ROUTE EXTRACTION, ROAD NAME DISAMBIGUATION AND EFFICIENT SPATIAL QUERY PROCESSING UNDER LOCATION CONSTRAINTS

A Dissertation in
Computer Science and Engineering
by
Xiao Zhang

© 2012 Xiao Zhang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2012
The dissertation of Xiao Zhang was reviewed and approved* by the following:

Prasenjit Mitra  
Professor of Information Science and Technology  
Thesis Advisor, Chair of Committee

Daniel Kifer  
Professor of Computer Science and Engineering

Robert Collins  
Professor of Computer Science and Engineering

Alan M MacEachren  
Professor of Geography

Raj Acharya  
Department Head and Professor of Computer Science and Engineering

*Signatures are on file in the Graduate School.
Abstract

Geospatial information has drawn more and more attention from both academia and industry nowadays. As web and mobile technology thrives, publishing, sharing and retrieving of geospatial information have become much easier and more common than before. People generate large amount of text contents containing geospatial information every day, such as travel blogs, reviews of interesting places to visit in a new city, human-generated route directions, etc. Such documents are rich in mentions of geo-spatial objects and accounts of movements. Meanwhile, the demand for efficient geo-spatial object retrieval under location-based constraints is growing fast. Digital maps, GPS on mobile devices are products under such increasing demand. However, even if geospatial information sources are plentiful and available, fulfilling the information need remains difficult because: (1) it remains a hard problem to identify and extract different types of geospatial information from text, and (2) building an accurate mapping between the textual mentions of landmarks, places to their unique latitudes and longitudes and recovering them on maps are very challenging due to the ambiguity in text. The absence of good solutions to the above-mentioned problems makes such geospatial information in
text hard to use. In addition, there lacks efficient algorithms for geospatial information retrieval given various location-based constraints. Such retrieval tasks are even more challenging on mobile devices since they suffer from limited storage, battery, computational power and update issues. A system which automatically extracts, disambiguates and geo-tags geospatial information in text and provides efficient solutions to location-based services will greatly satisfy people’s demand of sharing and retrieving of geospatial information. In this dissertation, we introduce our efforts towards this goal. We developed: (1) a solution for automatic route components and destination name extraction from text documents containing human-generated route directions. We explored and designed a variety of machine learning features and utilized sequence labeling machine learning models to identify meaningful route information from text. (2) a novel road name disambiguation algorithm specifically designed to work well in noisy environment, i.e., in the presence of inaccurate data and incomplete gazetteer. Although toponymn (place name) disambiguation has been studied extensively, existing methods fail to solve our problem. We creatively model the problem to be an exact-all-hop shortest path problem on a special type of graph, namely semi-complete directed $k$-partite graph, and developed a computationally-efficient algorithm to solve it. and (3) an efficient algorithm for a particular type of geospatial object retrieval task on mobile devices. The mobile device has the ability to access multiple broadcast channels simultaneously. Under this setting, we solved the transitive nearest neighbor query efficiently, which means to find two objects of different types so that the total traveling distance from the query point to these two objects is minimum. An optimization algorithm was also developed to optimize the performance. We show
these approaches in details and demonstrate their effectiveness in experiments.
# Table of Contents

List of Figures ix

List of Tables x

Chapter 1  
Introduction 1  
1.1 Geospatial Information Extraction from Text Route Directions 3  
1.2 Destination Name Extraction 6  
1.3 Road Name Disambiguation 7  
1.4 Geospatial Object Retrieval under Location Constraints 9  
1.5 Contributions 12  
1.6 Organization 12

Chapter 2  
Route Information Extraction from Web Pages 13  
2.1 Route Components Extraction 13  
2.1.1 Preliminaries 13  
2.1.2 Related Work 15  
2.1.2.1 Labeling Sequential Data 15  
2.1.2.2 Sentence Classification 15  
2.1.2.3 Sentence Extraction from HTML 16  
2.1.3 Delimiting Sentences in HTML 17  
2.1.4 Feature Set 17  
2.1.4.1 Basic Features 17  
2.1.4.2 Surficial Features 18  
2.1.4.3 Visual Features 19
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.4.4 Domain-specific Features</td>
<td>19</td>
</tr>
<tr>
<td>2.1.4.5 Other Features</td>
<td>20</td>
</tr>
<tr>
<td>2.1.5 Experiment Results</td>
<td>20</td>
</tr>
<tr>
<td>2.1.5.1 Building Data Sets</td>
<td>21</td>
</tr>
<tr>
<td>2.1.5.2 Document Classification</td>
<td>21</td>
</tr>
<tr>
<td>2.1.5.3 Sentence Extraction Evaluation</td>
<td>22</td>
</tr>
<tr>
<td>2.1.5.4 Cross Validation Method</td>
<td>23</td>
</tr>
<tr>
<td>2.1.5.5 Sentence Classification</td>
<td>24</td>
</tr>
<tr>
<td>2.1.5.6 System Diagram</td>
<td>26</td>
</tr>
<tr>
<td>2.1.5.7 Discussions and Further Analysis</td>
<td>27</td>
</tr>
<tr>
<td>2.2 Identifying Destination Names</td>
<td>29</td>
</tr>
<tr>
<td>2.2.1 Preliminaries</td>
<td>31</td>
</tr>
<tr>
<td>2.2.2 Methods</td>
<td>32</td>
</tr>
<tr>
<td>2.2.2.1 Destination Name Classification</td>
<td>32</td>
</tr>
<tr>
<td>2.2.2.2 Post-Processing for Improving Destination Name Recognition</td>
<td>33</td>
</tr>
<tr>
<td>2.2.3 Experiment Results</td>
<td>34</td>
</tr>
<tr>
<td>2.2.3.1 Evaluation of External Features</td>
<td>35</td>
</tr>
<tr>
<td>2.2.3.2 Evaluation of Post-Processing</td>
<td>35</td>
</tr>
<tr>
<td>2.2.4 Discussion on Feature Relevance</td>
<td>36</td>
</tr>
<tr>
<td>Chapter 3</td>
<td></td>
</tr>
<tr>
<td>Disambiguating and Geocoding Road Names using Exact-All-Hop Shortest Path Algorithm</td>
<td>38</td>
</tr>
<tr>
<td>3.1 Preliminaries</td>
<td>38</td>
</tr>
<tr>
<td>3.2 Related Work</td>
<td>40</td>
</tr>
<tr>
<td>3.2.1 Geographic Term Disambiguation</td>
<td>40</td>
</tr>
<tr>
<td>3.2.2 Hop-constrained Shortest Path Algorithms</td>
<td>41</td>
</tr>
<tr>
<td>3.2.3 Route Extraction from Text</td>
<td>41</td>
</tr>
<tr>
<td>3.3 Problem Formalization</td>
<td>42</td>
</tr>
<tr>
<td>3.4 Algorithm Description</td>
<td>45</td>
</tr>
<tr>
<td>3.4.1 An Illustrated Example</td>
<td>45</td>
</tr>
<tr>
<td>3.4.2 Notations</td>
<td>45</td>
</tr>
<tr>
<td>3.4.3 Algorithms</td>
<td>47</td>
</tr>
<tr>
<td>3.4.4 Proof of Correctness</td>
<td>49</td>
</tr>
<tr>
<td>3.4.5 Time Complexity Analysis</td>
<td>50</td>
</tr>
<tr>
<td>3.5 Experiments</td>
<td>52</td>
</tr>
<tr>
<td>3.5.1 Data Collection</td>
<td>52</td>
</tr>
<tr>
<td>3.5.2 Pre-Processing</td>
<td>53</td>
</tr>
<tr>
<td>3.5.3 Evaluation Results</td>
<td>53</td>
</tr>
</tbody>
</table>

