



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA CIVIL

VICTOR HAMANN PEREIRA

**DESENVOLVIMENTO DE MÉTODO COMPUTACIONAL PARA DEFINIÇÃO DE ROTAS BASEADAS
NO USO DO SISTEMA DE TRANSPORTE COLETIVO**

6 de dezembro de 2017

VICTOR HAMANN PEREIRA

**DESENVOLVIMENTO DE MÉTODO COMPUTACIONAL PARA DEFINIÇÃO DE ROTAS BASEADAS
NO USO DO SISTEMA DE TRANSPORTE COLETIVO**

Trabalho de Conclusão de Curso apresentado pelo acadêmico Victor Hamann Pereira à banca examinadora do Curso de Graduação de Engenharia Civil da Universidade Federal de Santa Catarina como requisito parcial para obtenção do título de Engenheiro Civil.

Professor orientador: Alexandre Hering Coelho, Dr.

6 de dezembro de 2017

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Pereira, Victor Hamann Pereira

Desenvolvimento de método computacional para definição de rotas baseadas no uso do sistema de transporte coletivo / Victor Hamann Pereira Pereira ; orientador, Alexandre Hering Coelho Coelho, 2017.

60 p.

Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, Centro Tecnológico,
Graduação em Engenharia Civil, Florianópolis, 2017.

Inclui referências.

1. Engenharia Civil. 2. Área de transportes. 3. Transporte coletivo. 4. Sistemas de informações geográficas. 5. Roteamento. I. Coelho, Alexandre Hering Coelho. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Civil. III. Título.

Victor Hamann Pereira

**DESENVOLVIMENTO DE MÉTODO COMPUTACIONAL PARA DEFINIÇÃO DE ROTAS BASEADAS
NO USO DO SISTEMA DE TRANSPORTE COLETIVO**

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do grau de Engenharia Civil e aprovado em sua forma final pelo Departamento de Engenharia Civil da Universidade Federal de Santa Catarina.

Florianópolis, 4 de dezembro de 2017.

Prof. Luciana Rohde, Dr.
Coordenador do curso

BANCA EXAMINADORA:



Prof. Alexandre Hering Coelho, Dr.
Orientador

Prof. Luciana Rohde, Dr.
Membro

Edésio Elias Lopes, Dr.
Membro

「必死に生きてこそ、その生涯は光を放つ」
織田信長

AGRADECIMENTOS

Agradeço aos meus pais pelo apoio e incentivo que me deram, e pelas lições que me ensinaram.

Ao professor e orientador Alexandre Hering Coelho pela dedicação tanto na sala de aula quanto na orientação deste trabalho, e sobretudo pela paciência.

Aos amigos e colegas do curso pela jornada compartilhada ao longo da graduação.

A todos que de alguma forma contribuíram para a realização deste trabalho.

RESUMO

O planejamento de rotas é um passo importante do uso do sistema de transporte coletivo, e uma apresentação adequada das informações de itinerários de ônibus e o uso de *trip planners* automatizados podem facilitar o uso do sistema pelos usuários. O presente trabalho busca o desenvolvimento e a implementação computacional de um método que, a partir de coordenadas geográficas de partida e chegada, e data e horário de partida, define rotas baseadas no sistema de transporte coletivo. Para isso foram coletados dados da região central de Florianópolis, consistindo da malha viária, pontos e itinerários das linhas de ônibus. Os dados coletados foram então processados de forma a possibilitar a análise de rotas. O método é implementado através do uso de sistemas de informações geográficas e banco de dados espacial, com o auxílio de programação de *scripts*. As rotas são definidas com base na rota de menor distância entre os pontos de partida e chegada. Como resultado são retornados rotas de ônibus ordenadas pela estimativa de tempo de percurso, acompanhadas de instruções detalhadas de como segui-las. Apesar de serem produzidos resultados em parte positivos, as limitações presentes no método o distanciam de uma aplicação funcional.

Palavras-chave: ônibus urbano, roteamento, sistemas de informações geográficas, banco de dados espacial.

ABSTRACT

Trip planning is an important step in the usage of the public transportation system, and an adequate presentation of itinerary information, as well as the usage of automated trip planners can help users in the use of the system. This paper seeks to develop and implement a computational method which, based on coordinates of origin and destination points, along with the departure date and time, defines transit routes. For this purpose, data from the central region of Florianópolis were collected, consisting of street network, bus stops and bus transit itineraries. The data were then processed in order to enable route analysis. The method is implemented through the use of geographic information systems and spatial database, with the aid of scripting. The routes are defined based on the shortest path between origin and destination. The results are bus routes sorted by estimated time taken, accompanied by detailed instructions on how to follow them. Despite partially positive results, the limitations of the method distance it from having practical use.

Keywords: urban bus, routing, geographic information systems, spatial database.

Lista de Siglas

ArcIMS *Arc Internet Map Server*

DARP *Dial-a-Ride Problem*

ESRI *Environmental Systems Research Institute*

GIS *Geographic Information System*

GRASS *Geographic Resources Analysis Support System*

MTC *Montreal Transit Commission*

OSM *OpenStreetMap*

PSQL *PostgreSQL*

SGBD *Sistema Gerenciador de Banco de Dados*

SIG *Sistema de Informações Geográficas*

SQL *Structured Query Language*

Lista de Figuras

1	Relação topológica entre polígonos	8
2	Função <i>snap</i>	12
3	Função <i>break</i>	12
4	Método	15
5	Preparação dos dados	16
6	Interface do <i>OpenStreetMap</i> (OSM)	17
7	Tabela de atributos da camada de linhas do OSM, extraída do QGIS	19
8	Tabela de atributos da camada de multilinhas	19
9	Tabela de atributos da camada de pontos	19
10	Horários de saída de ônibus retirados do <i>site</i> do Consórcio Fênix	20
11	Resultado da filtragem	21
12	Interseção entre <i>links</i> da malha	22
13	Procedimento para definição de rotas	25
14	Definição de uma rota	29
15	Divergência entre rotas	30
16	Estimativa do tempo levado	31
17	Exemplo de saída do método	34
18	Rota mostrada no mapa	35
19	Exemplo de saída com duas linhas de ônibus	35

Lista de Tabelas

1	Filtragem dos dados geométricos	18
2	Listas de pontos de ônibus por <i>link</i>	27
3	Trechos da rota	33
4	Trechos da rota agrupados	33

Sumário

1	Introdução	1
1.1	Justificativa	2
1.2	Objetivos	2
1.2.1	Objetivo geral	2
1.2.2	Objetivos específicos	2
1.3	Limitações do trabalho	3
2	Revisão bibliográfica	4
2.1	Trabalhos relacionados	4
2.2	Sistemas de informações geográficas	5
2.2.1	Topologia	7
2.3	Banco de dados	8
2.3.1	Sistema gerenciador de banco de dados	9
2.3.2	SQL	9
2.3.3	Banco de dados espacial	10
2.4	PostgreSQL	10
2.4.1	PostGIS	11
2.4.2	pgRouting	11
2.5	QGIS e GRASS GIS	11
2.6	Python	13
3	Método	14
3.1	Preparação dos dados	14
3.1.1	Criação do banco de dados espacial	14
3.1.2	Estudo de fontes de dados	17
3.1.3	Coleta e filtragem dos dados geométricos	18
3.1.4	Tratamento topológico da malha viária	21
3.1.5	Coleta dos horários de saída dos ônibus	23
3.2	Desenvolvimento do procedimento para definição das rotas	23
3.2.1	Dados de entrada	24
3.2.2	Rota de menor distância	24
3.2.3	Definição das rotas de ônibus	26
3.2.4	Estimativa de tempo levado para cada rota	30
3.2.5	Dados de saída	33

4 Resultados	34
5 Conclusões	36
5.1 Recomendações para trabalhos futuros	36
Referências	38
APÊNDICE A - Comandos SQL	40
APÊNDICE B - <i>Script</i> de inserção dos horários de ônibus	41
APÊNDICE C - <i>Script</i> de roteamento	42

1 Introdução

Mobilidade urbana é um tema que afeta a todos. Filas e congestionamentos levam ao aumento do tempo que as pessoas passam se locomovendo, resultando em uma queda da produtividade e da qualidade de vida da população. Através do estudo de melhores formas de planejamento e operação dos sistemas de transporte, seja na melhoria da infraestrutura viária ou do sistema de transporte coletivo, a área de transportes pode oferecer soluções para os problemas relacionados à mobilidade urbana.

Segundo CNT (2017), no Brasil, o principal meio utilizado para o transporte de passageiros é o rodoviário, principalmente devido à carência de oferta de outras infraestruturas de transporte.

Um modo muito comum no Brasil para o transporte coletivo é o ônibus. Uma parte importante do serviço de ônibus é a disponibilização de informações aos usuários, de forma que estas possam planejar suas viagens previamente.

O uso de sistemas de informações geográficas (SIG) pode ser muito útil para o desenvolvimento de sistemas que auxiliem no planejamento das viagens. Estes podem ser chamados de *trip planners*. Trépanier et al. (2005) definem *trip planner* como ferramentas web que criam itinerários baseados em transporte coletivo a partir de um par origem-destino informado pelo usuário.

Segundo Cherry et al. (2006), nos Estados Unidos, agências de transporte coletivo têm disponibilizado informações na internet, usando mapas, itinerários e *trip planners* automatizados. No Brasil, porém, é comum que estas informações sejam disponibilizadas de maneira limitada, mostrando os horários de saída dos ônibus e suas rotas, porém sem indicar, por exemplo, os horários em que os ônibus devem chegar em cada ponto de ônibus, e muitas vezes sequer quais linhas de ônibus passam pelos pontos, dificultando o uso do transporte coletivo.

Recentemente foi lançado o aplicativo floripanoponto¹, um *trip planner* que define rotas de ônibus baseado nas linhas de ônibus de Florianópolis, inclusive linhas que acessam o continente. Este aplicativo possui uma interface gráfica de fácil utilização e, além de definir rotas de um ponto a outro da cidade, disponibiliza os itinerários das linhas de ônibus e previsão de chegada nos pontos de ônibus. Este aplicativo é um bom exemplo de como disponibilizar as informações ao usuário, e seria interessante que este tipo de serviço fosse prestado também em outras cidades brasileiras.

Além de facilitar o uso do sistema de transporte coletivo pela população, *trip planners* podem ajudar os próprios provedores do serviço de ônibus a planejar e adaptar o sistema atual. Trépanier et al. (2005) realizaram um estudo baseado em dados do histórico de uso de um *trip planner* do site da *Montreal Transit Commission* (MTC) para determinar se estes dados seriam úteis ao planejamento do transporte coletivo. A conclusão do estudo foi que os dados podem ajudar na identificação de novos locais a serem acessados pelo sistema de transporte coletivo, no melhor entendimento do comportamento dos usuários e na atualização do SIG e do *trip planner* em si.

¹<https://www.floripanoponto.com.br/>

1.1 Justificativa

O sistema de transporte coletivo permite o deslocamento de pessoas sem a necessidade de possuir um automóvel próprio. Cherry et al. (2006) comentam que um dos maiores problemas associados ao uso de transporte coletivo é a apresentação da informação.

Uma melhor apresentação da informação relacionada ao transporte coletivo poderia incentivar seu uso. Segundo Silva (2000), sistemas de informações aos usuários garantem um aumento na qualidade do serviço ofertado aos passageiros.

Segundo International Energy Agency (2002) apud Lacerda (2006), veículos de passeio com capacidade para cinco passageiros correspondem a 62% do espaço ocupado por um ônibus urbano com capacidade para quarenta passageiros. Considerando isso, um maior uso do sistema de transporte coletivo poderia reduzir os problemas de congestionamento da malha viária. Além de reduzir o tráfego, os ônibus proporcionam grande mobilidade para a população:

"Os ônibus têm papel fundamental na mobilidade urbana, visto que são responsáveis por alimentar esses sistemas, pois são meios de transporte com maior capilaridade e que podem chegar a áreas mais distantes – e até mesmo mais isoladas – dos municípios, permitindo, assim, maior acessibilidade para a população." (CNT, 2017, p. 12)

Um método capaz de determinar as melhores linhas de ônibus a serem utilizadas para se locomover de um ponto a outro facilitaria o uso deste sistema, sendo muito útil para a sociedade.

1.2 Objetivos

1.2.1 Objetivo geral

Desenvolver e implementar, com recursos computacionais, um método que determine rotas de ônibus urbano com base em dados e recursos de *software* abertos.

1.2.2 Objetivos específicos

- Identificar e estudar ferramentas disponíveis para fornecimento de informações relativas a rotas de ônibus urbano;
- Estudar e aplicar técnicas de modelagem de banco de dados espacial;
- Coletar dados necessários para constituir um banco de dados capaz de dar suporte a um sistema de informações sobre rotas de ônibus urbano;
- Aplicar funcionalidades de sistemas de informações geográficas para processamento de dados espaciais relativos a infraestrutura de transporte;
- Analisar os resultados obtidos pelo método.

1.3 Limitações do trabalho

Este trabalho possui limitações que distanciam o funcionamento do método da realidade.

A malha viária é modelada como uma rede seguindo uma estrutura de grafo. Desta forma, a velocidade do percurso de cada *link* da malha é considerada como constante, e não são considerados os efeitos de curvas.

Em relação aos dados coletados, foi estudada somente a região central de Florianópolis, e só foram coletadas informações relacionadas a parte das linhas de ônibus que compõe o sistema de transporte coletivo da cidade.

O método desenvolvido neste trabalho opera de maneira estática, sem considerar dados de posicionamento de ônibus ou da operação do sistema de transporte em tempo real. Não são considerados, por exemplo, congestionamentos que poderiam reduzir a velocidade do percurso nas rotas de ônibus.

Além disso, as velocidades dos ônibus são consideradas como constantes, baseadas em informações disponibilizadas pelo provedor do serviço de ônibus, não levando em consideração informações sobre horários de pico.

A avaliação das rotas definidas é feita somente em relação à estimativa de tempo levado, sem considerar a distância total percorrida a pé ou custos de passagens.

2 Revisão bibliográfica

Nesta seção são explorados conceitos necessários à realização do trabalho. Primeiramente são apresentados trabalhos que tratam de desenvolvimento de *trip planners* e de roteamento. A análise destes trabalhos remete a conceitos intimamente relacionados: sistema de informações geográficas, uso de banco de dados com funções de relacionamento espacial e linguagem de programação. Então, a revisão aborda também estes conceitos na sua sequência.

2.1 Trabalhos relacionados

Hickman (2002) desenvolveu um método para definir rotas de ônibus para a rede de ônibus Sun Tran em Tucson, Arizona, que utiliza dados históricos da operação do sistema de ônibus para, além de determinar as rotas mais rápidas, calcular a probabilidade da rota levar o tempo estimado. Este método elimina rotas de maior tempo conforme é aplicado, de forma a reduzir a sua demanda por processamento computacional.

Cherry et al. (2006) desenvolveram um protótipo de *trip planner* que define rotas para a Sun Tran baseado em SIG. Este sistema permite ao usuário definir o local de partida e pontos os quais queira visitar clicando no mapa. O desenvolvimento foi feito com o *software Arc Internet Map Server* (ArcIMS) da *Environmental Systems Research Institute* (ESRI), capaz de apresentar informações geográficas através da Internet, e consistiu de três etapas:

- desenvolvimento de um mapa SIG, baseado em arquivos *shapefile*;
- criação de um serviço de mapas, que gera os mapas apresentados ao usuário a partir do original;
- *design* da interface utilizada pelo usuário.

