ANTARES: Um sistema Web de consulta de rotas de ônibus como serviço público

Rodrigo Bastos[1]
Patrícia A. Jaques[2]

Resumo: Este artigo apresenta um sistema Web para busca de rotas de ônibus como ferramenta de auxílio a usuários de transporte público. A aplicação desenvolvida usa o algoritmo A*, conhecido no campo de inteligência artificial, como base para a geração de rotas de ônibus. A API do Google Maps é empregada para ilustrar os resultados visuais e descritivos de dada busca. Dois experimentos foram realizados a fim de avaliar a usabilidade e corretude da ferramenta.

Palavras-chave: Sistemas de informação. Sistemas Web. Inteligência artificial**.**

Abstract: *This paper presents a Web system for bus trajectory search as a tool to help public transport users. The developed application uses the A* algorithm, known on the Artificial Intelligence field, as a base for the generation of bus routes. The Google Maps API is employed to illustrate the descriptive and visual results of a given search. Two experiments were accomplished in order to evaluate the system usability and correctness.*

Keywords: *Information systems. Web systems. Artificial intelligence.*

1 Introdução

A Web tem se constituído na mídia mais promissora desde a invenção da televisão. Enquanto a televisão é um meio de comunicação em que o emissor domina todo o processo e o receptor não pode responder em tempo real, a Web oferece a oportunidade de interação, através de uma tecnologia naturalmente aberta e possibilita a colaboração coletiva de novas ideias e obras [1].

A distância geográfica deixou de ser um entrave, mas a econômica (ricos e pobres), a cultural (acesso efetivo pela educação continuada) e a tecnológica (acesso e domínio ou não das tecnologias de comunicação) ganharam força. Contudo, com o constante crescimento do número de usuários da Web no mundo, ampliaram-se as possibilidades em torno de suas aplicações; assim, áreas como educação, comércio e entretenimento ganharam seu espaço na rede.

Além dos sistemas educacionais, de entretenimento ou corporativos, um outro tipo de aplicação vem ganhando espaço na Web: os sistemas destinados à prestação de serviços, ou, como são conhecidos, software como serviço (do inglês software as a service). Um exemplo deste tipo de aplicação está relacionado à pesquisa de telefones e estabelecimentos – a Telelista [2]. Nesta página Web são oferecidos telefones úteis, possibilidade de busca de telefones por atividades, serviço, produto, marca, entre outros, além de filtros adicionais por estado ou cidade. Por sua vez, os serviços oferecidos pelo site dos Correios [3] variam entre o monitoramento de uma mercadoria, consulta de CEP ou endereço, cálculos de prazos e preços, entre outros.

[1] Programa Interdisciplinar de Pós-graduação em Ciência da Computação - UNISINOS
{rodrigobas@gmail.com}
[2] Programa Interdisciplinar de Pós-graduação em Ciência da Computação - UNISINOS
{pjaques@unisinos.br}

doi: 10.5335/rbca.2010.005

A empresa Google [4] criou, recentemente, um serviço de localização por meio de imagens de satélites, mapas, definição de rotas, etc chamado Google Maps [5], por meio do qual são disponibilizadas diversas opções de consultas e visualizações ao usuário, como, por exemplo, pesquisar um determinado endereço, encontrar empresas ou negócios, visualização por mapa, satélite ou terreno, entre outros, através de uma interface amigável. Além disso, o Google Maps oferece a opção de integrar os diferentes tipos de mapas a qualquer site particular através de uma Interface de Programação de Aplicativos, ou, em inglês, Application Programming Interface (API), desenvolvida na linguagem JavaScript chamada Google Maps API [6].

Inspirado nas tecnologias e possibilidades do Google Maps, o presente trabalho propõe um complemento deste serviço destinado a auxiliar usuários do transporte público. O sistema proposto, desenvolvido para a Web, solicita ao usuário apenas a rua de origem e a rua de destino e fornece uma descrição completa de linhas de ônibus (e em que paradas) que ele deve utilizar para chegar ao seu destino. Além disso, pela integração com o Google Maps API, o sistema oferece ao usuário um mapa de visualização da trajetória prevista pelo sistema. O único sistema Web, do qual se tem conhecimento, que oferece informação semelhante no Brasil é o da EPTC [7][3]. Entretanto, a EPTC oferece um serviço simplificado neste sentido, visto que o site da empresa permite apenas visualizar o itinerário das linhas de ônibus e os seus horários. Assim, delega ao usuário todo o trabalho braçal de montar sua trajetória, pois só existe a informação de qual ônibus ou lotação passa por uma determinada rua.

Este artigo se encontra organizado como segue: na seção 2 é apresentada uma introdução aos algoritmos de busca inteligentes, com destaque ao algoritmo A*, que será aplicado na definição de trajetórias neste trabalho; a seção 3 descreve o Google Maps e como este serviço, utilizado para visualização de trajetórias no presente trabalho, pode ser inserido em outros sistemas web; a seção 4 realiza uma comparação do trabalho proposto com outros trabalhos relacionados; na seção 5 é descrito o sistema proposto e, na seção 6, são apresentados os resultados da avaliação com usuários; finalmente, na seção 7 são descritas algumas considerações finais e sugeridos trabalhos futuros a serem realizados para o aprimoramento do sistema.

2 Algoritmos de Busca da IA

O trabalho proposto utiliza-se de algoritmos de busca inteligente para definição das rotas de ônibus.

Algoritmos de busca são uma das mais poderosas abordagens para a resolução de problemas em inteligência artificial (IA) [8]. Trata-se de mecanismos de resolução de problemas universais que sistematicamente exploram alternativas e encontram uma sequência para a solução do problema proposto. Encontrada a sequência, há de se considerar se esta trata de uma solução eficiente ou não. Outra questão relevante refere-se aos recursos necessários à solução, ou seja, tempo e memória necessários para se chegar ao resultado [8].

Um problema de busca pode ser idealizado por meio da definição dos seguintes elementos [8]:

- um ou mais estados iniciais. Exemplo: o primeiro lance no jogo de xadrez;

- um ou mais estados finais. Exemplo: o xeque-mate;

- um espaço de estados, ou seja, todas as possíveis posições intermediárias entre o estado inicial e o estado final. São todos os lances possíveis entre o estado inicial e o estado final, por qualquer sequência de ações;

- um conjunto de ações que permitem passar de um estado para outro. As ações irão determinar quais são os estados sucessores de um determinado estado. Exemplo: as regras do jogo.

O problema de busca pode ser representado graficamente por meio de uma árvore de busca, definida como um conjunto de nós que representam os estados explorados na busca da solução. A árvore deve possuir um nó inicial (onde a busca inicia) e um nó objetivo (onde a busca termina), os quais são conectados por nós intermediários. O objetivo da busca é encontrar um caminho que ligue o nó inicial ao nó final. Como entrada tem-se a descrição do nó inicial, do objetivo e de um procedimento que conduza os nós aos seus sucessores; como saída tem-se uma sequência de nós que começa no nó inicial e termina no nó objetivo.

[3] Para as cidades de São Paulo e Belo Horizonte, a Google oferece um serviço semelhante, chamado de Google Transit. Maiores detalhes na seção de Trabalhos Relacionados.

Existem dois principais métodos para a resolução de problemas através de busca na IA: busca cega (exaustiva) e busca heurística (informada).

