UNISINOS

Programa Interdisciplinar de Pós-Graduação em

Computação Aplicada

Mestrado Acadêmico

Rodrigo Bastos

Determinação de Caminhos Mínimos em Aplicações de Transporte Público:

Um Estudo de Caso para a Cidade de Porto Alegre

São Leopoldo, 2013

Rodrigo Bastos

DETERMINAÇÃO DE CAMINHOS MÍNIMOS EM APLICAÇÕES DE TRANSPORTE PÚBLICO: Um Estudo de caso para a Cidade Porto Alegre

Dissertação de Mestrado apresentada como requisito parcial para a obtenção do título de Mestre, pelo Programa Interdisciplinar de Pós-Graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos – UNISINOS.

Orientador: Dra. Patrícia Augustin Jaques Maillard

Co-Orientador: Dr. Leonardo Dagnino Chiwiacowsky

São Leopoldo

2013

Ficha catalográfica

B327d Bastos, Rodrigo
Determinação de caminhos mínimos em aplicações de transporte público : um estudo de caso para a cidade de Porto Alegre / por Rodrigo Bastos. – 2013.

111 f. : il., 30cm.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, 2013.
Orientação: Profª. Drª. Patrícia Augustin Jaques Maillard ; Coorientação: Prof. Dr. Leonardo Dagnino Chiwiacowsky.

1. Sistemas de transportes inteligentes. 2. Sistemas de informação ao usuário. 3. Problema de caminhos mínimos. 4. Algoritmos de busca. I. Título.

Catalogação na Fonte:

Bibliotecária Vanessa Borges Nunes - CRB 10/1556

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus por me dar força e sabedoria para a execução deste trabalho.

A minha amada esposa Gabriela que soube compreender a minha ausência e pelo incentivo oferecido nos momentos mais difíceis do mestrado.

A toda a minha família que me incentivou e me deu força em todos os momentos.

Ao Tiago Sommer e ao Thiago Konrad pelo auxílio prestado no tratamento dos dados relativos ao transporte público de Porto Alegre.

Aos meus orientadores, Patrícia Jaques Maillard e Leonardo Dagnino Chiwiacowsky que me aconselharam, incentivaram e conduziram este trabalho rumo aos resultados.

Ao projeto SIMTUR (Edital CTIC/RNP), pela bolsa.

RESUMO

O crescente aumento do uso de automóveis e de motocicletas tem provocado uma contínua degradação no trânsito urbano das grandes metrópoles. Este cenário é agravado pelas deficiências nos atuais sistemas de transporte público, geradas, em parte, pela falta de informação ao usuário. O presente trabalho apresenta um modelo computacional para um sistema de informação ao usuário de transporte público. Ao contrário de outros trabalhos baseados no algoritmo clássico Dijkstra, a abordagem apresentada faz uso do algoritmo A* para resolução do problema de caminhos mínimos, presente neste contexto, a fim de reduzir o tempo de resposta de maneira que o modelo possa ser utilizado em um sistema real de informação ao usuário. O modelo proposto considera múltiplos critérios de decisão, como a distância total percorrida e o número de transbordos. Um estudo de caso foi realizado utilizando dados reais do transporte público da cidade Porto Alegre com o objetivo de avaliar o modelo computacional desenvolvido. Os resultados gerados foram comparados com aqueles obtidos através do emprego do algoritmo Dijkstra e indicam que a combinação do algoritmo A* com técnicas de aceleração permite reduzir, significativamente, a complexidade de espaço, o tempo de processamento e o número de transbordos.

Palavras-Chave: Sistemas de Transportes Inteligentes. Sistemas de Informação ao Usuário. Problema de Caminhos Mínimos. Algoritmos de Busca.

ABSTRACT

The increasing use of automobiles and motorcycles has caused a continuous degradation in the traffic of large cities. This scenario gets worse due to shortcomings in the current public transportation, which is entailed, in a certain way, by the lack of information provided to the user. This study shows a computing model for a public transportation user information system. Unlike other studies based on the classical Dijkstra's algorithm, the approach makes use of the algorithm A* to solve a shortest path problem to reduce the response time so that the model can be used in an real-time web information system. The proposed model takes into account multiple criteria of decision, such as total distance traveled and number of transfers and it was evaluated with data from Porto Alegre's public transportation. The results were compared to those ones obtained by the use of Dijkstra's algorithm and indicate that the combination of algorithm A* with acceleration techniques allows reducing significantly the space complexity, processing time and the number of transfers.

Keywords: Intelligent Transportation Systems, User Information Systems, Shortest Path Problem, Search Algorithms.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE SIGLAS

AF	*Arc Flags*
APTS	Sistemas Avançados de Transporte Público
ATMS	Sistemas Avançados de Gerenciamento de Tráfego
ATIS	Sistemas Avançados de Informação ao Viajante
AVCS	Sistemas Avançados de Controle Veicular
BRT	*Bus Rapid Transit*
CAN	*Controler Area Network*
CVO	Operação de Veículos Comerciais
EPTC	Empresa Pública de Transporte e Circulação
ETTM	Sistemas de Gestão Eletrônica de Tráfego e Pedágio
FIFO	*First In First Out*
GPS	*Global Positioning System*
HH	*Highway Hierarchies*
HNR	*Highway Node Routing*
ITS	Sistemas Inteligentes de Transporte
LIFO	*Last In First Out*
SAAT	Sistemas Automatizados de Arrecadação Tarifária
SAO	Sistemas de Ajuda à Operação
SIU	Sistemas de Informação ao Usuário
SPTrans	São Paulo Transporte
TNR	*Transit Node Routing*
TRENSURB	Empresa de Trens Urbanos de Porto Alegre
TRI	Transporte Integrado

SUMÁRIO

1 INTRODUÇÃO

O transporte é um motor do desenvolvimento econômico e do lazer em todo o mundo na sociedade contemporânea (PIARC, 2000). Os cidadãos dependem diariamente dos meios de transporte para acessar o local de trabalho, comprar suprimentos, visitar atrações turísticas e centros de lazer (como cinemas, shoppings, entre outros). Entretanto, as deficiências dos sistemas atuais de transporte público, assim como o crescente aumento do uso de automóveis e de motocicletas, têm provocado uma contínua degradação do trânsito urbano nas grandes metrópoles. O contínuo crescimento do modo de transporte rodoviário tem sido um dos maiores desafios para os governantes da maioria das cidades do mundo, devido à predominância do uso de automóveis. Cada novo veículo em circulação implica em aumento dos congestionamentos, da poluição atmosférica e na possibilidade de acidentes, exigindo mais recursos governamentais para controlar as consequências negativas provocadas por esta modalidade de transporte. Em paralelo, crescem as preocupações mundiais com relação à sustentabilidade em função dos custos econômicos, sociais e ambientais provocados pelos impactos decorrentes das atividades humanas.

No Brasil, diversas medidas têm sido tomadas com o intuito de mudar este panorama atual. Um exemplo deste fato é a lei 12.587 (BRASIL, 2012), criada em janeiro de 2012, que visa instituir diretrizes para a criação de uma política nacional de mobilidade urbana. Outros exemplos, restritos a algumas cidades brasileiras, abrangem a criação de faixas (corredores) destinadas exclusivamente aos veículos de transporte público e a restrição do uso de meios privados de locomoção (rodízio de placas) visando à promoção de meios de transportes coletivos, entre outros.

Contudo, como estas medidas não vêm associadas a um transporte público eficiente e uma abrangente quantidade de informações, geralmente não surtem o efeito esperado. Por outro lado, evidências indicam que um dos principais fatores de desmotivação no emprego do transporte coletivo é a falta de informação (FTA, 2006). Sendo assim, as grandes cidades necessitam de um sistema inteligente de informação que permita o planejamento efetivo das viagens de seus usuários. Esses sistemas podem fornecer informações em tempo real sobre possíveis itinerários e duração da viagem, assim como estimativa do tempo de espera e possíveis conexões.

Atualmente no Brasil existem alguns sistemas desta natureza disponibilizados aos usuários por empresas permissionárias ou órgãos gestores. Um exemplo é o Poatransporte (2012), disponibilizado pela Empresa Pública de Transporte e Circulação (EPTC) juntamente com a prefeitura da cidade de Porto Alegre, que permite ao usuário visualizar, em uma interface gráfica, as rotas das linhas de ônibus oferecidas na cidade de Porto Alegre. Entretanto, esse sistema, assim como outros disponíveis no Brasil, considera apenas o endereço de destino desejado e a partir desta informação exibe os pontos de parada próximos desta localização bem como as linhas de ônibus que passam por estes pontos. Este sistema, por exemplo, não permite que sejam informados um endereço de origem e um endereço de destino com o objetivo de determinar as linhas de ônibus aptas a percorrer este trajeto e também não considera eventuais transbordos (trocas de linhas) necessários durante o percurso.

Nas situações onde é preciso uma única linha de ônibus para percorrer um trajeto entre origem e destino desejado, uma consulta simples a um banco de dados seria suficiente para fornecer as informações a respeito do trajeto. Entretanto, nas situações onde é necessária a realização de um transbordo, o problema se torna mais complexo. Nestes casos, é preciso a utilização de algoritmos de roteamento, podendo este problema ser tratado como a determinação de um caminho mínimo entre um par de vértices em um grafo (CHERKASSKY; GOLDBERG; RADZIK, 1996). Em um grafo, um caminho é uma sequência alternada de vértices e arestas que inicia e termina em um vértice e o custo é definido segundo a soma dos pesos das arestas que compõem este caminho. Desta maneira, um caminho entre um vértice de origem u e um vértice de destino v é dito mínimo se não existir outro de menor custo entre u e v. O problema do caminho mínimo consiste na identificação de um caminho de menor custo entre um par de vértices (CORMEN et al., 2002).

Atualmente existem diversas estratégias de solução para o problema de caminhos mínimos aplicados ao transporte rodoviário, mas não muitas aplicadas ao transporte coletivo. O objetivo deste trabalho é propor um modelo computacional para o problema do caminho mínimo em uma malha de transporte público. Essa solução consiste na representação do problema através de um grafo e, como diferencial, o emprego do algoritmo A* para tratar o problema. A combinação do A* com técnicas de aceleração visa melhorar o desempenho do algoritmo, mantendo o menor custo do caminho e, ao mesmo tempo, reduzindo o número de transbordos. A partir de um endereço de origem e um endereço de destino é possível

determinar as melhores rotas de transporte público para o usuário, considerando como critérios de decisão a distância total percorrida e o número de transbordos. É possível também prover informações relevantes ao usuário a respeito do percurso incluindo nomes de ruas, localização de paradas e linhas de ônibus necessárias para percorrer o trajeto.

1.1 Objetivos

Esse trabalho tem como objetivo geral desenvolver um modelo computacional para o problema do caminho mínimo multicritério em uma malha de transporte público. No contexto dos trabalhos relacionados pesquisados, nenhum propõe a utilização do algoritmo A* para o problema do caminho mínimo em uma rede de transporte público e, portanto, tal aspecto pode ser destacado como uma das principais contribuições científicas deste trabalho. O termo multicritério, utilizado no contexto deste trabalho, se refere aos critérios de distância total percorrida e número de transbordos efetuados durante o percurso. O modelo deve fornecer informações detalhadas ao usuário sobre a rota (linhas de ônibus, paradas e conexões necessárias) e determinar as melhores rotas ao usuário considerando como critérios a distância percorrida e o número de linhas utilizadas e também reduzir o tempo de resposta, de maneira que o modelo possa ser utilizado em um sistema real de informação ao usuário.

Para tanto, os seguintes objetivos específicos foram estabelecidos:

a) Aplicar o algoritmo A* para o problema de caminho mínimo envolvendo transporte público;
b) Empregar técnicas de aceleração que permitam reduzir o tempo de resposta do algoritmo de busca;
c) Adequar o algoritmo de busca a fim de reduzir o número de transbordos;
d) Realizar uma validação experimental do modelo computacional desenvolvido através da execução de testes para o caso de uso da cidade de Porto Alegre.

1.2 Organização do Texto

Na organização deste trabalho, no capítulo 2 é revisado o estado da arte dos sistemas de transportes inteligentes, no Brasil e no mundo bem como os conceitos básicos relacionados aos sistemas avançados de transporte público, sua categorização, principais funções, tecnologias utilizadas e exemplos de aplicações existentes. No capítulo 3 são revisados os principais conceitos relacionados à Teoria dos Grafos necessários ao entendimento e desenvolvimento dos objetivos traçados, enquanto no capítulo 4 são apresentados os

algoritmos de busca de caminhos mínimos. No capítulo 5, é realizado um resumo sobre trabalhos relacionados à determinação de caminhos mínimos em aplicações rodoviárias e transporte público, enquanto no capítulo 6 é apresentado o trabalho realizado e os principais desafios relacionados ao problema. No capítulo 7 são apresentados os resultados obtidos e, no capítulo 8 são apresentadas as conclusões obtidas no desenvolvimento deste trabalho, possibilidades de utilização, evoluções e aperfeiçoamentos.

2 SISTEMAS DE TRANSPORTES INTELIGENTES

Os Sistemas de Transporte Inteligente, ou em inglês *Intelligent Transportation Systems* (ITS), são sistemas que utilizam tecnologia da informação como um mecanismo de resolução de problemas relacionados ao transporte em geral (PAPAGEORGIOU, 2003). Automação de autoestradas, sistemas automáticos de coletas de pedágios, sistemas de informação ao usuário e sistemas de controle de tráfego são exemplos de tecnologias utilizadas nos ITS.

O emprego destas tecnologias avançadas ao transporte coletivo urbano está se tornando uma realidade em várias cidades no Brasil e no mundo, graças à evolução constante nos setores de informática e de telecomunicação. Para Ferraz e Torres (2004), os ITS aplicados ao transporte coletivo urbano possibilitam atuar de forma direta na melhoria da segurança, na melhoria do controle da operação, no incremento da produtividade, na redução dos atrasos, na redução dos congestionamentos e na redução das emissões de poluentes.

Os ITS utilizam diferentes tecnologias nos vários setores dos transportes. Segundo Sussman (2000), os ITS podem ser categorizados em:

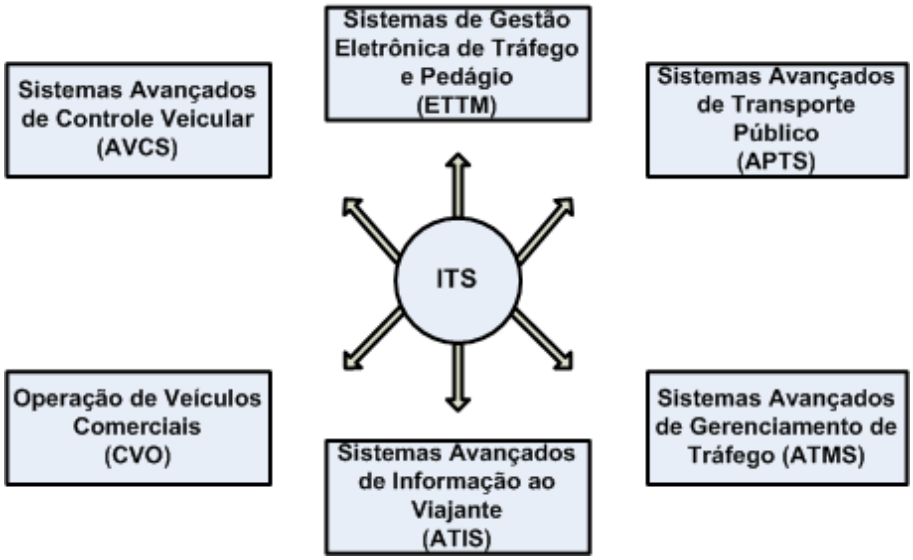
a) *Sistemas Avançados de Transporte Público*: os sistemas avançados de transporte público, ou em inglês *Advanced Public Transportation Systems* (APTS), têm como objetivo melhorar a segurança e a efetividade dos sistemas de transporte público. Os benefícios para os usuários incluem a redução dos tempos de espera, a segurança e a facilidade para o pagamento da tarifa, bem como informações precisas e atualizadas sobre itinerários e horários das linhas de ônibus;
b) *Sistemas Avançados de Gerenciamento de Tráfego*: os sistemas avançados de gerenciamento de tráfego, ou em inglês *Advanced Transportation Management Systems* (ATMS), têm como objetivo reduzir congestionamentos nas vias urbanas e/ou rurais, garantindo segurança por meio de controle e monitoramento de semáforos e utilização de câmeras de vídeo;
c) *Sistemas Avançados de Informação ao Viajante*: os sistemas avançados de gerenciamento de tráfego, ou em inglês *Advanced Traveler Information Systems* (ATIS), têm como objetivo prover informações ao viajante sobre a via, condições ambientais e trânsito. Fazem uso de sistemas de navegação e informação para garantir segurança ao motorista e para minimizar os congestionamentos;
d) *Operação de Veículos Comerciais*: os sistemas de operação de veículos comerciais, ou em inglês *Commercial Vehicle Operations* (CVO), têm como objetivo principal gerenciar a operação de veículos comerciais. Utilizam tecnologias para melhorar a gerência e o serviço de transporte de cargas com intuito de minimizar as interferências com relação às rotas e tempos perdidos, procurando garantir um alto nível de segurança.

e) *Sistemas Avançados de Controle Veicular*: os sistemas avançados de controle veicular, ou em inglês *Advanced Vehicle Control Systems* (AVCS), têm como objetivo melhorar a segurança viária, permitindo que os veículos auxiliem os motoristas. Para isto, os veículos são equipados com tecnologias que possibilitam ao condutor monitorar as condições de dirigibilidade e tomar medidas necessárias para evitar acidentes.

f) *Gestão Eletrônica de Tráfego e Pedágio*: os sistemas de gestão eletrônica de tráfego e pedágio, ou em inglês *Eletronic Toll and Traffic Management* (ETTM), têm como objetivo prover métodos eficientes para cobrança de pedágio com a finalidade de minimizar o tempo perdido no processo de cobrança e com isto reduzir congestionamentos.

Segundo Sussman (2000), os ITS empregam tecnologias de informação e comunicação aplicadas: (*i*) à melhoria do gerenciamento e da informação e comunicação, (*ii*) à melhoria do gerenciamento e da operação dos sistemas de transportes, (*iii*) à melhoria da eficiência no uso das vias, (*iv*) à melhoria da segurança viária, (*v*) ao aumento da mobilidade e (*vi*) à redução dos custos sociais, principalmente por meio da redução de tempos de espera. A Figura 1 apresenta as categorias que compõe um sistema ITS.

Figura 1 – Categorias dos sistemas ITS.



Fonte: Sussman (2000), adaptado pelo autor.

Salienta-se que o ponto de interesse deste trabalho está na utilização de tecnologias avançadas aplicadas ao transporte público, buscando integrar tecnologias da informação e comunicação necessárias à construção de um sistema capaz de detalhar a rota de viagem completa ao usuário de transporte coletivo. Dessa forma, na próxima seção serão abordados os principais conceitos envolvendo os APTS, suas funções e principais tecnologias utilizadas. Além disto, serão apresentados alguns exemplos de aplicações de APTS no Brasil e no mundo.

2.1 Sistemas Avançados de Transporte Público

Os APTS podem ser descritos como um conjunto de tecnologias que aumentam a eficiência e a segurança dos sistemas de transporte coletivo e proporcionam grande acesso à informação das operações do sistema (FTA, 2006). Segundo Chowdhury e Sadek (2003), os APTS buscam melhorar a eficiência, a produtividade e a segurança no sistema de transporte. O emprego de APTS permite também melhorar o nível de satisfação dos usuários e as condições de trabalho do motorista.

A utilização de tecnologias avançadas em transporte público, mais diretamente o transporte de ônibus, está bastante relacionada à melhoria da qualidade do serviço, como forma de proporcionar mais satisfação aos usuários do sistema. Equipamentos eletrônicos instalados nos veículos podem ajudar no controle das viagens, atuando como regulador de velocidade, aceleração, abertura e fechamento de portas, mudança de marchas, etc. Os sistemas que trabalham em tempo real possibilitam também o serviço de informação ao passageiro com precisão, possibilitando a redução os tempos de espera nos pontos de paradas e com isto melhorando a qualidade do serviço. Os sistemas automáticos de arrecadação tarifária reduzem os tempos de embarque e proporcionam mais segurança ao usuário, pois eliminam a necessidade de transações monetárias dentro dos veículos.

Conforme Chowdhury e Sadek (2003), os sistemas APTS facilitam o gerenciamento do transporte público por meio de um sistema de localização automática de veículos, ou em inglês *Automatic Vehicle Location* (AVL). Segundo o autor, o AVL possibilita a operação em tempo real, localizando o veículo no tempo e no espaço.

De maneira geral, os APTS são divididos em três categorias: Sistema de Ajuda à Operação (SAO), Sistemas de Informação ao Usuário (SIU) e Sistemas Automatizados de Arrecadação Tarifária (SAAT). Os conceitos básicos envolvendo estas categorias serão descritos a seguir.

2.1.1 Sistemas de Ajuda à Operação

Os SAOs auxiliam nas tarefas de gerência de redes de ônibus através da automação dos sistemas de transporte público. O conhecimento permanente da localização dos veículos e o controle da frota de viagem, através da identificação dos motivos de atrasos, adiantamentos ou falhas e também a comunicação em tempo real, permite uma atuação imediata na solução

dos problemas. A utilização de sistemas de ajuda à operação objetiva melhorar a produção relacionada ao modelo de rede, organização da programação de horários, monitoração das operações e gerência das informações, acarretando benefícios para a central de controle, motoristas e usuários.

Para realização do controle operacional é necessária a definição de uma rede de informações, sendo fundamental o conhecimento de quem as recebe e da forma como recebe. Segundo Drane e Rizos (1998), sistemas AVL têm como principais funções a informação ao passageiro, o gerenciamento do tráfego e a informação ao controlador do sistema, onde cada um requer diferentes especificações técnicas, especialmente com relação à precisão.

Um SAO é constituído basicamente de uma central de operações para controle e armazenamento dos dados, de sistemas de comunicação para coleta e transmissão dos dados e sistemas AVL. É realizado um pré-tratamento dos dados no veículo e um tratamento final na central de operações e de controle, de forma a assegurar a localização permanente do ônibus. É possível coletar os dados da localização do veículo na via e, além disto, obter dados referentes à velocidade, aceleração, tempo de parada, rotação do motor, número de passageiros, por trecho e horário.

As principais funções de um SAO, segundo Chowdhury e Sadek (2003), são:

a) *Garantia de comunicação*: garantir comunicação eficaz e possibilitar a gerência entre as unidades móveis (nos veículos) e a central de operação e controle;
b) *Dados de tempo de percurso:* obter dados de tempo de percurso em quantidade suficiente para avaliar os horários realizados e calibrar a tabela de horários. A análise minuciosa e detalhada dos dados de progressão do veículo na rede permite conhecer os tempos de percurso e os pontos críticos encontrados na linha, bem como os tempos médios de paradas (pontos de embarque/desembarque e cruzamentos) e de aceleração e desaceleração. O conhecimento da localização do veículo na rede pode ser obtido por meio de sistemas AVL;
c) *Autoregulação:* o serviço pode ser regulado pelo motorista do ônibus ou pelo operador da central de controle, comparando o que está sendo realizado com o que foi previamente planejado, podendo adiantar ou retardar uma viagem, alterar horários ou itinerários de modo individual conforme a necessidade;
d) *Regulação da linha:* obter os dados necessários para elaboração das ordens de regulação da linha. O conhecimento da posição relativa dos diferentes ônibus de uma linha e a possibilidade de comunicação entre a central de controle e motorista permite a atuação do controlador central a fim de modificar o serviço para minimizar possíveis perturbações para os passageiros;
e) *Regulação da rede*: o conhecimento a respeito da posição do ônibus na rede permite ao controlador central atuar imediatamente para modificar o serviço durante o

percurso com a inclusão de ônibus reserva, minimizando possíveis perturbações aos passageiros;

f) *Prioridade nas interseções semaforizadas:* garantir a prioridade nas interseções semaforizadas, de forma a assegurar o equilíbrio do sistema. O pedido de prioridade pode ser acionado toda vez que o ônibus se aproxima de um semáforo, modificando os tempos atuais (prolongando o sinal verde ou reduzindo o vermelho). Quando a regulação semafórica está associada a um SAO é possível verificar a progressão do ônibus e, em função disto, atuar ou não na priorização. Para que isto seja possível é necessário que exista comunicação entre a central de controle do SAO e a central de controle do tráfego responsável pelos planos semafóricos;

g) *Informação ao usuário:* o tratamento realizado sobre os dados permite a divulgação da informação ao passageiro, nas paradas ou em pontos específicos, sobre a progressão do ônibus na rede, reduzindo assim os tempos de espera nas paradas e a ansiedade dos passageiros;

h) *Elaboração do preço da tarifa:* fornecer os dados necessários para a elaboração dos custos elementares de produção, segundo os horários, localização, tipo de veículo, etc;

i) *Suporte na troca de dados:* garantir sistemas de transmissão de dados entre os elementos do sistema de maneira a assegurar confiabilidade dos dados, segurança e rapidez nas operações (ônibus, central e paradas), tanto para os operadores quanto para gestores.

2.1.2 Sistemas de Informação ao Usuário

Os SIUs permitem obter um conjunto de informações relativas a uma rede correspondendo a uma necessidade específica como tempo de espera, por exemplo, ou personalizada como o itinerário. Baseados em tecnologias avançadas de comunicação e transmissão de dados, os SIUs asseguram aumento da qualidade do serviço oferecido aos passageiros. Equipamentos eletrônicos instalados nos ônibus e nas vias permitem ao usuário obter informações a respeito dos horários, dos tempos de viagem e de espera e também sobre os itinerários dos ônibus. O processo de informação pode ser realizado nas residências, locais de trabalho, paradas, terminais, a bordo dos veículos ou em centros comerciais.

O foco do SIU está em melhorar a qualidade de serviço oferecido ao passageiro, informando horários, rotas em tempo real, reduzindo tempos de espera e, desta forma, atraindo mais usuários para a modalidade ônibus. Os tempos de transbordo, incluindo caminhada e espera, representam muitas vezes uma parcela significativa do tempo total da viagem. Segundo Chowdhury e Sadek (2003), o valor do tempo de transbordo é considerado pelos usuários como quatro vezes superior ao valor atribuído em relação ao tempo de viagem no veículo.

Para disponibilizar informações com um alto nível de precisão, os SIU fazem uso de tecnologias como telefone celular, monitores, painéis eletrônicos e computadores. Proporcionar informações quanto a horários, linhas e percursos do ônibus é um serviço de elevada importância para garantir um nível de qualidade aceitável requerido pelos usuários do transporte público urbano. A disponibilização da informação promove o transporte público, pois permite que as pessoas façam o planejamento de seus deslocamentos (Drane e Rizos, 1998).

As principais funções de um SIU, segundo Chowdhury e Sadek (2003), são:

a) *Promocional:* propor motivos para viagens e possíveis destinos, informar as pessoas sobre o transporte público como parte do pacote de facilidades ofertadas e melhorar a imagem do transporte público;
b) *Instrução:* informar os usuários a respeito da utilização do transporte público e divulgar as regras envolvidas no uso dos sistemas;
c) *Operacional:* informar sobre restrições e oportunidades associadas com o uso do sistema para diferentes tipos de viagens, capacitar pessoas para acesso à rede de transporte público, capacitar à realização de uma viagem, informar sobre mudanças na programação;
d) *Moderação:* aliviar a ansiedade do viajante, aumentar o controle do usuário sobre a escolha entre as opções disponíveis.

Ressalta-se que o ponto de interesse deste trabalho está na modelagem e construção de um sistema de informação ao usuário capaz de detalhar a rota de viagem completa de ônibus.

2.1.3 Sistemas Automatizados de Arrecadação Tarifária

Os SAATs automatizam o processo de cobrança tarifária no transporte coletivo urbano por ônibus e, desta forma, possuem grande importância na otimização da operação. A eliminação da transação monetária no interior dos veículos gerada pela implantação de um SAAT traz como principais benefícios a agilidade na cobrança das passagens, redução do tempo gasto no embarque e aumento na segurança para os usuários e tripulação. Além disso, a aplicação do sistema na integração temporal, em que o portador de um cartão inteligente poderá utilizar mais de um ônibus ou diferentes modos pagando uma tarifa única, dentro de um prazo preestabelecido, facilita o transporte e reduz o custo das viagens (Drane e Rizos, 1998).

As principais funções de um SAAT, segundo Chowdhury e Sadek (2003), são:

a) Aprimorar o conhecimento quantitativo da demanda efetiva, coletando dados sobre os pontos e horários de provisão;
b) Melhorar a oferta e demanda, em função do melhor conhecimento do carregamento, coletando dados referentes aos pontos de embarque e desembarque, data e hora, com detalhamento por nível de perfil de viagem (estudante, vale transporte, isento, etc.);
c) Permitir um controle mais eficaz das passagens evitando fraudes;
d) Adequar às políticas tarifárias, podendo definir os preços segundo a distância percorrida, horário, dia e perfil do usuário;
e) Criar uma estrutura de transportes que permita um sistema de tarifa multimodal e multisserviço.

2.1.4 Sistemas APTS no Brasil

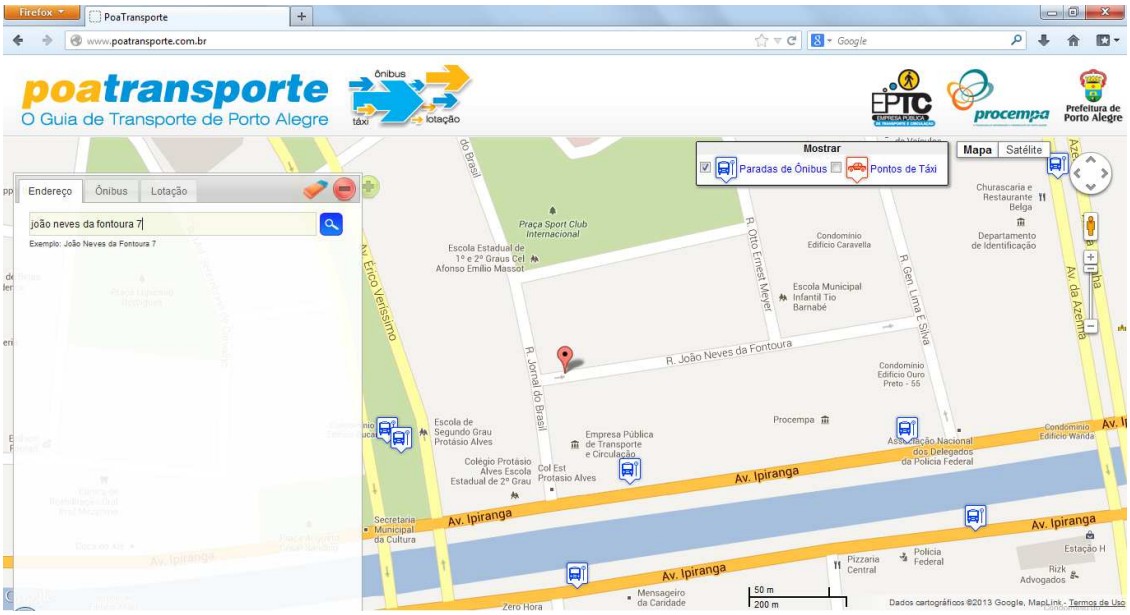
Atualmente, diversas cidades brasileiras estão aplicando tecnologias avançadas no transporte coletivo, por meio de sistemas APTS. Serão apresentadas a seguir, as experiências brasileiras mais conhecidas nas cidades de Porto Alegre, Curitiba e São Paulo.

Porto Alegre – por meio da EPTC, Porto Alegre dispõe de um sistema SAAT, através do transporte integrado (TRI). O TRI consiste em um sistema de bilhetagem eletrônica que é responsável pela arrecadação automática da passagem de ônibus, substituindo as carteiras de identificação e as fichas por cartões com créditos eletrônicos. O sistema tem como objetivo integrar itinerários e beneficiar usuários com isenção de tarifa de mais de uma linha para o mesmo trajeto, a chamada passagem integrada, na qual o usuário paga a primeira passagem inteira e recebe isenção na segunda. Para usufruir desta integração, o usuário tem trinta minutos, após o desembarque do primeiro ônibus, para embarcar em um segundo ônibus. O cálculo do tempo para realização da integração é realizado com base no tempo de trajeto de cada linha de ônibus, garantindo ao usuário, que independentemente de onde este desembarcar, ele tenha os trinta minutos garantidos para embarcar no segundo ônibus (PORTALTRANSPARENCIA, 2012).

