UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Uma Arquitetura Orientada a Serviços para Roteamento Personalizado

Elvis Rodrigues da Silva

(Mestrando)

Cláudio de Souza Baptista, PhD

(Orientador)

Campina Grande – PB
Abril de 2007

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Uma Arquitetura Orientada a Serviços para Roteamento Personalizado

Elvis Rodrigues da Silva

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande, como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Sistemas de Informação e Banco de Dados

Orientador: Cláudio de Souza Baptista, PhD

Campina Grande – PB
Abril de 2007

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

S586a
2007 Silva, Elvis Rodrigues da
Uma arquitetura orientada a serviços para roteamento personalizado / Elvis Rodrigues da Silva. ─ Campina Grande, 2007.
103p.: il.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Referências
Orientadores: Profº. Cláudio de Souza Baptista.

1– Algoritmos 2– Geoprocessamento 3– Teoria dos Grafos 4– Banco de Dados

I. Título.

CDU –004.021

"UMA ARQUITETURA ORIENTADA A SERVIÇOS PARA ROTEAMENTO PERSONALIZADO"

ELVIS RODRIGUES DA SILVA

DISSERTAÇÃO APROVADA EM 18.05.2007

PROF. CLÁUDIO DE SOUZA BAPTISTA, Ph.D
Orientador

PROF. ULRICH SCHIEL , Dr.
Examinador

PROF. ANSELMO CARDOSO DE PAIVA, Dr.
Examinador

CAMPINA GRANDE – PB

Determinação, coragem e autoconfiança são fatores decisivos para o sucesso. Não importa quais sejam os obstáculos e as dificuldades. Se estamos possuídos de uma inabalável determinação, conseguiremos superá-los. Independentemente das circunstâncias, devemos ser sempre humildes, recatados e despidos de orgulho.

(Buda)

Agradecimentos

À minha mãe, pelo incentivo, apoio incondicional e por suportar minha ausência nos finais de semana, noites e outros horários extra-expediente.

Aos meus amigos próximos, pelo apoio e pelos inesquecíveis momentos de escape nos raros dias de folga.

Aos meus amigos distantes, mas que conversaram comigo sempre que possível pela Internet, ajudando a aliviar o estresse diário.

Ao professor Cláudio pelo gratificante trabalho realizado nesses mais de quatro anos de colaboração.

Aos professores Anselmo Paiva e Ulrich Schiel pela contribuição ao trabalho através de críticas, idéias e sugestões.

E aos colegas da UFCG pelas sugestões, pelo esclarecimento de dúvidas e pelos momentos de descontração.

Sumário

Abreviações

GHM	Greedy Heuristic Method
GPS	Global Positioning System
HEPV	Hierarchical Encoded Path Views
HiTi	Hierarchical MulTi
iGIS	internet Geographic Information System
JPEG	Joint Pictures Expert Group
OGC	Open Geospatial Consortium
OSR	Optimal Sequenced Route
PDAs	Personal Digital Assistant
PoI	Point of Interest (Ponto de Interesse)
SGBD	Sistema de Gerenciamento de Bancos de Dados
SIG	Sistema de Informação Geográfica
SOA	Arquitetura Orientada a Serviços
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SVG	Scalable Vector Graphics
UDDI	Universal Discovery Description and Integration
UML	Unified Modeling Language
WSDL	Web Services Description Language
XML	eXtended Markup Language

Lista de Figuras

Lista de Quadros

Resumo

Os sistemas de roteamento vêm se tornando ferramentas bastante úteis recentemente. Eles pretendem ajudar os usuários a encontrar o caminho mais adequado entre dois lugares de acordo com a distância da viagem, o tempo do percurso, e outros critérios.

Esta dissertação apresenta uma arquitetura orientada a serviços e propõe um novo algoritmo de busca de rotas chamado *Coolest Path*. Este algoritmo habilita personalização multi-critério de acordo com a distância da viagem, o tempo, pontos turísticos e a simplicidade do caminho.

Além disso, a dissertação propõe restrições a serem acrescentadas aos caminhos calculados: *A-autonomy*, onde o usuário define uma constante A e o algoritmo provê um caminho com N paradas, uma a cada distância A; e *T-autonomy*, onde o usuário define uma constante T e o algoritmo provê um caminho com N paradas, uma a cada T unidades de tempo. Essas paradas são realizadas em pontos de interesse (ex. pontos turísticos).

Os algoritmos são fornecidos na forma de serviços Web, ou seja, são acessíveis de qualquer dispositivo conectado à Internet: um *browser desktop*, um celular ou um quiosque turístico localizado em um aeroporto. O serviço de roteamento é uma extensão ao *framework* iGIS.

Palavras chave: Roteamento; Sistemas de Informação Geográfica; *Web Services*.

Abstract

Routing systems have become very powerful tools recently. They help users in finding the most suitable path between two places using travel distance, time and others criteria.

This dissertation presents a routing system based on service-oriented architecture, which includes the proposal of an innovative algorithm known as coolest path. This algorithm enables multi-criteria personalization by using travel distance, time, points of interest and path simplicity.

Moreover, constrained paths are also supported including the implementation of *a-autonomy* in road networks and the proposal of a new algorithm: *t-autonomy*, which returns the best path with N stops, such that the travel time between any two consecutive points in the path is not greater than *t*.

These algorithms are implemented as an iGIS extension by using Web services technology.

Keywords: Routing; Geographical Information Systems; Web Services.

Capítulo 1 . Introdução

Muitas cidades possuem um trânsito extremamente complexo, e acaba tornando-se difícil para as pessoas chegarem aos locais aonde desejam ir. E só com muita prática, as pessoas conseguem ir aos lugares pelos caminhos mais curtos ou mais rápidos. Poderiam ser consultados mapas impressos, mas, nesse caso, os próprios usuários teriam que encontrar a informação que desejassem, além dos mapas impressos serem estáticos, enquanto mapas digitais podem ser dinâmicos e viabilizar uma certa interação com o usuário. Por exemplo, em um mapa digital das ruas e pontos turísticos de uma cidade, além da forma geométrica de cada rua, pode haver as informações dos estabelecimentos, como restaurantes, e seus proprietários, dos nomes de cada rua, etc., informações estas disponíveis a um clique do mouse do usuário. Também é importante destacar que as informações dos estabelecimentos mudam com o tempo (por exemplo, o cardápio de um restaurante, ou os horários dos filmes do cinema) e um mapa digital pode manter sempre atualizados esses dados.

Contudo, não basta que os mapas sejam digitais, pois ainda assim os usuários precisarão adquirir esses mapas de alguma forma (os aparelhos de navegação são falhos quando, por exemplo, o usuário vai a uma cidade da qual ele não tem dados, ou ainda quando mudam o sentido de uma rua e aqueles se tornam desatualizados). O ideal é que os mapas estejam disponibilizados em alguma tecnologia utilizando a Web, através de um Sistema de Informação Geográfica (SIG) [BM98, LGM+01] para Web, como por exemplo o iGIS [BLS+04]. A vantagem dos mapas estarem disponíveis na Web é que os usuários podem acessá-los de forma ubíqua, ou seja, do seu celular, do seu computador de casa ou de um cybercafé.

A tecnologia da informação vem evoluindo exponencialmente há alguns anos, chegando num patamar que viabiliza a realização de muitas idéias inovadoras antes apenas observadas em filmes e livros de ficção científica. Um exemplo claro são os aparelhos de navegação [ABr07, GQR07], que são populares na Europa e nos EUA há alguns anos, mas apenas recentemente as empresas passaram a disponibilizá-los comercialmente no Brasil. Estes dispositivos de navegação se utilizam de algoritmos para encontrar rotas entre dois determinados lugares, facilitando a movimentação de

pessoas através de caminhos que elas não conhecem adequadamente. À capacidade de encontrar essas rotas chamamos Roteamento.

Conjuntamente com a visualização dos mapas na Web e interação com os mesmos por parte dos usuários, deve haver uma camada de software responsável por encontrar rotas, de acordo com as preferências dos usuários. Roteamento constitui-se na análise de um conjunto de possíveis caminhos entre dois locais, determinando para o usuário um caminho que melhor se adapte aos seus requisitos. Sites como o MapLink [MLi07] e o Map24 [M2407] provêem essa funcionalidade. Se um usuário quer saber que caminho tomar de um determinado endereço a outro, basta que consulte um sistema com essa facilidade e um caminho será retornado para ele.

A maioria dos sistemas de geração de rotas entre dois lugares, seja na Web ou nos dispositivos de navegação, apresenta pouca personalização. Personalização nesse contexto significa permitir ao usuário definir o maior número possível de características do que seria um caminho ideal e encontrar uma rota que se adeqüe a essas características. Uma solicitação de caminho entre dois determinados lugares poderia vir, por exemplo, acompanhada da seguinte personalização: dar preferência a rotas mais simples e que passem em pontos turísticos.

Além dos sistemas de Roteamento existentes não serem adequadamente personalizados aos usuários, eles estão restritos a uma arquitetura proprietária e aos SIGs que eles mesmos disponibilizam. Para suprir essa deficiência, é necessário o uso de uma Arquitetura Orientada a Serviços (SOA). Uma das grandes vantagens de utilizar-se esse paradigma é permitir uma maior interoperabilidade entre aplicações e uma padronização da estrutura de acesso e comunicação aos serviços oferecidos pelas mesmas. Um *Web Service*, ou serviço Web, é uma interface que descreve uma coleção de operações acessíveis via rede através de troca de mensagens padronizadas em XML [ACK+03]. Uma ferramenta baseada em serviços Web é independente do cliente utilizado. Dessa forma, se há um serviço de Roteamento, uma aplicação cliente pode acessá-lo de um *browser* na Web, outra aplicação de um celular, e ainda outra de um aparelho de navegação, tornando possível várias aplicações diferentes utilizarem os mesmos dados e algoritmos e, ainda além, a mesma implementação dos algoritmos.

1.1 Objetivos

Essa dissertação visa o desenvolvimento de uma arquitetura de software flexível e interoperável de Roteamento, que atenda aos requisitos personalizados de cada usuário e

encontre as rotas que mais se adequem às suas necessidades, em tempo hábil.

Dessa forma, o objetivo principal da dissertação é fornecer algoritmos para busca de rotas personalizadas e fornecer um serviço Web para obtenção dessas rotas, que possa ser acessado de forma ubíqua. Como objetivos específicos, é possível citar:

- Prover um conjunto de características personalizáveis, de forma que os usuários possam ter várias opções de rotas, de acordo com a devida importância de cada uma dessas características;
- Possibilitar que os usuários acrescentem restrições aos seus caminhos, como paradas a cada X unidades de distância ou a cada X unidades de tempo. Essas paradas poderão ser em pontos de interesse da escolha dos usuários, como por exemplo, farmácias e restaurantes;
- Pré-materializar as rotas de todos os caminhos, armazenados de forma hierárquica, visando melhorar a velocidade na obtenção dos resultados, tendo em vista que os algoritmos são pesados e demoram a executar;
- Estender o software iGIS [BLS+04] para dar suporte a roteamento baseado em arquitetura SOA e Sistema de Gerenciamento de Bancos de Dados (SGBD) livre.

1.2 Estrutura da Dissertação

Este documento organiza-se da seguinte forma:

- Capítulo 2: Apresenta uma fundamentação teórica necessária para compreensão da dissertação, dividida em três partes: uma visão sobre Sistemas de Informação Geográfica, apresentando uma comparação entre sistemas *Desktop* e Web e explicando, em linhas gerais, o funcionamento do *framework* iGIS; apresenta o conceito de Grafos, algumas definições relacionadas e um conjunto de algoritmos para busca de caminhos; e introduz o conceito de Roteamento, juntamente com a análise de várias ferramentas para busca de rotas comercialmente disponíveis, além de fazer uma avaliação de alguns trabalhos relacionados importantes na área;
- Capítulo 3: Apresenta os algoritmos propostos para busca de rotas personalizadas e de rotas com restrições. Divide-se em duas partes: na primeira, reúne quatro algoritmos de busca de rotas utilizando heurísticas e apresenta o *Coolest Path*, um algoritmo original que visa encontrar a rota mais adequada às

preferências do usuário; na segunda, apresenta o conceito de caminhos com restrições, estendendo-o à busca de caminhos em redes de estradas, com algoritmos para duas possíveis restrições;

- Capítulo 4: Motiva o uso de *Web Services*, apresenta a arquitetura do protótipo desenvolvido para testar os algoritmos propostos e explana sobre os conceitos de hierarquização de grafos e materialização de rotas;
- Capítulo 5: Analisa o comportamento dos algoritmos e do protótipo através de um estudo de caso, com exemplos diversificados de rotas, sem e com restrições;
- Capítulo 6: Conclui o documento com uma avaliação objetiva dos resultados obtidos e um conjunto de trabalhos que podem vir a ser desenvolvidos no futuro.

Capítulo 2 . Fundamentação Teórica

Este capítulo tem como objetivo principal apresentar um conjunto de conhecimentos necessários para a compreensão da dissertação. Inicialmente, é apresentada uma noção de Sistemas de Informação Geográfica. Em seguida, são detalhados alguns conceitos básicos e outros mais avançados da Teoria dos Grafos. Por fim, o capítulo apresenta conhecimentos acerca de Roteamento, baseado nos trabalhos relacionados e em ferramentas de busca de rotas.

2.1 Sistemas de Informação Geográfica

Essa seção apresenta uma breve introdução aos Sistemas de Informação Geográfica, passando por uma visão abrangente da área, detalhando algumas características dos SIG, como formatos de dados e arquiteturas de implementação. A seção aponta também algumas vantagens de *Web mapping* sobre *Desktop mapping*, e conclui apresentando o *framework* iGIS.

2.1.1 Visão Geral

Um sistema de roteamento de veículos necessita fazer uso de uma ferramenta de exibição e manipulação de mapas, ou seja, um Sistema de Informação Geográfica [HCC98, LGM+01, WD04]. Um software de geoprocessamento faz-se necessário para que o usuário veja os mapas da região onde foi solicitada a rota, e a manipulação é importante para destacar o caminho obtido no mapa. Também permite que o usuário realize operações espaciais, como cálculo de distâncias e áreas, operações essas que, embora não sejam fundamentais a um sistema de roteamento, configuram-se como uma facilidade a mais.

Há um conjunto de possíveis definições para Sistemas de Informação Geográfica, cada uma apresentando suas particularidades. Algumas dessas definições seguem:

- De acordo com Burrough [BM98], é um "conjunto poderoso de ferramentas para coletar, armazenar, recuperar, transformar e visualizar dados sobre o mundo

real";

- De acordo com Longley et al. [LGM+01], são "uma classe especial de sistemas de informação que não se preocupam apenas com eventos, atividades e coisas, mas também com onde esses eventos, atividades e coisas acontecem ou existem";
- De acordo com Worboys & Duckham [WD04], são "sistemas de informação computadorizados utilizados para adquirir, modelar, armazenar, recuperar, compartilhar, manipular, analisar e apresentar dados referenciados geograficamente".

Longley et al. [LGM+01] listam três motivos pelos quais os SIGs devem ser utilizados:

- Quase tudo que acontece, acontece em algum lugar. Saber onde alguma coisa acontece é importante;
- Podemos identificar se os problemas são geográficos com facilidade;
- Com uma coleção de ferramentas, SIGs são capazes de preencher o espaço entre curiosidade científica e resolução prática de problemas.

Algumas características que os SIGs possuem são a capacidade de integrar informação espacial e textual e o fato de facilitar a tomada de decisão, gerando cenários espaciais que agrupam muita informação sumarizada.

Como exemplos de aplicações que se beneficiam de um SIG, é possível citar:

- Sistema de recursos hídricos de um estado, contendo os dados dos rios, açudes, estações pluviométricas, adutoras, além dos dados básicos dos municípios do estado. Um sistema assim seria útil para monitoramento do volume dos mananciais apenas clicando na forma geométrica do mesmo, por exemplo. Auxiliaria no apoio a decisão, pois seria possível ver rapidamente quais municípios estão necessitando da construção de um açude, ou ainda, quais municípios vão se beneficiar de uma determinada adutora;
- Sistema da tubulação de petróleo dos poços a uma refinaria. Este sistema seria muito útil no que diz respeito à análise de problemas. Por exemplo, se houver um vazamento em um oleoduto, que áreas serão afetadas pelo óleo? Há risco de desastre ambiental? Que transmissões de petróleo precisam ser interrompidas para impedir o escoamento enquanto o conserto é realizado? Com o uso de um SIG, todos esses dados poderão ser inferidos visualmente;

- Sistema de busca de rotas, que é o foco desta dissertação. Ou seja, um sistema que facilite a busca de caminhos entre dois lugares, utilizando-se de um SIG para realizar operações de distância e etc., de um banco de dados espacial para armazenar as geometrias das ruas e pontos de interesse e de um cliente SIG para exibir os dados e permitir a interação com o cliente.

Há dois métodos para representação de dados espaciais digitalmente:

- *Raster*: armazena dados gráficos usando uma matriz ou uma grade de células. A maioria dos dados vem na forma de imagem de satélite, mapas escaneados e dados de elevação. Cada célula pode ser considerada um *pixel* da imagem;
- *Vector*: armazena dados através de coordenadas x,y que podem ser ligadas para formar linhas e polígonos. Este método é melhor para dados discretos, ou seja, limites geopolíticos, cidades, ruas e etc. Parte do princípio que todas as feições do mundo podem ser representadas por pontos, linhas e polígonos.

Diante do conhecimento acerca do que são e para que servem os Sistemas de Informação Geográfica, é necessário decidir qual dessas ferramentas utilizar para o sistema de rotas. Há varias arquiteturas possíveis para implementação de um SIG [WD04], dentre essas temos:

- Relacional: todos os dados são armazenados em um SGBD relacional, e os dados geográficos são armazenados como atributos simples de tuplas de uma relação. Não é uma estratégia adequada, pois não permite a realização de consultas espaciais, além de ser complicado e dependente de muitas junções realizar determinadas consultas, quebrando a independência dos dados e reduzindo o desempenho;
- Dual: um SGBD relacional armazena os dados não-espaciais e um arquivo armazena os dados espaciais. Grande parte das ferramentas do mercado utiliza essa arquitetura, como por exemplo o ESRI ArcView [ESR07]. Um problema grave que podemos identificar é a dificuldade em manter a integridade entre as duas partes, além do que requer que o usuário saiba utilizar as partes distintas. Escolhendo essa alternativa, os dados de conectividade das ruas e informações acerca dos nomes seriam armazenados em um SGBD relacional, e as geometrias das ruas estariam em um arquivo binário;
- Integrada em SGBD relacional: todos os dados são armazenados em um SGBD relacional, mas os dados espaciais são guardados em campos binários, que não

capturam a semântica e trazem limitações de manipulação. A implementação de um sistema de roteamento seria semelhante à arquitetura dual, com a diferença de que as geometrias estarão armazenadas no SGBD, mas ainda de forma binária;

- Integrada com extensão espacial: utiliza um SGBD objeto-relacional para armazenar todos os dados. Os objetos espaciais são reconhecidos por uma extensão espacial do SGBD e podem ser realizadas manipulações nos mesmos, como cálculo de área, para exemplificar. Outra importante vantagem é a possibilidade de utilizar todos os recursos do banco de dados, como atomicidade, integridade, etc. Ou seja, todos os dados de um sistema de roteamento seriam armazenados no SGBD com extensão espacial, de forma que possam ser realizadas operações nas formas geométricas diretamente no banco de dados.

2.1.2 Web Mapping

Originalmente, os SIGs eram ferramentas *stand-alone*, ou seja, utilizadas em *desktops*, sem conectividade. Mas, com a popularização da Internet e o crescimento da velocidade de conexão das redes, começaram a ser desenvolvidos SIGs para Web [Pen97]. A primeira alternativa é conhecida como *Desktop Mapping*, enquanto a segunda como *Web Mapping*.

A princípio, *Desktop Mapping* ainda é bastante utilizado para confecção de mapas e manipulações mais avançadas. Contudo, *Web Mapping* é uma alternativa interessante para visualização de mapas e realização de consultas simples. Uma das vantagens é a acessibilidade, pois não é necessária a instalação de nenhuma ferramenta de software, apenas, em alguns casos, um *plug-in*. Mas a principal vantagem é a disponibilidade. Um SIG na Web pode ser acessado de qualquer dispositivo com a conectividade necessária, desde um micro-computador até um celular, passando por quiosques de turismo.

Para a atividade de Roteamento, é interessante que seja utilizado um SIG para Web, de forma que a funcionalidade de busca de rotas e os mapas com as rotas resultantes possam ser acessados de qualquer dispositivo com conectividade à Internet. Então, caso haja um cliente para plataformas móveis, será possível acessar o serviço de rotas de um PDA ou de um celular. Ou ainda, acessar de um quiosque turístico.

2.1.3 iGIS

O projeto iGIS é um *framework* para rápido desenvolvimento de aplicações geográficas na Web, baseado em tecnologias abertas, como a linguagem Java, os padrões do *Open Geospatial Consortium* (OGC) [OGC07] e a tecnologia XML de exibição de gráficos na Web *Scalable Vector Graphics* (SVG) [W3C07]. Uma das principais vantagens de ser um *framework* vem com a facilidade de acrescentar novas funcionalidades e acoplar componentes de software à medida que for preciso, reduzindo o tempo necessário para as aplicações entrarem em produção.

O iGIS [MBA+02, PSL+04, BLS+04, BNS+05, CPT+06] vem sendo desenvolvido pelo Laboratório de Sistemas de Informação (LSI) da Universidade Federal de Campina Grande (UFCG), em colaboração com o Grupo de Informática Aplicada (GIA) da Universidade Federal do Maranhão (UFMA) desde 2002, com financiamento inicial do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

A arquitetura do iGIS é composta de três camadas principais, como é possível observar na figura 2.1. A camada de dados é composta pelos SGBD's (Oracle [Ora07], PostgreSQL [Pos07a] e IBM DB2 [IBM07]), além dos arquivos espaciais ESRI *Shapefile* [ESR97]. Esta camada é responsável por armazenar os dados, espaciais e não-espaciais. O *framework* é facilmente extensível, então se um novo SGBD precisar ser acrescentado, basta estender algumas classes.

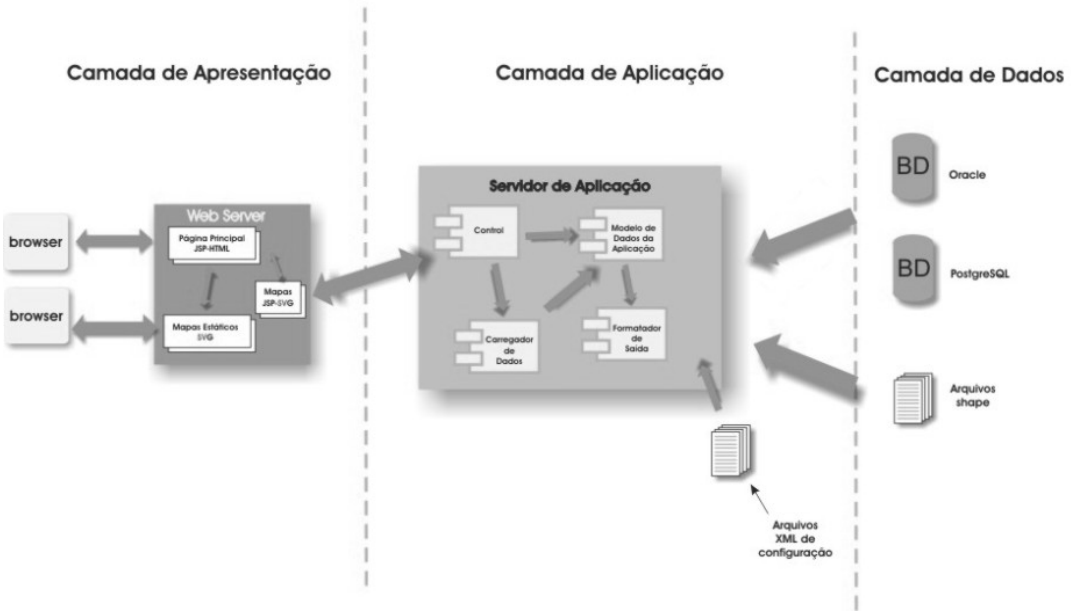

Figura 2.1 Arquitetura do iGIS

Na camada de aplicação está a lógica do sistema. Os mapas e fontes de dados são configurados através de arquivos XML, então um módulo do iGIS, o *datasources*, é responsável pela interface de acesso aos bancos de dados; outro módulo, chamado *model*, implementa a interface de geometrias definida pelo OGC; o módulo *formatter* converte os dados vindos do banco para o formato de apresentação ao usuário.

Na camada de apresentação estão os arquivos JSP com a interface do iGIS e o Javascript de controle das interações com o usuário. Por enquanto o iGIS utiliza dois formatos de saída: SVG e JPEG [JPG07], mas é importante ressaltar que, pelo fato de ser um *framework*, o iGIS possui muita flexibilidade, tornando possível o uso de outro formato acrescentando apenas algumas classes.

O iGIS foi escolhido como SIG cliente para o serviço de roteamento, pois, além das vantagens da ferramenta já acima citadas, há a disponibilidade do código-fonte aberto, de forma a poder acrescentar os trechos necessários para invocação do serviço e exibição dos resultados no mapa. Além disso, há uma versão móvel do iGIS, ou seja, o serviço poderá ser utilizado no celular, caso seja desejável.

2.2 Teoria dos Grafos

Essa seção tem a finalidade de apresentar alguns conceitos gerais da Teoria dos Grafos, dando ênfase aos mais conhecidos algoritmos para busca de caminhos: busca em profundidade e em largura, Dijkstra e A*. Os algoritmos são sucintamente apresentados e explicados em linhas gerais.

2.2.1 Visão Geral

Para representar redes de estradas num banco de dados poderia ser utilizado um conjunto de estruturas de dados distintos. Contudo, o conceito de grafo é o que mais se aproxima de uma rede de ruas conectadas por interseções [WD04, SC03].

Um grafo é um par $G = (V, E)$ de conjuntos [Die05], tal que os elementos do conjunto V são os vértices (ou nós) e os elementos do conjunto E são as arestas do grafo G. Uma aresta $e \in E$ deve ligar dois vértices $x, y \in V$, e também pode ser chamada xy. Neste caso, dizemos que os vértices x e y são adjacentes. Duas arestas são adjacentes se elas tiverem um vértice em comum. Um grafo $G' = (V', E')$ é dito como sendo um sub-grafo de G se $V' \subseteq V$ e $E' \subseteq E$.

Um grafo direcionado pode ser utilizado para representar as redes de ruas, mas de forma incompleta, ou seja, sem levar em consideração o sentido de cada rua. Um

grafo direcionado (ou dígrafo) é um par $G = (V, E)$ de tal forma que todas as arestas $e \in E$ tenham um vértice inicial e um final. Ou seja, suponha uma aresta xy de um grafo direcionado. O vértice inicial é x e o final y. Caso seja possível navegar de y para x, deve haver outra aresta direcionada yx.

O conceito de grafo direcionado se aproxima das ruas de uma cidade, entretanto, não representa as diferenças entre as ruas, ou seja, o porquê de escolher uma rua ao invés de outra. Essa diferença pode ser representada associando um determinado peso a cada rua, por exemplo a distância entre os vértices inicial e final no trecho em questão. Um grafo valorado [GT01] $G = (V, E)$ é um grafo que possui uma função $w(e)$ associada a cada aresta $e \in E$, chamada de peso da aresta e. Essa função pode representar o tamanho da aresta ou qualquer custo da transição, por exemplo, no sistema de rotas, pode estar relacionado à qualidade da pavimentação da rua. Algumas vezes, grafos direcionados são chamados de redes.

Um caminho em um grafo direcionado e valorado $G = (V, E)$, com a função w representando os pesos, é um sub-grafo não-vazio $P = (V', E')$, tal que $V' = \{x_0, x_1, ..., x_k\}$ e $E' = \{e_1, e_2, ..., e_k\}$, onde os x_i são distintos e $e_1 = x_0x_1$, $e_2 = x_1x_2$, ..., e $ek = x_{k-1}x_k$. O vértice x_0 é chamado de origem do caminho, enquanto x_k o destino. O custo total do caminho P é a soma dos pesos das arestas de P, ou seja, o tamanho, denotado $w(P)$, é definido como:

$$w(P) = \sum_{k=1}^{l} w(e_k)$$

Dessa forma, um caminho entre dois lugares em uma cidade pode ser representado pelo conceito de caminho em um grafo que represente a cidade.

2.2.2 Busca do Melhor Caminho

Para realizar a busca de um caminho em um grafo, é necessário utilizar uma estratégia de busca em espaço de estados [RN02]. Há um conjunto imenso de possíveis estratégias de busca, em vários níveis. Nas próximas seções haverá uma explanação sobre alguns algoritmos. Inicialmente serão apresentadas algumas estratégias mais simples, incluindo o algoritmo de Dijkstra, e em seguida apresentaremos A*, a estratégia escolhida para as buscas de caminhos na nossa abordagem.

Algoritmos simples

Dizemos que um algoritmo é cego quando ele não tem nenhuma informação sobre os nós que vai encontrar pela frente, sobre o número de passos ou uma indicação do caminho. Aqueles algoritmos que possuem alguma dessas informações são chamados algoritmos heurísticos e na próxima seção o mais conhecido destes algoritmos será explicado em detalhes.

Dois dos mais populares algoritmos cegos são o de Busca em Largura e o de Busca em Profundidade. O algoritmo de Busca em Largura [RN02] explora o nó inicial, em seguida todos os seus adjacentes, e os adjacentes deles. Suponhamos o grafo da figura 2.2, com cidades alemãs e a distância entre elas.

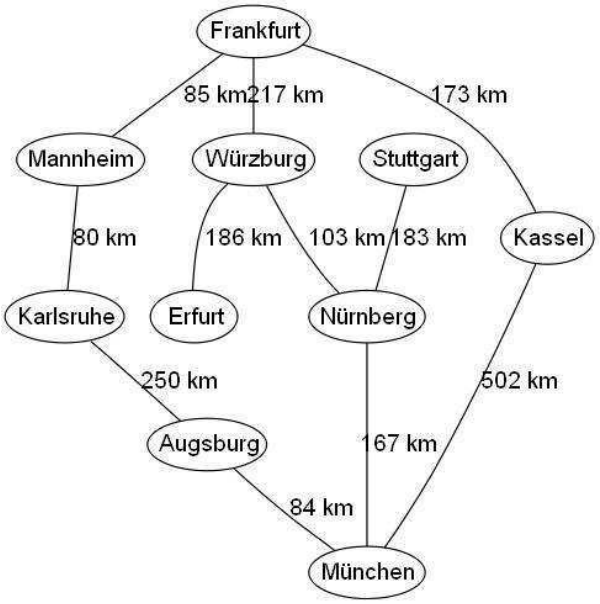

Figura 2.2 Grafo de rodovias alemãs (fonte: [Wik07a])

Organizando o grafo como uma árvore, torna-se mais fácil de compreender, conforme pode ser observado na figura 2.3. A numeração dos nós indica a ordem de caminhamento, tendo Frankfurt como origem.