O roteamento foi feito com base no algoritmo de Hickman (2002). Os dados de entrada do protótipo são os locais de partida e chegada, os *time points* em que o usuário gostaria de subir no ônibus (*time points* sendo pontos que têm horários de passagem do ônibus conhecidos) e o horário em que o usuário gostaria de chegar no local. A rota é então informada ao usuário através de um conjunto de instruções textuais de como segui-la. A principal limitação está relacionada à performance do sistema, que precisa gerar milhares de objetos para mostrar o mapa ao usuário.

Nienkotter (2017) desenvolveu um algoritmo de roteirização baseado no algoritmo de Clark & Wright para a análise de rotas de empilhadeiras em armazéns de grande porte. Este algoritmo leva em consideração as operações de *picking*, ou coleta, e de armazenamento de forma conjunta, ou seja, com as empilhadeiras fazendo ambas as operações, em vez de se definir um grupo de empilhadeiras para realizar cada operação.

Foi realizado um estudo de caso em uma empresa de grande porte, comparando a definição das rotas no momento do trabalho com as produzidas pelo modelo proposto. A partir deste estudo, porém, não foi observada economia de mão-de-obra nas soluções geradas pelo modelo.

Hawerorth (2017) realizou um estudo de roteirização no serviço Transporte Eficiente, em Joinville, propondo configurações de rotas baseadas no *Dial-a-Ride Problem* (DARP), que consiste de otimizar rotas para o transporte de um conjunto de pessoas com um ponto de partida e chegada, de forma a atender às suas restrições.

Foi desenvolvido um método baseado em uma heurística de programação que leva em consideração o atraso. O método foi implementado na linguagem de programação C#, e acabou apresentando tempo de execução elevado e gerando soluções não-factíveis.

2.2 Sistemas de informações geográficas

Para trabalhar com características geométricas é útil o estudo de SIG. SIG possibilitam a análise e processamento de dados relacionados geometricamente.

SIG, do inglês *geographic information system*, pode ser definido como um conjunto de ferramentas para coletar, armazenar, transformar e visualizar dados espaciais do mundo real para um conjunto particular de propósitos (BURROUGH, 1986).

Mais do que um conjunto de ferramentas, é importante definir SIG como sistemas computacionais, que permitem a automatização de tarefas de alta complexidade. Um aspecto de SIG a ser analisado é a maneira como os dados são representados.

Segundo Burrough (1986), todo SIG é uma representação computacional de aspectos do mundo real, e que, como seria impossível representar todos os elementos nos quais se tem interesse, SIG apresentam uma visão simplificada do mundo, frequentemente chamada de modelo. Para entender a maneira como SIG representam a realidade, é necessário entender o modelo de dados que utilizam.

Hagget e Chorley (1967) apud Burrough (1986) definem modelo como uma síntese de dados, usada para lidar com um sistema cuja escala ou complexidade seja demasiado elevada.

O modelo utilizado por SIG é baseado em elementos, ou tipos de dado, básicos, que quando agrupados e relacionados, representam o espaço, objetos contidos no espaço e as relações entre estes objetos.

Os elementos básicos utilizados por SIG são três: (MONMONIER, 1991):

- pontos;
- linhas;
- áreas.

Como representações da realidade, sob uma ótica geográfica, SIG estão intimamente relacionados com mapas, que em grande parte utilizam os mesmos elementos para a representar o mundo. Mapas rodoviários, por exemplo, utilizam uma combinação dos três elementos básicos: pontos para indicar a localização de marcos e pequenas cidades, linhas para indicar o comprimento e formato de rios e rodovias, e áreas para representar o tamanho e formato de cidades grandes.

Cada elemento representa objetos com foco em características diferentes destes. Segundo Güting (1994), um ponto representa um objeto cuja localização é importante, mas sua dimensão não. Linhas representam estruturas que permitam o deslocamento no espaço, ou conexões entre objetos no espaço. A área, em contraste ao ponto, representa objetos em que suas dimensão e formato são relevantes, como países.

O elemento utilizado para representar elementos do mundo real depende, entre outros fatores, da escala em que se pretende trabalhar. Segundo Monmonier (1991), a maioria dos mapas são menores do que a realidade que representam, e a escala indica o quão menor o mapa é.

Os mapas rodoviários do exemplo anterior trabalham com escalas relativamente grandes. Um mapa de uma cidade, por exemplo, teria uma escala menor. Neste contexto, o elemento representado por cada tipo de dado muda. A cidade em si passa a ser uma área, pois nesta escala sua dimensão é relevante, enquanto as linhas podem representar a malha viária e pontos a localização de pontos turísticos, entre outros.

A partir do agrupamento e relacionamento de elementos básicos, é possível definir elementos mais complexos para acomodar diferentes fenômenos do mundo real. Segundo Heywood et al. (2006), além dos três tipos básicos (ponto, linha e área), existem outros dois tipos de dado: redes e superfícies. Uma rede é um conjunto de linhas conectadas, podendo representar a malha viária e redes hidrográficas, por exemplo. A superfície é uma área tridimensional. Uma superfície pode ser usada para representar topografia ou variáveis não-topográficas como densidade demográfica e nível de poluição.

Com estes cinco tipos de dado, é possível representar a realidade a um alto nível de complexidade sem sobrecarregar o sistema com detalhes. É importante, porém, entender como exatamente esta representação é feita por um computador.

O olho humano é capaz de identificar formas eficientemente, mas um computador necessita de instruções exatas sobre como padrões espaciais devem ser manipulados e apresentados (BURROUGH, 1986 apud HEYWOOD et al., 2006).

A maneira como isto é feito afeta SIG em vários níveis, desde o armazenamento ao processamento e análise dos dados, sendo de suma importância entender as implicações de se utilizar cada abordagem. Heywood et al. (2006) afirma que existem duas principais formas de representação de entidades espaciais por computadores. São as abordagens vetorial e *raster*.

Os mesmos autores explicam que o modelo de dados *raster* utiliza células individuais como blo-

cos para construção de imagens que representam pontos, linhas, áreas, redes e superfícies. Neste modelo, o formato e tamanho das entidades espaciais é definido a partir do agrupamento das células. O tamanho da célula é importante, pois influencia como as entidades serão mostradas.

Imagens obtidas por satélite, por exemplo, podem ser vistas como um modelo *raster* da realidade, em que cada *pixel* da imagem corresponde a uma área, representada pela cor captada pelo satélite. A partir da análise destes *pixels* é possível mapear objetos do mundo real.

A abordagem vetorial trabalha os tipos de dado de maneira diferente. Segundo Heywood et al. (2006), o modelo vetorial de dados utiliza coordenadas cartesianas (x,y) para armazenar a forma de uma entidade espacial. Neste modelo, em contrapartida ao *raster*, o ponto é o bloco de construção mais básico a partir do qual os outros tipos de dado são definidos. Linhas e áreas são construídas conectando-se séries de pontos. Uma linha é um polígono aberto, enquanto uma área é um polígono fechado. Quanto mais complexo o formato de uma entidade, mais pontos são necessários à sua representação.

2.2.1 Topologia

A relação entre elementos representados espacialmente pode ser descrita com o uso da topologia. A partir das relações topológicas entre objetos é possível realizar diversas análises dos sistemas que estes compõem. A topologia é também uma ótima ferramenta para a manutenção e garantia da integridade de dados geométricos.

Segundo Obe e Hsu (2015), a topologia define as regras de inter-relação entre geometrias. Por exemplo, considerando polígonos que representem áreas vizinhas, a topologia descreve quais vértices destes polígonos são compartilhados. Se as áreas reais por algum motivo forem modificadas, o modelo deve ser modificado da mesma maneira. A relação topológica entre os polígonos garante a integridade dos dados quando as alterações forem feitas, sem sobreposição ou espaços vazios entre os elementos.

Na Figura 1 estão representadas três áreas vizinhas que sofreram modificações, mostrando as mudanças correspondentes nas suas representações. Após a alteração, os polígonos que representam as áreas continuam adjacentes uns aos outros, e as linhas que os delimitam continuam conectadas corretamente.

A topologia é particularmente útil para representar redes. Neste caso, a relação topológica entre as linhas e nós define quais linhas estão conectadas a quais nós. Neste sentido, a estrutura de uma rede de *links* conectados por nós pode ser descrita utilizando um grafo:

"Grafo é um modelo matemático que representa relações entre objetos. Um grafo é um conjunto dado por $G = (V, E)$, onde V é um conjunto finito de pontos, normalmente denominados de nós ou vértices e E é uma relação entre vértices, ou seja, um conjunto de pares em $V \times V$." (LINDEN, 2009)

Assim, cada nó é um vértice do grafo, enquanto cada *link* é um par ordenado de vértices, ou seja,

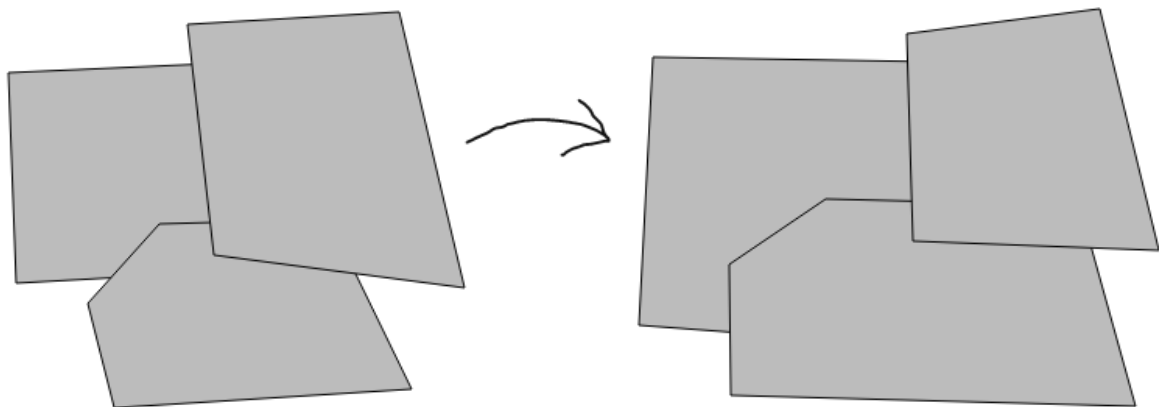


Figura 1: Relação topológica entre polígonos

uma aresta do grafo. A partir da estrutura de grafo, é possível fazer análises ligadas a roteamento, ou seja, como uma rede pode ser percorrida.

Para se representar o fluxo de uma rede, é fundamental que seja possível representar o sentido do fluxo. Segundo Linden (2009), um grafo pode ser direcionado ou não. No caso direcionado, as arestas somente podem ser percorridas no sentido início-fim, enquanto no caso não-direcionado isso não é necessário.

Em determinada malha viária, por exemplo, podem existir na mesma rede ruas de um sentido e de dois. Portanto, para que o sentido das ruas seja considerado, pode-se utilizar um grafo direcionado em que haja duas arestas para as ruas de dois sentidos, e somente uma para as de um sentido.

2.3 Banco de dados

Foram abordados até aqui assuntos relacionados à representação de dados geométricos. É importante entender também como estes dados são armazenados e gerenciados pelo computador.

Uma ferramenta importante a ser estudada é o banco de dados. Segundo Ramez e Navathe (2005), um banco de dados é uma coleção de dados com as seguintes propriedades:

- os dados armazenados representam aspectos do mundo real;
- o banco de dados é uma coleção lógica e coerente de dados;
- o banco de dados é projetado, construído e povoado por dados de forma a atender a um propósito específico.

O banco de dados é muito útil para o gerenciamento de dados. Um modelo de banco de dados muito utilizado é o modelo relacional. Um banco de dados relacional é um banco de dados no qual os dados armazenados são visualizados por meio de relações, ou seja, conjuntos de informações associadas (DATE, 2003).

Um banco de dados relacional, segundo Sumathi e Esakkirajan (2007), utiliza uma coleção de tabelas para representar tanto dados quanto a relação entre estes dados. Cada linha da tabela representa um elemento ou relação entre elementos, enquanto as colunas contêm características destes elementos.

Codd (1990) ressalta que os dados contidos no banco de dados devem ser identificados, de forma a garantir a integridade dos dados. Geralmente é reservada uma coluna para a identificação do elemento, para que seja possível rastreá-lo ao se realizar consultas.

Por exemplo, as arestas de um grafo podem ser representado minimamente pelos seus nós de início e fim. Logo a tabela que contiver estas arestas possuirá, no mínimo, as três seguintes colunas:

- identificação;
- nó de início;
- nó de fim.

Cada linha da tabela representa uma aresta com as características correspondentes aos valores em cada coluna.

2.3.1 Sistema gerenciador de banco de dados

O banco de dados armazena dados, mas para acessar estes dados e processá-los, ou realizar consultas mais complexas aos dados, relacionando várias tabelas, são utilizados sistemas gerenciadores de banco de dados (SGBD).

Ramez e Navathe (2005) definem um SGBD como um *software* que facilita a definição, construção, manipulação e compartilhamento de banco de dados. Segundo Sumathi e Esakkirajan (2007), um SGBD é capaz de manipular e retirar dados armazenados em um banco de dados relacional.

2.3.2 SQL

A comunicação com um SGBD é feita através de uma linguagem de consulta, com a qual são dadas ao SGBD as instruções de como trabalhar com os dados armazenados.

Segundo Codd (1990), uma linguagem de consulta deve possuir quatro comandos básicos:

- *retrieve* (coletar);
- *insert* (inserir);

- *update* (atualizar);
- *delete* (deletar).

Estas operações são a base para se manipular os dados armazenados em um banco de dados.

A linguagem padrão utilizada na comunicação com um SGBD é o *Structured Query Language* (SQL) (SUMATHI; ESAKKIRAJAN, 2007).

O SQL implementa estas operações através dos comandos "*select*", "*insert*", "*update*" e "*delete*".

2.3.3 Banco de dados espacial

Para que possam ser armazenados dados geométricos em bancos de dados, é necessária a implementação de dispositivos específicos para este propósito. Um banco de dados capaz de armazenar estes dados é denominado de banco de dados espacial.

Segundo Obe e Hsu (2015), um banco de dados espacial possui tipos de dado especificamente desenhados para armazenar objetos no espaço. Considerando isso, um banco de dados espacial pode ser parte de um SIG, especificamente a parte de armazenagem das camadas de dados.

Uma característica importante de bancos de dados espaciais é a capacidade de se indexar os dados espaciais. Güting (1994) explica que o principal propósito da indexação espacial é permitir a seleção de dados com base em relações geométricas. A indexação organiza o espaço e os objetos de forma que somente parte do espaço e um subconjunto dos objetos precisem ser considerados para realizar esta seleção. Isto acelera a manipulação e processamento de dados geométricos, viabilizando a realização de tarefas complexas que envolvam séries de consultas ao banco de dados espacial.