Além disto, desde julho de 2011, já é possível a realização de uma integração entre os modais ônibus-metrô nas cidades de São Leopoldo, Canoas e Porto Alegre. Através do cartão TRI ou através do cartão SIM, da Empresa de Trens Urbanos de Porto Alegre (TRENSURB), os usuários podem realizar viagens combinando trajetos de ônibus e metrô. Da mesma forma que a integração do TRI para ônibus, esta integração é válida mediante um intervalo de trinta minutos entre cada um dos trechos da viagem (TRENSURB, 2012).

A cidade dispõe também de um SIU chamado Poatransporte (2012), criado pela EPTC em parceria com a prefeitura de Porto Alegre. Neste sistema, é possível visualizar rotas de linhas de ônibus oferecidas na cidade de Porto Alegre. A partir de um endereço, é possível conhecer as paradas próximas e também as lotações e linhas de ônibus que passam pelas proximidades do local. A Figura 2 ilustra a interface do sistema. No lado esquerdo da figura é ilustrado um formulário com três abas, que permitem três diferentes formas de pesquisa: (i) Endereço, (ii) Ônibus e (iii) Lotação. A pesquisa por endereço permite que seja informado um endereço de destino para o qual se deseja visualizar as opções de transporte público. O resultado da pesquisa é exibido diretamente no mapa. Como esse sistema considera apenas o endereço desejado, não é possível visualizar um trajeto completo entre origem e destino, considerando possíveis transbordos que porventura sejam necessários.

Figura 2 – Sistema Poatransporte em Porto Alegre.



Fonte: Poatransporte (2012), adaptado pelo autor.

Curitiba – a cidade de Curitiba foi uma das primeiras na implantação dos sistemas de transporte rápido por ônibus (BRT), ou em inglês *Bus Rapid Transit*, ainda na década de setenta. Desde então, a capital do Paraná se tornou pioneira e também disseminadora desses serviços de ônibus rápido pelo mundo. O BRT é um sistema de transporte coletivo de passageiros que permite uma mobilidade urbana rápida, confortável, eficiente e segura por meio de infraestrutura segregada com prioridade de ultrapassagem, operação rápida e frequente, com excelência em serviços ao usuário (BRASIL, 2008).

Os sistemas BRT em Curitiba operam com ônibus em faixas exclusivas e segregadas do restante do tráfego, têm prioridade em cruzamentos, são monitorados em tempo real por dispositivos tecnológicos de rastreamento, além de utilizar sistemas de bilhetagem eletrônica. Os pontos de parada ou terminais de integração são projetados para facilitar as operações de embarque e desembarque de passageiros e os veículos têm maior capacidade de transporte. O resultado da adoção desse modelo evidenciou um conjunto de benefícios que merecem destaque: reduz o tempo das viagens; reduz o tempo de espera dos usuários nos pontos de parada; aumenta a confiabilidade no sistema, pois o usuário sabe exatamente em quanto tempo estará disponível o próximo ônibus através de painéis de mensagens variáveis; reduz a variabilidade do tempo total de viagem, pois o usuário possui informação sobre o tempo de duração de um percurso; proporciona mais conforto por utilizar veículos dotados de tecnologia de última geração, contribui para a redução dos impactos ambientais, pois operam com velocidade média constante, reduzindo assim o consumo de combustível e emissões atmosféricas (ITS, 2012). A Figura 3 ilustra a imagem de um SAO a disposição do condutor do veículo onde é possível transmitir informações a respeito de situações de emergência que podem ocorrer durante a viagem.

Figura 3 – Sistema BRT em Curitiba.

Fonte: Belem (2012), adaptado pelo autor.

São Paulo – o sistema municipal de transporte da cidade de São Paulo é composto por uma rede integrada, criada pela Secretaria Municipal de Transportes em parceria com a empresa São Paulo Transporte (SPTrans). Desde 1995, conta com sistemas de coleta automática de dados operacionais do sistema de transporte coletivo urbano, a fiscalização eletrônica. Em 1998, implantou um sistema de cobrança automática de tarifa com a instalação de validadores eletrônicos nos ônibus. Em 2004, realizou a implantação do bilhete único para estudantes e trabalhadores, sendo que em 2006, implantou a integração com os sistemas de metrô e trem. Esta integração possibilitou ao usuário a realização de três viagens no sistema através de ônibus e mais uma viagem através do trem (SPTRANS, 2012).

Desde 2011, os usuários de transporte público da cidade de São Paulo dispõem de um SIU chamado Olhovivo (2012). O sistema mostra em tempo real onde estão os ônibus e também o tempo estimado para a chegada deles até um determinado ponto ou estação. O sistema faz uso de dispositivos que operam segundo o padrão de sistemas de posicionamento global, ou em inglês *Global Positioning System* (GPS), instalados nos ônibus para transmitir as informações ao sistema. Conforme ilustrado na Figura 4, o sistema disponibiliza ao usuário três opções de consulta, denominados localizadores. Os objetivos de cada localizador são:

a) *De olho na linha*: permite consultar informações a respeito do trajeto das linhas e fornece informações sobre a localização dos ônibus (localização dos ônibus);
b) *De olho no ponto*: permite consultar informações a respeito do tempo e de quais linhas de ônibus se aproximam do ponto do usuário (próximos ônibus);
c) *De olho na via*: permite consultar informações a respeito do desempenho dos principais corredores viários da cidade (velocidade média e tempo de percurso).

Figura 4 – Sistema Olho Vivo em São Paulo.

Fonte: Olhovivo (2012), adaptado pelo autor.

2.1.5 Sistemas APTS no Mundo

Segundo Schein (2003), existe uma inter-relação sinérgica entre o sistema de auxílio da operação e o sistema de informação ao usuário. Para que um SIU possa disponibilizar informações confiáveis aos usuários, relativas a horários, pontos de parada, locais de integração, entre outros, é necessário que exista uma regularidade na prestação de serviço, que neste caso deve ser controlada por um SAO.

Na Europa e Estados Unidos, a utilização de sistemas de auxílio à operação e sistemas de informação ao usuário tiveram início na década de 80, quando a França implantou os primeiros sistemas avançados aplicados ao transporte coletivo urbano. A evolução dos sistemas permite atualmente o acompanhamento da frota e também o controle sobre a operação (SILVA, 2000). A Tabela 1 demonstra alguns exemplos de sistemas APTS implantados na Europa e Estados Unidos.

Tabela 1 - Sistemas APTS implantados pelo mundo.

Localidade	Implantação	Tipo de Sistema	Descrição
Madrid	-	SAO, SIU	Caminho ótimo
Barcelona	-	SAO, SIU	Localização contínua com transmissão on-line
Paris	1991	SAO, SIU	Localização contínua on-line
Holanda	1992	SAO, SIU	Tabela de horários e algoritmos de busca
Londres	1993	SIU	Localização discreta com transmissão em tempo real
Southampton	1994	SIU	Localização contínua em tempo real
Minneapolis	1994	SIU	Localização contínua
Los Angeles	1995	SIU	Localização contínua com transmissão on-line
Chicago	1996	SAO, SIU	Localização contínua com transmissão on-line
Orlando	1996	SAO, SIU, SAAT	Localização automática de veículos - AVL
Metz	1999	SAO, SIU	Localização discreta com radio transmissão em tempo real
Cleveland	2003	SAO, SIU, SAAT	Localização automática de veículos - AVL

Fonte: Silva 2000; Meyer 2000; Amundsen 2001 (apud SCHEIN, 2003).

De acordo com a tabela, existe uma predominância de sistemas SIU e SAO, com foco principal na localização dos veículos. Embora existam também sistemas SAAT nas cidades de Orlando e Cleveland, de um modo geral as soluções são restritas apenas a algumas cidades. Contudo, para que seja possível promover, de fato, a mobilidade urbana, as soluções precisam ser aplicadas em um âmbito global.

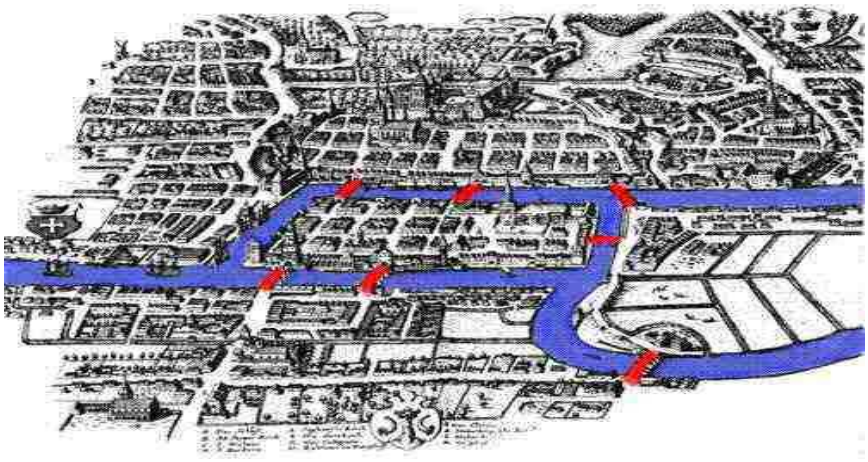
3 TEORIA DOS GRAFOS

Existem várias situações nas quais é importante representar o inter-relacionamento entre um conjunto finito de objetos. Os grafos são estruturas de dados presentes na ciência da computação que permitem representar centenas de problemas computacionais relevantes. Por esta razão, muitas áreas do conhecimento como matemática, telecomunicações, engenharia civil, gestão, entre outros, recorrem à teoria dos grafos para modelar e resolver problemas (GOODRICH; TAMASSIA, 2007). A modelagem através de grafos pode estar associada, por exemplo, a problemas que envolvem:

a) Determinação de caminhos e rotas;
b) Redes de comunicação;
c) Localização de facilidades (depósitos, hospitais, escolas, etc);
d) Desenhos de circuitos impressos;
e) Logística de distribuição de produtos;
f) Telecomunicações;
g) Comunicação móvel por rádio frequência;
h) Limpeza urbana.

A origem da teoria dos grafos remete ao século XVIII, pois é geralmente associada ao problema das pontes de Königsberg, território da antiga Prússia. Parte desta cidade localizava-se em duas ilhas do Rio Pregel, as quais estavam ligadas, às margens e uma a outra, através de sete pontes, conforme ilustrado na Figura 5. Os habitantes da cidade desejavam passar pelas sete pontes numa caminhada contínua sem passar duas vezes por qualquer uma das pontes. Em 1736, o matemático suíço Leonhard Euler (1707-1783) analisou o problema trocando as áreas de terra por pontos (vértices) e as pontes por arcos (arestas). Ele provou que o problema não tinha solução, dando início assim ao que chamamos hoje de *Teoria dos Grafos*.

Figura 5 – Sete pontes de Königsberg.

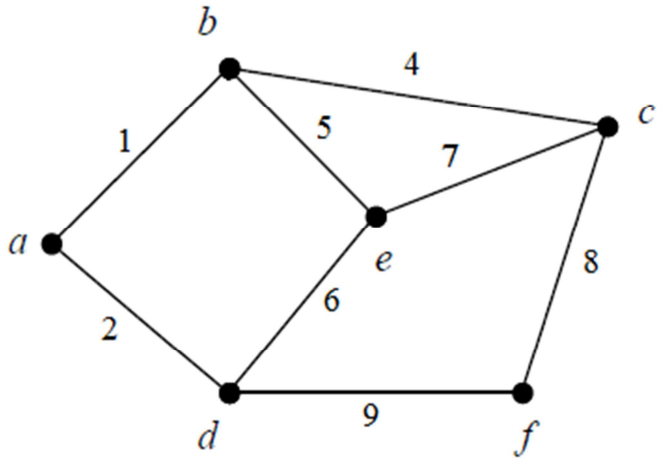


Fonte: Dmat (2012), adaptado pelo autor.

3.1 Definições e Notações Básicas

Um grafo é uma estrutura de abstração que representa um conjunto de elementos denominados vértices e suas relações de interdependência ou arestas. Um grafo pode ser representado matematicamente por $G = (V, E)$, onde V denota o conjunto de vértices da estrutura, e E o conjunto das arestas ou ligações entre os vértices (GOODRICH; TAMASSIA, 2007). A Figura 6 ilustra um exemplo de um grafo.

Figura 6 – Representação de um grafo.



Fonte: Elaborado pelo autor.

Neste exemplo, o conjunto de vértices V é formado pelos vértices (a, b, c, d, e, f), enquanto o conjunto de arestas E é formado pelas arestas (1, 2, 4, 5, 6, 7, 8, 9). Os termos nós,

nodos e vértices são usualmente sinônimos, assim como arcos e arestas. Em uma situação real, o grafo ilustrado na Figura 6 pode representar um mapa rodoviário, onde os vértices são as cidades e as arestas são estradas ou vias que ligam estas cidades. Os valores ou pesos de cada aresta poderiam representar a distância entre as respectivas cidades.

As arestas de um grafo são representadas por um par *(i, j)*, com $i \neq j$ e $i, j \in V$, em que *i* e *j* são vértices finais da aresta *(i, j)*. Uma aresta é dita dirigida (orientada) se for representada por um par ordenado de vértices distintos e não dirigida (não orientada) se for representada por um par não ordenado de vértices distintos. Um arco dirigido *(i, j)* pode ser visto como uma rua de um só sentido, que permite fluxo apenas de *i* para *j*, enquanto um arco não dirigido *(i, j)* pode ser visto como uma rua de dois sentidos, que permite fluxo em ambas as direções (de *i* para *j* e de *j* para *i*) (NETTO, 1996).

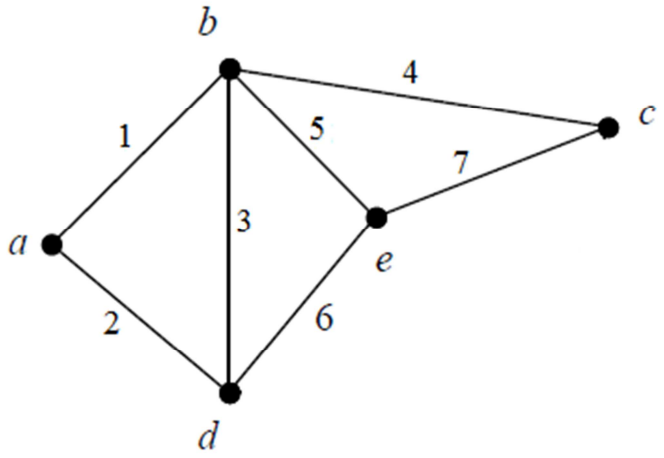
Baseado nestas definições, os grafos podem ser separados em três tipos:

a) Grafos dirigidos (ou orientados): todas as arestas são dirigidas;
b) Grafos não dirigidos (não orientados): todas as arestas são não dirigidas;
c) Grafos mistos: algumas arestas são dirigidas e outras são não dirigidas.

Em um grafo, dois vértices conectados por uma aresta são chamados de vértices finais ou pontos finais da aresta. Se uma aresta é dirigida, seu primeiro ponto final é sua origem e o outro é o seu destino. Dois vértices são ditos adjacentes se forem pontos finais da mesma aresta. Uma aresta é dita incidente a um vértice se o vértice for um dos pontos finais desta aresta. As arestas incidentes de um vértice são arestas dirigidas cujo destino é aquele vértice. O grau de um vértice *v* em um grafo não dirigido, denotado *deg(v)*, é o número de vértices incidentes a *v*. O grau de entrada e o grau de saída de um vértice *v* em um grafo dirigido são os números de arestas incidentes em *v* e de *v*, respectivamente, e são denotados *indeg(v)* e *outdeg(v)* (GOODRICH; TAMASSIA, 2007).

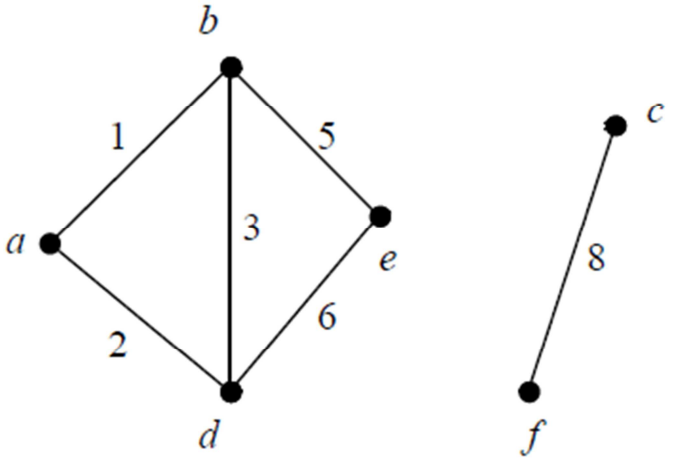
Um grafo pode ser categorizado também em função da conectividade de seus vértices. O grafo *G = (V, E)* é dito ser conexo quando existe pelo menos um caminho ligando cada par de vértices deste grafo, enquanto é dito ser desconexo quando existe pelo menos um par de vértices que não está ligado por nenhum caminho. A Figura 7 ilustra um exemplo de grafo conexo e a Figura 8 ilustra um exemplo de um grafo desconexo.

Figura 7 – Exemplo de grafo conexo.



Fonte: Elaborado pelo autor.

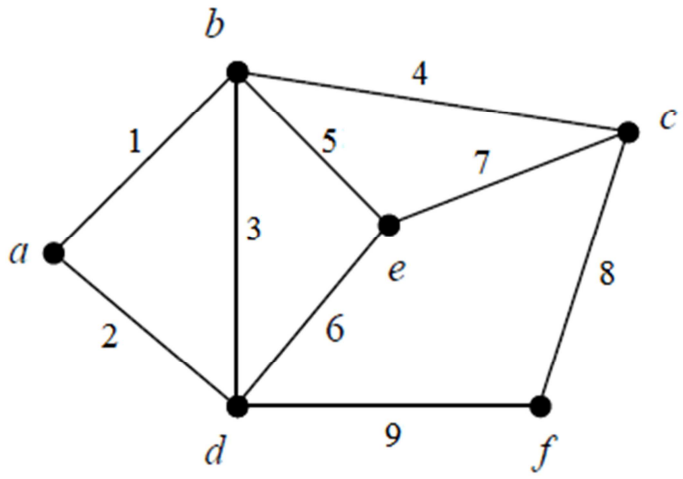
Figura 8 – Exemplo de grafo desconexo.



Fonte: Elaborado pelo autor.

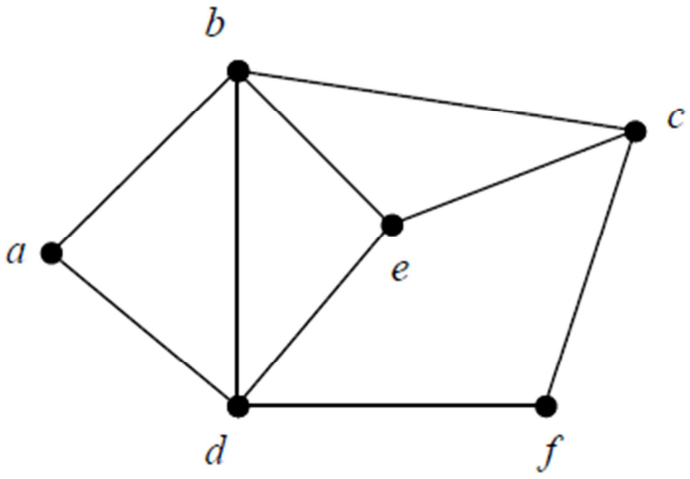
Um grafo *G = (V, E)* é dito ponderado quando existem valores numéricos associados as suas arestas ou mesmo vértices, enquanto é dito rotulado quando existem atribuições associadas aos seus vértices (numéricas ou alfabéticas). A Figura 9 demonstra um exemplo de um grafo ponderado e a Figura 10 demonstra um exemplo de um grafo rotulado.

Figura 9 – Exemplo de um grafo ponderado.



Fonte: Elaborado pelo autor.

Figura 10 – Exemplo de um grafo rotulado.



Fonte: Elaborado pelo autor.

Um caminho em um grafo é uma sequência alternada de vértices e arestas que inicia e termina em um vértice, de modo que cada aresta seja incidente de seu antecessor e incidente em seu sucessor. Um ciclo é um caminho em que os vértices de início e fim são os mesmos. Um caminho é dito simples se cada vértice do caminho for distinto, enquanto um ciclo simples é um caminho em que cada vértice do ciclo é distinto, com exceção do primeiro e último. Um caminho dirigido é um caminho em que todas as arestas são dirigidas e percorridas em sua direção (GOODRICH; TAMASSIA, 2007).

3.2 Operações em Grafos

A manipulação e a representação dos dados relacionados a um grafo, assim como para outros tipos abstratos de dados, envolve um conjunto específico de operações. Em uma linguagem orientada a objetos, estas operações poderiam ser tratadas como métodos de uma classe. Um grafo é uma coleção de elementos que são armazenados em determinadas *posições* do grafo, neste caso seus vértices e arestas (GOODRICH; TAMASSIA, 2007). Portanto, podem-se armazenar informações sobre o grafo tanto em seus vértices quanto em suas arestas, ou mesmo em ambos.

Segundo Goodrich e Tamassia (2007), as operações básicas envolvendo grafos são:

a) *vertices ( ):* devolve uma coleção de todos os vértices do grafo;
b) *edges ( ):* devolve uma coleção de todas as arestas do grafo;
c) *incidentEdges (v):* devolve uma coleção de arestas incidentes no vértice *v*;
d) *opposite (v, e):* devolve o vértice final da aresta *e* separado do vértice *v*; um erro ocorre se *e* não é incidente a *v*;
e) *(e):* devolve um arranjo armazenando os vértices finais da aresta *e*;
f) *areAdjacent (v, w):* retorna verdadeiro se os vértices *v* e *w* são adjacentes;
g) *replace (v, x):* troca o elemento armazenado no vértice *v* com *x*;
h) *replace (e, x):* troca o elemento armazenado na aresta *e* com *x*;
i) *insertVertex (x):* insere um novo vértice com o elemento *x*;
j) *insertEdge (v, w, x):* insere uma nova aresta não dirigida com vértices finais *v* e *w* com o elemento *x*;
k) *removeVertex (v):* exclui o vértice *v* e todas as arestas incidentes nele e devolve *v*;
l) *removeEdge (e):* exclui a aresta *e* e retorna o elemento *e*.

O custo de cada uma destas operações é diferente de acordo com a representação utilizada para o grafo. De acordo com as características específicas do problema, estas operações podem ser mais ou menos frequentes. Em função disto, o desempenho do algoritmo está diretamente relacionado à escolha da representação computacional que deve ser utilizada.

3.3 Representação Computacional

A eficiência computacional de um algoritmo para resolver problemas envolvendo grafos não depende apenas das suas características intrínsecas, mas também das estruturas de dados utilizadas para representar o grafo no computador (formas de armazenar e manipular os dados associados ao grafo) e para armazenar os resultados necessários ao algoritmo (NETTO,

1996). Geralmente, para representar um grafo no computador são necessários dois tipos de informação:

a) A topologia do grafo (estrutura dos vértices e das arestas);
b) Os dados (custos, capacidades, distâncias) associados aos vértices e as arestas;

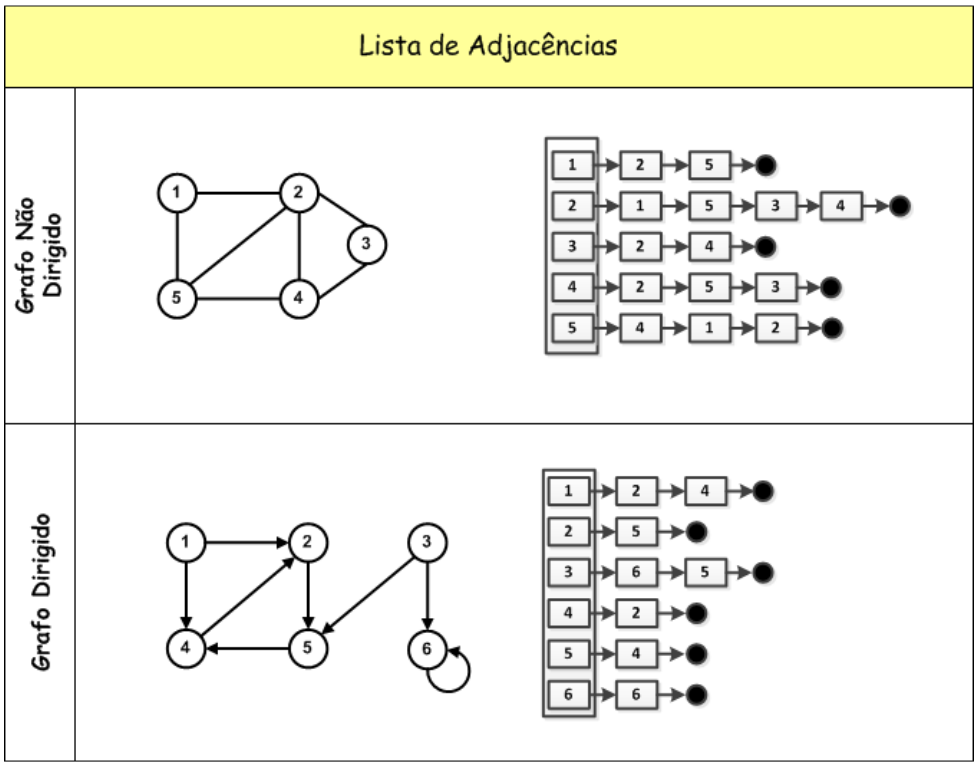
Normalmente, o esquema utilizado para armazenar a topologia do grafo irá sugerir, naturalmente, uma forma de armazenar a informação associada às arestas e aos vértices.

Segundo Cormen et. al (2002), as representações mais comuns de um grafo são através de listas de adjacências ou matriz de adjacências. Listas de adjacências são, em geral, mais utilizadas porque fornecem um meio compacto de representar grafos esparsos – aqueles para os quais $|E|$ é muito menor que $|V|^2$, onde $|E|$ representa o número de arestas do grafo e $|V|$ representa o número de vértices do grafo. Matrizes de adjacências são mais utilizadas quando o grafo for denso – $|E|$ está próximo de $|V|^2$ – ou quando é necessário saber com rapidez se existe uma aresta conectando dois vértices.

3.3.1 Lista de Adjacências

A representação de um grafo $G = (V, E)$ através de lista de adjacências consiste num arranjo Adj de $|V|$ listas, uma para cada vértice em V. Para cada $u \in V$, a lista de adjacências $Adj[u]$ contém todos os vértices v tais que existe uma aresta $(u, v) \in E$, ou seja, o arranjo $Adj[u]$ consiste em todos os vértices adjacentes a u em G (CORMEN et al., 2002). A Figura 11 ilustra um exemplo de representação através de listas de adjacências para dois grafos não ponderados: um dirigido e um não dirigido.

Figura 11 – Representação através de lista de adjacências.



Fonte: Cormen et al. (2002), adaptado pelo autor.

As listas de adjacências podem ser adaptadas para representar também grafos ponderados, onde as arestas tem um peso associado. Se $G = (V, E)$ é ponderado com função peso de aresta w, o peso $w(u, v)$ da aresta $(u, v) \in E$ pode ser simplesmente armazenado com o vértice v na lista de adjacências de u. A representação de listas de adjacências é bastante robusta e flexível neste aspecto, podendo ser modificada para trabalhar com outras variantes de grafos.

A Tabela 2 ilustra o desempenho da implementação das principais operações do grafo com listas de adjacências assumindo que as coleções V e E de vértices incidentes são todas implementadas com listas duplamente encadeadas. Para um vértice v, o espaço usado pela coleção de incidentes de v é proporcional ao grau de v, ou seja, ele é $O(deg(v))$. Neste caso, o espaço requerido pela lista de adjacência é $O(|V| + |E|)$.

Tabela 2 - Tempos de execução dos métodos através de lista de adjacências.

Operação	Complexidade
vertices	O (|*V*|)
edges	O (|*E*|)
endVertices, opposite	O (1)
incidentEdges (v)	O (deg(v))
areAdjacent (v, w)	O (min(deg(v), deg(w))
replace	O (1)
insertVertex, insertEdge, removeEdge	O (1)
removeVertex	O (deg(v))

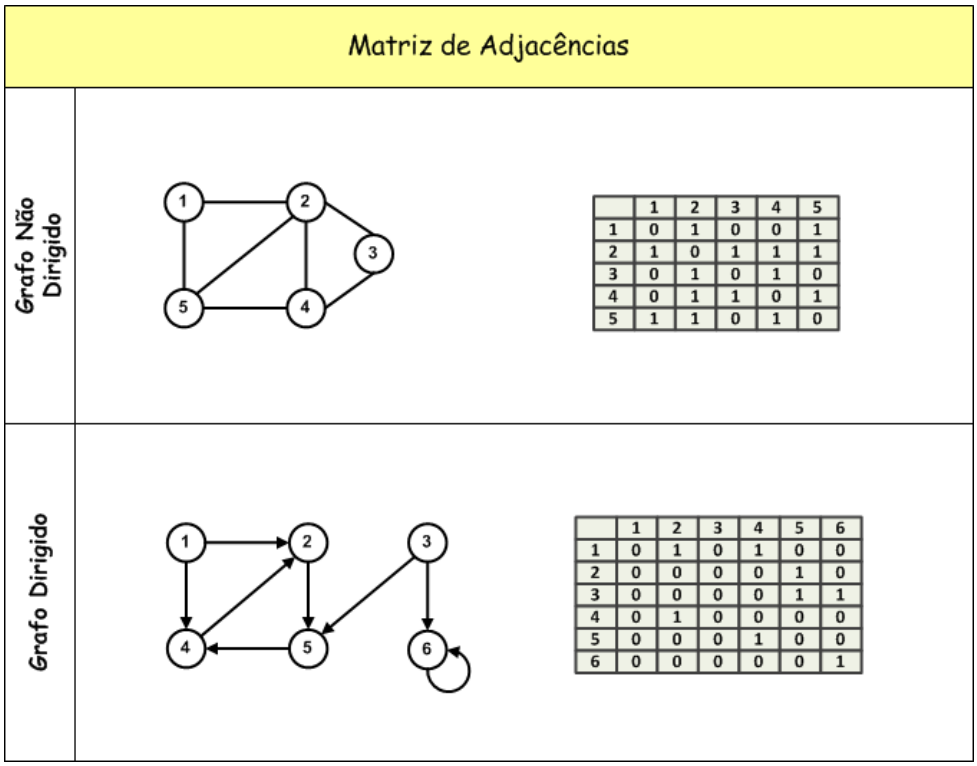
Fonte: Goodrich e Tamassia (2007), adaptado pelo autor.

3.3.2 Matriz de Adjacências

Para representar um grafo *G = (V, E)* através de uma matriz de adjacências é necessário supor que o grafo seja rotulado. A representação através de matriz de adjacências consiste em construir uma matriz quadrada de ordem |*V*|, onde cada posição da matriz *(i, j)* deverá receber como valores *0* ou *1*. Caso o vértice na posição *i* seja adjacente ao vértice na posição *j*, então a posição *(i, j)* da matriz deverá receber 1 como valor e 0 caso contrário (CORMEN et al., 2002). A Figura 12 ilustra um exemplo de representação através de uma matriz de adjacência para dois grafos não ponderados: um dirigido e um não dirigido.

A matriz de adjacências também pode ser usada para representar grafos ponderados. Se *G = (V, E)* é um grafo ponderado com função peso de aresta *w*, o peso *w(u, v)* da aresta *(u, v)* $\in$ *E* é simplesmente armazenado como a entrada na linha *u* e coluna *v* da matriz de adjacências. Caso não exista uma aresta pode ser armazenado um valor nulo, embora em muitos casos seja conveniente usar um valor zero ou ∞.

Figura 12 – Representação através de matriz de adjacência.



Fonte: Cormen et al. (2002), adaptado pelo autor.

A Tabela 3 ilustra o desempenho da implementação das principais operações do grafo com matriz de adjacências. Pode-se observar que a lista de adjacência é mais eficiente do que a matriz de adjacência em relação à complexidade de espaço e de tempo para todos as operações, exceto para o método *areAdjacent()*.

Tabela 3 - Tempos de execução dos métodos através de matriz de adjacências.

Operação	Complexidade
vertices	O ($\|V\|$)
edges	O ($\|E\|$)
endVertices, opposite, areAdjacent	O (1)
incidentEdges (v)	O ($\|V\|$ + deg(v))
replace, insertEdge, removeEdge	O (1)
insertVertex, removeVertex	O ($\|V\|^2$)

Fonte: Goodrich e Tamassia (2007), adaptado pelo autor.