2.1 Busca Cega

A busca cega ou exaustiva não sabe qual é o melhor nó, entre as fronteiras[4] possíveis, que deve ser expandido em busca do menor custo de caminho deste nó até o nó final (objetivo) [8]. Esses algoritmos se baseiam na estrutura do espaço de estados e determinam estratégias sistemáticas para sua exploração, ou seja, seguem uma estratégia fixa no momento de visitar os nós que representam os estados do problema. Trata-se também de algoritmos exaustivos, de modo que podem acabar recorrendo a todos os nós do problema para achar a solução. O custo desses algoritmos pode ser proibitivo na maioria dos problemas reais; portanto, seu uso deve se limitar a problemas pequenos. Existem, basicamente, duas estratégias de busca cega: em largura e em profundidade, as quais possuem derivações.

2.2 Busca Heurística

A busca heurística utiliza conhecimento específico do problema na escolha do próximo nó a ser expandido através da aplicação de uma função de avaliação a cada nó na fronteira do espaço de estados [8]. O A* é o algoritmo de busca heurística que vem sendo mais empregado.

O algoritmo A* tenta minimizar o custo total da solução usando uma função de avaliação para escolher um nó vizinho do espaço de estados de menor valor para ser expandido. Essa função de avaliação visa prever a distância deste nó até o objetivo. No A*, a função de avaliação é definida por $f(n) = g(n) + h(n)$, onde $g(n)$ é a distância real entre o nó origem e o nó candidato à expansão e $h(n)$ (também chamada de função heurística) é um custo estimado do nó candidato até o nó destino. Caso se esteja desenvolvendo um sistema que defina a melhor rota de carro entre duas cidades x e y, $g(n)$ seria a distância real entre x e w (uma cidade qualquer intermediária) e $h(n)$, uma distância estimada entre w e y, por exemplo, a distância euclidiana. Para que A* seja eficiente, $h(n)$ deve ser sempre admissível, ou seja, nunca deve superestimar o custo real. No exemplo dado de rotas entre duas cidades, a distância euclidiana é uma heurística admissível, visto que a distância em linha reta entre duas cidades nunca será maior que a distância real entre estas duas cidades.

2.3 A*: Algoritmo Aplicado no Trabalho

No sistema Web proposto, o algoritmo utilizado para definir as trajetórias foi o A*. Em razão do propósito da aplicação, o algoritmo escolhido deve possuir uma solução completa, pois deve apresentar sempre, pelo menos, uma solução quando existir. A solução encontrada também deve ser ótima (a estratégia sempre encontra a melhor solução quando existem diferentes soluções), pois, caso contrário, irá gerar um custo desnecessário de deslocamento ao usuário. Quanto maior a distância, maior tempo de locomoção gasto e, geralmente, maior número de linhas de ônibus a tomar. Em relação a esses dois critérios, os algoritmos de busca em largura, custo uniforme ou aprofundamento iterativo (que são algoritmos de busca cega) também poderiam ser escolhidos, mas a definição pelo A* ocorreu também em função de tempo e espaço. Por se tratar de um sistema *web*, este deverá ter um tempo de resposta satisfatório; neste ponto, o algoritmo que melhor gerencia os recursos e oferece um bom desempenho é o A*. Originalmente, o algoritmo A* é definido conforme a Figura 1.

Por se tratar de um algoritmo de busca heurística, é necessário que o algoritmo A* tenha algum conhecimento específico sobre o problema (heurística) para poder estimar qual é o melhor nodo da fronteira a ser expandido. Neste caso, para garantir que os melhores caminhos possam ser expandidos na ordem correta, o algoritmo faz uso de uma fila de prioridade (PriorityQueue) para armazenar os nodos que estão ordenados na fila de acordo com o seu custo estimado (a sua distância estimada) até o nó objetivo, como pode ser visto na linha 2 da Figura 1. Na primeira chamada do método, a fila de prioridade possui apenas o nodo inicial (origem) que foi inserido na linha 3. Isso garante que a condição imposta na linha 4 seja verdadeira e, dessa maneira, o laço seja executado. Na linha 5, um nodo é removido da fila de prioridade (sempre o primeiro) para na linha 6 verificar se aquele nodo é o objetivo (destino) ou não. Se o estado do nó for o objetivo desejado, o algoritmo retorna o resultado; caso contrário, continua a execução na linha 7, inserindo na fila de prioridade todos os possíveis

[4] Fronteira é o conjunto de nós sucessores que ainda não foi explorado (ainda não se testou se era o estado solução).

sucessores (nodos que podem ser atingidos pelo nodo atual), de forma ordenada de acordo com a heurística, e volta para o laço da linha 4. Esse processo é repetido até que o objetivo seja alcançado ou a fila de prioridade esteja vazia. Neste último caso, o problema não tem solução, ou seja, não é possível definir um caminho possível dados o estado inicial e o objetivo.

```
1   function A*(Estado inicial): Nodo
2       PriorityQueue(f) fronteira {fila ordenada por f}
3       fronteira.add (new Nodo(inicial))
4       while not fronteira.isEmpty() do
5           Nodo n <- fronteira.remove()
6           if n.getEstado().éMeta() then
7               return n
8           end if
9           fronteira.add(n.sucessores())
10      end while
11  return null
```

Figura 1. Implementação genérica do algoritmo A* [8]

3 Google Maps

O Google Maps é um serviço de pesquisa e imagens de satélite da Terra gratuito na Web, fornecido pela empresa Google. O serviço do Google Maps oferece vários tipos de mapas, desde os mais simples, com nomes de ruas, rodovias, empresas, pontos turísticos, até complexos, animados ou não, com possibilidades de traçar rotas com origem e destino, informações sobre quilometragem, entre outros. Este serviço pode ser utilizado diretamente no endereço http://maps.google.com.br.

Uma outra opção bastante útil disponível no Google Maps é a possibilidade de consultar rotas informando apenas a origem e o destino. A ferramenta oferece um detalhamento tanto visual quanto descritivo das ruas que o usuário terá de percorrer para chegar ao destino. Para fazer uso desta opção basta utilizar o link "*Como chegar"* disponível na tela.

Através de uma API JavaScript criada pela Google é possível embutir esses mapas em páginas Web particulares. Esta API, que oferece uma integração com AJAX, é toda documentada, possui diversos exemplos e está disponível em http://code.google.com/apis/maps/documentation.

Para realizar a inserção dos mapas nas páginas Web, primeiramente deve ser realizada a importação da biblioteca, que nada mais é do que uma referência ao endereço da API do Google. Neste momento também deve ser informada uma chave individual que é vinculada a uma conta do Google.

Cada elemento do mapa na API é uma espécie de classe na orientação a objetos, onde podem ser instanciados, modificados, habilitados ou desabilitados e assim por diante, por meio de construtores, propriedades e métodos. Pode-se, por exemplo, habilitar ou não o zoom nos mapas, habilitar ou não as opções de visualização dos mapas por satélite ou terreno, criar ícones próprios em substituição aos ícones padrões, entre outros.

No sistema proposto, foram utilizadas as opções de traçar trajetórias que ficam em destaque no mapa, caixas de texto que demarcam pontos importantes do percurso, opções de navegação no mapa e zoom, além das opções de visualização por mapa, satélite ou híbrido.