No caso da busca de um caminho, o algoritmo vai encerrar quando encontrar o nó de destino desejado. A Busca em Largura é um algoritmo completo, ou seja, vai sempre encontrar uma solução, caso ela exista. Contudo, o algoritmo não é ótimo, pois, caso o grafo seja valorado, pode não encontrar a melhor solução, uma vez que vai encontrar aquela com menos arestas da origem ao destino, o que nem sempre corresponde aos menores pesos.

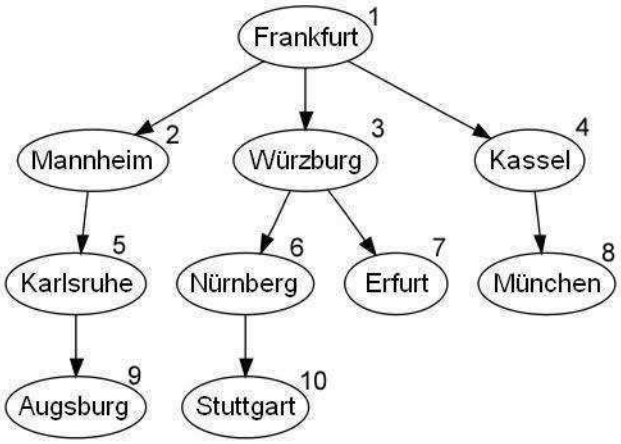

Figura 2.3 Árvore de caminhamento de uma Busca em Largura (fonte: [Wik07a])

O algoritmo de Busca em Profundidade [RN02] expande os nós até não encontrar mais nós adjacentes. Em seguida, retorna ao nó pai e caminha pelos demais adjacentes deste, até que não haja mais possibilidades. O processo é repetido, ou seja, o procedimento retorna ao nó pai e caminha pelos demais adjacentes, até que não haja mais nós a visitar ou quando o destino for encontrado. Assim como na Busca em Largura, organizar os nós em árvore facilita o entendimento, como pode ser analisado na figura 2.4. Novamente, a ordem de caminhamento do grafo da figura 2.2 tendo Frankfurt como origem segue a numeração.

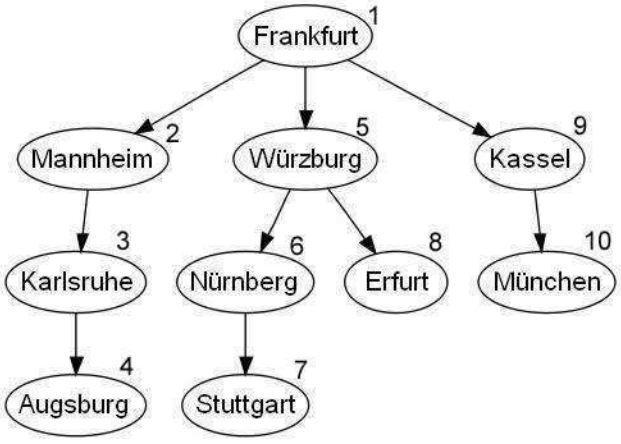

Figura 2.4 Árvore de caminhamento de uma Busca em Profundidade (fonte: [Wik07a])

Da mesma maneira que a Busca em Largura, a Busca em Profundidade é completa, mas não ótima, pois encontra o resultado, caso exista, mas nem sempre o melhor resultado, pois pára assim que encontra o destino, sem analisar as demais formas de se alcançar este nó.

Talvez o mais conhecido algoritmo de busca de melhor caminho em grafos seja o Algoritmo de Dijkstra [Dij59], um algoritmo de estratégia gulosa [CLR+01] que resolve o problema de busca de melhor caminho para grafos direcionados e valorados. O algoritmo funciona apenas para arestas com pesos positivos, característica que normalmente é verdadeira no problema de roteamento.

O algoritmo de Dijkstra utiliza duas funções auxiliares. A função IniciaOrigemUnica, que pode ser observada no algoritmo 2.1, realiza alguns procedimentos de inicialização: para cada nó do Grafo *G* (linha 2) define o antecessor (π) desse nó como *null*, ou seja, desconhecido (linha 3) e a distância da origem (*d*) igual a infinito (linha 4), exceto para o vértice de origem, cuja distância é zero (linha 5).

```
1  IniciaOrigemUnica (G (V, E), s) {
2    para cada v ∈ V faça
3      π[v] = null;
4      d[v] = ∞;
5    d[s] = 0;
6  }
```

Algoritmo 2.1 Inicialização dos dados

A outra função auxiliar utilizada pelo algoritmo de Dijkstra é a função Relaxa, que consiste de um teste para verificar se há a possibilidade de alcançar um determinado vértice de forma mais barata que a atual, ou seja, com um custo inferior. O algoritmo 2.2 exibe o algoritmo Relaxa, que recebe dois vértices (*u* e *v*) e a função que guarda os pesos das arestas (*w*). Na linha 2 é realizada a verificação se a distância da origem até *v* previamente calculada é maior que a distância da origem até *u* somada com a distância entre *u* e *v* e, caso seja, define a nova distância para *v* (linha 3) e o novo predecessor de *v*, que será *u* (linha 4).

```
1  Relaxa (u, v, w) {
2    se d[v] > d[u] + w(u, v) então
3      d[v] = d[u] + w(u, v);
4      π[v] = u;
5  }
```

Algoritmo 2.2 Relaxamento de um vértice

O algoritmo de Dijkstra pode ser observado no algoritmo 2.3. Um conjunto de vértices *S* é mantido, para armazenar os vértices cujos custos finais já foram encontrados. *Q* mantém o conjunto dos vértices do grafo *G* (linha 4). O algoritmo então percorre cada um dos vértices de *Q* (linha 5), partindo daquele que tem o menor custo de caminho da origem (linha 6), e por este motivo é um algoritmo guloso. Este vértice *u* é acrescentado a *S* (linha 7) e para cada um dos seus adjacentes *v* (linha 8), é chamado o procedimento Relaxa, visando determinar se o caminho da origem para *v* via *u* é mais rápido que o atual.

```
1  Dijkstra (G (V, E), w, s) {
2    IniciaOrigemUnica (G, s);
3    S = ∅;
4    Q = V;
5    enquanto Q <> ∅ faça
6      u = ExtraiMenor(Q);
7      S = S ∪ {u};
8      para cada v ∈ Adj[u] faça
9        Relaxa (u, v, w);
10 }
```

Algoritmo 2.3 Dijkstra

Após a execução do algoritmo, basta partir do vértice de destino desejado e caminhando pela função predecessor até encontrar a origem. Esta versão do algoritmo de Dijkstra encontra os menores caminhos de uma determinada origem para todos os demais vértices do grafo. O algoritmo tem custo de execução $O(n^2)$, onde *n* é o número de vértices em *V*. O algoritmo de Dijkstra é completo e é ótimo, pois sempre encontra a melhor solução. Contudo, necessita percorrer o grafo por completo para encontrar a solução e se torna inviável quando o grafo é muito grande.

A figura 2.5 apresenta um exemplo de caminhamento em grafo utilizando o algoritmo de Dijkstra.

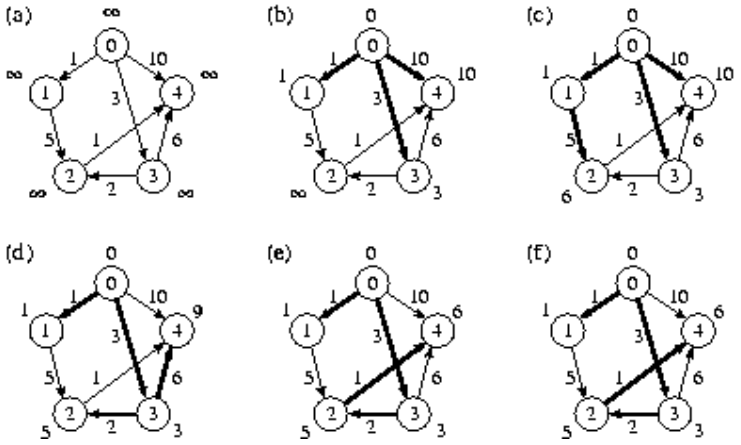

Figura 2.5 Passo a passo da execução do algoritmo de Dijkstra (fonte: [Ziv03])

A*

Algoritmos que possuem algum conhecimento prévio acerca dos nós ainda não visitados são chamados de algoritmos informados, ou de busca heurística. Algoritmos de busca heurística são mais eficientes que os cegos, principalmente devido ao seu tempo de execução, uma vez que podem evitar nós que previamente sabem que não vão lhe interessar para este caso específico. O algoritmo com heurística para busca em grafos mais utilizado é o A* (A - Estrela) [RN02, WD04].

A base do algoritmo A* é que para cada vértice tenha-se uma função $f(x) = g(x) + h(x)$, onde $g(x)$ é o custo real necessário para chegar da origem a este determinado vértice e $h(x)$ é uma função heurística que estima o custo do vértice x ao destino do caminho, de forma que $f(x)$ pode ser considerado o custo total do caminho passando por x. A estratégia do algoritmo é, a cada passo, escolher o nó com menor valor de $f(x)$, ou seja, que tenha chance de ter o menor custo real do caminho.

De acordo com Russel e Norvig [RN02], A* é um algoritmo completo e ótimo, desde que a função heurística $h(x)$ seja admissível, ou seja, nunca superestime o custo real de x ao destino. O tópico seguinte desta seção apresenta duas funções heurísticas e uma comparação entre elas. A complexidade de tempo do algoritmo A* depende de uma boa heurística, e é polinomial quando a seguinte condição é satisfeita:

$$\left|h(x) - h^*(x)\right| \leq O(\log h^*(x))$$

, onde $h(x)$ representa a heurística de x e $h^*(x)$ representa o valor real do custo de x.

O algoritmo A* pode ser observado no algoritmo 2.4. Inicialmente, o custo g da origem é definido como zero (linhas 2 e 4), o custo h como a estimativa heurística

(linhas 3 e 5) e o custo *f* como a soma de ambos (linha 6). Quando a heurística do vértice ao destino for igual a zero, o algoritmo pára (linhas 16 e 17), e enquanto isso não acontecer, varre os nós, selecionando sempre o de menor *f* (linha 10). Para cada um dos nós *v* sendo varridos, são calculados os custos dos seus nós adjacentes (linhas 13 a 15). O custo *g* do nó adjacente é o custo *g(x)* do vértice atual *v* somado com o custo da transição de *v* ao adjacente. A heurística *h* é estimada e *f* é a soma de *g* e *h*.

```
1  A*(origem, destino) {
2     g = 0
3     h = EstimativaHeuristica(origem, destino )
4     origem.g = g
5     origem.h = h
6     origem.f = g + h
7     Adicione origem à Lista
8     Concluido = false
9     enquanto(não Concluido) faça
10       EstadoAtual = MenorF(Lista)
11       Remove EstadoAtual da Lista
12       para cada adjacente S de EstadoAtual faça
13         S.g=EstadoAtual.g+Custo(EstadoAtual,S)
14         S.h = EstimativaHeuristica(S, destino)
15         S.f = g + h
16         se ( S.h == 0 ) então
17           Concluido = true
18         senão
19           Adicione S à Lista
20 }
```

Algoritmo 2.4 A*

A figura 2.6 apresenta um exemplo de execução do algoritmo A* em um grafo simples, direcionado e valorado, onde a origem do caminho é o vértice *A* (em azul) e o destino o vértice *J* (em laranja). Cada um dos passos da execução do algoritmo está retratado nas partes a) a h) da figura. Os vértices visitados são marcados em vermelho, e as arestas que levam a seus adjacentes também. No momento em que os adjacentes são recuperados, são calculadas suas heurísticas, apresentadas ao lado dos nós, entre parênteses. O valor nas arestas é o peso real de transição entre dois vértices.

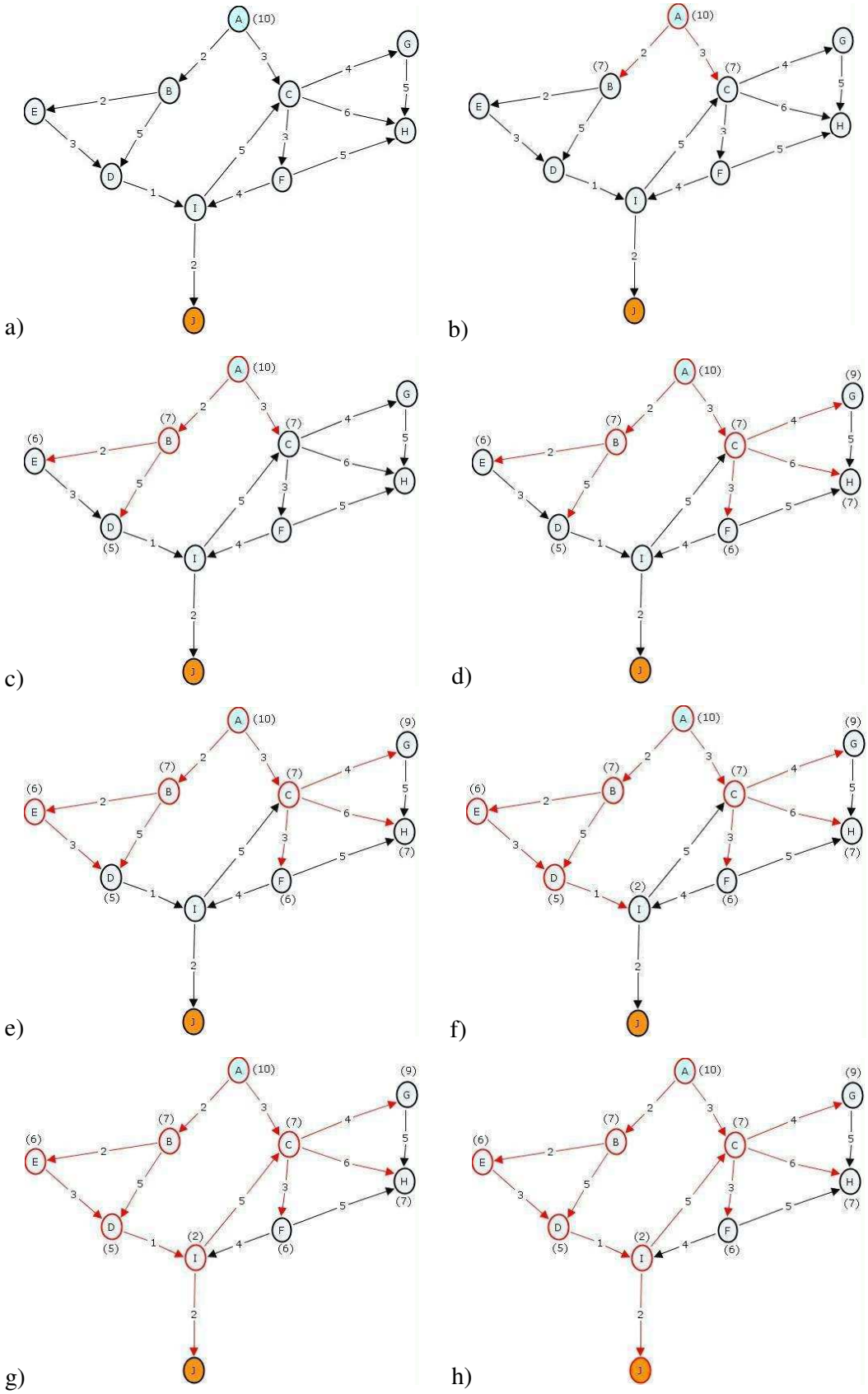

Figura 2.6 Execução do algoritmo A* em um grafo

Heurísticas

Há um conjunto de funções que podem ser utilizadas como heurísticas [Sal06], então qual deveria ser utilizada no algoritmo A* utilizado para roteamento? A primeira que pode vir à mente é a distância Euclidiana, que é definida da seguinte forma:

$$d(p1, p2) = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

, onde p1 = (x1, y1) e p2 = (x2, y2)

A distância Euclidiana é uma heurística admissível, pois a distância em linha reta entre dois pontos não pode ser maior que a distância via rede entre esses mesmos pontos. Contudo, há alguns casos em que ela não se aplica adequadamente, como pode ser demonstrado na figura 2.7. A distância Euclidiana entre os nós *A* e *B* é pequena, enquanto a distância via rede é bem superior, ou seja, a heurística ficou muito abaixo do custo real. Quanto maior o valor da função heurística, desde que não supere o real, mais eficiente será o algoritmo A*.

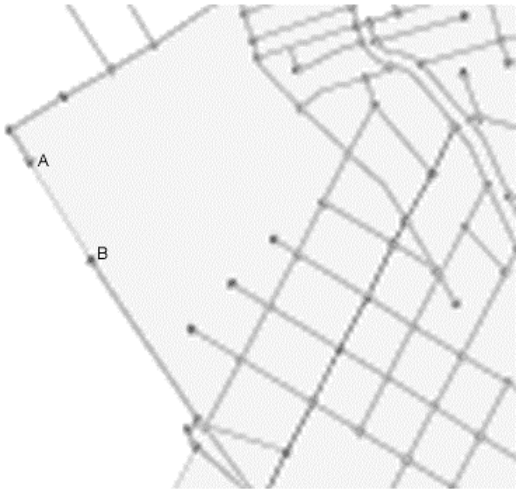

Figura 2.7 A distância Euclidiana entre *A* e *B* está distante do custo real de rede

Uma segunda possibilidade de heurística é a distância Manhattan, que tem esse nome devido à padronização das ruas de Manhattan, USA, onde as distância entre as esquinas são todas iguais. A distância Manhattan também é conhecida como distância de blocos, pois funciona como um caminhamento entre blocos de tamanho uniforme, como pode ser observado na figura 2.8.

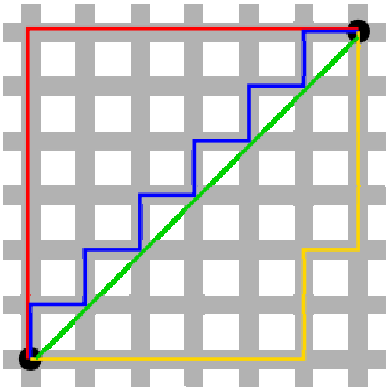

Figura 2.8 Distância Manhattan (linhas vermelha, azul e amarela) com tamanho 12 em comparação com a distância Euclidiana (linha verde) com tamanho de aproximadamente 8,48 (fonte: [Wik07b])

A distância Manhattan é uma distância de Minkowski de norma 1 [Sal06]. A distância de Minkowski de norma p pode ser definida pela equação abaixo:

$$dp(x_i, y_i) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

Para o caso em que p = 1, temos a distância Manhattan, conforme a seguinte equação:

$$d(p1, p2) = |x1 - x2| + |y1 - y2|$$

, onde p1 = (x1, y1) e p2 = (x2, y2).

A distância Manhattan foi escolhida como heurística para os algoritmos descritos nas seções subseqüentes, por representar melhor os blocos das cidades e o caminhamento através das ruas. Shekhar et al. [SKC93] apresentam uma análise de vários algoritmos para menor caminho em redes de estradas e realizam uma comparação entre usar a distância Euclidiana e a Manhattan como heurística do algoritmo A*. A distância Manhattan é indicada, apesar de não ser uma heurística admissível, por ser mais rápida, mesmo retornando uma solução apenas sub-ótima em algumas possibilidades. A economia no tempo de cálculo das rotas com distância Manhattan ao invés de Euclidiana faz valer a pena obter soluções sub-ótimas, mas não tão demoradas.

2.3 Roteamento

Esta seção tem a finalidade de introduzir o tema Roteamento, motivando a necessidade de sua utilização. É apresentado um conjunto de ferramentas de busca de rotas disponíveis no mercado, realizando-se uma comparação entre elas. Por fim, alguns

trabalhos relacionados ao tema são relacionados, citando vantagens e desvantagens de cada abordagem, e realizando uma comparação entre eles através de alguns aspectos levantados.

2.3.1 Visão Geral

Roteamento pode ser definido como um estudo de possíveis rotas entre duas localizações, de forma a selecionar a mais adequada para a situação. O termo é mais conhecido pelos mecanismos visando alcançabilidade em redes de computadores distintas, mas há alguns anos vem sendo também utilizado na área de Sistemas de Informação Geográfica, relacionado à busca de rotas entre duas localizações através de redes de estradas, ou *Spatial Networks* [SC03].

Roteamento pode ser utilizado em geoprocessamento em muitas ocasiões:

- Quando o usuário não sabe o caminho entre dois lugares, ele pode solicitar uma rota entre estes;
- Quando, apesar de conhecer possíveis caminhos, um determinado usuário gostaria de obter o caminho mais agradável de acordo com um conjunto de requisitos, por exemplo, o mais turístico;
- Como ferramenta importante em quiosques turísticos, facilitando para os visitantes da cidade encontrarem caminhos para chegarem a pontos turísticos, monumentos históricos, restaurantes, etc.;
- Para fugir de pontos de tráfego, engarrafamentos e etc.

Na próxima seção será feita uma revisão de ferramentas de roteamento disponíveis no mercado, e na seção seguinte será apresentada uma revisão bibliográfica de trabalhos relacionados a este tema.

2.3.2 Softwares no Mercado

Serviços de rotas na Web

O Maplink [MLi07] possui dois serviços de rotas, um dentro de uma cidade e outro entre cidades. O serviço que busca dentro da cidade permite que sejam buscadas rotas em vias principais, mais curtas ou a pé; também permite selecionar até três tipos de estabelecimentos para serem exibidos ao longo da rota; ainda apresenta descrições em forma de mapa e textual passo-a-passo para rota; exibe distância total da rota e provável preço da corrida de táxi pelo percurso; apresenta mapas ampliados para cada trecho;

gera uma versão para impressão; e permite acrescentar pontos intermediários de parada na rota, onde o usuário escolhe se deseja em ordem seqüencial ou otimizada. Já o serviço de rotas entre cidades tem as seguintes características: permite acrescentar cidades intermediárias; calcula o consumo de combustível desde que o usuário forneça o consumo médio do veículo e o preço do combustível; permite rotas mais rápidas ou mais curtas; a ordem das paradas pode ser seqüencial ou otimizada; permite selecionar o tipo de veículo (moto, carro pequeno, caminhonete, etc.); apresenta um mapa da rota e o detalhamento textual passo-a-passo, incluindo os custos dos pedágios. O Maplink apresenta como principais desvantagens a pouca personalização; a arquitetura proprietária; algumas limitações na quantidade de pontos de interesse que podem ser visualizados na tela; e o fato da rota entre cidades não contemplar a origem e o destino como endereços das cidades, mas apenas exibir o trajeto da saída da primeira cidade para a entrada da segunda. Como principal vantagem, destaca-se a possibilidade de se gerar um caminho seqüencial ou otimizado, quando pontos intermediários são escolhidos.

O Apontador [Apo07] também apresenta rotas dentro de uma cidade ou entre cidades, contemplando na rota entre cidades a origem e o destino como endereços específicos, e não apenas como pontos ligados por rodovias. Dentro das cidades, possui as seguintes características: escolha entre vias principais, desviando do trânsito, caminho mais curto ou a pé; permite que o usuário evite pedágios; apresenta a rota num mapa e em descrição passo-a-passo; ampliação de cada trecho da rota. Na função entre cidades, possibilita definir o preço do combustível, o gasto do veículo e a velocidade média; escolha entre um leque de opções para tipos de veículos; pode ser por vias principais, pelo caminho mais curto ou desviando do trânsito, e evitar pedágios. Como desvantagens do Apontador, novamente não há muita personalização e a arquitetura é proprietária, apesar do serviço ser gratuito. Também não permite pontos intermediários nas rotas. A principal vantagem é a possibilidade de rotas entre endereços específicos de cidades diferentes.

O Mappy [Map07] oferece um serviço de rotas que funciona intra e inter cidades, com as seguintes características: permite acrescentar pontos intermediários nas rotas; permite que se escolha entre veículo, pedestre ou usando serviços como ônibus, trens, etc.; no caso de se escolher utilizar os transportes públicos, o serviço é oferecido apenas para alguns países, e o usuário deve informar o horário e a data aproximados de partida; a rota pode ser expressa ou evitando pedágios; o tipo de veículo também pode

ser informado, juntamente com os dados necessários para o cálculo do consumo de combustível; apresenta os resultados em um passo-a-passo extremamente detalhado, além de visualização em forma de mapa. O Mappy oferece mais personalização que as demais ferramentas, mas ainda de forma limitada, e novamente a arquitetura é proprietária. A principal vantagem é a riqueza de informações oferecidas, como condições climáticas, placas de avisos nas estradas, localização de radares, etc.

O Map24 [M2407] oferece rotas intra e inter cidades, com as seguintes características: permite escolher um conjunto grande de categorias de estabelecimentos para serem exibidos na rota; as rotas podem ser de carro ou a pé; permite escolher entre mais rápida e mais curta; a descrição da rota pode ser compacta ou detalhada; o usuário pode escolher o nível de utilização de rodovias no trajeto: muito, normal, pouco ou evitar; o usuário tem a possibilidade de não utilizar rotas com pedágio, rodovias, barcos ou ferrovias; apresenta resultado em mapa com caminhamento 3D e descrições detalhadas. Como desvantagens, a personalização é pouca e a arquitetura é proprietária. Uma vantagem interessante é a riqueza da interface, que possui um vôo 3D seguindo a rota, além de identificação de altitude e muitas informações obtidas com facilidade. Todavia, essas vantagens são apenas da interface do cliente e não dizem respeito ao cálculo das rotas em si.

O Google Maps [GM07] apresenta um serviço simples, no qual é possível buscar rotas entre dois endereços quaisquer, estejam na mesma cidade ou não. As rotas são exibidas em mapas (nos quais é possível ampliar cada trecho e ver imagens de satélite) e através de um passo-a-passo. O serviço permite que novos destinos sejam acrescentados na rota mesmo após ela ter sido calculada, ou seja, se foi solicitada uma rota de A para B, é possível acrescentar C e tornar a rota de A para C, passando por B. Novamente a personalização é pouca, e os dados são oferecidos para acesso externo gratuito, mas isso não ocorre com o serviço de rotas, cuja arquitetura é proprietária.

O Yahoo Maps [YM07] possui um serviço de rotas muito semelhante ao do Google Maps, com o acréscimo da funcionalidade *Roundtrip*. Se um usuário faz uma rota de A para C, passando por B e pede a funcionalidade *Roundtrip*, o serviço proverá também a rota de C para A, fazendo o retorno, o que é interessante para planejamento de viagens.

Navegadores veiculares

Há um conjunto de ferramentas de navegação disponíveis no mercado brasileiro atualmente [Pin07]. O Navegador Quatro Rodas [GQR07] possui cobertura de setenta cidades do Brasil, incluindo todo o conteúdo do Guia Quatro Rodas, visão noturna e diurna e orientação por voz em quatro idiomas. Busca rotas mais curtas ou mais rápidas e permite ter como destino um endereço ou algum dos 14.000 pontos de interesse cadastrados, ou pontos incluídos pelo próprio usuário.

O navegador Airis [ABr07] possui uma quantidade de dados inferior, limitando-se a algumas cidades e a apenas 2.500 pontos de interesse. Permite que o próprio usuário cadastre locais mais visitados, locais favoritos e seus próprios pontos de interesse. Recalcula as rotas automaticamente caso uma determinada rua esteja bloqueada. Trás caminhos mais rápidos, mais curtos e para pedestres, e os comandos são por voz. Tem ainda a função de cores noturnas. Permite a realização de rotas interestaduais multiponto.

O navegador Delphi Nav200 [Ari07] possui dados de algumas cidades e 2.600 pontos de interesse (embora haja uma previsão de expansão para 30.000 pontos ainda no primeiro semestre de 2007), e também permite que o usuário acrescente seus pontos favoritos. Tem a função de visão noturna, comandos por voz, caminhos mais curtos, mais rápidos e para pedestres, recalcula rotas automaticamente em caso de bloqueio de ruas e possui um alerta de velocidade caso o usuário ultrapasse a máxima permitida. Permite a aquisição de mapas de outros países. Possibilita ainda a realização de rotas interestaduais multiponto.

O Navisystem DOTB-300 [Nav07], possui a maior cobertura, abrangendo cerca de 500 cidades brasileiras e 8.800 pontos de interesse e permite o acréscimo de novos pontos pelo usuário. Possui comandos por voz em mais de um idioma, diferencia entre navegação automóvel e pedestre, permite rotas multiponto e evitar pedágios, áreas congestionadas ou estradas perigosas.

Os sistemas permitem pouca personalização, mas são úteis principalmente pelo fato de serem móveis. Contudo, o custo de aquisição de um sistema de navegação veicular ainda é muito alto, e as atualizações dos dados precisam ser realizadas pela empresa, que pode cobrar um ônus adicional para tanto. Um serviço de rotas que fosse acessível pela Internet poderia ser acessado de um celular ou mesmo de um PDA, que tem a tela maior, e facilitaria a visibilidade das rotas.

Quiosques turísticos

O Guia Fácil [GF07] é um sistema presente em quiosques turísticos disponíveis em algumas das grandes cidades do Brasil, em hotéis de luxo e aeroportos. O sistema é instalado em um terminal que pode ser acessado por qualquer pessoa. Não há um serviço de rotas, mas é possível ver os pontos turísticos, as ruas, etc. Um serviço provedor de rotas entre as localizações seria um excelente acréscimo a uma ferramenta como essa, que tem como principal desvantagem o fato de não prover essa capacidade de buscar caminhos.

Quadro comparativo

O quadro 2.1 apresenta uma síntese das funcionalidades oferecidas pelas ferramentas acima descritas e as oferecidas pelo serviço de roteamento proposto neste trabalho. A última linha da tabela apresenta o serviço proposto. É importante ressaltar que com relação ao Acesso Móvel, utilizar objetos móveis para torná-lo completo é um trabalho a ser realizado no futuro. A principal contribuição da ferramenta reside no fato da arquitetura ser extremamente flexível e de oferecer uma alta personalização para os usuários.

