Detecção Automática de Rotas de Ônibus

Leone Pereira Masiero, Marco A. Casanova, Marcelo T. M. Carvalho

Laboratório de Tecnologia em Computação Gráfica (TeCGraf)
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro – Rio de Janeiro, RJ – Brasil

{leone,casanova,tilio}@tecgraf.puc-rio.br

Abstract. *This paper describes an algorithm for machine learning of routes traveled by an urban bus fleet by analyzing the sequences of points (geo-referenced) sent by devices installed in each bus. The work includes tests with real data and also discusses some practical aspects that can detect the street segments of any errors inherent in the use of GPS.*

Resumo*. Este trabalho descreve um algoritmo para aprendizagem automática de rotas percorridas por uma frota de ônibus urbanos através da análise das seqüências de pontos (geo-referenciados) enviados por dispositivos instalados em cada ônibus. O trabalho inclui testes com dados reais e discute ainda alguns aspectos práticos para detecção de trechos decorrentes dos erros inerentes ao uso de GPS.*

1.Introdução

A correlação de eventos em tempo real [1,6] é uma tecnologia importante no contexto de gerência de redes, detecção de intrusos [2] e processamento de logs gerados por sistemas [3]. No contexto de geoprocessamento dinâmico, Hornsby [5] argumenta que uma possível área de interesse consiste na identificação de rotas percorridas por objetos móveis ou a detecção de eventos de interesse em um grupo de objetos móveis, como um conjunto de navios partindo de um porto devido a uma explosão. O monitoramento de frotas [4] oferece um segundo exemplo de aplicações de geoprocessamento que dependem de processamento de eventos em tempo real.

Este trabalho concentra-se no problema de aprendizagem automática de rotas percorridas por uma frota de ônibus urbanos através da análise das seqüências de pontos (geo-referenciados) enviados por dispositivos instalados em cada ônibus. Este problema é interessante por várias razões. Primeiro, o monitoramento de uma frota de ônibus urbanos naturalmente necessita que as rotas que os ônibus seguem sejam definidas com precisão. Sem isto, é impossível detectar, entre outros pontos, se um ônibus está fora de rota, ou se está trafegando acima do limite de velocidade para o trecho. Porém, a definição manual de cada rota é um processo bastante laborioso, sujeito a variações sazonais, devido a feriados, obras temporárias, entre outros fatores. De fato, a solução adequada para este problema provou ser um fator importante para o sucesso da implementação de uma aplicação de monitoramento de frotas realizada por um dos autores. Por outro lado, pela própria natureza de uma linha urbana de ônibus, os trajetos percorridos sucessivamente pelos ônibus repetem-se dentro de um mesmo período do dia ou dentro de um intervalo de dias específico (como durante o carnaval). Assim, é viável construir um algoritmo que aprenda rotas analisando as

sequências de pontos enviados por dispositivos instalados em cada ônibus e acessando o mapeamento das ruas da cidade coberta pela linha de ônibus.

O algoritmo para aprendizagem de rota apresentado neste trabalho é parte de um esforço mais amplo para construir um sistema de monitoramento de frotas de ônibus, em andamento no Laboratório de Tecnologia em Computação Gráfica (TeCGraf) do Departamento de Informática da Pontifícia Universidade Católica do Rio de Janeiro. O sistema está estruturado em módulos separados para recepção, interpretação, controle, correlação, armazenamento persistente e visualização de seqüências de dados enviados por dispositivos instalados nos ônibus.

Este trabalho está organizado da seguinte forma. A seção 2 apresenta alguns conceitos relacionados à detecção de rotas e a organização do banco de dados. A seção 3 apresenta o algoritmo de detecção de rotas. A seção 4 apresenta os testes realizados. Por fim, a seção 5 contém as conclusões do trabalho.

2. Modelo de Rotas

Um *trecho* é uma partição indivisível de uma rua, de modo que seja vedada a qualquer ônibus a possibilidade de percorrer apenas uma fração de um trecho. Desta forma, trechos só podem ser conectados entre si pelas suas extremidades. Entretanto, os trechos podem apresentar forma totalmente arbitrária, sendo assim representados através de polilinhas. Neste trabalho, convencionamos que um trecho pode ser trafegado em ambos os sentidos.