As próximas seções da revisão são dedicadas a abordar algumas ferramentas capazes de implementar os conceitos de SIG e banco de dados espacial vistos até agora.

2.4 PostgreSQL

PostgreSQL (PSQL) é uma ferramenta capaz de gerenciar bancos de dados:

"O PostgreSQL é um SGBD relacional, utilizado para armazenar informações de soluções de informática em todas as áreas de negócios existentes, bem como administrar o acesso a essas informações." (MILANI, 2008, página 25)

O PSQL é *open source*, sendo uma ferramenta muito acessível para o desenvolvimento de sistemas baseados em banco de dados.

Existem diversas extensões que podem ser instaladas em bancos de dados PSQL.

2.4.1 PostGIS

Para permitir o armazenamento de dados geométricos em bancos de dados PSQl, é utilizada e extensão PostGIS.

A extensão PostGIS define uma série de funções para o processamento de dados geométricos. O PostGIS permite a realização de análises de caráter geométrico de dados, como cálculo de distância entre elementos, se estes se interceptam ou estão contidos uns nos outros.

2.4.2 pgRouting

Para adicionar ao PSQl funções de roteamento e de análise topológica de redes, é instalada a extensão pgRouting (PGROUTING TEAM, 2017). Para que seja instalado o pgRouting, é necessário a instalação prévia do PostGIS.

Entre as funções implementadas pelo pgRouting, destaca-se a função `pgr_dijkstra`. Esta função encontra a rota de menor custo entre dois nós de um grafo (PGROUTING TEAM, 2017), podendo este ser direcionado ou não. O custo é um valor associado às arestas, normalmente *links* de uma malha viária, do grafo e pode ser arbitrado ou calculado a partir de características de cada *link*, podendo ser usado, por exemplo, o comprimento do *link* como custo. Neste caso o caminho encontrado é o de menor distância.

Os dados de entrada para esta função são uma rede de *links* conectadas por nós, o valor do custo para se percorrer o *link*, e os nós de início e fim. No caso não-direcionado, é considerado o mesmo custo em ambos os sentidos, enquanto no caso direcionado deve ser informado o custo para cada sentido. Valores negativos indicam que não é possível percorrer o *link* neste sentido.

2.5 QGIS e GRASS GIS

Duas ferramentas muito úteis capazes de implementar funcionalidades SIG são o QGIS e o *Geographic Resources Analysis Support System* (GRASS). Estas têm um alto nível de integração, sendo possível utilizar funções do GRASS em dados carregados no QGIS.

Segundo Obe e Hsu (2015), QGIS é uma ferramenta para visualização, edição e análise de SIG. O QGIS possui um alto nível de integração com o PostGIS, permitindo a conexão direta entre as camadas de dados armazenadas no banco de dados espacial.

Segundo Neteler e Mitasova (2007), GRASS é um SIG que trabalha com camadas de dados *raster* e vetoriais, com processamento integrado de imagem e visualização de subsistemas de dados.

O GRASS possui uma série de funções para o processamento computacional de dados, que são agrupadas em módulos. Em especial pode-se citar as funções para o tratamento topológico de camadas de dados vetoriais. Para isso é usado o módulo *v.clean* (GRASS DEVELOPMENT TEAM, 2017). As funções deste módulo realizam pequenas correções para garantir a integridade do modelo.

A função *snap* está ilustrada na Figura 2. Esta função conecta vértices a outro vértice que esteja mais próximo do que um valor de tolerância dado como entrada (GRASS DEVELOPMENT TEAM, 2017). Desta forma, *links* que tenham seu nó de início ou fim muito próximo a outro nó têm seu tamanho e formato ajustados. Isto garante que uma rota gerada possa passar por esses *links*. Esta função nem sempre deve ser aplicada. Se a conexão não existir nas ruas reais sendo representadas pelos *links*, estes não devem ser conectados um ao outro, pois isso seria uma representação incorreta da malha.

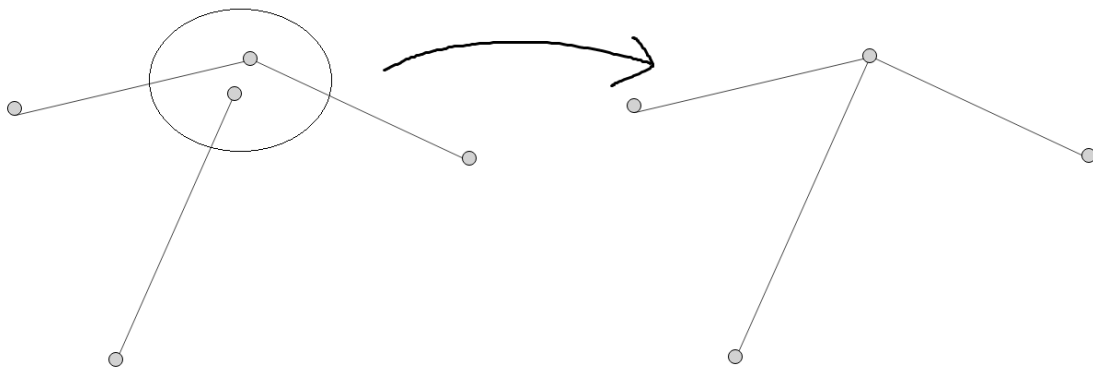


Figura 2: Função *snap*

A função *break*, ilustrada na Figura 3, quebra linhas nas interseções com outras linhas (GRASS DEVELOPMENT TEAM, 2017). No caso mostrado na Figura 3 os dois *links* interceptantes são quebrados na sua interseção, resultando em quatro *links*, todos conectados pelo nó central. Assim como a função *snap*, esta nem sempre deve ser aplicada. Em alguns casos, como viadutos e túneis, não existe uma conexão real entre as ruas representadas pelos *links*. Nestes casos não se deve utilizar esta função.

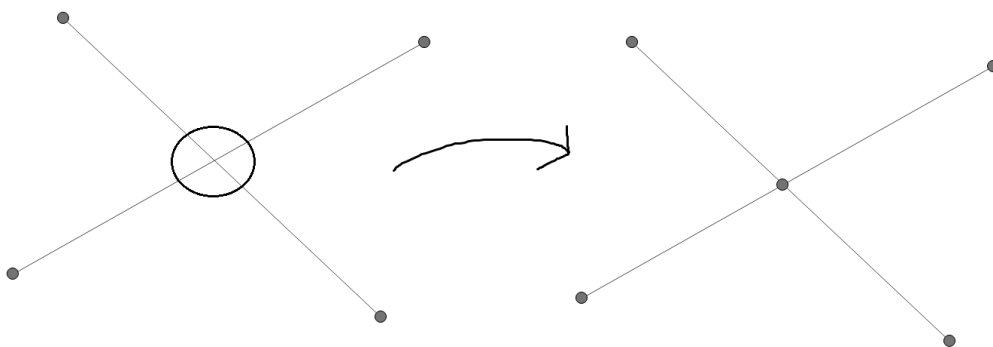


Figura 3: Função *break*

2.6 Python

A partir da programação de *scripts* integrados às ferramentas abordadas anteriormente, é possível utilizar várias funções de diferentes *software* de forma sequencial, utilizando os resultados de uma função como parâmetro de outra. Isto aumenta o nível de complexidade das tarefas que podem ser realizadas, e permite a automatização de processos.

Uma linguagem de programação capaz de aplicar este conceito muito bem é o Python, por possuir várias bibliotecas capazes de integrar programação a sistemas de banco de dados. Segundo Lutz (2011), há interfaces que permitem a comunicação entre *scripts* em Python e bancos de dados relacionais como MySQL, PostgreSQL e Oracle. Programando em Python, é possível desenvolver aplicações complexas rapidamente, utilizando as bibliotecas já disponíveis.

3 Método

Para se definir as rotas de ônibus, são necessárias informações sobre a infraestrutura de transporte coletivo, consistindo de quatro elementos:

- malha viária;
- itinerários das linhas de ônibus;
- localização dos pontos de ônibus;
- horários de saída dos ônibus.

A partir destes dados, é possível o desenvolvimento de um procedimento que defina as rotas. Este procedimento deve acessar os dados coletados e extrair informações geométricas e não-geométricas, e então retornar os itinerários de cada rota, distâncias percorridas e tempo levado.

Neste sentido é útil o estudo de formas de coleta, armazenamento e processamento destes dados, e de análise de redes de transporte. Tendo isso em mente, o método é dividido em duas partes, a primeira sendo a preparação dos dados (Seção 3.1), e a segunda o desenvolvimento do procedimento para definição das rotas (Seção 3.2). O método do trabalho está ilustrado de forma geral no fluxograma da figura 4.

3.1 Preparação dos dados

A preparação dos dados envolve a coleta, armazenamento e processamento de dados de forma a possibilitar a análise de rotas. O processo de preparação dos dados está ilustrado no fluxograma da Figura 5, e seus passos serão descritos em maior detalhe nas seções 3.1.1 a 3.1.5.

3.1.1 Criação do banco de dados espacial

Com o intuito de armazenar os dados, foi criado o banco de dados. Este foi criado a partir do PostgreSQL. Para torná-lo um banco de dados espacial, foi instalada a extensão PostGIS. O pgRouting foi instalado para possibilitar a análise de roteamento, importante para os passos descritos mais adiante, na seção 3.2. Isto foi feito através de linha de comando, com os comandos² "create database" e "create extension":

```
CREATE DATABASE mydb;  
CREATE EXTENSION postgis;  
CREATE EXTENSION pgrouting;
```

²A lista com os comandos SQL utilizados neste trabalho encontra-se no Apêndice A.

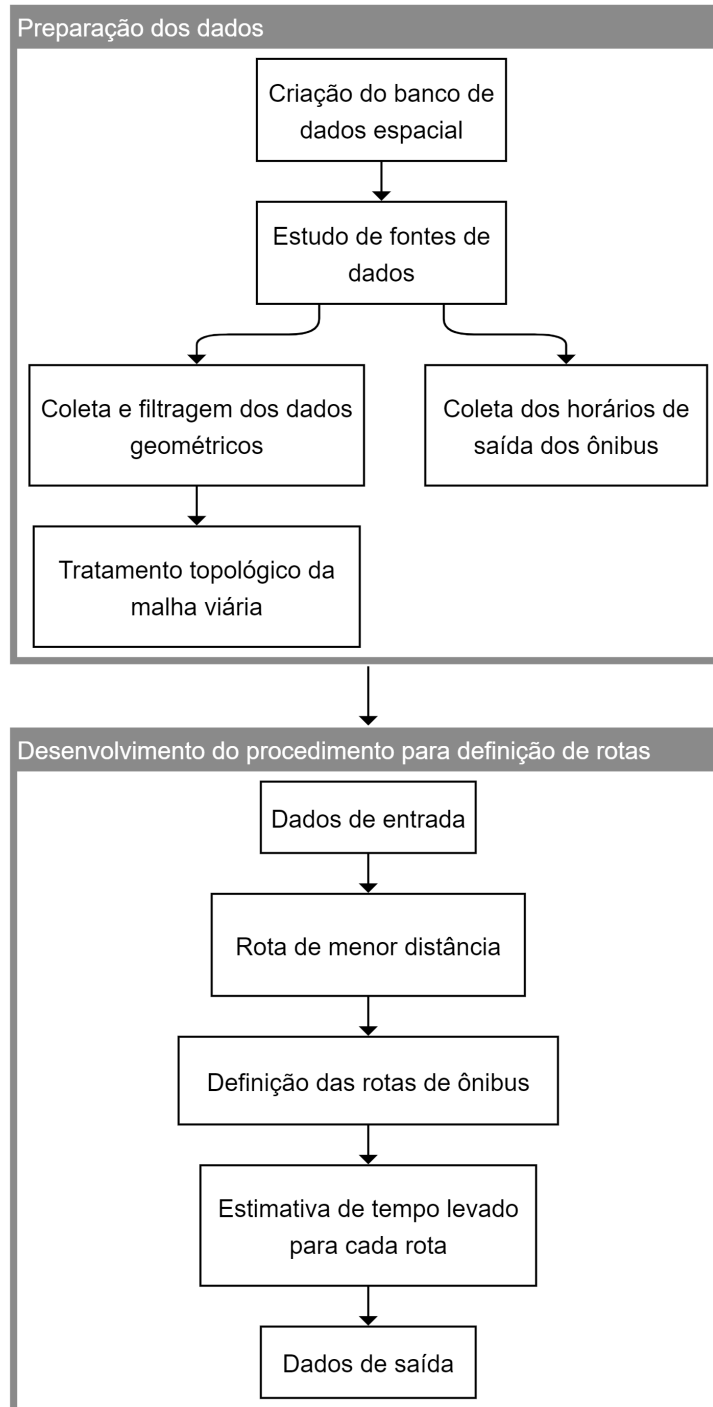


Figura 4: Método

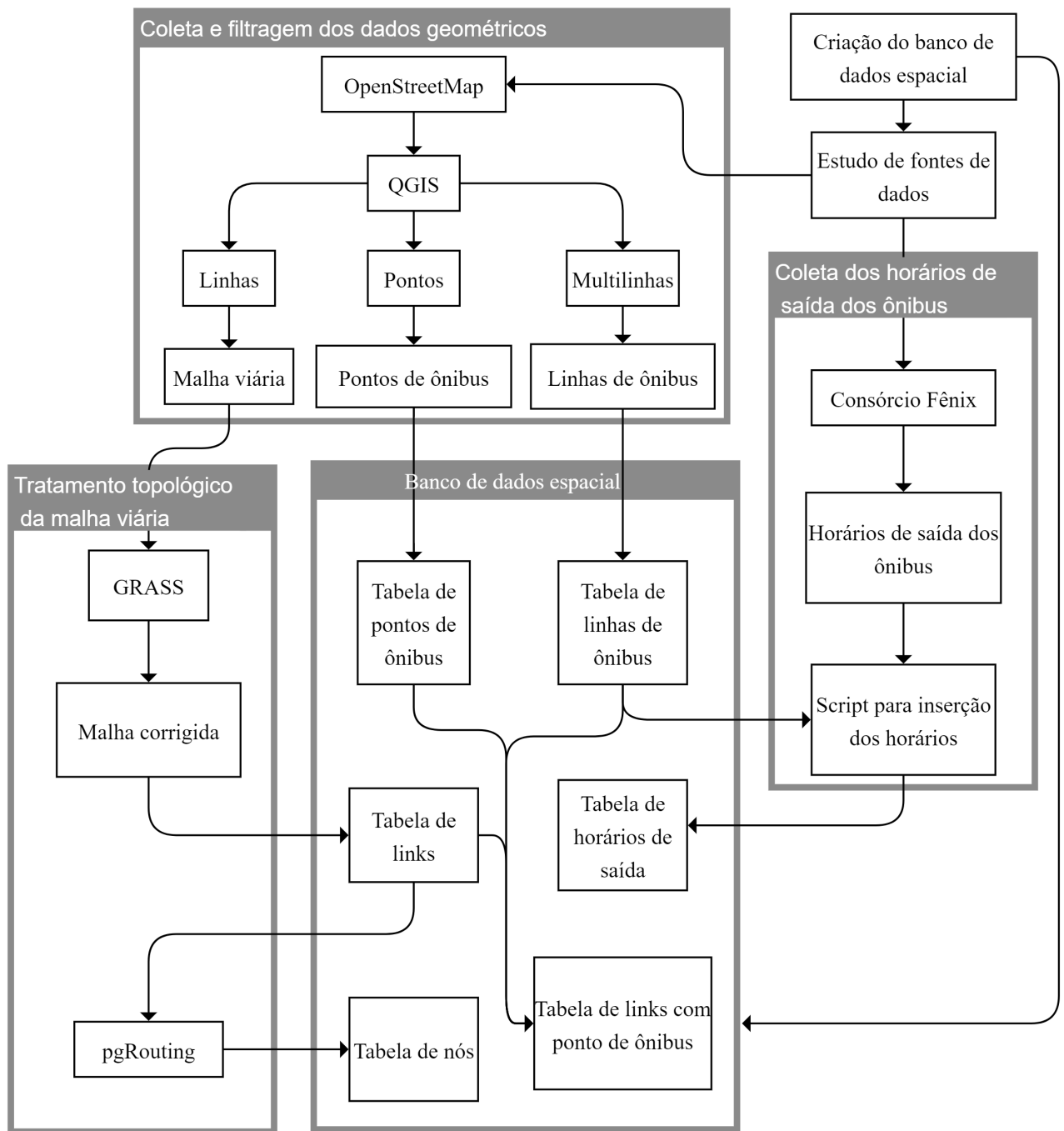


Figura 5: Preparação dos dados

Desta forma é possível armazenar os dados geométricos no banco de dados conforme estes são coletados.