Quadro 2.1 Comparação entre ferramentas

Características / Ferramentas	Rotas intra e inter cidades integradas	Nível de personalização	Arquitetura flexível	Acesso móvel	Paradas intermediárias
Maplink	Não	Médio	Não	Inexistente	Endereços quaisquer
Apontador	Sim	Médio	Não	Inexistente	Inexistente
Mappy	Sim	Médio	Não	Inexistente	Endereços quaisquer
Map24	Sim	Médio	Não	Inexistente	Inexistente

Características / Ferramentas	Rotas intra e inter cidades integradas	Nível de personalização	Arquitetura flexível	Acesso móvel	Paradas intermediárias
Google Maps	Sim	Baixo	Não	Inexistente	Endereços quaisquer
Yahoo Maps	Sim	Baixo	Não	Inexistente	Endereços quaisquer
Navegador Quatro Rodas	Não	Baixo	Não	Completo	Endereços quaisquer
Navegador Airis	Sim	Baixo	Não	Completo	Endereços quaisquer
Navegador Delphi	Sim	Baixo	Não	Completo	Endereços quaisquer
Navegador Navisystem	Não	Médio	Não	Completo	Endereços quaisquer
Guia Fácil	Não	Baixo	Não	Parcial (em quiosques)	Inexistente

2.3.3 Trabalhos Relacionados

Consultas espaciais em redes de estradas

Há um conjunto de consultas espaciais que estão sendo adaptadas aos poucos para redes de estradas. Yoo e Shekhar [YS05] apresentam o problema de encontrar o PoI (Ponto de Interesse) mais próximo a uma rota, dada a posição atual na mesma. Há duas abordagens, uma que aumenta menos o caminho e outra que desvia menos da rota. O artigo traz vários algoritmos para solucionar o problema, incluindo uma versão com materialização dos dados.

Yiu et al. [YMP05] abordam o problema de encontrar o vizinho mais próximo a um conjunto de pontos em redes de estradas. Este problema é chamado pelos autores de vizinho mais próximo agregado. Exemplo: cinco amigos estão em partes diferentes de

uma cidade e querem ir ao restaurante mais próximo a eles. A função de agregação pode variar. Exemplo: restaurante com menor distância total (soma das distâncias individuais) para todos (SUM) ou restaurante com menor distância máxima por usuário (MAX). Com SUM, um pode ter distância 2km e outro 50km, desde que a soma seja a menor. Com MAX, reduz as distâncias de todos, embora a soma seja maior que com SUM.

Shahabi et al. [SKS02] apresentam uma solução para o problema de encontrar os *k* vizinhos mais próximos em objetos móveis. Exemplo: encontrar os *k* primeiros PoIs com as menores distâncias de um ponto de consulta móvel (um carro, por exemplo). O artigo relata três maneiras de solucionar o problema de calcular as distâncias em tempo de execução: aproximar por uma função mais rápida, como distância euclidiana com indexação espacial, uma abordagem ruim, já que a margem de erro é grande; pré-computar todas as distâncias, que, de acordo com os autores não dá certo numa rede dinâmica de objetos móveis; e aproximar as distâncias com um espaço multidimensional e métricas de Minkowski. Os autores detalham no artigo a última abordagem, e dizem que os valores de distâncias podem ser diferentes dos originais, mas a ordem dos resultados é preservada, o que é o foco neste caso. É importante destacar que não é realmente impossível optar pela segunda abordagem: se fossem materializadas as distâncias entre todos os vértices da rede, estáticos, e os PoIs, também fixos, quando um objeto móvel solicitasse um cálculo, seria suficiente encontrar qual dos vértices ao seu alcance direto possui a menor distância para o devido PoI.

Gupta et al. [GKR04] propõem uma solução para o problema de calcular consultas espaciais (de faixa, de junção ou de interseção) em redes de estradas. A solução apresentada é associar códigos binários aos nós e utilizar a distância de Hamming para saber a distância entre eles. A distância de Hamming é definida como o número de posições em que os valores são diferentes. Por exemplo, a distância entre 101 e 111 é 1, pois apenas o segundo bit é diferente.

Além de usar distância de Hamming, a solução proposta embute o grafo planar num hipercubo com muitas dimensões. Toda essa etapa é realizada previamente, restando em tempo de execução apenas o cálculo da distância de Hamming. A distância entre dois nós adjacentes é um, então eles têm códigos que diferem em apenas um bit. E assim por diante, ou seja, dois nós com distância quatro possuem quatro bits diferentes. O processo não leva em consideração os aspectos dinâmicos das redes de estradas, como o tráfego. A Figura 2.9 tem um exemplo do mapeamento.

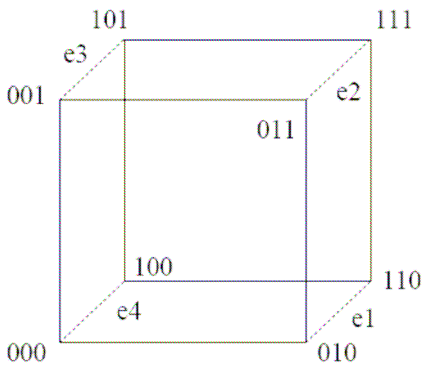

Figura 2.9 Códigos de Hamming (fonte: [GKR04])

Menor caminho em redes de estradas

Liu [Liu96] aborda técnicas de Inteligência Artificial para melhorar o cálculo de rotas. Então se utiliza de bases de conhecimento e raciocínio baseado em casos para inferir informações acerca das preferências do usuário e fazer um corte no espaço de busca. Baseia-se em quatro comportamentos descobertos em estudos: as pessoas preferem viajar nas estradas principais; o processo de busca de rotas das pessoas é realizado minimizando o ângulo entre a direção atual e a direção para o destino; as pessoas gostam de viajar em rotas familiares; as pessoas aprendem de experiências de viagem. Com base nesses conhecimentos e no histórico do usuário, o espaço de busca é restringido e então aplica-se o algoritmo A*.

Richter et al. [RKF04] afirmam que caminhos mais curto e mais rápido são fáceis de calcular e levam o usuário ao destino, mas as instruções das mesmas são inadequadamente apresentadas e difíceis de seguir, pois desrespeitam os requisitos do usuário e o conhecimento no processamento da informação da rota. Dois conjuntos de instruções de rotas podem ser considerados idênticos se levam da mesma origem para o mesmo destino pela mesma rota. Porém, podem ser pragmaticamente diferentes para diferentes usuários, visto que a maneira de apresentar as instruções de um dos conjuntos pode ser mais clara para alguns usuários e do outro conjunto para outros usuários, pois as pessoas diferem de acordo com seu conhecimento do local, etc. Por exemplo, “siga em frente, em frente novamente e então vire à direita” pode ser muito diferente de “vire à direita no terceiro cruzamento”, mas levam à mesma rota. Os autores afirmam que é interessante na descrição de rotas utilizar-se de: referências globais, como os pontos

cardeais, montanhas ou o mar; características da estrutura, como a inclinação das ruas (por exemplo, "desça a colina"); monumentos (como prédios, igrejas, estátuas, praças, etc.), placas de trânsito, e os pontos de decisão, ou seja, interseções onde o usuário deve decidir por onde ir. Exemplos: "siga o rio até encontrar a igreja"; "vire à esquerda após a praça"; "após passar o museu, procure um McDonalds e dobre à direita após passar por ele".

Kanoulas et al. [KDX+06] destacam a importância de levar em consideração os dados de tráfego ao computar rotas. Sua abordagem é baseada em rotas mais rápidas, de forma que a média de velocidade é definida para cada trecho de rua, de acordo com o horário e o dia da semana. Através de um intervalo de tempo viável para iniciar o caminho, retorna o melhor horário para obter-se a rota mais rápida. Contudo, a desvantagem desta abordagem consiste em ser baseada em condições de tráfego pré-armazenadas, o que funciona bem em horários de pico, mas não consideram congestionamentos aleatórios e obstruções das estradas.

Sharifzadeh e Shahabi [SS06] definem a *Optimal Sequenced Route* (OSR – Rota Seqüencial Ótima), que consiste de uma consulta onde a ordem do PoI é importante. Por exemplo, um usuário pode querer consultar uma rota que passa primeiro em um banco e em seguida em um supermercado. O artigo apresenta algoritmos para caminhos com essas restrições, que habilitam o usuário a escolher os PoIs em uma ordem especificada ou não. Entretanto, o usuário não pode acrescentar nenhuma restrição à rota em si, apenas que ela deve passar por estes pontos de interesse.

Cálculo do menor caminho com restrições

Um menor caminho com restrição é um caminho entre dois pontos com alguma restrição que complique a computação da resposta. Terrovitis et al. [TBP+05] apresentam soluções para dois tipos de menores caminhos com restrições, em espaço euclidiano.

Dada uma coleção de pontos e uma constante *a* (chamada constante de autonomia), um caminho *a-autonomy* de *S* para *D* é uma seqüência de caminhos $P1 \rightarrow P2 \rightarrow ... \rightarrow Pn$, onde cada *Pn* é um caminho entre dois pontos e *P1* deve iniciar-se em *S* e *Pn* deve terminar em *D*, com a distância entre quaisquer dois pontos consecutivos não ultrapassando *a*. Por exemplo, um avião deve fazer um vôo da cidade *A* para a cidade *B*, cuja distância é *d*. Se *a* é a distância máxima que o avião deve

percorrer sem abastecer (a autonomia do avião) e $d > a$, que aeroportos intermediários devem ser escolhidos para minimizar a distância total do vôo?

Dada uma coleção de pontos e uma constante *k*, um caminho *k-stops* entre *S* e *D* é uma seqüência de caminhos *P1*→*P2*→...→*Pk*, onde cada *Pn* é um caminho entre dois pontos e *P1* deve iniciar-se em *S* e *Pn* deve terminar em *D*, com exatamente *k* pontos intermediários. Um exemplo de uso do *k-stops* é um veículo de entregas que obtém as cargas num ponto s e termina num ponto *d*. No caminho tem que entregar em *k* clientes, onde *k* depende da capacidade de carga. Que *k* clientes deveria escolher para minimizar a distância total da viagem?

A meta é minimizar ambos os caminhos, ou seja, no caminho *a-autonomy* a distância entre dois pontos deve ser menor que *a*, mas o mais próxima possível deste. E em *k-stops*, deve-se encontrar os *k* pontos mais próximos do menor caminho. Terrovitis et al. propõem um algoritmo genérico, que primeiro encontra uma solução sub-ótima com um algoritmo rápido, depois usa esse resultado para restringir o espaço de busca e então computa o menor caminho usando o espaço restante. Para a busca da solução sub-ótima, traz dois algoritmos pra cada tipo de caminho e para a busca ótima, utiliza A* em *a-autonomy* e Bellman-Ford [Bel68, FF62] em *k-stops*.

A limitação desta abordagem se concentra no fato de estar voltada para problemas em espaço euclidiano, como por exemplo o cálculo das escalas de um avião dada a autonomia em quilômetros que ele pode passar no ar sem reabastecer. Contudo, se for desejada uma consulta em redes de estradas, a solução proposta não resolve o problema de forma adequada.

Penalização de curvas

É importante levar em conta o custo das curvas ao planejar um caminho. Curvas são consideradas complicadores das rotas, pois para cada curva uma nova instrução deve ser dada. Há ainda o custo da desaceleração para efetuar uma curva, das leis de trânsito, do tempo de espera para fazer a conexão (se há carros em outro sentido), do raio da curva, caso seja um veículo longo, etc.

Caldwell [Cal61] e Winter [Win01, Win02] propõem a inversão do grafo de ruas, ou seja, as arestas tornar-se-iam nós e vice-versa. Dessa forma, as curvas seriam arestas e poderiam receber um determinado peso. Contudo, essa estratégia funciona apenas para quando for desejado aplicar um custo somente aos vértices, falhando se for

necessário um custo para as curvas e outro para os trechos das ruas.

Duckham e Kulik [DK03] defendem que nem sempre o caminho mais curto é o melhor, pois o usuário pode não conhecer bem a área e os caminhos mais curtos por vezes fazem uso de ruas estreitas e pouco utilizadas. Os autores sugerem então o uso do caminho mais simples, do inglês *Simplest Path*, ou seja, o caminho que possua maior simplicidade. É preferível fazer menos curvas e andar um pouco mais do que fazer curvas com muitas interseções.

Kirby e Potts [KP69] descrevem um caminho como uma seqüência de arcos ao invés de uma seqüência de nós, de maneira que seja possível definir o custo do caminho tanto em termos dos custos dos arcos quanto das penalidades para cada par de arcos.

Modelagem de redes de estradas

Speicys et al. [SJK03] apresentam uma estratégia de modelagem de dados que envolvem objetos móveis, PoIs e redes de estradas. Os autores se utilizam de dois tipos de modelos: uma representação 2D e uma em forma de grafo. A representação 2D armazena segmentos e conexões. Os segmentos guardam as posições geográficas de origem e de destino, o sentido do segmento (da origem para o destino, do destino para a origem, bidirecional sem retorno ou bidirecional com retorno) e propriedades adicionais (por exemplo, limite de velocidade). As conexões armazenam sua posição geográfica, os segmentos que se encontram com a conexão e a matriz de conexão (para cada par de segmentos, se é possível trafegar do primeiro para o segundo por essa conexão). Cada PoI tem um conjunto de características que o descrevem, e sua localização (ponto geográfico, segmento e direção na qual é acessível – da origem para o destino, do destino para a origem ou ambos – cada PoI pode ter várias entradas, então várias localizações). Cada ponto de consulta tem sua localização geográfica, o segmento e a direção que está tomando (origem para o destino, destino para a origem ou indefinida).

A representação em grafo armazena os vértices, as arestas e as co-arestas. Cada aresta tem um vértice de origem, um de destino, seu peso e seu tamanho. A relação co-aresta captura pares de arestas que representam a mesma rua, uma indo e uma vindo, onde é possível fazer retorno. Cada PoI guarda a aresta onde se encontra, a posição no peso (distância em unidades de peso da origem) e a posição no tamanho. Cada ponto de consulta armazena a aresta, a posição no peso, a posição no tamanho, a velocidade (unidades de peso por tempo) e o instante de tempo quando os dados foram capturados

pela última vez.

A representação é muito rica, pois leva em consideração o sentido das ruas, a possibilidade de retorno, o tráfego e a condição da pavimentação. É possível armazenar o custo de cada curva nas conexões, junto à matriz de permissão de trânsito entre dois segmentos. Por fim, os autores abordam uma forma de transformação entre os dois tipos de modelagens.

Vazirgiannis e Wolfson [VW01] propõem uma estratégia de modelagem para objetos móveis em redes de estradas, além de uma linguagem para realização de consultas. É um artigo muito voltado para a implementação em bancos de dados espaço-temporais, com proposta de comandos SQL (por exemplo, *always between*, *along path*, etc.) e estruturas de indexação. Infelizmente o artigo está desatualizado, pois foi voltado para o uso de Oracle 8. Com as versões 9i e 10g, o SGBD ganhou muitas funcionalidades, que cobrem parte do que foi sugerido no artigo.

Jiang e Claramunt [JC04] propõem um novo modelo de generalização para selecionar ruas características numa rede de estradas. Utilizam-se de medidas, como o grau de centralidade de um nó, o fechamento da centralidade e a centralidade intermediária. O grau de centralidade de um nó também pode ser definido como o número de nós que se conectam com ele através de arestas. O fechamento da centralidade de um nó pode ser definido como a divisão do número total de nós (subtraindo o nó atual) dividida pelo somatório das distâncias do nó atual para todos os demais do grafo. Já a centralidade intermediária de um nó representa o quanto esse nó faz parte de caminhos entre nós, ou seja, é um intermediário de outros caminhos.

A estratégia apresentada pode ser utilizada para encontrar as ruas principais de um grafo, ruas essas que podem receber algum destaque diferente nos cálculos, como por exemplo, velocidade média mais alta.

Timpf e Heye [TH02] propõem medidas de complexidade para descrever as características físicas de nós, arestas e rotas, ou seja, quão difícil uma rota seria para um viajante. O trabalho é focado em rotas multi-meio, ou seja, pedestres que andam trechos a pé, pegam metrô em outros, ônibus, etc.

Materialização de caminhos

Jing et al. [JHR98, JHR96] apresentam uma estratégia chamada HEPV (*Hierarchical Encoded Path Views*) para busca do melhor caminho materializando

parcialmente os caminhos pré-calculados, e dividindo o grafo em hierarquias. A idéia começou a ser desenvolvida por Huang et al. [HJR95, HJR97]. Materializar todas as possibilidades ocupa muito espaço, além do que, em sistemas de navegação, é mais importante saber o próximo passo do que o caminho completo. Dessa forma, a materialização é realizada de forma parcial, ou seja, armazenando apenas o nó destino, o próximo nó no caminho e o peso total do caminho. Ainda assim, a materialização ocupa muito espaço ($O(n^2)$ no pior caso, onde n é o número de nós e se cada nó é alcançável a partir do outro no grafo). Portanto, divide-se o grafo em uma partição de fragmentos e hierarquiza-se utilizando um super-grafo.

Partição é um conjunto de fragmentos do grafo. Cada fragmento é um conjunto de vértices e arestas, de modo que uma determinada aresta só faça parte de um fragmento e o conjunto de todos os nós e arestas dos fragmentos seja igual ao grafo original. Nós que fazem parte de mais de um fragmento são chamados de nós-borda. Então um super-grafo é o conjunto de todos os nós-borda da partição, e se existe um caminho entre eles no grafo original, um arco é criado para representá-lo no super-grafo, com o custo igual ao total do caminho, conforme pode ser visto na figura 2.10. Quando o peso de uma aresta muda, é necessário atualizar apenas o fragmento ao qual ela pertence e o super-grafo.

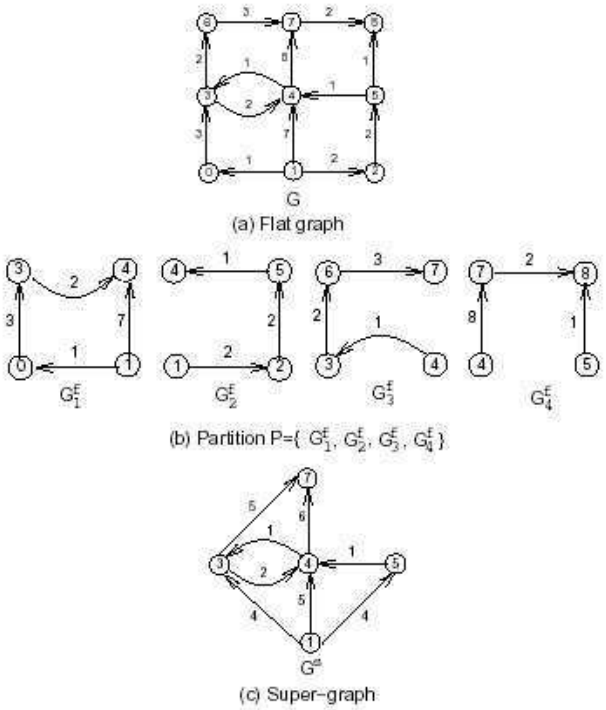

Figura 2.10 Criação do super-grafo (fonte: [JHR96])

Também há o conceito de hierarquias. Por exemplo, é possível dividir uma cidade em bairros, então cada cidade pode ser tratada como uma partição onde os bairros são os fragmentos. Mas num nível acima da hierarquia podemos enxergar cada cidade como um fragmento de uma grande partição de cidades.

A busca do melhor caminho se resume a cinco possibilidades:

- Os nós fonte e destino fazem parte do super-grafo. Nesse caso, basta retornar o caminho entre eles;
- O nó fonte não faz parte do super-grafo, mas o destino faz. Então um nó intermediário do super-grafo deve ser encontrado, nó esse que deve pertencer ao mesmo fragmento do nó fonte. Então o caminho do fonte ao intermediário somado ao caminho do intermediário ao destino é retornado;
- O nó fonte faz parte do super-grafo, mas o destino não. Basta inverter a situação anterior e proceder de forma semelhante;
- Nenhum dos nós faz parte do super-grafo, mas fazem parte do mesmo fragmento. Basta retornar o melhor caminho entre eles;
- Nenhum dos nós faz parte do super-grafo, e estão em fragmentos diferentes. Então um nó-borda do fragmento de origem e um do fragmento de destino devem ser encontrados, e o melhor caminho vai ser a junção do melhor caminho do nó-fonte ao nó-borda escolhido do seu fragmento com o melhor caminho entre os nós-borda e o melhor caminho do nó-borda do fragmento de destino ao nó final.

Em alguns casos, há necessidade de ter-se mais níveis na hierarquia. Nesse caso, o procedimento se repete. O super-grafo é particionado e um novo super-grafo, do nível acima, é encontrado. Um exemplo de hierarquização em três níveis pode ser visto na figura 2.11. As possibilidades para busca do caminho aumentam, pois agora são vários níveis e fragmentos, então a origem pode estar em um fragmento do nível mais inferior, e o destino em outro, e para obter a rota seria necessário caminhar para o nível intermediário, para o superior e fazer o caminho inverso. Mas, apesar do aparente aumento da complexidade, a idéia é a mesma.

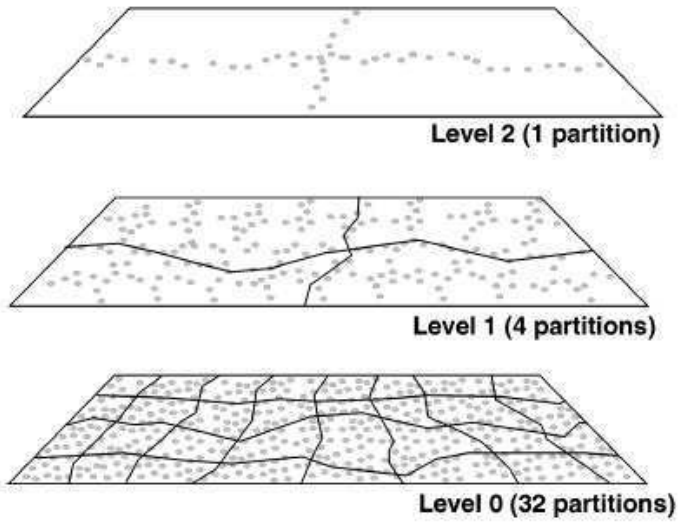

Figura 2.11 Exemplo de hierarquia em três níveis (fonte: [JHR98])

Jung e Pramanik [JP02] apresentam uma estratégia de materialização chamada HiTi, de forma a reduzir o tempo na busca de uma rota. Utiliza a mesma idéia do HEPV, ou seja, o particionamento do grafo em hierarquias, com uma espécie de super-grafo. A estratégia parece ser mais eficiente, mas é também mais complexa. HiTi é ainda muito dependente de vários níveis de hierarquia. Tanto HEPV como HiTi são interessantes para acelerar o processo de materialização e reduzir o espaço de armazenamento.

Shekhar et al. [SFG97] apresentam uma análise das estratégias de materialização e de hierarquias. A materialização total tem um custo muito alto, o que leva os autores a proporem uma materialização parcial, no sentido de armazenar apenas algumas tabelas e outras não. Os autores ainda abordam a necessidade de melhorar a velocidade de acesso, colocando dados na memória. Mas a memória é limitada, então apenas algumas das visões materializadas vão para a memória, enquanto outros ficam em disco. Após análises de desempenho, o artigo sugere quais visões materializar e quais se deve colocar na memória.

Com o crescimento dos discos rígidos, armazenamento não tem sido mais um problema, e não é mais um custo alto realizar a materialização total dos dados, como sugerido pelos artigos referenciados. A hierarquização ainda é uma abordagem interessante, pois aumenta muito pouco o custo de processamento unir os trechos de caminhos dos diferentes fragmentos para obter o resultado. Mas o custo de

armazenamento total é pequeno em comparação com a perca em desempenho e o aumento em complexidade da materialização parcial.

Serviços móveis

Almeida e Güting [AG05] abordam estratégias de indexação das trajetórias de objetos móveis em redes de estradas. Os autores apresentam estruturas de indexação que contemplem objetos móveis em redes de estradas, principalmente fazendo extensões a estruturas de indexação espacial, como *R-Tree*, e aplicando comportamento temporal.

Balke et al. [BKU03a, BKU03b] apresentam um serviço *wireless* que provê planejamento de rotas, para celular ou PDA. O usuário diz suas preferências em grau de importância – sem importância, fraco, médio, importante ou muito importante – e o software escolhe a rota que mais se adeqüe. O serviço coleta dados de diferentes fontes e *Web Services*, usando a técnica de recuperação textual *SR-Combine Top-k*. De acordo com os autores, a resposta é obtida em apenas 3 segundos, com cliente iMode e servidor IBM DB2.

O artigo destaca ainda a importância de se levar em consideração a preferência dos usuários. Entre as possibilidades relatadas, estão o tamanho da rota, a quantidade de congestionamentos e os tamanhos dos mesmos, se há obras nas ruas, as condições climáticas, a qualidade da rota e a probabilidade estatística de haver congestionamento. O usuário seleciona para cada uma dessas características o seu grau de importância e informa quantas rotas deseja obter. Dessa forma, as k rotas mais adequadas são obtidas e apresentadas. O artigo não detalha o algoritmo para realização da busca das rotas, apenas explica superficialmente o funcionamento.

Brilingaite et al. [BJZ05] propõem um serviço que, uma vez instalado em um celular com GPRS e GPS, armazena as rotas realizadas pelo usuário, e o alerta de possíveis problemas de tráfego, condições climáticas, etc., além de encontrar PoIs próximos à rota. O artigo detalha a primeira parte, de obtenção da rota em tempo de execução, ou seja, enquanto ela é percorrida. Utiliza Java e Oracle 9i Spatial, com PL/SQL. A proposta está ainda incompleta, prejudicando uma avaliação mais adequada.

Shekhar e Liu [SL94] apresentam uma interessante justificativa para fazer sistemas de transporte em dispositivos móveis: barateamento da tecnologia; dispositivos móveis com grande capacidade de processamento; e os viajantes são móveis, então seus sistemas também devem ser.

Torrens et al. [THP+04] abordam o *reality*, um gerador de planos de viagem, ou seja, organiza um conjunto de atividades com local, início e fim e divididas em tarefas e sub-tarefas. Define a melhor ordem de realização das atividades, e se preocupa com as preferências do usuário. Utiliza técnicas de Inteligência Artificial na implementação. Uma sugestão interessante seria o acréscimo de um software para busca de rotas, de forma que auxiliasse os usuários nas cidades que visitarem.

Outro serviço para turistas é apresentado por Malaka e Zipf [MZ00]. O DEEP MAP é um serviço baseado em agentes que apresenta informações, rotas e etc. de acordo com as necessidades do usuário. O turista pode escolher seu meio de transporte e passar suas necessidades via texto. Há um processador de linguagem natural para entender os questionamentos do usuário e há grande preocupação com a interface. A abordagem poderia ser explicada de forma mais aprofundada.

Outras abordagens

Corona e Winter [CW01a, CW01b] apresentam a necessidade de realizar-se um estudo comparativo entre as necessidades de motoristas e de pedestres no uso de sistemas de navegação. As necessidades dos pedestres são diferentes, de acordo com os autores, e devem ser mais bem elucidadas. O estudo levanta ontologias para sistemas de navegação voltados para motoristas e para os voltados para pedestres, tentando elencar o que importa em cada um deles. Uma interessante constatação é a de que os pedestres trafegam mais vagarosamente e, portanto, vêem mais detalhes. São capazes de encontrar referências com mais facilidade, têm o tempo a seu favor.

Golledge [Gol95] afirma que as pesquisas na área de roteamento são muito computacionais, sem se preocupar se os humanos realmente gostariam que fosse dessa maneira. A navegação feita por humanos é com medidas inexatas, pois não temos a precisão das máquinas. O autor realizou experimentos em laboratório com pessoas selecionando rotas em mapas de papel e em mapas digitais. Um dos tipos de rota levantados pelo autor é a rota *scenic*, ou turística, que passa em mais pontos turísticos.

Quadro comparativo

O quadro 2.2 apresenta uma análise comparativa entre alguns dos trabalhos acima relacionados.

Quadro 2.2 Comparativo entre os trabalhos relacionados

Características / Artigos	Redes de estradas	Paradas intermediárias em pontos de interesse	Consulta pelo mais próximo	Busca de rotas	Utiliza informações do usuário	Preocupação com as instruções das rotas	Leva dados de tráfego em consideração	Considera os custos das curvas	Materialização	Objetos móveis
Yoo e Shekhar	S	S	S	N	N	N	N	N	S	S
Yiu et al.	S	N	S	N	N	N	N	N	N	S
Shahabi et al.	S	N	S	N	N	N	N	N	N	S
Gupta et al.	S	N	N	N	N	N	N	N	N	N
Liu	S	N	N	S	*	N	N	N	N	N
Richter et al.	S	N	N	N	N	S	N	N	N	N
Kanoulas et al.	S	N	N	S	N	N	S	N	N	N
Sharifzadeh e Shahabi	S	S	N	S	N	N	N	N	N	N
Terrovitis et al.	N	S	N	S	N	N	N	N	N	N
Caldwell	S	N	N	N	N	N	N	S	N	N
Winter	S	N	N	N	N	N	N	S	N	N
Duckham e Kulik	S	N	N	S	N	N	N	S	N	N
Kirby e Potts	S	N	N	N	N	N	N	S	N	N
Jing et al.	S	N	N	S	N	N	N	N	S	N
Jung e Pramanik	S	N	N	S	N	N	N	N	S	N
Shekhar et al.	S	N	N	S	N	N	N	N	S	N
Almeida e Güting	S	N	N	N	N	N	N	N	N	S
Balke et al.	S	N	N	S	**	N	N	N	N	N
Brilingaite et al.	S	N	N	N	N	N	S	N	S	S
Shekhar e Liu	S	N	N	N	N	N	N	N	N	S
Malaka e Zipf	S	N	N	S	N	N	N	N	N	N

* Histórico ** Personalização

Capítulo 3 . Busca de Caminhos Personalizada e com Restrições

O objetivo deste trabalho é desenvolver uma arquitetura orientada a serviços para busca de rotas entre duas localizações de acordo com as preferências do usuário, possibilitando-lhe a fazer restrições. O usuário poderá personalizar a sua rota com um conjunto de características e a ferramenta deverá retornar para o usuário a rota que melhor se adequar às suas preferências. Além disso, o usuário poderá definir algumas restrições para o caminho encontrado.

Neste capítulo serão apresentados: na seção 3.1, o *Coolest Path*, o algoritmo para busca de rotas de acordo com a personalização do usuário; e na seção 3.2, os caminhos com restrição *a-autonomy* e *t-autonomy* para redes de estradas.

3.1 Coolest Path

Foram levantadas quatro características personalizáveis, que o usuário pode avaliar como: pouca importância; média importância; e alta importância. As características levantadas servem para exemplificar o trabalho, mas outras podem ser facilmente acrescentadas. São elas: tamanho (o usuário pode dizer o quão curto ele deseja seu caminho); simplicidade (o quão simples – com menos interseções e decisões confusas – o usuário deseja seu caminho); turismo (o quão turístico o usuário deseja seu caminho – nesse caso busca-se passar por pontos turísticos); e tempo (o quão rápido o usuário deseja chegar ao destino – essa característica depende de muitas outras, como da qualidade da pavimentação das ruas).

A proposta do serviço é oferecer ao usuário o *Coolest Path*, ou seja, o caminho mais agradável, de acordo com as preferências do usuário, totalmente personalizado. Nas próximas seções serão apresentados algoritmos para encontrar caminhos com base em cada uma dessas características, uma descrição do problema e o algoritmo do *Coolest Path*, que reúne todas as características e seus níveis de personalização.