vii
Chapter 4

Processing Transitive Nearest Neighbor Queries in Multi-channel Access Environment

4.1 Preliminaries and Related Work

4.1.1 System Model for Broadcast Environment

4.1.2 Related Work on Processing Spatial Queries

4.2 Problem Analysis

4.2.1 Answering TNN Query

4.2.2 Deficiencies of Existing Algorithms

4.3 New Algorithms for TNN

4.3.1 Double-NN-Search Algorithm

4.3.2 Hybrid-NN-Search Algorithm

4.3.2.1 Useful Distance Metrics

4.3.2.2 Algorithm Description

4.3.2.3 Updating and Pruning Strategy

4.3.2.4 Adjustments in Implementation

4.4 Optimization of TNN Algorithms

4.4.1 ANN Pruning Condition

4.4.2 Factors Affecting Approximation Quality

4.5 Performance Evaluation

4.5.1 Algorithms with Exact Search

4.5.1.1 Access Time

4.5.1.2 Tune-in Time

4.5.2 Optimization

4.5.2.1 ANN vs. eNN

4.5.2.2 Impact of Sizes of Datasets

4.5.3 Distribution

Chapter 5

Conclusions and Future Work

5.1 Conclusions

5.2 Future Work

Appendix A

Sentence Labeling Example

A.1 An Example of Sentence Classification

Bibliography
List of Figures

1.1 An example of a human-generated driving direction document . . . 5
1.2 An example of transitive nearest neighbor query . . . . . . . . . . . 11

2.1 Experimental Results . . . . . . . . . . . . . . . . . . . . . . . . . 24
2.2 Detailed Analysis of Each Class . . . . . . . . . . . . . . . . . . . 25
2.3 System Architecture of First Model of GeoCAM . . . . . . . . . . . 26
2.4 Effects of External Features on Classification Results . . . . . . . 36
2.5 Effects of Post-Processing on Classification Results . . . . . . . . 37

3.1 Simple methods fail in noisy environment . . . . . . . . . . . . . . 39
3.2 A simple example . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
3.3 graph model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
3.4 A Simple Example of the Algorithm . . . . . . . . . . . . . . . . . . 46
3.5 Paths with the Largest Number of Hops . . . . . . . . . . . . . . . 55
3.6 Paths with the Second Largest Number of Hops . . . . . . . . . . . 56

4.1 System Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
4.2 NN Search on R-tree . . . . . . . . . . . . . . . . . . . . . . . . . . 61
4.3 Linear Access in Broadcast Channel . . . . . . . . . . . . . . . . . . 62
4.4 Search Range Determination . . . . . . . . . . . . . . . . . . . . . . 66
4.5 Distance Metrics . . . . . . . . . . . . . . . . . . . . . . . . . . . . 70
4.6 Updating and Pruning . . . . . . . . . . . . . . . . . . . . . . . . . 73
4.7 Updating and Pruning . . . . . . . . . . . . . . . . . . . . . . . . . 76
4.8 Overlap Area . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 78
4.9 Access Times . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 83
4.10 Access Time Analysis . . . . . . . . . . . . . . . . . . . . . . . . . 84
4.11 Tune-in Time VS. Density . . . . . . . . . . . . . . . . . . . . . . . 85
4.12 Applying ANN Optimization Technique . . . . . . . . . . . . . . . . 86
4.13 Hybrid-NN with ANN . . . . . . . . . . . . . . . . . . . . . . . . . 87
# List of Tables

2.1 HTML tag examples ................................................. 17  
2.2 Sample Regular Expressions to extract domain-specific features .. 20  
2.3 Example of Place Noun Dictionary Entries .......................... 20  
2.4 Sample search key words and number of Docs obtained ............ 21  
2.5 Document classification results .................................... 22  
2.6 Confusion Matrices of Naïve Bayes for Document Classification .. 22  
2.7 Confusion Matrices of MaxEnt for Document Classification ....... 23  
2.8 Sentence extraction results ........................................ 23  
2.9 Evaluation of sentence classification after adjustments ............ 28  
2.10 POI matching examples ........................................... 29  
2.11 Correlation and Mutual Information between feature sets and class labels (sorted in decreasing order) ......................... 37  