3.4 Comparativo entre Representações Computacionais

A matriz de adjacência booleana foi a primeira representação usada para grafos. No entanto, este fato não é surpreendente, pois a matriz de adjacência tem um apelo natural como

estrutura matemática. A lista de adjacência surgiu posteriormente, devido à necessidade de métodos mais rápidos para a maioria das operações e também devido à sua eficiência em termos de espaço (GOODRICH; TAMASSIA, 2007).

Por outro lado, a representação por lista de adjacência possui uma desvantagem de não possuir um modo mais rápido para determinar se uma aresta *(u, v)* está presente no grafo do que procurar por *v* nas listas de adjacências *Adj[u]*. Esta desvantagem pode ser contornada utilizando uma representação por matriz de adjacências do grafo, mas utilizando assintoticamente mais memória (CORMEN et al., 2002).

Por ser recomendada para grafos esparsos, como é o caso deste trabalho, a representação que será utilizada serão as listas de adjacências. Embora o grafo seja estático, as rotas são dinâmicas (definidas pelo usuário) fazendo com que as adjacências de um vértice sejam o ponto principal na determinação do caminho mínimo que, neste caso, são naturalmente representadas por uma lista de adjacências.

4 ALGORITMOS DE BUSCA

Algoritmos de busca são uma das mais poderosas abordagens para a resolução de problemas universais. São mecanismos que exploram alternativas sistematicamente e encontram uma sequência para a solução do problema proposto. Encontrada esta sequência é necessário considerar se a solução é eficiente ou não e, além disto, considerar os recursos necessários para se encontrar esta solução em termos de tempo e memória (RUSSEL; NORVIG, 1995). A utilização de algoritmos de busca pode estar associada, por exemplo, a problemas que envolvem:

a) Planejamento de viagens;
b) Planejamento de operações militares;
c) Planejamento de limpeza urbana;
d) Planejamento de distribuição de produtos;
e) Roteamento de redes de computadores.

O foco deste trabalho é a utilização de algoritmos de busca para solução de problemas envolvendo planejamento de viagens através de transporte público, ou mais especificamente, a determinação de caminhos mínimos através de transporte público. Em função disto, nas próximas seções serão descritos os conceitos, características e propriedades relacionados ao problema do caminho mínimo, e também, os conceitos, características e diferenças entre os métodos de busca existentes: *(i)* busca cega e *(ii)* busca heurística.

4.1 Problema do Caminho Mínimo

Na teoria dos grafos, um problema de caminho mínimo envolve um grafo orientado ponderado $G = (V, E)$, com função peso w: E $\rightarrow R$, onde R relaciona arestas com pesos de valores reais. O peso do caminho $p = \{v_1, v_2, ..., v_k\}$ é o somatório dos pesos das arestas que compõem este caminho (CORMEN et al., 2002).

$$w(p) = \sum_{i=1}^{k} w\,(v_{i-1}, v_i)$$

Assim, o peso do caminho mínimo pode ser definido como:

$$\delta\,(u, v) = \begin{cases} \min\{w(p): u \rightarrow v\} \text{ se existe um caminho de } u \text{ até } v \\ \infty, \text{caso contrário} \end{cases}$$

Um caminho mínimo entre um vértice *u* e um vértice *v* é definido como qualquer caminho *p* com peso $w(p) = \delta(u, v)$. Os pesos das arestas podem representar medidas, no lugar de distâncias. Geralmente os pesos das arestas representam tempo, distância, custo, penalidades, perdas ou outra informação que se acumule linearmente ao longo do caminho a qual se deseja minimizar (CORMEN et al., 2002).

Segundo Cormen et. al (2002), existem basicamente quatro variações do problema do caminho mínimo:

a) *Problema do caminho mínimo de origem única*: em um grafo $G = (V, E)$, consiste em determinar um caminho mínimo a partir de um vértice *s* para todos os vértices $v \in V$;
b) *Problema do caminho mínimo de destino único*: consiste em determinar um caminho mínimo até um determinado vértice de destino *t* a partir de um vértice *v*. Invertendo o sentido de cada aresta do grafo, pode-se reduzir esse problema a um problema de origem única;
c) *Problema do caminho mínimo de par único*: consiste em determinar o caminho mínimo entre um vértice *u* até um vértice *v*. Ao resolver o problema de origem única com o vértice de origem *u*, este problema também é resolvido;
d) *Problema do caminho mínimo entre todos os pares*: consiste em determinar o caminho mínimo entre um vértice *u* até *v* para todo par de vértices *u* e *v*.

A propriedade de que todo caminho mínimo possui outros caminhos mínimos em seu interior é denominada subestrutura ótima (CORMEN et al., 2002). Dado um grafo orientado e ponderado $G = (V, E)$, com função peso w: E $\rightarrow R$, onde $p = \{v_1, v_2, ..., v_k\}$ é um caminho mínimo entre o vértice v_1 e o vértice v_k. Para quaisquer *i* e *j* tais que $1 \leq i \leq j \leq k$, onde $p_{ij} = \{v_i, v_{i+1}, ..., v_j\}$ é um subcaminho entre os vértices v_i e v_j, então p_{ij} também será um caminho mínimo de v_i até v_j.

Em alguns tipos de problemas envolvendo caminhos mínimos podem existir arestas com pesos negativos. Se o grafo $G = (V, E)$ não contém nenhum ciclo de peso negativo acessível a partir de uma origem *s*, então para todo $v \in V$, o peso do caminho mínimo $\delta(s, v)$ permanece bem definido, mesmo tendo um valor negativo. Entretanto, caso exista um ciclo de peso negativo acessível a partir de *s*, os pesos dos caminhos mínimos não estão bem definidos, pois sempre será possível encontrar um caminho de peso menor cada vez que o ciclo negativo for percorrido. Um ciclo de peso negativo é um caminho onde o primeiro e o último vértice são os mesmos e o custo deste caminho é negativo (CORMEN et al., 2002).

Em problemas de caminhos mínimos, uma questão recorrente é a forma de representação deste caminho. Uma representação utilizada com muita frequência é semelhante

a uma representação em árvore. Em um grafo $G = (V, E)$ mantém-se para cada vértice $v \in V$ um predecessor $\Pi [v]$ que neste caso é outro vértice ou uma referência nula, comumente representada por NIL. Os algoritmos de busca definem geralmente os atributos Π de tal forma que o caminho mínimo pode ser representado simplesmente percorrendo a cadeia de predecessores com origem em um vértice v de forma contrária e recursiva (CORMEN et al., 2002).

Denomina-se estimativa de caminho mínimo o atributo $d[v]$, que neste caso representa um limite superior sobre o peso do caminho mínimo desde a origem s até v. O processo de relaxamento de uma aresta consiste em testar se é possível melhorar o caminho mínimo para v, encontrado até o momento pela passagem de u e, neste caso, atualizar $d[v]$ e $\Pi[v]$. Uma etapa de relaxamento pode diminuir o valor da estimativa de caminho mínimo $d[v]$ e atualizar o campo predecessor de v, $\Pi[v]$ (CORMEN et al., 2002).

Conforme Cormen et. al (2002), existem basicamente seis propriedades fundamentais relacionadas a problemas de caminhos mínimos:

a) *Desigualdade triangular*: determina que para cada aresta $(u, v) \in E$, temos $\delta(s, v) \leq \delta(s, u) + w(u, v)$;
b) *Propriedade do limite superior*: determina que sempre que temos $d[v] \geq \delta (s, v)$ para todos os vértices $v \in V$ e, uma vez que $d[v]$ alcança o valor $\delta (s, v)$, ele nunca muda;
c) *Propriedade de nenhum caminho*: determina que caso não exista nenhum caminho de s para v, então sempre temos $d[v] = \delta (s, v) = \infty$;
d) *Propriedade de convergência*: determina que se $s \rightarrow u \rightarrow v$ é um caminho mínimo em G, para algum $u, v \in V$, e se $d[u] = \delta (s, u)$ em qualquer instante antes de se relaxar a aresta (u, v), então $d[v] = \delta (s, v)$ em todos os momentos posteriores;
e) *Propriedade de relaxamento do caminho*: determina que se $p = \{v_1, v_2, ..., v_k\}$ é um caminho mínimo de $s = v_1$ a v_k, e as arestas de p são relaxadas na ordem $(v_1, v_2), (v_2, v_3), ..., (v_{k-1}, v_k)$, então $d[v_k] = \delta (s, v_k)$;
f) *Propriedade de subgrafo predecessor*: determina que se $d[v] = \delta (s, v)$ para todo $v \in V$, o subgrafo predecessor é uma árvore de caminhos mais curtos com raiz em s.

4.2 Conceitos Gerais de Algoritmos de Busca

A utilização de algoritmos de busca em problemas de caminho mínimo envolve a definição de quatro conceitos: *(i)* o estado inicial, *(ii)* a função sucessor, *(iii)* o teste objetivo e *(iv)* o custo do caminho (RUSSEL; NORVIG, 1995).

O estado inicial consiste na definição da origem da busca. A função *sucessor* consiste na determinação das ações possíveis de serem executadas a partir do estado atual.

Implicitamente, o estado inicial e a função *sucessor* definem o espaço de estados do problema. O espaço de estados é o conjunto de todos os estados acessíveis a partir do estado inicial. O espaço de estados forma um grafo onde os vértices são os estados e as arestas as ações. O teste objetivo determina se um dado estado é ou não o objetivo da busca e o custo do caminho é uma função que atribui um custo numérico a cada caminho (RUSSEL; NORVIG, 1995).

Em um grafo representado por $G = (V, E)$, onde V é um conjunto de vértices e E o conjunto de arestas, um arco $a = (id, i, j)$ é uma aresta dirigida entre os vértices i e j, identificada por *id* e com um custo c_a. A Figura 13 demonstra o pseudocódigo de um algoritmo genérico de busca. A função sucessor é denotada por *suc(i)* e retorna um conjunto de arestas $E' \in E$ com todas as arestas $a = (id, i, x)$ que ligam o vértice i a um outro vértice x qualquer. Esta operação é análoga a operação *incidentEdges(v)* no grafo do espaço de estados (ver seção 3.2). Todas as arestas retornadas pela função *suc(i)*, são guardadas em uma lista $OPEN \in E$, também chamada de fronteira, para serem avaliadas pelo algoritmo futuramente. A ação de invocar a função *suc(i)* para um vértice i é comumente chamada de expandir o vértice i. A função *select(OPEN)* retorna o próximo vértice a ser explorado com o objetivo de testar se faz parte do caminho. O algoritmo invoca sucessivamente as funções *suc(i)* e *select(OPEN)* para buscar o caminho entre o vértice de origem e o vértice de destino. O processo é iterativo e gera uma árvore de busca, onde cada caminho da árvore do vértice *raiz* até um vértice x qualquer representa uma solução parcial, um caminho existente entre os vértices o e x, denotado por $p_{o,x}$ e com um custo $c_{o,x}$ (JAQUES et al., 2012a). A notação $p_{o,d}$ representa uma lista que guarda os arcos que compõem o caminho do vértice o ao vértice d.

Figura 13 – Pseudocódigo de um algoritmo de busca genérico de caminho mínimo.

```
Input: nó o ∈ V de origem, nó d ∈ V de destino
Output: um caminho p_o,d

1  OPEN ← o ;
2  while OPEN ≠ ∅ do
3  |    i ← select(OPEN) ;
4  |    if i == d then
5  |    |    return p_o,d ;
6  |    end
7  |    else
8  |    |    E´ ← suc(i);
9  |    |    foreach a = (i,j) ∈ E´ do
10 |    |    |    p_o,j ← p_o,i + a ;
11 |    |    |    OPEN ← OPEN + j ;
12 |    |    end
13 |    end
14 end
```

Fonte: Jaques (2012a), adaptado pelo autor.

Segundo Russel e Norvig (1995), a eficiência dos algoritmos de busca pode ser avaliada basicamente sob quatro aspectos:

a) *Completeza*: consiste na garantia de identificação de uma solução, quando esta existir, ou seja, um algoritmo é dito completo quando sempre encontra uma solução;
b) *Otimização*: consiste na identificação da melhor solução, nas situações onde existirem diferentes soluções;
c) *Complexidade de tempo*: consiste no tempo gasto pelo algoritmo para encontrar uma solução;
d) *Complexidade de espaço*: consiste na quantidade de memória necessária para realizar a busca.

As estratégias adotadas pelos algoritmos de busca podem ser separadas basicamente em duas categorias: *(i)* busca sem informação e *(ii)* busca com informação (RUSSEL; NORVIG, 1995). A estratégia de busca sem informação é comumente conhecida como busca cega e a estratégia de busca com informação é conhecida como busca heurística.

4.3 Algoritmos de Busca Cega

A busca cega, ou também chamada exaustiva, não sabe qual é o melhor vértice, entre as fronteiras possíveis, que deve ser expandido em busca do menor custo de caminho do

vértice atual até o vértice final (objetivo). Os algoritmos baseados nesta estratégia não dependem de informações próprias do problema para resolvê-lo. Estes algoritmos são baseados na estrutura do espaço de estados e determinam estratégias sistemáticas para sua exploração, ou seja, seguem uma estratégia fixa no momento de visitar os vértices que representam os estados do problema. Trata-se também de algoritmos exaustivos, de modo que podem acabar recorrendo a todos os vértices do problema para achar a solução. O custo destes algoritmos pode ser proibitivo na maioria dos problemas reais e, portanto, seu uso deve limitar-se a problemas pequenos. A vantagem dessas técnicas decorre do fato que não se faz necessário possuir nenhum conhecimento adicional sobre o problema, visto que elas são sempre aplicáveis (RUSSEL; NORVIG, 1995).

4.3.1 Busca em Largura

A busca em largura, ou também chamada de busca em extensão, encontra a solução de menor custo de caminho através da expansão dos vértices por níveis. Primeiramente, são expandidos todos os vértices do primeiro nível, depois todos os vértices do segundo nível, e assim por diante. Esta busca está associada a um problema de custo exponencial em razão do número de vértices gerados a partir da expansão de cada vértice. Desta maneira, esta busca pode esgotar rapidamente a memória do computador, além de consumir grande tempo para a execução completa (RUSSEL; NORVIG, 1995).

Segundo Russel e Norvig (1995), a busca em largura pode ser implementada com uma fila do tipo *first in first out* (FIFO) que mantém os vértices visitados inicialmente, garantindo assim que serão também expandidos primeiramente, ou seja, a fronteira *OPEN* será uma fila. A fila FIFO armazena todos os sucessores gerados no final da fila, assegurando assim que os vértices de baixa profundidade serão expandidos antes dos vértices mais profundos. Nesse caso, a função *select(OPEN)* sempre retornará o primeiro elemento inserido na fronteira. A busca em largura gera caminhos mínimos para grafos não ponderados, ou seja, quando as arestas possuem peso igual a um. Neste caso, o custo de um caminho é igual ao número de arestas presentes naquele caminho.

4.3.2 Busca em Profundidade

A busca em profundidade expande o vértice que se encontra no nível mais profundo daquele ramo do grafo, podendo assim encontrar a solução ou não. Quando essa busca chega

ao vértice do nível mais profundo sem encontrar a solução, retorna à superfície para expandir outro ramo da grafo. Esta busca deve ser evitada quando as árvores geradas são muito profundas ou geram caminhos infinitos (RUSSEL; NORVIG, 1995).

Segundo Russel e Norvig (1995), a busca em profundidade pode ser implementada com uma fila do tipo *last in first out* (LIFO), ou também chamada pilha. Nesse caso, a fronteira *OPEN* será uma pilha e a função *select(OPEN)* sempre retornará o elemento inserido mais recentemente na fronteira. Outra possibilidade, é implementar a busca em profundidade através de uma função recursiva que invoca a si mesma sucessivamente para cada um de seus filhos.

4.3.3 Algoritmo Dijkstra

O algoritmo Dijkstra é indicado para problemas de caminhos mínimos de uma origem única em um grafo orientado ponderado $G = (V, E)$, onde os pesos das arestas são não negativos. O algoritmo mantém um conjunto S de vértices cujos pesos finais de caminhos mínimos desde a origem s, já foram determinados. O algoritmo seleciona repetidamente o vértice u de menor estimativa mínima de caminhos mais curtos, adiciona u a S e relaxa todas as arestas que saem de u. O relaxamento das arestas, neste caso, é realizado com o objetivo de testar se é possível melhorar o caminho mínimo para um determinado vértice v encontrado até o momento pela passagem no vértice u. A Figura 14 demonstra o pseudocódigo do algoritmo Dijkstra.

Figura 14 – Pseudocódigo do algoritmo Dijkstra.

```
Input: nó o ∈ V de origem, nó d ∈ V de destino
Output: um caminho p_o,d

1  OPEN ← o ;
2  while OPEN ≠ ∅ do
3      i ← select(OPEN) ;
4      if i == d then
5          return p_o,d ;
6      end
7      else
8          E´ ← suc(i);
9          foreach a = (i,j) ∈ E´ do
10             if c_o,i + c_a < c_o,j then
11                 c_o,j ← c_o,i + c_a ;
12             end
13             p_o,j ← p_o,i + a ;
14             OPEN ← OPEN + j ;
15         end
16     end
17 end
```

Fonte: Jaques (2012a), adaptado pelo autor.

Segundo Cormen et. al (2002), o algoritmo Dijkstra implementa a fronteira OPEN por meio de uma fila de prioridades de vértices tendo como chave os valores correspondentes ao custo do caminho. Desta maneira, o algoritmo garante que os vértices que serão expandidos primeiramente são aqueles com a menor estimativa de caminho mínimo. Na Figura 14, a notação $p_{o,d}$ representa uma lista que guarda os arcos que compõem o caminho do vértice *o* ao vértice *d*. Se já existe um caminho conhecido entre *o* e *x*, o algoritmo apenas substitui o caminho antigo pelo novo caminho, caso o novo caminho tenha um custo menor (linhas 10-11). Isso é chamado de relaxamento. Pelo fato do algoritmo sempre escolher o caminho a princípio mais econômico, pode-se afirmar que utiliza uma estratégia gulosa.

4.3.4 Busca Bidirecional

A busca bi-direcional é regida pela ideia de execução de duas buscas simultâneas, uma direta a partir do estado inicial e outra inversa a partir do objetivo, parando quando as duas

buscas se encontram em um ponto intermediário. A principal motivação da implementação deste método é a redução do espaço da busca. (RUSSEL; NORVIG, 1995).

Conforme Russel e Norvig (1995), a busca bidirecional pode ser implementada fazendo com que uma ou ambas as buscas verifiquem cada vértice antes de ser expandido, para saber se o vértice está na borda da outra busca, encontrando assim a solução.

4.4 Algoritmos de Busca Heurística

Algoritmos de busca heurística utilizam conhecimento específico do problema na escolha do próximo vértice a ser expandido através da aplicação de uma função de avaliação, denominada função heurística. As funções heurísticas são as formas mais comuns de introduzir conhecimento adicional do problema no algoritmo de busca. Uma função heurística estima o caminho mais econômico entre um vértice *n* até um vértice objetivo, com base em informações relacionadas ao problema. Uma função heurística deve ser admissível, ou seja, nunca deve superestimar o custo para alcançar o objetivo. (RUSSEL; NORVIG, 1995).

4.4.1 Busca Gulosa

A busca gulosa, ou também chamada busca gulosa pela melhor escolha, tenta expandir o vértice mais próximo ao vértice final com base na estimativa feita por uma função de avaliação, denotada por *f(n)*. A função de avaliação *f(n)*, neste caso, é calculada somente com base no valor da função heurística, denotada por *h(n)*, ou seja, neste caso o algoritmo considera *f(n) = h(n)*. Deste modo, pode-se dizer que a busca gulosa avalia os vértices para expansão utilizando apenas a função heurística. O custo da busca é minimizado e ela não expande vértices fora do caminho, escolhendo o caminho mais econômico à primeira vista (RUSSEL; NORVIG, 1995).

Segundo Russel e Norvig (1995), a fronteira *OPEN* na busca gulosa pode ser implementada por meio de uma fila de prioridades de modo que manterá a borda em ordem ascendente de valores de *f(n)*. A busca gulosa é semelhante à busca em profundidade, pelo fato de preferir seguir um único caminho até o objetivo.

4.4.2 Algoritmo A*

O algoritmo A* tenta expandir o vértice mais próximo ao vértice final com base na estimativa feita pela função heurística, mas sem desconsiderar o custo para alcançar cada vértice. Deste modo, o algoritmo A* avalia os vértices para expansão usando as informações do custo do caminho e da estimativa realizada pela função heurística. Assim como a busca gulosa, o algoritmo A* não expande vértices fora do caminho, escolhendo o caminho mais econômico à primeira vista, mas se diferenciando basicamente através do teste objetivo (RUSSEL; NORVIG, 1995).

Da mesma forma que a busca gulosa, a fronteira no algoritmo A* pode ser implementada por meio de uma fila de prioridades com o objetivo de manter a borda em ordem ascendente de valores. O algoritmo A* combina a busca gulosa (econômica, porém não completa e nem ótima) com o algoritmo Dijkstra (ineficiente, porém completo e ótimo) (RUSSEL; NORVIG, 1995).

4.5 Comparativo entre Algoritmos de Busca

Conforme já revisado na seção 4.2, a eficiência dos algoritmos de busca é medida sob quatro aspectos, como completeza, otimização, complexidade de tempo e espaço. A Tabela 4 demonstra a eficiência dos algoritmos de busca citados anteriormente. Neste contexto, *b* significa fator de ramificação da árvore de busca, *d* a profundidade da solução mais rasa na árvore de busca e *m* a profundidade máxima da árvore de busca. Entende-se por fator de ramificação o número máximo de sucessores de um vértice. As anotações sobrescritas utilizadas apresentam o seguinte significado: ^{1} completa se *b* é finito, ^{2} ótima se os custos dos passos são todos idênticos e ^{3} se ambos os sentidos utilizam busca em extensão (RUSSEL; NORVIG, 1995). Não foram localizadas informações a respeito da complexidade de espaço para o algoritmo de Dijkstra, mas sua complexidade de tempo depende de *V* que representa o número de vértices e *E* que representa o número de arestas.

Tabela 4 - Comparação entre os algoritmos de busca.

Critério	Largura	Profundidade	Dijkstra	Bidirecional	Gulosa	A*
Complexidade de Tempo	O (b^{d+1})	O (b^{m})	O (($\|V\|+\|E\|$) log $\|V\|$)	O (b^{d})	O (b^{m})	O (b^{d})
Complexidade de Espaço	O (b^{d+1})	O (b^{m})	--	O ($b^{d/2}$)	O (b^{m})	O (b^{d})
Ótima	Sim[2]	Não	Sim	Sim[2,3]	Não	Sim
Completa	Sim[1]	Não	Sim	Sim[1,3]	Não	Sim

Fonte: Russel e Norvig (1995), Cormen et. al (2002).

O algoritmo A* foi escolhido para ser utilizado neste trabalho devido a duas características fundamentais: ser completo e ótimo. Em função do propósito deste trabalho, o algoritmo necessita ser completo, pois precisa sempre encontrar uma solução caso ela exista. Além disto, o algoritmo necessita ser ótimo, pois deve sempre encontrar a melhor solução em termos de distância ou tempo, evitando assim custos desnecessários ao usuário.

Existem outros algoritmos, como o Dijkstra, por exemplo, que assim como o A* também geram soluções ótimas e completas. Entretanto, existem outras características que também necessitam ser consideradas, como a complexidade de espaço e a complexidade de tempo e, neste caso, o algoritmo A* apresenta melhores resultados (FU; SUN; RILETT, 2006). Segundo Sedgewick e Vitter (1986), o algoritmo A* encontra o caminho mínimo em grafos que usam distância euclidiana como heurística com um esforço computacional *O(n)*, enquanto o Dijkstra necessita *O(n* log *n)*. Golden e Ball (1978) demonstraram que o algoritmo A* expande menos do que 10% dos nós que seriam expandidos pelo Dijkstra. Jaques et al. (2012a) demonstraram que o algoritmo A* gera soluções de menor custo do que o algoritmo de busca em largura.

5 TRABALHOS RELACIONADOS

Durante o levantamento bibliográfico, foram encontrados trabalhos que tratam sobre o emprego de algoritmos de busca na identificação de caminhos mínimos em redes viárias e redes de transporte público. Dentre os critérios considerados na comparação com este trabalho estão: *(i)* tipo de modal considerado, *(ii)* algoritmo utilizado para determinação do caminho mínimo, *(iii)* utilização do algoritmo na sua forma pura ou não, *(iv)* técnicas de aceleração de busca empregadas e *(v)* critérios de custo analisados na determinação do caminho mínimo. As próximas seções apresentam os trabalhos relacionados.

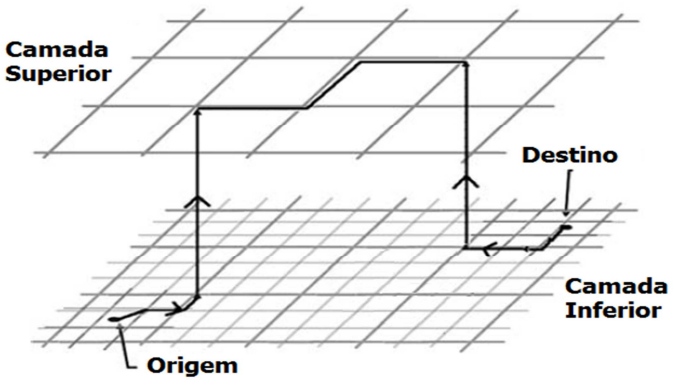
5.1 Algoritmo híbrido de caminho mínimo para a navegação de veículos

Cho e Lan (2008) realizaram um estudo sobre algoritmos de busca aplicados ao problema de caminho mínimo entre um par único de vértices em redes viárias. Neste estudo, técnicas de busca heurística como busca hierárquica, bidirecional e A* foram combinados a fim de desenvolver algoritmos híbridos que reduzam o espaço e o tempo da pesquisa. A busca hierárquica dividiu as rodovias em duas camadas:

a) Camada superior: compreende rodovias de alta mobilidade, como as rodovias expressas;
b) Camada inferior: compreende rodovias de alta acessibilidade, como as rodovias urbanas e rurais.

A busca hierárquica foi combinada com a busca bidirecional, criando um conceito denominado bidirecional hierárquico. O novo conceito de busca bidirecional hierárquica representado pela camada superior e camada inferior pode ser visualizado na Figura 15.

Figura 15 – Conceito de busca bidirecional hierárquica.



Fonte: Cho e Lan (2008), adaptado pelo autor.

A busca baseada no modelo híbrido bidirecional hierárquico, neste caso, é realizada em três fases: *(i)* calcular o caminho mínimo entre o vértice de origem e um vértice próximo da entrada da camada inferior, *(ii)* calcular o caminho mínimo entre o vértice de destino e um vértice mais próximo da saída da camada inferior, *(iii)* calcular o caminho mínimo entre o vértice de entrada e o vértice de saída da camada superior. Baseados nestes conceitos foram criadas três versões de algoritmos híbridos: D-D-D, A-A-A e A-D-A. Os algoritmos híbridos criados se diferenciam pelos algoritmos utilizados nas três fases da busca. O algoritmo D-D-D utiliza o Dijkstra em todas as fases e o algoritmo A-A-A utiliza o A* em todas as fases. O algoritmo A-D-A utiliza o Dijkstra na camada superior e o A* nas outras fases da busca. Para avaliar o desempenho dos algoritmos híbridos foram realizados testes comparativos com dados de uma rodovia real com mais de 37.000 vértices e 500.000 arestas. As instâncias dos problemas teste entre um vértice de origem e um vértice de destino foram geradas aleatoriamente e os resultados apresentados sempre demonstrados em relação ao Dijkstra. Utilizando o critério de distância percorrida, o algoritmo A* puro demonstra ser duas vezes mais rápido que o Dijkstra, o algoritmo D-D-D é aproximadamente 7.600 vezes mais rápido, o algoritmo A-D-A é 10.800 vezes mais rápido e o algoritmo A-A-A é 14.600 vezes mais rápido. Utilizando o critério de tempo o algoritmo A* demonstra ser aproximadamente duas vezes mais rápido que o Dijkstra, o algoritmo D-D-D é aproximadamente 21.900 vezes mais rápido, o algoritmo A-D-A é aproximadamente 25.500 vezes mais rápido e o algoritmo A-A-A é 33.400 vezes mais rápido. Os autores atribuem esta grande diferença de desempenho principalmente a redução do espaço da busca.

5.2 Combinação de técnicas de aceleração hierárquicas e direcionadas a objetivo para o algoritmo Dijkstra

Bauer et al. (2010) realizaram um estudo sistemático das principais técnicas de aceleração aplicadas em algoritmos de busca para a resolução do problema de caminho mínimo em grandes redes viárias. Segundo os autores, existem basicamente duas abordagens para acelerar a busca de caminhos mínimos: abordagens hierárquicas e abordagens direcionadas a objetivo. O objetivo principal deste estudo foi a revisão de combinações prévias já realizadas e a proposta de novas combinações a fim de obter perspectivas gerais de sucesso em diferentes cenários.

As abordagens hierárquicas basicamente realizam podas de ramos do grafo de maneira que o algoritmo Dijkstra deve relaxar apenas arestas suficientemente importantes de cada vértice. As principais técnicas hierárquicas são:

a) *Reach*: é técnica que realiza a poda de ramos que possuem um alcance muito pequeno, reforçada por uma inteligente integração de atalhos que representam todo o caminho no grafo original.
b) *Highway Hierarchies* (HH): é uma técnica que define uma região para cada vértice do grafo através dos seus *H* vizinhos mais próximos. A hierarquia é obtida após a definição de uma rede resultante para cada vértice de baixo grau, através de um procedimento recursivo. Os vértices removidos durante a iteração *i* de pré-processamento definem o nível de hierarquia *i-th*. O algoritmo de pesquisa é um Dijkstra bidirecional com restrições para relaxar arestas de vértices de baixo nível, que estejam distante da origem ou do destino.
c) *Highway Node Routing* (HNR): é uma técnica que define uma hierarquia de grafos sobrepostos a partir de uma sequencia do conjunto de vértices. O nível de sobreposição do grafo consiste de um conjunto de vértices e um conjunto de arestas que asseguram a propriedade de que todas as distâncias entre os vértices são iguais às distâncias correspondentes no grafo subjacente. O algoritmo de pesquisa bidirecional leva vantagem quando aplicado em um grafo sobreposto multinível, pois não necessita se deslocar para o nível inferior da hierarquia, resultando em um espaço de busca bastante reduzido.
d) *Transit Node Routing* (TNR): é uma técnica baseada em uma observação intuitiva utilizada por seres humanos. O objetivo principal é reduzir o espaço de busca através da definição de poucas, mas importantes, junções de tráfego a partir da origem, denominadas de vértices de acesso para frente. Analogamente, são definidas as junções de tráfego a partir do destino, denominadas de vértices de acesso para trás. A união dos vértices de acesso para frente com os vértices de acesso para trás resulta em um espaço de busca reduzido denominado roteamento de nós de trânsito.