4 Trabalhos Relacionados

O trabalho desenvolvido por Tarik Attar [9] apresenta basicamente uma comparação entre os algoritmos A* e o *Ant Colony Optimisation* com os resultados gerados pela API do Google Maps para problemas envolvendo roteamento. Para isso, foi desenvolvido um sistema que procura investigar algoritmos de rotas dentro do contexto *web*.

Para o desenvolvimento deste sistema Web foram utilizadas tecnologias como a Personal Home Page (PHP), como linguagem de programação; o Mysql [10], como gerenciador de banco de dados, e a Geography Markup Language (GML), como linguagem para apresentação de informação de objetos geográficos básicos.

Embora o trabalho de Attar e o trabalho proposto utilizem ferramentas e tecnologias semelhantes, os objetivos propostos por ambos são diferentes. O presente trabalho faz uso do algoritmo A* para a construção de um sistema Web, com o objetivo de indicar o menor número de ônibus possível a usuários de transporte público, utilizando o Google Maps como ferramenta visual para exibição das trajetórias. Por outro lado, o trabalho realizado por Tarik Attar utiliza o algoritmo A* em conjunto com o algoritmo Ant Colony Optimisation a fim de comparar os resultados obtidos com o Google Maps na definição de rotas de carro. Além disso, a modelagem utilizada no presente trabalho apresenta-se mais complexa em razão da necessidade de considerar ônibus, ruas, paradas e direção dos ônibus. Por fim, Tarik Attar considera como heurística no algoritmo A* apenas a distância linear, ao passo que a presente proposta utiliza também a distância manhattan para fins de comparação.

O Google Transit [11] é o novo sistema Web criado pela empresa Google, que tem por finalidade principal definir rotas através de transporte público. Além disso, permite a localização de paradas de ônibus e a consulta de informações sobre estações de trem e horários. A cobertura do sistema ainda é restrita a alguns países, como Estados Unidos, Canadá, Japão, Taiwan, Austrália, Áustria, França, Itália, Polônia, Rússia e Suíça. No Brasil, o serviço encontra-se em funcionamento apenas para as cidades de São Paulo e Belo Horizonte.

Embora o Google Transit e o trabalho proposto possuam objetivos semelhantes, não foi possível comparar tecnicamente os dois trabalhos, pois não foram encontrados artigos que pudessem fornecer informações técnicas do Google Transit. Em relação aos serviços prestados, o Google Transit possui algumas funcionalidades adicionais, tais como mostrar o valor total que deve ser pago para o trajeto, horários dos transportes, assim como várias opções de trajetos. Um ponto diferencial do trabalho proposto é disponibilizar um módulo administrador para cadastro e gerenciamento de dados.

O site Hotstop.com (http://hotspot.com) oferece um serviço muito semelhante ao oferecido pelo Google Transit. Através deste é possível informar um endereço de origem e um endereço de destino e solicitar ao sistema que exiba a trajetória. O sistema fornece também a possibilidade de exibição da trajetória para ser percorrida de ônibus, de táxi, a pé ou com as três opções ao mesmo tempo. A cobertura deste sistema, entretanto, é restrita a apenas algumas cidades dos Estados Unidos.

Da mesma forma que o Google Transit, não foram encontrados artigos que descrevessem tecnicamente o HotSpot.

5 Sistema Desenvolvido

O sistema desenvolvido é formado por dois principais módulos: o de administrador e o de usuário. O módulo de administrador permite o cadastro e atualização de toda informação necessária para gerenciamento do sistema: ruas, paradas, usuários, controles de acesso e ônibus. Já o módulo do usuário contém a principal funcionalidade do sistema: o sistema de consulta de trajetória de ônibus.

Para realizar a pesquisa de trajetórias é necessário informar o nome da rua de origem e o nome da rua de destino, sendo o número (na rua) de origem e destino opcionais (Figura 2). O usuário deve também informar a heurística a ser utilizada para calcular a distância entre as paradas, que pode ser distância manhattan ou distância euclidiana, conforme será explicado na seção 5.2.1.

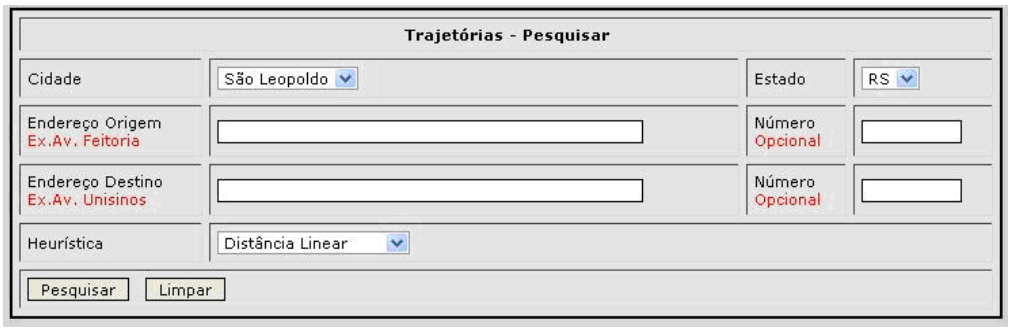

Figura 2. Tela de pesquisa de trajetórias**.**

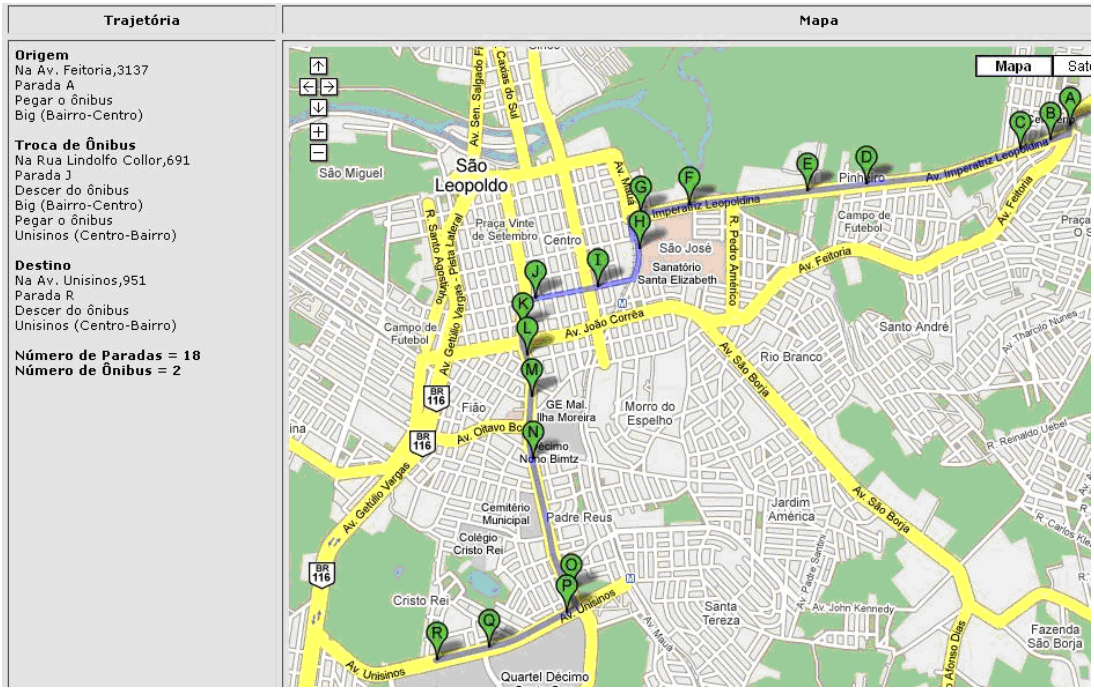

Figura 3. Tela de resultado da pesquisa de trajetórias

O resultado da pesquisa pode ser visto na Figura 3 por meio de uma tabela com duas colunas: Trajetórias e Mapa. A primeira coluna desta tabela, com o título Trajetória, refere-se ao resultado descritivo gerado pelo sistema através do algoritmo de busca A*. Neste caso são gerados sempre dois tópicos: a origem e o destino. Tanto na origem quanto no destino são apresentadas informações sobre a rua, número, parada e o ônibus que o usuário deve utilizar. Um tópico adicional pode ser gerado quando houver necessidade de o usuário efetuar uma troca de ônibus durante o percurso chamado Troca de Ônibus. Neste tópico também são apresentadas informações sobre a rua, número e parada na qual o usuário deve desembarcar de um ônibus e embarcar em outro.