3.1 Statistics .................................................................. 53  

4.1 Terminology Definition .............................................. 64  
4.2 Parameter Setting ..................................................... 82  
4.3 Fail Rate .................................................................. 88  

A.1 MEMM model on sentence classification ................................. 93  

x
Chapter 1

Introduction

The development in Web technology has given people more power to share information. There is a significant amount of geospatial information in the contents shared by users. For example, people write travel experiences in blogs, share road construction information and report accidents in status updates, provide directions for businesses and writing reviews for places visited. These user-generated contents contain rich mentions of geospatial objects such as landmarks and places of interest, and movement trajectories such as driving routes and hiking trails. A few web applications allow users to manually tag the shared contents with a location on digital maps (Web photo albums such as Picasa\(^1\) and Flickr\(^2\)) and draw trajectories on maps based on the extracted coordinates from users’ vehicular GPS devices \([1]\). Automatic geo-tagging systems associate GPS devices with a photo or video camera to provide latitude and longitude coordinates with the multimedia contents. However, they do not automatically geo-tag places in textual contents. Geo-referencing systems (such as GeoLocate\(^3\)) process locality strings in natural language and reference the coordinates of the geospatial objects in the strings. However, such systems place strict constraints on the input string and do not perform well on general text containing geospatial information. Besides,

\(^{1}\)http://picasaweb.google.com/home
\(^{2}\)http://www.flickr.com/
\(^{3}\)http://www.museum.tulane.edu/geolocate/
they do not separate geospatial information from non-geospatial information. We need an automatic system that handles general textual contents, extracts geospatial information such as landmarks, place names and movement information. In order to make the extracted information useful, the system must further inference the coordinates of geospatial objects and movement trajectories. One obstacle is the ambiguity of mentions of geospatial objects. One place name, if searched in a geospatial database, may yield more than one geo locations. Such ambiguity exists for city and town names, such as “Springfield”\(^4\), as well as road names, such as “Main Street” and “Second Street”. Place name disambiguation remains a challenging problem. The errors introduced by human authors and inaccurate data in the geo databases make the problem even more difficult. For example, the author of a route description may use a wrong or obsolete road name, searching for this road name may yield a set of wrong roads. The geospatial database may not contain all the roads for a road name. However, once such a system is built, various applications can be developed on top of it, for example, indexing geospatial objects mentioned in text and providing efficient retrieval services, studying users’ geospatial interests, detecting road condition changes and accidents, and etc.

Meanwhile, the demand for geospatial information retrieval is increasing. People rely on digital maps and GPS on mobile devices (smart phones and laptops) to find the route to the place they want to visit. Although these tools are powerful, they do not support queries with complicated location constraints, such as TNN (transitive nearest neighbor queries [2]). Researchers have conducted extensive studies on multi-dimensional index structures for spatial data, e.g. R-tree [3] and STR packing algorithms [4]. Different types of queries with spatial constraints and efficient query processing algorithms have been proposed, such as \(k\)NN [5], \(r\)NN [6], \(g\)NN [7]. Efficient query processing techniques for mobile devices have also been studied as a response to the increasing demand on location based services on mobile devices. Query processing on mobile devices faces multiple challenges: mobile devices do not have enough power sources, storage spaces or processing powers. As the number of mobile device users increase, the shortage of bandwidth is becoming a challenging issue. Processing spatial queries on mobile devices must

\(^4\)see the disambiguation page for “Springfield” on Wikipedia: http://en.wikipedia.org/wiki/Springfield
take into consideration these constraints.

In this dissertation, we introduce our contributions in the area of geospatial information extraction, disambiguation and retrieval. Our first problem aims at extracting landmarks and accounts of movements from text. We studied a data set containing human-generated route directions crawled from the Web and proposed an algorithm to extract different route components, namely destination, origin and route descriptions (or instructions). We evaluated different machine-learning models and analyzed the effect of different sets of features for the extraction task.

Our second problem is the road name ambiguity problem in a noisy environment. Given a sequence of ambiguous road names extracted from a route description and a gazetteer, we use a novel approach to model the problem as a shortest path problem on a graph. By carefully constructing the graph and placing constraints on the shortest path problem, our algorithm successfully solves the disambiguation problem in noisy environments, where the road names can be wrong or obsolete and the gazetteer may have incomplete data. Our third problem involves the task of geospatial object retrieval under location constraints on mobile devices. We considered the multi-channel access broadcast environment and proposed an efficient query processing and an optimization algorithm for a particular type of query: transitive nearest neighbor query.

1.1 Geospatial Information Extraction from Text Route Directions

Human-generated route directions are text descriptions of routes from specified origins to destinations. They contain sequences of road names, landmarks, decision points and actions to take on the decision points in order to travel from the origin to the destination. Such text descriptions are often seen on the direction page of a website of businesses, schools and other organizations. Human-generated route directions have been studied in spatial information science, cognitive psychology, geography and linguistics for understanding human cognition of spatial information [8, 9, 10, 11]. They have also seen potential application in improving the quality of routes generated by automatic navigation systems, such as Google Maps.
and GPS devices [12, 13]. An automatic system to extract, understand text route directions and visualize them on the map, if implemented successfully, could bring tremendous benefits to the ongoing research and future applications. Human-generated route directions are text descriptions of routes from specified origins to destinations. They contain sequences of road names, landmarks, decision points and actions to take on the decision points in order to travel from the origin to the destination. They give detailed step-by-step instructions to travelers to reach a specific physical location from a starting point or region and contain rich expressions of movement patterns, landmarks and decision points. Such text descriptions are often seen on the direction page of a web site of businesses, schools and other organizations. Since 1970s, they have been studied in spatial information science, cognitive psychology, geography and linguistics for understanding human cognition of spatial information [14, 8, 10, 9, 11].