3.1.2 Estudo de fontes de dados

O passo seguinte à criação do banco de dados foi estudar fontes para os dados necessários ao trabalho. Para dados geométricos, a fonte escolhida foi o *OpenStreetMap* (OSM)³, por conter informações geométricas, georreferenciadas, da infraestrutura de transporte suficientes para o desenvolvimento do método. Segundo Haklay e Weber (2008), o projeto OSM é uma coleção de dados que providencia mapas viários gerados por seus usuários. Os mesmos autores explicam que a função de exportação do OSM permite o *download* de dados do OSM em diferentes formatos vetoriais e *raster* para processamento. A Figura 6 mostra a interface do OSM acessado via web. É possível delimitar a área a ser exportada através da própria interface gráfica.

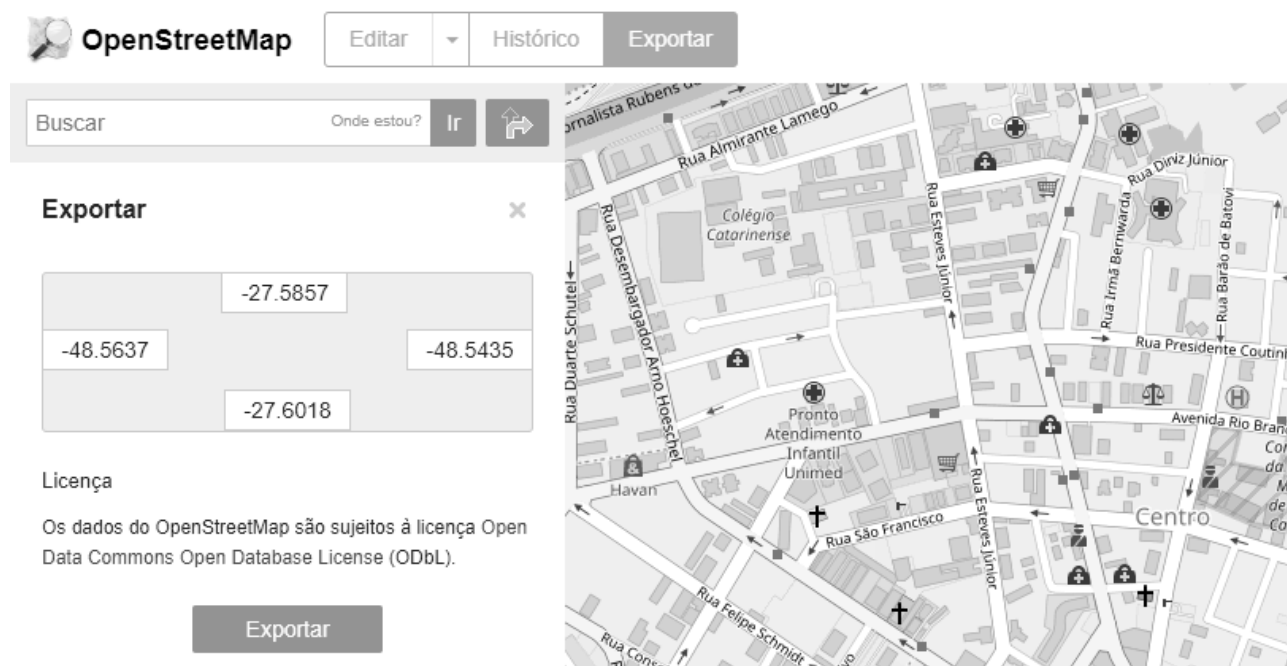


Figura 6: Interface do OSM

Foram importados dados do OSM em formato vetorial, guardadas em um arquivo no formato osm. Este arquivo foi então aberto no QGIS e suas camadas de dados foram carregadas. A partir das tabelas de atributos das camadas de dados, estes foram examinados para descobrir exatamente quanta informação estava disponível. A Figura 7 mostra a tabela de atributos da camada de linhas importada. Cada coluna da tabela representa alguma característica das linhas. As colunas relevantes para este trabalho são as colunas *name* e *highway*, que indicam, no caso de linhas que representem

³<https://www.openstreetmap.org/>

ruas, o nome da rua e o tipo de rua que a linha representa.

Além de linhas foram importadas camadas de multilinhas e pontos. Suas tabelas de atributos estão mostradas nas Figuras 8 e 9. No caso das multilinhas, na coluna *other_tags* é possível descobrir quais multilinhas representam linhas de ônibus ("route"=>"bus"), e a coluna *name* indica o nome da linha. Para os pontos, a coluna *highway* indica quais pontos são pontos de ônibus (*bus_stop*).

O *site* do Consórcio Fênix⁴ disponibiliza horários de saída e tempos estimados para o percurso total das linhas de ônibus. Na Figura 10 é mostrado um exemplo das informações encontradas.

3.1.3 Coleta e filtragem dos dados geométricos

Foram coletados, a partir do OSM, dados da região central de Florianópolis vista a familiaridade do autor com a malha viária e linhas de ônibus presentes nesta região, possibilitando a inspeção dos dados coletados. A inspeção dos dados vetoriais foi feita visualmente e a partir das tabelas de atributos no QGIS, de forma semelhante ao mostrado na seção 3.1.2.

As camadas de dados coletadas tiveram de ser filtradas por conterem mais dados do que o necessário, o que causaria interferência no procedimento para a definição das rotas. Além das rodovias, as linhas representam outros elementos como hidrovias e divisas administrativas. Se esses não fossem removidos, a representação da malha viária estaria incorreta, e rotas passando por linhas que não representam ruas poderiam ser geradas. Algumas das multilinhas coletadas representam rodovias estaduais, e algumas das linhas de ônibus tinham parte de seu traçado fora da área coletada. A maioria dos pontos coletados não representam pontos de ônibus mas sim outros objetos e localizações. Portanto foi utilizado o *QGIS* para filtrar estes dados de forma a se obter somente os dados desejados, conforme a tabela 1.

Tabela 1: Filtragem dos dados geométricos

Tipo de dado	Dado obtido
Linha	Malha viária
Multilinha	Traçado das linhas de ônibus
Ponto	Pontos de ônibus

A filtragem foi feita a partir das informações nas tabelas de atributos das camadas de dados (Figuras 7, 8 e 9). Foram utilizados os seguintes comandos SQL a na janela de filtragem do QGIS. Foram selecionadas linhas em que a coluna *highway* não estivesse vazia:

```
highway IS NOT NULL
```

Foram selecionados pontos em que a coluna *highway* estivesse definida como "bus_stop":

```
highway = 'bus_stop'
```

⁴<http://www.consorciofenix.com.br/horarios/>

	osm_id	name	highway	z_order	other_tags
1	454467253	Rua Conselheiro Mafra	pedestrian	0	
2	454467254		pedestrian	0	
3	454467256	Rua Conselheiro Mafra	pedestrian	0	
4	454467257	Rua Conselheiro Mafra	pedestrian	0	
5	458355805		unclassified	3	
6	458355806		living_street	0	
7	458355807	Servidão Luiz Vendelino Scholler	living_street	0	"oneway"=>"no", "surface"=>"paving_stones"
8	458355808	Servidão Cacapava	living_street	0	
9	458355809		living_street	0	
10	458355810	Rua José Felix Vieira	residential	3	"source:name"=>"IBGE"

Figura 7: Tabela de atributos da camada de linhas do OSM, extraída do QGIS

	osm_id	name	type	other_tags
1	2940034	Ônibus 846: Cacupé, TISAN => TITRI	route	transport:version"=>"2", "ref"=>"846", "route"=>"bus"
2	3177671	Ônibus 174: Saco Grande via João Paulo, TITRI => Bairro	route	ref"=>"174", "route"=>"bus"
3	2168842	Ônibus 138: Volta ao Morro Pantanal Sul	route	ref"=>"138", "route"=>"bus", "to"=>"TICEN"
4	178168	SC-404	route	

Figura 8: Tabela de atributos da camada de multilinhas

	osm_id	name	barrier	highway	other_tags
19	4483160778	Garagem Pio XII			"amenity"=>"parking", "fee"=>"yes", "lit"=>"yes"
20	4483161895				"amenity"=>"restaurant"
21	4483970899				"tourism"=>"viewpoint"
22	4484311889			crossing	
23	4484311890			crossing	
24	4497483636			<u>bus_stop</u>	"addr:street"=>"Rua Dep. Antonio Edu Vieira"
25	4510299826	Autoescola Geração Júnior			"amenity"=>"driving_school"

Figura 9: Tabela de atributos da camada de pontos

consórciofênix horários notícias atualiza

buscar horário e itinerário da linha

137 - Volta ao Morro Pantanal Norte

137 - Volta ao Morro Pantanal Norte

Característica: Alimentadora TICEN
Tempo de percurso: 00:54 aproximado
Alterada em: 06/03/2017

Tarifa - Região Única
Cartão: R\$ 3,71
Dinheiro: R\$ 3,90

Exibir Itinerário Exibir Mapa

Dia: 15/11/2017 operação com quadro de horário de Domingo.

Dias Úteis - Saída TICEN - Plataforma A lado 2

05:05	05:40E	06:12
07:06	07:19	07:32E
08:48	09:16	09:46
11:32E	11:51	12:09
13:13	13:30	13:47
15:06	15:26	15:46
16:49	17:04	17:19
18:23	18:40E	19:00
20:55	21:26E	21:59

Sábado - Saída TICEN - Plataforma A lado 2

05:00	06:00	06:20
07:40	08:00	08:30
10:30	11:00	11:20
12:40	13:00	13:20
15:00	15:40	16:20
19:00	19:40	20:20
23:00	23:40	

Figura 10: Horários de saída de ônibus retirados do site do Consórcio Fênix

E por fim, multilinhas com a tag "route"=>"bus_stop":

```
other_tags LIKE '%"route"=>"bus"%'
```

No caso das linhas de ônibus, foram removidas as linhas que estivessem visivelmente incompletas, por estarem parcialmente fora da área coletada. Além disso, foram deixadas somente onze linhas, o que já é suficiente para o desenvolvimento do método.

Na Figura 11 está ilustrada parte dos dados antes e depois do processo de filtragem. É possível observar que a maioria dos pontos, que representam árvores, lojas e outras localidades, são removidos, deixando somente os pontos de ônibus. Além disso, linhas que não representam a malha viária foram

removidas.

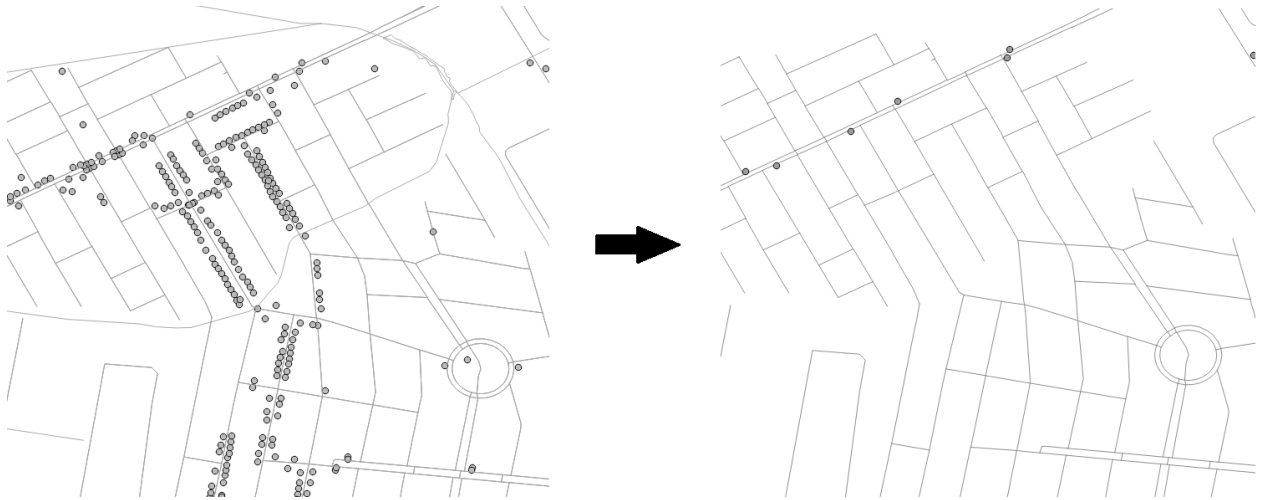


Figura 11: Resultado da filtragem

Após a filtragem das camadas de dados geométricas, os pontos e linhas de ônibus foram inseridas no banco de dados através do QGIS. Ao se fazer isso, foram criadas automaticamente as tabelas no banco de dados contendo as mesmas informações que as tabelas de atributos, com adição de uma coluna de identificação dentro do banco de dados e uma contendo a geometria dos elementos inseridos.

3.1.4 Tratamento topológico da malha viária

Quando são coletados dados de uma malha viária, é possível que haja problemas na modelagem e que a representação geométrica das vias não esteja conectada corretamente. Nestes casos, para que seja possível gerar um grafo a partir da malha, é necessário o tratamento topológico desta malha.

Nos dados da malha viária coletados para este trabalho, foram encontrados alguns problemas desta natureza. A Figura 12 é um *screenshot* do QGIS onde é mostrado um caso de interseção entre *links* da malha. A linha mais grossa representa um único *link* que está atravessando outros nos pontos circulados.

O tratamento topológico foi feito com o módulo *v.clean* do GRASS. Para isso a camada de dados contendo a malha viária foi exportada do QGIS para um arquivo *shapefile*, que foi então importado pelo GRASS. Então foram utilizadas as funções *break* e *snap*. A função *break* quebrou os *links* da malha nas interseções detectadas. A função *snap* foi então utilizada para garantir a conexão entre os *links* da malha.