3.1.1 Algoritmos Auxiliares

Esta seção vai apresentar quatro algoritmos simples para busca de rotas, cada um baseado em uma característica principal. O primeiro deles é o *Shortest Path*, que tem como meta encontrar o caminho mais curto entre uma origem e um destino.

Dado um grafo G = (V, E) e uma função w(e) associada a cada aresta e ∈ E, um caminho P = (V', E'), tal que V' = $\{x_0, x_1, ..., x_k\}$ e E' = $\{e_1, e_2, ..., e_k\}$, onde os x_i são distintos e $e_1 = x_0x_1$, $e_2 = x_1x_2$, ..., e $e_k = x_{k-1}x_k$, é considerado um caminho mais curto (ou *Shortest Path*) se w(P) tiver o menor valor para todos os possíveis caminhos P entre x_0 e x_k, desde que a função w(e) represente a distância a ser percorrida na aresta e.

O algoritmo 3.1 apresenta uma função auxiliar, utilizada em vários dos algoritmos, para extrair o caminho e retorná-lo ao usuário. Basicamente percorre os vértices, partindo do destino para a origem caminhando através da propriedade *pai* (linha 5), que indica o vértice anterior a este no caminho. Os vértices são adicionados e uma pilha (linhas 3 e 6) que, ao final, deve apenas ser desempilhada pelo cliente, pois já está na ordem correta. O algoritmo para *Shortest Path* é baseado em A* e usa como heurística a distância Manhattan (que pode ser calculada como no algoritmo 3.2).

```
1  ExtraiCaminho (origem, destino) {
2    v = destino;
3    caminho.add(v);
4    enquanto (v != origem) {
5      v = v.pai;
6      caminho.add(v);
7    }
8    return caminho;
9  }
```

Algoritmo 3.1 Função para extrair o caminho

```
1  Heuristica (origem, destino) {
2        return |origem.X - destino.X| + |origem.Y -
destino.Y|;
3  }
```

Algoritmo 3.2 Função heurística utilizando distância Manhattan

O algoritmo *Shortest Path* pode ser observado no algoritmo 3.3. Inicialmente, são calculados e definidos os valores g, h e f do vértice de origem (linhas 2 a 4) e este é acrescentado a uma lista de nós em aberto (linha 5). Em seguida (linhas 6 a 29), o algoritmo percorre todos os vértices em aberto, daqueles de menor custo (de acordo com o valor de f) para os de maior (linha 7). Uma vez que o vértice é percorrido, ele é acrescentado à lista de nós fechados (linha 8). Caso o vértice de destino seja encontrado, o caminho é retornado (linhas 9 e 10). Para cada vértice adjacente ao nó sendo visitado, seu pai é definido como o visitado, os custos até o momento são somados, a heurística para o destino é calculada e o vértice é adicionado à lista aberta, caso não esteja lá (linhas 14 a 27). Ao final, o caminho é retornado (linha 30).

```
1  ShortestPath (origem, destino) {
2    origem.G = 0;
3    origem.H = Heuristica(origem, destino);
4    origem.F = origem.G + origem.H;
5    listaAberta.add(origem);
6    enquanto (listaAberta não for vazia) {
7      v = listaAberta.menor;
8      listaFechada.add(v);
9      se (v = destino)
10        return ExtraiCaminho(origem, destino);
11      para cada (adjacente = adjacente de v) {
12        se (adjacente estiver na listaFechada)
13          próxima iteração;
14        se (adjacente não estiver na listaAberta) {
15          adjacente.pai = v;
16          adjacente.G = v.G + custo(v, adjacente);
17          adjacente.H = Heuristica(adjacente, destino);
18          adjacente.F = adjacente.G + adjacente.H;
19          listaAberta.add(adjacente);
20        } senão {
21          se (v.G + custo(v, adjacente) < adjacente.G) {
22            adjacente.pai = v;
```

```
23            adjacente.G = v.G + custo(v, adjacente);
24            adjacente.H = Heuristica(adjacente, destino);
25            adjacente.F = adjacente.G + adjacente.H;
26          }
27        }
28      }
29    }
30    return ExtraiCaminho(origem, destino);
31  }
```

Algoritmo 3.3 *Shortest Path*

Dado um grafo G = (V, E) e uma função w(x) associada a cada vértice $x \in V$, um caminho P = (V', E'), tal que V' = $\{x_0, x_1, ..., x_k\}$ e E' = $\{e_1, e_2, ..., e_k\}$, onde os x_i são distintos e $e_1 = x_0x_1$, $e_2 = x_1x_2$, ..., e $e_k = x_{k-1}x_k$, é considerado um caminho mais simples (ou *Simplest Path*) se w(P) tiver o menor valor para todos os possíveis caminhos P entre x_0 e x_k, desde que a função w(x) represente o custo para se realizar a transição entre duas arestas pelo vértice x.

O custo de transição entre duas arestas é definido adaptando os custos propostos por Duckham & Kulik [DK03], conforme pode ser observado na figura 3.1.

Figura 3.1 Custos das transições entre arestas

O algoritmo para *Simplest Path* é baseado no proposto por Duckham & Kulik [DK03], como pode ser observado no algoritmo 3.4. Inicialmente, o algoritmo define o custo da origem para cada aresta como infinito (linhas 2 a 4). Em seguida, para cada

aresta adjacente à origem, define o custo como sendo zero (linhas 5 a 8). Os passos seguintes se referem à execução do algoritmo propriamente dito (linhas 9 a 17), que são repetidos enquanto houverem arestas não visitadas. O algoritmo tem início selecionando a aresta com menor custo da origem (linha 10) e adicionando-a à lista de visitadas (linha 11). Então, o algoritmo percorre cada uma das adjacentes dessa aresta de menor custo (linhas 12 a 16), definindo seu novo custo como sendo o da menor somado com o custo dela para a adjacente (linha 14), caso esse custo seja menor que o atual da adjacente (linha 13). Ao final, o algoritmo chama uma função para extrair o caminho (linha 18), que pode ser observada no algoritmo 3.5 e é semelhante à do algoritmo 3.1.

```
1  SimplestPath (origem, destino) {
2    para cada (e = aresta) {
3      e.custodaOrigem = infinito;
4    }
5    para cada (vizinho = adjacente da origem) {
6      e = (origem, vizinho) // aresta entre eles
7      e.custodaOrigem = 0;
8    }
9    enquanto (arestas\visitadas > 0) {
10      menor = (arestas\visitadas).menor;
11      visitadas.add(menor);
12      para cada (arestaVizinha = adjacente a menor) {
13        if (arestaVizinha.custodaOrigem >
          (menor.custodaOrigem + custo(menor,
          arestaVizinha)) {
14          arestaVizinha.custodaOrigem =
            menor.custodaOrigem + custo(menor,
            arestaVizinha);
15        }
16      }
17    }
18    return ExtraiCaminhoSimplest(origem, destino);
19  }
```

Algoritmo 3.4 *Simplest Path*

```
1  ExtraiCaminhoSimplest (origem, destino) {
2    v = destino;
3    caminho.add(v);
4    enquanto (v != origem) {
5      v = menorCustoTerminandoEm(v).inicial;
6        caminho.add(v);
7    }
8    return caminho;
9  }
```

Algoritmo 3.5 Função para extrair o caminho do *Simplest Path*

O *Scenic Path* tem como meta encontrar o caminho da origem ao destino passando pela maior quantidade de pontos turísticos. Contudo, não é interessante uma rota que passe por todos os pontos turísticos da cidade e isso não vai acontecer porque o caminho é encontrado utilizando um algoritmo baseado em A* e a heurística é a distância Manhattan, ou seja, o caminho segue na direção do destino, passando pelo máximo de pontos turísticos, mas respeitando a orientação do destino. Isso ocorre porque o custo de cada aresta é a soma da distância Manhattan com o custo turístico (sendo este normalizado e invertido) e a aresta com menor soma é escolhida. Então, a quantidade de pontos turísticos é importante, mas a distância Manhattan não permite que o caminho se desvie muito de sua orientação.

Dado um grafo G = (V, E) e uma função w(e) associada a cada aresta e $\in$ E, um caminho P = (V', E'), tal que V' = $\{x_0, x_1, ..., x_k\}$ e E' = $\{e_1, e_2, ..., e_k\}$, onde os x_i são distintos e $e_1 = x_0x_1$, $e_2 = x_1x_2$, ..., e $e_k = x_{k-1}x_k$, é considerado um caminho turístico (ou *Scenic Path*) se w(P) tiver o maior valor para todos os possíveis caminhos P entre x_0 e x_k que não se desviarem demais – ou seja, respeitarem a soma do custo turístico com a distância Manhattan de valor mínimo – da orientação para o destino, desde que a função w(e) represente a quantidade de pontos turísticos da aresta e.

O algoritmo para *Scenic Path* pode ser observado no algoritmo 3.6. Inicialmente, o vértice de origem tem seus pesos calculados e definidos (linhas 2 a 4) e é acrescentado a uma lista de nós em aberto (linha 5). Em seguida (linhas 6 a 29), o algoritmo percorre todos os vértices em aberto, daqueles de maior custo para os de menor (linha 7), pois

quanto mais pontos turísticos, melhor. Uma vez que o vértice é percorrido, ele é acrescentado à lista de nós fechados (linha 8). Caso o vértice de destino seja encontrado, o caminho é retornado (linhas 9 e 10). Para cada vértice adjacente ao nó sendo visitado, seu pai é definido como o visitado, os custos até o momento são somados, a heurística para o destino é calculada e o vértice é adicionado à lista aberta, caso não esteja lá (linhas 14 a 27). É importante ressaltar que no *Scenic Path*, quanto maior o custo, melhor, de forma diferente do *Shortest Path* (conforme o sinal de maior (>) na linha 21). Ao final, o caminho é retornado (linha 30).

```
1  ScenicPath (origem, destino) {
2    origem.G = 0;
3    origem.H = Heuristica(origem, destino);
4    origem.F = origem.G + origem.H;
5    listaAberta.add(origem);
6    enquanto (listaAberta não estiver vazia) {
7      v = listaAberta.maior;
8      listaFechada.add(v);
9      se (v = destino)
10        return ExtraiCaminho(origem, destino);
11     para cada (adjacente = adjacente de v) {
12       se (adjacente estiver na listaFechada)
13         próxima iteração;
14       se (adjacente não estiver na listaAberta) {
15         adjacente.pai = v;
16         adjacente.G = v.G + custoTuristico(v,
           adjacente);
17         adjacente.H = Heuristica(adjacente, destino);
18         adjacente.F = adjacente.G + adjacente.H;
19         listaAberta.add(adjacente);
20       } senão {
21         se (v.G + custoTuristico(v, adjacente) >
           adjacente.G) {
22           adjacente.pai = v;
23           adjacente.G = v.G + custoTuristico(v,
```

```
                adjacente);
24              adjacente.H = Heuristica(adjacente, destino);
25              adjacente.F = adjacente.G + adjacente.H;
26            }
27          }
28        }
29      }
30      return ExtraiCaminho(origem, destino);
31    }
```

Algoritmo 3.6 *Scenic Path*

O *Fastest Path* tem como objetivo encontrar o caminho mais rápido entre uma origem e um destino. Contudo, para deduzir o tempo necessário para realizar o percurso de um caminho é necessário levar em consideração um conjunto de características:

- A distância percorrida em cada trecho do percurso;
- A velocidade média para aquele trecho;
- A qualidade da pavimentação do trecho;
- A existência de tráfego no local;
- A existência de interrupções (obras, eventos, etc.) no trecho em questão.

A distância percorrida em cada aresta é calculada pelo tamanho em metros do trecho da rua. A velocidade média é medida em quilômetros por hora. Dessa forma, para encontrar o tempo basta dividir a distância pela velocidade. Os demais fatores levados em consideração serão multiplicados pela velocidade antes do cálculo do tempo médio de percurso.

A qualidade da pavimentação de um trecho de rua pode pertencer a uma das categorias a seguir, com seus respectivos pesos:

- Asfalto de alta qualidade: 1,00;
- Asfalto sem buracos: 0,95;
- Asfalto com poucos buracos: 0,90;
- Asfalto com muitos buracos: 0,80;
- Paralelepípedos sem buracos: 0,85;
- Paralelepípedos com poucos buracos: 0,75;

- Paralelepípedos com muitos buracos: 0,70;
- Leito natural sem buracos: 0,65;
- Leito natural com poucos buracos: 0,60;
- Leito natural com muitos buracos: 0,55.

O nível de tráfego em um trecho de rua pode pertencer a uma das categorias a seguir, com seus respectivos pesos:

- Tráfego livre: 1,00;
- Tráfego pequeno: 0,90;
- Tráfego médio: 0,70;
- Tráfego grande: 0,50;
- Trânsito parado: 0,10.

O nível de interrupção em um trecho de rua pode pertencer a uma das categorias a seguir, com seus respectivos pesos:

- Rua livre: 1,00;
- Interrupção pequena: 0,90;
- Interrupção média: 0,70;
- Interrupção grande: 0,50;
- Rua interditada: 0,00.

Dado um grafo G = (V, E) e uma função w(e) associada a cada aresta e $\in$ E, um caminho P = (V', E'), tal que V' = $\{x_0, x_1, ..., x_k\}$ e E' = $\{e_1, e_2, ..., e_k\}$, onde os x_i são distintos e $e_1 = x_0x_1$, $e_2 = x_1x_2$, ..., e $e_k = x_{k-1}x_k$, é considerado um caminho mais rápido (ou *Fastest Path*) se w(P) tiver o menor valor para todos os possíveis caminhos P entre x_0 e x_k, desde que a função w(e) represente a seguinte equação:

$$w(e) = \frac{tamanho(e)}{velocidade(e) * interrupcoes(e) * trafego(e) * qualidade(e)}$$

O algoritmo para *Fastest Path* é baseado em A* e usa como heurística a distância Manhattan, como pode ser observado no algoritmo 3.7. Inicialmente, o vértice de origem tem seus pesos calculados e definidos (linhas 2 a 4) e é acrescentado a uma lista de nós em aberto (linha 5). Em seguida (linhas 6 a 29), o algoritmo percorre todos os vértices em aberto, daqueles de menor custo para os de maior (linha 7). Uma vez que

o vértice é percorrido, ele é acrescentado à lista de nós fechados (linha 8). Caso o vértice de destino seja encontrado, o caminho é retornado (linhas 9 e 10). Para cada vértice adjacente ao nó sendo visitado, seu pai é definido como o visitado, os custos até o momento são somados, a heurística para o destino é calculada e o vértice é adicionado à lista aberta, caso não esteja lá (linhas 14 a 27). O custo utilizado nos cálculos chama a função CustoFastest (linhas 16, 21 e 23). Ao final, o caminho é retornado (linha 30).

A função de custo utilizada pelo algoritmo está detalhada no algoritmo 3.8. Inicialmente, o algoritmo calcula as modificações na velocidade média da aresta (linha 3), multiplicando-a por fatores das interrupções, do tráfego e da qualidade. Em seguida, o tempo é calculado pela equação Tempo = Distância / Velocidade (linha 4).

```
1  FastestPath (origem, destino) {
2    origem.G = 0;
3    origem.H = Heuristica(origem, destino);
4    origem.F = origem.G + origem.H;
5    listaAberta.add(origem);
6    enquanto (listaAberta não estiver vazia) {
7      v = listaAberta.menor;
8      listaFechada.add(v);
9      se (v = destino)
10        return ExtraiCaminho(origem, destino);
11      para cada (adjacente = adjacente de v) {
12        se (adjacente estiver na listaFechada)
13          próxima iteração;
14        se (adjacente não estiver na listaAberta) {
15          adjacente.pai = v;
16          adjacente.G = v.G + CustoFastest(v, adjacente);
17          adjacente.H = Heuristica(adjacente, destino);
18          adjacente.F = adjacente.G + adjacente.H;
19          listaAberta.add(adjacente);
20        } else {
21          if (v.G + CustoFastest(v, adjacente) <
            adjacente.G) {
22            adjacente.pai = v;
```

```
23            adjacente.G = v.G + CustoFastest(v,
              adjacente);
24            adjacente.H = Heuristica(adjacente, destino);
25            adjacente.F = adjacente.G + adjacente.H;
26          }
27        }
28      }
29    }
30    return ExtraiCaminho(origem, destino);
31  }
```

Algoritmo 3.7 *Fastest Path*

```
1  CustoFastest (origem, destino) {
2    e = (origem, destino); // aresta entre eles
3    novaVelocidade = e. velocidade * e.interrupções *
     e.tráfego * e.qualidade;
4    tempo = e.distância / novaVelocidade;
5    return tempo;
6  }
```

Algoritmo 3.8 Custo do *Fastest Path*

3.1.2 Algoritmo *Coolest Path*

Um algoritmo ideal permitiria a possibilidade de combinar todos os algoritmos da seção anterior de forma que o usuário pudesse escolher que características são mais ou menos importantes para ele. A proposta do *Coolest Path* é justamente oferecer ao usuário quatro características (distância, simplicidade, capacidade turística e tempo) e três níveis de importância para cada uma delas (importância baixa, média ou alta). O usuário então poderá escolher a importância de cada uma dessas características e o caminho será calculado com base nesses níveis.

Dado um grafo G = (V, E) e uma função custo w(e) associada a cada aresta e $\in$ E, um caminho P = (V', E'), tal que V' = $\{x_0, x_1, ..., x_k\}$ e E' = $\{e_1, e_2, ..., e_k\}$, onde os x_i são distintos e $e_1 = x_0x_1$, $e_2 = x_1x_2$, ..., e $e_k = x_{k-1}x_k$, é considerado um *Coolest Path* se o custo total do caminho w(P) tiver o menor valor para todos os possíveis caminhos P entre x_0 e x_k, desde que a função w(e) represente o custo *Coolest* da aresta. A obtenção

deste caminho é explicada a seguir.

Para cada aresta *e*, é calculado o peso de acordo com cada uma das características. Os pesos são normalizados para a faixa de valores da distância, de forma a poderem ser utilizados em uma única equação, visando obter o peso da aresta de acordo com as preferências do usuário. A equação do custo de uma aresta é a seguinte:

$$w(e) = \frac{\sum_{i=1}^{n} relevância_i * custo_i}{\sum_{i=1}^{n} relevância_i}, \text{ quando } \sum_{i=1}^{n} relevância_i \neq 0 \text{ ou}$$

$$w(e) = \frac{\sum_{i=1}^{n} custo_i}{n}, \text{ quando } \sum_{i=1}^{n} relevância_i = 0$$

, onde i varia de 1 a n, sendo n é o número de características personalizáveis (no algoritmo proposto, n = 4).

Dessa forma, w(P) terá o seguinte valor:

$$w(P) = \sum_{j=1}^{k} w(e_j)$$

, onde $w(e_j)$ representa o custo da aresta j, j varia de 1 a k, sendo k o número de arestas e P representa um possível caminho entre a origem e o destino.

Seja $P(x_0x_k)$ o conjunto de todos os possíveis caminhos de x_0 a x_k. Será escolhido o caminho P que satisfizer a seguinte condição:

$$w(P) = \min(w(P_l)), \text{ para todo Pl} \in P(x_0x_k)$$

O algoritmo do *Coolest Path* é baseado em A* com distância Manhattan como heurística e seu diferencial reside na função de cálculo do custo para cada aresta. Cada uma das características citadas (distância, simplicidade, capacidade turística e tempo) pode ser avaliada de acordo com os seguintes pesos:

- Pouca importância: 0,0;
- Média importância: 0,5;
- Alta importância: 1,0.

O algoritmo *Coolest Path* pode ser observado em detalhes no algoritmo 3.9. Inicialmente, o vértice de origem tem seus pesos calculados e definidos (linhas 2 a 4) e é acrescentado a uma lista de nós em aberto (linha 5). Em seguida (linhas 6 a 29), o algoritmo percorre todos os vértices em aberto, daqueles de menor custo para os de

maior (linha 7). Uma vez que o vértice é percorrido, ele é acrescentado à lista de nós fechados (linha 8). Caso o vértice de destino seja encontrado, o caminho é retornado (linhas 9 e 10). Para cada vértice adjacente ao nó sendo visitado, seu pai é definido como o visitado, os custos até o momento são somados, a heurística para o destino é calculada e o vértice é adicionado à lista aberta, caso não esteja lá (linhas 14 a 27). O custo utilizado nos cálculos chama a função CustoCoolest (linhas 16, 21 e 23). Ao final, o caminho é retornado (linha 30).

```
1  CoolestPath (origem, destino) {
2    origem.G = 0;
3    origem.H = Heuristica(origem, destino);
4    origem.F = origem.G + origem.H;
5    listaAberta.add(origem);
6    enquanto (listaAberta não estiver vazia) {
7      v = listaAberta.menor;
8      listaFechada.add(v);
9      se (v = destino)
10        return ExtraiCaminho(origem, destino);
11      para cada (adjacente = adjacente de v)
12        se (adjacente estiver na listaFechada)
13          próxima iteração;
14        se (adjacente não estiver na listaAberta) {
15          adjacente.pai = v;
16          adjacente.G = v.G + CustoCoolest(v, adjacente);
17          adjacente.H = Heuristica(adjacente, destino);
18          adjacente.F = adjacente.G + adjacente.H;
19          listaAberta.add(adjacente);
20        } senão {
21          se (v.G + CustoCoolest(v, adjacente) <
            adjacente.G) {
22            adjacente.pai = v;
23            adjacente.G = v.G + CustoCoolest(v,
              adjacente);
24            adjacente.H = Heuristica(adjacente, destino);
```

```
25              adjacente.F = adjacente.G + adjacente.H;
26            }
27          }
28        }
29      }
30      return ExtraiCaminho(origem, destino);
31  }
```

Algoritmo 3.9 *Coolest Path*

A função de custo utilizada no *Coolest Path* é representada pelo algoritmo 3.10. O passo da linha 2 tem como finalidade encontrar a aresta entre os dois vértices em questão. Em seguida, cada um dos custos é encontrado. O custo do *shortest* é o tamanho da aresta (linha 3). O custo do *simplest* é por vértice. Contudo, para normalizá-lo com os demais, é necessário que seja um custo de uma aresta. Entre as alternativas encontradas, foi escolhido realizar a média dos custos dos vértices da aresta em questão, de forma a obter uma aproximação do custo para qualquer uma das curvas a ser escolhida. Então, o custo médio da aresta é calculado e normalizado para a escala do *shortest* (linha 4). O custo *scenic* é invertido, pois quanto mais pontos turísticos, melhor. Então, é realizado um processo de inversão e em seguida conversão para a escala do *shortest* (linha 5). O custo do *fastest* é calculado usando a função CustoFastest, anteriormente descrita, e também convertido para a escala (linha 6). Por fim, o custo *coolest* da aresta em questão é calculado utilizando uma média ponderada com base na relevância de cada uma das características, conforme visto na equação explicada anteriormente.

```
1  CustoCoolest (origem, destino) {
2    e = (origem, destino); // aresta entre eles
3    custoShortest = e.tamanho;
4    custoSimplest = converte(custoMedioArestas(e));
5    custoScenic = converte(inverte(e.custoScenic));
6    custoFastest = converte(CustoFastest(origem,
     destino));
7    custo = ((relevancia_shortest * custoShortest) +
     (relevancia_simplest * custoSimplest) +
```

```
      (relevancia_scenic * custoScenic) +
      (relevancia_fastest * custoFastest)) /
      (relevancia_shortest + relevancia_simplest +
      relevancia_scenic + relevancia_fastest);
8     return custo;
9   }
```

Algoritmo 3.10 CustoCoolest

3.2 Caminhos com Restrições em Redes de Estradas

Os caminhos com restrição propostos por Terrovitis et al. [TBP+05] limitam-se ao espaço euclidiano, ou seja, as rotas entre os pontos são realizadas em linhas retas, não se aplicando a situações em redes de estradas. Pode ser interessante uma situação, por exemplo, em que um usuário solicita uma rota da sua casa em uma cidade *A* para um hotel em uma cidade *B*, mas não quer dirigir mais que 300 km sem fazer uma pausa para descansar em algum posto de gasolina. Um caminho *a-autonomy*, com a = 300 km poderia encontrar uma rota satisfatória. Contudo, a solução proposta no artigo segue caminhos em linhas retas, não se aplicando a redes. Além disso, trata os pontos de interesse como uma única categoria. Poderia ser interessante se o usuário tivesse a opção de definir em que tipo de PoI ele deseja realizar a parada.

Foi escolhido o caminho *a-autonomy* para estender a redes de estradas. Também foi proposto um outro caminho com restrição, o *t-autonomy*, que segue o mesmo princípio do *a-autonomy*, mas utiliza o tempo ao invés do espaço. O algoritmo *a-autonomy* em redes de estradas é inovador, pois permite que o usuário encontre pontos intermediários em uma rota inter ou intra cidades, e não apenas em espaço euclidiano. Também o algoritmo *t-autonomy* é uma proposta inovadora no sentido de que permite ao usuário restringir a busca por pontos intermediários a uma escala temporal, e não apenas espacial. Os algoritmos estendidos serão apresentados em detalhes nas seções seguintes. Os tipos de ponto de interesse também podem ser escolhidos pelo usuário. Além disso, as rotas utilizadas nos algoritmos são encontradas através do *Coolest Path*, o que representa mais uma inovação frente ao artigo original, que busca apenas rotas em espaço euclidiano, utilizando para tanto uma linha reta.

O resultado é que os usuários podem personalizar adequadamente as suas rotas, por exemplo, podem solicitar uma rota parando a cada 2 km em farmácias, usando uma rota que seja a mais turística e razoavelmente simples também.

3.2.1 A-Autonomy

Voltando ao exemplo acima citado, a rota entre as cidades *A* e *B* parando a cada 300 km em postos de gasolina pode ser encontrada utilizando o algoritmo *a-autonomy* adaptado para redes de estradas e, internamente, utilizando o *Coolest path* para encontrar o caminho mais turístico.

Dado um grafo $G = (V, E)$, e um caminho $P = (G', V')$, obtido através do *Coolest path*, este caminho pode ser considerado um caminho *a-autonomy* se, dada uma constante *a*, a cada *a* unidades de distância percorridas no caminho *P* (ou um valor próximo de *a*), houver um ponto de interesse, de um tipo pré-estabelecido pelo usuário.

A implementação de *a-autonomy* é realizada utilizando dois algoritmos, um recursivo, o *RoadNetworksGHMA*, que realiza realmente a busca; e outro que realiza procedimentos de inicialização dos dados e chamada do algoritmo recursivo, o *RoadNetworksAAutonomy*.

O algoritmo *RoadNetworksAAutonomy* pode ser visto no algoritmo 3.11. Ele recebe como parâmetros a origem, o destino e a constante de autonomia *a*. O primeiro passo executado é o cálculo do *CoolestPath* dessa origem para esse destino, utilizando as preferências do usuário em relação à distância, simplicidade, turismo e rapidez (linha 2). Em seguida, o custo em distância desse caminho é obtido, através da função CustoDistancia (linha 3). Por fim, a função recursiva é chamada, passando como parâmetros a origem, o destino, *a*, o vértice sendo percorrido (origem), o caminho inicial, seu custo e o caminho final até agora , que é vazio (linha 5).

```
1  RoadNetworksAAutonomy (origem, destino, a) {
2    caminhoInicial = CoolestPath (origem, destino);
3    custoInicial = CustoDistancia (caminhoInicial, a);
4    caminhoAteAgora = {};
5    return RoadNetworksGHMA (origem, destino, origem, a,
     caminhoInicial, custoInicial, caminhoAteAgora);
6  }
```

Algoritmo 3.11 RoadNetworksAAutonomy

A função recursiva *RoadNetworksGHMA* tem sua implementação em pseudo-código exibida no algoritmo 3.12. Se o custo do caminho for menor que *a*, sinal de que não haverá paradas, e o algoritmo retorna o caminho como ele está (linhas 23 a 25). Mas

se o custo for maior que *a* (linha 2), o algoritmo seleciona o ponto *o* no caminho que tem uma distância de *p* mais próxima a *a* (linha 3). Em seguida, busca o PoI mais próximo a *o* (*pNN*), entre o conjunto de PoIs pré-estabelecido pelo usuário (linha 4). No passo seguinte, o algoritmo busca o caminho entre *p* e *pNN* e calcula o custo do caminho (linhas 5 e 6). Se o custo for maior que *a*, busca o próximo vizinho (linhas 7 a 9). Se todos os vizinhos forem examinados e a resposta não for encontrada, o algoritmo retorna *null*, como um sinal de que nada foi achado (linhas 10 a 12). Mas se encontrar, acrescenta este ponto no caminho definitivo (linha 13), calcula o novo caminho e seu custo, usando o definitivo e um *CoolestPath* (linhas 14 e 15). Em seguida, o algoritmo é chamado recursivamente, passando *pNN* como o novo p (linha 16). Se houver resultado da chamada diferente de *null*, o algoritmo retorna este resultado (linhas 17 a 19), senão vai para o próximo vizinho (linhas 19 a 21).

```
1  RoadNetworksGHMA (origem, destino, p, a, caminho, custo,
caminhoAteAgora) {
2    se (custo > a) {
3      o = ponto em caminho com distância de p igual a a
       (se não houver, retorna o caminho)
4      pNN = NN(o);
5      caminhoP = CoolestPath (p, pNN);
6      custoP = CustoDistancia (caminhoP, a);
7      se (custoP > a) {
8        próximo NN
9      }
10      se (pNN == null) {
11        return null;
12      } senão {
13        caminhoAteAgora = caminhoAteAgora +
          CoolestPath (ultimoPonto, pNN);
14        novoCaminho = caminhoAteAgora + CoolestPath (pNN,
          destino);
15        novoCusto = CustoDistancia (novoCaminho, a);
16        result = RoadNetworksGHMA (origem, destino, pNN,
          a, novoCaminho, novoCusto, caminhoAteAgora);
```

```
17          se (result != null) {
18            return result;
19          } senão {
20            próximo NN
21          }
22        }
23      } senão {
24        return caminho;
25      }
26    }
```

Algoritmo 3.12 RoadNetworksGHMA

3.2.2 T-Autonomy

Voltando ao exemplo introdutório da seção anterior, suponhamos que um determinado usuário deseje realizar uma viagem entre duas cidades, pelo caminho mais turístico. Além disso, ele deseja parar em postos de gasolina para descansar e se alimentar a cada 3hs de viagem. Essa rota pode ser encontrada utilizando o algoritmo *t-autonomy* adaptado para redes de estradas e, internamente, utilizando o *Coolest path* para encontrar o caminho mais turístico.

Dado um grafo $G = (V, E)$, e um caminho $P = (G', V')$, obtido através do *Coolest path*, este caminho pode ser considerado um caminho *t-autonomy* se, dada uma constante t, a cada t unidades de tempo percorridas no caminho P (ou um valor próximo de t), houver um ponto de interesse, de um tipo pré-estabelecido pelo usuário.

A implementação de t-*autonomy* é realizada utilizando dois algoritmos baseados nos algoritmos do *a-autonomy*, um recursivo, o *RoadNetworksGHMT*, que realiza realmente a busca; e outro que realiza procedimentos de inicialização dos dados e chamada do algoritmo recursivo, o *RoadNetworksTAutonomy*.