As web technology thrives, a large amount of such documents become available. However, in order to utilize them, we must first extract meaningful route information from text. Human-generated route directions contain useful information about routes, as well as contents irrelevant to finding paths (e.g. advertisement, business hours, etc). Although humans mostly manage to follow these route directions, such manual techniques do not scale to a large corpora of documents. Dealing with real-world corpora requires a scalable information system that automatically detects and extracts route directions in web pages. A challenging task in building such a system is to extract meaningful route components, namely destinations, origins and instructions (or actions) from contents other than route directions.

In route direction web pages, destination refers to the location where the route ends, usually the business, organization or institute hosting the web site, e.g. “Directions to the Campus”. Origin specifies the starting point of the route and helps travelers to choose which set of instructions they should follow in order to arrive to the destination, e.g. “From New York”. Instructions are a set of actions to follow at specified landmarks or decision points such as highways or intersections, e.g. “Merge onto US-220 S toward US-322 ”. In direction web pages, route components are expressed in the form of a complete sentence, an independent phrase or a single word. We will use the term “sentence” to refer to them in the rest of the dissertation. Automatic route part extraction involves classifying
sentences into one of four classes: 1) “destination”, 2) “origin”, 3) “instruction” or 4) information irrelevant to route directions, namely “other”. Figure 1.1 shows an example of a route description from a webpage.

Figure 1.1: An example of a human-generated driving direction document

The first task of route components extraction is to delimit sentences in HTML documents. The difficulty of this task is that HTML authors frequently use HTML structural, positional and visual features, such as columns and table items, as indicators of sentence boundaries and omit traditional sentence boundary indicators, such as punctuation marks, capitalization of the initial word in sentence and abbreviations.

Sunayama et al.[15] proposed a method to utilize HTML tags and period mark to extract sentences from Japanese web pages. However, their approach is not suitable for English language or route components (details will be discussed in Section 2.1.2.3). We propose an alternative algorithm that utilizes both HTML tag information and natural language knowledge to delimit sentences from HTML documents.

After the sentences are extracted from HTML documents, the second task is to classify the sentences into one of the four classes mentioned above. Previous work [16], [17] examined classification models based on independence assumptions such as Naive Bayes [18] and Maximum Entropy [19]. They assume the sentences are independent from each other. However, in our scenario, the route part sentences
display a strong sequential nature. For example, a destination is usually followed by an origin; an origin is usually followed by a set of instructions and instructions are usually grouped together. Based on this observation, we propose to use sequence labeling models such as Conditional Random Fields [20] and MEMMs [21] for sentence classification. These models consider the inter-dependencies between route part sentences and improve classification accuracy.

Our proposed method is implemented in the first module of the GeoCAM system. It classifies an HTML document into two classes: those that contain directions and those that do not. Then it applies our proposed methods to extract route components.

1.2 Destination Name Extraction

Destinations are end points of route directions. Among all route components, destinations are potentially the most critical part. Endpoints are regarded as critical in the conceptualization of events [22] but they are also essential in reasoning about other linguistically encoded information in route directions [23]. Therefore, we focus specifically on extracting destinations.

Previous work on identifying route information from direction documents [9] extracts route components at a sentence level, i.e., the whole sentence containing the destination is extracted. Without knowing what exactly the destinations are, this information will be hard to use for geocoding. Besides, the recognition accuracy for destination sentences is not satisfactory, as reported. In [24], the authors used rule-based method to extract place names and geocode them. The route is then recovered by connecting the geocoded places on the map. However, it does not distinguish destinations from other landmarks and decision points along the routes. The best guess, one can make to find the destination, is to use the last place name as the destination. However, the locations where destinations appear in the text is far less regular than the other route information [25]. The lack of automatic and accurate recognition systems leave the heavy burden of identifying the destinations to human annotators.

Destinations are frequently referred to by their names in the text, for example,

\[^5\text{http://www.geovista.psu.edu/GeoCAM/index.html}\]
“College of Information Science and Technology” and “Saint Paul’s Church”. Destination names are a particular type of named entities. They co-exist with other named entities such as road names, place names and landmarks in the route direction documents. Due to the high accuracy of existing named entity recognition methods and systems (see [26] and [27] for surveys on NER), we focus our study on the unique characteristics of destination names and solve the destination name recognition problem by applying named entity classification techniques. First, we exploit existing named entity recognition systems to extract all named entities from route direction documents. Then, we build a binary classifier to classify the extracted named entities to be either “destinations” or “non-destinations”. After that, we apply a post-processing algorithm, which re-labels un-recognized destinations according to their similarity to recognized names, to improve recognition accuracy. When solving the binary classification task, we explore various feature sets that use general knowledge about what could be a destination, syntactic features, multiple data sources such as domain URL registrants’ names and online phone books, and so forth. Using our approach increases the accuracy with which destinations can be identified automatically.

1.3 Road Name Disambiguation

In addition to information extraction, another required yet challenging task is road name disambiguation. A road is a unique artificial geographic feature on the earth surface. In a gazetteer or geographic database, a road is often represented by a set of connected line segments and/or curves, such as in OpenStreetMap [28]. Each road name, if searched in the gazetteer, can yield more than one road. Throughout this dissertation, we use the term “road” to refer to the unique artificial geographic feature and the term “road name” to refer to the name which is assigned to the road. Ambiguities are often seen in local road names, such as “Main Street” and “Second Street”. Such ambiguities even exist for interstate highways, for example, “Interstate 405” has three disconnected segments on the west coast of the US; one bypass near Seattle, WA, one bypass near Los Angeles, CA and one loop in Portland, OR. Road name disambiguation is to find a unique road (also called

\[\text{http://en.wikipedia.org/wiki/Interstate\_405}\]
“true road”) referred to by the road name in the text. It has become an important and required task for automatic processing and visualizing human-generated route directions.