Um *caminho* é uma seqüência $(S_1,...,S_n)$ de trechos, onde cada trecho está associado a uma restrição temporal (intervalo de tempo). Uma *rota* é uma composição $(P_1,...,P_n)$ de caminhos, onde todos os caminhos da rota devem começar no mesmo trecho S_1 e terminar no mesmo trecho S_n. Todos os caminhos a serem percorridos por veículos de de uma mesma rota devem ter no mínimo um trecho intermediário diferente entre todos eles, como é mostrado na figura 1. Uma rota é *bidirecional* quando pode ser percorrida em ambas as direções; caso contrário é *unidirecional*. Uma rota é *circular* quando o primeiro ponto de S_1 coincide com o último ponto de S_n.

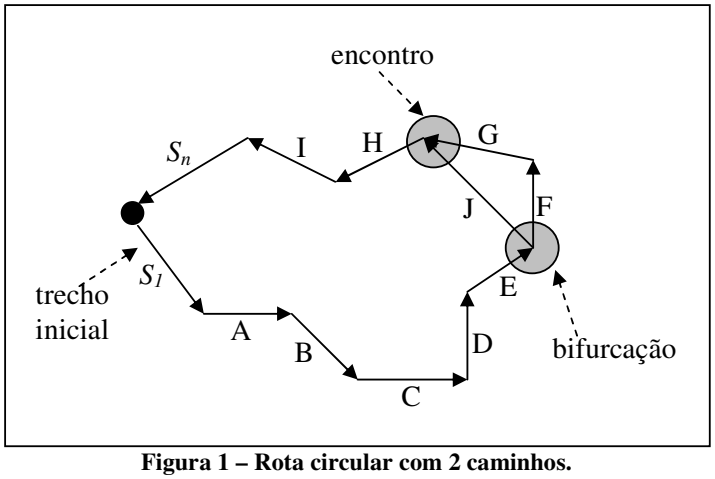

Figura 1 – Rota circular com 2 caminhos.

O modelo do banco de dados deve refletir exatamente estas definições de trecho. Se o mapeamento urbano não considera a noção de trecho, é necessário particionar cada rua em trechos, de forma apropriada. Cada trecho possuirá atributos previamente determinados, como a velocidade máxima permitida. Assim, se há uma escola, por

exemplo, onde a velocidade máxima permitida é diferente do resto da rua, deve-se tomar o cuidado para definir o trecho da escola como um trecho separado.

Dada a representação conceitual de uma rota com as definições de trecho e caminho, a representação computacional de uma rota pode ser definida por um grafo direcionado. Cada vértice do grafo contém um conjunto de trechos subsequentes que não tenham entre si bifurcações nem encontros, e um intervalo de tempo que representa a união dos intervalos de tempo dos trechos que pertencem ao vértice. Por exemplo, o vértice do grafo da figura Figura 2 que contém os trechos F e G terá como intervalo de tempo a união dos intervalos de tempo dos trechos F e G. Se o trecho F possuir um intervalo de tempo entre t_1 e t_2 (onde $t_2 > t_1$) e o trecho G possuir um intervalo de tempo entre t_3 e t_4 (onde $t_4 > t_3 > t_2$), a união desses dois intervalos seria um intervalo de tempo entre t_1 e t_4, que determina o intervalo de tempo do vértice do grafo que contém os dois trechos F e G. As arestas do grafo representam as conexões entre os trechos, como pode ser visto na figura 2.

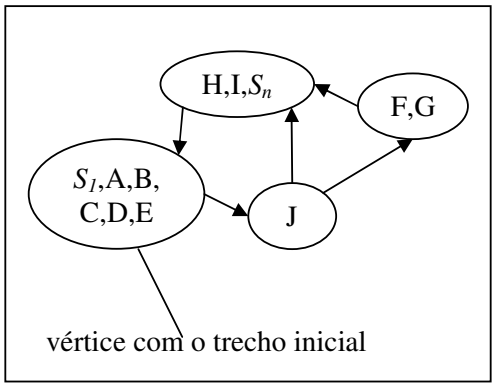

Figura 2 – O grafo como representação computacional da rota da figura 1.

3. Algoritmo para aprendizagem de rotas

O algoritmo para aprendizagem de rotas deve ser capaz de identificar os trechos que compõem a rota e as restrições temporais associadas a cada trecho ou a cada conjunto de trechos. Para detectar uma rota, o algoritmo possui duas fases: na primeira fase, o caminho que o veículo percorreu é detectado, onde cada trecho é adicionado ao caminho com seu intervalo de tempo; na segunda fase, a rota é atualizada com o caminho que foi detectado na fase anterior. No que se segue, consideraremos apenas rotas circulares.

