANTI, Thomas L.; ORLIN, James B. *Network flows: theory, algorithms, and applications*. [S.I.]: 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.I.]: Rio de Janeiro, Brasil, 2007. p. 1–12.
- BASTIAN, Mathieu; HEYMANN, Sébastien; JACOMY, Mathieu. Gephi: An Open Source Software for Exploring and Manipulating Networks. In: *Internacional AAAI Conference on Weblogs and Social Media*. [S.I.]: 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.I.]: Cambridge and New York: The MIT Press and McGraw-Hill Book Company, 2nd. Edition, 2001.
- CROCKFORD, Douglas. *JavaScript: The Good Parts*. [S.I.]: (O'Reilly Media), 2008.
- GOLDBARG, Marco; GOLDBARG, Elizabeth. *Grafos: Conceitos, algoritmos e aplicações*. [S.I.]: 1a Edição, Editora Elsevier, 2012.
- GOODRICH, Michael T.; TAMASSIA, Roberto. *Estrutura de Dados e Algoritmos em JAVA*. [S.I.]: (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.