Road name disambiguation belongs to the scope of toponym (place name) disambiguation, which has been studied extensively. Traditional toponym disambiguation focuses on point or reginal geographic features, such as city names, island names, etc. However, road names have displayed their unique characteristics which makes the disambiguation task challenging. Heuristic rules used in existing work [29, 30] do not work on road names. Population information is often used to disambiguate a city or town’s name, but such information makes no sense for a road. Location qualifiers, such as state names or abbreviations, can be found around a city name. However, in human-generated route directions, road names are often used without a city or state name around, for example, “Atherton St.” is used instead of “Atherton St. PA”; “PA 15” can sometimes be written as “15”. Ontologies of toponyms have been used for disambiguation [31, 32]. Yet, these ontologies are built upon regional features such as cities, states and countries, not for roads. Data-driven methods use external information, such as Wikipedia, about a place name to learn co-occurrences or probabilities of nearby place names [33, 34]. However, it is difficult to find profile pages for all or a majority of roads given a road name. For example, the Wikipedia page for “Main Street” only covers a very limited number of roads with the name “Main Street”7, thus limiting its power to be used for disambiguation.

In addition, human-generated route directions introduces a noisy environment. First, the authors of the directions may use wrong or obsolete road names. Second, the gazetteers are often incomplete. For example, Google Maps allow users to report missing roads. Examples of missing roads can be easily seen in its help forum 8. OpenStreetMaps [28] consists of user-contributed data and is constantly updated. Therefore, given a road name extracted from the text, it is possible that the set of retrieved road in the gazetteer does not contain the true road referred to by the name in the text. The presence of inaccurate data and incomplete gazetteer make the disambiguation problem even more challenging.

7http://en.wikipedia.org/wiki/Main_Street_(disambiguation)
8http://www.google.com/support/forum/p/maps
In this dissertation, we present our achievements in solving the problem of road name disambiguation in human-generated route directions. We model the problem using a semi-complete directed $k$-partite graph (defined formally in Section 3.3), in which each disjoint vertex set corresponds to a road name extracted from text, and each vertex in a set corresponds to one of the roads with the name. Each vertex is connected to all vertices in other sets by a directed edge. The direction of the edge is from the vertex whose road name appeared in front of the other name. The weight of an edge corresponds to the distance between the two closest points of the two roads. Disambiguation problem is converted to a shortest path problem. If no noise exists, the problem can be solved by a simple shortest path algorithm, such as Dijkstra or Bellman-Ford algorithm. In a noisy environment, if a set does not contain the true road or one road name, the algorithm may fail to find other true roads (examples given in 3.3). To tackle the noisy data problem, we allow skips and optimize two criteria, informally, minimizing the path weight and maximizing the number of hops on this path. Given any vertex $v$ and any number of hops $h$, we finds the shortest path starting from $v$ with exactly $h$ hops, namely Exact-All-Hops Shortest Path (EAHSP) problem. Although the general multi-constrained shortest path problem has been proven to be NP-complete [35], given the characteristics of our graph, we developed a polynomial time solution for the EAHSP problem. The time complexity is $O(k^3n^2)$ where $n$ is the number of vertices in each of the $k$ set.

1.4 Geospatial Object Retrieval under Location Constraints

Wireless broadcast has been used widely in various applications (e.g., TV, Radio and GPS). It efficiently uses limited bandwidth to facilitate information dissemination to an arbitrary number of users simultaneously. This feature has attracted a lot of interests and effort from the research community to develop wireless data broadcast techniques, such as [36], [37] etc., in the past decade.

Most prior research on wireless data broadcast assumes that a mobile device can only monitor and receive data from one channel at a time. Given data broadcast in
multiple channels on air, a mobile device has to "switch" among channels in order to receive data from other channels. However, with technological advances, this assumption no longer holds. In the near future (even today), mobile devices (e.g., a portable equipped with multiple wireless radio interfaces or a dual-mode/dual-standby cell phones) will be able to access to multiple channels simultaneously. In this dissertation, we assume a mobile device has the ability to process queries using the information simultaneously received from multiple channels and focus on the query processing of transitive nearest neighbor (TNN) search - a new query type that involves multiple datasets. To the best of our knowledge, this is the first research on query processing over a simultaneous access of multiple broadcast channels.

Here, we first define transitive nearest neighbor (TNN) search:

Given a query point \( p \), and two datasets \( S \) and \( R \), TNN returns a pair of objects \((s, r) \in S \times R\) such that \( \forall (s', r') \in S \times R, (\text{dis}(p, s) + \text{dis}(s, r)) \leq (\text{dis}(p, s') + \text{dis}(s', r')) \)

where \( \text{dis}(p, s) \) represents the Euclidean distance between the two points \( p \) and \( s \).

Figure 1.2 shows an example of transitive nearest neighbors. The two data sets are \( S = \{s_1, s_2, s_3, s_4\} \), and \( R = \{r_1, r_2, r_3, r_4\} \). \( s_2 \) and \( r_3 \) gives the minimum traveling distance from query point to \( s_2 \), then to \( r_3 \).

Many applications of TNN queries exist in our daily life. For example, Mr. Smith is new to a city and he wants to find a post office to send his friends some post cards first and then go to a restaurant to have dinner. TNN gives him a post office and a restaurant with minimal total travel distance.

