Providing Location-privacy in Opportunity Mobile Social Network

by

Rui Huang

A Thesis submitted to the Faculty of Graduate and Postdoctoral Studies

in partial fulfillment of the requirements for the degree of

**MASTER OF APPLIED SCIENCE**

in Electrical and Computer Engineering

Ottawa-Carleton Institute of Electrical and Computer Engineering

University of Ottawa

Ottawa, Canada

September 3, 2017

© Rui Huang, Ottawa, Canada, 2017

Abstract

Todo

Acknowledgements

Todo

Table of Contents

[Abstract i](#_Toc498097959)

[Acknowledgements ii](#_Toc498097960)

[Table of Contents iii](#_Toc498097961)

[List of Figures iv](#_Toc498097962)

[List of Tables v](#_Toc498097963)

[Chapter 1 Vertex reduce algorithm 1](#_Toc498097964)

[1.1. Ignorable vertex and reserved vertex 1](#_Toc498097965)

[1.2. Vertex reducing 3](#_Toc498097966)

[1.2.1. Remove ignorable vertices 3](#_Toc498097967)

[1.2.2. Tidy reserved vertices connections 3](#_Toc498097968)

[1.2.3. Iterations 4](#_Toc498097969)

[1.3. Vertex assembling 5](#_Toc498097970)

[REFERENCES: 8](#_Toc498097971)

List of Figures

[Figure 5.1 Remove ignorable vertices 4](file:///D:\rayseesea\github\Rui_Thesis\Rui_Huang_thesis_VRA.docx#_Toc498096517)

[Figure 5.2 Tidy reserved connections 5](file:///D:\rayseesea\github\Rui_Thesis\Rui_Huang_thesis_VRA.docx#_Toc498096518)

List of Tables

**未找到图形项目表。**

# 

# Vertex reduce algorithm

Movement of network nodes is a significant factor for the performance of DTNs. To evaluate the performance of different DTN protocols, researchers presented different kinds of movement models, like in [2]. Since shortest path algorithm is an essential and time-cost part of creating many movement models, it is an important topic to optimize the time-cost for simulators calculating the all-pairs shortest path problem in the entire map.

The best known non-negative edge weight undirected map all-pairs shortest path algorithm [3] has a complexity , which is still expensive when *n* is huge. For most real-world city map, there are thousands of points in a single square kilometer. Since a tens square kilometers map is reasonable for a DTN protocol evaluation, a simulator must cut down the time-cost of the calculation of the all-pairs shortest path.

Real-world map developers often use short straight lines to present curves, like in [4], so that a several-meters curves may contain tens of points. It is obviously that we do not need to calculate shortest path for every one of these points. In this paper, we present an Vertex Reduce Algorithm (VRA) to reduce the number of points before the process of the shortest path algorithm. The basic idea is that VRA removes all points whose degrees are less than 3 from the map, while keeps the result of all-pairs shortest path algorithm correct.

The rest of this paper is organized as follows: 5.1 presents some basic lemmas and definitions. The process of VRA is described in 5.2 and 5.3.

## Ignorable vertex and reserved vertex

Vertices in the graph are considered as ignorable and reserved ones. In each iteration of VRA, we remove the ignorable vertex from the graph, while the reserved vertices compose a new graph.

Definition: the vertex whose degree is larger than 2 is the reserved vertex (R).

Definition: the vertex whose degree is smaller than or equal to 2 is called the ignorable vertex (G).

Definition: if two reserved vertices (e.g. ) are connected by a sequence of ignorable vertices (e.g. ), then the sequence from to (i.e., ) is a line-segment (LS).

A reserved vertex can belong to different line-segments, while an ignorable vertex belongs to a unique line-segment. Both the reserved vertex and the ignorable vertex are called vertices (V)

Definition: the shortest route between two vertices inside a line segment is called the inner shortest path (SPI), Let denote the inner shortest path between two vertices (i.e. ) inside a line segment.

Lemma1: If two vertices (i.e. , ; ) are in the same line-segment, the shortest path between them is

Since an all-pairs SPI has a complexity equal to or smaller than and the *n* here is much fewer than the number of points on the entire map, the time-cost for SPI is ignorable comparing to the time-cost of the entire map all-pairs shortest path calculation.