O algoritmo *RoadNetworksTAutonomy* pode ser visto no algoritmo 3.13. Ele recebe como parâmetros a origem, o destino e a constante de autonomia t. O primeiro passo executado é o cálculo do *CoolestPath* dessa origem para esse destino, utilizando as preferências do usuário em relação à distância, simplicidade, turismo e rapidez (linha 2). Em seguida, o custo em tempo desse caminho é obtido, através da função CustoTempo (linha 3). Por fim, a função recursiva é chamada, passando como parâmetros a origem, o destino, t, o vértice sendo percorrido (origem), o caminho

inicial, seu custo e o caminho final até agora , que é vazio (linha 5).

```
1  RoadNetworksTAutonomy (origem, destino, t) {
2    caminhoInicial = CoolestPath (origem, destino);
3    custoInicial = CustoTempo (caminhoInicial, t);
4    caminhoAteAgora = {};
5    return RoadNetworksGHMT (origem, destino, origem, t,
     caminhoInicial, custoInicial, caminhoAteAgora);
6  }
```

Algoritmo 3.13 RoadNetworksTAutonomy

A função recursiva *RoadNetworksGHMT* tem sua implementação em pseudo-código exibida no algoritmo 3.14. Se o custo do caminho for menor que *t*, sinal de que não haverá paradas, e o algoritmo retorna o caminho como ele está (linhas 23 a 25). Mas se o custo for maior que *t* (linha 2), o algoritmo seleciona o ponto *o* no caminho que tem uma distância de *p* mais próxima a *t* (linha 3). Em seguida, busca o PoI mais próximo a *o* (*pNN*), entre o conjunto de PoIs pré-estabelecido pelo usuário (linha 4). No passo seguinte, o algoritmo busca o caminho entre *p* e *pNN* e calcula o custo do caminho (linhas 5 e 6). Se o custo for maior que *a*, busca o próximo vizinho (linhas 7 a 9). Se todos os vizinhos forem examinados e a resposta não for encontrada, o algoritmo retorna *null*, como um sinal de que nada foi achado (linhas 10 a 12). Mas se encontrar, acrescenta este ponto no caminho definitivo (linha 13), calcula o novo caminho e seu custo, usando o definitivo e um *CoolestPath* (linhas 14 e 15). Em seguida, o algoritmo é chamado recursivamente, passando *pNN* como o novo p (linha 16). Se houver resultado da chamada diferente de *null*, o algoritmo retorna este resultado (linhas 17 a 19), senão vai para o próximo *nearest neighbor* (linhas 19 a 21).

```
1  RoadNetworksGHMT (origem, destino, p, t, caminho, custo,
caminhoAteAgora) {
2    se (custo > t) {
3      o = ponto em caminho com tempo de p igual a t
       (se não houver, retorna o caminho)
4      pNN = NN(o);
5      caminhoP = CoolestPath (p, pNN);
```

```
6       custoP = CustoTempo (caminhoP, t);
7       se (custoP > t) {
8         próximo NN
9       }
10       se (pNN == null) {
11         return null;
12       } senão {
13         caminhoAteAgora = caminhoAteAgora +
           CoolestPath (ultimoPonto, pNN);
14         novoCaminho = caminhoAteAgora + CoolestPath (pNN,
           destino);
15         novoCusto = CustoTempo (novoCaminho, t);
16         result = RoadNetworksGHMT (origem, destino, pNN,
           t, novoCaminho, novoCusto, caminhoAteAgora);
17         se (result != null) {
18           return result;
19         } senão {
20           próximo NN
21         }
22       }
23     } senão {
24       return caminho;
25     }
26   }
```

Algoritmo 3.14 RoadNetworksGHMT

Capítulo 4 . Serviço de Roteamento

A quase totalidade das ferramentas de roteamento disponíveis no mercado trabalha com uma arquitetura proprietária, ou seja, caso alguém deseje desenvolver um cliente para se comunicar com essas ferramentas, terá que aprender a maneira delas conversarem, quando isto for possível. Para suprir essa deficiência, é necessário o uso de uma Arquitetura Orientada a Serviços. Uma das grandes vantagens de utilizar-se esse paradigma é permitir uma maior interoperabilidade entre aplicações e uma padronização da estrutura de acesso e comunicação aos serviços oferecidos pelas mesmas. Um *Web Service*, ou serviço Web, é uma interface que descreve uma coleção de operações acessíveis via rede através de troca de mensagens padronizadas em XML [ACK+03]. Uma ferramenta que fosse um serviço Web seria independente do cliente utilizado.

Este capítulo tem como objetivo apresentar a estrutura interna do serviço de Roteamento desenvolvido para validar os algoritmos propostos no Capítulo 3. A seção 4.1 apresenta a arquitetura do serviço, justificando adequadamente as escolhas tecnológicas. A seção 4.2 apresenta o conceito de hierarquização dos dados, enquanto a pré-materialização dos resultados e a organização das visões no banco de dados são explanadas na seção 4.3. Por fim, a seção 4.4 explica em detalhes o fluxo de execução de uma atividade.

4.1 Arquitetura do Sistema

Web services são descritos usando uma notação XML formal chamada descrição do serviço. Esta descrição abrange todos os detalhes necessários para interagir com o serviço, incluindo o formato das mensagens, e os protocolos de transporte e localização. A interface esconde a implementação do serviço, permitindo que este seja usado independentemente de plataforma de *software* ou mesmo de *hardware* e ainda independente da linguagem de programação na qual tenha sido implementado. Os *Web services* são ainda simples de utilizar e inclusive de desenvolver.

A figura 4.1 define os componentes básicos da arquitetura de *Web services*. O *Service Provider* representa um servidor que oferece serviços. Esse servidor possui os

serviços e suas descrições, e publica essas descrições num catálogo, representado como *Service Registry*. O *Service Requestor*, ou seja, o cliente, busca os serviços no catálogo e após encontrar o que mais se adequa às suas necessidades, inicia uma conversação com o mesmo.

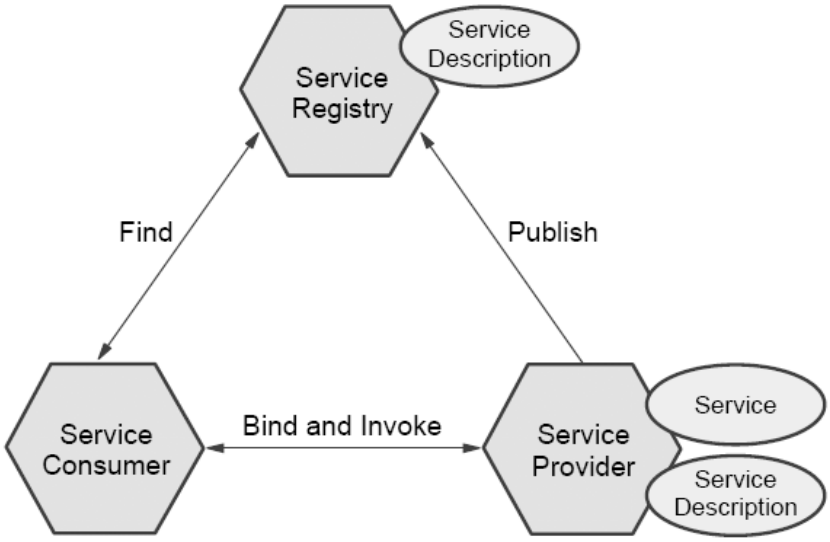

Figura 4.1 Componentes básicos de um *Web service* (fonte: [EAA+04])

O padrão mais utilizado para registro de serviços é o *Universal Discovery Description and Integration* (UDDI) [CHV+07], um padrão mantido pela organização OASIS cuja função é atuar como intermediário entre os clientes de serviços e os provedores dos mesmos. UDDI especifica uma maneira de armazenar e recuperar informações sobre serviços e especialmente o nome do provedor e as interfaces técnicas.

A comunicação entre esses atores da arquitetura de serviços é usualmente feita utilizando-se *Simple Object Access Protocol* (SOAP) [GHM+03]. SOAP é um protocolo independente de linguagem de programação, de protocolo de transporte e de rede, que permite a um cliente interagir com um provedor de serviços remoto. É um protocolo para trocar informação estruturada num meio-ambiente distribuído, descentralizado. Utiliza-se de XML para definir um *framework* de troca de mensagens extensível que permite comunicação através de vários protocolos, independente de semânticas específicas às aplicações.

O padrão mais utilizado para a descrição desses serviços é o *Web Services*

Description Language (WSDL) [CMR+07], que especifica as características operacionais de um *Web service* com notação XML. Propõe-se a responder às questões: O quê? (Sobre o quê é o serviço); Onde? (Onde é localizado o serviço); e Como? (Como utilizar o serviço).

A implementação de um *Web service* utilizando a API JAX-RPC [NSS02] da Sun está dividida em quatro etapas:

- Definição da interface, que deve conter os métodos que um cliente pode invocar. Cada método corresponde a uma operação oferecida pelo serviço;
- Implementação da interface, que corresponde à própria execução das operações do serviço. A interface e sua implementação servem para gerar automaticamente o arquivo de definição do *Web service* (WSDL) e um arquivo de mapeamento;
- *Deployment* do serviço, ou seja, o empacotamento de tudo e a instalação em um servidor de aplicações, como o *Java Sun Application Server*;
- Publicação do serviço em um catálogo, de forma que clientes possam descobrir a existência do *Web service* e aprender como invocá-lo. Essa etapa é opcional, uma vez que o provedor do serviço pode não desejar disponibilizá-lo para o público em geral, tornando-o restrito a quem for avisado do mesmo.

Para validar os algoritmos propostos, foi desenvolvido um sistema, cuja arquitetura pode ser observada na figura 4.2.

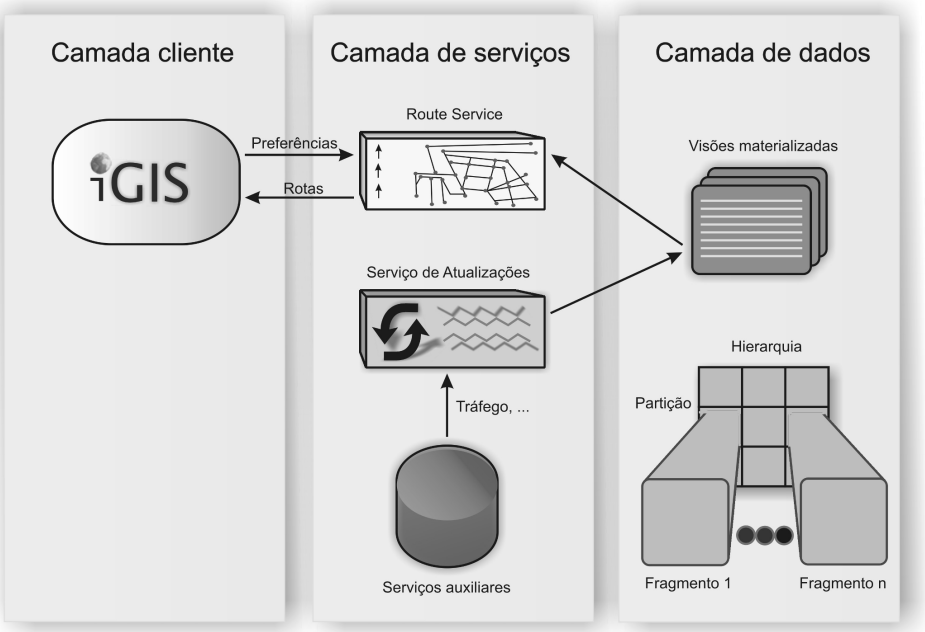

Figura 4.2 Arquitetura do sistema de busca de rotas

A arquitetura é composta de três camadas: dados, serviços e cliente. A camada de dados é formada por um SGBD PostgreSQL [Pos07a], que armazena a hierarquia de grafos na qual estão organizados os dados, e as visões materializadas com rotas pré-calculadas. A camada de serviços possui o serviço de Roteamento, que realiza a materialização das rotas calculadas com o *Coolest path* e retorna para o cliente os caminhos desejados, sejam eles sem restrições, *a-autonomy* ou *t-autonomy*; e o serviço de atualização das rotas, que entra em ação quando interrupções são cadastradas e faz-se necessário atualizar as rotas que passem pela rua na qual a interrupção foi inserida. Por fim, a camada cliente possui um software cliente que acessa o serviço, permitindo que o usuário entre com os parâmetros de personalização. Na aplicação de validação dos algoritmos propostos, foi utilizado o *framework* de *Web mapping* iGIS [BLS+04] como cliente do serviço.

Os algoritmos de busca de caminho necessitam de um alto tempo de processamento, de forma que para que o serviço seja eficiente, as rotas serão pré-computadas *offline* e apenas "montadas" *online*, conforme as idéias de materialização propostas por Jing et al. [JHR98]. Os dados das ruas são armazenados em uma hierarquia de grafos, de forma a reduzir o espaço de armazenamento das visões materializadas. A seção 4.2 apresenta detalhes sobre a hierarquização do banco de dados, enquanto a 4.3 explica como é realizada a pré-materialização.

Os dados são armazenados no PostgreSQL conforme uma adaptação da modelagem sugerida por Speicys et al. [SJK03]. Ou seja, há os dados em espaço bidimensional e os dados de conectividade em forma de grafos também.

O *Web service* de rotas foi desenvolvido utilizando a tecnologia JAX-RPC, de Java *Enterprise Edition* [NSS02]. A figura 4.3 apresenta um diagrama UML [Fow03] de classes do serviço.

A interface *RouteServiceIF*, implementada pela classe *RouteService*, define o *Web service* e as seguintes operações:

- *GetLinksRota*: recebe uma coordenada de origem e uma de destino, e os parâmetros de preferência de cada um dos itens (*shortest*, *simplest*, *scenic* e *fastest*), e retorna para o usuário uma coleção de arestas, ou seja, o *Coolest path* a ser exibido no mapa, de acordo com as preferências encaminhadas;

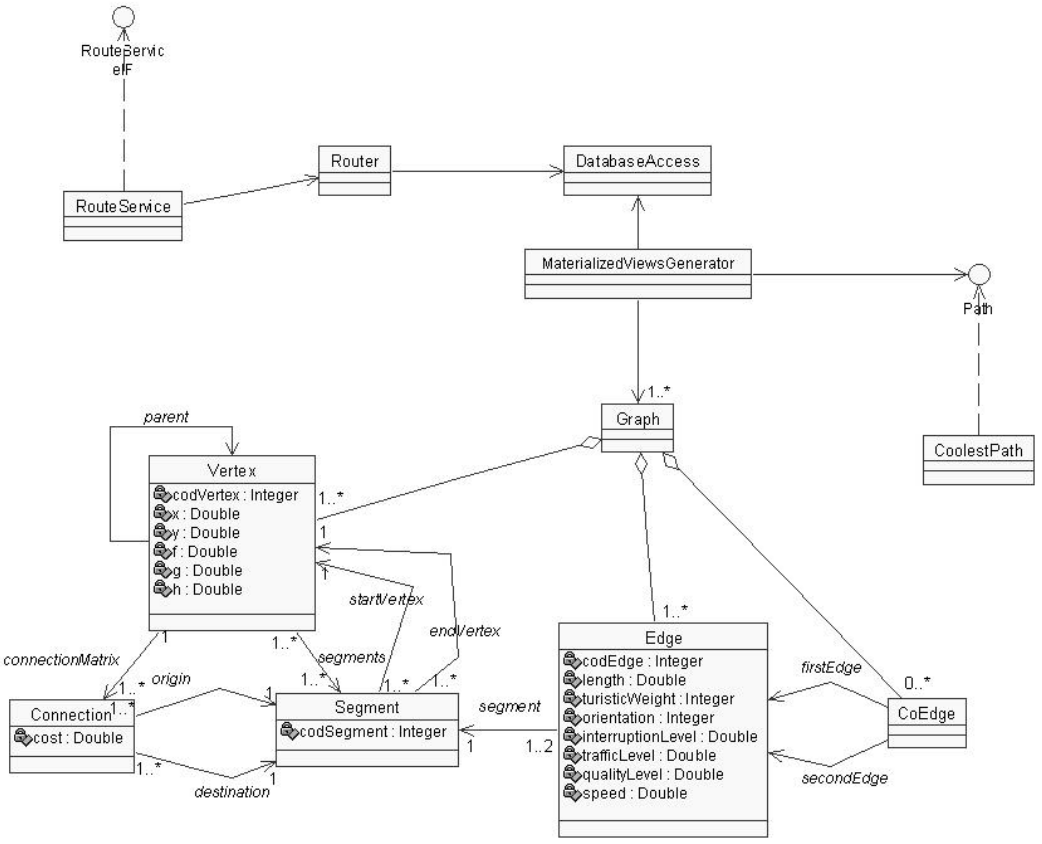

Figura 4.3 Diagrama de classes do serviço de rotas

- *GetLinksRotaAAutonomy*: recebe as coordenadas de origem e destino, os parâmetros de preferência, o tipo de ponto de interesse no qual o usuário deseja realizar as paradas, e a constante de autonomia *a*, e retorna para o usuário um objeto composto *RouteReturn*, que possui a coleção da arestas que representam o caminho a ser exibido no mapa e também a coleção de PoIs nos quais serão feitas paradas;
- *GetLinksRotaTAutonomy*: recebe as coordenadas de origem e destino, os parâmetros de preferência, o tipo de ponto de interesse no qual o usuário deseja realizar as paradas, e a constante *t*, e retorna para o usuário um objeto composto *RouteReturn*, que possui a coleção das arestas que representam o caminho a ser exibido no mapa e também a coleção de PoIs nos quais serão feitas paradas.

O serviço de rotas encaminha as requisições para a classe *Router*, que é o núcleo do sistema e realiza as operações de busca, utilizando a classe *DatabaseAccess*, que possui uma conexão com o banco de dados e centraliza o acesso aos dados nele armazenados.

A geração das rotas é de responsabilidade da classe *MaterializedViewsGenerator*, que recebe como parâmetros os dados da hierarquia, utiliza a classe *DatabaseAccess* para carregar cada um dos grafos dos diferentes níveis hierárquicos e gerar as visões materializadas, que são inseridas no banco de dados. Para calcular os caminhos, utiliza um objeto da interface *Path*, que pode representar qualquer tipo de caminho. Neste caso, utiliza-se a implementação *CoolestPath*.

A classe *Graph* representa um grafo do banco de dados, e possui uma coleção de vértices (*Vertex*), uma de arestas (*Edge*) e uma de co-arestas (*CoEdge*). É importante lembrar que uma co-aresta representa um par de arestas inversas, representando o mesmo segmento de rua. A modelagem dessas classes segue a idéia proposta por Speicys et al. [SJK03].

O serviço de rotas foi acoplado como mais uma funcionalidade disponível no *framework* de *web mapping* iGIS, bastando para isso o acréscimo de um botão na interface, que permite abrir uma janela na qual o usuário vai definir as suas preferências. Em seguida, o *framework* usa um cliente estático do *Web service* e chama a operação desejada.

Há algumas informações sobre as ruas das cidades que são dinâmicas. Como exemplos, é possível citar o tráfego e as interrupções nas ruas, seja devido a feiras ou eventos, acidentes ou obras. O serviço de atualização é responsável por atualizar as visões materializadas afetadas por esses dados, ou seja, aquelas tuplas nas quais o parâmetro *fastest* é significante.

O serviço inicialmente iria monitorar um outro serviço, de obtenção de dados de tráfego e recalcular as visões sempre que mudanças ocorressem. Contudo, o tempo de processamento aumentaria consideravelmente. Então, apenas as ruas principais das cidades são observadas. As ruas principais são encontradas utilizando as medidas propostas por Jiang e Claramunt [JC04], e o serviço recalcula apenas as rotas que passarem por essas ruas, e quando a situação do tráfego nelas mudar.

Foi desenvolvida uma aplicação simples para cadastro de interrupções. O usuário responsável por fazer esse cadastro escolhe a aresta e descreve a interrupção, bem como seu nível, como pode ser observado na figura 4.4. Em seguida, o serviço de atualização seria acionado para recalcular os caminhos que passarem pelo trecho em questão. Mas essa atualização foi deixada para trabalhos futuros.

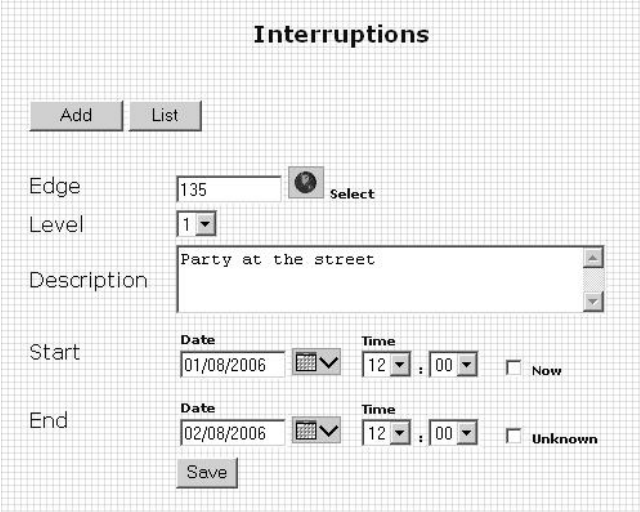

Figura 4.4 Cadastro de interrupções

4.2 Hierarquia dos Dados

Para pré-materializar todas as rotas de todas as possíveis origens para todos os possíveis destinos, seria necessário um espaço de armazenamento imenso. Uma possível solução para este problema é organizar o grafo das cidades em uma hierarquia, de forma que para cada pedaço de cada nível da hierarquia, sejam pré-calculadas as possíveis rotas, e em seguida seja necessário apenas montar a rota total, caso esta necessite atravessar os níveis e pedaços da hierarquia.

Seguindo a proposta de hierarquização apresentada por Jing et al. [JHR98], o grafo completo das cidades foi particionado em um conjunto de fragmentos. Para haver uma divisão lógica, foi determinado que cada bairro seria um fragmento. Os nós que fazem parte de mais de um fragmento são chamados de nós-borda e são colocados num nível acima da hierarquia. Ainda assim, o grafo desse nível pode ficar grande, dependendo do número de cidades e do tamanho destas. Então o procedimento é repetido. O grafo é novamente dividido em fragmentos, desta vez um para cada cidade, e os nós-borda são considerados em um nível hierárquico acima. A figura 4.5 apresenta sumariamente como se organiza a hierarquia, de forma semelhante à figura 2.11.

As cidades estão marcadas de *A* a *E*, e os bairros de 1 a 25 (estes visíveis apenas no nível 0). O nível 0 contém todos os nós, enquanto o nível 1 contém os nós de ligação entre os bairros e o nível 2 os nós de ligação entre as cidades.

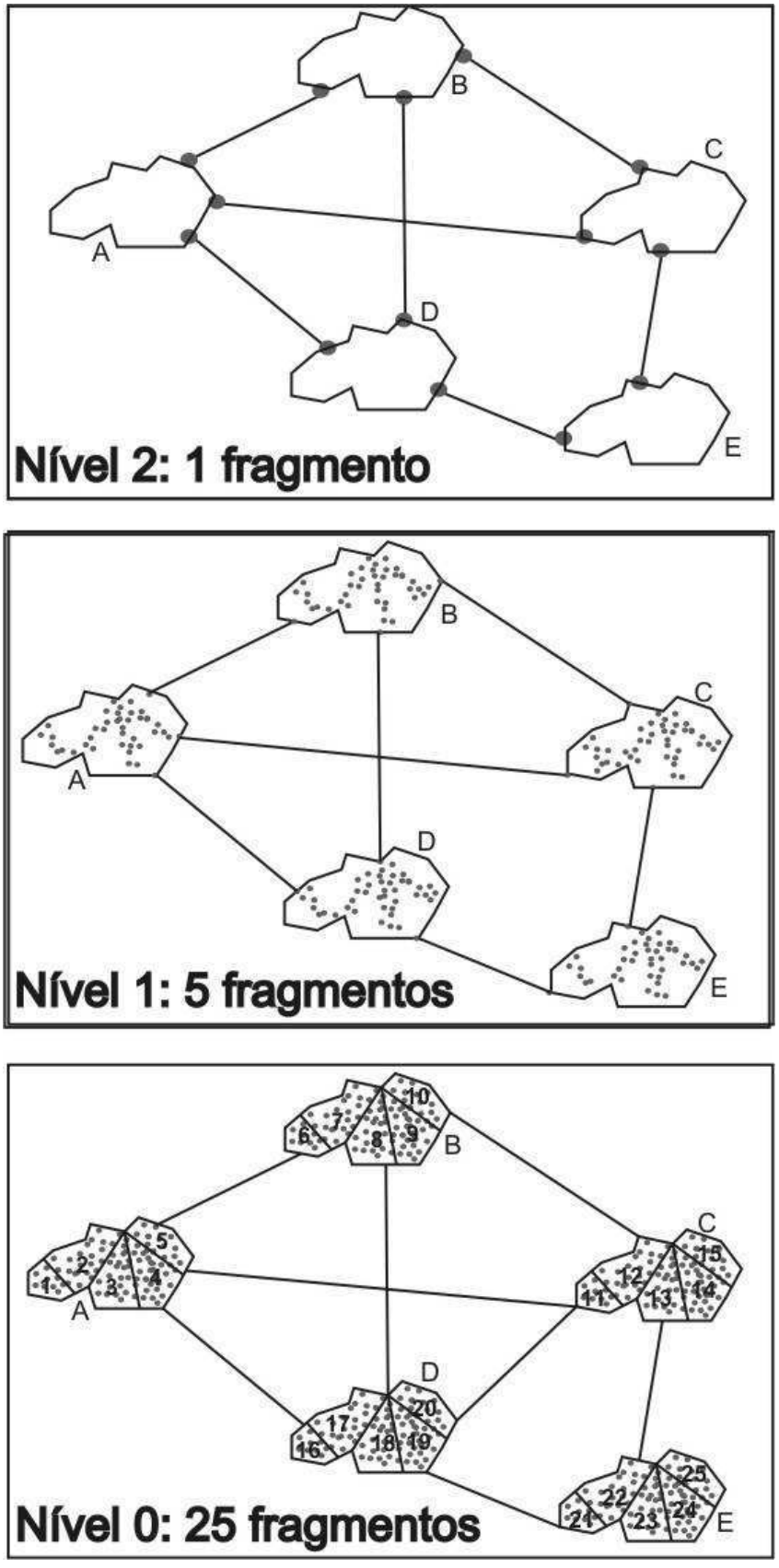

Figura 4.5 Organização hierárquica dos dados

Caso haja necessidade de acrescentar mais níveis hierárquicos, é só repetir o procedimento, ou seja, novamente dividir em fragmentos e ir acrescentando níveis até que o grafo seja pequeno o suficiente.

4.3 Pré-Materialização das Rotas

Cada um dos fragmentos de cada um dos níveis da hierarquia é considerado individualmente um grafo. Cada um desses grafos é carregado em memória pelo gerador de visões materializadas, e todos os caminhos possíveis de todas as origens para todos os destinos são gerados e armazenados no banco de dados.

Os caminhos variam de acordo com o nível de cada um dos atributos *shortest*, *simplest*, *scenic* e *fastest*. Então, o banco de dados tem três tabelas de visões materializadas, uma para cada nível da hierarquia. Caso novos níveis da hierarquia sejam necessários, basta acrescentar novas tabelas, uma para cada nível, e adaptar o algoritmo de obtenção das rotas para que ele conheça os novos níveis. As visões existentes são:

- *Nivel0View*: armazena os caminhos do nível inferior da hierarquia, representados pelos bairros. Possui os seguintes atributos: *numfragment* (o número identificador do fragmento no qual essa rota se encontra); *shortest*, *simplest*, *scenic* e *fastest* (o nível de cada um desses atributos, utilizados no cálculo do caminho); *codorigin* (o vértice de origem do caminho); *coddestination* (o vértice de destino do caminho); *cost* (o custo total do caminho); e *path* (o caminho, formado por uma lista de vértices);
- *Nivel1View*: armazena os caminhos do nível intermediário da hierarquia, representados pelas cidades. Possui os seguintes atributos: *numfragment* (o número identificador do fragmento/cidade no qual essa rota se encontra); *shortest*, *simplest*, *scenic* e *fastest* (o nível de cada um desses atributos, utilizados no cálculo do caminho); *codorigin* (o vértice de origem do caminho); *coddestination* (o vértice de destino do caminho); *cost* (o custo total do caminho); e *path* (o caminho, formado por uma lista de vértices);
- *Nivel2View*: armazena os caminhos do nível superior da hierarquia, representados pelos nós-borda entre as cidades. Como nível superior, não é particionado, sendo apenas um grafo. Possui os seguintes atributos: *shortest*, *simplest*, *scenic* e *fastest* (o nível de cada um desses atributos, utilizados no cálculo do caminho); *codorigin* (o vértice de origem do caminho);

coddestination (o vértice de destino do caminho); *cost* (o custo total do caminho); e *path* (o caminho, formado por uma lista de vértices).

4.4 Fluxo de Execução

Para que a execução do serviço seja bem compreendida, suponhamos que um usuário realize uma consulta a-*autonomy*. Abaixo segue uma lista de passos realizados para obter o resultado:

1. Usuário acessa o iGIS, seleciona um ponto de origem e um de destino e faz uma consulta a-*autonomy*;
2. *RouteService* recebe a requisição, organiza os parâmetros e encaminha para o algoritmo *RoadNetworksAAutonomy*;
3. Algoritmo chama método para obter o *CoolestPath* entre a origem e o destino;
4. O algoritmo de obtenção do *CoolestPath* acessa as visões materializadas e monta o caminho, navegando nos vários níveis da hierarquia, e de acordo com os parâmetros *shortest*, *simplest*, *scenic* e *fastest* passados pelo usuário;
5. O algoritmo *RoadNetworksGHMAAutonomy* é chamado para encontrar pontos de interesse intermediários no caminho descoberto. Para tal, utiliza uma função de *Nearest Neighbor*, encontra pontos, calcula o custo de ir até eles, e caso seja menor que *a*, o ponto é escolhido. O algoritmo é chamado recursivamente até que a distância do último ponto ao destino seja inferior a *a*. Caso não seja possível encontrar pontos intermediários do tipo escolhido pelo usuário, o algoritmo retorna uma mensagem avisando que não foi possível encontrar uma rota;
6. Em seguida, uma função auxiliar é chamada para converter o caminho, que originalmente é uma coleção de vértices, para uma coleção de arestas, coleção esta que é retornada para o cliente;
7. O iGIS então exibe no mapa o caminho retornado.

A figura 4.6 apresenta um diagrama de seqüência para representar os passos acima enunciados. A numeração do diagrama segue a realizada na lista de passos.