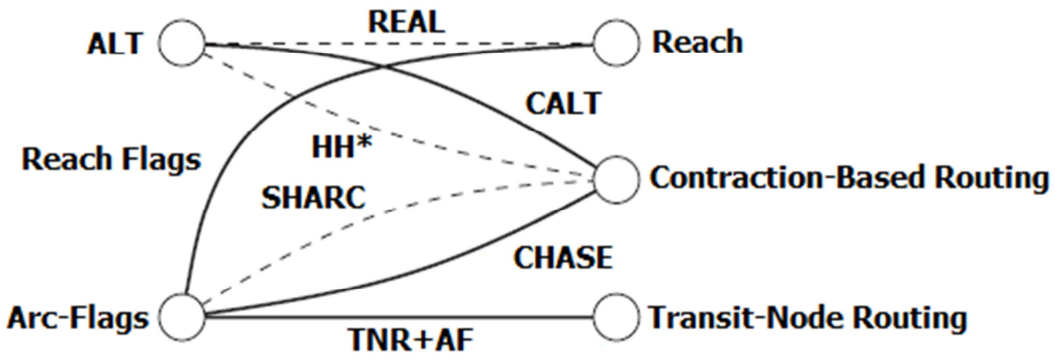
As abordagens direcionadas a objetivo basicamente trabalham no direcionamento da busca, dando preferência a arestas que diminuam a distância para o destino e desconsiderando arestas que possam não pertencer ao caminho mínimo. As principais técnicas direcionadas a objetivo são:

a) *Alt*: é uma técnica baseada no algoritmo A*, na definição de marcos e no teorema da desigualdade triangular. Após selecionar um pequeno número de vértices, denominados marcos, as distâncias da origem e do destino para cada marco são pré-computadas para todos os vértices. Considerando os vértices de origem e destino, a desigualdade triangular produz para cada marco dois limites inferiores. O maior valor entre estes limites inferiores é utilizado como parâmetro durante a busca A*. Esta técnica requer um pequeno tempo de pré-processamento e possui uma velocidade razoável, mas consome demasiado espaço de memória para grandes redes.
b) *Arc-Flags* (AF): é uma técnica baseada na definição de uma partição *C* para o grafo. Uma partição C_i é uma família de conjuntos de vértices de tal forma que cada vértice *V* está contido exatamente em uma única partição. Um elemento de cada partição é

denominado célula. Um rótulo é atribuído a cada aresta *E*. Um rótulo contém, para cada célula, uma *flag* $AFc_i(E.)$ Esta *flag* somente é marcada como verdadeira caso exista um caminho mínimo para o vértice em c_i começando em *E*. Uma versão do algoritmo Dijkstra modificado só considera na busca arestas para os quais a *flag* da célula de destino é verdadeira. Esta técnica possui um algoritmo de busca bastante rápido, mas um tempo de pré-processamento muito elevado.

Segundo os autores, muitas técnicas de aceleração de busca podem ser combinadas. A Figura 16 exibe uma visão geral sobre a combinação de técnicas de aceleração. As técnicas de aceleração são ilustradas na imagem como vértices de um grafo, onde as abordagens direcionadas a objetivo estão à esquerda e as abordagens hierárquicas estão à direita. As arestas pontilhadas indicam uma combinação já existente e as arestas contínuas indicam as combinações avaliadas neste trabalho.

Figura 16 – Visão geral das combinações de técnicas de aceleração.



Fonte: Bauer et al. (2010), adaptado pelo autor.

Para avaliar o desempenho das combinações de técnicas de aceleração foram realizados testes comparativos com dados de várias redes viárias da Europa Ocidental e Estados Unidos. Foram gerados aleatoriamente dez mil problemas testes com vértices de origem e destino distintos. Como medidas de efetividade foram avaliadas o consumo de memória e tempo de consulta. Em relação ao consumo de memória, a medição ocorreu em termos do número de vértices expandidos por consulta. De modo geral, as principais conclusões que os autores chegaram foram:

a) Em grafos esparsos, o algoritmo denominado CHASE (combinação entre *Contraction Hierarchies* e *Arc Flags)* produz excelentes resultados com baixo esforço de pré-processamento.
b) O algoritmo CHASE é superado pelo TNR em redes viárias quando utilizado o tempo de viagem como peso das arestas, mas com uma diferença muito pequena.
c) O algoritmo TNR melhora significativamente quando combinado com uma técnica direcionada a objetivo.

d) Em grafos mais densos, o algoritmo denominado CALT (combinação entre *Contraction Hierarchies* e *Alt)* produz bons resultados.
e) A combinação de métodos hierárquicos com métodos direcionados a objetivo, em geral, geram bons resultados para diferentes tipos de grafos.
f) As abordagens hierárquicas são mais indicadas para grafos esparsos.
g) As abordagens direcionadas a objetivo são mais indicadas para grafos mais densos.

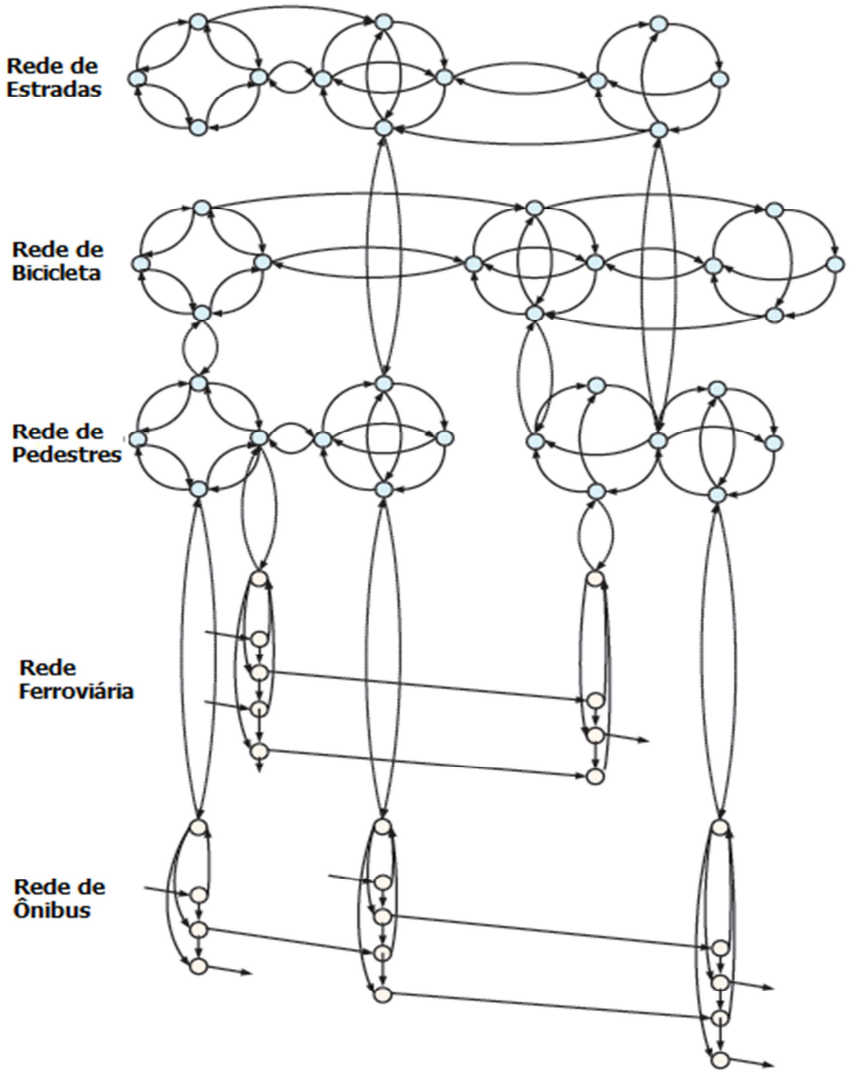
5.3 Um modelo de rede de transporte multimodal para sistemas avançados de informação ao viajante

Zhang et al. (2011) realizaram um estudo sobre a modelagem de sistemas de transporte multimodais. Segundo os autores, não existem abordagens de modelagem que permitam aos viajantes realizar consultas em diferentes modos de transporte onde são considerados, por exemplo, atributos como tempos de viagem dinâmicos e tabelas de horários em redes de grande escala. Em razão disto, estabeleceram como objetivo principal desenvolver e testar um modelo de transporte multimodal genérico para aplicações ATIS.

Primeiramente modelaram a rede de transporte multimodal a partir de um ponto de vista abstrato de modo que a rede fosse classificada em modo privado e modo público. Em seguida, inspirados em uma técnica conhecida como *super rede,* construíram uma representação da rede multimodal de transporte utilizando conexões de transferência. Entre todos os modos, a rede de pedestres desempenha um papel importante nas regras de modelagem das conexões de transferência. Foram previstos dois tipos de vértices: *(i)* vértices físicos e *(ii)* vértices de eventos. No modo de transporte privado, os vértices físicos representam segmentos de redes enquanto no modo de transporte público os vértices físicos representam paradas de ônibus ou estações e os vértices de eventos representam partidas e chegadas. Os vértices físicos possuem como atributos principais coordenadas geográficas e os vértices de eventos possuem tipo, horário e fatores relacionados com serviços, como por exemplo, linha de ônibus e sequencia de paradas de ônibus. Foram definidos dois tipos de conexão de transferência: *(i)* um entre o mesmo modo (transferência de uma linha para outra) e *(ii)* outro entre diferentes modos (transferência de ônibus para o trem). A rede a pé (rede de pedestre) possui um papel fundamental na transferência entre modos, pois todas as conexões de transferência são conectadas a esta rede e assim sempre poderia haver a caminhada envolvida nestas transferências. A Figura 17 ilustra a representação da rede de transporte multimodal proposta com as conexões de transferência entre os modos. São consideradas no

modelo as redes rodoviárias, de bicicleta, a pé, de ônibus e de trem. Segundo os autores, foi considerado na elaboração do modelo apenas o cálculo do tempo de viagem.

Figura 17 – Representação da rede multimodal de transporte.



Fonte: Zhang et al. (2011), adaptado pelo autor.

Para avaliar a viabilidade do modelo proposto, foram desenvolvidos dois algoritmos, um para compilar as informações da rede de transporte multimodal com base em dados de redes de estradas e serviços de transporte público e outro capaz de encontrar rotas multimodais de caminhos mínimos e, neste caso, o algoritmo utilizado foi o Dijkstra. Para avaliar os algoritmos foram realizados testes com dados de uma rede de transporte público da cidade de Eindhoven, na Holanda. Nestes dados constam informações a respeito de trajetos a pé, de bicicleta, de carro e de ônibus. Para realização dos testes algumas premissas foram assumidas:

a) O início da viagem como sendo dia 11/02/2011 às 18h10min.
b) A velocidade de caminhada como sendo de 4.5 km/h.
c) A velocidade de bicicleta como sendo de 12 km/h.
d) A velocidade de carro sendo limitada pela velocidade máxima da via.

Os dados utilizados incluem 4.654 vértices e 6.819 arestas relacionadas à rede a pé, 4.646 vértices e 6.781 arestas relacionadas à rede de bicicleta, 4.755 vértices e 6.957 arestas relacionadas a rede de carro e 1.736 paradas de ônibus e 121.584 eventos de partidas e chegadas de acordo com as tabelas de horários das linhas de ônibus da cidade. No trabalho não foram relatados detalhes dos testes realizados, mas o tempo de computação para a determinação da rota multimodal foi relatado como sendo geralmente inferior a um segundo. O principal problema relatado está relacionado com o algoritmo responsável pela compilação dos dados, devido ao seu alto tempo de computação. A partir dos testes realizados, os autores verificaram a validade e viabilidade do modelo multimodal genérico proposto e confirmam que o algoritmo Dijkstra pode ser usado para encontrar o caminho mínimo considerando como custo o tempo de viagem.

5.4 Algoritmo Dijkstra desenvolvido para pesquisa de caminhos mínimos e simulação

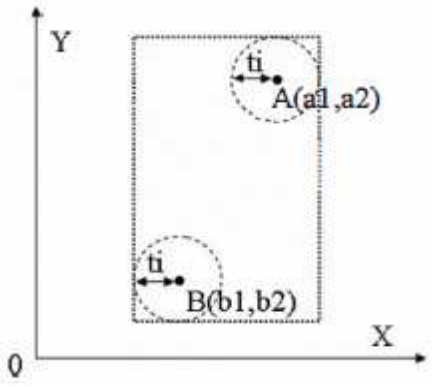
Chao e Hongxia (2010) realizaram um estudo sobre a aplicação do algoritmo Dijkstra na resolução do problema de caminho mínimo em redes viárias e identificaram que mesmo poderia ser aperfeiçoado caso o sentido da busca fosse levado em consideração. Segundo os autores, a versão pura do algoritmo Dijkstra apesar de resolver o problema de caminho mínimo de maneira precisa, possui um tempo de computação muito elevado quando utilizado em situações reais. O principal problema apontado no algoritmo original é que grande parte das operações realizadas durante o processo de busca são desnecessárias, pois a informação a respeito do posicionamento dos vértices no grafo em relação ao vértice de destino não é considerada. Desta forma, foi estabelecido como objetivo propor uma versão melhorada do algoritmo Dijkstra para determinação de caminhos mínimos em redes viárias que considere o sentido da busca e com isto possa reduzir o tempo de computação do algoritmo.

Segundo os autores, sistemas de transporte urbano são compostos de muitas intersecções e conexões entre estradas onde estas ligações formam uma complexa rede de tráfego. A modelagem de sistemas desta natureza, por exemplo, deve prever segmentos de estradas, intersecção entre segmentos de estradas, comunicação em dois sentidos,

comunicação em sentido único, etc. O processo de modelagem é na realidade uma abstração da rede tráfego urbano para um diagrama de redes da teoria dos grafos, onde um plano de rede de tráfego pode ser abstraído para um grafo ponderado dirigido.

O algoritmo Dijkstra puro, de maneira aproximada, pode ser descrito pela definição de uma série de círculos concêntricos em torno do ponto de origem, onde a direção ou localização do ponto de destino não é considerada durante o processo de busca. A complexidade de tempo para o algoritmo Dijkstra puro é de $O\ (n^2)$, mas se o número de vértices do grafo for relativamente grande, especialmente quando possui mais de cento e três ordens de magnitude, o número de operações realizadas pelo algoritmo cresce rapidamente. Para reduzir o número de operações realizadas pelo algoritmo foi proposta uma modelagem baseada na definição de regiões alvo prioritárias no grafo. A Figura 18 ilustra a determinação de regiões alvo prioritárias durante a busca. As letras A e B representam vértices de um grafo, onde o vértice A indica o ponto de origem e o vértice B indica o ponto de destino. Os vértices A e B são estabelecidos como sendo centro de um raio de tamanho *ti*, de modo que as regiões prioritárias incluem ao redor da origem e do destino todos os vértices mais próximos. Em sistemas reais de transporte, a probabilidade de o caminho mínimo estar entre estas regiões é muito alta. No processo de busca, o tamanho do raio *ti* aumenta a cada iteração, de modo que a primeira pesquisa pode não localizar um caminho mínimo dentro desta primeira região. Na próxima iteração, o raio de tamanho *ti* é ampliado de maneira a abranger uma área maior de pesquisa. Este processo é repetido até que o caminho mínimo seja localizado ou até que a área de pesquisa seja expandida para o grafo inteiro.

Figura 18 – Determinação de regiões alvo prioritárias durante a busca.



Fonte: Chao e Hongxia (2010), adaptado pelo autor.

Para avaliar a abordagem baseada na definição de regiões alvo prioritárias foram realizados testes comparativos com o algoritmo Dijkstra puro utilizando dados de redes de

tráfego da cidade de Jinan, na China, totalizando 2.375 vértices e 5.641 arestas. Foram realizadas três mil execuções utilizando como critérios de custo o tempo de viagem e distância total percorrida. O critério de eficiência utilizado foi o tempo de processamento dos algoritmos. A abordagem baseada na definição de regiões alvo prioritárias apresentou-se superior ao algoritmo Dijkstra puro em relação ao tempo de processamento utilizando como custo tanto o tempo de viagem como a distância total percorrida. Segundo os autores, esta diminuição no tempo de processamento ocorreu em função da redução do espaço de busca proporcionado pela definição de regiões alvo prioritárias.

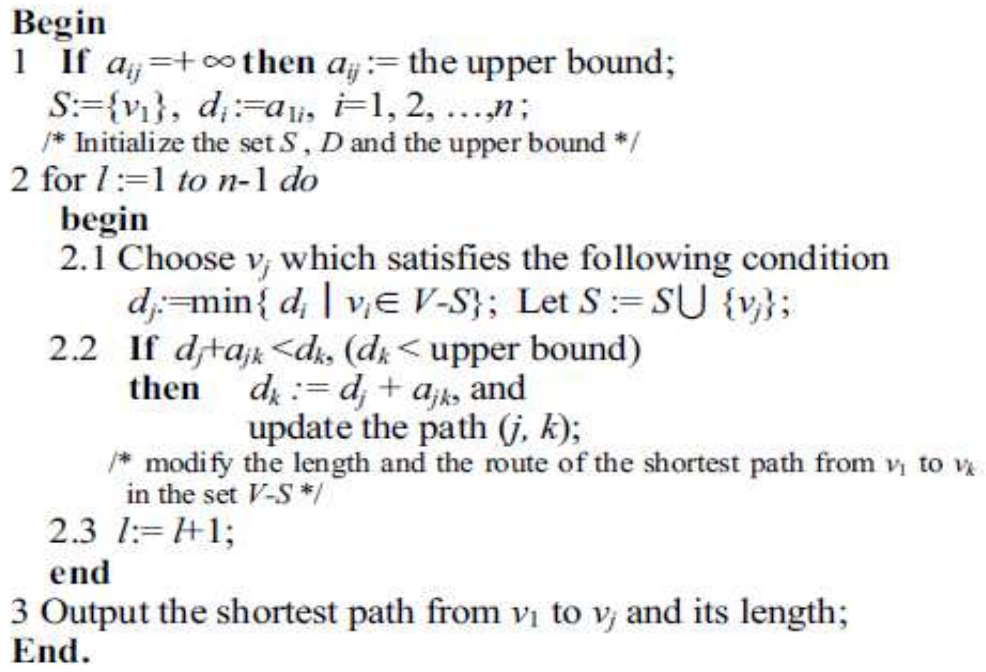
5.5 Algoritmo prático para caminhos mínimos em redes de transporte

Zhang, Jigang e Duan (2010) realizaram um estudo sobre a aplicação do algoritmo Dijkstra na resolução do problema de caminho mínimo em redes viárias e identificaram que o mesmo poderia ser aprimorado caso fosse definido um limite superior no processo de busca. Segundo os autores, a versão clássica do algoritmo Dijkstra foi projetada para resolver o problema de caminho mínimo de fonte única em um grafo estático e resolve o problema de maneira precisa, mas possui um tempo de computação muito elevado, pois necessita avaliar todo o grafo durante o processo de busca. Desta forma, foi estabelecido como objetivo propor uma versão melhorada do algoritmo Dijkstra para determinação de caminhos mínimos em redes viárias com base na definição de um limite superior durante o processo de busca que permita reduzir o tempo de computação do algoritmo.

Segundo os autores, em uma rede de transporte é possível estimar com antecedência a distância entre dois vértices e, a partir disto, definir um valor máximo, denominado limite superior, que deverá ser utilizado no processo de busca a fim de reduzir o número de atualizações de distância realizadas nos vértices pelo algoritmo Dijkstra clássico. Em um vértice arbitrário *V*, caso a distância atual para o vértice inicial seja maior que o limite superior, confirma-se que *V* não faz parte do caminho mínimo entre o vértice de origem e o vértice de destino e, desta forma, toda atualização de distância para o vértice *V* não será necessária. Além disto, os vértices sucessores que são expandidos no algoritmo Dijkstra clássico e estão distantes do vértice de origem não serão incluídos no restante do processo de busca pelo caminho mínimo fazendo com o espaço de busca seja reduzido a um subgrafo do grafo original. A Figura 19 ilustra o pseudocódigo do algoritmo proposto denominado *IMP_DA* baseado no algoritmo Dijkstra clássico. O limite superior é empregado durante a

fase de inicialização da matriz de adjacência. Primeiramente assume-se um valor arbitrário de limite superior e ao invés de atribuir o valor ∞ na posição da matriz, onde não existe uma aresta, é atribuído o valor do limite superior, conforme demonstrado no item um do pseudocódigo. O processo de busca inicia no item 2 sendo executado *n* vezes conforme o número de vértices do grafo. As atualizações de distância somente são consideradas caso o vértice avaliado satisfaça as condições mínimas relacionadas ao limite superior. Ao final da execução o algoritmo retorna o caminho mínimo entre um vértice de origem e um vértice de destino.

Figura 19 – Pseudocódigo do algoritmo IMP_DA.



Fonte: Zhang, Jigang e Duan (2010), adaptado pelo autor.

Para avaliar o algoritmo *IMP_DA* proposto foram realizados testes comparativos com o algoritmo Dijkstra clássico utilizando dados de redes de tráfego reais de uma sub rede do estado de Maryland, nos Estados Unidos, totalizando 150 vértices e 176 arestas. Os critérios de eficiência utilizados foram o tempo de processamento e o número de atualizações de distâncias realizadas sobre os vértices. O critério de custo utilizado na comparação foi a distância total percorrida. Como resultado, o algoritmo *IMP_DA* demonstrou ser superior ao algoritmo Dijkstra clássico no tempo de processamento, sendo até 8% mais rápido, e também reduziu no número de atualizações de distâncias realizadas sobre os vértices. Segundo os

autores, a diminuição do tempo de processamento foi possível em função da redução do espaço de busca proporcionada pela utilização do limite superior.

5.6 Engenharia de grafos expandidos por tempo para informações rápidas de horários

Delling, Pajor e Wagner (2008) realizaram um estudo sobre a determinação de caminhos mínimos envolvendo tabelas de horários de transporte público através de uma abordagem de grafos expandidos por tempo. Segundo os autores, existe uma abundância de trabalhos focados na utilização de técnicas de aceleração para redes viárias, mas não muitos trabalhos focados em adaptar estas técnicas para grafos que utilizam informações de horários. Em razão disto, foi proposto um modelo capaz de obter tempos de consulta mais rápidos e com menos consumo de espaço do que o modelo tradicional.

Conforme Delling, Pajor e Wagner (2008), existem duas abordagens para modelar informações de horários:

a) Dependente de tempo: são abordagens mais simples para modelar informações de horários. O modelo condensado, por exemplo, utiliza este tipo de abordagem. A modelagem, neste caso, geralmente prevê um vértice para cada estação e uma aresta para cada conexão direta entre duas estações. O peso das arestas representa o tempo mínimo de viagem sobre duas estações. Neste modelo não é considerado o horário real de partida de uma determinada estação, ou seja, o tempo de viagem não depende do dia e da hora e tão pouco considera o tempo necessário para a realização do transbordo. Como resultado, o tempo de viagem representa tão somente um limite inferior do tempo real de viagem. Abordagens dependentes de tempo geralmente possuem tempos de consulta menores do que as abordagens expandidas por tempo, uma vez que também trabalham com menos informações.
b) Expandida por tempo: são abordagens mais complexas, pois permitem uma modelagem mais flexível de restrições adicionais do problema. O modelo realístico expandido por tempo, por exemplo, utiliza este tipo de abordagem. A modelagem, neste caso prevê três tipos de vértices para representar eventos de horários. Partidas e chegadas são usadas para modelar as conexões elementares. Para cada conexão elementar, um vértice de chegada e um vértice de partida são criados e uma aresta é inserida entre eles. É utilizado também o modelo de transferência entre vértices. Para cada evento de partida um vértice de transferência é criado e conectado ao respectivo vértice de partida tendo peso zero. Para assegurar o tempo mínimo de transferência em uma determinada estação, uma aresta de cada vértice de chegada é inserida para o menor vértice de transferência (considerando tempo). Para cada aresta do grafo expandido o peso é definido pela diferença de tempo das arestas dos vértices conectados.

O modelo proposto pelos autores se baseia no modelo realístico expandido por tempo e no modelo de rota, que também segue uma abordagem expandida por tempo. No modelo,

além do algoritmo Dijkstra tradicional, foram utilizadas técnicas de contração e técnicas de aceleração para o busca de caminho mínimo. A técnica de contração, utilizada durante a fase de pré-processamento, basicamente remove vértices irrelevantes do grafo e acrescenta atalhos de maneira a manter a distância correta entre os vértices restantes. Como o algoritmo Dijkstra visita vértices desnecessários durante a fase de busca, foram introduzidas técnicas de aceleração como o bloqueio de vértices. Esta técnica, adaptada para redes expandidas por tempo, bloqueia vértices de partida irrelevantes durante a busca Dijkstra. Além disto, foram combinadas no modelo técnicas de aceleração direcionadas a objetivo como a AF e Alt.

Para avaliar o modelo proposto, foram realizados testes comparativos com o modelo realístico expandido por tempo utilizando o algoritmo Dijkstra tradicional, sem qualquer técnica de aceleração aplicada. Foram utilizados dados de uma rede ferroviária da Europa Central com 30.517 estações e 1.775.052 conexões elementares. Foram utilizados também dados de uma rede local de ônibus de Berlim, na Alemanha, com 2.874 paradas e 744.000 conexões. Como os tempos de transferências não foram fornecidos foram gerados valores aleatórios entre cinco e dez minutos para as ferrovias e entre três e cinco minutos para redes de ônibus. O desempenho dos algoritmos foi mensurado a partir da execução de mil consultas onde as estações de origem e destino foram geradas aleatoriamente. Os resultados indicaram que o modelo proposto reduz significativamente os tempos de consulta em relação ao modelo realístico expandido por tempo considerando como critério único o tempo de viagem. Os tempos de consulta produzidos pela nova abordagem chegam a ser 56.000 mais rápidos do que o algoritmo Dijkstra tradicional.

5.7 Algoritmos de planejamento de rotas para sistemas de transporte público

Liu et al. (2001) realizaram um estudo sobre a geração de planos de viagem envolvendo transporte público. Segundo os autores, planejar rotas de viagem através de transporte público é diferente de planejar rotas de viagem em redes viárias simples, pois no contexto de transporte público existem restrições de rota que os veículos em geral devem respeitar. Desta forma, o objetivo principal deste estudo foi propor duas estratégias capazes de captar e representar tais restrições e determinar planos de viagem. A primeira estratégia utiliza uma abordagem baseada em matrizes de adjacência e conectividade entre matrizes para representar a conectividade entre rotas e suas restrições. A segunda estratégia utiliza uma abordagem baseada na definição de *hubs* de transferência

A estratégia baseada em matrizes de adjacência identifica cada localização da rede com um número único. O valor de uma célula da matriz M_{ij} identifica o número de maneiras diretas onde é possível se deslocar de um local *i* para um local *j*. Caso não haja possibilidade deslocamento entre *i* e *j* o valor atribuído é zero. A matriz obtida representa a rede de transporte público de maneira que para determinar se existem formas de deslocamento entre uma origem e um destino é necessário verificar se o valor da célular M_{ij} é maior que zero. Baseado nesta estratégia foi desenvolvido o algoritmo denominado *Path Planning*. O algoritmo proposto, além de ser completo e ótimo também é capaz de minimizar o número de transferências.

A estratégia baseada em *hubs* de transferência define paradas prioritárias de acordo com a relativa importância das mesmas. Paradas que servem para mais de quinze rotas são selecionadas como *hubs.* Paradas de metrô e tradicionais sistemas de trens são automaticamente considerados como *hubs*. Para simplificar a tarefa de planejamento de caminhos são eleitos pelo menos dois *hubs* em cada rota. Baseado nesta estratégia foi desenvolvido o algoritmo denominado *Path Planning 2*. O algoritmo proposto é completo, mas não é ótimo, uma vez que pode recomendar um plano de viagem que requer a transferência entre *hubs* quando existe um plano de viagem indireto que exige a transferência em uma parada que não é um *hub* para uma viagem desejada.

Apesar de não serem relatados experimentos e resultados práticos sobre os algoritmos neste trabalho, os autores afirmam que a modelagem explícita de restrições de rota e *hubs* fornece uma oportunidade de encontrar planos de viagem de forma mais eficiente do que a modelagem em nível de parada. Além disto, estatísticas de tempo coletados em vários testes de campo indicam que o algoritmo pode calcular planos de viagem de maneira satisfatória dentro de poucos segundos.

5.8 Comparativo entre Trabalhos Relacionados

Com o objetivo de sintetizar e ao mesmo tempo sistematizar a comparação entre os trabalhos relacionados descritos anteriormente, é apresentada a Tabela 5. Entre os itens comparados estão: o tipo de modal considerado no trabalho; qual algoritmo foi utilizado para determinação do caminho mínimo; se foi utilizado um algoritmo puro ou modificado; se foi aplicada ou não alguma técnica de aceleração na busca; qual o critério de custo considerado na determinação do caminho mínimo.

Tabela 5 - Comparação entre os trabalhos relacionados.

Autor	Modal	Algoritmo (s) Utilizado (s)	Algoritmo Puro	Técnicas de Aceleração	Critério de Custo
Cho e Lan (2008)	Viário	Dijkstra, A*	Não	Hierárquica, Bidirecional	Tempo e Distância
Bauer et al. (2010)	Viário	Dijkstra	Não	*Reach*, HH, HNR, TNR, AF, Alt	Distância
Zhang et al. (2011)	Multimodal	Dijkstra	Sim	-	Tempo
Chao e Hongxia (2010)	Viário	Dijkstra	Não	Dijkstra Modificado	Tempo e Distância
Zhang, Jigang e Duan (2010)	Viário	Dijkstra	Não	Dijkstra Modificado	Distância
Delling, Pajor e Wagner (2008)	Transporte Público	Dijkstra	Não	AF, Alt, Bloqueio de Vértices	Tempo
Liu et al. (2001)	Transporte Público	-	Não	*Path Planning, Path Planning* 2	-

Fonte: Elaborado pelo autor.

De acordo com a tabela, embora o transporte público seja abordado como modal em alguns trabalhos, a maioria trata sobre a determinação de caminhos mínimos em redes viárias. Entretanto, um único trabalho (Zhang et al., 2011) foi localizado onde múltiplos modais foram considerados na construção de um modelo computacional. Em relação ao algoritmo de busca empregado, o Dijkstra é o mais utilizado, sendo que apenas um trabalho (Liu et al., 2001) não utilizou o mesmo em sua pesquisa. Por outro lado, um algoritmo A* foi utilizado apenas em um trabalho (Cho e Lan, 2008) e, neste caso, foi utilizado na resolução de caminhos mínimos envolvendo redes viárias. Em apenas um trabalho o algoritmos empregados não foram melhorados através de uma técnica de aceleração. Segundo os autores, os algoritmos clássicos como o Dijkstra, por exemplo, resolvem o problema de caminho mínimo, mas com um custo computacional muito elevado quando aplicados em grandes massas de dados envolvendo problemas reais. Por esta razão, várias técnicas de aceleração são testadas, como as técnicas hierárquicas e bidirecionais (Cho e Lan, 2008), (Bauer et al., 2010), (Delling, Pajor e Wagner, 2008). Alguns trabalhos (Chao e Hongxia, 2010), (Zhang, Jigang e Duan, 2010), apesar de não utilizarem técnicas de aceleração consagradas, optaram por realizar melhorias no algoritmo Dijkstra no intuito de melhorar o seu desempenho. Em relação ao critério de custo considerado, é possível perceber certo equilíbrio entre a distância e o tempo de percurso do caminho mínimo.

Com o objetivo de facilitar a comparação com os trabalhos relacionados, a Tabela 6 apresenta, com relação ao trabalho realizado, a definição dos mesmos critérios utilizados anteriormente.

Tabela 6 - Comparação com o trabalho realizado.

Autor	Modal	Algoritmo (s) Utilizado (s)	Algoritmo Puro	Técnicas de Aceleração	Critério de Custo
Rodrigo Bastos	Transporte Público (Ônibus)	A*	Não	Bidirecional, Direta	Distância e Número de Transbordos

Fonte: Elaborado pelo autor.

No presente trabalho, o modal considerado foi o transporte público, específicamente o ônibus. O algoritmo empregado para a determinação do caminho mínimo foi o A*. O algoritmo não foi aplicado na sua forma pura, haja vista que foi modificado através da combinação de técnicas de aceleração, como a bidirecional e a direta. A distância de viagem e o número de transbordos realizados se constituem nos múltiplos critérios de custo avaliados pelo algoritmo. Nenhum dos trabalhos relacionados citados anteriormente propõe a utilização do A* para o problema do caminho mínimo em uma rede de transporte público e, por isto, sendo isto destacado como uma das principais contribuições científicas deste trabalho.

6 TRABALHO REALIZADO

O trabalho realizado pode ser resumido como a elaboração de um modelo computacional para o problema do caminho mínimo multicritérios em uma malha de transporte público. Essa solução consiste na modelagem da base de dados, na representação do problema através de um grafo e também no emprego de técnicas de aceleração de desempenho aplicadas ao algoritmo A* a fim de tratar o problema e também reduzir tempo de resposta, de maneira que o modelo possa ser utilizado em um sistema real de informação ao usuário. A partir de um ponto de origem e um ponto de destino é possível determinar as melhores rotas de transporte público para o usuário, considerando como critérios de decisão a distância total percorrida e o número de linhas utilizadas. Também é possível prover informações relevantes ao usuário a respeito do percurso incluindo a localização de paradas mais próximas a origem e ao destino e também linhas de ônibus necessárias para percorrer o trajeto. Um protótipo do modelo foi implementado e avaliado através de um estudo de caso com os dados provenientes do sistema de transporte público da cidade de Porto Alegre. Esses dados foram obtidos através da EPTC (2012).