A detecção de um caminho percorrido começa e termina nas seguintes condições. Quando o ônibus sai da garagem, antes que ele chegue à posição inicial de viagem, o ônibus permanece no estado "fora de serviço". Assim que o ônibus chega a sua posição inicial de viagem, o estado do ônibus muda para o estado "em serviço". Assim que o ônibus entra no estado "em serviço", a detecção do caminho deste ônibus é iniciada. Ao chegar novamente à posição de início de viagem, a detecção do caminho termina e começa novamente. Ao final da jornada, o motorista deve indicar que o trabalho foi terminado naquele dia.

A composição de um caminho é feita através de tuplas de segmento e intervalo de tempo permitindo que a mesma instância de segmento possa fazer parte de um outro caminho, possivelmente associado a outro intervalo de tempo. A detecção do caminho é um algoritmo de atualização (figura Figura 3) que a cada sinal GPS (GPSData) coleta o trecho mais indicado em relação ao sinal GPS recebido (linha 3) e atualiza a

composição do caminho (linha 5). A atualização da composição do caminho depende de o trecho coletado já pertencer ou não ao caminho que está sendo detectado. Se o trecho já pertencer ao caminho, apenas o tempo final do intervalo de tempo do trecho é atualizado no caminho (linha 10). Se o trecho ainda não pertencer ao caminho, o trecho é adicionado (linha 12) e seu intervalo de tempo é iniciado com o tempo inicial e final iguais ao tempo do sinal GPS.

```
1  updatePath(Path path, GpsData gpsData) {
2      Segment lastSegment = path.getLastSegment();
3      Segment segment = CollectSegment(gpsData, lastSegment);
4      Time time = gpsData.getTime();
5      path.update(segment, time);
6  }
7
8  updatePathSegment(Segment segment, Time time) {
9      if (this.contain(segment)) {
10         this.updateSegmentTimeInterval(segment, time);
11     } else {
12         this.addSegment(segment, time);
13     }
14 }
```

Figura 3 – Algoritmo de detecção do caminho percorrido por um ônibus.

Ao finalizar a detecção do caminho, o próximo passo é atualizar a rota. O algoritmo de atualização da rota está ilustrado na figura 4. No algoritmo, ao chegar o caminho detectado, se a rota ainda não possuir nenhum caminho, todos os trechos e seus respectivos intervalos temporais do caminho detectado são adicionados em um único vértice e este vértice é adicionado ao grafo que representa a rota (linha 6 até a linha 12). Se a rota já possuir algum caminho, então o algoritmo seleciona cada trecho e o respectivo intervalo de tempo do caminho detectado e verifica se o trecho já foi adicionado na rota (linha 18 a linha 37). Se o trecho ainda não foi adicionado, ele é adicionado com seu intervalo de tempo e cria-se uma aresta entre o último trecho verificado (se existir) e o que acabou de ser adicionado. Se o trecho já foi adicionado, o seu intervalo de tempo é atualizado na rota e verifica se já existe uma aresta entre o último trecho verificado (se existir) e o trecho que acabou de ter seu intervalo de tempo atualizado. Se a aresta não existir, então é adicionada a aresta com o último trecho verificado (se existir) e o último trecho que teve seu intervalo de tempo atualizado. Ao atualizar a rota quando ela já possui um caminho, é necessário expandir os vértices para facilitar o algoritmo. A expansão dos vértices separa cada trecho em um vértice separado mantendo a ordem entre os trechos. Ao finalizar a atualização a rota é colapsada, colocando os trechos sebsequentes sem bifurcação nem encontro dentro de um mesmo vértice.

4. Testes

Para testar o algoritmo para aprendizagem de rotas, foram utilizados dados artificiais criados por um gerador de coordenadas com *timestamp* (para cada coordenada, um *timestamp*). Dado um conjunto de identificadores de trechos de rua e o número de coordenadas por trecho de rua, o gerador retorna um conjunto de coordenadas com *timestamp*.

Para simular a imprecisão do GPS, o gerador de coordenadas com *timestamp* utiliza uma imprecisão opcional que é aplicada às coordenadas geradas. Como pode ser visto na figura Figura 5, as coordenadas são geradas de maneira imprecisa sendo