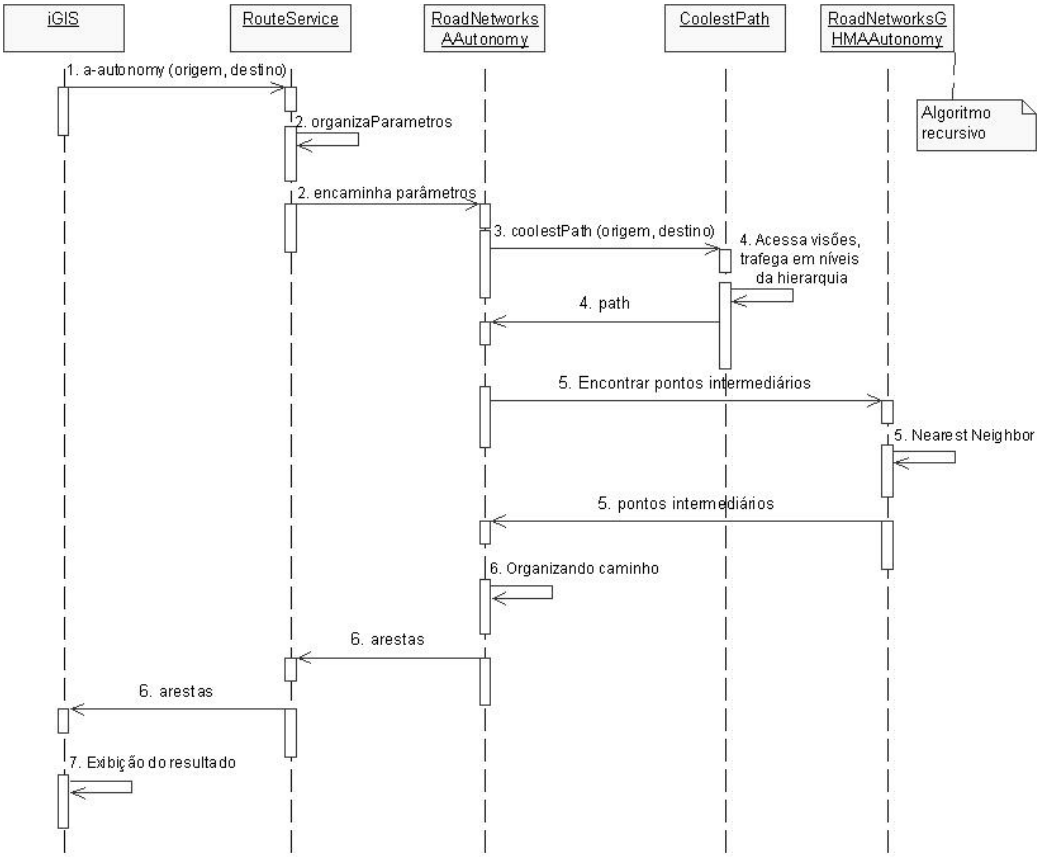

Figura 4.6 Diagrama de seqüência de uma operação *a-autonomy*

Capítulo 5 . Estudo de Caso

Este capítulo tem como objetivo apresentar um estudo de caso realizado com a finalidade de testar o serviço desenvolvido para validar os algoritmos propostos. Foram obtidos um conjunto de dados de ruas e rodovias e várias buscas de rotas foram realizadas, variando os parâmetros de preferência de as possibilidades de restrição, de forma a poder avaliar a qualidade dos algoritmos.

A organização dos dados utilizados nos exemplos será explicada na seção 5.1. As consultas realizadas e a análise dos resultados podem ser observadas na seção 5.2 e uma avaliação do desempenho obtido pelos algoritmos é descrita na seção 5.3.

5.1 Organização dos Dados

Para realizar os testes, foram obtidos dados de ruas da cidade de Campina Grande, Paraíba. A estes dados foi acrescentado um conjunto de medidas, como por exemplo os custos das curvas. Os grafos foram montados para alguns bairros da cidade e então esses dados foram replicados para gerar outras cidades, que foram interconectadas através de rodovias. A organização do banco de dados foi feita utilizando uma adaptação das técnicas de modelagem propostas por Speicys et al [SJK03], conforme detalhado na seção 2.3.3. O esquema do banco de dados pode ser visualizado na figura 5.1.

As interseções entre as ruas são vértices do banco de dados e são cadastradas na tabela *Vertex*, com um código e as coordenadas geográficas. O PostgreSQL possui uma extensão para suporte espacial chamada Postgis [Pos07b], padronizada de acordo com o Open Geospatial Consortium [OGC07], e permite que consultas espaciais sejam realizadas sobre os objetos geográficos. As coordenadas geográficas e outros dados relacionados aos trechos de ruas são armazenados na tabela *Segment*, onde cada tupla tem um vértice de origem e um de destino. Cada tupla em *Segment* tem uma ou duas arestas (tuplas da tabela *Edge*), podendo representar uma rua uni ou bidirecional. Uma tupla da tabela *Vertex* tem ainda uma matriz de conexão (*ConnectionMatrixVertex*), que diz quais segmentos de origem e destino passam por esse vértice, e o custo dessa transição. Quando o segmento é bidirecional, as duas arestas têm um relacionamento

com uma mesma tupla da tabela *CoEdge*.

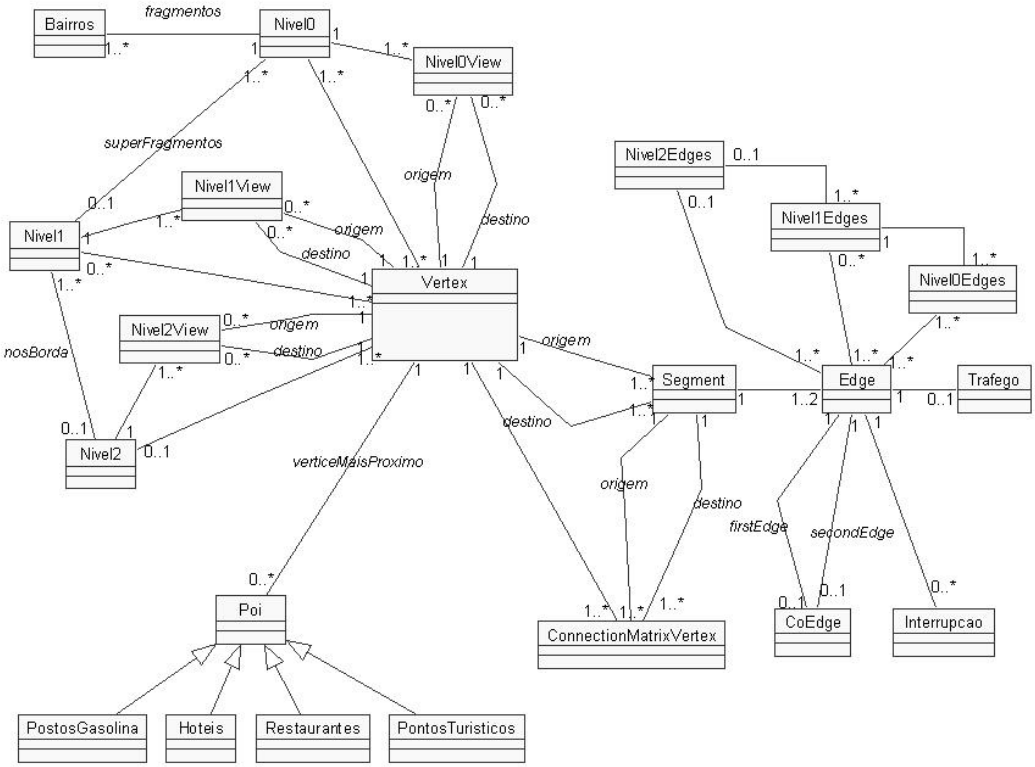

Figura 5.1 Esquema lógico do banco de dados

A tabela *Edge* tem um relacionamento com a tabela *Trafego*, que pode ou não ter informações sobre o nível de tráfego de cada aresta. A tabela *Interrupcao* também, pode ou não ter uma ou muitas interrupções em cada trecho de rua.

A tabela *Poi* possui informações sobre os pontos de interesse cadastrados. Essa tabela possui um relacionamento com o vértice mais próximo do PoI. Os caminhos são entre vértices, então é necessário saber de qual vértice pode-se alcançar cada PoI. Os quatro tipos de pontos de interesse cadastrados herdam de *Poi*: *PostosGasolina*, *Hoteis*, *Restaurantes* e *PontosTuristicos*. Caso seja desejado ter outro tipo de ponto de interesse, é só acrescentar mais uma tabela herdando de *Poi*.

Cada bairro (tabela *Bairros*) representa um fragmento da partição (tabela *Nivel0*). Cada fragmento tem um conjunto de vértices e um vértice pode fazer parte de mais de um fragmento (quando se localiza na interseção entre dois bairros, por exemplo). Os vértices que estão em mais de um fragmento fazem parte da tabela *Nivel1*, que também é particionada em alguns fragmentos e, novamente, os vértices-borda estão na tabela *Nivel2*, tabela esta que não é fragmentada. Da mesma forma que os vértices, as

arestas fazem parte de uma partição (*Nivel0Edges*), as que estão em mais de um fragmento são arestas-borda (*Nivel1Edges*) e subseqüentemente, *Nivel2Edges*.

Por fim, os caminhos pré-calculados são armazenados nas visões: *Nivel0View* para o nível mais baixo da hierarquia; *Nivel1View* para o nível intermediário; e *Nivel2View* para o nível superior. Todas possuem relacionamento com um vértice de origem e um de destino.

5.2 Exemplos de Rotas

Para realizar as consultas de busca de rotas no *framework* iGIS, o usuário deve selecionar a origem e o destino desejados e, em seguida, definir seus parâmetros, como pode ser observado na figura 5.2.

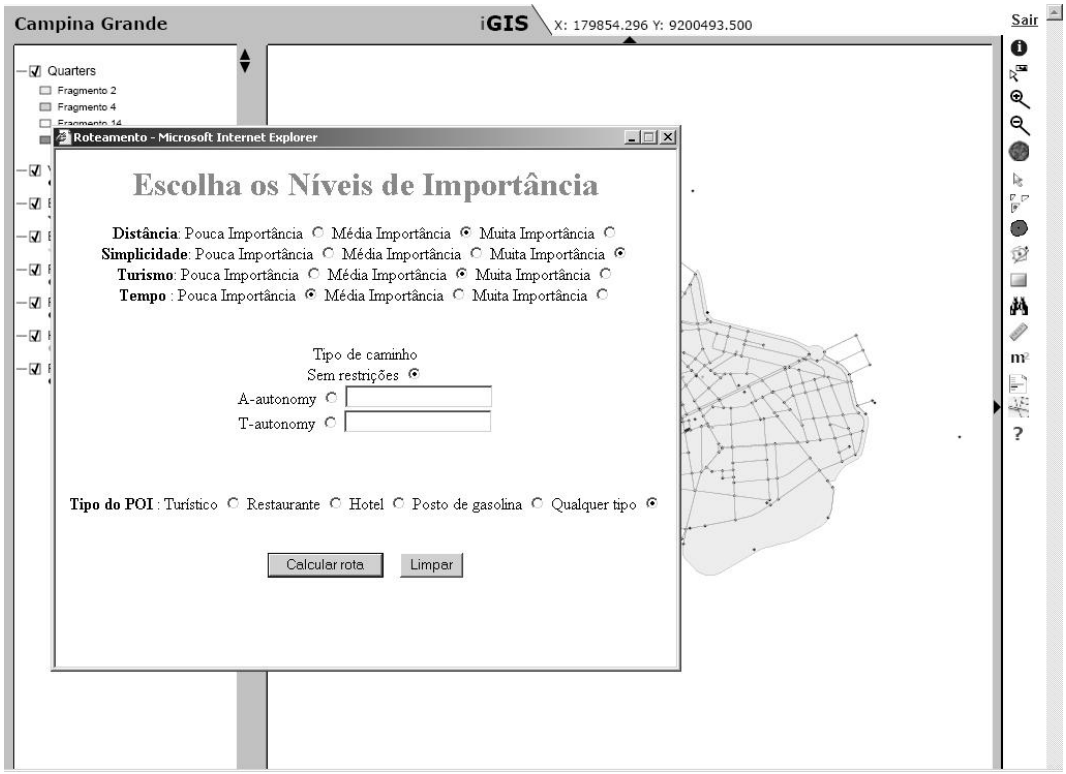

Figura 5.2 Tela de busca de rotas no iGIS

Para comparar os resultados com diferentes níveis atribuídos pelos usuários às variáveis *shortest*, *simplest*, *scenic* e *fastest* de um *Coolest path* sem restrições, uma mesma rota foi solicitada alterando estes parâmetros. A figura 5.3 apresenta uma rota do ponto *A* ao ponto *B*, com o parâmetro *shortest* definido como de Muita Importância e os demais como de Pouca Importância.

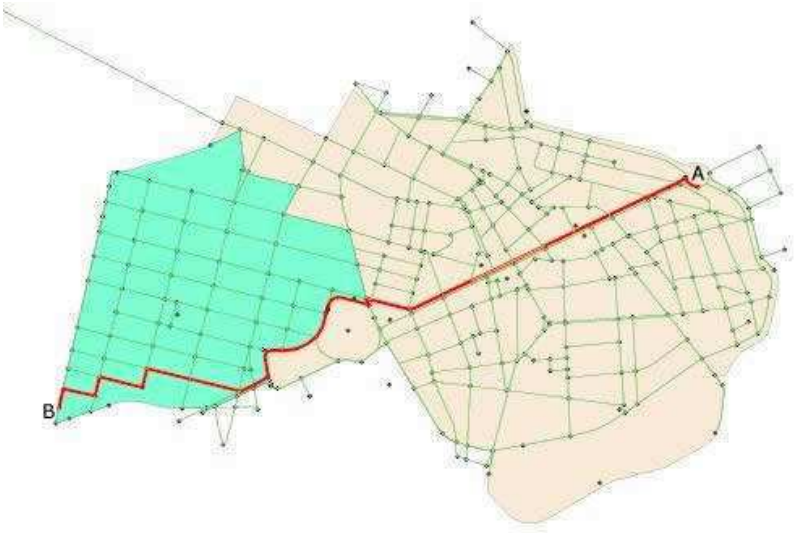

Figura 5.3 Rota de *A* a *B* maximizando o parâmetro *shortest*

A figura 5.4 apresenta uma rota do ponto *A* ao ponto *B*, com o parâmetro *simplest* definido como de Muita Importância e os demais como de Pouca Importância. No exemplo com o *shortest*, o caminho foi bastante direto. Já no exemplo do *simplest*, nota-se que são feitas poucas curvas. No ponto marcado com o número 1, o algoritmo decidiu realizar a curva, pois a função de custo de cada aresta caracteriza-se pela soma do custo de simplicidade com a heurística até o destino. E nesse caso, a aresta escolhida tinha um valor heurístico bem menor, o que deixou a soma total inferior à aresta que seguia em frente. Mas o resultado não foi gravemente afetado por esta escolha, uma vez que o caminho permaneceu simples.

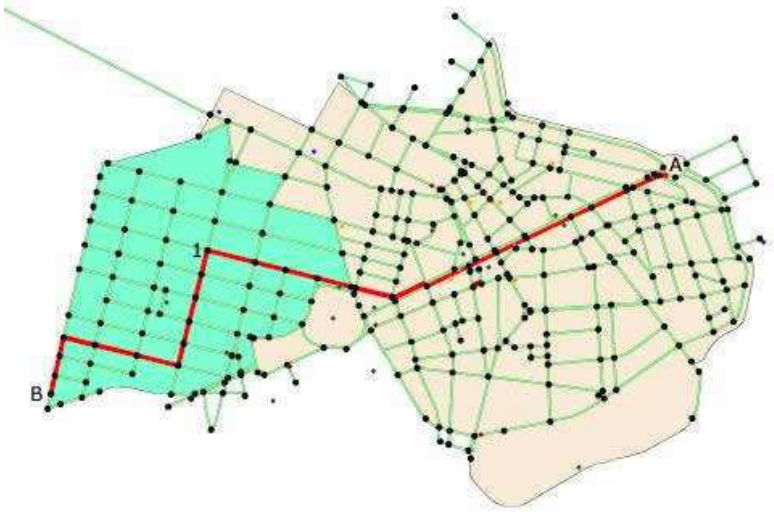

Figura 5.4 Rota de *A* a *B* maximizando o parâmetro *simplest*

A figura 5.5 apresenta uma rota do ponto *A* ao ponto *B*, com o parâmetro *scenic* definido como de Muita Importância e os demais como de Pouca Importância. Os pontos numerados são turísticos, então o caminho percorrido passa em sete deles. O caminho é semelhante ao encontrado na figura 5.3, mas pode-se perceber que agora o caminho passa no ponto turístico 5, enquanto na figura 5.3 passava por trás dele.

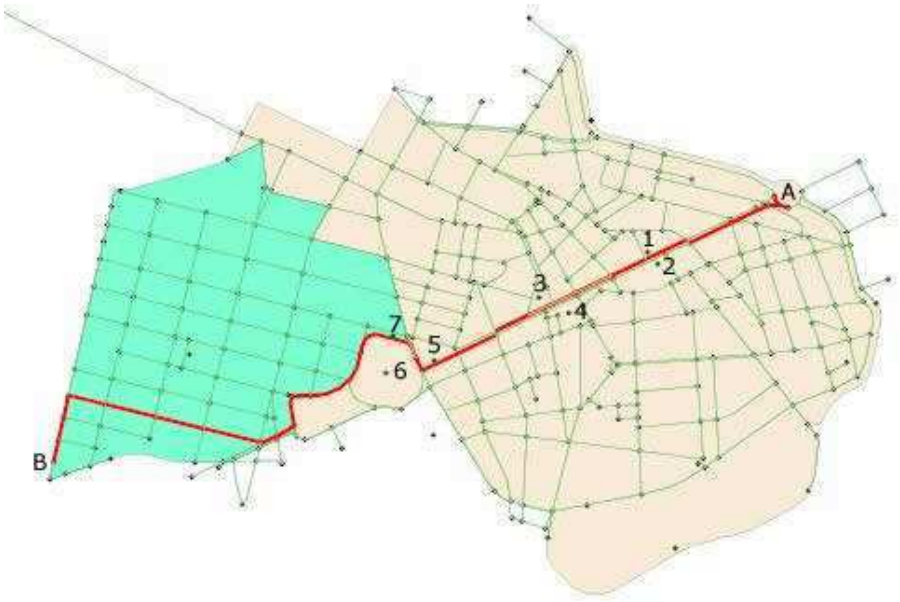

Figura 5.5 Rota de *A* a *B* maximizando o parâmetro *scenic*

A figura 5.6 apresenta uma rota do ponto *A* ao ponto *B*, com o parâmetro *fastest* definido como de Muita Importância e os demais como de Pouca Importância. O trecho assinalado no mapa com o número 1 é o que diferencia o caminho gerado na figura 5.6 do caminho da figura 5.3. Na figura 5.3 o caminho seguiu direto, por uma única aresta, pois o custo da distância era menor do que seguindo por três arestas, como a figura 5.6. Mas o custo do tempo de viagem é menor seguindo pelas três arestas, porque a qualidade da pavimentação é maior, já que o trecho escolhido pelo caminho mais curto tem pavimentação em leito natural. Quando o custo total é calculado, vale mais a pena seguir pelo caminho um pouco mais longo, mas sem buracos.

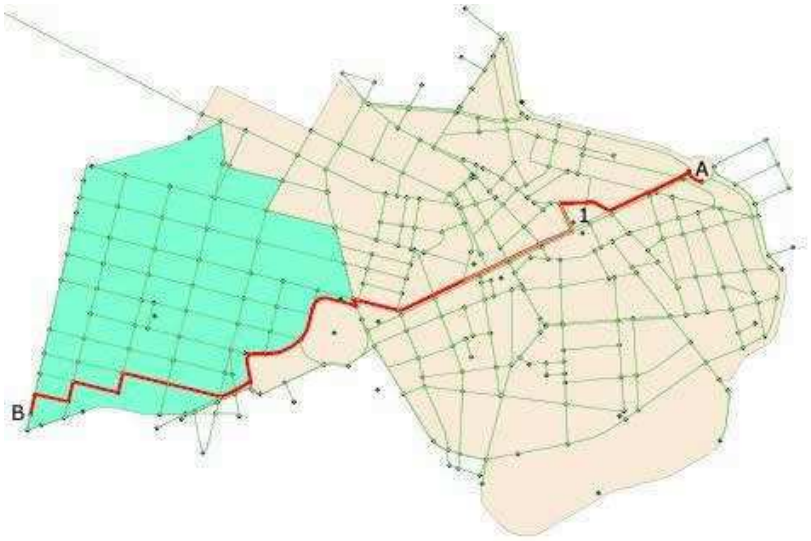

Figura 5.6 Rota de *A* a *B* maximizando o parâmetro *fastest*

Um exemplo de caminho entre duas cidades pode ser visualizado nas figuras 5.7 e 5.8. Os parâmetros foram: *shortest* – Muita Importância; *simplest* – Média Importância; *scenic* – Pouca Importância; *fastest* – Média Importância. A média ponderada dos valores de cada característica, de acordo com os parâmetros definidos pelo usuário, foi menor pelo caminho selecionado, devido a fatores como custo das curvas, qualidade da pavimentação, tráfego e interrupções, além da distância e da velocidade média em cada trecho.

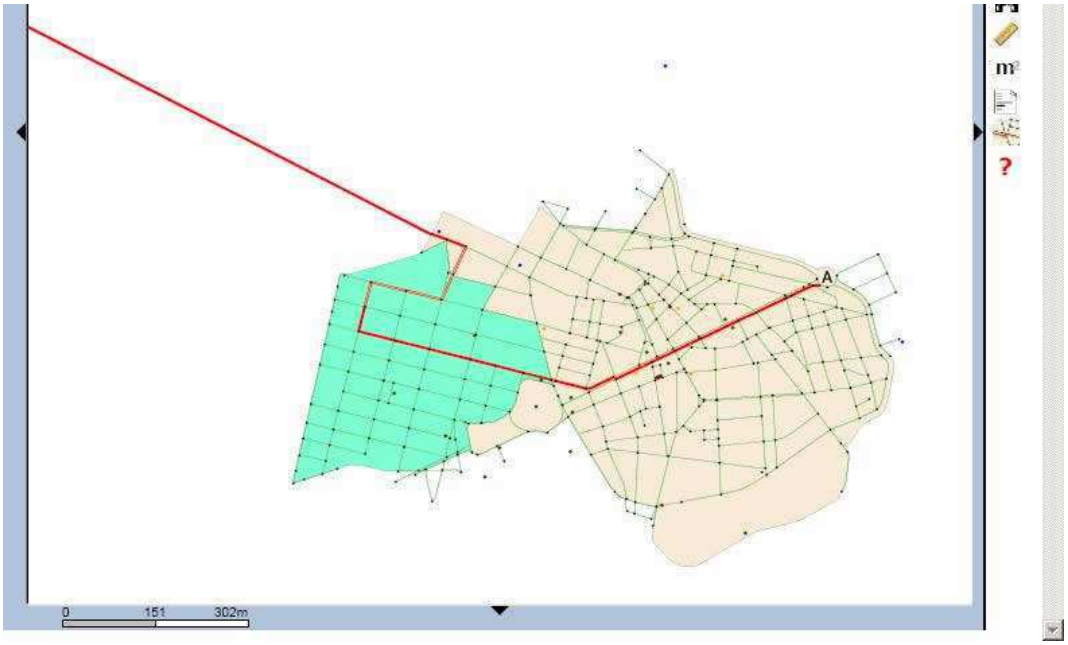

Figura 5.7 Primeira parte de um *Coolest path* entre duas cidades, partindo de *A*, com *shortest* Muita Importância, *simplest* Média, *scenic* Pouca e *fastest* Média

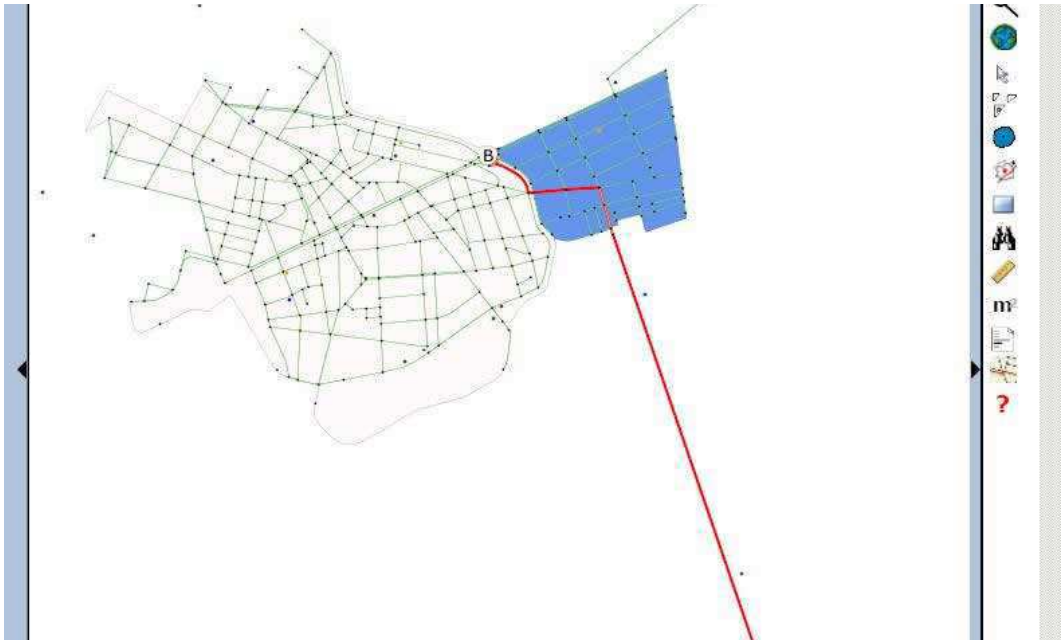

Figura 5.8 Segunda parte de um *Coolest path* entre duas cidades, chegando a *B*, com *shortest* Muita Importância, *simplest* Média, *scenic* Pouca e *fastest* Média

Outro exemplo de caminho entre duas cidades, entre pontos diferentes do exemplo anterior, pode ser visualizado nas figuras 5.9 e 5.10. Os parâmetros foram: *shortest* – Pouca Importância; *simplest* – Muita Importância; *scenic* – Média Importância; *fastest* – Pouca Importância. Da mesma forma que na rota anterior, a média ponderada entre os fatores obtida para esta rota foi a menor.

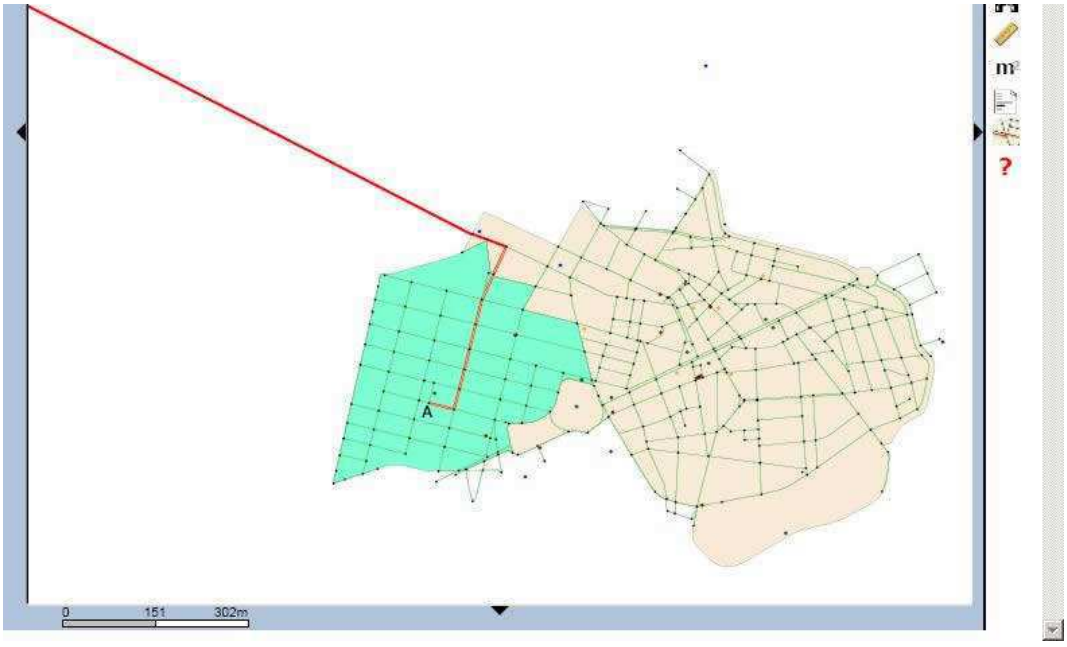

Figura 5.9 Primeira parte de um *Coolest path* entre duas cidades, partindo de *A*, com *shortest* Pouca Importância, *simplest* Muita, *scenic* Média e *fastest* Pouca

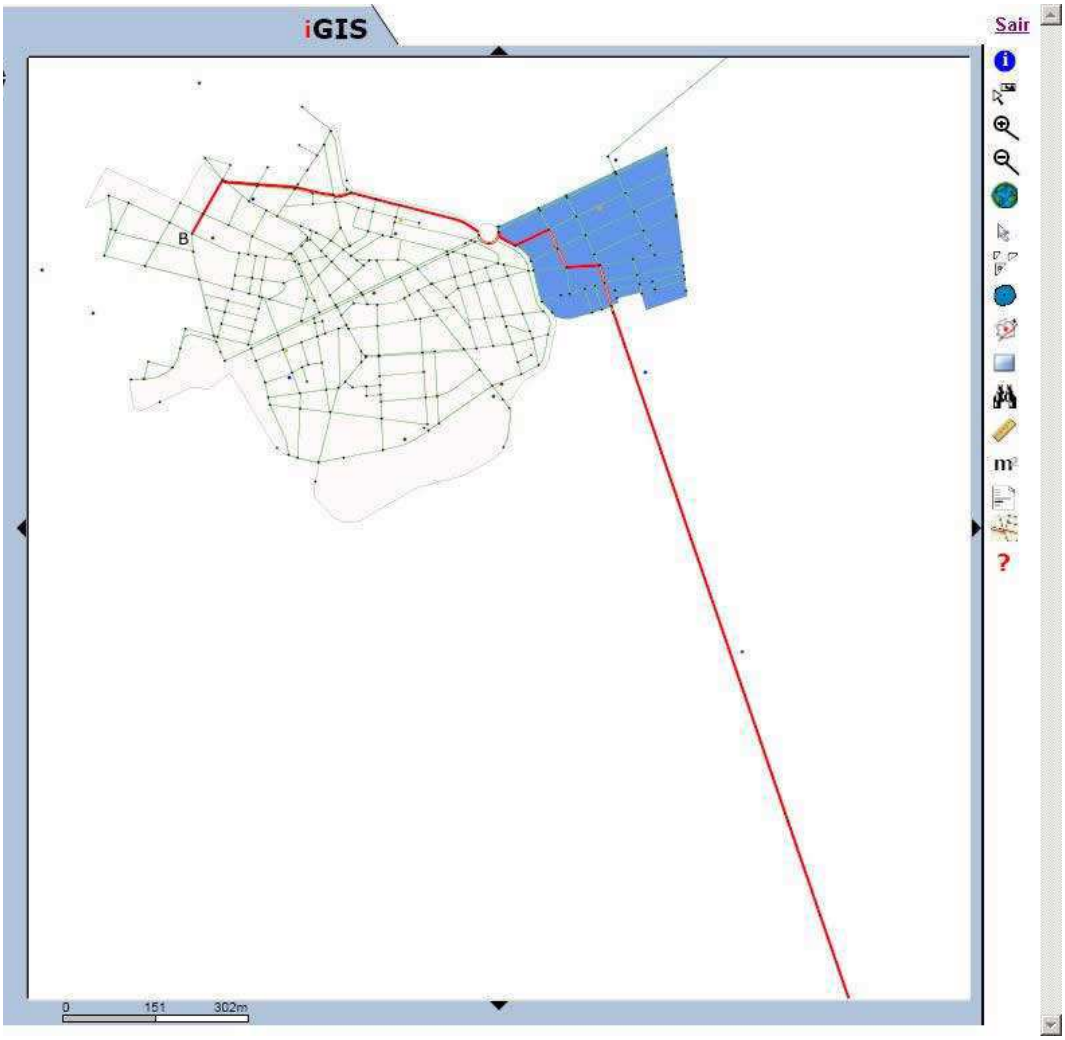

Figura 5.10 Segunda parte de um *Coolest path* entre duas cidades, chegando a *B*, com *shortest* Pouca Importância, *simplest* Muita, *scenic* Média e *fastest* Pouca

Após verificar alguns exemplos de *Coolest path* sem restrições, vamos analisar alguns com restrições. A figura 5.11 apresenta uma consulta *a-autonomy*, com a constante *a* = 1000 metros, *shortest* com Muita Importância, *simplest* com Pouca Importância, *scenic* com Pouca Importância, *fastest* com Média Importância, e permitindo que as paradas sejam em qualquer tipo de PoI. O caminho parte de *A* a *E*, e pára em três PoIs, *B*, *C*, e *D*. Foi calculado o *Coolest path* de acordo com os parâmetros definidos pelo usuário e em seguida este caminho foi sendo modificado para abarcar os PoIs necessários.