6.1 Definição do Problema do Caminho Mínimo

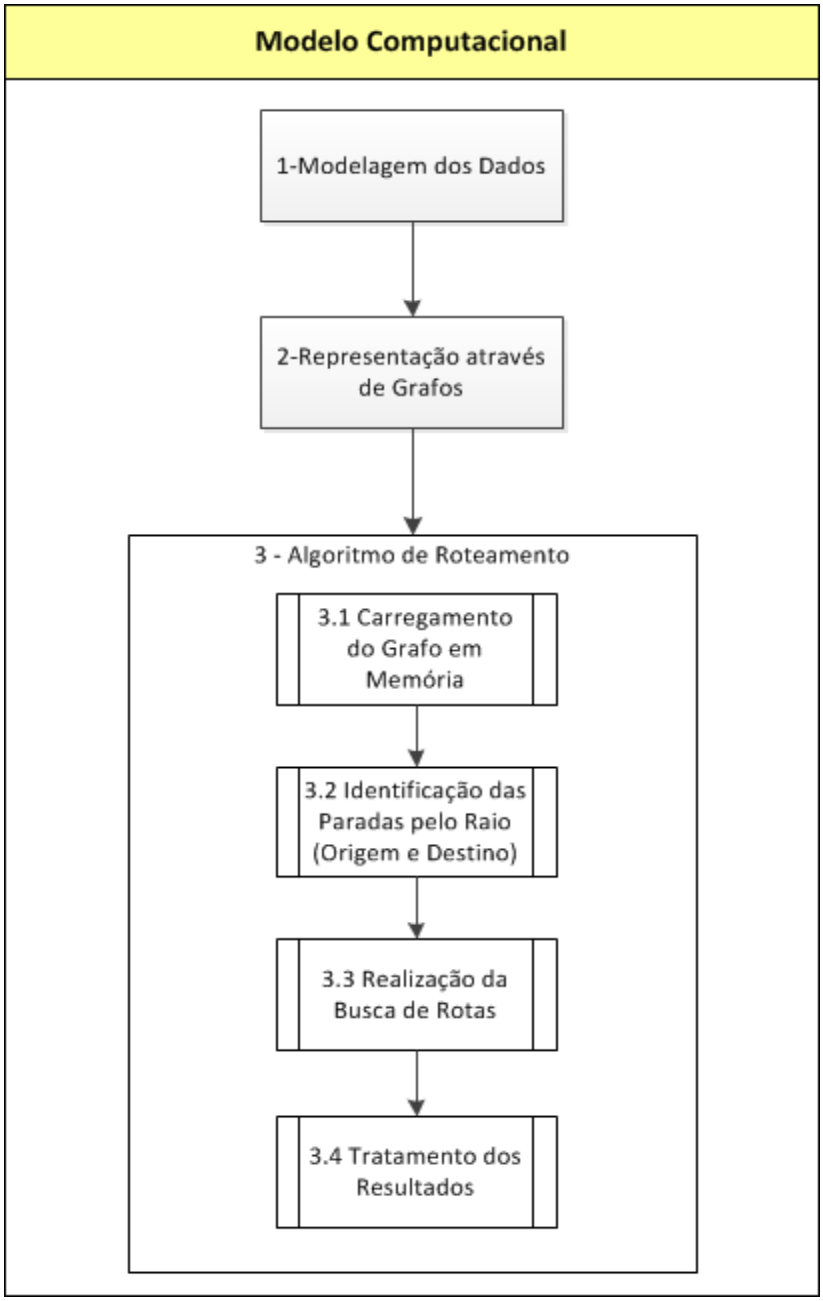
A definição formal do problema do caminho mínimo entre um par de vértices de um grafo, já definido na seção 4, envolve a determinação do caminho de menor custo entre um vértice de origem e um vértice de destino. Quando o problema é definido em termos de transporte público, a determinação do caminho de menor custo é realizada em função de uma parada de origem e uma parada de destino. Neste caso, as paradas são representadas pelos vértices do grafo e um trecho percorrido por uma linha de ônibus entre duas paradas é representado por uma aresta.

Desta forma, a definição do problema do caminho mínimo em uma malha de transporte público presente neste trabalho consiste em determinar o caminho de menor custo entre uma parada de origem e uma parada de destino, onde o custo é definido em termos de múltiplos critérios, como é o caso da distância e número de linhas utilizadas para percorrer o trajeto. Os múltiplos critérios considerados como custo do caminho foram avaliados individualmente ou em conjunto.

6.2 Modelo Computacional

O modelo computacional elaborado pode ser descrito a partir de uma série de etapas que podem ser visualizadas através da Figura 20. Fazem parte destas etapas: *(i)* a modelagem da base de dados, *(ii)* a representação do problema através de grafos e *(iii)* a implementação de um algoritmo de roteamento responsável pelo cálculo das rotas.

Figura 20 – Modelo computacional elaborado.



Fonte: Elaborado pelo autor.

A seguir, são apresentadas e detalhadas as etapas presentes no modelo computacional.

6.3 Modelagem dos Dados

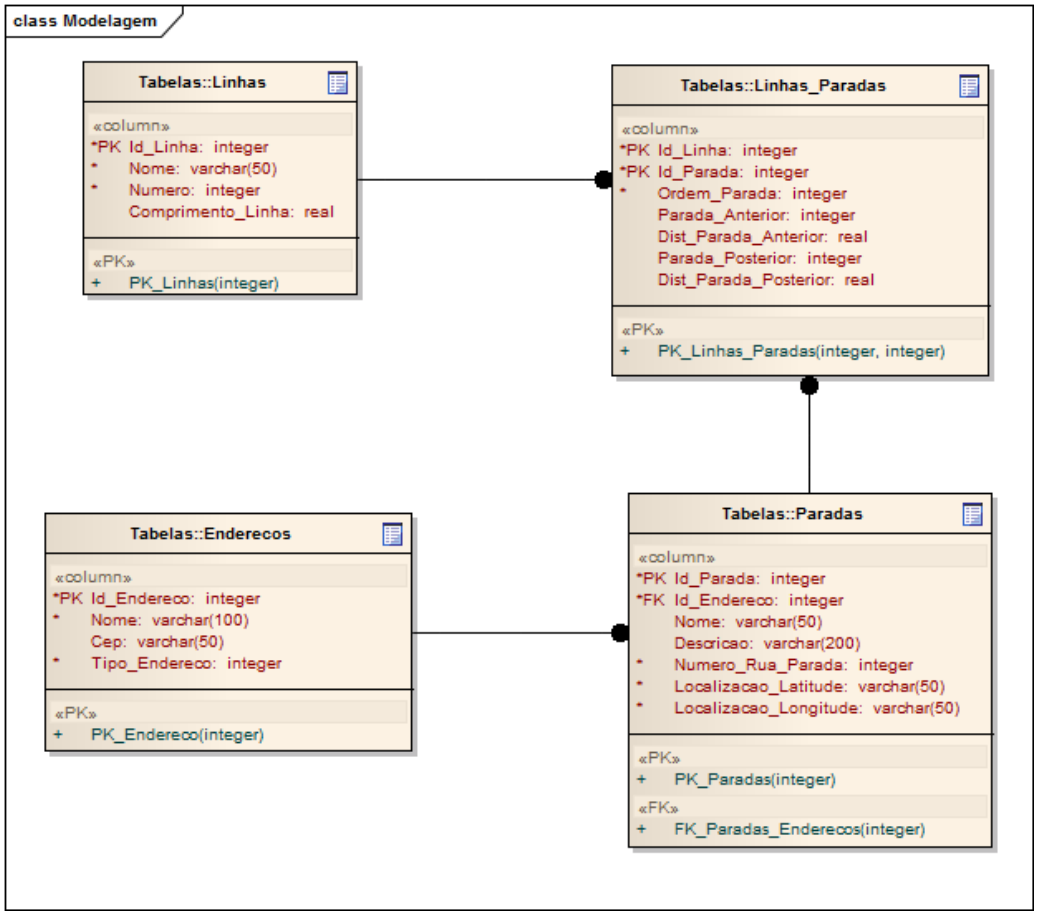
Um banco de dados é um conjunto de dados relacionados que possuem significância intrínseca, representando aspectos do mundo real que devem ser mantidos para atender os requisitos de um sistema. Existem cinco tipos de banco de dados: *(i)* hierárquico, *(ii)* rede, *(iii)* relacional, *(iv)* objeto-relacional e *(v)* objeto (OLIVEIRA, 2002). O tipo de banco de dados utilizado neste trabalho é o relacional, onde o objetivo é armazenar um grupo de objetos em um dicionário de dados, de forma a tornar rápida e segura a manipulação das informações contidas nestes objetos.

O banco de dados escolhido para este trabalho é o PostgreSQL. Este banco de dados é um sistema de código aberto, ou em inglês *open source*, com mais de quinze anos de desenvolvimento com forte reputação de confiabilidade, integridade de dados que é compatível com os principais sistemas operacionais existentes (POSTGRESQL, 2012).

As informações necessárias para determinação de caminhos mínimos em transporte público foram modeladas através do modelo de entidade e relacionamento. O principal objetivo da modelagem de dados é apresentar os requisitos das informações do negócio, através de uma representação gráfica denominada modelo de dados (OLIVEIRA, 2002).

A modelagem da base de dados é uma etapa essencial para construção do modelo computacional. Isto porque os algoritmos de busca, em geral, requerem um número mínimo de informações a respeito do problema para que possam ser aplicados da maneira adequada e produzam os resultados esperados. O estado final da modelagem realizada é demonstrado através da Figura 21.

Figura 21 – Estado final da modelagem da base de dados.



Fonte: Elaborado pelo autor.

Em uma malha de transporte público, basicamente existem linhas de ônibus, vias e paradas. As linhas, que são pré-estabelecidas pelas empresas, percorrem determinadas paradas localizadas nas vias. Na modelagem, a tabela *Enderecos* armazena informações a respeito das vias de uma determinada cidade ou região. A tabela *Paradas* armazena informações sobre as paradas e a tabela *Linhas* armazena informações sobre as linhas de ônibus. A tabela *Linhas_Paradas* é uma entidade associativa, que guarda informações sobre as paradas que compõem uma determinada linha de ônibus.

A tabela *Enderecos* é composta de quatro atributos: *Id_Endereco*, *Nome*, *Cep* e *Tipo_Endereco*. O atributo *Id_Endereco* é chave primária da tabela e, por isto, é numérico e sequencial. O atributo *Nome* armazena o nome do endereço e o atributo *Cep* sua localização. O atributo *Tipo_Endereco* define qual é o tipo de endereço (Rua, Avenida, Estrada, etc.).

A tabela *Paradas* é composta por sete atributos: *Id_Parada*, *Id_Endereco*, *Nome*, *Descricao*, *Numero_Rua_Parada*, *Localizacao_Latitude*, *Localizacao_Longitude*. O atributo *Id_Parada* é chave primária da tabela e, por isto, é numérico e sequencial. O atributo

Id_Endereco é uma chave estrangeira da tabela *Endereco* e representa um relacionamento de um (1) para *N* com esta tabela. Isto quer dizer que em um (1) endereço podem existir *N* paradas, ou também que cada parada está localizada em um único endereço. Os atributos *Nome, Descricao* e *Numero_Rua_Parada* armazenam informações a respeito de pontos de referência e localização da parada na via e, com isto, são utilizadas para prover informações ao usuário. Os atributos *Localizacao_Latitude* e *Localizacao_Longitude* armazenam coordenadas geográficas que definem exatamente o ponto onde cada parada está localizada.

A tabela *Linhas* é composta de quatro atributos: *Id_Linha, Nome, Numero* e *Comprimento_Linha.* O atributo *Id_Linha* é chave primária da tabela e, por isto, é numérico e sequencial. O atributo *Nome* armazena o nome da linha e o atributo *Numero* o número de identificação da mesma. O atributo *Comprimento_Linha* armazena informações sobre o comprimento total da linha, do ponto inicial ao final.

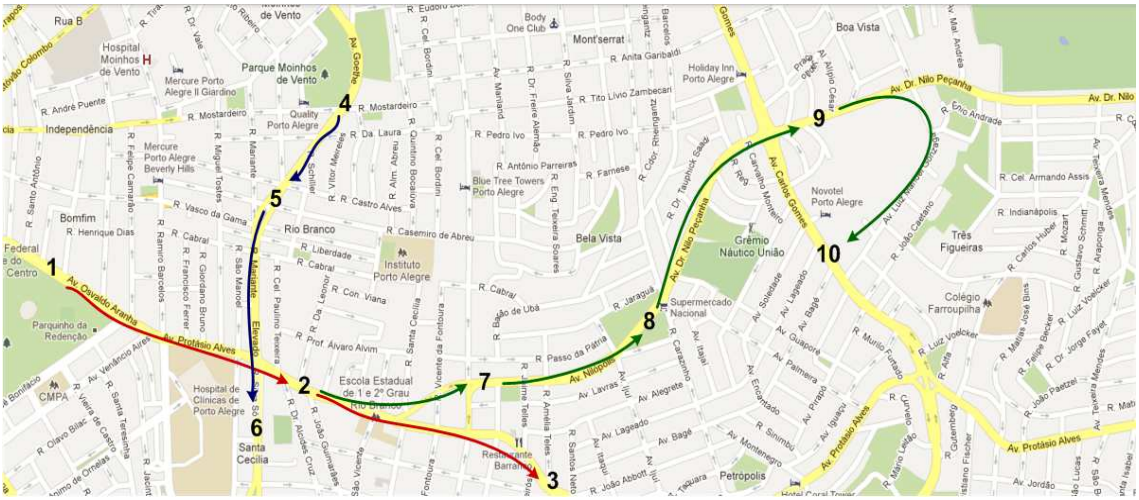
A tabela *Linhas_Paradas* é composta por sete atributos: *Id_Linha, Id_Parada, Ordem_Parada, Parada_Anterior, Dist_Parada_Anterior, Parada_Posterior, Dist_Parada_Posterior*. Os atributos *Id_Linha* e *Id_Parada* são chaves estrangeiras das tabelas Linhas e Paradas, respectivamente e juntas, formam uma chave primária composta da tabela. Esta relação representa que uma linha de ônibus é composta de diversas paradas e, ao mesmo tempo, uma parada pode fazer parte de diferentes linhas de ônibus. O atributo *Ordem_Parada* indica a ordem em que as paradas serão percorridas da origem (parada inicial) para o destino (parada final). Os atributos *Parada_Anterior* e *Parada_Posterior* representam as paradas anteriores ou posteriores de uma determinada linha, tomando como base a parada "atual" do caminho que está sendo avaliada. Por consequência, os atributos *Dist_Parada_Anterior* e *Dist_Parada_Posterior* representam as distâncias entre estas paradas.

6.4 Representação Através de Grafos

A representação do problema do caminho mínimo em uma malha de transporte público é realizada naturalmente através de um grafo dirigido, desconexo, rotulado e ponderado. O grafo é dirigido, pois, neste caso, respeita as direções das vias públicas e também a ordem pré-estabelecida do percurso da linha, entre a parada inicial e final. É desconexo, porque existem paradas que não são acessíveis através de uma linha de ônibus. O grafo é rotulado porque as paradas são identificadas através de coordenadas geográficas e é

ponderado porque são atribuídos custos as suas arestas. É considerado como custo a distância entre as paradas. Além disto, os vértices, neste caso, representam paradas de ônibus e as arestas representam um trecho de linha de ônibus entre duas paradas. A Figura 22 ilustra um exemplo de mapa rodoviário com a identificação de algumas linhas de ônibus e paradas.

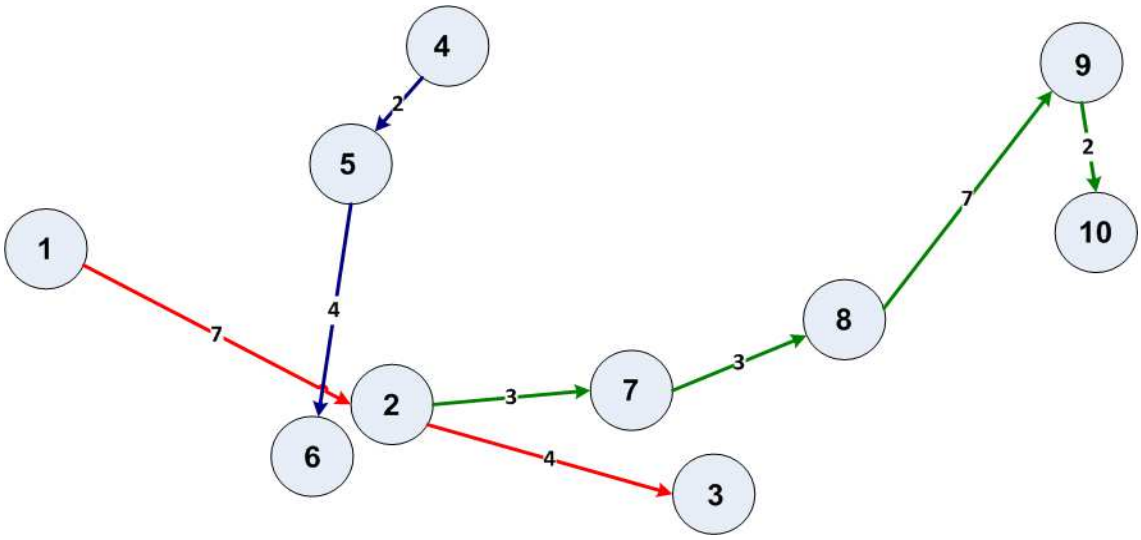
Figura 22 – Exemplo de mapa com identificação de algumas linhas e paradas.



Fonte: GoogleMaps (2012), adaptado pelo autor.

Na Figura 22, os números representam paradas definidas de forma arbitraria. Os ônibus desta região passam por estas paradas, respeitando o percurso pré-definido de acordo com cada linha (cada linha é representada por uma cor diferente: vermelho, verde e azul). Como exemplo, uma determinada linha poderia iniciar o trajeto na parada 1, seguir para a parada 2 e finalizar o trajeto na parada 3. Outra linha inicia o percurso na parada 4, segue para a parada 5 e finaliza o percurso na parada 6 e, por fim, uma linha inicia o trajeto na parada 2, segue para a parada 7, depois para a parada 8, em seguida para a parada 9 e finaliza o trajeto na parada 10. O mesmo exemplo, representado através de um grafo é demonstrado na Figura 23.

Figura 23 – Exemplo de uma representação através de grafos.



Fonte: Elaborado pelo autor.

No grafo da Figura 23, as dez paradas são representadas pelos vértices numerados de um a dez. As oito arestas representam a direção do percurso de cada linha e também o custo associado que, neste caso, representa a distância entre as paradas.

6.5 Algoritmo de Roteamento

A implementação do algoritmo de roteamento capaz de calcular as melhores rotas via transporte público pode ser descrita a partir das seguintes etapas: *(i)* carregamento do grafo em memória, *(ii)* identificação das paradas pelo raio, *(iii)* realização da busca de rotas, *(iv)* emprego de técnicas de aceleração na busca de rotas e *(v)* tratamento dos resultados. Como entrada, o algoritmo recebe informações sobre o ponto de origem e sobre o ponto de destino, definidos através de coordenadas geográficas, e também a informação da distância máxima aproximada que o usuário aceita percorrer a pé. Como saída, o algoritmo lista as melhores opções de rota ao usuário, considerando como critérios a distância total percorrida e o número de linhas necessárias para percorrer o trajeto. A seguir, são detalhadas as etapas necessárias para o desenvolvimento do algoritmo de roteamento.

6.5.1 Carregamento do Grafo em Memória

O carregamento do grafo em memória é o primeiro processo executado pelo algoritmo de roteamento. Conforme revisado na seção 3.4, o grafo é representado computacionalmente através de listas de adjacências. Para isto, são realizadas consultas a base de dados e para cada

registro existente na tabela *Paradas* é criado um vértice. Para cada vértice, é criada uma lista de adjacências de arestas de entrada e uma lista de adjacências de arestas de saída com base nos registros existentes nas tabelas *Linhas_Paradas* e *Linhas*. Como estrutura auxiliar, é carregada em memória uma lista das linhas de ônibus com suas respectivas paradas, mantidas de forma ordenada conforme o percurso da linha.

6.5.2 Identificação das Paradas pelo Raio

Para que seja possível calcular rotas via transporte público, é necessário identificar as paradas de origem e as paradas de destino que deverão ser avaliadas durante a fase de busca das rotas. Para isto, é necessário estimar quais são as paradas mais próximas do ponto de origem e do ponto de destino. Este processo é realizado através do cálculo da distância euclidiana comumente denominada distância linear entre os pontos de origem e pontos de destino e os demais vértices (paradas) do grafo. A distância euclidiana é calculada a partir das informações de latitude e longitude das paradas, usando fórmulas da trigonometria esférica descritas a seguir (GORE, 1907).

$$\cos\ (b) = \cos\ (a).\cos\ (c) + \ \text{sen}\ (a)\ .\text{sen}\ (c).\cos\ (B)$$

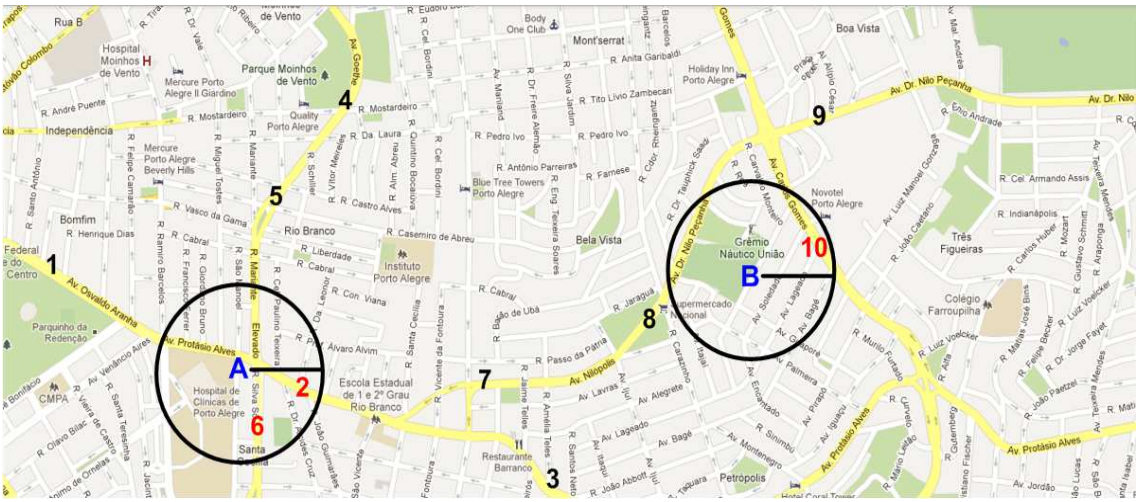
$$\cos\ (c) = \cos\ (b).\cos\ (a) + \ \text{sen}\ (b)\ .\text{sen}\ (a).\cos\ (C)$$

$$\cos\ (a) = \cos\ (b).\cos\ (c) + \ \text{sen}\ (b)\ .\text{sen}\ (c).\cos\ (A)$$

Primeiramente deve ser calculado individualmente o valor dos três arcos que juntos formam o triângulo esférico. O arco A é definido pela diferença entre as longitudes dos pontos. O arco B e o arco C são calculados segundo a diferença das latitudes dos pontos em relação à maior latitude terrestre, ou seja, 90 graus. A distância euclidiana é o resultado da multiplicação do arco *a* pelo arco completo da circunferência terrestre, e pela divisão deste resultado por 360 graus.

Considerando como parâmetro a distância máxima aproximada a ser percorrida a pé pelo usuário, é definido o raio que é utilizado para determinação das paradas próximas. Com isto é gerado um conjunto de paradas de origem ($O \subseteq V$) com tamanho |O| e um conjunto de paradas de destino ($D \subseteq V$) com tamanho |D| que, combinadas, irão gerar |O||D| entradas para o algoritmo responsável pela busca de rotas, aumentando assim, a possibilidade de localização de um ou mais caminhos mínimos. A Figura 24 ilustra um exemplo de determinação das paradas de origem e paradas de destino através do raio.

Figura 24 – Exemplo da determinação das paradas de origem e destino através do raio.



Fonte: GoogleMaps (2012), adaptado pelo autor.

Na Figura 24, os pontos A e B representam respectivamente o ponto de origem e o ponto de destino da busca definidos de maneira arbitrária. Os números representam paradas distribuídas ao longo de uma determinada região ou cidade no mapa. A partir da informação da distância máxima aproximada a ser percorrida a pé, são identificadas as paradas 2 e 6 como origem e a parada 10 como destino. Desta forma, o algoritmo de busca de rotas deverá avaliar não só a parada 2 como origem e a parada 10 como destino como também a parada 6 como origem e a parada 10 como destino.

6.5.3 Realização da Busca de Rotas

Para construir um modelo computacional para o problema do caminho mínimo em uma malha de transporte público é fundamental definir um mecanismo de busca de caminhos em grafos que garanta esta solução. Geralmente os mecanismos utilizados, nestas situações, são algoritmos da área de pesquisa operacional. Entretanto, este trabalho fez uso de um algoritmo da área de inteligência artificial, conhecido como A* a fim de melhorar o tempo de resposta, e compara esta solução com algoritmos estabelecidos na área tal como o Dijkstra.

O algoritmo A*, assim como os outros algoritmos de busca (ver algoritmo de busca genérico – Figura 13), possui as funções *suc(i)* e *select (OPEN)*. A função *suc(i)*, neste caso, retorna um conjunto de relações linhas paradas, que ligam a parada atual (o parâmetro *i* da função) a várias outras paradas através das linhas. Todas as relações <linhas, paradas> retornadas pela função *suc(i)*, são guardadas no conjunto *OPEN*, para serem avaliadas pelo algoritmo futuramente. A função *select(OPEN)* retorna a próxima parada a ser explorada com

o objetivo de testar se faz parte do caminho. O algoritmo invoca sucessivamente as funções *suc(i)* e *select(OPEN)* para buscar o caminho entre a parada de origem e a parada de destino. O processo, neste caso, é iterativo e gera uma árvore de busca, onde cada caminho entre a parada inicial até uma parada *x* qualquer representa uma solução parcial do problema.

A Figura 25 apresenta o pseudocódigo do algoritmo A*. Enquanto os algoritmos de busca genéricos utilizam o custo real do caminho, o algoritmo A* utiliza uma função heurística para estimar o melhor vértice a ser expandido, substituindo o caminho antigo pelo novo caminho, no caso do novo ter um custo menor (linhas 10 a 12). Essa solução melhora significativamente o desempenho do algoritmo de busca, sem interferir no seu caráter ótimo.

Figura 25 – Pseudocódigo do algoritmo A*.

```
Input: nó o ∈ V de origem, nó d ∈ V de destino
Output: um caminho p_o,d

1  OPEN ← o ;
2  while OPEN ≠ ∅ do
3  |   i ← select(OPEN) ;
4  |   if i == d then
5  |   |   return p_o,d ;
6  |   end
7  |   else
8  |   |   E´ ← suc(i) ;
9  |   |   foreach a = (i,j) ∈ E´ do
10 |   |   |   if c_o,i + c_a + e_j,d < F_o,j then
11 |   |   |   |   c_o,j ← c_o,i + c_a ;
12 |   |   |   |   F_o,j ← c_o,i + c_a + e_j,d ;
13 |   |   |   end
14 |   |   |   p_o,j ← p_o,i + a ;
15 |   |   |   OPEN ← OPEN + j ;
16 |   |   end
17 |   end
18 end
```

Fonte: Jaques (2012a), adaptado pelo autor.

O A* é um algoritmo heurístico, ou seja, utiliza uma heurística com o objetivo de reduzir o espaço da busca e, assim, reduzir a complexidade de espaço e tempo (NILSSON, 1971). Por consequência, é necessário que o algoritmo A* tenha algum conhecimento específico sobre o problema para estimar qual é o melhor vértice da fronteira que deve ser expandido, ou seja, para definir qual será o retorno da função *select(OPEN)*.

Para o critério de distância, o algoritmo A* deste trabalho utiliza como heurística a distância euclidiana. A distância euclidiana foi escolhida por ser admissível, ou seja, nunca superestima o custo para alcançar o objetivo, uma vez que a menor distância entre dois pontos é uma linha reta (RUSSEL; NORVIG, 1995).

Para garantir que os melhores caminhos possam ser expandidos na ordem correta, o conjunto *OPEN* é tratado como uma fila de prioridade. Os vértices são ordenados com base no valor do custo da aresta c_a somado ao valor da função heurística $e_{j,d}$, onde $e_{j,d}$ representa o custo estimado entre o vértice j e o vértice de destino d (JAQUES et al., 2012a).

Os algoritmos de busca heurística diferem dos algoritmos ótimos tradicionais por utilizar conhecimentos a respeito do problema para realizar a busca do caminho mínimo. Desta forma, os algoritmos heurísticos são classificados em função da estratégia de busca que utilizam. Basicamente existem quatro estratégias de busca: *(i)* limitação da área de busca, *(ii)* decomposição do problema de busca, *(iii)* limitação dos arcos pesquisados e *(iv)* a combinação das técnicas anteriores (FU; SUN; RILETT, 2006). O algoritmo A* puro utiliza a estratégia de limitação da área de busca, pois através de uma função heurística restringe a busca dentro de uma área delimitada, sendo esta geralmente menor do que a área de busca de algoritmos tradicionais como o Dijkstra (FU; SUN; RILETT, 2006) (GOLDEN; BALL, 1978).

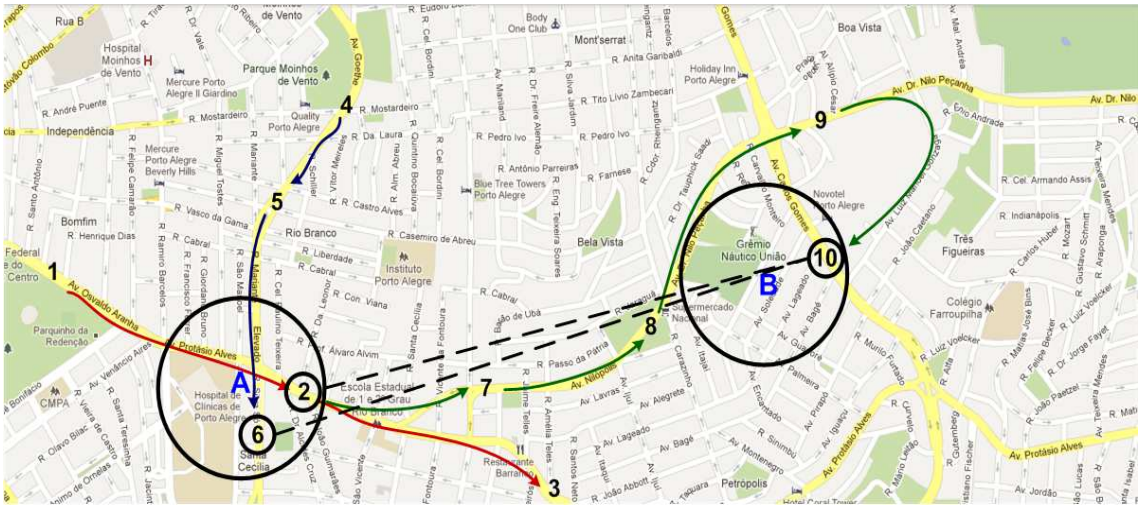
6.5.4 Emprego de Técnicas de Aceleração na Busca de Rotas

Devido à grande quantidade de informações que é necessário armazenar e manipular em um sistema real de transporte público foram utilizadas diferentes técnicas de aceleração de desempenho no algoritmo A*. Apesar do algoritmo A* ser o mais empregado entre os algoritmos heurísticos, a combinação de técnicas pode ser mais eficaz do que sua utilização de forma isolada (FU; SUN; RILETT, 2006). Em função disto, foram aplicadas no algoritmo técnicas de aceleração de busca, como a busca direta e a busca bidirecional.

O conceito da busca direta, utilizado neste trabalho, está relacionado a uma observação intuitiva do problema de caminho mínimo em transporte público. Antes de iniciar o algoritmo A*, é possível identificar rapidamente se existem linhas de ônibus que percorrem diretamente a parada de origem e também a parada de destino em seu percurso através de uma consulta ao grafo. Esta consulta é realizada para cada parada de origem e destino identificadas através da lista de paradas obtidas na pesquisa de acordo com o raio. Caso a consulta não localize ao

menos uma linha, o algoritmo A* é executado a partir da parada de origem, sendo provável neste caso a necessidade de ao menos um transbordo. A busca direta é utilizada não apenas de forma isolada, mas também em conjunto com o algoritmo A*, sendo aplicada não só entre o vértice de origem e o vértice de destino como também entre vértices intermediários com o mesmo objetivo da consulta realizada inicialmente. Esta técnica é semelhante a estratégia de decomposição do problema de busca. A Figura 26 ilustra um exemplo da realização da busca direta.