A segunda coluna desta tabela, com o título Mapa, refere-se ao resultado visual gerado pelo sistema através da API do Google Maps. Este mapa contém o trajeto completo que o usuário deve percorrer para ir da origem para o destino.

5.1 Modelagem da Base de Dados

O primeiro passo para desenvolver este sistema foi realizar a modelagem da base de dados. Nesta etapa a preocupação foi com as informações que seriam necessárias armazenar no banco de dados para que o algoritmo

de busca conseguisse definir as trajetórias corretamente. Assim, a modelagem da base de dados em seu estado final pode ser vista na Figura 4.

A tabela Ruas tem apenas um ID (inteiro e sequencial) e um nome e faz um relacionamento de um (1) para (N) com a tabela de Paradas, que, por sua vez, também tem um ID (inteiro e sequencial), coordenadas de latitude e longitude, que definem o ponto exato onde a parada está localizada, e o campo numeroRuaParada, que identifica em que altura (número) esta parada está localizada na rua. Esse relacionamento representa que numa determinada rua podem existir diversas paradas, mas uma parada está localizada em apenas uma rua. A tabela Onibus tem apenas um ID (inteiro e sequencial) e um nome, que é utilizado para identificar o nome da linha, e faz um relacionamento de (N) para (N) com a tabela Paradas. Para representar um relacionamento deste tipo, foi necessário criar a tabela Onibus_Paradas. Este relacionamento representa que um ônibus pode percorrer diversas paradas e, ao mesmo tempo, uma mesma parada pode ser percorrida por diferentes ônibus.

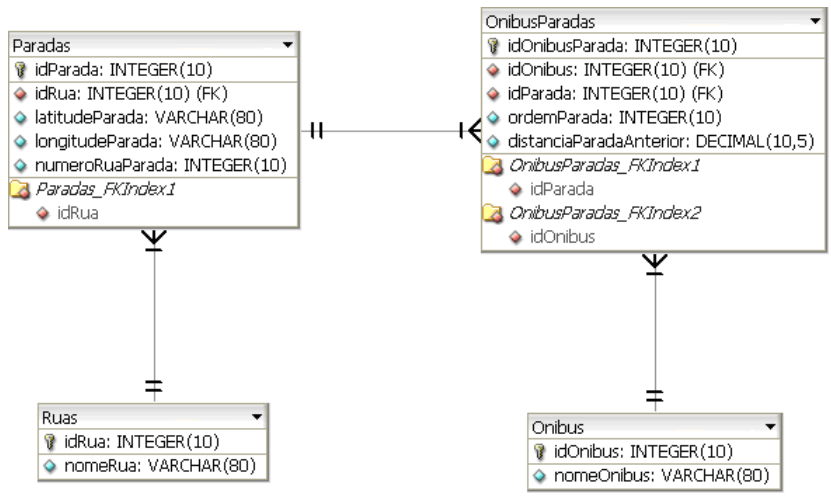

Figura 4. Estado final da modelagem da base de dados

5.2 Modelagem do Sistema

O segundo passo para desenvolver este sistema foi definir a modelagem das entidades através do diagrama de classes disponível na Unified Modeling Language (UML). O diagrama de classes do ponto de vista de especificação composto pelas principais classes e métodos do sistema, pode ser visualizado na Figura 5.

A classe TrajetoriaAction, através do método pesquisar(), inicia o processo de pesquisa de trajetórias no sistema, recebendo todos os dados informados pelo usuário na tela de pesquisa (rua de origem, número de origem, rua de destino, número de destino e o método heurístico). Os campos da tela foram mapeados e inseridos na classe TrajetoriaBean. A classe TrajetoriaBO através do método pesquisar() é responsável por validar todas as regras de negócio (rua de origem não cadastrada, rua de destino não cadastrada, rua de origem não possui parada, rua de destino não possui parada) antes de instanciar a classe Busca, método busca(). Este método cria um objeto do tipo fila de prioridade (*PriorityQueue*) contendo os dados do endereço de origem através da classe Node, que contém informações principalmente sobre o ônibus e a parada em questão. Somente depois de definidas as origens e o destino é que o método Aestrela()[5] inicia efetivamente a definição de trajetórias; para isso, faz uso dos métodos da classe Problema para definir, por exemplo, os estados sucessores de um determinado nodo. O método Aestrela() tem como retorno um objeto da classe Node com toda a solução do problema. Acessando este objeto através de recursão é possível gerar uma lista que contém todos os pontos que compõem a trajetória.

[5] Quando o autor se referenciar ao método de busca implementado, utilizará como nomenclatura o Aestrela, enquanto a referência ao algoritmo original será o A*.