É importante notar que estas funções foram aplicadas em todos os *links* da malha. Não foi feita uma análise para se determinar quais *links* deveriam ser ignorados, como *links* que representem viadutos ou túneis.

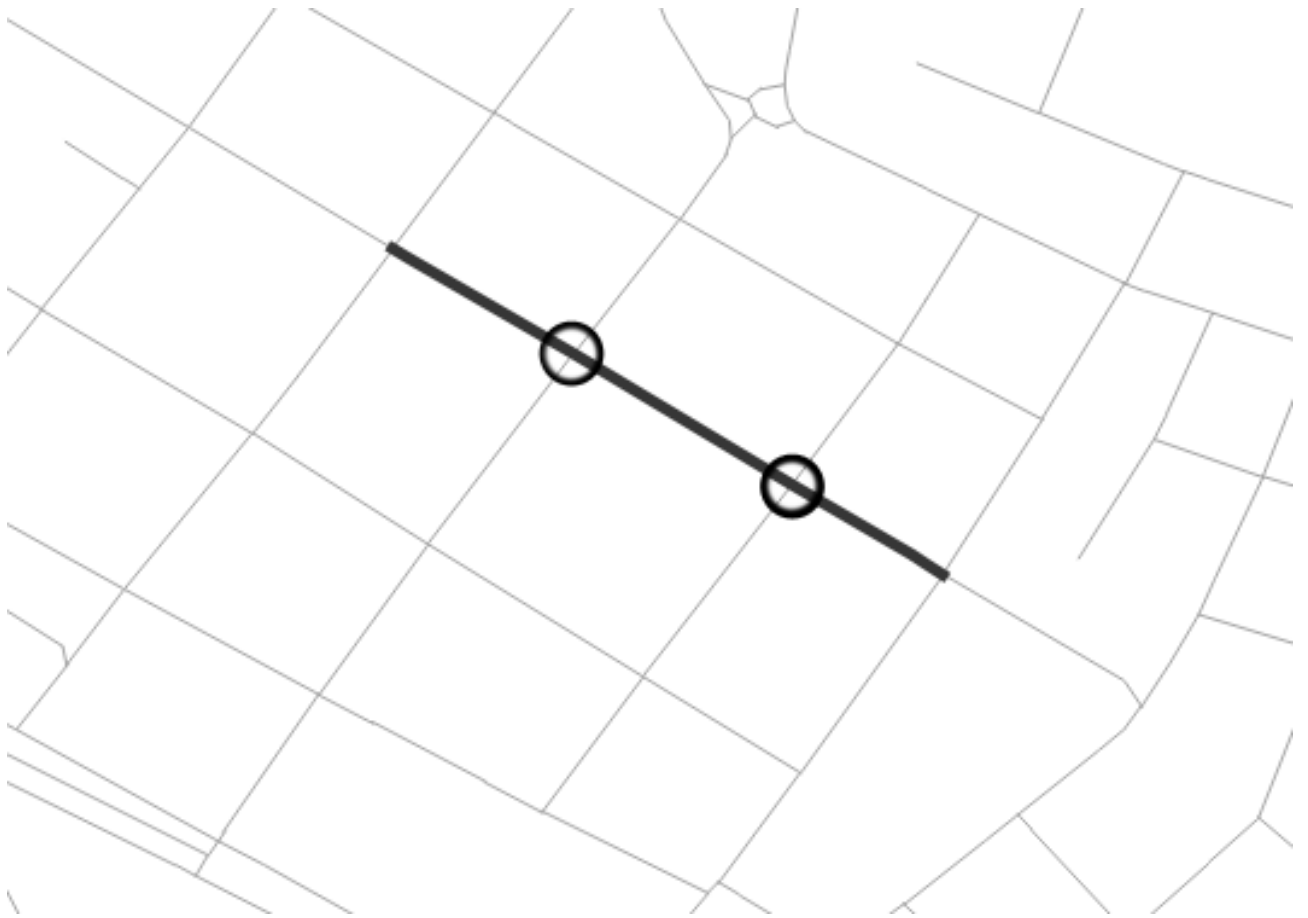


Figura 12: Interseção entre *links* da malha

Com a malha corrigida, esta foi inserida no banco de dados, gerando a tabela de *links*. Então foram criadas duas colunas nesta tabela para conter os nós de início (*source*) e fim (*target*), com os seguintes comandos:

```
ALTER TABLE bus_lines ADD COLUMN source integer ;  
ALTER TABLE bus_lines ADD COLUMN target integer ;
```

Na sequência utilizada a função *pgr_create_topology* do *pgrouting* a partir da geometria dos *links* para gerar a tabela de nós e preencher as colunas de nós de início e fim, dando como parâmetros o nome da tabela de *links* (*edges*) e a tolerância (distância mínima entre nós a serem gerados):

```
SELECT pgr_create_topology('edges',0.0001);
```

Desta forma os *links* são representados em forma de grafo, possibilitando o uso da função de roteamento.

3.1.5 Coleta dos horários de saída dos ônibus

Com os dados geométricos todos inseridos no banco de dados, foram coletados os horários de saída e tempos de percurso total para cada linha de ônibus. Os tempos de percurso das linhas são utilizadas para se estimar a velocidade média dos ônibus. Os horários indicam a disponibilidade do serviço.

Estes dados foram obtidos no *site* do Consórcio Fênix, em formato de texto. Os horários são diferentes dependendo do dia da semana. Em Florianópolis, os horários de saída dos ônibus são mais frequentes em dias úteis e menos em sábados, domingos e feriados. Algumas linhas operam somente em dias úteis.

Para armazenar os horários de saída foi criada no banco de dados uma tabela com quatro colunas:

- identificação;
- linha de ônibus;
- dia da semana (útil, sábado ou domingo);
- horário de saída.

Isto foi feito executando o seguinte comando:

```
CREATE TABLE bus_dept_times (  
id PRIMARY KEY,  
bus_line_id integer REFERENCES bus_lines ,  
day_group varchar(4) ,  
time time);
```

A identificação da linha de ônibus está relacionada à tabela de linhas de ônibus. Desta forma, com o dia da semana e linha utilizada, é possível consultar os horários de saída da linha.

Para cada uma das linhas de ônibus coletadas, os seus horários foram copiados e inseridos no banco de dados através de um *script* em Python, encontrado no Apêndice B deste trabalho. Um exemplo do comando SQL utilizado pelo *script* para inserir cada horário está mostrado a seguir:

```
INSERT INTO bus_dept_times (bus_line_id , day_group , time)  
VALUES (5 , ' util ' , 16:20:00);
```

Desta forma foi possível preencher a tabela de maneira rápida e eficiente.

3.2 Desenvolvimento do procedimento para definição das rotas

O procedimento para definição das rotas de ônibus foi desenvolvido a partir de programação de um *script* na linguagem Python, que se encontra no Apêndice C. Este *script* realiza consultas ao banco de dados baseadas nos dados de entrada. A partir das informações obtidas nas consultas é definida a

rota de menor distância, e a partir desta são definidas as rotas de ônibus. Então é feito o cálculo do tempo para cada rota. As rotas são então apresentadas em ordem crescente de tempo levado. Este procedimento está ilustrado no fluxograma da Figura 13 e será descrito em maior detalhe nas seções 3.2.1 a 3.2.5.

3.2.1 Dados de entrada

Os dados de entrada do método são os seguintes:

- ponto de partida;
- ponto de chegada;
- data e horário de partida.

Estes são definidos no próprio *script*. Desta forma o *script* precisa ser alterado sempre que os dados de entrada forem alterados.

Neste trabalho, pontos de partida e chegada são definidos por suas coordenadas x e y, e são utilizados na determinação da rota de menor distância.

O dia da semana, extraído da data, e o horário de partida determinam a disponibilidade do serviço de transporte coletivo com base nos horários de saída armazenados no banco de dados.

3.2.2 Rota de menor distância

Primeiramente é utilizada a função `ST_Distance` do PostGIS para se determinar os nós da malha mais próximos aos pontos de partida e chegada. Esta função recebe como parâmetros duas geometrias, e retorna a sua distância. A função é aplicada para se determinar a distância de todos os nós aos pontos de partida e chegada, e são encontrados os nós com os menores valores retornados. Isto é feito com o seguinte comando:

```
SELECT id , ST_Distance(the_geom::geography ,ST_GeomFromText('POINT
(-48.552658 -27.599111)',4326)) AS dist FROM nodes ORDER BY dist LIMIT
1;
```

Há dois pontos a serem observados no comando:

- a função `ST_GeomFromText` é usada para gerar uma geometria de ponto a partir de coordenadas;
- como as geometrias estão definidas em graus decimais, é utilizado o termo `::geography` para que a distância seja retornada em metros.

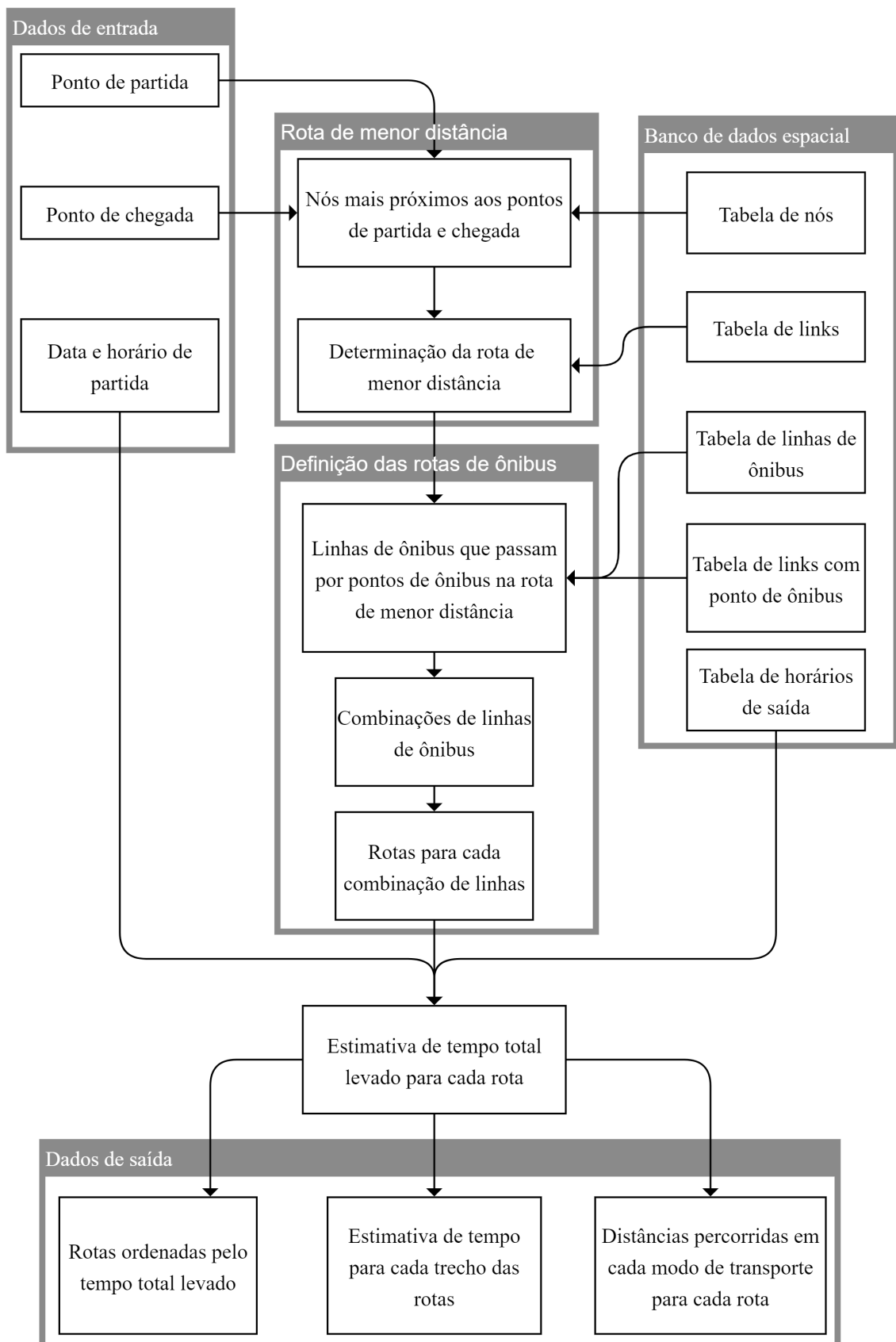


Figura 13: Procedimento para definição de rotas

É então utilizada a função `pgr_dijkstra` do `pgRouting` para se determinar a rota de menor distância entre estes nós. Como entrada da função são informados os *links* da malha, mais especificamente seu nó de início, nó de fim e comprimento (para o cálculo da distância percorrida), e os nós encontrados pela função `ST_Distance`. O comprimento dos *links* é calculado através da função `ST_Length` do `PostGIS`. O roteamento é feito no modo não-direcionado, pois a rota pode ser percorrida a pé. É retornada a sequência de *links* da rota de menor distância. O comando SQL utilizado é o seguinte:

```
SELECT node , edge , cost , seq FROM pgr_dijkstra (
' SELECT id , source , target , ST_Length(the_geom:: geography) AS cost
FROM edges ', {1}, {2}, false );
```

- o comprimento de cada *link*, em metros, é definido como o custo, de forma que a rota resultante é a de menor distância;
- {1} e {2} são substituídos pelos ids dos nós de início e fim da rota;
- o termo "*false*" nesta posição indica que o grafo não é direcionado. Se este fosse direcionado, seria necessário informar o parâmetro "*reverse_cost*".

3.2.3 Definição das rotas de ônibus

Para acelerar o procedimento de definição das rotas de ônibus, foi criada uma coluna na tabela de pontos de ônibus relacionando estes aos *links* da malha, de forma a se determinar quais *links* possuem pontos de ônibus:

```
ALTER TABLE bus_stops ADD COLUMN closest_edge integer;
```

Foi utilizada a função `ST_Intersects` entre os *links* e as linhas de ônibus, e a partir dos resultados positivos (*links* interceptados pelas linhas) foi utilizada a função `ST_Distance` entre os *links* e pontos de ônibus para se determinar os *links* mais próximos a cada ponto. A coluna `closest_edge` de cada ponto foi então definida com o id do *link* correspondente. Foi utilizado o seguinte comando para realizar esta operação:

```
UPDATE bus_stops SET closest_edge = ide FROM (
SELECT DISTINCT idb , first_value(ide) OVER (PARTITION BY idb ORDER BY
dist) AS ide FROM (
SELECT b.id as idb , e.id as ide , ST_Distance(e.the_geom , b.the_geom) AS dist
FROM edges e , bus_stops b , bus_lines l
WHERE ST_Intersects(e.the_geom , l.the_geom)) AS foo
) AS bar WHERE id = idb;
```


Para se definir as rotas de ônibus a partir da rota de menor distância, é inicialmente consultada a tabela do banco de dados que define quais *links* têm pontos de ônibus:

```
SELECT closest_edge FROM bus_stops ;
```

A partir dos valores retornados são determinados quais *links* da rota de menor distância têm pontos de ônibus. Então é utilizada a função ST_Intersects entre estes *links* e as linhas de ônibus para se determinar quais destas passam pelos *links* selecionados:

```
SELECT l.id FROM bus_lines l, edges e
WHERE e.id = {1} AND ST_Intersects(e.the_geom, l.the_geom);
```

- {1} é substituído pelo id do *link*.