Lemma2: We assume that there are two different line-segments (i.e., and ). We pick a vertex from and a vertex from , then the shortest path between and is

where . Here and are in the same line-segment, so we can use lemma1 to calculate , so does . The is the only part we need to calculate using all-pair shortest path algorithms. We should notice that and could be the same vertex, but it does not make any difference to the lemma.

## Vertex reducing

VRA iteratively removes ignorable vertices from the graph until only reserved vertices remained. In each iteration, we remove all ignorable vertices from the graph but keep the edges. If there are more than one routes between a pair of reserved, we keep the shortest one and remove others.

### Remove ignorable vertices

Since the degree of ignorable vertices is no more than 2, an ignorable vertex has only 0, 1 or 2 neighbours. In the case of 0 or 1 neighbour, we simply delete the ignorable vertex; if it has two neighbours, we connect its two neighbours before it is removed, as shown in Figure 5.1. When we remove an ignorable vertex, the weight of the line which connects its two neighbours is equal to the sum of its recent two lines’ weights (i.e., ). After we remove all ignorable vertices in a line-segment, the two reserved vertices at the ends of the line-segment are connected with a line directly, whose weight is the sum of all the intermediate ones’ (i.e., ).



Figure 5.1 Remove ignorable vertices

### Tidy reserved vertices connections

After we remove all ignorable vertices, all the reserved vertices are connected directly. However, it is possible that there are several connections between a pair of reserved vertices. The shortest route between a pair of neighbour vertices makes other longer ones redundant obviously, so that we remove all routes except the shortest one, as shown in Figure 5.2. We assume that is the shortest line in their three connections, then and are removed.



Figure 5.2 Tidy reserved connections

### Iterations

The whole algorithm is shown in Algorithm 1. The input of the algorithm is the original entire map, which is called . In the *i*th iteration, VRA makes a copy of as . If , the is the output of the previous (i.e., ) iteration. VRA removes all ignorable vertices in the as described in 5.2.1 and remove all unnecessary routes between reserved vertices as described in 5.2.2. In other words, is smaller and smaller as *i* increases, because we always remove ignorable vertices from them. If is equal to , which means that the *i*th iteration makes no modification on the graph, the algorithm ends.

Algorithm 1 vertex reducing

**function** reduce(*Graphi*)

Copy *Graphi* to *Graphi*+1

**for** every ignorable vertex *Gu* in *Graphi*+1 **do** remove(*Gu*) (see 5.2.1)

**for** every pair of reserved vertices **do** remove redundant connections (see 5.2.2)

**end function**

**function** VRA(*Graph*0)

i=0

**while** i=0 or there are any differences between *Graph*i-1 and *Graph*i

**do**

**call** reduce(*Graph*i)

i=i+1

**end**

**end function**

## Vertex assembling

We assume that the vertex reducing process stops at the *i*th iteration. Then the is the input of an all-pairs shortest path algorithm. Let denote the shortest path from to in . We can infer the all-pairs shortest path of based on with lamme1 and lamme2. The algorithm ends when we get the result of . The algorithm is shown in Algorithm 52.

Algorithm 52

function

for all ignorable vertices in as

for all other vertices in as

if and are in the same line-segment then

Use lemma 1 to calculate their shortest path.

else

Use lemma 2 to calculate their shortest path.

endif

end

end

end function

function

while i > 0 do

call

end

end function

We start from the and assemble all intermediate graphs, until we get the result of . For any intermediate , we have already get the all-pairs shortest path result of its reserved vertices in the previous iteration (i.e. ). Then we calculate the shortest path from any ignorable vertices to all others. When the two vertices are in the same line-segment, we use lemma1 to calculate their shortest path. The lemma 1 includes 4 parts: , , and . Since we already get the result of in the previous iteration, we just need to calculate 3 parts, which is easy and has a small scale. If the two vertices are in different line-segments, we use lemma2 to calculate the shortest path. We also already get all the parts from the previous iteration so that we just need to deal with their parts.

**REFERENCES**:

1. A. Lastname1, B. Lastname2, C. Lastname3 and D. Lastname4, “Article name,” in Publication name, vol. 13, issue 2, pp. 223–244, 2017.
2. Working Day Movement Model
3. all pairs shortest path algorithm with expected running time O(n2logn)
4. https://www.openstreetmap.org/