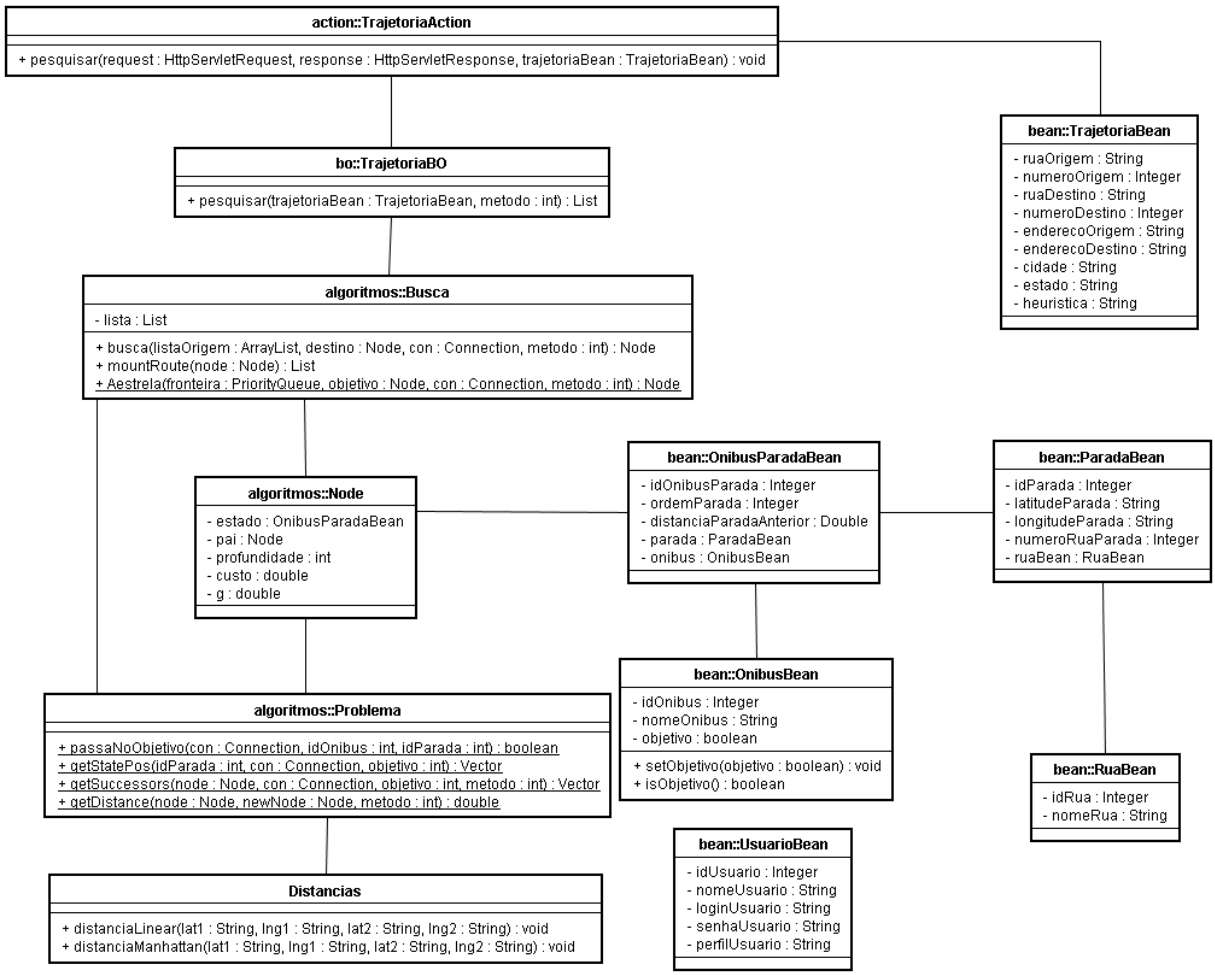

Figura 5. Diagrama de classes do sistema web

5.3 Definindo a Trajetória de Ônibus

Foi necessário definir, inicialmente, os componentes do problema.

- Estado inicial: parada mais próxima na rua e numeração onde o usuário se encontra inicialmente (a origem do trajeto);

- Estado final (objetivo): parada mais próxima na rua e numeração aproximada onde o usuário deseja chegar (o destino do trajeto);

- Ações: se mover para parada próxima em uma linha de ônibus;

- Heurística: distância linear (geometria euclidiana) e a distância manhattan (geometria pombalina).

A busca utiliza tanto a distância linear como a distância manhattan como função heurística entre a parada candidata e a parada destino, de acordo com a opção selecionada pelo usuário na interface do sistema. A distância linear foi escolhida não só por se tratar da menor distância entre dois pontos como pela preocupação do sistema em oferecer ao usuário o menor caminho e o menor número de linhas possíveis. Já a distância manhattan foi escolhida por motivos de comparação de resultados com a distância linear, já que costuma oferecer a menor distância entre dois pontos quando se trata de uma malha urbana. Foram omitidos todos os demais aspectos do mundo real, como trânsito, condições da estrada, tempo, entre outros.

Definidos os componentes do sistema, foi necessário adequar o algoritmo de busca A* para as condições impostas pelo problema.

O algoritmo A*, na sua versão final, implementado neste trabalho pode ser visto na Figura 6 através do método Aestrela(). No texto usaremos Aestrela para se referir à versão modificada para o trabalho do algoritmo original A*.

Antes de invocar o método Aestrela() é necessário criar uma fila de prioridade. A fila de prioridade na versão modificada do A* definida para o trabalho não possui apenas um estado inicial como a definição genérica do algoritmo A*. Ela possui a mesma quantidade de estados iniciais do que a quantidade de ônibus que cruzam a parada de origem. Isso ocorre porque os estados (nodos) neste caso representam a relação de um ônibus com uma parada, ou seja, quantos ônibus diferentes cruzam aquela parada. A definição da primeira parada localizada na rua de origem que será utilizada na busca ocorre pela subtração do número informado pelo usuário na interface e do número de paradas existentes naquela rua. A parada que possuir a menor diferença em relação às demais será a parada escolhida. Neste momento inicial, os estados iniciais (nodos) são inseridos na fila de prioridade, o que garante que o algoritmo não irá recomendar dois ou mais ônibus quando poderia recomendar apenas um, evitando, assim, trocas desnecessárias ao usuário.

```
156 private static Node Aestrela(PriorityQueue<Node> fronteira,
157     Node objetivo, Connection con, int metodo) {
158     Node n = null;
159     Vector<Node> v = null;
160     ArrayList<Node> listaFechada = new ArrayList<Node>();
161 
162     //se fronteira nao esta vazia continua
163     while (!fronteira.isEmpty()) {
164         //retira o cabeca da fronteira
165         n = fronteira.poll();
166         //verifica se o estado retirado eh o objetivo
167         //se for retorna e sai do metodo
168         if (n.getEstado().getIdParada().intValue() ==
169             objetivo.getEstado().getIdParada().intValue())
170             return n;
171         //se a lista fechada ainda nao expandiu aquele
172         // caminho, adiciona na lista e expande
173         // (obtem os sucessores possiveis)
174         if (!listaFechada.contains(n)) {
175             listaFechada.add(n);
176             v = n.getSuccessors(con,objetivo.getEstado().
177                     getIdParada().intValue(),metodo);
178             //adiciona sucessores na fronteira
179             for (int i = 0; i < v.size(); i++)
180                 fronteira.add(v.get(i));
181         }
182     }
183     return null;
184 }
```

Figura 6. Implementação do algoritmo A* para o trabalho

A definição da parada de destino (objetivo) ocorre da mesma forma que a parada de origem, pois ambas são definidas pelo critério de proximidade, isto é, o sistema identifica a parada mais próxima na mesma rua que o usuário informou na interface do sistema. Além da fila de prioridade e do objetivo, o método Aestrela() recebe a informação sobre qual método heurístico utilizar, de acordo com a seleção do usuário na interface do sistema.