Não é considerado o sentido das linhas de ônibus, ou seja, o método supõe que todas as linhas vão e voltam pelo mesmo caminho. É comum que isso ocorra, mas em alguns casos há diferenças entre o caminho de ida e volta. Além disso, pode haver diferenças nos horários de saída para cada sentido da linha.

Com isso são formadas listas de quais *links* da rota de menor distância têm acesso a quais linhas de ônibus. A partir destas listas é possível combinar linhas de ônibus para formar rotas com mais de uma linha.

Para combinações de mais de uma linha de ônibus, de forma a simplificar o procedimento, somente são considerados válidos os casos onde o último ponto de ônibus de cada linha precede ou coincide com o primeiro ponto da próxima. A tabela 2 mostra um exemplo de uma rota pela qual passam três linhas de ônibus. O "x" representa pontos de ônibus na rota por onde passa a linha de ônibus da coluna respectiva. Neste caso seriam consideradas as linhas individualmente e a combinação da Linha 1 com a Linha 3. O primeiro ponto da Linha 2 precede o último da Linha 1, enquanto o último ponto da Linha 2 sucede o primeiro da Linha 3. Com essa simplificação, combinações entre a Linha 2 e uma das outras não são consideradas.

<i>Link</i>	Linha 1	Linha 2	Linha 3
1			
2	x	x	
3			
4	x	x	x
5			
6		x	x
7			
8			

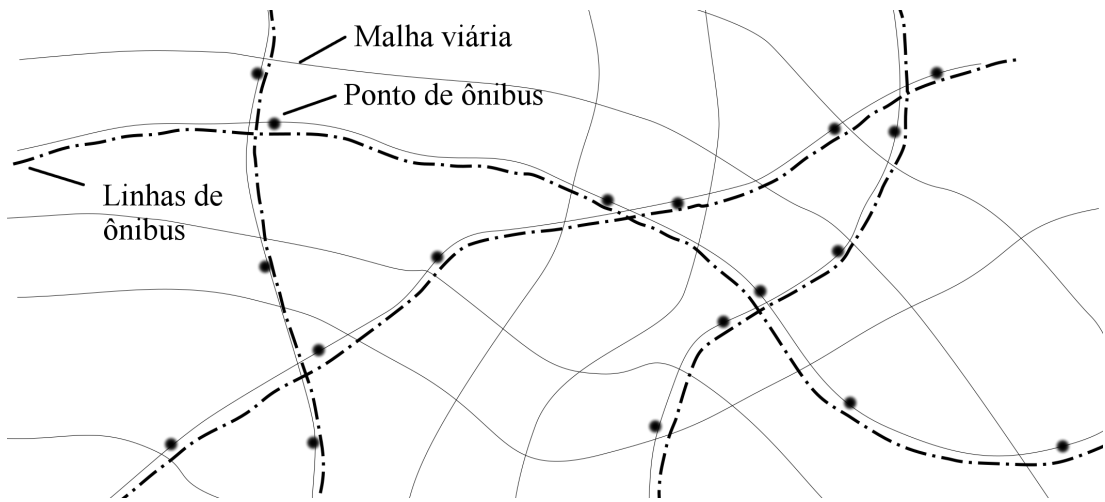
Tabela 2: Listas de pontos de ônibus por *link*

Com as combinações definidas as rotas são definidas. Baseado nos pontos de embarque e desembarque de cada rota, é atribuído o modo de transporte para cada *link*. Para os *links* entre um ponto de embarque e desembarque da mesma linha de ônibus, o modo de transporte é o ônibus, para os outros, a pé.

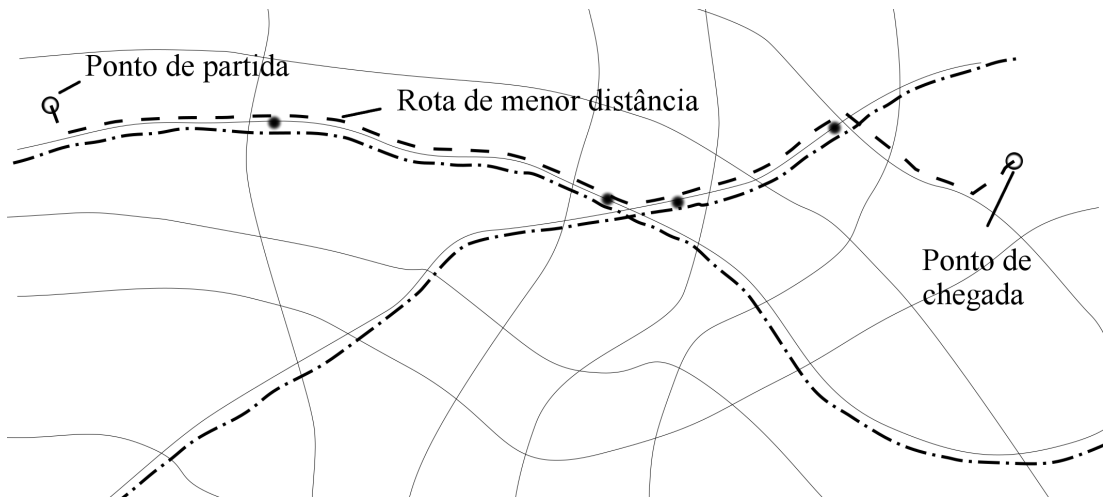
A Figura 14 ilustra o procedimento de definição de uma rota. A Figura 14a ilustra um exemplo de malha viária com linhas e pontos de ônibus. Na Figura 14b é definida a rota de menor distância, e são mostrados então somente pontos de ônibus nesta rota, e linhas que passam por estes pontos. A Figura 14c mostra um exemplo de rota combinando as duas linhas de ônibus que passam pela rota de menor distância.

É possível que haja divergências entre a rota de menor distância e as rotas definidas. Isto ocorre pois a condição para que uma linha de ônibus seja considerada pelo método é que haja pontos de ônibus na rota de menor distância pelos quais a linha passe. Neste sentido, se a linha de ônibus possuir um traçado diferente do da rota de menor distância, porém interceptá-la em pelo menos dois pontos de ônibus (embarque e desembarque), esta será levada em consideração na definição das rotas. Na Figura 15 é possível observar um exemplo disso: os traçados são distintos no trecho mostrado, mas se interceptam no ponto de ônibus indicado, possibilitando o embarque.

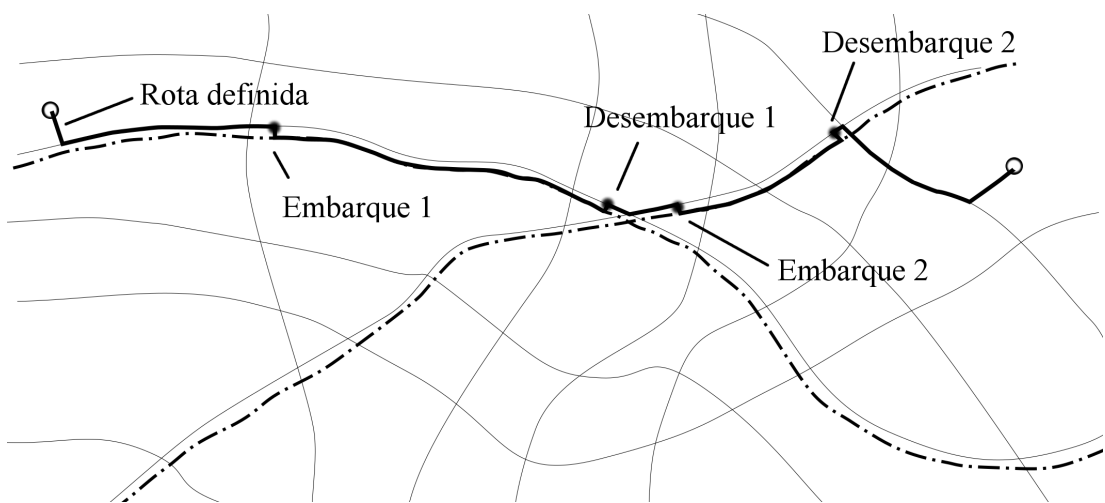
A diferença na distância não é considerada pelo método. O caminho percorrido é considerado como igual ao da rota de menor distância. Isto causa pequenas discrepâncias nas distâncias medidas para a parte da rota percorrida pelo ônibus, porém a parte percorrida a pé e os pontos de embarque e desembarque não são afetados.



(a) Malha viária com linhas e pontos de ônibus



(b) Rota de menor distância



(c) Rota definida

Figura 14: Definição de uma rota

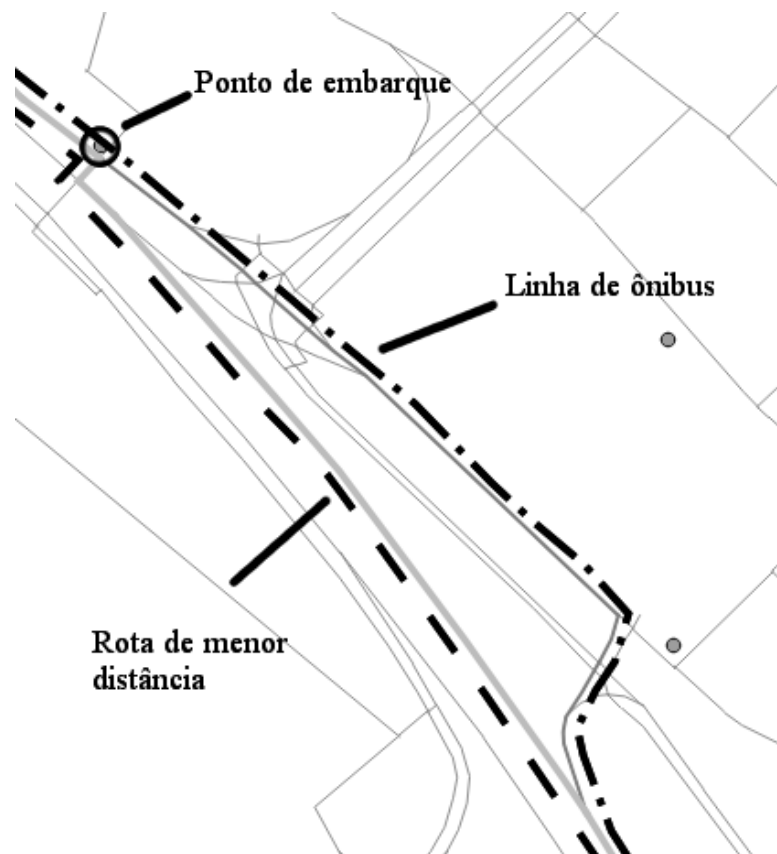


Figura 15: Divergência entre rotas

3.2.4 Estimativa de tempo levado para cada rota

Para cada rota definida na seção 3.2.3 é estimado o tempo levado para que se determine as rotas mais rápidas. O procedimento para a estimativa do tempo levado está ilustrado na Figura 16.

Os tempos estimados são divididos em três grupos:

- tempo de percurso do usuário a pé;
- tempo de percurso do usuário utilizando o ônibus;
- tempos de espera para o usuário subir no ônibus.

Para cada rota as distâncias percorridas a pé e de ônibus são calculadas a partir dos pontos de embarque e desembarque e dos *links* da rota inicial. É então aplicada uma velocidade, diferente para cada modo de transporte, para se estimar o tempo levado em cada trecho da rota, conforme a Equação 1.

$$t = \frac{d}{v} \quad (1)$$

Onde:

- t é o tempo levado;

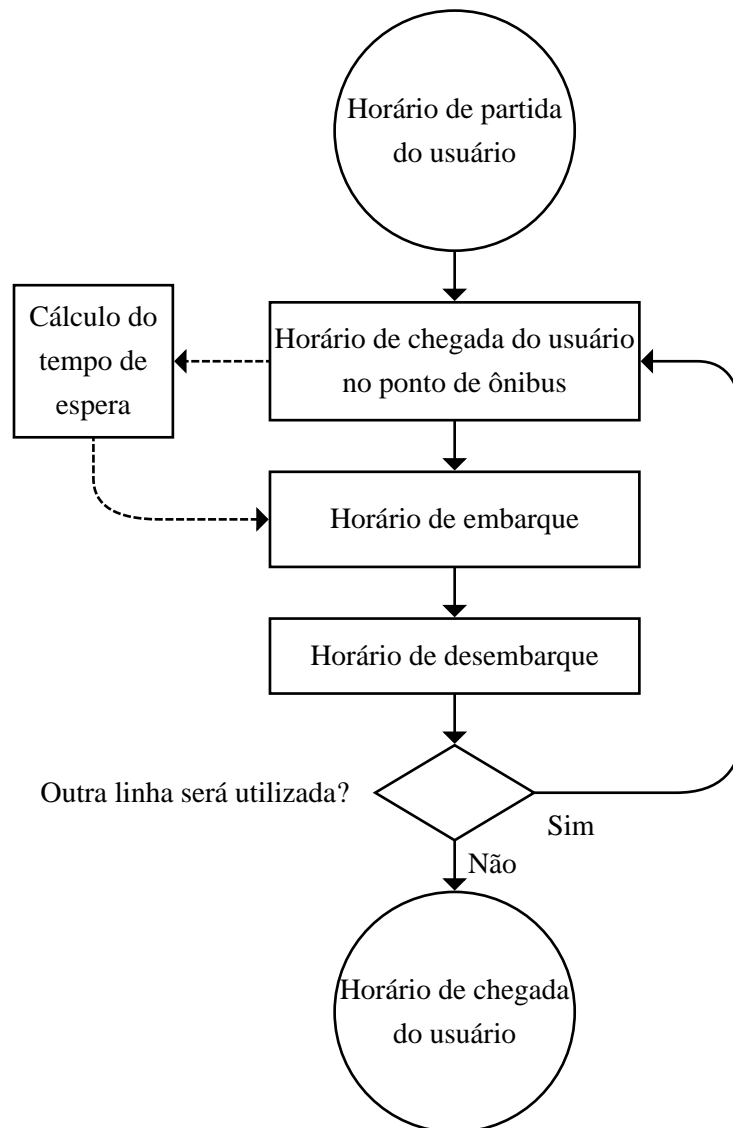


Figura 16: Estimativa do tempo levado

Nota: As linhas contínuas representam o tempo levado ao longo do percurso da rota.

- d é a distância percorrida;
- v é a velocidade de locomoção.

A velocidade a pé foi considerada como $0,91 \text{ m/s}$ (KNOBLAUCH et al., 1996). A velocidade do ônibus v_{onibus} (Equação 2) foi calculada dividindo a distância total da linha de ônibus d_{linha} pelo tempo estimado de percurso da linha t_{linha} , obtido do *site* do Consórcio Fênix.

$$v_{onibus} = \frac{d_{linha}}{t_{linha}} \quad (2)$$

O tempo total levado t_{total} é a soma do tempo levado em cada trecho e dos tempos de espera, como mostrado na Equação 3:

$$t_{total} = \sum_{i=1}^n (t_{pei} + t_{ei} + t_{pdi}) + t_{pc} \quad (3)$$

Onde:

- n é o número de linhas de ônibus utilizadas;
- t_{pe} é o tempo levado para o usuário chegar no ponto de embarque, a partir do ponto de partida ou do último ponto de desembarque;
- t_e é o tempo de espera até o ônibus chegar no ponto de embarque, a partir do momento em que o usuário chega;
- t_{pd} é o tempo levado para o usuário, no ônibus, chegar ao ponto de desembarque;
- t_{pc} é o tempo levado para o usuário chegar ao ponto de chegada.