Figura 26 – Exemplo de realização da busca direta.



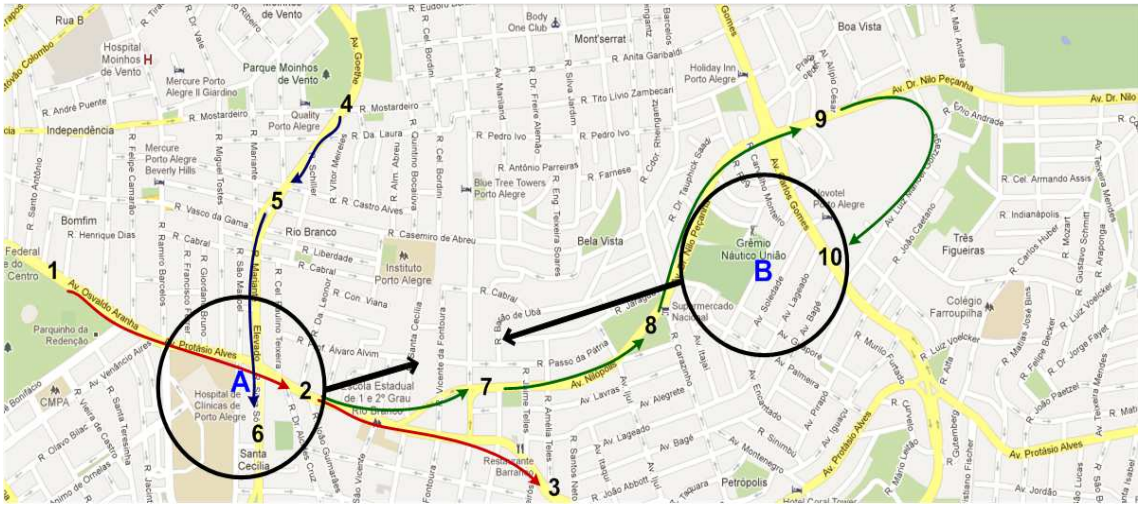
Fonte: GoogleMaps (2012), adaptado pelo autor.

A Figura 26 utiliza o mesmo exemplo de grafo apresentado na Figura 22 e na Figura 24. Os números representam as paradas e as linhas, nas cores vermelho, verde e azul representam o percurso das linhas de ônibus. Através do raio, foram identificadas as paradas 2 e 6 como paradas de origem e a parada 10 como parada de destino. A busca direta, representada na figura pelas linhas pontilhadas, é realizada utilizando a parada 2 como origem e a parada 10 como destino e também utilizando a parada 6 como origem e parada 10 como destino. No exemplo apresentado, a busca direta conseguirá localizar um caminho direto existente entre a parada de origem 2 e a parada de destino 10. A busca direta identifica este possível caminho através das arestas de saída do vértice de origem e das arestas de entrada do vértice de destino. Caso exista uma aresta de saída do vértice de origem e uma aresta de entrada do vértice de destino com uma linha de ônibus em comum é possível concluir que existe uma linha que percorre diretamente ambas as paradas.

A busca bidirecional é uma estratégia de decomposição do problema de busca que divide o procedimento em dois sub processos, um realizado da origem para o destino e outro

realizado do destino para a origem (FU; SUN; RILETT, 2006). A busca bidirecional aplicada no algoritmo A* é possível através das informações previstas na modelagem dos dados. A tabela denominada *Linhas_Paradas* possui referências tanto para a parada posterior como para a parada anterior, fazendo com que seja possível a aplicação desta técnica no algoritmo. A Figura 27 ilustra um exemplo da realização da busca bidirecional.

Figura 27 – Exemplo de realização da busca bidirecional.



Fonte: GoogleMaps (2012), adaptado pelo autor.

A Figura 27 também utiliza o mesmo exemplo de grafo apresentado na Figura 22 e Figura 24. Neste caso, a busca bidirecional é realizada utilizando a parada 2 como origem e a parada 10 como destino e também utilizando a parada 6 como origem e parada 10 como destino. Entretanto, em cada processo de busca, duas buscas simultâneas são realizadas: *(i)* uma direta a partir da parada de origem e *(ii)* outra inversa a partir da parada de destino. Caso estas duas buscas se encontrem em um estado intermediário é possível determinar um caminho entre a parada de origem e a parada de destino.

A combinação das técnicas de aceleração aplicadas no algoritmo A*, permitiu a geração de quatro versões do algoritmo de busca de rotas: A* unidirecional, A* bidirecional, A* unidirecional direto e A* bidirecional direto. Estas quatro versões do algoritmo de busca de rotas serão avaliadas no capítulo 7.

6.5.5 Tratamento dos Resultados

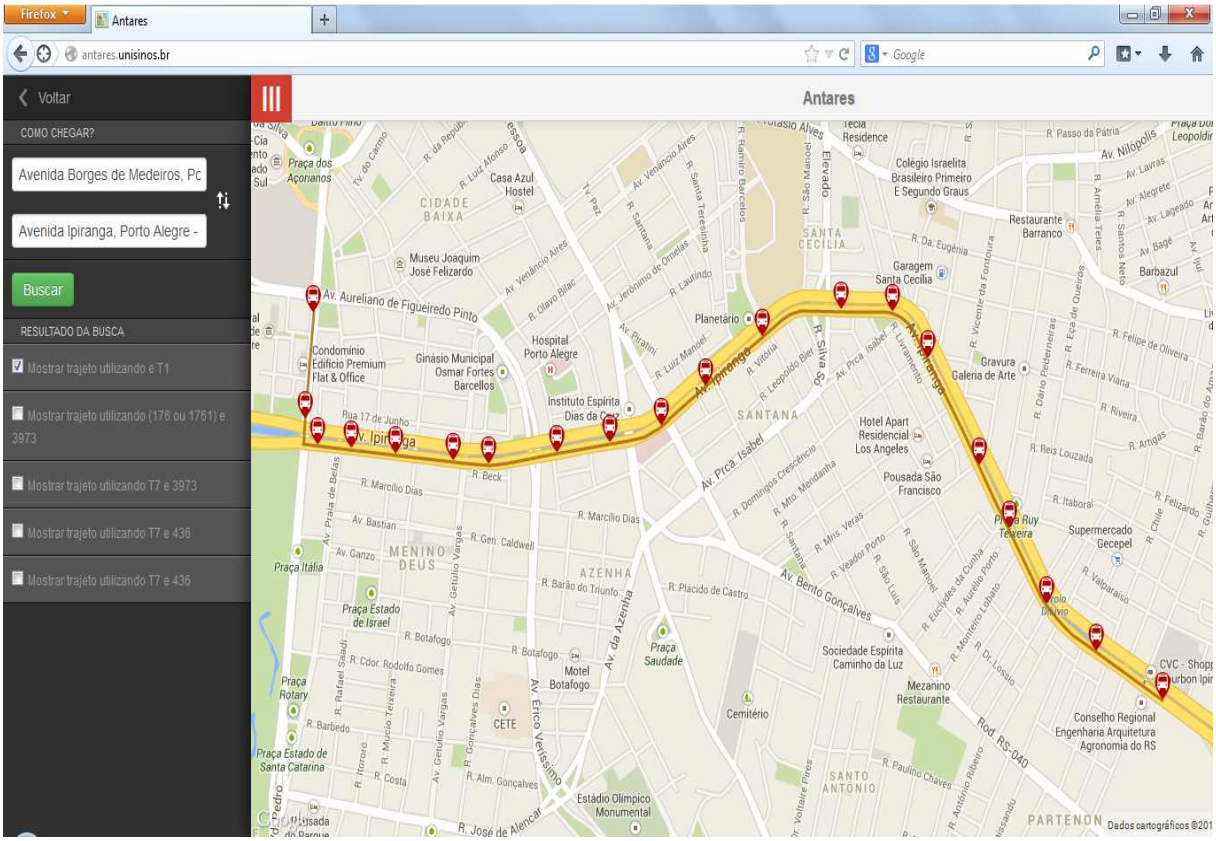
O último procedimento realizado pelo algoritmo de roteamento é o tratamento dos resultados. Tendo em vista que várias rotas podem ser identificadas em razão das diferentes

combinações de paradas de origem e destino avaliadas pelo algoritmo de busca de rotas, é necessário realizar uma filtragem dos resultados, de maneira que apenas as melhores opções sejam sugeridas ao usuário. Para isto, cada possível rota identificada pelo algoritmo é armazenada em uma lista. Ao término das avaliações de cada combinação de paradas, os resultados são ordenados de acordo com os critérios de custo considerados, como a distância total percorrida e o número de transbordos. O número de resultados retornados pelo algoritmo de roteamento é fixado em cinco resultados, mas por se tratar de um parâmetro do modelo, pode ser alterado conforme a necessidade.

6.6 Exemplo de Utilização do Modelo Computacional

Um exemplo da utilização do modelo computacional desenvolvido é descrito no trabalho de Abreu (2013). Neste trabalho, foi realizada a integração de uma interface gráfica com o modelo computacional. Um exemplo do resultado desta integração pode ser visualizado através Figura 28.

Figura 28 – Exemplo de interface gráfica integrada ao modelo computacional.



Fonte: Abreu (2013), adaptado pelo autor.

Na Figura 28, no lado esquerdo é possível visualizar dois campos de um formulário. Nestes campos é permitido informar um endereço de origem e um endereço de destino para realização da busca de opções de trajeto via transporte público. Ao acionar o botão *Buscar* na interface, é realizado primeiramente a conversão destes endereços em coordenadas geográficas, através de um serviço disponibilizado pelo Google Maps (2012), conhecido como geolocalização. Em seguida, a interface invoca o algoritmo de roteamento, passando os parâmetros de entrada necessários (latitude e longitude de origem, latitude e longitude de destino e distância máxima aproximada a ser percorrida a pé) através de uma chamada via *webservices*[1]. No exemplo, o algoritmo de roteamento retorna cinco opções de trajeto admissíveis. Na interface são exibidas informações detalhadas sobre cada uma das rotas possíveis incluindo localização das paradas, nomes de linhas e conexões necessárias para realização do percurso.

[1] *Webservices* é uma solução utilizada na integração de sistemas e na comunicação entre diferentes aplicações via protocolo HTTP. São componentes que permitem às aplicações enviar e receber dados em formato XML.

7 AVALIAÇÃO DO MODELO

Para avaliação do modelo computacional proposto, foi realizado um estudo de caso com dados provenientes da cidade de Porto Alegre. Os dados foram requisitados formalmente para a EPTC (2012) e disponibilizados impressos em papel. A partir de uma digitalização, foram gerados três arquivos *bus-stops.csv*, *routes.csv* e *routestops.csv*. O arquivo *bus-stops.csv* contém um identificador e também a latitude e longitude de cada parada de ônibus. O arquivo *routes.csv* possui um identificador e a descrição das linhas de ônibus e o arquivo *routestops.csv* contém a descrição das paradas que compõem cada linha de ônibus. Em resumo, esses arquivos possuem informações sobre 4.423 paradas, 757 linhas de ônibus, e 33.450 relações de linhas e paradas. Os dados provenientes dos arquivos foram importados para a base de dados e complementados com informações a respeito das distâncias. As distâncias reais entre as paradas foram calculadas e atualizadas após a importação dos dados, uma vez que estas não foram disponibilizadas pela EPTC[2].

Todas as avaliações foram realizadas em um computador Dell Vostro 3450 de 64-bits com processador Intel I5 de 2.4 GHz e 4 Gb de memória RAM e utilizando Windows 7 como sistema operacional. A implementação dos algoritmos foi realizada utilizando a linguagem de programação Java, versão 7 também de 64-bits. Como banco de dados foi utilizado PostgreSQL versão 9.1 para Windows. A seguir, é descrito e detalhado o processo de validação e avaliação das diferentes versões do algoritmo de busca de rotas.

7.1 Validação dos Algoritmos de Busca de Rotas

Para validação dos algoritmos de busca de rotas, foram realizadas comparações entre as duas versões do algoritmo A* (unidirecional e bidirecional) e o algoritmo Dijkstra puro. Para isto, foram geradas 1.000 instâncias do problema, onde cada instância consiste na escolha aleatória de uma parada de origem e uma parada de destino. A eficiência dos algoritmos, conforme já revisado na seção 4.2, é medida sob quatro aspectos, como completeza, otimização, complexidade de espaço e complexidade de tempo. Em razão disto, foram avaliados o custo da solução, o número de vértices expandidos e o tempo de

2 Alguns dados disponibilizados pela EPTC apresentavam informações incorretas de algumas linhas de ônibus e paradas e, em razão disto, foram desconsideradas nas avaliações.

processamento de cada algoritmo. Os resultados de cada comparação são apresentados a seguir. Para uma melhor visualização e interpretação, a ordem em que os resultados para cada instância são apresentados nos gráficos, bem como e as cores utilizadas na identificação de cada algoritmo, podem variar conforme o critério avaliado.

7.1.1 Custo da solução

A comparação relacionada ao custo da solução tem como objetivo avaliar o caráter completo e ótimo dos algoritmos. Para isto, para cada instância do problema gerada aleatoriamente, foi avaliada a distância total da solução gerada por cada algoritmo. O Gráfico 1 ilustra a comparação de custo das soluções obtidas com o emprego do algoritmo Dijkstra e o com o emprego do algoritmo A* unidirecional e o Gráfico 2 ilustra a comparação de custo das soluções obtidas com o emprego do algoritmo Dijkstra e com o emprego do algoritmo A* bidirecional.

Gráfico 1 – Comparação de custo entre Dijkstra e A* unidirecional.

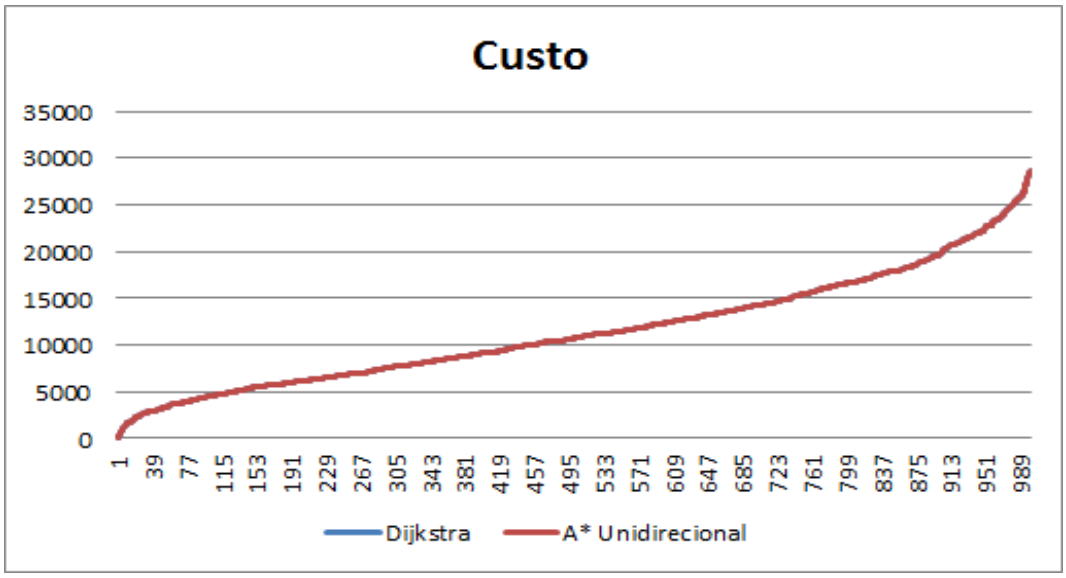
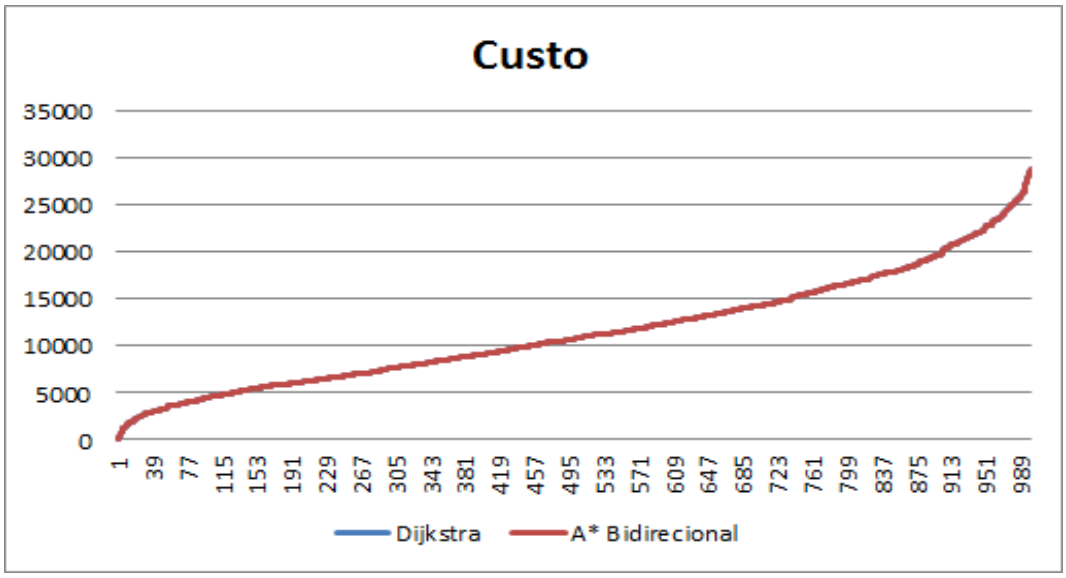


Gráfico 2 – Comparação de custo entre Dijkstra e A* bidirecional.



Nos gráficos, o eixo horizontal representa as instâncias dos problemas gerados aleatoriamente, ordenadas pela distância, e o eixo vertical representa o custo da solução, que corresponde à distância total percorrida em metros. Tanto o algoritmo A* unidirecional quanto o algoritmo A* bidirecional geraram soluções análogas ao algoritmo Dijkstra puro para todas as instâncias avaliadas. Desta maneira, é possível afirmar que as duas versões do algoritmo A* (unidirecional e bidirecional) são também ótimas e completas para o problema de caminho mínimo envolvendo rotas de transporte público.

7.1.2 Número de vértices expandidos

A comparação relacionada ao número de vértices expandidos tem como objetivo avaliar o desempenho dos algoritmos em relação à complexidade de espaço. Para isto, para cada instância do problema gerada aleatoriamente, foi medido o número de vértices expandidos por cada algoritmo, para encontrar a solução. O Gráfico 3 ilustra a comparação de número de vértices expandidos obtidos com o algoritmo Dijkstra e com o algoritmo A* unidirecional e o Gráfico 4 ilustra a comparação de número de vértices expandidos obtidos com o algoritmo Dijkstra e com o algoritmo A* bidirecional.

Gráfico 3 – Comparação de vértices expandidos entre Dijkstra e A* unidirecional.

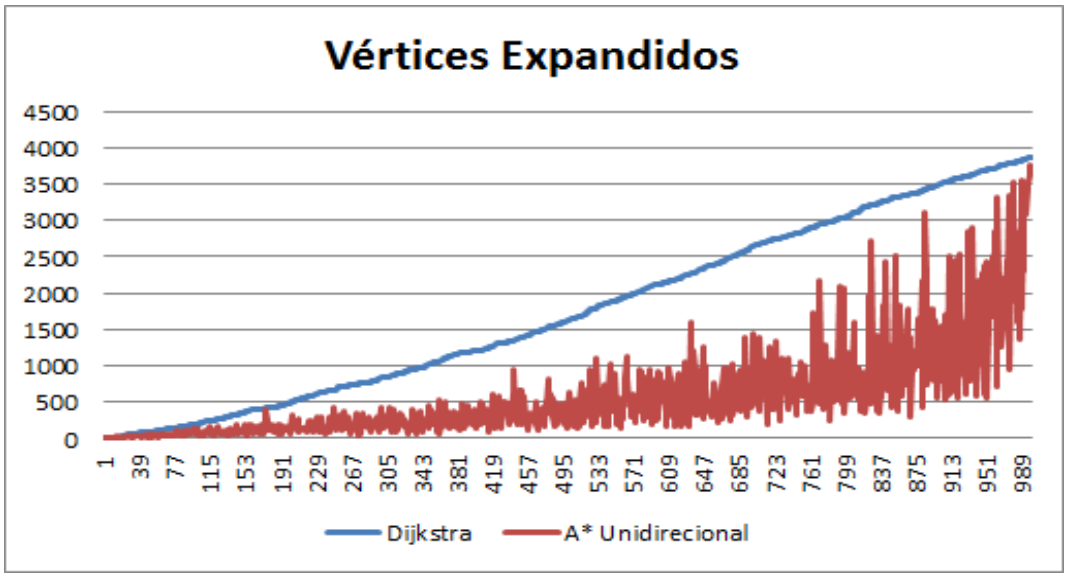
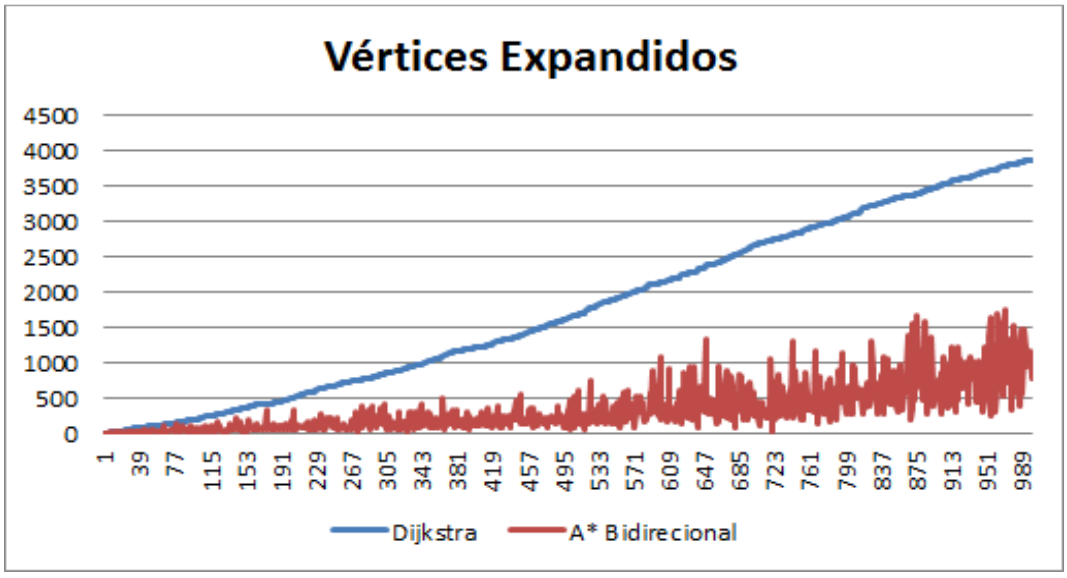


Gráfico 4 – Comparação de vértices expandidos entre Dijkstra e A* bidirecional.



Nos gráficos, o eixo horizontal representa as instâncias dos problemas gerados aleatoriamente e o eixo vertical representa o número de vértices expandidos. O algoritmo A* unidirecional expandiu um número de vértices inferior ao Dijkstra em 99,2% dos casos avaliados e a maior diferença identificada foi de 668,72% vértices expandidos a mais pelo algoritmo Dijkstra em relação ao algoritmo A* unidirecional. O algoritmo A* bidirecional expandiu um número de vértices inferior ao Dijkstra em 99,4% dos casos avaliados e a maior diferença identificada foi de 1.451,56% vértices expandidos a mais pelo algoritmo Dijkstra em relação ao algoritmo A* bidirecional.

7.1.3 Tempo de processamento

A comparação relacionada ao tempo de processamento tem como objetivo avaliar o desempenho dos algoritmos em relação à complexidade de tempo. Para isto, para cada instância do problema gerada aleatoriamente, foi medido o tempo gasto por cada algoritmo para encontrar a solução. O Gráfico 5 ilustra a comparação de tempo de processamento obtido com o emprego do algoritmo Dijkstra e com o emprego do algoritmo A* unidirecional e o Gráfico 6 ilustra a comparação de tempo de processamento obtido com o emprego do algoritmo Dijkstra e com o emprego do algoritmo A* bidirecional.

Gráfico 5 – Comparação de tempo de processamento entre Dijkstra e A* unidirecional.

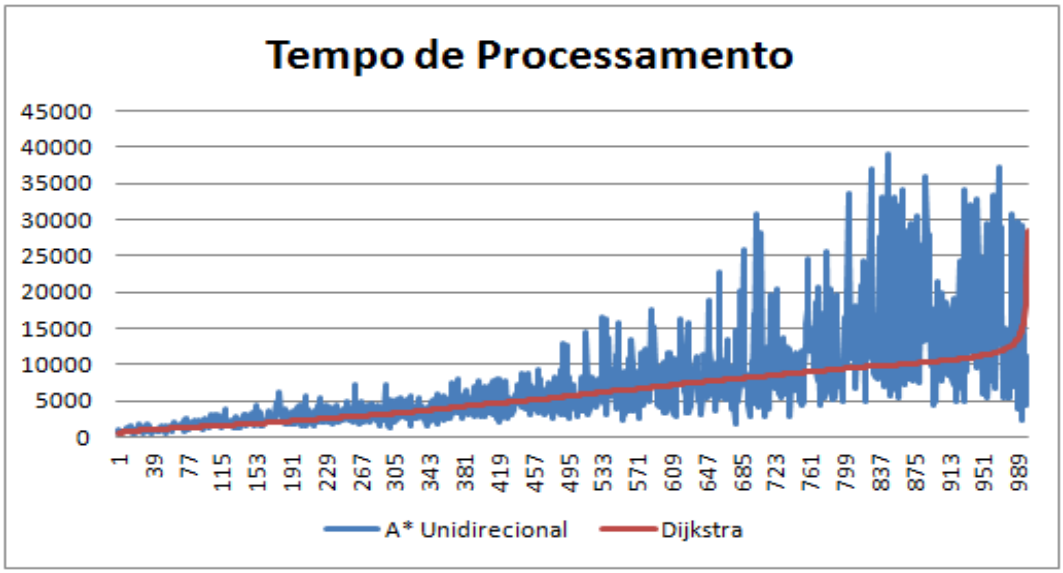
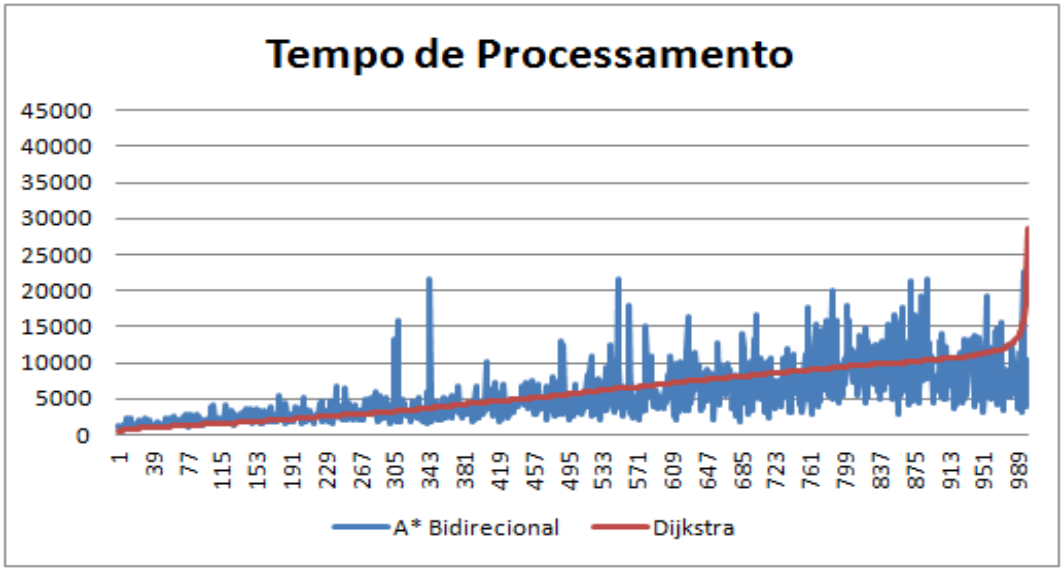


Gráfico 6 – Comparação de tempo de processamento entre Dijkstra e A* bidirecional.

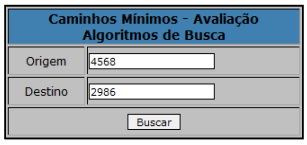


Nos gráficos, o eixo horizontal representa as instâncias dos problemas gerados aleatoriamente e o eixo vertical representa o tempo de processamento em microssegundos. O algoritmo A* unidirecional apresentou um tempo de processamento inferior ao Dijkstra em 41% dos casos avaliados. A maior diferença identificada, onde o Dijkstra teve um tempo de processamento inferior ao A* unidirecional, foi de 397,84%. No caso contrário, a maior diferença identificada, onde o A* unidirecional teve um tempo de processamento inferior ao Dijkstra, foi de 257,90%. O algoritmo A* bidirecional apresentou um tempo de processamento inferior ao Dijsktra em 56% dos casos avaliados. Para os casos em que o Dijkstra teve um tempo de processamento inferior ao A* bidirecional, a maior diferença identificada foi de 576,80%. Para o caso oposto (A* bidirecional teve um tempo de processamento inferior ao Dijkstra), a maior diferença identificada foi de 273,25%.

7.2 Avaliação dos Algoritmos de Busca de Rotas

Conforme visto na seção anterior, as duas versões do algoritmo A* (unidirecional e bidirecional) resolvem o problema de caminho mínimo de forma ótima e completa da mesma forma que o algoritmo Dijkstra, sendo possível inclusive obter uma significativa redução da complexidade de espaço e de tempo. Entretanto, nenhum dos algoritmos minimiza o número de transbordos sugeridos na sua solução. Quando o problema de caminho mínimo é definido em termos de transporte público, minimizar o número de transbordos passa a ser tão importante quanto minimizar o custo do caminho. Isto porque, em uma situação real, cada transbordo necessário implica em um custo adicional ao usuário, correspondente ao valor da passagem e também em tempo gasto na realização do transbordo. A Figura 29 ilustra um exemplo de número de transbordos sugeridos pelos algoritmos em sua solução de rota.

Figura 29 – Exemplo de número de transbordos.