Apesar da particularidade imposta pelo problema de possuir diversos estados iniciais, o método Aestrela() propriamente dito obedece à definição genérica do algoritmo A*. Todavia, cabe definir detalhes sobre o método que obtém os sucessores (getSucessors()) de um determinado estado (Figura 7) (ver Figura 7). Este método é responsável por estabelecer quem são os estados possíveis de serem alcançados através do estado atual. Para isso, primeiramente devem ser pesquisados quais são os ônibus que cruzam a parada atual. Para cada ônibus retornado, devem ser buscadas as próximas paradas a partir da parada atual. A próxima parada de um determinado ônibus pode ser obtida pela tabela onibusparadas, coluna ordemParada. A relação de cada ônibus com a sua próxima parada e as informações sobre custo do caminho, profundidade e heurística são armazenadas numa lista (vetor), que representa, assim, os sucessores possíveis do nodo atual. Da mesma forma que na

definição do algoritmo A*, o método getSucessors() obtém os sucessores do estado atual para poder inserir na fronteira. A Figura 7 ilustra a implementação deste método no sistema.

```
170 public static Vector<Node> getSuccessors(Node node,Connection con,
171         int objetivo,int metodo) {
172     Vector<Node> listaSucessores = new Vector<Node>();
173     Node newNode = null;
174     OnibusParadaBean onibusParadaBean = null;
175     Vector<OnibusParadaBean> lista = getStatePos(node.getEstado().
176             getIdParada().intValue(),con,objetivo);
177     for (int i=0; i<lista.size(); i++) {
178          if (node.getPai()!=null && node.getEstado().getIdOnibusParada()
179                  == node.getPai().getEstado().getIdOnibusParada()) continue;
180             onibusParadaBean = lista.get(i);
181             newNode = null;
182             newNode = new Node(onibusParadaBean);
183             newNode.setCusto(node.getCusto()+onibusParadaBean.
184                     getDistanciaParadaAnterior());
185             newNode.setProfundidade(node.getProfundidade()+1);
186             newNode.setPai(node);
187             //acrescendo o custo para mudar de onibus
188             if (node.getEstado().getIdOnibus() == null)
189                 newNode.setG(newNode.getCusto()+
190                         getDistance(node,newNode,metodo));
191             else if (onibusParadaBean.getIdOnibus().intValue() ==
192                 node.getEstado().getIdOnibus().intValue())
193                 newNode.setG(newNode.getCusto()+
194                         getDistance(node,newNode,metodo));
195             else
196                 newNode.setG(newNode.getCusto()+
197                         getDistance(node,newNode,metodo)
198                         +Busca.PENALIDADE);
199             listaSucessores.add(newNode);
200     }
201     return listaSucessores;
202 }
```

Figura 7. Implementação do método getSucessors()

```
8  public class Node implements Comparable<Node> {
9
10     // o estado
11     private OnibusParadaBean estado;
12     // o pai
13     private Node pai;
14     // profundidade da solução
15     private int profundidade = 0;
16     // custo de ter gerado o nodo (acumulado)
17     private double custo = 0;
18     // g = custo do caminho(banco) + heurística (linear ou manhattan)
19     private double g = 0;
20
21     //  ------------------------------------------------------- Construtores
22
23     public Node() { }
24
25     public Node(OnibusParadaBean estado) {
26         this.estado = estado;
27     }
28
29     //  ------------------------------------------------------- Métodos Acessor/Mutator
30
31     public OnibusParadaBean getEstado() {
32         return estado;
33     }
34
35
36     public void setEstado(OnibusParadaBean estado) {
37         this.estado = estado;
38     }
39
40
```

Figura 8. Implementação da classe Node

Por ser completo, o algoritmo A* sempre acha uma solução caso exista e, dessa forma, retorna ao final um nodo (Figura 8), que possui toda a solução do problema. Acessando este nodo através de recursão, é criada uma lista com todo o trajeto que o usuário irá percorrer, com informações detalhadas das paradas e ônibus, desde a origem até o destino.

5.3.1 Fórmulas Utilizadas na Função Heurística

O algoritmo realiza o cálculo da distância de forma automática por meio das informações de latitude e longitude da parada candidata e a parada destino. Para isso, foram criadas duas funções no sistema que calculam a distância entre dois pontos na terra, uma de acordo com a distância linear, ou também chamada euclidiana, e outra de acordo com a distância manhattan. Essas duas formas de cálculo de distâncias entre os dois pontos são implementadas buscando comparar os resultados obtidos, já que em algumas situações, como, por exemplo, em problemas envolvendo rotas, podemos obter grandes distorções de resultado de acordo com a heurística utilizada, que deve ser sempre admissível, ou seja, nunca superestimar o custo real. Ambas as funções recebem como parâmetros duas latitudes e duas longitudes (das paradas) e retornam a distância em quilômetros entre os dois pontos. Os métodos finais relativos ao cálculo das distâncias linear e manhattan podem ser vistos na Figura 9, respectivamente.

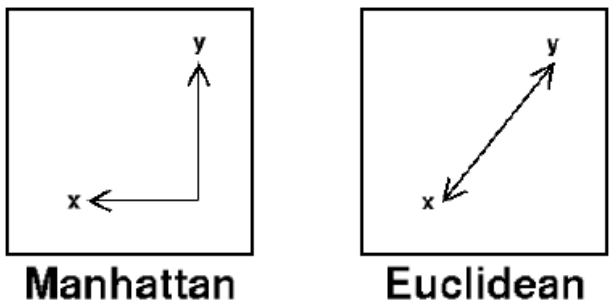

Figura 9. Distância entre dois pontos conforme os dois métodos considerados

O método de cálculo da distância linear baseia-se na trigonometria esférica [12] para definir a distância entre dois pontos na superfície terrestre a partir de suas latitudes e longitudes. Para este cálculo usa-se a fórmula da lei esférica dos cossenos definida em (1), onde *distance* será a distância final em quilômetros (km) entre os dois pontos, *lat1* e *long1* representam a latitude e longitude em radianos do 1º ponto, e *lat2* e *long2* representam a latitude e longitude em radianos do 2º ponto. *R* corresponde ao raio da terra que equivale aproximadamente a 6.371 km e acos, sin e cos representam, respectivamente, as funções trigonométricas arco-cosseno, seno e cosseno.

$$\text{distance} = \text{acos}(\sin(\text{lat1}).\sin(\text{lat2}) + \cos(\text{lat1}).\cos(\text{lat2}).\cos(\text{long2}-\text{long1})).R \quad (1)$$

O método de cálculo da distância manhattan baseia-se na fórmula matemática visualizada em (2).

$$|x_1-x_2|+|y_1-y_2| \quad (2)$$

Para aplicar a fórmula é necessário converter as latitudes e longitudes de graus para quilômetros [11]. As latitudes são convertidas simplesmente multiplicando o seu valor em graus por 111.111 quilômetros (pois 1 grau corresponde a 111.111 quilômetros). As longitudes são convertidas multiplicando-se o seu valor em graus também por 111.111 quilômetros e, além disso, multiplicando este resultado pelo cosseno da latitude (visto que este valor pode sofrer uma variação de 1% para menos quanto mais próximo da linha do Equador, ou para mais quanto mais próximo dos polos em razão do achatamento terrestre). Após a obtenção dos valores das latitudes e longitudes dos pontos em quilômetros, basta aplicar a fórmula em (2) para obtermos a distância manhattan entre

os dois pontos em quilômetros, onde x1 e x2 correspondem às longitudes convertidas para km dos pontos e y1 e y2 representam as suas latitudes convertidas.

5.4 Implementação do Sistema

O sistema Web utiliza o paradigma orientado a objetos através da linguagem de programação Java [13], versão 1.5. Sendo este um sistema Web, é utilizado a Java Server Pages (JSP) em conjunto com o framework Model View Controller (MVC) Struts da Apache [14], versão 1.3.8. Para isso, o servidor de aplicação utilizado é o Tomcat, também da Apache, versão 5.0. O sistema gerenciador de banco de dados é o Mysql na sua versão para o sistema operacional Windows.

6 Avaliação do Sistema

Nesta seção são descritos os experimentos realizados com o objetivo de avaliar aspectos da usabilidade e da corretude dos resultados apresentados pelo sistema Web. Para isso, duas metodologias foram aplicadas: testes com usuários e testes unitários.