O tempo de espera t_e depende do horário em que o usuário chega no ponto de ônibus h_{up} (Equação 4):

$$t_e = f(h_{up}) \quad (4)$$

A estimativa do tempo total levado t_{total} , portanto, é feita trecho a trecho, de maneira sequencial. O horário h_{up} é a soma do horário de partida do usuário h_p com os tempos calculados até o usuário chegar no ponto de embarque (Equação 5).

$$h_{upi} = h_p + \sum_{j=1}^i t_{pej} + \sum_{j=1}^{i-1} (t_{ej} + t_{pdj}) \quad (5)$$

Para se estimar o tempo de espera t_e , é feita uma estimativa do tempo que o ônibus levará para chegar ao ponto de embarque t_{op} . Isto é feito utilizando a função `pgr_dijkstra` dando como entrada o primeiro nó de linha de ônibus e o nó mais próximo ao ponto de ônibus, e os *links* por onde passam linhas de ônibus. A distância obtida é então dividida pela velocidade. O tempo resultante é então subtraído do horário em que o usuário chega no ponto de embarque h_{up} , resultando no horário em que o ônibus deveria partir para chegar ao mesmo tempo em que o usuário, como mostrado na Equação 6.

$$h'_{os} = h_{up} - t_{op} \quad (6)$$

Este horário é então comparado com os horários de saída da linha de ônibus e dia da semana correspondentes. Se o ônibus partir antes do horário calculado, o usuário não o alcançará a tempo. Logo

é considerado o horário de saída h_{os} imediatamente após o calculado (h'_{os}). A diferença entre estes horários é o tempo de espera para o respectivo ponto de embarque (Equação 7).

$$t_e = h_{os} - h'_{os} \quad (7)$$

3.2.5 Dados de saída

Por fim, as rotas são ordenadas pelo tempo levado e são mostradas as de menor tempo. Os *links* percorridos por cada rota são agrupados pelo modo de transporte e pelo nome da rua correspondente, para facilitar a leitura dos resultados.

O agrupamento está ilustrado nas tabelas 3 e 4. A tabela 3 mostra a rua, distância, modo de transporte e tempo levado para cada *link* da rota. A tabela 4, por sua vez mostra estas informações agrupadas por modo de transporte e rua.

Tabela 3: Trechos da rota

Rua	Comprimento	Modo de transporte	Tempo
Rua A	100 m	A pé	60 s
Rua A	200 m	Ônibus	40 s
Rua A	500 m	Ônibus	100 s
Rua B	300 m	Ônibus	60 s
Rua B	200 m	Ônibus	40 s
Rua B	400 m	Ônibus	80 s
Rua C	600 m	Ônibus	120 s
Rua C	300 m	A pé	180 s
Rua C	200 m	A pé	120 s

Tabela 4: Trechos da rota agrupados

Rua	Comprimento	Modo de transporte	Tempo
Rua A	100 m	A pé	60 s
Rua A	700 m	Ônibus	140 s
Rua B	900 m	Ônibus	180 s
Rua C	600 m	Ônibus	120 s
Rua C	500 m	A pé	300 s

São informadas as distâncias percorridas a pé e de ônibus, além das estimativas de tempo levado para cada trecho da rota.

4 Resultados

Os resultados do método são conjuntos de instruções para o uso do sistema de transporte coletivo. Devido às limitações do método e simplificações feitas, a qualidade dos resultados varia significativamente dependendo das particularidades de cada caso.

A Figura 17 mostra um exemplo da saída do método. No caso são mostradas somente as duas melhores rotas identificadas. Para cada rota são mostradas as distâncias a serem percorridas a pé e de ônibus, o tempo estimado para cada trecho e o tempo de espera para o usuário subir no ônibus. Além disso são mostradas as distâncias e tempo totais. Conforme descrito anteriormente, pode-se observar que o método não considera o sentido da linha de ônibus: as linhas da primeira e segunda opções tem sentidos opostos, mas ambas aparecem no resultado. Por outro lado, as distâncias são condizentes com a realidade.

Opção 1:

Linhas:

Ônibus 136: Volta ao Morro Carvoeira Sul

Horário de partida: 10:54.

Percorrer 296 metros na Rua Lauro Linhares. (2 minutos)

Esperar 7 minutos para subir na linha Ônibus 136: Volta ao Morro Carvoeira Sul.

Percorrer 2383 metros na Rua Lauro Linhares. (7 minutos)

Descer do ônibus.

Percorrer 1151 metros na Rua Lauro Linhares. (11 minutos)

Horário de chegada: 11:24

Distância total a pé: 1447 metros

Distância total de ônibus: 2382 metros

Tempo total: 30 minutos.

Opção 2:

Linhas:

Ônibus 137: Volta ao Morro Pantanal Norte

Horário de partida: 10:54.

Percorrer 296 metros na Rua Lauro Linhares. (2 minutos)

Esperar 7 minutos para subir na linha Ônibus 137: Volta ao Morro Pantanal Norte.

Percorrer 2043 metros na Rua Lauro Linhares. (6 minutos)

Descer do ônibus.

Percorrer 1491 metros na Rua Lauro Linhares. (14 minutos)

Horário de chegada: 11:27

Distância total a pé: 1787 metros

Distância total de ônibus: 2042 metros

Tempo total: 32 minutos.

Figura 17: Exemplo de saída do método

A Figura 18 mostra uma rota correspondente ao resultado da Figura 17.

A Figura 19 mostra um exemplo da saída com mais de uma linha sendo utilizada. Neste caso a

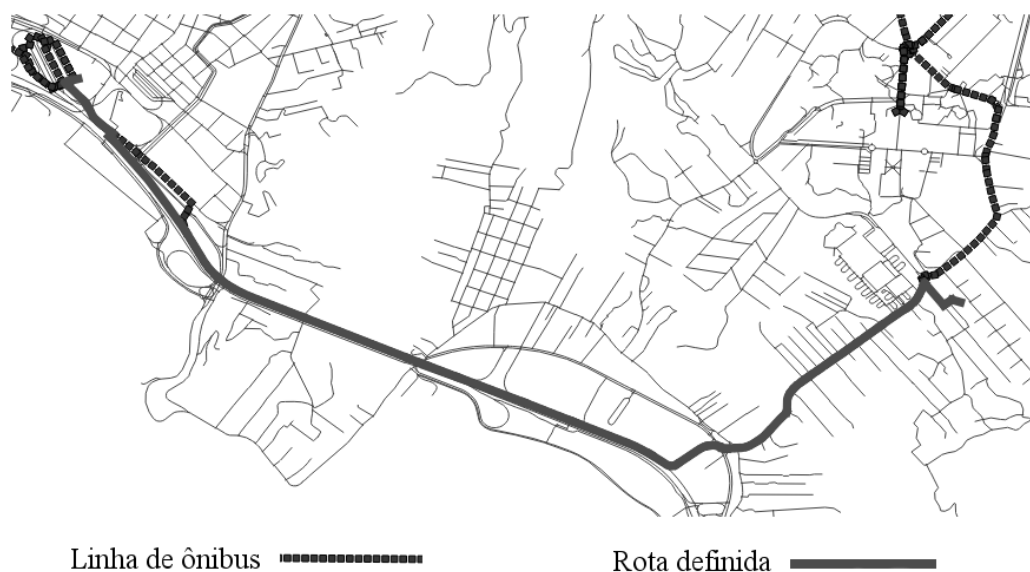


Figura 18: Rota mostrada no mapa

saída é semelhante, exceto que há dois pontos de embarque e dois de desembarque.

Opção 3:

Linhas:

Ônibus 131: Agronômica via Gama D'Eça, TITRI => TICEN,

Ônibus 177: Santa Mônica, UFSC => TITRI

Horário de partida: 11:03.

Percorrer 296 metros na Rua Lauro Linhares. (2 minutos)

Esperar 3 minutos para subir na linha Ônibus 131: Agronômica via Gama D'Eça, TITRI => TICEN.

Percorrer 454 metros na Rua Lauro Linhares. (1 minutos)

Descer do ônibus.

Percorrer 1389 metros na Rua Lauro Linhares. (13 minutos)

Esperar 0 minutos para subir na linha Ônibus 177: Santa Mônica, UFSC => TITRI.

Percorrer 540 metros na Rua Lauro Linhares. (1 minutos)

Descer do ônibus.

Percorrer 1151 metros na Rua Lauro Linhares. (11 minutos)

Horário de chegada: 11:38

Distância total a pé: 2835 metros

Distância total de ônibus: 994 metros

Tempo total: 35 minutos.

Figura 19: Exemplo de saída com duas linhas de ônibus

Rotas com mais de uma linha de ônibus raramente são os melhores resultados, principalmente devido aos tempos de espera. Isto também ocorre pela pequena área utilizada no trabalho, pois dos casos em que poderia ser usada uma combinação de linhas, normalmente é possível utilizar somente uma linha.

5 Conclusões

Ao longo do trabalho foi criado um sistema que a partir dos dados de entrada do usuário retorna rotas para uso das linhas de ônibus.

Durante o desenvolvimento do método foram aplicados diversos conceitos relacionados a SIG, banco de dados espacial e *scripting*, utilizando *software* apropriados para a coleta, processamento e gerenciamento dos dados.

O QGIS se mostrou muito útil para a visualização e inspeção dos dados, sendo de fácil uso. A sua integração com o PostGIS, em particular, facilitou a inserção de dados no banco de dados espacial.

A coleta de dados a partir do OSM apresentou alguns problemas. Quando se exporta camadas de dados em arquivos *shapefile* do OSM, estas perdem grande parte de suas características topológicas. Neste sentido, o GRASS se mostrou muito eficiente para o tratamento destas falhas, processando-as rapidamente, sem gerar mais *links* do que o necessário.

O banco de dados espacial baseado no PostgreSQL se mostrou útil ao gerenciamento dos dados coletados, facilitando o relacionamento destes dados.

A geração de rotas a pé e de ônibus se mostrou uma tarefa de alta complexidade, tanto em relação ao processamento dos dados quanto à programação com base nestes, sendo necessárias uma série de simplificações no método.

Devido a estas simplificações, a qualidade dos resultados é reduzida. A não-consideração do sentido das linhas de ônibus, em particular, distancia o método de uma aplicação funcional. Com algumas medidas relativamente simples, como alterações na maneira com que as rotas são definidas, seria possível melhorar um pouco os resultados, mas eliminar os resultados ruins é uma tarefa mais complexa, exigindo um estudo mais aprofundado de como trabalhar os dados geométricos coletados.

A saída do método, feita através de linha de comando, é bem informativa e detalhada, porém não permite a visualização das rotas geradas através de um mapa, o que facilitaria a interpretação dos resultados.

5.1 Recomendações para trabalhos futuros

- Considerar rotas entre os pontos de ônibus próximos aos pontos de partida e chegada, de forma a aumentar a probabilidade de que as linhas passantes por estes pontos sejam utilizadas. Isto melhoraria os resultados ao se reduzir a distância percorrida a pé pelo usuário.
- Considerar o caminho efetivamente percorrido pelo ônibus nas rotas definidas, de forma a reduzir a discrepância entre a distância percorrida apresentada no resultado e a real.
- Estudar o desenvolvimento de aplicações gráficas para possibilitar a definição visual, através de um mapa, dos dados de entrada, e a representação visual das rotas resultantes.

- Estudar de forma mais aprofundada as características geométricas das linhas de ônibus e como relacioná-las aos *links* da malha viária poderia ajudar a considerar o sentido das linhas, melhorando consideravelmente os resultados do método.

Referências

- BURROUGH, Peter A. **Principles of Geographical Information Systems for Land Resources Assessment**. 1986. Pp. 5, 6.
- CHERRY, Christopher; HICKMAN, Mark; GARG, Anirudh. Design of a Map-Based Transit Itinerary Planner. **Journal of Public Transportation**, v. 9, n. 2, 2006. Pp. 1, 2, 4.
- CNT. Pesquisa Mobilidade da População Urbana, 2017. Pp. 1, 2.
- CODD, E. F. **The Relational Model for Database Management**. 1990. P. 9.
- DATE, C. J. **Instructor's Manual for An Introduction to Database Systems**. 8. ed. 2003. P. 9.
- GRASS DEVELOPMENT TEAM. **GRASS GIS 7.2.3 Reference Manual**. 2017. Pp. 11, 12.
- GÜTING, Ralf Hartmut. An Introduction to Spatial Database Systems. **Praktische Informatik IV, FernUniversität Hagen**, 1994. Pp. 6, 10.
- HAGGET, P; CHORLEY, R J. Models, paradigms and the new geography. **Models in Geography**, 1967. P. 5.
- HAKLAY, Mordechai; WEBER, Patrick. OpenStreetMap: User-Generated Street Maps. **IEEE Computer Society**, 2008. P. 17.
- HAWEROTH, Flávia. **Aplicação de Roteirização e Programação de Veículos no Transporte Público de Pessoas com Deficiência no Município de Joinville - SC**. 2017. Universidade Federal de Santa Catarina. P. 5.
- HEYWOOD, Ian; CORNELIUS, Sarah; CARVER, Steve. **An Introduction to Geographical Information Systems**. 2006. Pp. 6, 7.
- HICKMAN, Mark. Robust Passenger Itinerary Planning Using Transit AVL Data. **Transportation Research Board**, 2002. P. 4.
- INTERNATIONAL ENERGY AGENCY. **Bus Systems for the Future: Achieving Sustainable Transport Worldwide**. 2002. P. 2.
- KNOBLAUCH, Richard; PIETRUCHA, Martin; NITZBURG, Marsha. Field Studies of Pedestrian Walking Speed and Start-Up Time. **Transportation Research Board**, 1996. P. 31.
- LACERDA, Sander Magalhães. Precificação de congestionamento e transporte coletivo urbano. **BNDES Setorial**, 2006. P. 2.
- LINDEN, Ricardo. Técnicas de Agrupamento. **Revista de Sistemas de Informação da FSMA**, n. 4, 2009. Pp. 7, 8.
- LUTZ, Mark. **Programming Python**. 2011. P. 13.