Two TNN algorithms, namely \textit{Approximate-TNN-Search} algorithm and \textit{Window-Based-TNN-Search} algorithm, have been proposed in the one-channel wireless broadcast environment. However, the \textit{Approximate-TNN-Search} algorithm may fail to answer TNN queries over real datasets, while the \textit{Window-Based-TNN-Search} algorithm was not designed for simultaneous access to multiple channels. Thus, our research goal is to devise efficient algorithms for multiple channel broadcast environments. The TNN algorithms we developed is based on an \textit{estimate-filter} query processing paradigm, which consist of two phases: 1) determine/estimate a search range that contains all qualified data objects; and 2)

\footnote{A dual-mode dual-standby cell phones allows its users to stay on-line and send/receive signals in both GSM and CDMA networks simultaneously.}
filter unqualified objects from this search range. (see Section 3.1 for more details and example) We propose new algorithms, namely, Double-NN-Search algorithm and Hybrid-NN-Search algorithm. They use different strategies in phase 1 and introduce more parallelism than Window-Based-TNN-Search in query processing. These algorithms employ exact nearest neighbor (eNN) search algorithm to obtain a search range (during the estimate phase). Our research shows that instead of eNN search, an approach based on approximate nearest neighbor (ANN) search may reduce the overall energy consumption. We propose a new heuristic to perform approximate nearest neighbor search on R-tree. The heuristic is dynamically optimized based on R-tree heights and sizes of the two datasets involved in the TNN query. We also conduct a comprehensive set of experiments to show the effectiveness of these algorithms and optimization techniques.
1.5 Contributions

We briefly state our contributions in this section. The detailed introduction and comparison to related work will be given in each of the following section.

Although human-generated route directions have been studied for a long time, no existing work has developed an automatic approach to extract useful route information from text. Our work enables automatic and scalable information extraction from human-generated route descriptions.

Toponym (place name) disambiguation has been studied. However, road name disambiguation problem has not caught the attention of researchers. We show that the road name ambiguity exists and the problem is hard to solve given the noisy environment, i.e. the road name provided by the authors may be wrong and the geospatial database may not have complete information. In this dissertation, we show that existing disambiguation algorithms will fail in our situation, then we novally define the problem as an exact-all-hop shortest path problem on a semi-complete directed $k$-partite graph. These concepts will be defined formally in Chapter 3. After the formal problem statement, we propose an efficient algorithm to solve the problem.

Spatial query processing on mobile devices has been studied extensively. However, not many existing works have considered utilizing the multi-channel-access ability of new mobile devices. In this dissertation, we propose efficient algorithm for processing transitive nearest neighbor queries on a mobile device that can access multiple channels simultaneously. We give the details of the algorithms, and an optimization technique.

1.6 Organization

The rest of the dissertation is organized as follows: Chapter 2 introduces our work on route component extraction and destination name identification from human-generated route descriptions. Chapter 3 shows our work on road name disambiguation in noisy environment. Chapter 4 presents our work on processing TNN query in multi-channel environment. Chapter 5 concludes this dissertation and gives future work.
Route Information Extraction from Web Pages

2.1 Route Components Extraction

In this section, we discuss the techniques to extract route components from HTML documents. We first give the preliminaries and related work, then propose our algorithm for sentence delimitation in HTML documents. After that, we introduce our machine-learning based algorithm for sentence classification.

2.1.1 Preliminaries

Human-generated driving direction documents contain the following route components [9]: destinations, origins and route parts (or instructions); in addition, such a document may also contain information irrelevant to driving directions, such as advertisements. We have seen Figure 1.1 in Section 1.1 as an example, with the three route components extracted. A typical direction document contains only one destination and 3 - 5 origins. For each origin, a set of instructions is provided.

We define important components in a route direction document as follows:

**Destination:** The place to which a person travels, usually the name or the address of the place.

**Origin:** The place or region a person comes from, usually a city, an orientation
(such as North and South), or a highway name.

**Instruction:** A set of actions a person should follow in order to reach a specified destination from an origin.

**Other:** Any contents other than the above three route direction parts, such as phone numbers or advertisements.

Route components are carried by complete sentences, phrases or even single words. Given an HTML document containing route directions, the first step of route components extraction is to find the objects to classify - the sentences. Sentence delimitation in HTML is different from the delimitation in plain text. First, HTML authors frequently use HTML structural, positional and visual features to indicate sentence boundaries, instead of punctuation marks. For example, a sentence may be bounded by columns or table. Second, when converting an HTML document into a plain text document, text pieces belonging to different sentences can be concatenated. Thus converting HTML document to plain text and then using existing sentence delimitation tools (e.g. LingPipe\(^1\)) will fail to successfully extract sentences. Moreover, HTML tags, such as `<B>` and `<A>`, break a sentence into pieces. Therefore, using tags to delimit sentences will not be accurate.

The above problems happen because sentence boundary information in directions generated by humans uses both HTML tags and natural language rules inconsistently. Thus an effective sentence delimitation algorithm should take into consideration both the HTML tags and natural language knowledge. We propose such an algorithm in Section 2.1.3 and define the problem below:

**Definition 1 (HTML Sentence Delimitation).** *Given an HTML document, HTML sentence delimitation is to delimit the sentences carrying complete and independent route components information in HTML source code.*

Sentences extracted from HTML will be further assigned route components class labels by the sentence classifier. We define the route components classification problem as follows:

\(^1\)http://alias-i.com/lingpipe
Definition 2 (route components Classification). Given the list of sentences extracted from an HTML document containing route directions, the route components Classification task is to accurately assign each sentence the following class labels: destination, origin, instruction or other.

2.1.2 Related Work

2.1.2.1 Labeling Sequential Data

Labeling sequential data is a task of assigning class labels to sequences of observations. Application of labeling sequential data includes Part of Speech (POS) tagging and entity extraction. Sequential data has two characteristics: 1) statistical dependencies between the objects we want to label, and 2) the set of features contained by the object itself. Unlike traditional classification models that make independence assumptions and only model the features within each object, such as Naïve Bayes [18] and Maximum Entropy, sequence modeling methods exploit the dependence structure among the objects.