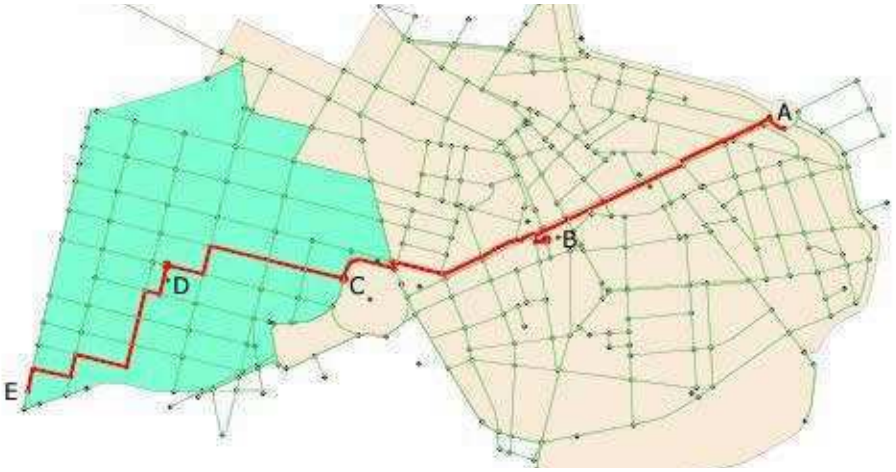

Figura 5.11 Caminho *a-autonomy* com *a* = 1000 e tipo de PoI = qualquer

A figura 5.12 apresenta uma consulta *a-autonomy* semelhante à anterior, mas com a constante *a* = 1500 metros, *shortest* com Pouca Importância, *simplest* com Média Importância, *scenic* com Média Importância, *fastest* com Pouca Importância, e definindo que as paradas sejam em restaurantes. O caminho parte de *A* a *D*, e pára em dois PoIs, *B* e *C*. Foi calculado o *Coolest path* de acordo com os parâmetros definidos pelo usuário e em seguida este caminho foi sendo modificado para abarcar os PoIs necessários. O caminho da figura 5.12 difere um pouco do exibido na figura 5.11, por exemplo no trecho destacado pelo número 1, no qual o caminho passa pela frente de um ponto turístico, já que neste caminho o parâmetro *scenic* foi definido como de Média Importância, enquanto no anterior tinha sido definido como de Pouca Importância.

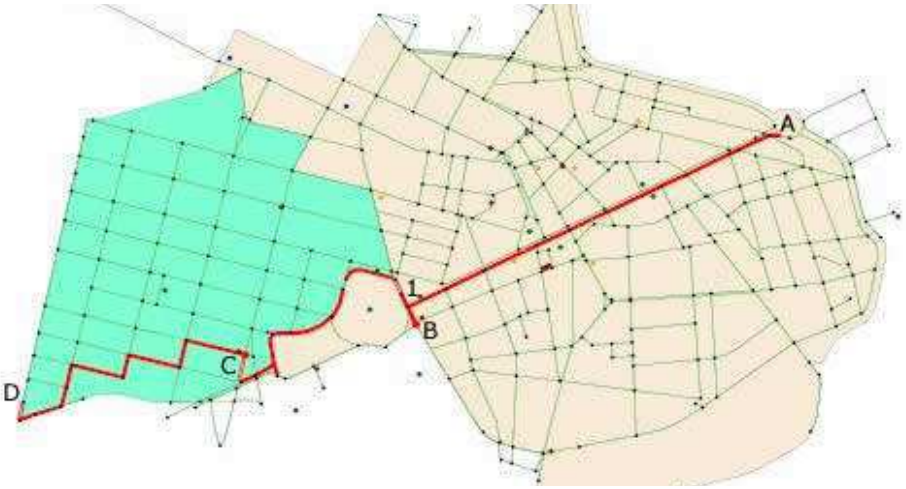

Figura 5.12 Caminho *a-autonomy* com *a* = 1500 e tipo de PoI = restaurante

A figura 5.13 apresenta uma consulta *t-autonomy*, com a constante t = 100 segundos, *shortest* com Pouca Importância, *simplest* com Pouca Importância, *scenic* com Muita Importância, *fastest* com Pouca Importância, e restringindo as paradas a pontos turísticos. O caminho parte de *A* a *E*, e pára em três PoIs, *B*, *C*, e *D*. Além das paradas em pontos turísticos, o caminho passa por mais alguns pontos turísticos, destacados pela numeração 1 a 5, uma vez que o *Coolest path* escolhido deu preferência a uma rota turística.

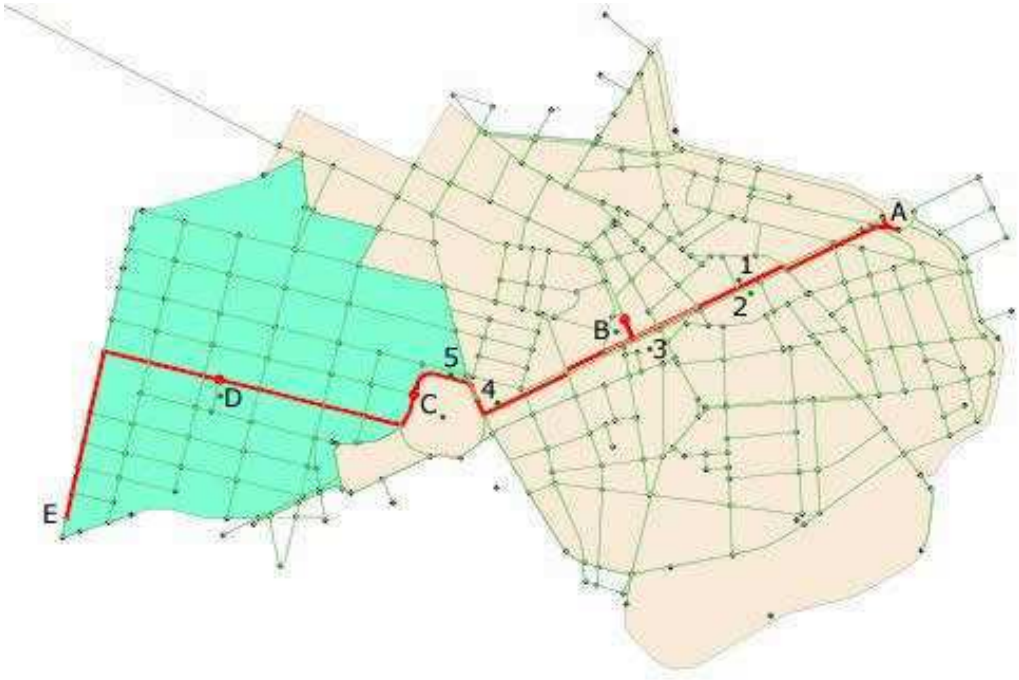

Figura 5.13 Caminho *t-autonomy* com t = 100 e tipo de PoI = turístico

A figura 5.14 apresenta uma consulta *t-autonomy* semelhante à anterior, mas com a constante t = 150 segundos, *shortest* com Pouca Importância, *simplest* com Pouca Importância, *scenic* com Muita Importância, *fastest* com Pouca Importância, e restringindo as paradas a pontos turísticos. O caminho parte de *A* a *C*, e pára no PoI *B*. Além da paradas em um ponto turístico, o caminho passa por mais alguns pontos turísticos, destacados pela numeração 1 a 7, uma vez que o *Coolest path* escolhido deu preferência a uma rota turística.

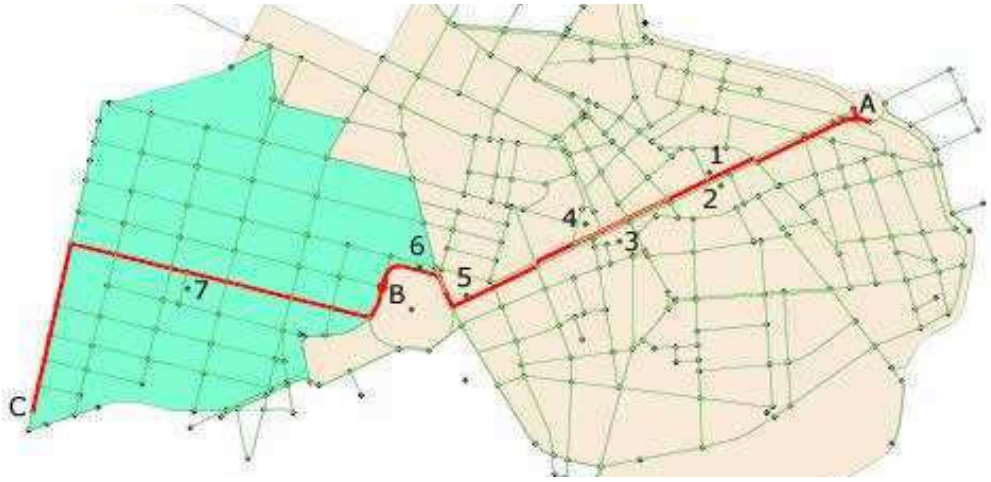

Figura 5.14 Caminho *t-autonomy* com *t* = 150 e tipo de PoI = turístico

Um exemplo de consulta com restrição entre cidades pode ser visto nas figuras 5.15, 5.16 e 5.17. É uma consulta *a-autonomy* com *a* = 10000 metros, *shortest* com Muita Importância, *simplest* com Pouca Importância, *scenic* com Pouca Importância, *fastest* com Média Importância, e paradas em qualquer tipo de PoI. Na figura 5.15 podemos ver a origem, representada por *A*. Na figura 5.16 podemos ver um ponto de parada intermediário, representado por *B*. E na figura 5.17 podemos ver o destino, representado por *C*.

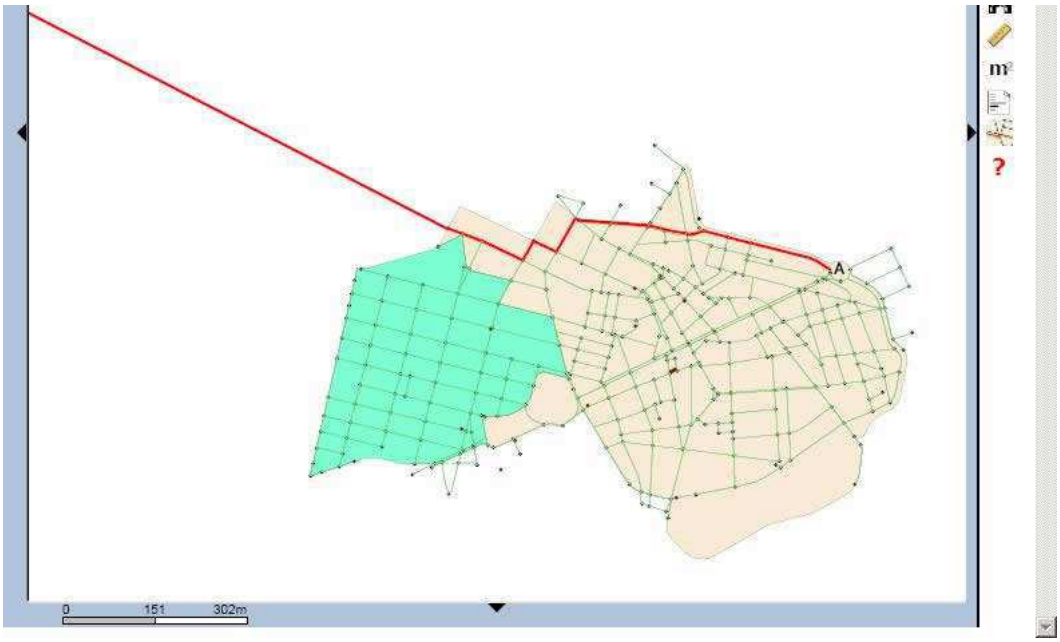

Figura 5.15 Caminho *a-autonomy* partindo de *A*, com *a* = 10000, tipo de PoI = qualquer

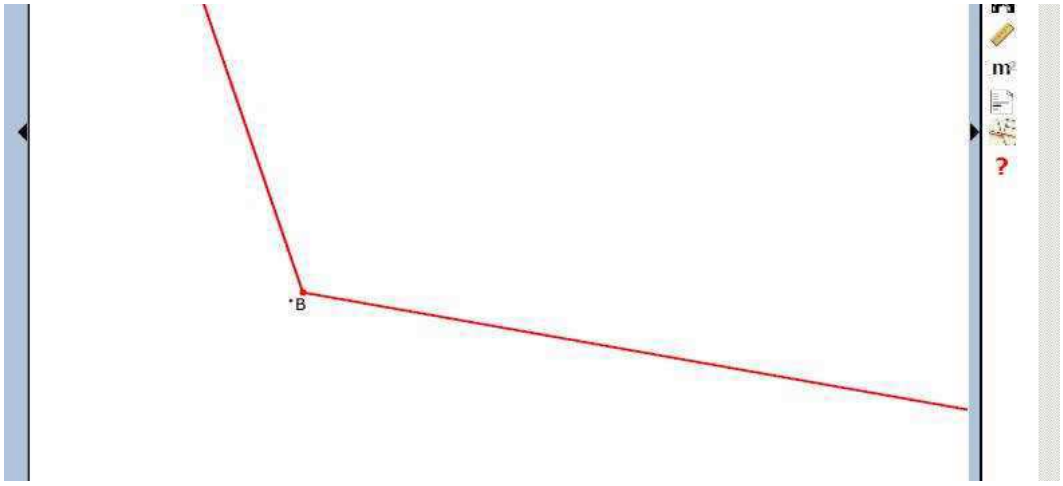

Figura 5.16 Caminho *a-autonomy* passando por *B*, com *a* = 10000, tipo de PoI = qualquer

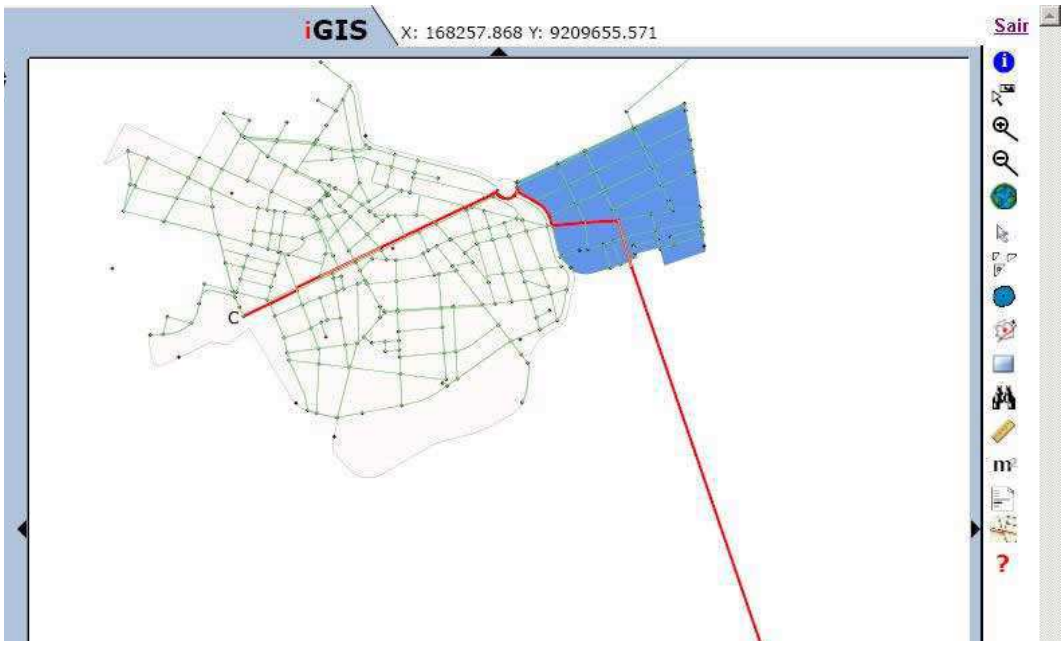

Figura 5.17 Caminho *a-autonomy* chegando a *C*, com *a* = 10000, tipo de PoI = qualquer

5.3 Avaliação de desempenho

Para comparar a eficácia dos algoritmos *a-autonomy* e *t-autonomy*, foi realizada uma avaliação de desempenho. As variações de consultas aos dados gerados pelo *Coolest Path* têm tempos de resposta semelhantes, uma vez que as rotas estão pré-materializadas e o tempo de acesso é apenas o de montar os pedaços da rota.

O tempo de execução dos algoritmos *a-autonomy* e *t-autonomy* não depende somente da distância da rota e dos valores de *a* e *t*. As paradas intermediárias são realizadas em pontos de interesse, então a localização desses pontos é um fator influente

no tempo de resposta. A figura 5.18 apresenta o tempo de execução de uma consulta *a-autonomy* com distância de rede fixa em 2991 metros, parâmetros *shortest*, *simplest*, *fastest* e *scenic* definidos como de Pouca Importância, e variando o valor de *a*. O custo está dividido em tempo gasto com o *Coolest path* e o tempo de execução dos demais passos do algoritmo. O algoritmo *a-autonomy* realiza várias consultas de *Coolest path*, e esse tempo depende da posição dos pontos de interesse. Por exemplo, quando *a* é igual a 1250, o tempo cresce consideravelmente, pois foi necessário realizar mais buscas para encontrar pontos de interesse adequados.

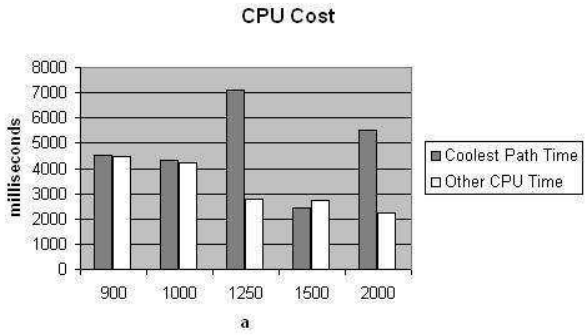

Figura 5.18 Gráfico do tempo de execução de uma consulta *a-autonomy* com distância de rede fixa em 2991 metros

Já quando *a* é igual a 1500, o tempo foi bastante reduzido, pois os pontos intermediários foram encontrados com facilidade, provavelmente nas proximidades de 1500 metros. O tempo de execução dos demais passos do algoritmo é reduzido quando aumentamos o valor de *a*, pois quanto maior *a*, menos pontos intermediários e, conseqüentemente, menos chamadas recursivas na execução do algoritmo. A figura 5.19 apresenta o custo de acesso ao banco de dados, que se comporta de acordo com o custo de execução do *Coolest path*. As consultas para ambos os gráficos foram realizadas permitindo paradas em quaisquer tipos de pontos de interesse.

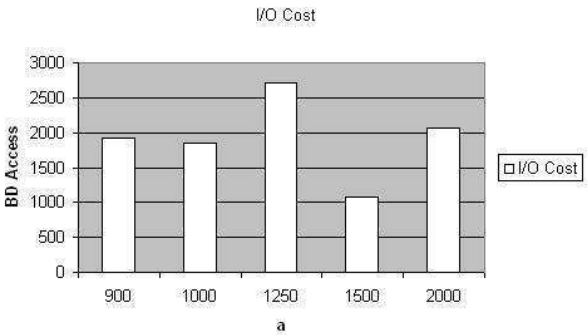

Figura 5.19 Gráfico do custo de acesso ao banco de uma consulta *a-autonomy* com distância de rede fixa em 2991 metros

A figura 5.20 apresenta o tempo de execução da mesma consulta, alterando o parâmetro *shortest* para Muita Importância, enquanto a figura 5.21 com o parâmetro *fastest* sendo de Muita Importância. A execução se comporta de forma semelhante, exceto quando *a* for igual a 1000, caso em que o tempo do *Coolest path* aumentou nestes dois últimos gráficos, novamente devido à localização dos pontos de interesse.

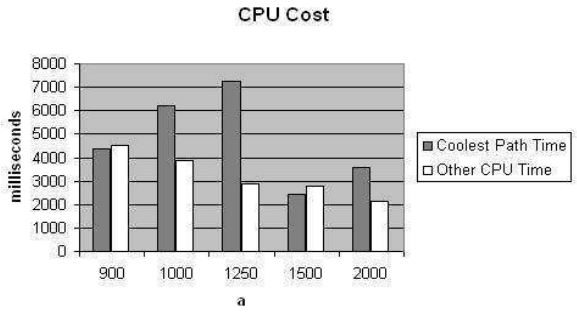

Figura 5.20 Gráfico do tempo de execução de uma consulta *a-autonomy* com o parâmetro *shortest* sendo de Muita Importância

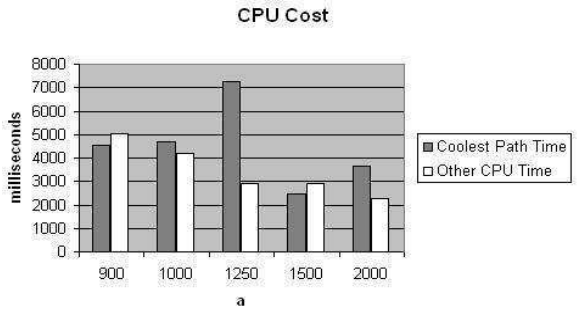

Figura 5.21 Gráfico do tempo de execução de uma consulta *a-autonomy* com o parâmetro fas*test* sendo de Muita Importância

Outro teste realizado foi fixar o parâmetro *a* e variar a distância entre a origem e o destino da rota. A figura 5.22 apresenta o tempo de execução de uma consulta *a-autonomy* com *shortest*, *simplest*, *scenic* e *fastest* definidos como de Pouca Importância, permitindo paradas em todos os tipos de PoI e com *a* definido como 900 metros.

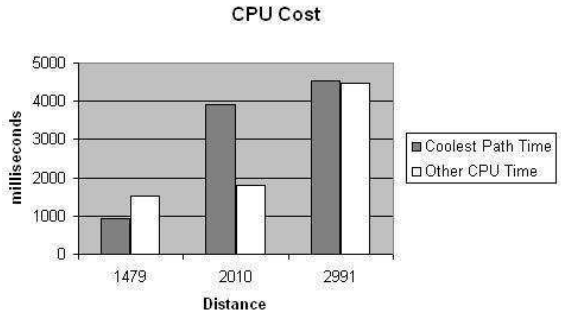

Figura 5.22 Gráfico do tempo de execução de uma consulta *a-autonomy* com *a* igual a 900 metros

Como era esperado, quanto maior a distância, maior o tempo de execução. Da mesma forma, com os acessos ao banco de dados, como pode ser observado na figura 5.23.

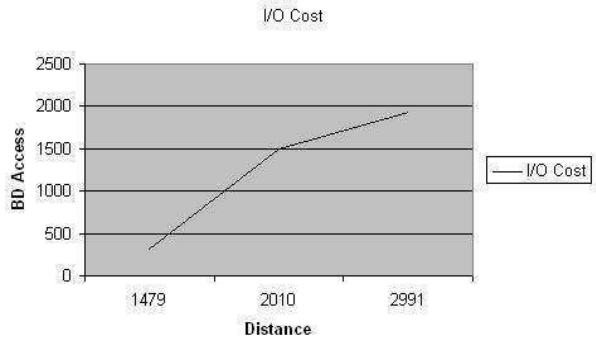

Figura 5.23 Gráfico do custo de acesso ao banco de dados de uma consulta *a-autonomy* com *a* definido como sendo 900 metros

O algoritmo de *t-autonomy* se comporta de forma semelhante. A figura 5.24 apresenta o tempo de execução da mesma consulta da figura 5.18, mas utilizando *t-autonomy*. O tempo do *Coolest path* depende da localização dos pontos de interesse e o tempo do restante do algoritmo tende a decrescer com o aumento do valor de *t*. A figura 5.25 apresenta o custo de acesso ao banco de dados que, novamente, segue a mesma lógica.

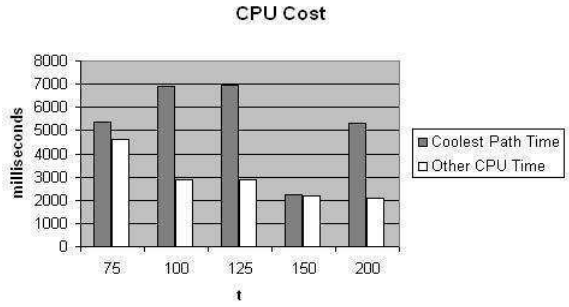

Figura 5.24 Gráfico do tempo de execução de uma consulta *t-autonomy* com distância de rede fixa em 2991 metros

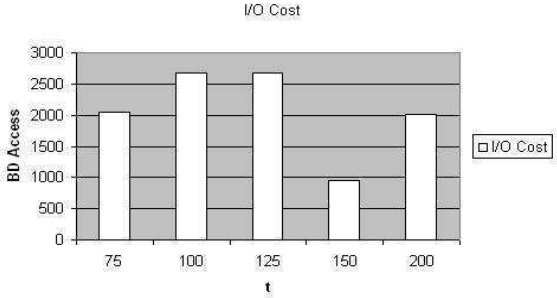

Figura 5.25 Gráfico do custo de acesso ao banco de dados de uma consulta *t-autonomy* com distância de rede fixa em 2991 metros

As figuras 5.26 e 5.27 apresentam o tempo de execução alterando os parâmetros *shortest* para Muita Importância e *fastest* para Muita Importância, respectivamente. Em ambos os gráficos é possível perceber que o tempo de execução do *Coolest path* foi reduzido quando *t* é igual a 125 segundos, o que é explicado pela localização dos PoIs.

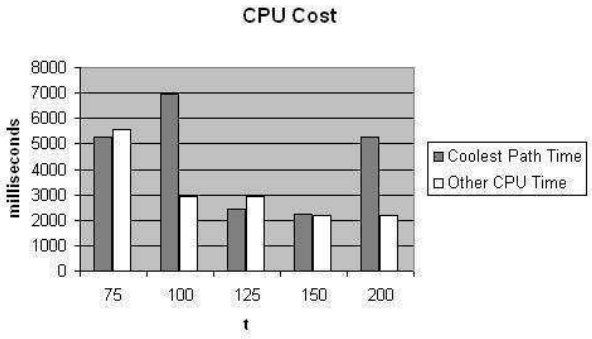

Figura 5.26 Gráfico do tempo de execução de uma consulta *t-autonomy* com o parâmetro *shortest* sendo de Muita Importância

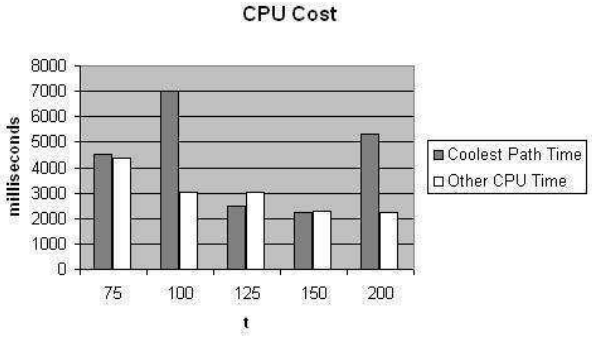

Figura 5.27 Gráfico do tempo de execução de uma consulta *t-autonomy* com o parâmetro *fastest* sendo de Muita Importância

As figuras 5.28 e 5.29 apresentam o tempo de execução e o custo de acesso ao banco de dados com o parâmetro *t* fixo em 75 segundos e variando o tamanho da rota (ou, nesse caso, o tempo total para percorrê-la). A tendência de aumentar o tempo de execução e os acessos ao banco conforme o destino se distancia da origem se repete.

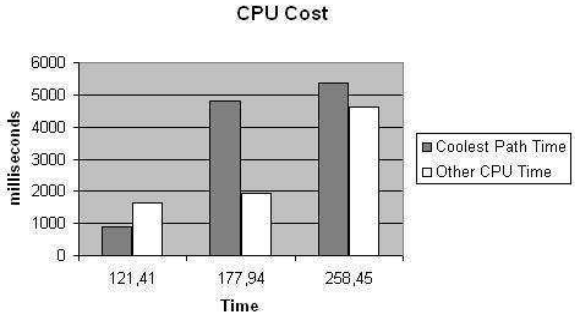

Figura 5.28 Gráfico do tempo de execução de uma consulta *t-autonomy* com *t* fixo em 75 segundos

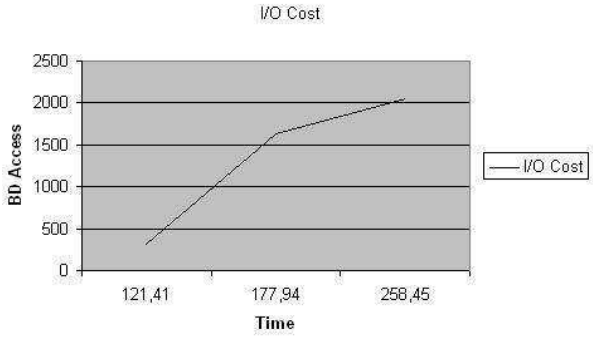

Figura 5.29 Gráfico do custo de acesso ao banco de dados de uma consulta *t-autonomy* com *t* igual a 75 segundos

Capítulo 6 . Conclusão

Graças aos avanços da computação, serviços de geoprocessamento têm sido cada vez mais utilizados, para dar suporte decisório e auxiliar no dia-a-dia das pessoas. Roteamento é uma funcionalidade extremamente útil nestes serviços, no sentido de possibilitar aos usuários descobrirem rotas entre lugares. É também interessante que esses serviços sejam flexíveis e interoperáveis, de forma que possam ser acessados de qualquer dispositivo com conectividade à Internet. Os serviços de busca de rotas oferecidos pelo mercado carecem de flexibilidade e de adaptação às necessidades dos usuários. É ainda interessante que as rotas obtidas possam estar de acordo com um conjunto de características que os usuários esperam delas.