- MILANI, André. **PostgreSQL: Guia do Programador**. 2008. P. 10.
- MONMONIER, Mark. **How to Lie with Maps**. 1991. Pp. 5, 6.
- NETELER, Markus; MITASOVA, Helena. **Open Source Gis A Grass Gis Approach**. 2007. P. 11.
- NIENKOTTER, Maiko Andrei. **Proposta de um modelo heurístico para a roteirização de empilhadeiras em um armazém de grande porte**. 2017. Universidade Federal de Santa Catarina. P. 4.
- OBE, Regina O.; HSU, Leo S. **PostGIS in Action**. 2015. Pp. 7, 10, 11.
- PGROUTING TEAM. **pgRouting Manual**. 22 jul. 2017. P. 11.
- RAMEZ, Elmasri; NAVATHE, Shamkant B. **Sistemas de Banco de Dados**. 2005. Pp. 8, 9.
- SILVA, D. **Sistemas Inteligentes no Transporte Público por Ônibus**. 2000. Diss. (Mestrado) – Universidade Federal do Rio Grande do Sul. P. 2.
- SUMATHI, S.; ESAKKIRAJAN, S. **Fundamentals of Relational Database Management Systems**. 2007. Pp. 9, 10.
- TRÉPANIÉ, Martin; CHAPLEAU, Robert; ALLARD, Bruno. Can Trip Planner Log Files Analysis Help in Transit Service Planning? **Journal of Public Transportation**, v. 8, n. 2, 2005. P. 1.

APÊNDICE A - Comandos SQL

Seção 3.1.1:

```
CREATE DATABASE mydb;
CREATE EXTENSION postgis;
CREATE EXTENSION pgrouting;
```

Seção 3.1.3: (somente a condição WHERE, usados na janela de filtragem do QGIS)

```
highway IS NOT NULL
highway = 'bus_stop'
other_tags LIKE '%"route"=>"bus"%'
```

Seção 3.1.4:

```
ALTER TABLE bus_lines ADD COLUMN source integer;
ALTER TABLE bus_lines ADD COLUMN target integer;
SELECT pgr_create_topology('edges',0.0001);
```

Seção 3.1.5:

```
CREATE TABLE bus_dept_times (
  id PRIMARY KEY,
  bus_line_id integer REFERENCES bus_lines,
  day_group varchar(4),
  time time);
INSERT INTO bus_dept_times (bus_line_id,day_group,time)
VALUES (5,'util',16:20:00);
```

Seção 3.2.2:

```
SELECT id, ST_Distance(the_geom::geography,ST_GeomFromText('POINT(-48.552658 -27.599111)',4326)) AS
  dist FROM nodes ORDER BY dist LIMIT 1;
SELECT node,edge,cost,seq FROM pgr_dijkstra(
'SELECT id, source, target, ST_Length(the_geom::geography) AS cost
FROM edges', {1}, {2}, false);
```

Seção 3.2.3:

```
ALTER TABLE bus_stops ADD COLUMN closest_edge integer;
UPDATE bus_stops SET closest_edge = ide FROM (
SELECT DISTINCT idb, first_value(ide) OVER (PARTITION BY idb ORDER BY dist) AS ide FROM (
SELECT b.id as idb,e.id as ide,ST_Distance(e.the_geom,b.the_geom) AS dist FROM edges e, bus_stops b,
  bus_lines l
WHERE ST_Intersects(e.the_geom,l.the_geom)) AS foo
) AS bar WHERE id = idb;
SELECT closest_edge FROM bus_stops;
SELECT l.id FROM bus_lines l, edges e
WHERE e.id = {1} AND ST_Intersects(e.the_geom,l.the_geom);
```

APÊNDICE B - *Script* de inserção dos horários de ônibus

```
1 from datetime import datetime ,time ,timedelta
2 import psycopg2
3
4 bus_line_id = 39
5 db_name = "mydb"
6 password = "pg"
7
8 class bus_dept:
9     pass
10
11 horarios = open("horarios.txt")
12 current_group = ""
13 bus_depts = []
14 for t in horarios.read().replace("\n", " ").split(" "):
15     if t in ["util", "sab", "dom"]:
16         current_group = t
17         continue
18     b = bus_dept()
19     b.group = current_group
20     b.time = datetime.strptime(t[0:5], "%H:%M").time()
21     bus_depts.append(b)
22
23 horarios.close()
24
25 values = []
26 for b in bus_depts:
27     values.append("{0}, '{1}', '{2}'".format(bus_line_id, b.group, b.time))
28     print(b.group, b.time)
29
30 con = psycopg2.connect("host=localhost dbname={0} user=postgres
31     password={1}".format(db_name, password))
32 con.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)
33 with con:
34     cur = con.cursor()
35     cur.execute("INSERT INTO bus_dept_times (id_line, day_group, time) VALUES
36         {0}".format(", ".join(values)))
```

APÊNDICE C - *Script* de roteamento

```
1  # -*- coding: utf-8 -*-
2  import psycopg2
3  from datetime import datetime, time, timedelta
4
5  # =====
6  class outer_node(object): # representa o primeiro e ultimo nos da rota
7      def __init__(self, id, distance):
8          self.id = id
9          self.distance = distance
10 class route_segment(object): # representa os links da rota
11     def __init__(self, seq, length, edge = -1, node = -1, has_bus_stop = False, street = ""):
12         self.edge = edge
13         self.node = node
14         self.has_bus_stop = has_bus_stop
15         self.bus_lines = []
16         self.seq = seq
17         self.length = length
18         self.street = street
19 class bus_line(object): # representa linhas de onibus
20     def __init__(self, id, speed, name=""):
21         self.id = id
22         self.name = name
23         self.speed = speed
24 class combination(object): # representa uma combinação de linhas de onibus
25     def __init__(self, bus_lines = [], time = timedelta()):
26         self.bus_lines = bus_lines
27         self.time = time
28 class route_group(object): # representa um agrupamento de links dentro da rota
29     def __init__(self):
30         self.length = 0.0
31         self.riding_bus = False
32         self.has_embark_point = False
33         self.has_disembark_point = False
34     def time_taken(self):
35         if self.riding_bus:
36             #metros/metros por segundo
37             return timedelta(seconds=self.length/(self.bus_line.speed))
38         return timedelta(seconds=self.length/(0.97))
39 # =====
40 def generate_outer_edges(x, y, node_id, route_table):
41     cur.execute("""
42     INSERT INTO {2} (the_geom)
43     SELECT ST_MakeLine(ST_GeomFromText('POINT({0} {1})',4326),the_geom) FROM {4} WHERE id =
44         {3}""".format(x, y, route_table, node_id, node_table))
45 def get_closest_node(x, y):
46     cur.execute("SELECT id, ST_Distance(the_geom::geography,ST_GeomFromText('POINT({}
47         {})',4326)) AS dist FROM {} ORDER BY dist LIMIT 1".format(x, y, node_table))
48     node = cur.fetchone()
49     return outer_node(id = node[0], distance = node[1])
48 def group_route(embark_list, disembark_list, bus_lines): # agrupa os links por modal e por rua
49     groups = []
50     current_group = route_group()
```



```

51     for s in route:
52         if s.street:
53             current_group.street = s.street
54             break
55     for s in route:
56         if s.seq in embark_list + disembark_list or (s.street and s.street !=
           current_group.street):
57             groups.append(current_group)
58             current_group = route_group()
59             if s.seq in embark_list:
60                 current_group.has_embark_point = True
61                 current_group.first_node = s.node
62             if s.seq in disembark_list:
63                 current_group.has_disembark_point = True
64             current_group.riding_bus = True in list(s.seq >= embark_list[i] and s.seq <
           disembark_list[i] for i in range(len(embark_list)))
65             for i in range(len(embark_list)):
66                 if s.seq >= embark_list[i] and s.seq < disembark_list[i]:
67                     current_group.riding_bus = True
68                     current_group.bus_line = bus_lines[i]
69             current_group.street = s.street if s.street else groups[-1].street
70             current_group.length += s.length
71     groups.append(current_group)
72     return groups
73 def calculate_waiting_time(bl,node,current_time): # calcula o tempo de espera
74     start_nodes_table = "bus_line_start_nodes"
75     cur.execute("SELECT id_node FROM {0} WHERE id_line = {1}".format(start_nodes_table,bl.id))
76     start_node = cur.fetchone()[0]
77     cur.execute("""
78     SELECT cost FROM pgr_dijkstra(
79     'SELECT e.id, e.source, e.target, ST_Length(e.the_geom::geography) as cost FROM {0} e, {1}
           b WHERE b.id = {2} AND ST_Intersects(e.the_geom,b.the_geom)',
80     {3},{4},false)
81     """).format(edge_table,bus_line_table,bl.id,start_node,node))
82     time_to_bus_stop = timedelta(seconds=((sum(line [0] for line in cur.fetchall())/20))
83
84     weekday = departure_time.weekday()
85     util,sab,dom = [0,1,2,3,4],[5],[6]
86     if weekday in util:
87         day_group = "util"
88     elif weekday in sab:
89         day_group = "sab"
90     elif weekday in dom:
91         day_group = "dom"
92
93
94     cur.execute("SELECT time FROM bus_dept_times WHERE id_line = {0} AND day_group =
           '{1}'".format(bl.id,day_group))
95     time_list = list(line[0] for line in cur.fetchall())
96     waiting_time = min(t.seconds for t in ((current_time - time_to_bus_stop -
           datetime.combine(current_time.date(),time)) for time in time_list) if t > timedelta())
97     return timedelta(seconds=waiting_time)
98 def calculate_cost(comb): # calcula o tempo em segundos com base nas combinacoes selecionadas
99     embark_list = sorted(bl.first_instance for bl in comb.bus_lines)

```

```

100     disembark_list = sorted(bl.last_instance for bl in comb.bus_lines)
101     comb.instructions = ["Horário de partida: {0:%H:%M}.".format(departure_time.time())]
102     current_time = departure_time
103     groups = group_route(embark_list, disembark_list, comb.bus_lines)
104     for g in groups:
105         if g.has_embark_point:
106             waiting_time = calculate_waiting_time(g.bus_line.g.first_node, current_time)
107             comb.instructions.append("Esperar {0} minutos para subir na linha
108                                     {1}.".format(waiting_time.seconds//60, g.bus_line.name))
109             current_time += waiting_time
110         if g.has_disembark_point:
111             comb.instructions.append("Descer do ônibus.")
112             comb.instructions.append("Percorrer {0:0.0f} metros na
113                                     {1}.".format(g.length, g.street) + " ({}
114                                     minutos)".format(g.time_taken().seconds//60) if g.time_taken().seconds >= 60
115                                     else " (menos de 1 minuto)")
116             current_time += g.time_taken()
117     comb.instructions.append("Horário de chegada: {0:%H:%M}.".format(current_time.time()))
118     comb.instructions.append("Distância total a pé: {0:0.0f} metros".format(sum(g.length for g
119                                     in groups if not g.riding_bus)//1))
120     comb.instructions.append("Distância total de ônibus: {0:0.0f} metros".format(sum(g.length
121                                     for g in groups if g.riding_bus)//1))
122     comb.instructions.append("Tempo total: {0}
123                               minutos.".format((current_time-departure_time).seconds//60))
124
125     comb.total_time = current_time - departure_time
126
127     # =====
128     db_name = "mydb"
129     password = "pg"
130     bus_stops_table = "bus_stops_sample"
131     bus_line_table = "bus_lines_sample"
132     node_table = "nodes"
133     edge_table = "edges"
134     start_x, start_y = -48.552658, -27.599111
135     finish_x, finish_y = -48.518273, -27.607809
136     departure_time = datetime.now()-timedelta(hours=10)
137     bus_line_cond = ""
138     # =====
139     con = psycopg2.connect("host=localhost dbname={0} user=postgres
140                            password={1}.".format(db_name, password))
141     con.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)
142     with con:
143         cur = con.cursor()
144         #busca id e nome das linhas de onibus
145         cur.execute("SELECT id, name, ST_Length(the_geom:: geography), time_taken FROM
146                     {0}.".format(bus_line_table))
147         all_bus_lines = list(bus_line(id=line[0], name=line[1], speed=line[2]/line[3].seconds) for
148                               line in cur.fetchall())
149         # busca quais links estao proximos de ponto de onibus (tabela criada para isso)
150         cur.execute("SELECT closest_edge from {0}.".format(bus_stops_table))
151         edges_with_bus_stop = list(line[0] for line in cur.fetchall())
152         first_node = get_closest_node(start_x, start_y)
153         last_node = get_closest_node(finish_x, finish_y)
154         # encontra uma rota sem considerar direção ou linhas de onibus, e considerando custo como o

```

```

    comprimento do link
144 cur.execute("""
145 SELECT node, edge, cost, seq FROM pgr_dijkstra(
146 'SELECT id, source, target, ST_Length(the_geom::geography) as cost
147 FROM {0}',
148 {1},{2},false)
149 """).format(edge_table, first_node.id, last_node.id)
150 route = []
151 bus_lines = []
152 for row in cur.fetchall(): #busca as informacoes dos links
153     s = route_segment(node = row[0], edge = row[1], has_bus_stop = row[1] in
        edges_with_bus_stop, length = row[2], seq = row[3])
154     if s.has_bus_stop: # se o link atual possui um ponto de onibus, busca quais linhas
        de onibus passam pelo link definindo o primeiro e ultimo pontos de cada linha
155         cur.execute("SELECT b.id FROM {0} b, {1} e WHERE {3} e.id = {2} AND
            ST_Intersects(e.the_geom,b.the_geom)").format(bus_line_table, edge_table, s.edge, bus_line_table)
156         ids = list(row[0] for row in cur.fetchall())
157         s.bus_lines = list(bl for bl in all_bus_lines if bl.id in ids)
158         for bl in s.bus_lines:
159             bl.last_instance = s.seq
160             if bl not in bus_lines:
161                 bl.first_instance = s.seq
162                 bus_lines.append(bl)
163         route.append(s)
164 cur.execute("DELETE FROM sample_route")
165 for s in route: #insere a rota na tabela sample
166     cur.execute("""
167     INSERT INTO sample_route
168     SELECT the_geom FROM edges
169     WHERE id = {0}""").format(s.edge)
170 generate_outer_edges(start_x, start_y, first_node.id, "sample_route")
171 generate_outer_edges(finish_x, finish_y, last_node.id, "sample_route")
172 cur.execute("SELECT id,name FROM edges WHERE name is not null AND id =
        ANY(ARRAY[{0}])").format(", ".join(list(str(s.edge) for s in route)))
173 for row in cur.fetchall(): #define o nome da rua de cada link
174     for s in route:
175         s.street = row[1] if s.edge == row[0] else ""
176 route.insert(0,route_segment(seq = 0, length = first_node.distance))
177 route.append(route_segment(seq = max(s.seq for s in route) + 1, length =
        last_node.distance))
178 # remove linhas de onibus que apareçam somente uma vez
179 bus_lines = list(bl for bl in bus_lines if bl.first_instance != bl.last_instance)
180 combs = []
181 for bl1 in bus_lines: # cria as combinacoes de linhas
182     combs.append(combination([bl1]))
183     for bl2 in bus_lines:
184         if bl2.first_instance >= bl1.last_instance:
185             combs.append(combination([bl1,bl2]))
186 for comb in combs: # calcula o tempo para cada combinação
187     calculate_cost(comb)
188 nl= 666
189 nc = len(combs)
190 for i in range(nl) if nl < nc else range(nc): # mostra as instrucoes para as n melhores
        combinacoes

```

```
191         comb = sorted(combs, key=lambda r: r.total_time)[i]
192         print(comb.bus_lines[0].speed)
193         print("Opção {0}:\n\nLinhas: \n{1} \n".format(i + 1, "\n".join(list(l.name
194             for l in comb.bus_lines))))
195         print("\n".join(comb.instructions)+"\n")
```