Graphical models are a natural choice for labeling sequences. Hidden Markov Models (HMMs) [39] [40], based on a directed graphical model, have been widely used in labeling sequences. HMMs model the joint probability distribution \( p(y, x) \) where \( x \) represents the features of the objects we observed and \( y \) represents the classes or labels of \( x \) we wish to predict. Another approach based on a directed graphical model, Maximum Entropy Markov Models (MEMMs) [21], combines the idea of HMMs and Maximum Entropy (MaxEnt) [19]. Conditional Random Fields (CRFs) [20] are based on an undirected graphical model, thus avoids the label-bias problem [20]. CRFs directly model the conditional distribution \( p(y|x) \). It follows the maximum entropy principle [41] shared by MaxEnt and MEMMs. CRFs have been successfully applied to many applications such as text processing [42] and chemical entity recognition [43].

2.1.2.2 Sentence Classification

Sentence classification has been studied in previous work in different domains. Khoo et al., evaluated various machine learning algorithms in an email-based helpdesk corpus [16]. Zhou et al., studied the multi-document biography summarization
problem based on sentence classification \cite{44}. However, in the two works, the sentences are treated independently from each other. No interdependencies were considered.

Jindal and Liu studied the problem of identifying comparative sentences in text documents \cite{17}. Their proposed approach is a combination of class sequential rule (CSR) and machine learning. CSR is based on sequential pattern mining, which is to find all sequential patterns that satisfy a user-specified minimum support constraint. That makes CSR fundamentally different from our sequential data labeling task.

Hachey and Grover evaluated a wide range of machine learning techniques for the task of predicting the rhetorical status of sentences in a corpus of legal judgements \cite{45}. They examined classifiers making independence assumptions, such as Naïve Bayes and SVM. They also report results of a Maximum Entropy based model for sequence tagging \cite{46}. This approach is similar to the framework of MEMM. However, only one sequence labeling model is evaluated and the features for sentence classification are limited. We identified richer sets of features that are effective for sentence classification.

\subsection{Sentence Extraction from HTML}

Sunayama et al.\cite{15} proposed an approach to extract sentences from HTML documents in order to solve the web page summarization problem. They used block-level tags, link tags \texttt{(<A>)} and period mark to segment text. Then they rearrange the text pieces by putting small pieces together to guarantee the length of a sentence. However, we discovered that the rearrangement in this paper will be disastrous for route components because destinations and origins are usually short independent phrases, thus should not be concatenated to other text. Besides, we assume that the text should be separated by HTML tags except inline-tags and treat \texttt{(<A>)} tags differently. We found that \texttt{(<A>)} tags should be used to concatenate, instead of segmenting, adjacent text pieces. We also considered more natural language knowledge, such as abbreviations and punctuation marks other than period.
2.1.3 Delimiting Sentences in HTML

Based on the observation that sentence boundaries are indicated by natural language knowledge together with visual and structural features introduced by HTML tags, we propose an algorithm which utilizes indicators from both sides in sentence delimitation.

Our approach first converts an HTML document into a DOM tree and then traverses the tree in a depth-first order. The text nodes encountered will be stored in a list of text except for the following two cases: if text is a child node of a tag node in a pre-defined tags-to-skip list (this list contains tag nodes of which the text children will not be visible when the HTML is rendered by the browser), the algorithm skips this text node; if two text nodes are separated by a tag in the pre-defined tags-to-concatenate list (this list contains the tags which do not indicate sentence boundaries), then the two text pieces are concatenated and put in the list of text. Then the algorithm uses natural language knowledge to further segment each text piece into sentences. Table 2.1 gives some examples in the tags-to-concatenate list and the tags-to-skip list, and Algorithm 1 shows the details.

<table>
<thead>
<tr>
<th>Tags to concatenate</th>
<th>Tags to skip</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRONG</td>
<td>SCRIPT</td>
</tr>
<tr>
<td>I</td>
<td>STYLE</td>
</tr>
<tr>
<td>FONT</td>
<td>OBJECT</td>
</tr>
<tr>
<td>EM</td>
<td>OPTION</td>
</tr>
<tr>
<td>B</td>
<td>IMG</td>
</tr>
</tbody>
</table>

Table 2.1: HTML tag examples

2.1.4 Feature Set

Various sets of features have been extracted for machine learning models for the sentence classification task. Our feature sets can be categorized as follows:

2.1.4.1 Basic Features

Basic features refer to the Bag-Of-Words features. Similar to document classification, we use the appearance of terms in each sentence as the first set of features.
Algorithm 1 Sentence Delimitation in HTML

**Input:** An HTML document \(doc\), a tags-to-skip list \(skipList\), a tags-to-concatenate list \(concatList\)

**Output:** A list of sentences \(sList\)

**Procedure:**
1. \(sList \leftarrow \emptyset; tList \leftarrow \emptyset; String\ t \leftarrow \emptyset; flag \leftarrow true\)
2. parse \(doc\) into DOM tree \(dTree\);
3. repeat
4. let \(n\) be the next node to visit during depth-first traversal of \(dTree\);
5. if \(n\) is Text Node then
6. append \(n\)’s text to \(t\);
7. else if \(n\) is Tag Node then
8. if \(n\) is in \(skipList\) then
9. skip the subtree rooted at \(n\);
10. else if \(n\) is in \(concatList\) then
11. if \(flag == false\) then
12. \(t \leftarrow \emptyset\);
13. end if
14. \(flag \leftarrow true\);
15. else
16. \(flag \leftarrow false\); put \(t\) into \(tList\); \(t \leftarrow \emptyset\);
17. end if
18. end if
19. until all nodes in \(dTree\) has been visited or skipped;
20. for each text piece \(t\) in \(tList\) do
21. parse \(t\) into sentences and put them into \(sList\);
22. end for
23. return \(sList\);

After tokenization, the terms are converted to lowercase. However, different from document classification, traditional stopwords in IR play an important role in route components. For example, stopwords like “take”, “onto” and “at” are essential in instructions. Therefore, we also evaluated the effect of traditional IR stopwords in route part classification.