Resultados localizados:

Dijkstra	A* - Unidirecional	A* - Bidirecional	A* - Unidirecional Direto	A* - Bidirecional Direto
Expandidos = 1497 Duração = 5,1026432E7 Custo Real = 7.890,93566 Profundidade = 35 **Quantidade Linhas = 13**	Expandidos = 235 Duração = 2,4950336E7 Custo Real = 7.890,93566 Profundidade = 35 **Quantidade Linhas = 13**	Expandidos = 152 Duração = 1,6961297E7 Custo Real = 7.890,93566 Profundidade = 34 **Quantidade Linhas = 10**	Expandidos = 1 Duração = 3.943.154,00 Custo Real = 8.683,63677 Profundidade = 27 **Quantidade Linhas = 1**	Expandidos = 1 Duração = 8.073.363,00 Custo Real = 8.683,63677 Profundidade = 27 **Quantidade Linhas = 1**
Parada = 4568 - Linha = Parada = 732 - Linha = 243 Parada = 1721 - Linha = 87 Parada = 3749 - Linha = 87 Parada = 1707 - Linha = 87 Parada = 1745 - Linha = 87 Parada = 1709 - Linha = 330 Parada = 1713 - Linha = 455 Parada = 1717 - Linha = 455 Parada = 1716 - Linha = 455 Parada = 1744 - Linha = 87 Parada = 1715 - Linha = 87 Parada = 446 - Linha = 330 Parada = 447 - Linha = 330 Parada = 1724 - Linha = 87 Parada = 1742 - Linha = 87 Parada = 1722 - Linha = 202 Parada = 1710 - Linha = 431 Parada = 1110 - Linha = 88 Parada = 1708 - Linha = 340 Parada = 3026 - Linha = 405 Parada = 2133 - Linha = 340 Parada = 1159 - Linha = 340 Parada = 552 - Linha = 346 Parada = 553 - Linha = 343 Parada = 551 - Linha = 340 Parada = 3016 - Linha = 340 Parada = 332 - Linha = 476 Parada = 838 - Linha = 476 Parada = 348 - Linha = 436 Parada = 2995 - Linha = 436 Parada = 2992 - Linha = 405 Parada = 763 - Linha = 405 Parada = 2985 - Linha = 405 Parada = 2986 - Linha = 405	Parada = 4568 - Linha = Parada = 732 - Linha = 243 Parada = 1721 - Linha = 87 Parada = 3749 - Linha = 87 Parada = 1707 - Linha = 87 Parada = 1745 - Linha = 87 Parada = 1709 - Linha = 330 Parada = 1713 - Linha = 455 Parada = 1717 - Linha = 455 Parada = 1716 - Linha = 455 Parada = 1744 - Linha = 87 Parada = 1715 - Linha = 87 Parada = 446 - Linha = 330 Parada = 447 - Linha = 330 Parada = 1724 - Linha = 87 Parada = 1742 - Linha = 87 Parada = 1722 - Linha = 202 Parada = 1710 - Linha = 431 Parada = 1110 - Linha = 88 Parada = 1708 - Linha = 340 Parada = 3026 - Linha = 405 Parada = 2133 - Linha = 340 Parada = 1159 - Linha = 340 Parada = 552 - Linha = 346 Parada = 553 - Linha = 343 Parada = 551 - Linha = 340 Parada = 3016 - Linha = 340 Parada = 332 - Linha = 476 Parada = 838 - Linha = 476 Parada = 348 - Linha = 436 Parada = 2995 - Linha = 436 Parada = 2992 - Linha = 405 Parada = 763 - Linha = 405 Parada = 2985 - Linha = 405 Parada = 2986 - Linha = 405	Parada = 4568 - Linha = Parada = 732 - Linha = 243 Parada = 1721 - Linha = 87 Parada = 3749 - Linha = 87 Parada = 1707 - Linha = 87 Parada = 1745 - Linha = 87 Parada = 1709 - Linha = 330 Parada = 1713 - Linha = 455 Parada = 1717 - Linha = 455 Parada = 1716 - Linha = 455 Parada = 1744 - Linha = 87 Parada = 1715 - Linha = 87 Parada = 446 - Linha = 330 Parada = 447 - Linha = 330 Parada = 1724 - Linha = 87 Parada = 1742 - Linha = 87 Parada = 1710 - Linha = 88 Parada = 1110 - Linha = 87 Parada = 1708 - Linha = 340 Parada = 3026 - Linha = 405 Parada = 2133 - Linha = 340 Parada = 1159 - Linha = 340 Parada = 552 - Linha = 343 Parada = 553 - Linha = 343 Parada = 551 - Linha = 340 Parada = 3016 - Linha = 340 Parada = 332 - Linha = 476 Parada = 838 - Linha = 476 Parada = 348 - Linha = 436 Parada = 2995 - Linha = 436 Parada = 2992 - Linha = 405 Parada = 763 - Linha = 405 Parada = 2985 - Linha = 405 Parada = 2986 - Linha = 405	Parada = 4568 - Linha = Parada = 732 - Linha = 430 Parada = 994 - Linha = 430 Parada = 637 - Linha = 430 Parada = 635 - Linha = 430 Parada = 633 - Linha = 430 Parada = 631 - Linha = 430 Parada = 757 - Linha = 430 Parada = 630 - Linha = 430 Parada = 628 - Linha = 430 Parada = 623 - Linha = 430 Parada = 1945 - Linha = 430 Parada = 626 - Linha = 430 Parada = 625 - Linha = 430 Parada = 1373 - Linha = 430 Parada = 3048 - Linha = 430 Parada = 3046 - Linha = 430 Parada = 1739 - Linha = 430 Parada = 3021 - Linha = 430 Parada = 824 - Linha = 430 Parada = 2999 - Linha = 430 Parada = 2113 - Linha = 430 Parada = 352 - Linha = 430 Parada = 2992 - Linha = 430 Parada = 763 - Linha = 430 Parada = 2985 - Linha = 430 Parada = 2986 - Linha = 430	Parada = 4568 - Linha = Parada = 732 - Linha = 430 Parada = 994 - Linha = 430 Parada = 637 - Linha = 430 Parada = 635 - Linha = 430 Parada = 633 - Linha = 430 Parada = 631 - Linha = 430 Parada = 757 - Linha = 430 Parada = 630 - Linha = 430 Parada = 628 - Linha = 430 Parada = 623 - Linha = 430 Parada = 1945 - Linha = 430 Parada = 626 - Linha = 430 Parada = 625 - Linha = 430 Parada = 1373 - Linha = 430 Parada = 3048 - Linha = 430 Parada = 3046 - Linha = 430 Parada = 1739 - Linha = 430 Parada = 3021 - Linha = 430 Parada = 824 - Linha = 430 Parada = 2999 - Linha = 430 Parada = 2113 - Linha = 430 Parada = 352 - Linha = 430 Parada = 2992 - Linha = 430 Parada = 763 - Linha = 430 Parada = 2985 - Linha = 430 Parada = 2986 - Linha = 430

Fonte: Elaborado pelo autor.

Na Figura 29, é apresentado um formulário com os campos origem e destino. O campo origem está preenchido com o valor 4568 e o campo destino está preenchido com o valor 2986, correspondendo respectivamente a uma parada de origem e a uma parada de destino, definidas de maneira não aleatória. A tabela intitulada **Resultados localizados** apresenta o resultado do caminho gerado por cada algoritmo (Dijkstra, A* unidirecional, A* bidirecional, A* unidirecional direto e A* bidirecional direto). Através da informação da quantidade de linhas necessárias em cada solução, presente na figura, é possível identificar uma redução no número de transbordos nas duas versões do algoritmo A* que são combinadas com a busca direta, apesar do custo do caminho (custo real na figura), nestes casos, ser maior que os demais algoritmos.

Conforme visto na seção 6.5.4, a busca direta é empregada originalmente para acelerar a busca de rotas, entretanto, sua função é mais ampla, pois permite também reduzir o número de transbordos. Nesta seção, são apresentados os resultados da comparação entre as duas versões do algoritmo A* (unidirecional direto e bidirecional direto) com relação ao algoritmo Dijkstra puro. Para isto, foram gerados aleatoriamente 1.000 instâncias do problema, onde cada instância consiste na escolha aleatória de uma parada de origem e uma parada de destino.

Em razão disto, foram avaliados o número de transbordos, o custo da solução, o número de vértices expandidos e o tempo de processamento de cada algoritmo. Os resultados de cada comparação são apresentados a seguir. Para uma melhor visualização e interpretação, a ordem em que os resultados para cada instância são apresentados nos gráficos, bem como e as cores utilizadas na identificação de cada algoritmo, podem variar conforme o critério avaliado.

7.2.1 Número de Transbordos

A avaliação relacionada ao número de transbordos tem como objetivo avaliar as soluções geradas pelos algoritmos quanto ao número de linhas de ônibus necessárias para realização do percurso sugerido. Para isto, para cada instância do problema gerada aleatoriamente, foi medido o número de transbordos sugeridos por cada algoritmo. O Gráfico 7 ilustra a comparação do número de transbordos obtidos com o emprego do algoritmo Dijkstra e do algoritmo A* unidirecional e o Gráfico 8 ilustra a comparação do número de transbordos obtidos com o emprego do algoritmo Dijkstra e do algoritmo A* bidirecional. O Gráfico 9 ilustra a comparação do número de transbordos obtidos com o emprego do algoritmo Dijkstra e com o emprego do algoritmo A* unidirecional direto e o Gráfico 10 ilustra a comparação do número de transbordos obtidos com o emprego do algoritmo Dijkstra e com o emprego do algoritmo A* bidirecional direto.

Gráfico 7 – Comparação de transbordos entre o Dijkstra e A* unidirecional.

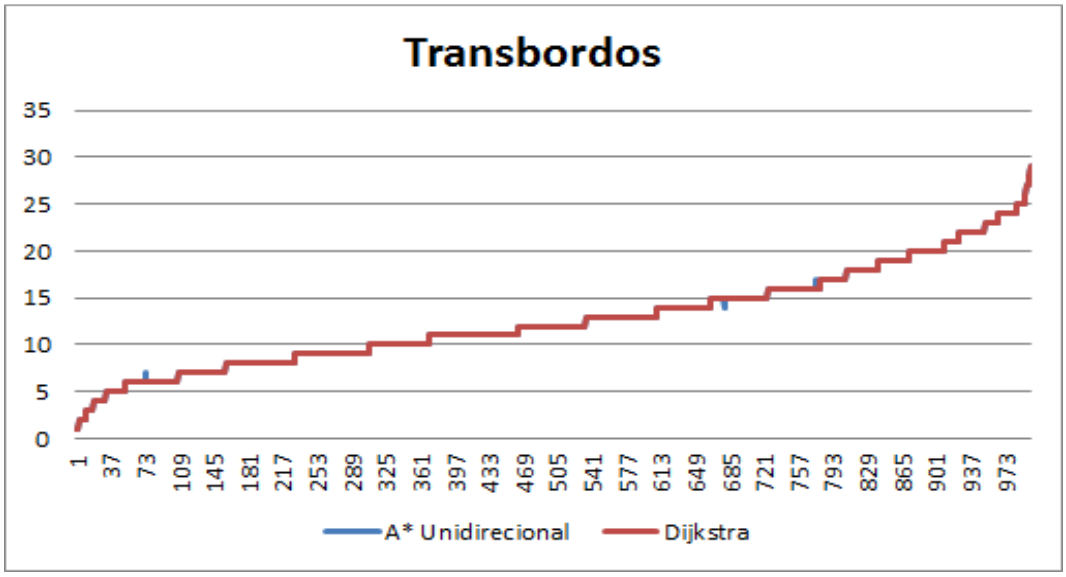


Gráfico 8 – Comparação de transbordos entre Dijkstra e A* bidirecional.

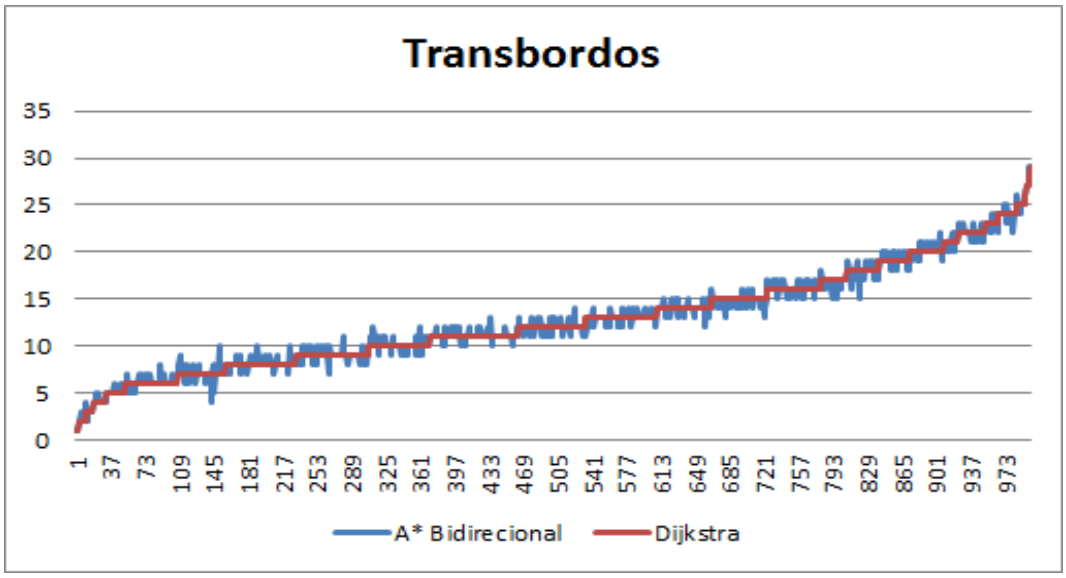


Gráfico 9 – Comparação de transbordos entre Dijkstra e A* unidirecional direto.

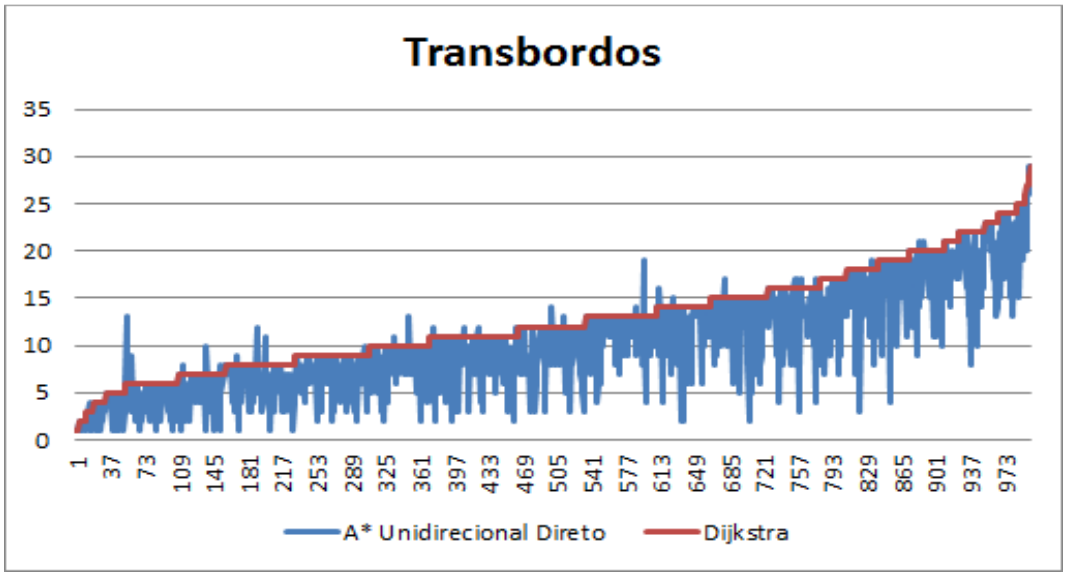
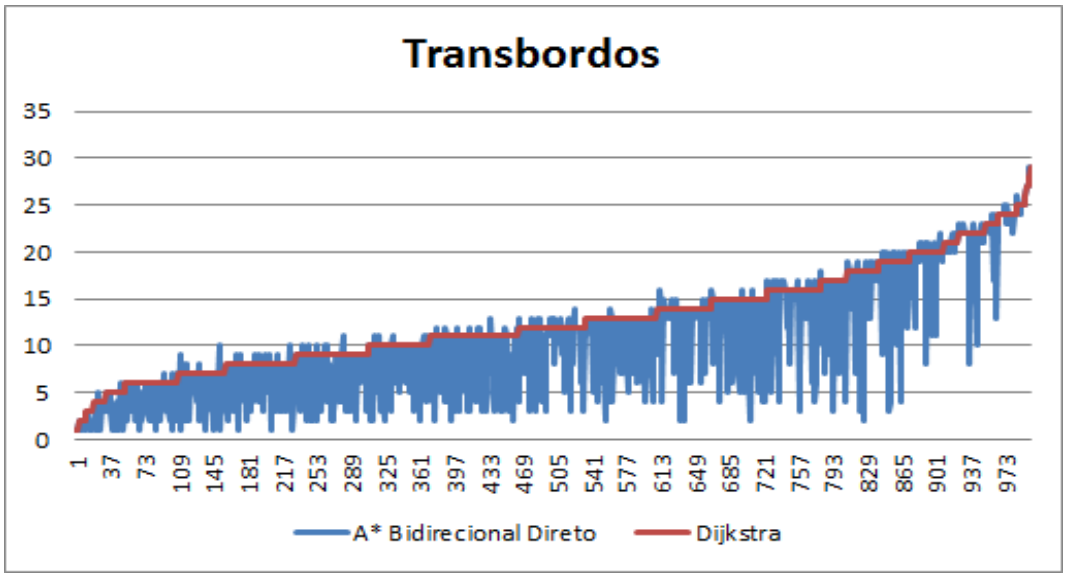


Gráfico 10 – Comparação de transbordos entre Dijkstra e A* bidirecional direto.



Nos gráficos, o eixo horizontal representa as instâncias dos problemas gerados aleatoriamente e o eixo vertical representa o número de transbordos. Em 99,7% dos casos, o algoritmo A* unidirecional sugere o mesmo número de transbordos que o Dijkstra. O algoritmo A* bidirecional sugere o mesmo número de transbordos que o Dijsktra em 67,7% dos casos e em outros 15,5% dos casos, sugere um número de transbordos inferior ao Dijkstra. O algoritmo A* unidirecional direto sugere o mesmo número de transbordos que o Dijkstra em 26,7% dos casos e em outros 70,1% dos casos, sugere um número de transbordos inferior ao Dijkstra. O algoritmo A* bidirecional direto sugere o mesmo número de transbordos que o Dijkstra em 38,9% dos casos e em outros 49,8% dos casos, sugere um número de transbordos inferior ao Dijkstra.

7.2.2 Custo da solução

Conforme visto na seção anterior, as duas versões do algoritmo A* (unidirecional direto e bidirecional direto) foram capazes de reduzir o número de transbordos. Entretanto, nestes casos, os algoritmos deixam de apresentar soluções ótimas, pois relaxam a restrição do custo em função desta necessidade. Para avaliar o custo destas soluções em relação à solução ótima, para cada instância do problema gerada aleatoriamente foi avaliado a distância total da solução gerada por cada algoritmo em relação ao algoritmo Dijkstra. O Gráfico 11 ilustra a comparação de custo das soluções obtidas com o emprego do algoritmo Dijkstra e com o emprego do algoritmo A* unidirecional direto e o Gráfico 12 ilustra a comparação de custo

das soluções obtidas com o emprego do algoritmo Dijkstra e com o emprego do algoritmo A* bidirecional direto.

Gráfico 11 – Comparação de custo entre Dijkstra e A* unidirecional direto.

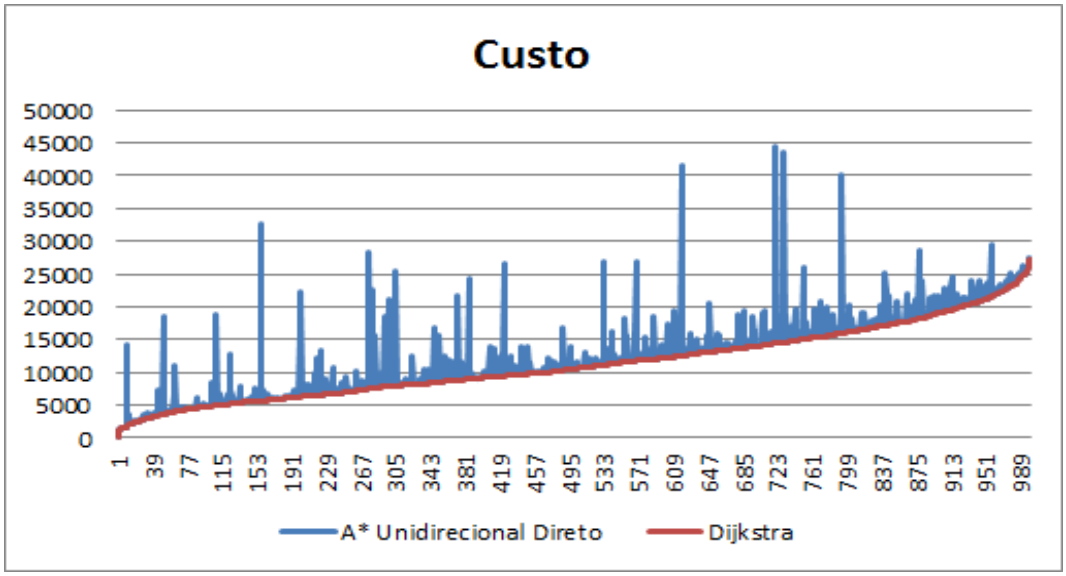
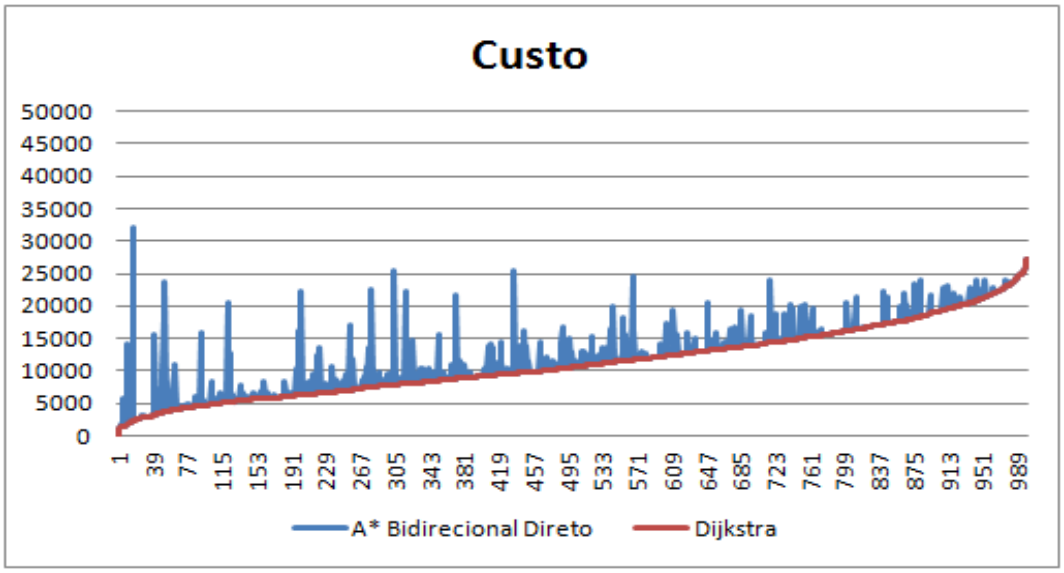


Gráfico 12 – Comparação de custo entre Dijkstra e A* bidirecional direto.



Nos gráficos, o eixo horizontal representa as instâncias dos problemas gerados aleatoriamente, ordenadas pela distância, e o eixo vertical representa o custo da solução, que corresponde à distância total percorrida em metros. O algoritmo A* unidirecional direto gerou uma solução análoga ao Dijkstra em 42,7% dos casos e o algoritmo A* bidirecional direto gerou uma solução análoga ao Dijkstra em 66,1% dos casos. Com isto, é possível concluir que as duas versões do algoritmo A* (unidirecional direto e bidirecional direto) são também

completas, mas não ótimas para o problema de caminho mínimo envolvendo rotas de transporte público em razão da necessidade de redução do número de transbordos.

7.2.3 Número de vértices expandidos

A avaliação relacionada ao número de vértices expandidos tem como objetivo avaliar o desempenho das duas versões do algoritmo A* (unidirecional direto e bidirecional direto) em relação à complexidade de espaço. Para isto, para cada instância gerada aleatoriamente, foi medido o número de vértices expandidos por cada algoritmo para encontrar a solução. O Gráfico 13 ilustra a comparação de número de vértices expandidos obtidos com o algoritmo Dijkstra e com o algoritmo A* unidirecional direto e o Gráfico 14 ilustra a comparação de número de vértices expandidos obtidos com o algoritmo Dijkstra e com o algoritmo A* bidirecional direto.

Gráfico 13 – Comparação de vértices expandidos entre Dijkstra e A* unidirecional direto.

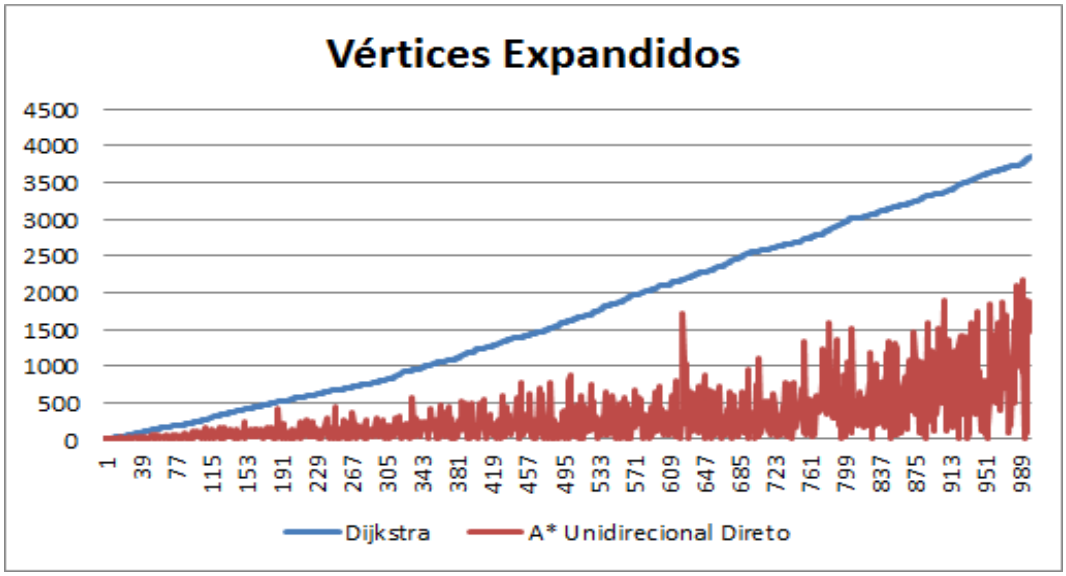
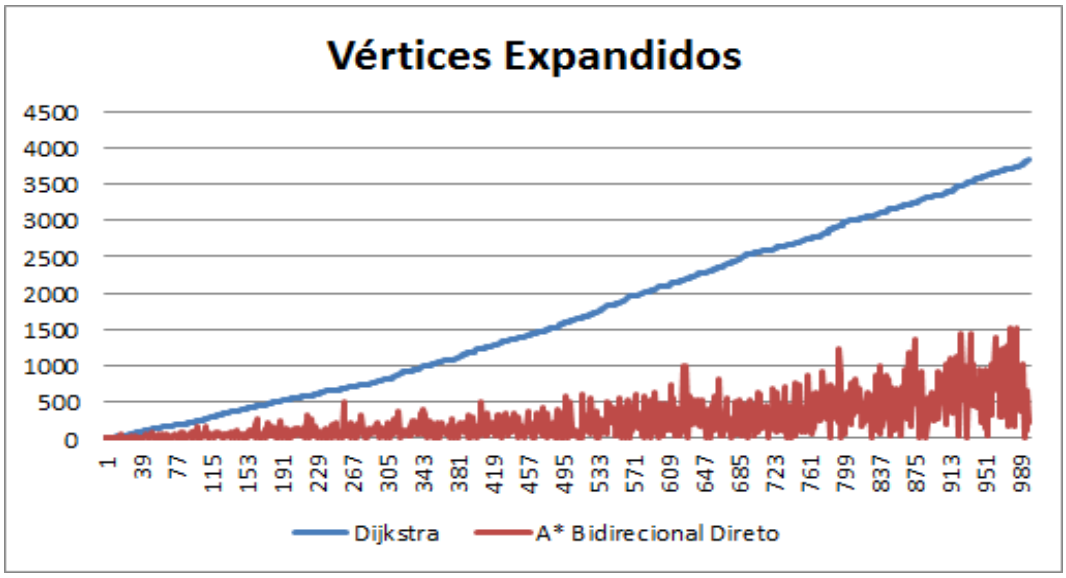


Gráfico 14 – Comparação de vértices expandidos entre Dijkstra e A* bidirecional direto.



Nos gráficos, o eixo horizontal representa as instâncias dos problemas gerados aleatoriamente e o eixo vertical representa o número de vértices expandidos. O algoritmo A* unidirecional direto expandiu um número de vértices inferior ao Dijkstra em 99,8% dos casos avaliados e a maior diferença identificada foi de 37.890% vértices expandidos a mais pelo algoritmo Dijkstra em relação ao A* unidirecional direto. O algoritmo A* bidirecional direto expandiu um número de vértices inferior ao Dijkstra em 99,7% dos casos avaliados e a maior diferença identificada foi de 19.942% vértices expandidos a mais pelo algoritmo Dijkstra em relação ao algoritmo A* bidirecional direto.

7.2.4 Tempo de processamento

A avaliação relacionada ao tempo de processamento tem como objetivo avaliar o desempenho das duas versões do algoritmo A* (unidirecional direto e bidirecional direto) em relação à complexidade de tempo. Para isto, para cada instância gerada aleatoriamente, foi medido o tempo gasto por cada algoritmo para encontrar a solução. O Gráfico 15 ilustra a comparação de tempo de processamento obtido com o emprego do algoritmo Dijkstra e com o emprego do algoritmo A* unidirecional direto e o Gráfico 16 ilustra a comparação de tempo de processamento obtido com o emprego do algoritmo Dijkstra e com o emprego do algoritmo A* bidirecional direto.

Gráfico 15 – Comparação de tempo de processamento entre Dijkstra e A* unidirecional direto.

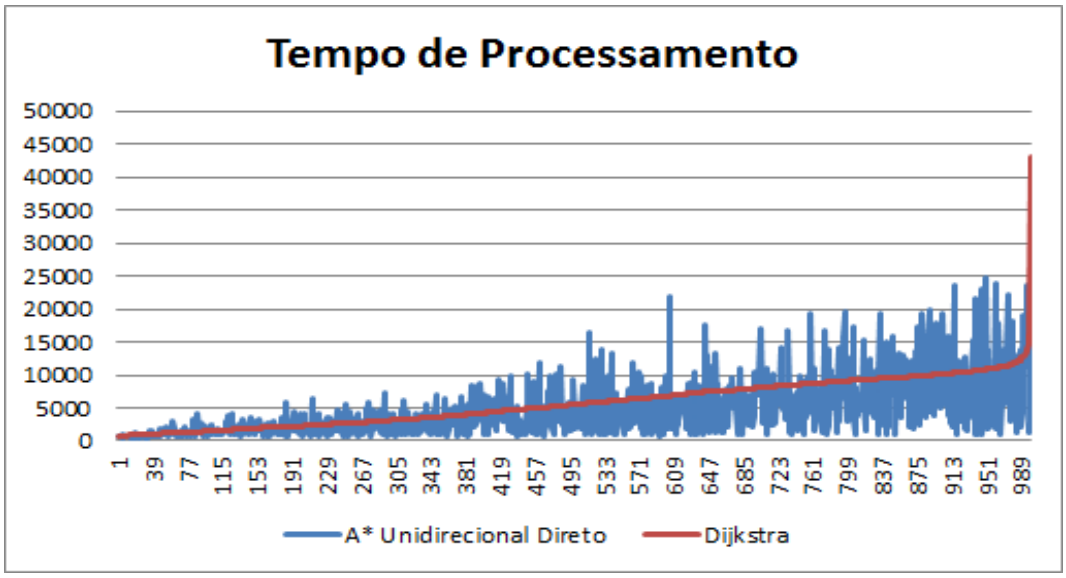
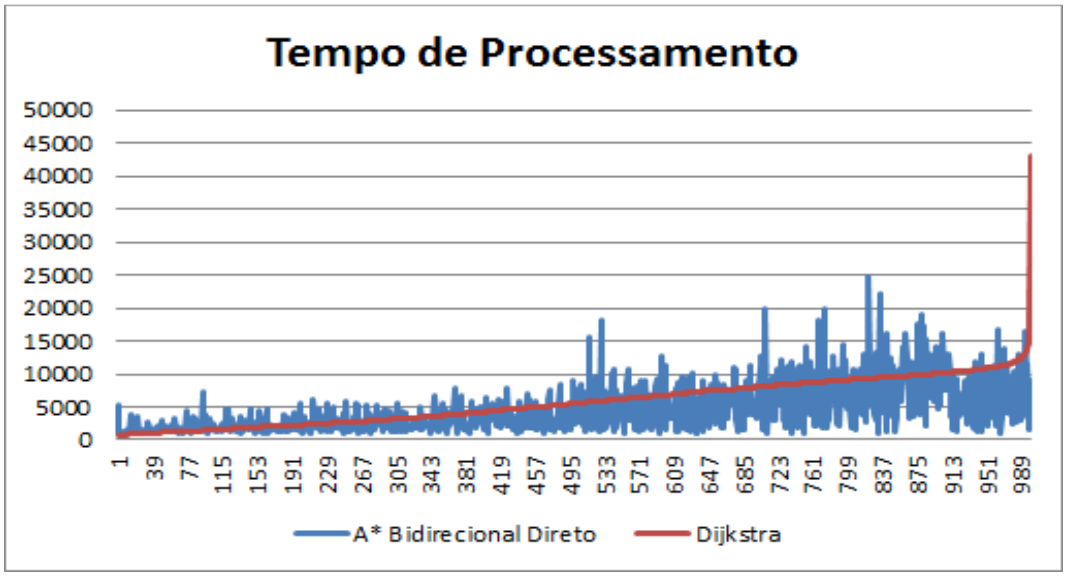


Gráfico 16 – Comparação de tempo de processamento entre Dijkstra e A* bidirecional direto.