É de grande complexidade implementar um serviço interoperável que ofereça um alto grau de personalização e flexibilidade ao usuário. As principais contribuições deste trabalho são apresentadas na seção 6.1. Foi identificado um conjunto de possíveis trabalhos futuros a serem realizados em uma eventual continuação desta dissertação, trabalhos estes detalhados na seção 6.2.

6.1 Principais Contribuições

A principal contribuição desta dissertação é o desenvolvimento de uma suíte de algoritmos para busca de rotas com alto nível de personalização e possibilidades de acréscimo de restrições, implementados no serviço de busca de rotas *RouteService*, uma arquitetura baseada em *Web Services*. A seguir, uma lista de contribuições do trabalho:

- Implementação em Arquitetura Orientada a Serviços: oferece padronização de acesso e interoperabilidade, de forma que o *RouteService* pode ser acessado de qualquer dispositivo com conectividade à Web, sendo necessário apenas que o dispositivo se conecte com o *Web Service*. Desta forma, pode ser realizada uma busca de rotas ao mesmo serviço através de um computador *desktop*, de um quiosque turístico ou de um celular. As ferramentas oferecidas pela Internet e avaliadas na dissertação não possuem esta arquitetura flexível, mas sim uma proprietária;

- Personalização: oferece um conjunto de critérios aos quais o usuário pode atribuir pesos, de forma a obter uma rota adequada às suas necessidades. Os critérios oferecidos são: distância (tamanho do caminho); simplicidade (facilidade de seguir o caminho); capacidade turística (quantidade de pontos turísticos no caminho); e tempo (tempo levado para perfazer o caminho). A cada uma destas características, o usuário pode atribuir os seguintes pesos: Pouca Importância; Média Importância; e Muita Importância. O serviço oferece o *Coolest path*, um algoritmo que realiza a média ponderada dos critérios acima, de acordo com os pesos definidos pelo usuário, e encontra a rota que mais se aproximar do desejado. De acordo com a pesquisa realizada, não há outra abordagem que ofereça similar nível de personalização em busca de rotas;
- Restrições: oferece a possibilidade do usuário acrescentar restrições em suas rotas, onde por restrições entende-se critérios para realização de paradas intermediárias. O serviço oferece duas restrições: *a-autonomy*, onde o usuário define uma constante *a* e um tipo de Ponto de Interesse, e a rota encontrada realizará paradas em pontos desse tipo a cada *a* unidades de distância; e *t-autonomy*, onde o usuário define uma constante *t* e um tipo de Ponto de Interesse desejado, e a rota obtida realizará paradas em pontos desse tipo a cada *t* unidades de tempo. O usuário então define suas preferências e se deseja uma dessas duas restrições na rota, e o serviço encontrará o *Coolest path,* acrescentando a este as paradas necessárias, de acordo com a restrição escolhida. A principal contribuição é a extensão dos algoritmos de busca de rotas com restrições para redes de estradas, uma vez que eles tinham sido propostos por Terrovitis et al. apenas em espaço euclidiano. Ademais, o *t-autonomy*, pela pesquisa bibliográfica realizada, é uma proposta original;
- Materialização de rotas: o serviço pré-calcula as rotas para todas as possibilidades de pesos das características de acordo com uma organização hierárquica do grafo das ruas. O grafo é particionado e para cada fragmento todas as possibilidades de rotas são calculadas e materializadas em visões do banco de dados. Quando uma rota é solicitada, há um algoritmo para montar o caminho, obtendo os trechos de rota de cada fragmento da hierarquia.

Foi realizado um estudo de ferramentas de busca de rotas disponíveis na Web e construído um quadro comparativo entre eles, conforme visto na seção 2.3.2. O quadro 6.1 apresenta a comparação entre as ferramentas, acrescentando ao final o *RouteService*.

Quadro 6.1 Comparando ferramentas com o *RouteService*

Características / Ferramentas	Rotas intra e inter cidades integradas	Nível de personalização	Arquitetura flexível	Acesso móvel	Paradas intermediárias
Maplink	Não	Médio	Não	Inexistente	Endereços quaisquer
Apontador	Sim	Médio	Não	Inexistente	Inexistente
Mappy	Sim	Médio	Não	Inexistente	Endereços quaisquer
Map24	Sim	Médio	Não	Inexistente	Inexistente
Google Maps	Sim	Baixo	Não	Inexistente	Endereços quaisquer
Yahoo Maps	Sim	Baixo	Não	Inexistente	Endereços quaisquer
Navegador Quatro Rodas	Não	Baixo	Não	Completo	Endereços quaisquer
Navegador Airis	Sim	Baixo	Não	Completo	Endereços quaisquer
Navegador Delphi	Sim	Baixo	Não	Completo	Endereços quaisquer
Navegador Navisystem	Não	Médio	Não	Completo	Endereços quaisquer
Guia Fácil	Não	Baixo	Não	Parcial (em quiosques)	Inexistente
RouteService	**Sim**	**Alto**	**Sim**	**Parcial (apenas acesso, sem objetos móveis)**	**Em Pontos de Interesse**

É importante destacar que o *RouteService* é o único que oferece alto nível de personalização e com arquitetura flexível. O acesso móvel é apenas parcial, mas as buscas com objetos móveis monitorados por GPS estão listadas em trabalhos futuros. Da mesma forma, é um trabalho futuro permitir paradas intermediárias em quaisquer endereços, seja um ponto de interesse ou não.

Na seção 2.3.3 foi apresentado um quadro comparativo entre os trabalhos científicos relacionados. O quadro 6.2 apresenta o mesmo comparativo, acrescentando ao final o *RouteService*. A única característica que o serviço não atende é a disponibilização de instruções de rota em forma textual, que está listada como um trabalho futuro. Mas esta característica é simples e muitos sistemas já a oferecem, de maneira que não é uma grande contribuição acadêmica a sua implementação. Vale destacar que na característica de informações do usuário, o *RouteService* atende com personalização, e na característica de objetos móveis, o serviço é apenas acessível via dispositivos móveis, mas funcionará com objetos monitorados via GPS em trabalhos futuros.

Quadro 6.2 Comparando trabalhos relacionados com o *RouteService*

Características / Artigos	Redes de estradas	Paradas intermediárias em pontos de interesse	Consulta pelo mais próximo	Busca de rotas	Utiliza informações do usuário	Preocupação com as instruções das rotas	Leva dados de tráfego em consideração	Considera os custos das curvas	Materialização	Objetos móveis
Yoo e Shekhar	S	S	S	N	N	N	N	N	S	S
Yiu et al.	S	N	S	N	N	N	N	N	N	S
Shahabi et al.	S	N	S	N	N	N	N	N	N	S
Gupta et al.	S	N	N	N	N	N	N	N	N	N
Liu	S	N	N	S	*	N	N	N	N	N
Richter et al.	S	N	N	N	N	S	N	N	N	N
Kanoulas et al.	S	N	N	S	N	N	S	N	N	N
Sharifzadeh e Shahabi	S	S	N	S	N	N	N	N	N	N
Terrovitis et al.	N	S	N	S	N	N	N	N	N	N

Características / Artigos	Redes de estradas	Paradas intermediárias em pontos de interesse	Consulta pelo mais próximo	Busca de rotas	Utiliza informações do usuário	Preocupação com as instruções das rotas	Leva dados de tráfego em consideração	Considera os custos das curvas	Materialização	Objetos móveis
Caldwell	S	N	N	N	N	N	N	S	N	N
Winter	S	N	N	N	N	N	N	S	N	N
Duckham e Kulik	S	N	N	S	N	N	N	S	N	N
Kirby e Potts	S	N	N	N	N	N	N	S	N	N
Jing et al.	S	N	N	S	N	N	N	N	S	N
Jung e Pramanik	S	N	N	S	N	N	N	N	S	N
Shekhar et al.	S	N	N	S	N	N	N	N	S	N
Almeida e Güting	S	N	N	N	N	N	N	N	N	S
Balke et al.	S	N	N	S	**	N	N	N	N	N
Brilingaite et al.	S	N	N	N	N	N	S	N	S	S
Shekhar e Liu	S	N	N	N	N	N	N	N	N	S
Malaka e Zipf	S	N	N	S	N	N	N	N	N	N
RouteService	**S**	**S**	**S**	**S**	****	**N**	**S**	**S**	**S**	*****

* Histórico

** Personalização

*** O serviço pode ser acessado de um dispositivo móvel, mas não trabalha com GPS

6.2 Trabalhos Futuros

Diante do que foi implementado nos algoritmos e no serviço de roteamento, foram levantadas algumas funcionalidades interessantes que não foram realizadas, seja pela ausência de tempo ou por não serem o escopo dessa dissertação. Seguem a seguir um conjunto de trabalhos a serem realizados no futuro, entre novas funcionalidades,

correções e melhorias:

- Uma possível melhoria ao sistema seria guardar no banco de dados um histórico das consultas realizadas por cada usuário. Diante deste histórico, poderia ser implementado um sistema de Raciocínio Baseado em Casos, que utilizaria regras de Inteligência Artificial para inferir consultas. Por exemplo, se um usuário sempre consulta caminhos priorizando a distância em detrimento das demais características, o sistema poderia inferir e numa próxima consulta, oferecer a distância priorizada e poupar o tempo do usuário;
- Para se adequar melhor à realidade, as rotas poderiam ser multi-meios, ou seja, o usuário poderia selecionar um conjunto de meios de transporte (como trens, metrôs, ônibus, a pé, etc.) e o sistema encontraria rotas divididas. Por exemplo: "Ande quinhentos metros em linha reta, entre na estação do metrô, pegue-o e desça na estação X. Em seguida, caminhe trezentos metros e pegue o ônibus Z, etc.";
- Terrovitis et al. [TBP+05] sugerem o algoritmo *k-stops* para caminhos com restrição em espaço euclidiano. Seria interessante adaptar esse algoritmo para redes de estradas, como foi feito com *a-autonomy*. Também seria interessante investigar outros tipos de restrições que poderiam ser utilizadas;
- O tempo para trafegar pelos caminhos poderia ser mais exato, e para tanto seria necessário levar em consideração o tempo de espera dos carros em semáforos e o tempo de transição necessário para realizar-se as curvas. Portanto, poderia ser interessante acrescentar essas variáveis aos cálculos de tempo;
- É importante que as rotas sejam retornadas na forma de instruções textuais ao usuário, de forma que o caminho encontrado possa ser melhor compreendido;
- Seria de grande utilidade se o serviço de busca de rotas pudesse ser utilizado em dispositivos móveis, como celulares e PDAs (*Personal Digital Assistant*). Pelo fato da arquitetura ser orientada a serviços, ela pode ser acessada de qualquer cliente, bastando para este comunicar-se utilizando o padrão de *Web services*. É necessário então utilizar um cliente móvel para testar a comunicação com o serviço e a obtenção das rotas;
- O serviço tornar-se-ia extremamente útil caso tratasse objetos móveis com localização, de forma que fosse possível realizar busca de rotas através de dispositivos móveis com GPS (*Global Positioning System*), tendo como origem

a localização atual do usuário. Muitas vezes um usuário se desvia da rota original por motivo de erro ou por alguma outra razão, e seria interessante se as rotas se adaptassem em tempo de execução à nova posição do usuário. Por exemplo, se o usuário errasse uma transição o caminho seria recalculado com a mudança de rota;

- Seria interessante realizar uma análise de desempenho mais acurada dos algoritmos propostos, de forma a identificar possíveis melhorias no tempo de execução. Para tal, será necessário utilizar um conjunto maior de dados, com ruas de grandes cidades, interconectadas por várias rodovias. Também seria possível verificar a eficácia dos algoritmos em uma quantidade de dados superior;
- O gerador de visões materializadas implementado é lento e há um conjunto de melhorias que podem ser realizadas na implementação e na forma como os dados são acessados no banco de dados, de forma a melhorar seu desempenho. Também seria interessante a realização de um refatoramento no software, visando torná-lo mais simples, facilitando a extensibilidade;
- Para que o serviço torne-se mais realista, seria interessante que fosse possível tratar os dados de tráfego em tempo real, ou seja, verificar o atual tráfego das ruas e levá-lo em consideração no cálculo das rotas. Para tanto, há duas possibilidades: os dados de tráfego poderiam ser atualizados por um agente humano com freqüência; ou sensores poderiam ser instalados nas ruas, de forma a captar a movimentação e inferir o tráfego atual;
- Hoje as restrições de paradas intermediárias podem ser realizadas apenas em pontos de interesse. Seria interessante acrescentar um tipo de restrição no qual o usuário pudesse realizar uma busca de rota composta, com pontos intermediários em quaisquer endereços fornecidos pelo usuário. Por exemplo, o usuário solicita uma rota entre o endereço *A* e o *B*, mas ele deseja que essa rota passe por *C* e *D* antes de chegar a *B*;
- Uma última atividade a ser implementada seria a atualização automática das visões materializadas de acordo com o cadastro de novas interrupções, funcionalidade esta que não pôde ser realizada devido à falta de tempo.

Referências Bibliográficas

[ABr07] Airis Brasil. Navegador GPS Airis. Disponível em: <http://www.airis.com.br/index2.asp?Pais=BR>. Acesso em: janeiro de 2007.

[ACK+03] Alonso, G.; Casati, F.; Kuno, H.; Machiraju, V. *Web Services*, Springer, 2003.

[AG05] Almeida, V. T. de; Güting, R. H. *Indexing the Trajectories of Moving Objects in Networks*. Geoinformatica, 9, 1, p. 33-60, 2005.

[Apo07] Apontador. Seu Guia de Ruas na Internet. Disponível em: <http://www.apontador.com.br>. Acesso em: fevereiro de 2007.

[Ari07] Ariasat – Soluções em rastreamento, logística e telemetria. Navegador GPS Automotivo Delphi NAV200. Disponível em: <http://www.ariasat.com.br/>. Acesso em: março de 2007.

[Bel68] Bellman, R. *On a routing problem*. Quarterly of Applied Mathematics, 16, p. 87–90, 1958.

[BJZ05] Brilingaite, A.; Jensen, C. S.; Zokaite, N. *Enabling routes as context in mobile services*. Technical Report – Department of Computer Science, Aalborg University, 2005.

[BKU03a] Balke, W.-T.; Kießling, W.; Unbehend, C. *Personalized Services for Mobile Route Planning*. Proceedings of International Conference on Data Engineering, Bangalore, India, 2003. IEEE Computer Society.

[BKU03b] Balke, W-T.; Kießling, W.; Unbehend, C. *Performance and Quality Evaluation of a Personalized Route Planning System*. Proceedings of Simpósio Brasileiro de Bancos de Dados, Manaus, Amazonas, Brasil, 2003. SBC.

[BLS+04] Baptista, C. S.; Leite Jr., F. L.; Silva, E. R.; Paiva, A. C. *Using Open Source GIS in e-Government Applications*. In Proceedings of Third International Conference on e-Government (EGOV), Zaragoza, Spain, 2004, Lecture Notes in Computer Science - Springer.

[BNS+05] Baptista, C. S.; Nunes, C. P.; Sousa, A. G.; Silva, E. R.; Leite Jr., F. L.; Paiva, A. C. *On Performance Evaluation of Web GIS Applications*. In Proceedings of 2nd International Workshop on Geographic Information Management, 15th International Workshop on Database and Expert Systems Applications (DEXA'05), 2005. IEEE Computer Society.

[BM98] Burrough, P.; McDonnell, R. *Principles of Geographical Information Systems*, Oxford Press, 1998.

[Cal61] Caldwell, T. *On Finding Minimum Routes in a Network with Turn Penalties*. Communications of the ACM, 4, 2, p. 107-108, 1961.

[CHV+07] Clement, L.; Hately, A.; Von Riegen, C.; Rogers, T. UDDI Version 3.0.2, UDDI Spec Technical Committee Draft, Dated 2004-10-19. Disponível em <http://uddi.org/pubs/uddi_v3.htm>. Acesso em: abril de 2007.

[CLR+01] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 2001.

[CMR+07] Chinnici, R.; Moreau, J-J.; Ryman, A.; Weerawarana, S. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C Working Draft 26 March 2007. Disponível em <http://www.w3.org/TR/wsdl20/>. Acesso em: abril de 2007.

[CPT+06] Costa, D. C.; Paiva, A. C.; Teixeira, M. M.; Baptista, C. S.; Silva, E. R. A *Progressive Transmission Scheme for Vector maps in low bandwidth environments based on device rendering*. In Proceedings of 3rd International Workshop on Conceptual Modeling for Geographic Information Systems (CoMoGIS2006). Tucson, Arizona, USA, 2006. LNCS. Springer Verlag.

[CW01a] Corona, B.; Winter, S. *Navigation information for pedestrians from city maps*. Proceedings of AGILE Conference on Geographic Information Science, Brno, Czech Republic, 2001. Association Geographic Information Laboratories Europe.

[CW01b] Corona, B.; Winter, S. *Guidance of car drivers and pedestrians*. Technical Report – Department of Geomatics, The University of Melbourne, 2001.

[Die05] Diestel, R. *Graph Theory*. Third Edition, Springer-Verlag, New York, 2005.

[Dij59] Dijkstra, E. W. *A note on two problems in connection with graphs*. Numerische Mathematik, 1, 1, p. 269-271, 1959.

[DK03] Duckham, M.; Kulik, L. *"Simplest" Paths: Automated Route Selection for Navigation*. Proceedings of Spatial Information Theory: Foundations of Geographic Information Science, Kartause Ittingen, Switzerland, 2003. Lecture Notes in Computer Science.

[EAA+04] Endrei, M.; Ang. J.; Arsanjani, A.; Chua, S.; Comte, P.; Krogdahl, P.; Luo, M.; Newling, T. *Patterns: Service-Oriented Architecture and Web Services, IBM Redbooks*, Draft Document for Review March 31, 2004.

[ESR97] ESRI. ESRI Shapefile Technical Description. White paper, 1998, ESRI. Disponível em: <http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf>. Acesso em: janeiro de 2007.

[ESR07] ESRI. ArcView - Desktop GIS for Mapping, Data Integration, and Analysis. Disponível em: <http://www.esri.com/software/arcview/>. Acesso em: janeiro de 2007.

[FF62] Ford, L.; Fukelson, D. Flows in networks. Princeton University Press, 1962.

[Fow03] Fowler, M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Third Edition, Addison-Wesley Professional, 2003.

[GF07] Guia Fácil. Terminais Guia Fácil. Disponível em: <http://www.guiafacilnet.com/>. Acesso em: fevereiro de 2007.

[GHM+03] Gudgin, M.; Hadley, M.; Mendelsohn, N.; Moreau, J-J.; Nielsen, H. F. SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation 24 June 2003. Disponível em <http://www.w3.org/TR/2003/REC-soap12-part1-20030624/>. Acesso em: abril de 2007.

[GKR04] Gupta, S.; Kopparty, S.; Ravishankhar, C. V. *Roads, Codes and Spatiotemporal Queries*. Proceedings of Symposium on Principles of Database Systems, Paris, France, 2004. ACM Press.

[GM07] Google Maps. Disponível em: <http://maps.google.com/>. Acesso em: fevereiro de 2007.

[Gol95] Golledge, R. G. *Path Selection and Route Preference in Human Navigation: A Progress Report*, Proceedings of Spatial Information Theory: A Theoretical Basis for GIS, Semmering, Austria, 1995. Lecture Notes in Computer Science.

[GQR07] Guia Quatro Rodas. Navegador Guia Quatro Rodas. Disponível em: <http://www.navegadorguiaquatrorodas.com.br/index.php>. Acesso em: janeiro de 2007.

[GT01] Goodrich, M. T.; Tamassia, R. *Algorithm Design: Foundations, Analysis, and Internet Examples*. John Wiley & Sons, 2001.

[HCC98] Heywood, I.; Cornelius, S.; Carver, S. *An Introduction to Geographical Information Systems*. Addison Wesley, 1998.

[HJR95] Huang, Y. W.; Jing, N.; Rundensteiner, E. A. *Hierarchical Path Views: A Model Based on Fragmentation and Transportation Road Types*. Proceedings of International Workshop on Advances in Geographic Information Systems, 1995, Baltimore, Maryland. ACM Press.

[HJR97] Huang, Y.-W.; Jing, N; Rundensteiner, E. A. *A Hierarchical Path View Model for Path Finding in Intelligent Transportation Systems*. Geoinformatica, 1, 2, p. 125-159, 1997.

[IBM07] IBM Software. DB2 9. Disponível em: <http://www-306.ibm.com/software/data/db2/>. Acesso em: janeiro de 2007.

[JC04] Jiang, B.; Claramunt, C. *A Structural Approach to the Model Generalization of an Urban Street Network*. Geoinformatica, 8, 2, p. 157-171, 2004.

[JHR96] Jing, N.; Huang, Y.-W.; Rundensteiner, E. A. *Hierarchical Optimization of Optimal Path Finding for Transportation Applications*. Proceedings of Conference on Information and Knowledge Management, 1996, Rockville, Maryland. ACM Press.

[JHR98] Jing, N.; Huang, Y.-W.; Rundensteiner, E. A. *Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation*. IEEE Transactions on Knowledge and Data Engineering, 10, 3, p. 409-432, 1998.

[JP02] Jung, S.; Pramanik, S. *An Efficient Path Computation Model for Hierarchically Structured Topographical Road Maps*. IEEE Transactions on Knowledge and Data Engineering, 14, 5, p. 1029-1046, 2002.

[JPG07] Joint Photographic Experts Group. JPEG Committee. Disponível em: <http://www.jpeg.org/>. Acesso em: janeiro de 2007.

[KDX+06] Kanoulas, E.; Du, Y.; Xia, T.; Zhang, D. *Finding Fastest Paths on A Road Network with Speed Patterns*. Proceedings of International Conference on Data Engineering, 2006. IEEE Computer Society.

[KP69] Kirby, R. F.; Potts, R. B. *The minimum route problem for networks with turn penalties and prohibitions*. Proceedings of Transportation Research, 1969.

[LGM+01] Longley, P.; Goodchild, M.; Maguire, D.; Rhind, D. *Geographic Information Systems and Science*. John Wiley, Chichester, 2001.

[Liu96] Liu, B. *Intelligent Route Finding: Combining Knowledge, Cases and An Efficient Search Algorithm*. Proceedings of European Conference on Artificial Intelligence, Budapest, Hungary, 1996. John Wiley & Sons, Ltd.

[M2407] Map24. Mapas e Rotas Rodoviárias e Urbanas para Brasil, Europa e EUA. Disponível em: <http://www.br.map24.com/>. Acesso em: janeiro de 2007.

[Map07] Mappy. Road Guide. Disponível em: <http://www.mappy.com>. Acesso em: fevereiro de 2007.

[MBA+02] Miranda, R. V.; Baptista, C. S.; Almeida, R. R.; Catão, B.; Pazinatto, E. *IGIS: um Framework para Sistemas de Informações Geográficas em N-Camadas usando um SGBD Objeto-Relacional*. In Proceedings of Brazilian Symposium on GeoInformatics (GeoInfo) 2002 , Caxambu, Brasil, SBC.

[MLi07] MapLink. Rotas Urbanas e Rodoviárias. Disponível em: <http://maplink.uol.com.br/index.asp>. Acesso em: janeiro de 2007.

[MZ00] Malaka, R.; Zipf, A. *Deep Map - challenging IT research in the framework of a tourist information system*. Proceedings of Information and communication technologies in tourism, Wien, New York, USA, 2000. Springer.

[Nav07] Navisystem. Navisystem DOTB-300. Disponível em: <http://www.navisystem.com.br/>. Acesso em: março de 2007.

[NSS02] Nagappan, R.; Skoczylas, R.; Sriganesh, R. P. *Developing Java Web Services: Architecting and Developing Secure Web Services Using Java*, Wiley, 2002.

[OGC07] Open Geospatial Consortium. OGC Website. Disponível em: <http://www.opengeospatial.org/>. Acesso em: janeiro de 2007.

[Ora07] Oracle. Oracle 10g Database. Disponível em: <http://www.oracle.com/index.html>. Acesso em: janeiro de 2007.

[Pen97] Peng, Z-R. *An Assessment of the Development of Internet GIS*. In proceedings of Urban and Regional Information Systems Association Conference, Toronto, 1997.

[Pin07] Pinheiro, D. Uol Tecnologia. GPS funciona como guia de ruas online dentro do carro. Disponível em: <http://tecnologia.uol.com.br/ultnot/2007/02/08/ult4213u31.jhtm>. Acesso em: março de 2007.

[Pos07a] PostgreSQL. PostgreSQL 8.2. Disponível em: <http://www.postgresql.org/>. Acesso em: janeiro de 2007.

[Pos07b] PostGIS. PostGIS 1.2. Disponível em: <http://www.postgis.org/>. Acesso em: abril de 2007.

[PSL+04] Paiva, A. C.; Silva, E. R.; Leite Jr., F. L.; Baptista, C. S. *A Multiresolution Approach for Internet GIS Applications*. Proceedings of 1st International Workshop on Geographic Information Management, 15th International Workshop on Database and Expert Systems Applications (DEXA'04), Zaragoza, Spain, 2004. IEEE Computer Society.

[RKF04] Richter, K-F.; Klippel, A.; Freksa, C. *Shortest, Fastest, -- but what Next? A Different Approach to Route Directions*. Proceedings of Geoinformation und Mobilität, - von der Forschung zur praktischen Anwendung. Beiträge zu den Münsteraner GI-Tagen, Münster, Germany, 2004. IfGIprints.

[RN02] Russel, S.; Norvig, P. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[Sal06] Salkind, N. J. (Editor) *Encyclopedia of Measurement and Statistics*, Sage Publications, Inc., 2006.

[SC03] Shekhar, S.; Chawla, S. *Spatial Databases: A Tour*, Prentice Hall, 2003.

[SFG97] Shekhar, S.; Fetterer, A.; Goyal, B. *Materialization Trade-Offs in Hierarchical Shortest Path Algorithms. Proceedings of Advances in Spatial Databases*, 1997, Berlin, Germany. Lecture Notes in Computer Science.

[SJK03] Speicys, L.; Jensen, C. S.; Kligys, A. *Computational data modeling for network-constrained moving objects*. Proceedings of ACM International Symposium on Advances in Geographic Information Systems, New Orleans, Louisiana, USA, 2003. ACM Press.

[SKC93] Shekhar, S.; Kohli, A.; Coyle, M. *Path Computation Algorithms for Advanced Traveller Information Systems (ATIS)*. Proceedings of International Conference on Data Engineering, Vienna, Austria, 1993. IEEE Computer Society.

[SKS02] Shahabi, C.; Kolahdouzan, M. R.; Sharifzadeh, M. *A road network embedding technique for k-nearest neighbor search in moving object databases*. Proceedings of ACM International Symposium on Advances in Geographic Information Systems, McLean, VA, USA, 2002. ACM Press.

[SL94] Shekhar, S.; Liu, D-R. *Genesis and Advanced Traveler Information Systems (ATIS): Killer Applications for Mobile Computing*. Proceedings of Workshop on Mobile and Wireless Information Systems, New Brunswick, NJ, 1994. National Science Foundation.

[SS06] Sharifzadeh, M.; Shahabi, C. *Additively Weighted Voronoi Diagrams for Optimal Sequenced Route Queries*. Proceedings of Workshop on Spatio-Temporal Database Management, 2006. ACM Press.

[TBP+05] Terrovitis, M.; Bakiras, S.; Papadias D.; Mouratidis, K. *Constrained Shortest Path Computation*. Proceedings of 8th Internacional Symposium on Advances in Spatial and Temporal Databases, Rio de Janeiro, Brasil, 2005. Springer Lecture Notes in Computer Science.

[TH02] Timpf, S.; Heye, C. *Complexity of routes in multi-modal wayfinding (extended abstract)*. Proceedings of International Conference on Geographic Information Science,

Boulder, Colorado, USA, 2002. GIScience.

[THP+04] Torrens, M.; Hertzog, P.; Pu, P.; Faltings, B. *Towards an intelligent mobile travel assistant*. Proceedings of ACM Symposium on Applied Computing, Nicosia, Cyprus, 2004. ACM Press.

[VW01] Vazirgiannis, M.; Wolfson, O. *A Spatiotemporal Model and Language for Moving Objects on Road Networks*. Proceedings of Symposium on Spatial and Temporal Databases, Redondo Beach, CA, USA, 2001. Lecture Notes in Computer Science.

[W3C07] World Wide Web Consortium. Scalable Vector Graphics. Disponívem em: <http://www.w3.org/Graphics/SVG/>. Acesso em: janeiro de 2007.

[WD04] Worboys, M.; Duckham, M. *GIS: A Computing Perspective*, CRC Press, 2004.

[Wik07a] Wikipédia. Breadth-first Search. Disponível em: <http://en.wikipedia.org/wiki/Breadth-first_search>. Acesso em: fevereiro de 2007.

[Wik07b] Wikipédia. Taxicab Geometry. Disponível em: <http://en.wikipedia.org/wiki/Taxicab_geometry>. Acesso em: fevereiro de 2007.

[Win01] Winter, S. *Weighting the Path Continuation in Route Planning*. Proceedings of ACM International Symposium on Advances in Geographic Information Systems, Atlanta, GA, USA, 2001. ACM Press.

[Win02] Winter, S. *Modeling Costs of Turns in Route Planning*. Geoinformatica, 6, 4, p. 345-361, 2002.

[YM07] Yahoo! Maps, Driving Directions, and Traffic. Disponível em: <http://maps.yahoo.com>. Acesso em: fevereiro de 2007.

[YMP05] Yiu, M. L.; Mamoulis, N.; Papadias, D. *Aggregate Nearest Neighbor Queries in Road Networks*. IEEE Transactions on Knowledge and Data Engineering, 17, 6, p.

820-833, 2005.

[YS05] Yoo, J. S.; Shekhar, S. *In-Route Nearest Neighbor Queries*. Geoinformatica, 9, 2, p. 117-137, 2005.

[Ziv03] Ziviani, N. *Projeto de Algoritmos com Implementações em Pascal e C*. Editora Thomson, 2003.