posicionadas a uma certa distância dos trechos das ruas. A imprecisão pode ser negativa ou positiva, determinando de qual lado do trecho da rua a coordenada será posicionada.

```
updateRoute(Path path) {
    if (path.size() == 0) {
        return;
    }
    if (this.size() == 0) {
        PathObject firstObject = path.getObject(0);
        Vertex vertex = new Vertex();
        for (int i = 0; i < path.size(); i++) {
            PathObject pathObject = path.getObject(i);
            vertex.addPathObject(pathObject);
        }
        this.addVertex(vertex);
    } else {
        this.expand();
        Vertex lastVertex = null;
        for (int i = 0; i < path.size(); i++)
        {
            PathObject pathObject = path.getObject(i);
            Vertex vertex = this.getVertex(pathObject);
            if (vertex != null) {
                vertex.update(pathObject);
                if (lastVertex != null) {
                    if (!this.containEdge(lastVertex, vertex)) {
                        Edge edge = new Edge(lastVertex, vertex);
                        this.addEdge(edge);
                    }
                }
            } else {
                vertex = new Vertex();
                vertex.addPathObject(pathObject);
                this.addVertex(vertex);
                if (lastVertex != null) {
                    Edge edge = new Edge(lastVertex, vertex);
                    this.addEdge(edge);
                }
            }
            lastVertex = vertex;
        }
        this.collapse();
    }
}
```

Figura 4 – Algoritmo de atualização da rota.

A imprecisão do GPS pode causar alguns problemas aos sistemas que realizam processamento com esses dados. A figura 5 ilustra um desses problemas. Um dispositivo GPS em um ônibus que trafega a rua na parte superior da figura, da esquerda para a direita, gera sinais GPS com uma determinada imprecisão até que um ponto acaba ficando, incorretamente, mais próximo da rua na parte inferior da figura. Ao coletar os trechos de rua, armazenados no banco de dados, mais próximos da coordenada do sinal GPS em questão, o trecho de rua mais indicado para ser o escolhido seria o trecho da rua na parte inferior da figura por ser o trecho de rua mais próximo da coordenada atual. Mas, neste caso, o algoritmo de coleta dos trechos leva em consideração o último trecho de rua coletado e o primeiro critério para escolha do trecho é a continuação do trecho anterior. Assim, dos trechos coletados da base de dados mais próximos da posição do ônibus, se algum deles possuir conexão com o

último trecho de rua, este será o trecho de rua escolhido; caso contrário, o trecho de rua mais próximo da posição do ônibus será o escolhido.

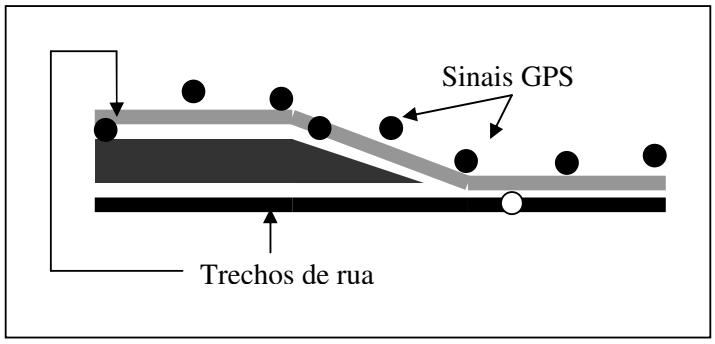

Figura 5 – Visualização da imprecisão das coordenadas geradas pelo gerador de coordenadas.

5. Conclusões

Este trabalho apresentou inicialmente um modelo para rotas. Em seguida, descreveu brevemente um algoritmo para aprendizagem automática de rotas de ônibus. Por fim, discutiu alguns aspectos práticos para detecção de trechos decorrentes dos erros inerentes ao uso de GPS. O trabalho encontra-se em estágio preliminar e precisa ser estendido para detecção de rotas bidirecionais e para acomodar variações sazonais de rotas mais complexas, dependentes da época do ano (carnaval, independência, etc) ou de eventos pontuais (acidentes, obras, etc), entre outros.

Referências

[1] Vaarandi, R., "SEC - a lightweight event correlation tool". IEEE Workshop on IP Operations and Management, pp. 111-115, 2002.

[2] Kruegel, C., Toth, T., Kerer, C., "Decentralized Event Correlation for Intrusion Detection". Proceedings ICISC 2001.

[3] Clemente, R. G., "Uma Arquitetura para Processamento de Eventos de Log em Tempo Real". Dissertação de Mestrado, Departamento de Informática da Pontifícia Universidade Católica do Rio de Janeiro, 2008.

[4] Symes, D. J., "Automatic Vehicle Monitoring: A Tool for Vehicle Fleet Operations". IEEE Transactions on Vehicular Technology, vol. VT-29, No.2, May 1980.

[5] Hornsby, K. S., "Modeling Moving Objects: Future Research Possibilities". Colloquium for Andrew U. Frank's 60th Birthday, 2008.

[6] Jakobson, G., Weissman, M., Brenner, L., Lafond, C., Matheus, C., "GRACE: Building Next Generation Event Correlation Services". Proceedings of the 7th Network Operations and Management Symposium, pp. 701-714, April 2000.