Nos gráficos, o eixo horizontal representa as instâncias dos problemas gerados aleatoriamente e o eixo vertical representa o tempo de processamento em microssegundos. O algoritmo A* unidirecional direto apresentou um tempo de processamento inferior ao Dijkstra em 69,3% dos casos avaliados. A maior diferença identificada onde o Dijkstra teve um tempo de processamento inferior ao A* unidirecional direto foi de 320,76%. A maior diferença identificada onde o A* unidirecional direto teve um tempo de processamento inferior ao Dijkstra foi de 537,56%. O algoritmo A* bidirecional direto apresentou um tempo de processamento inferior ao Dijkstra em 64,5% dos casos avaliados. A maior diferença identificada onde o Dijkstra teve um tempo de processamento inferior ao A* bidirecional

direto foi de 263,64%. A maior diferença identificada onde o A* bidirecional direto teve um tempo de processamento inferior ao Dijkstra foi de 483,30%.

7.3 Avaliação dos Resultados

Nas seções anteriores foram demonstrados os resultados da avaliação realizada com dois algoritmos de busca de caminhos em grafo, o Dijkstra e o A* (unidirecional). A combinação de técnicas de aceleração como a busca bidirecional e a busca direta permitiram a geração e a avaliação de outras variações do algoritmo A*, como o A* bidirecional, o A* unidirecional direto e o A* bidirecional direto.

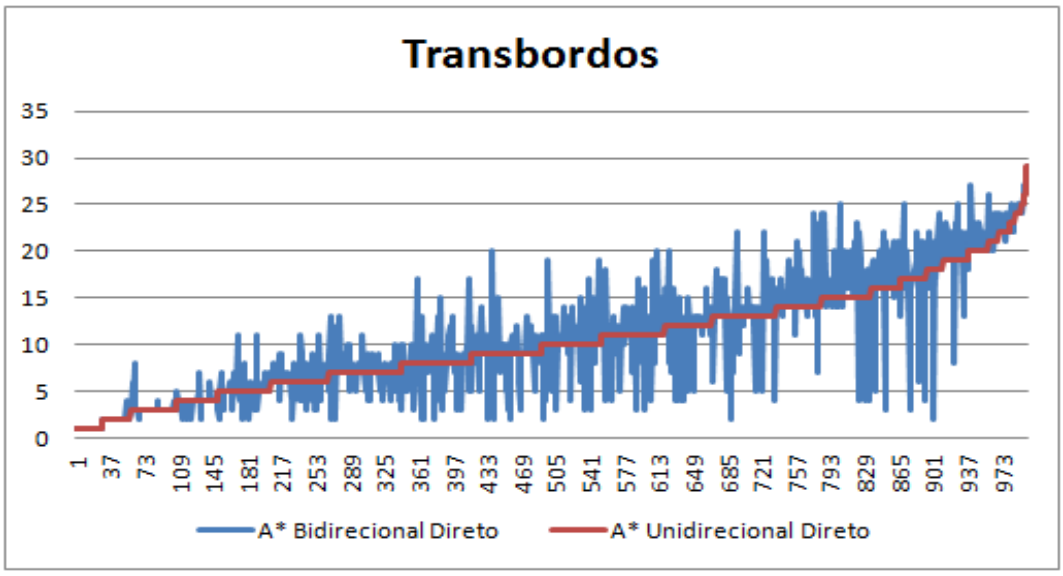
Os resultados da avaliação de duas versões do algoritmo A* (unidirecional e bidirecional) demonstraram que os algoritmos são ótimos e completos e por isto, podem ser utilizados para a determinação de caminhos mínimos em termos de transporte público. Além disto, a versão do algoritmo A* unidirecional reduziu a complexidade de espaço em mais de 99% dos casos avaliados e a complexidade de tempo em 41%. A versão do algoritmo A* bidirecional reduziu a complexidade de espaço em mais 99% dos casos avaliados e a complexidade de tempo em 56%.

Com o objetivo de acelerar o processo de busca e também reduzir o número de transbordos foram desenvolvidas e avaliadas duas outras versões algoritmo A*, a versão unidirecional direta e a versão bidirecional direta. Os resultados demonstraram que as duas versões do algoritmo são também completas e por isto, podem ser empregadas na determinação de caminhos mínimos através de transporte público. Estas versões, entretanto, não possuem um caráter ótimo, em relação ao custo do caminho, em função da necessidade de redução do número de transbordos. Os resultados indicaram que o algoritmo A* unidirecional direto sugere um número de transbordos inferior ao Dijkstra em 70,1% dos casos avaliados e o algoritmo bidirecional direto sugere um número de transbordos inferior ao Dijkstra em 49,8% dos casos avaliados. Os resultados demonstraram também que a versão unidirecional direta reduziu a complexidade de espaço em mais de 99% dos casos avaliados e a complexidade de tempo em 69,3% enquanto a versão bidirecional direta reduziu a complexidade de espaço em mais de 99% dos casos avaliados e a complexidade de tempo em 64,5% em comparação ao Dijkstra.

7.3.1 Escolha do algoritmo de busca de rotas

Para que seja possível definir o algoritmo mais adequado para ser utilizado na busca de rotas de transporte público, foram realizadas comparações entre as duas versões de melhor desempenho, conforme os resultados apresentados nas seções anteriores. Para isto, foram realizadas comparações entre as duas versões do algoritmo A* (unidirecional direto e bidirecional direto). Foram gerados 1.000 instâncias do problema, onde cada instância consiste na escolha aleatória de uma parada de origem e uma parada de destino. A eficiência dos algoritmos foi avaliada em razão do número de transbordos, o custo da solução, o número de vértices expandidos e o tempo de processamento de cada algoritmo. O Gráfico 17 ilustra a comparação do número de transbordos obtidos com o emprego do algoritmo A* unidirecional direto e com o emprego do algoritmo A* bidirecional direto. O Gráfico 18 ilustra a comparação do custo das soluções obtidas com o emprego do algoritmo A* unidirecional direto e com o emprego do algoritmo A* bidirecional direto. O Gráfico 19 ilustra a comparação de número de vértices expandidos obtidos com o emprego do algoritmo A* unidirecional direto com o algoritmo A* bidirecional direto. O Gráfico 20 ilustra a comparação de tempo de processamento obtido com o emprego do algoritmo A* unidirecional direto e com o emprego do algoritmo A* bidirecional direto.

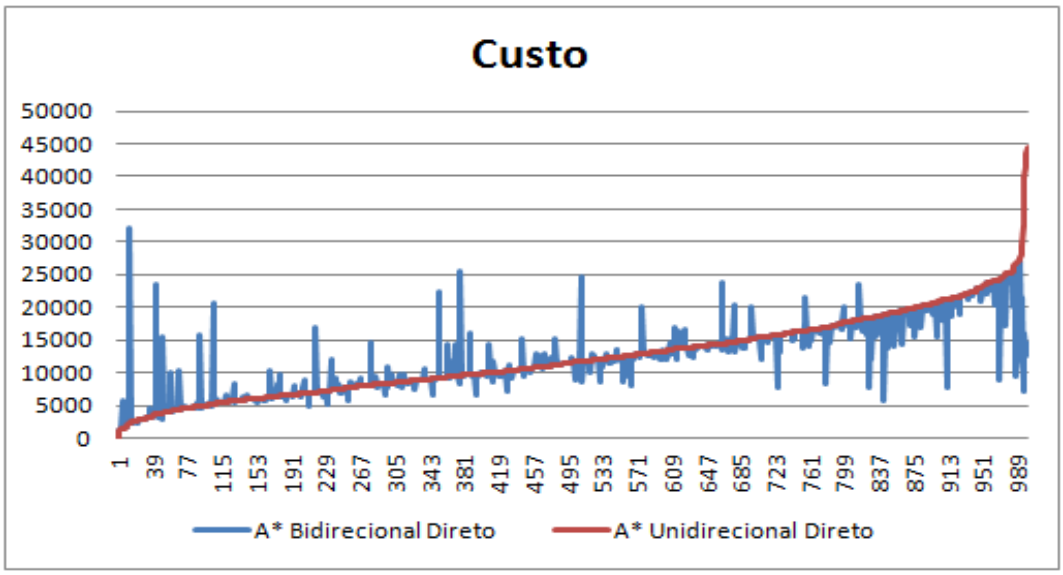
Gráfico 17 – Comparação de transbordos entre A* unidirecional direto e A* bidirecional direto.



No Gráfico 17, o eixo horizontal representa as instâncias dos problemas gerados aleatoriamente e o eixo vertical representa o número de transbordos. Em 42,3% dos casos, o algoritmo A* unidirecional direto sugere o mesmo número de transbordos que o algoritmo A*

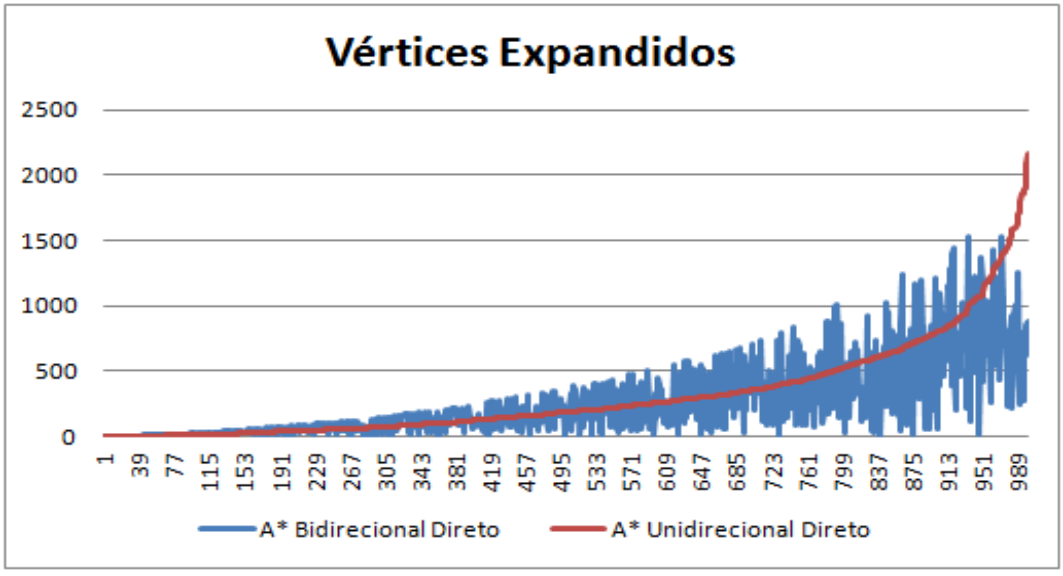
bidirecional direto e, em 39,1% dos casos, o algoritmo A* unidirecional direto sugere um número de transbordos inferior ao algoritmo A* bidirecional direto.

Gráfico 18 – Comparação de custo entre A* unidirecional direto e A* bidirecional direto.



No Gráfico 18, o eixo horizontal representa as instâncias dos problemas gerados aleatoriamente e o eixo vertical representa o custo da solução, que corresponde à distância total percorrida em metros. O algoritmo A* unidirecional direto gerou uma solução análoga ao algoritmo A* bidirecional direto em 57,6% dos casos e, 10,7% dos casos, o algoritmo A* unidirecional direto gerou uma solução inferior ao algoritmo A* bidirecional direto.

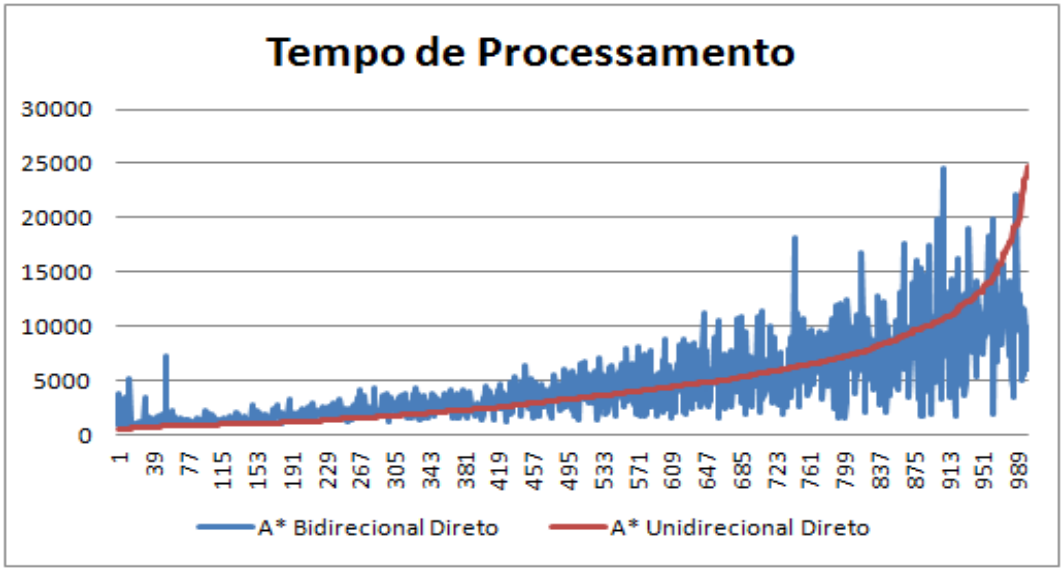
Gráfico 19 – Comparação de vértices expandidos entre A* unidirecional direto e A* bidirecional direto.



No Gráfico 19, o eixo horizontal representa as instâncias dos problemas gerados aleatoriamente e o eixo vertical representa o número de vértices expandidos. O algoritmo A*

unidirecional direto expandiu um número de vértices inferior ao algoritmo A* bidirecional direto em 50,3% dos casos avaliados e, em 46,7% dos casos, o algoritmo A* bidirecional direto expandiu um número de vértices inferior ao algoritmo A* unidirecional direto.

Gráfico 20 – Comparação de tempo de processamento entre A* unidirecional direto e A* bidirecional direto.



No Gráfico 20, o eixo horizontal representa as instâncias dos problemas gerados aleatoriamente e o eixo vertical representa o tempo de processamento em microssegundos. O algoritmo A* unidirecional direto apresentou um tempo de processamento inferior ao algoritmo A* bidirecional direto em 64,6% dos casos avaliados e, em 35,4% dos casos, o algoritmo A* bidirecional direto apresentou um tempo de processamento inferior ao algoritmo A* unidirecional direto.

Considerando como objetivo a redução do número de transbordos e também a redução do tempo de resposta, foi estabelecido como critérios de desempenho prioritários para o algoritmo o número de transbordos sugeridos e o tempo de processamento. De acordo com os resultados apresentados anteriormente, o algoritmo A* unidirecional direto sugere um número de transbordos menor ou igual ao A* bidirecional direto em 81,4% dos casos. Além disto, o algoritmo A* unidirecional direto obteve um tempo de processamento inferior ao A* bidirecional direto em 64,6% dos casos. Tendo em vista estes resultados, o algoritmo escolhido para a realização da busca de rotas do modelo computacional foi o algoritmo A* unidirecional direto.

8 CONCLUSÃO

No presente trabalho foram descritas todas as etapas necessárias para a construção de um modelo computacional genérico capaz de determinar caminhos mínimos em uma malha de transporte público segundo múltiplos critérios. O objetivo deste modelo é fornecer a possibilidade de busca de rotas através de transporte público, mediante a informação de um ponto de origem e de um ponto de destino. Com estas informações, o modelo computacional é capaz de determinar as melhores rotas, considerando como critérios a distância percorrida e o número de linhas utilizadas, e também fornecer informações detalhadas ao usuário sobre a rota, como por exemplo, linhas de ônibus, paradas e eventuais conexões.

Para a construção do referencial teórico, foi revisado, no capítulo 2, o estado da arte dos sistemas de transportes inteligentes e os conceitos básicos relacionados aos sistemas avançados de transporte público. Foram detalhadas as categorias existentes, os objetivos, funções e também tecnologias empregadas em sistemas existentes no Brasil e no mundo. No capítulo 3 foram revisados os principais conceitos relacionados à Teoria dos Grafos. Baseado nesta revisão, foi possível representar o problema corretamente através de grafos em função das diferentes categorias existentes e também realizar a escolha da representação computacional a ser utilizada no modelo computacional, a lista de adjacências. No capítulo 4 foram apresentados os principais conceitos relacionados ao problema de caminho mínimo e também as características e diferenças dos principais métodos de busca existentes. Com isto, o algoritmo A* foi escolhido, pois além de ser ótimo e completo, apresenta melhores resultados relacionados à complexidade de espaço e a complexidade de tempo em relação a algoritmos clássicos como o Dijkstra (FU; SUN; RILETT, 2006).

No capítulo 5 são apresentados trabalhos relacionados ao emprego de algoritmos de busca na identificação de caminhos mínimos em redes viárias e de transporte público. Nestes trabalhos, foram avaliados o tipo de modal considerado, algoritmo de busca e técnicas de aceleração utilizadas e critérios de custo utilizados para determinação de caminhos mínimos. A maioria dos trabalhos trata sobre a determinação de caminhos mínimos em redes viárias, embora existam trabalhos relacionados a caminhos mínimos em redes de transporte público. O algoritmo de busca mais utilizado é o Dijkstra e o algoritmo A* é utilizado em apenas um trabalho, sendo que nenhuns dos trabalhos propõem a utilização do A* para o transporte público. A maioria dos trabalhos também propõe algum tipo de melhoria nos algoritmos,

geralmente através de alguma técnica de aceleração já estabelecida. Os autores destacam a importância do uso destas técnicas quando a determinação de caminhos mínimos envolve problemas reais ou envolvendo grandes massas de dados, pois algoritmos clássicos como o Dijkstra, por exemplo, resolvem o problema de caminho mínimo, mas com um custo computacional muito elevado. Dentre os critérios de custo considerados, o tempo e a distância estão distribuídos de forma balanceada.

Ao longo deste trabalho, foram realizados alguns estudos relacionados ao problema da determinação de caminhos mínimos através de transporte público. Foi realizado o desenvolvimento de uma aplicação onde os algoritmos A* puro, busca em largura e Dijkstra foram utilizados na resolução de problemas envolvendo transporte público, a fim de comparar os resultados obtidos. A heurística utilizada para o algoritmo A* foi a distância linear e os critérios de comparação utilizados foram o número de vértices expandidos de cada algoritmo e tempo de processamento. Em relação ao número de vértices expandidos, o A* e o Dijkstra apresentaram resultados semelhantes e melhores do que o algoritmo de busca em largura. Em relação ao tempo de processamento, o algoritmo de busca em largura demonstrou melhores resultados do que os algoritmos A* e Dijkstra, que neste caso, apresentaram resultados semelhantes. A metodologia empregada foi a geração de 532 problemas randômicos baseados em dados reais da cidade de Porto Alegre, obtidos junto a EPTC (2012). Este estudo rendeu a publicação de um artigo intitulado *Um Estudo de Caso com o Uso do A* para o Problema do Caminho Mais Curto Em Transporte Público* na Conferência Brasileira de Sistemas Inteligentes, realizada em Curitiba em 2012 (JAQUES et al., 2012a).

Também foram realizados estudos a respeito das necessidades relacionadas à construção de sistemas de transportes inteligentes. O artigo intitulado *Provendo Informações para Atores do Sistema de Transporte Público: Um Passo na Direção de Sistemas de Transportes Inteligentes* tratou de questões como o papel dos sistemas de transportes inteligentes frente à crise de mobilidade urbana no cenário nacional. Foram descritas questões relacionadas ao uso das tecnologias da informação e comunicação como formas de tornar o transporte público urbano mais interessante e atraente ao usuário. Para isto, foram detalhados os requisitos de infraestrutura e informação para a construção de um ATIS e também de um sistema de monitoramento em frota de ônibus como soluções tecnológicas, tendo como objetivo facilitar o acesso, o entendimento e a melhora no uso e gerenciamento do transporte público urbano. O artigo foi publicado no XXVI ANPET – Congresso de Pesquisa e Ensino em Transportes, realizada em Joinville em 2012 (JAQUES et al., 2012b).

O modelo computacional desenvolvido pode ser descrito, basicamente, através de três etapas: *(i)* a modelagem da base de dados, *(ii)* a representação do problema através de grafos e *(iii)* a implementação de um algoritmo de roteamento. A etapa de modelagem dos dados consiste em armazenar e manipular os dados provenientes de transporte público em um banco de dados. A etapa de representação do problema consiste, primeiramente, em identificar o tipo de grafo mais adequado ao problema estando relacionada com a etapa de modelagem dos dados, pois é preciso identificar na modelagem qual entidade representa um vértice e qual entidade representa uma aresta. A terceira etapa consiste na implementação de um algoritmo de roteamento capaz de determinar rotas em transporte público. Entre outros procedimentos de responsabilidade do algoritmo de roteamento, o principal é o cálculo das rotas.

A combinação do algoritmo A* com técnicas de aceleração, como a busca bidirecional e a busca direta, permitiu a avaliação de quatro versões do algoritmo responsável pelo cálculo de rotas. Avaliações realizadas com a versão do algoritmo A* unidirecional e com o algoritmo A* bidirecional, além de demonstrar que os algoritmos são completos e ótimos, demonstraram que é possível reduzir a complexidade de espaço e de tempo. Entretanto, quando o problema de caminho mínimo está relacionado a transporte público, é necessário reduzir também o número de transbordos a fim de evitar custos desnecessários aos usuários. Em razão disto, foi desenvolvida uma técnica de aceleração que também é capaz de reduzir o número de transbordos, a chamada busca direta.

Com isto, foram geradas e avaliadas duas outras versões do algoritmo A*, a versão unidirecional direta e a versão bidirecional direta. Estas versões, apesar de não serem ótimas em relação ao custo do caminho, permitem reduzir significativamente o número de transbordos, o tempo de processamento e o número de vértices expandidos durante a busca. A partir dos estudos realizados, indica-se o uso do algoritmo A* unidirecional direto como algoritmo responsável pela busca de rotas no modelo computacional, em razão dos melhores resultados apresentados em comparação aos demais algoritmos avaliados.

Durante a construção do modelo computacional, foram identificadas algumas melhorias e aperfeiçoamentos que podem ser desenvolvidas futuramente de maneira a tornar o modelo mais rico e próximo das necessidades reais de um usuário do transporte público. Dentre as melhorias possíveis cabe destacar:

a) Tempo de viagem: ao invés de utilizar a distância total como custo, o tempo de viagem também poderia ser utilizado, através de uma abordagem expandida por tempo ou dependente de tempo. Embora a abordagem expandida por tempo seja mais

realista, segundo Delling, Pajor e Wagner (2008), a abordagem dependente de tempo pode ser facilmente incluída no modelo. Para isto, bastaria incluir na modelagem dos dados as informações de tempo relacionadas às paradas posteriores e anteriores de cada parada, de acordo com a respectiva linha.

b) Trajetos a serem percorridos a pé: atualmente o modelo não considera pequenos deslocamentos a pé por parte do usuário para ir de uma parada a outra. Caso não exista uma parada comum entre duas linhas consideradas durante a busca de rotas, o algoritmo não é capaz de sugerir um transbordo.

c) Previsão de tempo de espera: a previsão de tempo de espera para a chegada das linhas nas paradas é uma informação bastante útil ao usuário. Para isto, a partir da tabela de horários das linhas, poderia ser calculado um tempo aproximado para a chegada das linhas na parada desejada a partir de uma velocidade média considerada.

d) Limitação de número de transbordos: o algoritmo de roteamento poderia limitar o número de transbordos sugeridos descartando rotas muito dispendiosas aos usuários, já que dificilmente estas opções serão consideradas como viáveis do ponto de vista prático e financeiro.

Apesar das oportunidades de melhorias apresentadas, os resultados das avaliações demonstraram que o modelo computacional desenvolvido é válido e pode ser utilizado em uma situação real para determinação de caminhos mínimos em uma malha de transporte público. Por se tratar de um modelo genérico, pode ser utilizado para qualquer cidade ou região, bastando para isto, alimentar a base de dados com as informações de transporte público daquela localidade. Além disto, tecnologias como *webservices,* permitem integrar o modelo computacional a interfaces gráficas voltadas para *web* ou dispositivos móveis. As principais contribuições presentes neste trabalho são: *(i)* a aplicação do algoritmo A* para determinação de caminhos mínimos em transporte público, algo não visto nos trabalhos relacionados; e *(ii)* a implementação da técnica de aceleração denominada busca direta, que combinada com o algoritmo A* é capaz de reduzir o número de transbordos, a complexidade de tempo e a complexidade de espaço para o problema do caminho mínimo em uma malha de transporte público.

REFERÊNCIAS

ABREU, S. L. F. *Proposta de uma nova interface para o sistema de transporte inteligente Antares: Um estudo de usabilidade*. 2013. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação). Universidade do Vale do Rio dos Sinos. Orientador: Patrícia Augustin Jaques Maillard.

BAUER, R., DELLING, D., SANDERS, P., SCHIEFERDECKER, D., SCHULTES, D., and WAGNER, D. *Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm*. 2010. ACM J. Exp. Algor.15, Article 2.3 (February 2010), 31 pages.

BELEM. *Belém 400 anos*. 2012. Disponível em: <http://belem400.blogspot.com.br/2012/08/como-e-o-brt-de-curitiba-para-entender.html>. Acesso em: 21 outubro 2012.

BRASIL. Decreto-Lei nº 12.587, de 3 de janeiro de 2012. *Institui as diretrizes da Política Nacional de Mobilidade Urbana*. Diário Oficial [da República Federativa do Brasil], Brasília, Ano CXLIX, n.3, 4 jan. 2012. Seção 1, pt1.

BRASIL. *Manual de BRT*. 2008. Guia de Planejamento. República Federativa do Brasil. Disponível em: <http://multimidia.brasil.gov.br/biblioteca/manual_brt.pdf >. Acesso em: 21 outubro 2012.

CORMEN, T.H., LEISERSON, C.E., RIVEST, R.L., STEIN, C.,. *Algoritmos: Teoria e Prática*. 2002. Tradução da 2ª edição americana. Rio de Janeiro, Campus.

CHAO, Y., HONGXIA, W. *Developed Dijkstra Shortest Path Search Algorithm and Simulation*. 2010. International Conference On Computer Design And Applications (ICCDA 2010).

CHERKASSKY, B.V., GOLDBERG, A.V., and RADZIK, T., *Shortest paths algorithms: Theory and experimental evaluation***.** 1996. Mathematical Programming, 73(2): 129-174.

CHO, H.J., LAN, C.L. *Hybrid shortest path algorithm for vehicle navigation*. 2008. Springer Science Business Media. J Supercomput (2009) 49: 234-247.

CHOWDHURY, M. A., SADEK, A., *Fundamentals of Intelligent Transportation Systems Planning*. 2003. London: Artech House.

DELLING, D., PAJOR, T., WAGNER, D. *Engineering Time-Expanded Graphs for Faster Timetable Information*. 2008. 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2008).

DMAT. *Departamento de Matemática*. 2012. Disponível em: <http://www.uc.pt/fctuc/dmat>. Acesso em: 07 outubro 2012.

DRANE, C., RIZOS, C. *Positioning Systems in intelligent transportation systems*. 1998. London: Artech House.

FERRAZ, A. C. P., TORRES, I. G. E., *Transporte Público Urbano*. 2004. 2ª edição. São Carlos: Rima.

FTA. *Advanced Public Transportation Systems: The State of the Art – Update 2006*. 2006. Technical report, FTA, Washignton, DC. Disponível em: <http://www.fta.dot.gov/documents/APTS_State_of_the_Art.pdf>. Acesso em: 01 outubro 2012.

FU, L., SUN, D., RILETT, L. *Heuristic shortest path algorithms for transportation applications: State of the art*. 2006. Computers & Operations Research, 33(11):3324–3343.

GOLDEN, B. L., BALL, M.. *Shortest paths with euclidean distances: An explanatory model*. 1978. Networks, 8(4):297–314.

GOODRICH, M. T., TAMASSIA, R., *Estrutura de Dados e Algoritmos em Java*. 2007. 4ª edição. Porto Alegre: Bookman.

GORE, J. H. *Elements of plane and spherical trigonometry*. 1907. G.P. Putnamâs sons.

ITS. *ITS Brasil – Sistemas de Transportes Inteligentes*. 2012. Disponível em: <http://www.itsb.org.br/BRT.html>. Acesso em: 21 outubro 2012.

JAQUES, P., BASTOS, R., PASIN, M., CHIWIACOWSKY, L. D., SOMMER, T., KONRARD, T. 2012a. *Um Estudo de Caso com o Uso do A* para o Problema do Caminho Mais Curto em Transporte Público*. In: Brazilian Conference on Intelligent System - Encontro Nacional de Inteligência Artificial, 2012, Curitiba.

JAQUES, P., PASIN, M., CHIWIACOWSKY, L., BAZZAN, A.L.C., MORAES, R., BASTOS, R. 2012b. *Provendo Informações para Atores do Sistema de Transporte Público: Um Passo na Direção de Sistemas de Transportes Inteligentes*. XXVI ANPET – Congresso de Pesquisa e Ensino em Transportes, 2012, Joinville.

LIU, C. L., PAI, T. W., CHANG, C. T., HSIEH, C. M. *Path-Planning Algorithms for Public Transportation Systems*. 2001. IEEE Intelligent Transportation Systems Conference Proceedings – Oakland (CA) USA – August 25-29, 2001.

NETTO, P. O. B., *Grafos: Teoria, Modelos, Algoritmos*. 1996. 1ª edição. São Paulo: Edgar Blücher.

NILSON, N. J. *Problem-Solving Methods in Artificial Intelligence*. 1971. McGraw-Hill.

OLHOVIVO. *Sistema de Monitoramento do Transporte*. 2012. Disponível em: <http://olhovivo.sptrans.com.br/>. Acesso em: 21 outubro 2012.

OLIVEIRA, C.H. P, *SQL. Curso Prático*. 2002. 1ª edição. São Paulo: Novatec.

PAPAGEORGIOU, M., DIAKAKI, C., DINOPOULOU, V., KOTSIALOS, A., WANG, Y. 2003. *Review of road traffic control strategies*. Proceedings of the IEEE, 91(12): 2043–2067.

PIARC. *Intermodality: Measures to Stimulate Public Transport Usage*. 2000. Technical report. Disponível em: <http://www.piarc.org/en/order-library/4028-en-

Intermodality:%20Measures%20to%20Stimulate%20Public%20Transport%20Usage.htm>. Acesso em: 30 setembro 2012.

POATRANSPORTE. *O guia do transporte em Porto Alegre*. 2012. Disponível em: <http://www.poatransporte.com.br>. Acesso em: 01 setembro 2012.

PORTALTRANSPARENCIA. *O portal da transparência da prefeitura de Porto Alegre*. 2012. Disponível em: <http://www2.portoalegre.rs.gov.br/eptc/default.php?p_secao=152>. Acesso em: 21 outubro 2012.

POSTGRESQL. *The world´s most advanced open source database*. 2012. Disponível em: <http://www.postgresql.org/>. Acesso em: 26 outubro 2012.

RUSSEL, S., NORVIG. P. *Artificial Intelligence: A Modern Approach*. 1995. Prentice Hall.

SCHEIN, A. L. *Sistema de Informação ao Usuário como Estratégia de Fidelização e Atração*. 2003. 148 f. Dissertação (Mestrado em Engenharia de Produção) – Programa de Pós-Graduação em Engenharia de Produção. Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, 2003.

SEDGEWICK, R.,VITTER, J. S. *Shortest paths in euclidean graphs*. 1986. Algorithmica, 1(1-4):31–48.

SILVA, D. M. *Sistemas Inteligentes no Transporte Coletivo por Ônibus*. 2000. 144 f. Dissertação (Mestrado em Engenharia de Produção) – Programa de Pós-Graduação em Engenharia de Produção. Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, 2000.

SPTRANS. *São Paulo Transportes - Prefeitura de São Paulo*. 2012. Disponível em: <http://www.sptrans.com.br/a_sptrans/ >. Acesso em: 21 outubro 2012.

SUSSMAN, J., *Introduction to Transportation Systems*. 2000. Boston, London: Artech House.

TRENSURB. Empresa de Trens Urbanos de Porto Alegre. 2012. Disponível em: <http://www.trensurb.gov.br/paginas/paginas_detalhe.php?codigo_sitemap=11 >. Acesso em: 21 outubro 2012.

ZHANG, J., LIAO, F., ARENTZE, T., TIMMERMANS, H. *A multimodal transport network model for advanced traveler information systems*. 2011. Procedia Social and Behavioral Sciences 20 (2011) 313–322.

ZHANG, Z., JIGANG, W., DUAN, X. *Practical Algorithm for Shortest Path on Transportation Network*. 2010. International Conference On Computer And Information Application (ICCIA 2010).