Para realização da avaliação utilizou-se uma base de dados sintética com informações reais de seis linhas de ônibus, 19 ruas e 68 paradas da cidade de São Leopoldo, no Rio Grande do Sul. Essas informações foram coletadas de maneira manual pelos próprios autores, ou seja, os percursos dos ônibus foram percorridos de carro para que as informações fossem coletadas.

6.1 Testes com Usuários

A metodologia utilizada visa verificar a usabilidade do sistema. Usabilidade se refere à capacidade de um sistema ser compreendido, aprendido e atrativo ao utilizador [15].

Para a avaliação optou-se pela utilização do sistema por uma amostra de usuários voluntários, constituindo-se desta maneira em uma amostra por conveniência. O perfil da amostra é apresentado na seção 6.1.1.

Os testes com os usuários foram realizados em São Leopoldo (Rio Grande do Sul), nas dependências da Universidade do Vale do Rio dos Sinos, e em Canoas (Rio Grande do Sul), nas dependências de uma empresa da área de transporte em Canoas entre 1 e 24 de outubro de 2008.

Para a realização da avaliação, um e-mail foi enviado aos usuários contendo um documento com as informações sobre a avaliação, forma de acesso ao sistema e perguntas. Para participar da avaliação, o usuário deveria ler as informações no documento, acessar o sistema Web, responder às perguntas e enviar um e-mail com o formulário respondido para o endereço fornecido no documento.

6.1.1 Perfil da Amostra

A amostra utilizada nos testes com usuários constitui-se como não probabilística e com um total de 17 indivíduos. Dos usuários entrevistados 47% habitam São Leopoldo, 29% moram em Porto Alegre, 12% em Canoas, 6% em Esteio e 6% em Montenegro (Figura 10). Os indivíduos que não habitam a cidade de São Leopoldo, a qual foi mapeada para essa avaliação, conhecem a cidade ou por estudar ou por trabalhar nela. Em relação à escolaridade, 41% dos indivíduos possuem superior completo; a mesma proporção de indivíduos está cursando o ensino superior e o restante, 18%, possui ensino médio completo. Ainda, 82% dos entrevistados são do sexo masculino e 18% do sexo feminino.

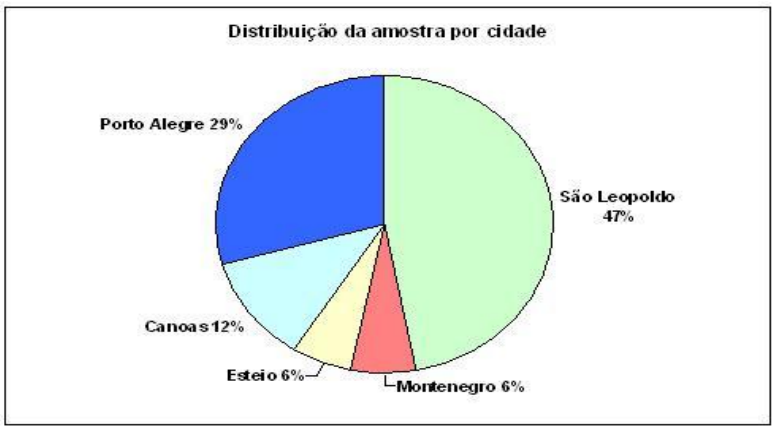

Figura 10. Perfil da amostra por cidade

6.1.2 Resultados

Os participantes da avaliação responderam às seguintes questões relativas ao sistema Web:

Q1) As telas do sistema são intuitivas para o preenchimento da informação necessária? (Sim/Não) Você tem sugestão de melhorias?

Observou-se que um total de 17 participantes respondeu sim à questão, ou seja, 100% dos usuários consideram o sistema intuitivo.

Q2) As informações de saída (mapa com trajeto, nome das ruas, linhas de ônibus, etc.) apresentadas pelo sistema são suficientes? (Sim/Não) Contribua com suas sugestões.

Observou-se que um total de 14 participantes respondeu sim a esta questão, ou seja, 82,35% dos usuários consideram que as informações de saída são suficientes para localização.

Q3) As alternativas de ônibus apresentadas pelo sistema são aquelas que você costuma utilizar? (Sim/Não) Caso afirmativo, elas são melhores ou piores? Contribua com suas sugestões (caso o sistema não contemple o trajeto que você costuma fazer, teste com o trajeto exemplo fornecido anteriormente).

Observou-se que cinco participantes responderam sim à questão, ou seja, 29,41% dos usuários utilizam as linhas que foram apresentadas pelo sistema.

Q4) Você achou o sistema útil e gostou de utilizá-lo? (Sim/Não) Justifique.

Observou-se que um total de 16 participantes respondeu sim a esta questão, ou seja, 94,12% dos usuários gostaram de utilizar o sistema.

Q5) Você utilizaria o sistema se ele estivesse disponível em um site web? (Sim/Não) Justifique.

Observou-se que um total de 16 participantes respondeu sim a esta questão, ou seja, 94,12% dos usuários utilizariam o sistema se estivesse disponível na Web.

Considerando a opinião geral dos participantes desta avaliação, existem evidências da aprovação do sistema por parte dos usuários no que diz respeito à usabilidade, utilidade e relevância das informações apresentadas. Para a maioria dos usuários o sistema proposto seria utilizado. A Tabela 1 sintetiza o resultado final da avaliação com os usuários.

Tabela 1. Resultado final da avaliação com os usuários

Questões	Respostas		
	Sim %	Não %	Total
Q1	100	0	100
Q2	82,35	17,65	100
Q3	29,41	70,59	100
Q4	94,12	5,88	100
Q5	94,12	5,88	100

A Tabela 2 ilustra alguns comentários considerados relevantes realizados pelos usuários durante a execução da avaliação.

Levando em conta que a avaliação fora realizada com a participação de uma amostra selecionada de usuários voluntários e não probabilística, cabe destacar que os resultados obtidos não podem ser generalizados, ou seja, a leitura e interpretação devem ser focadas apenas na amostra.

Tabela 2. Comentários relevantes dos usuários

Nº	Resp.	Comentários
Q1	Sim	"...tela de consulta é bastante direta e intuitiva..."
Q2	Sim	"... as informações são de ótimo aproveitamento..."
	Não	"...pontos marcados (A,B,C) não possuem legenda sendo difícil saber do que se trata. São paradas de ônibus?..."
Q3	Sim	"...uma das alternativas de ônibus que utilizo para ir ao meu trabalho..."
	Não	"...não utilizo ônibus em São Leopoldo..."
Q4	Sim	"...maravilhoso. Pensando grande isto poderia ser de grande ajuda tanto a população que utiliza ônibus diariamente quanto aqueles que desejam pegar um ônibus em um lugar desconhecido."
	Não	"...como não posso avaliar os resultados produzidos não posso afirmar de se usaria ou não..."
Q5	Sim	"...é uma maneira muito fácil de conseguirmos informações sobre lugares desconhecidos, agilizando a locomoção..."
	Não	"...no momento atual não utilizaria pois tenho carro e 95% das vezes que saio de casa utilizo meu veículo..."

6.2 Testes Unitários

Os objetivos propostos com a utilização desta metodologia visam verificar se o sistema apresenta resultados adequados em função das entradas fornecidas pelo usuário. Como o objetivo principal do sistema é traçar rotas, o único módulo submetido aos testes unitários foi o módulo de pesquisa de trajetórias. Por meio desses testes buscou-se também realizar a comparação dos dois critérios heurísticos existentes no sistema: a distância linear e a distância manhattan.

Para realização dos testes optou-se por uma amostra pré-selecionada de oito ruas, que garantiram 14 diferentes entradas no sistema, de maneira que pelo menos um teste fosse realizado considerando os seguintes cenários: Rua de origem não cadastrada; Rua de destino não cadastrada; Rua de origem não possui parada; Rua de destino não possui parada; Trajetória não pode ser definida, pois não há ônibus cadastrados que façam o percurso; Trajetória pode ser definida.

Esses cenários procuraram verificar a consistência do sistema. Observou-se que em todas as situações o sistema apresentou um comportamento correto, ou seja, notificava o usuário que não era possível fornecer a trajetória em virtude da informação insuficiente ou ele fornecia uma trajetória correspondente.

Para a validação do último cenário (Trajetória pode ser definida), oito pesquisas foram realizadas no total, utilizando como heurística a distância linear e a distância manhattan. Não foram encontradas diferenças significativas em relação às heurísticas utilizadas. Observou-se que, em alguns poucos casos, a distância Manhatan leva a um pior desempenho do sistema, ao contrário do que se esperava. Acredita-se que isso se deve ao fato de a malha urbana de São Leopoldo não ter um formato predominantemente quadricular.

Considerando os cenários de teste identificados e também a amostra de ruas utilizada para a execução dos testes unitários, existem evidências do funcionamento adequado do sistema no que diz respeito ao módulo de pesquisa de trajetórias.

Levando em conta que os testes foram realizados com uma amostra de ruas selecionada por conveniência, é importante salientar que os resultados obtidos não podem ser generalizados, ou seja, a leitura e interpretação devem ser focadas apenas na amostra.

Além disso, o fato de os resultados apresentados entre a distância linear e a distância manhattan serem os mesmos também não pode ser conclusivo ou generalizado. Outros experimentos se fazem necessários quando a base de dados for complementada, pois o número de ruas e paradas cadastradas mostrou-se pequeno para chegar a uma conclusão mais precisa.

7 Conclusões e trabalhos futuros

Inspirado no crescimento dos sistemas destinados à Web e também na prestação de serviços, o presente trabalho propôs um sistema Web que pudesse auxiliar efetivamente usuários de transporte coletivo através da consulta e visualização das trajetórias de ônibus.

O sistema Web construído representa a primeira implementação referente ao seu desenvolvimento. Para que seja dado como um produto comercial, alguns aprimoramentos devem ser realizados e algumas limitações devem ser contornadas:

Trajetos percorridos a pé: atualmente o sistema pesquisa trajetos apenas de ônibus, isto é, não estão sendo contemplados pequenos deslocamentos a pé por parte do usuário para ir de uma parada a outra para tomar outro ônibus. Nestas situações o sistema retorna uma mensagem de erro informando que não foi possível definir uma rota.

Limitação de 25 pontos na GOOGLE MAPS API: a limitação de 25 pontos (paradas) imposta pela Google Maps API praticamente inviabiliza a utilização dos mapas para o propósito deste trabalho, pois isto faz com a trajetória visual apresentada esteja limitada a 25 paradas, mesmo que o sistema esteja hoje preparado para recomendar um número bem maior do que este. Uma opção para contornar este problema poderia ser pesquisar e utilizar outra API semelhante, como, por exemplo, a Yahoo Maps API (http://developer.yahoo.com/maps/) ou o Live Search Maps (http://dev.live.com/blogs/devlive/archive/2008/08/06/391.aspx) que porventura não tenham a mesma limitação.

Pesquisa de ruas: a forma de pesquisa e exibição das ruas que estão cadastradas pode ser melhorada de maneira que ofereça ao usuário a possibilidade de saber o nome correto das ruas ou as ruas cadastradas para realização da pesquisa. Atualmente, o sistema acusa erro de rua não cadastrada caso o usuário não informe exatamente o nome da rua.

Tempo de percurso: uma informação bastante útil que poderia ser acrescentada, mas que hoje não existe, é o tempo total aproximado do percurso.

Horários dos ônibus: outra informação que poderia ser bastante útil seria a informação sobre os horários dos ônibus. Caso não seja possível saber exatamente o horário em que os ônibus cruzam as paradas, pelo menos a informação do horário em que os ônibus saem da garagem seria de grande utilidade.

Interface amigável para dispositivos móveis: pretende-se desenvolver uma interface de acesso ao sistema para dispositivos móveis que permita ao usuário acessar o sistema em qualquer lugar.

Estes e outros aprimoramentos serão desenvolvidos em uma dissertação de mestrado que estenderá o sistema proposto.

Agradecimentos

Os autores agradecem ao professor Dr. Leonardo Chiwiacowsky pelo auxílio nos cálculos baseados em trigonometria esférica.

Referências

[1] LEVY, P. **Collective Intelligence: Mankind´s Emerging World in Ciberspace (1st ed.)**. New York: Basic Books, 1997.

[2] TELELISTAS. TeleListas.net: Lista Telefônica - 102 online - Brasil. Disponível em: <http://www.telelistas.net/>. Acesso em: 4 mar. 2008.

[3] CORREIOS. Correios: encomendas, rastreamento, telegramas, cep, cartas, selos, agências e mais. Disponível em: <http://www.correios.com.br/>. Acesso em: 4 mar. 2008.

[4] GOOGLE. Google. Disponível em: <http://www.google.com>. Acesso em: 5 mar. 2008.

[5] GOOGLE MAPS. Google Maps. Disponível em: <http://maps.google.com.br/>. Acesso em: 28 fev. 2008.

[6] GOOGLE MAPS API. Google Maps API - Google Code. Disponível em: <http://code.google.com/apis/maps/>. Acesso em: 1 jul. 2008.

[7] EPTC. EPTC: Empresa Pública de Transporte e Circulação. Disponível em: <http://www2.portoalegre.rs.gov.br/eptc/>. Acesso em: 28 fev 2008.

[8] RUSSEL, S. & NORVIG, P. **Inteligência artificial**: uma abordagem moderna (3rd ed.). Campus, 2004.

[9] ATTAR, T. **An investigation on routing algorithms web-based applications**. Master, Napier University. Disponível em: <http://www.tarikattar.com/blog/wp-content/uploads/2008/05/tarikattardissertation.pdf>. Acesso em: 7 mai. 2009.

[10] MYSQL. Mysql. Disponível em: <http://dev.mysql.com/>. Acesso em: 7 jul. 2008.

[11] GOOGLE. TRANSIT. Google Transit. Google Transit. Disponível em: <http://www.google.com.br/transit>. Acesso em: 7 maio 2009.

[12] AYRES JÚNIOR, F. **Trigonometria**: plana e esférica. São Paulo: McGraw-Hill, 1976.

[13] SUN. Sun Microsystems. Disponível em: <http://www.sun.com>. Acesso em: 27 maio 2008.

[14] APACHE. Apache: Apache Software Foundation. Disponível em: <http://www.apache.org/>. Acesso em: 27 mai. 2008.

[15] NIELSEN, J. **Usability Engineering.** San Francisco: Morgan Kaufmann, 1993.