2.1.4.2 Surficial Features

Surficial features refer to the visual features that can be observed directly from the sentence, for example, whether a sentence has only one word, whether a sen-
tence consists of characters other than letters and digits, whether all the words are capitalized. We chose this set of features in order to characterize the route components expressed in single words or phrases. For example, destinations frequently appear as the name of a business or an organization and all the words in the name are capitalized; sentences having no letters or digits in it are frequently labeled as “other”.

2.1.4.3 Visual Features

We extracted a set of HTML visual features such as whether a sentence is a title, a link or a heading, etc. This is based on our observation that HTML authors usually use different visual features for different route components. For example, titles of HTML documents usually contain the destination; destinations and origins are usually in Headings; links in HTML are usually irrelevant to route components.

2.1.4.4 Domain-specific Features

One set of domain-specific features are language patterns. We identified a set of frequent patterns in direction descriptions. Such patterns include highway names and particular verb phrases, such as “turn left to ...”, “merge onto ...” and “proceed ... miles...”. A rule-based approach using string pattern matching is applied to generate this set of features. A set of rules is predefined and carefully examined. Table 2.2 gives a set of sample regular expressions and some examples of language patterns in the text (HighwayPS is a pattern string for matching highway names). We designed 25 regular expressions to extract frequent language patterns for instructions, 2 for destinations, 1 for origin and 2 for other. Note that we tried to make the set of regular expressions as compact as possible. So if two or more phrases or word combinations can be put into one regular expression, we do so to reduce the number of regular expression in the rule set. So the number of matched phrases and word combinations is much larger than the number of regular expression.

Another set of domain-specific features are nouns and noun phrases that can be encoded in a dictionary. We created a dictionary of frequent nouns and noun phrases referring to a place or a location, such as “hotel”, “restaurant”,
“campus”, etc. The dictionary has 110 entries. We build this dictionary based on our observation that the entries are usually the agencies hosting the driving direction web pages and these nouns or noun phrases frequently appear in the destinations. Table 2.3 gives some examples of the dictionary entries.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Regular Expressions</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST 1</td>
<td>.*follow \s{1,5}((?: \d{1,5})? \s{1,5})<em>smile\s?.</em></td>
<td>“follow 3.4 miles”</td>
</tr>
<tr>
<td>INST 2</td>
<td>“.<em>exit \s+(?:at \s+)?” + HighwayPS + “.</em>”</td>
<td>“exit at PA Ruote 23”</td>
</tr>
<tr>
<td>DEST</td>
<td>“\s*(?:driving)?\s*direction</td>
<td>directions</td>
</tr>
</tbody>
</table>

Table 2.2: Sample Regular Expressions to extract domain-specific features

2.1.4.5 Other Features

In addition to the above feature sets, we also included a set of “Window Features”. Window features capture the characteristics of surrounding text of a sentence. Window features are extracted after the surficial and language pattern features are extracted. It checks the existence of one or a set of specified features in the window surrounding the current sentence. For example, whether there is an “INST” feature in the sentence before or after the current sentence; whether the previous and following 2 sentences all have a certain feature, etc.

2.1.5 Experiment Results

In this section, we first describe how we build our data set. Then we evaluate the performance of sentence delimitation and classification algorithms.

<table>
<thead>
<tr>
<th>school</th>
<th>university</th>
<th>department</th>
<th>hotel</th>
</tr>
</thead>
<tbody>
<tr>
<td>campus</td>
<td>center</td>
<td>inn</td>
<td>resort</td>
</tr>
<tr>
<td>lab</td>
<td>headquarter</td>
<td>park</td>
<td>pavilion</td>
</tr>
<tr>
<td>division</td>
<td>complex</td>
<td>fair</td>
<td>theatre</td>
</tr>
<tr>
<td>vineyard</td>
<td>orphanage</td>
<td>studio</td>
<td>stadium</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 2.3: Example of Place Noun Dictionary Entries
2.1.5.1 Building Data Sets

A set of over 11,000 web pages containing route directions were identified using the search results of the Yahoo! search engine. The search engine was queried with a set of carefully selected keywords such as “direction, turn, mile, go”, “turn, mile, follow, take, exit” etc. since they are typically present within documents containing route directions. Manual examination shows 96% of these documents contain route directions. A randomly selected subset of 10,000 web pages from the random sampling of the web using the method proposed by M. R. Henzinger, et al. [66] is used as the negative examples. Table 2.4 shows some examples of search queries and number of unique documents obtained from the returned result pages.

2.1.5.2 Document Classification

The document classification task is to separate HTML documents containing route directions from those not containing route directions. We build our document classification models based on “Bag-Of-Words” assumption in which the word order is ignored. HTML tags and scripts are removed before training the classifiers. Tokens are converted to lowercase before classification. We noticed that many characteristic words in route directions are stop-words, such as “take”, “right” and “onto”. We compared the results of both stop-word removed and stop-word
Table 2.5: Document classification results

<table>
<thead>
<tr>
<th>Round</th>
<th>Naïve Bayes w/o S.W.</th>
<th>Naïve Bayes w/ S.W.</th>
<th>MaxEnt w/o S.W.</th>
<th>MaxEnt w/ S.W.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>95.50%</td>
<td>94.99%</td>
<td>98.00%</td>
<td>97.20%</td>
</tr>
<tr>
<td>2</td>
<td>95.70%</td>
<td>94.89%</td>
<td>98.50%</td>
<td>97.60%</td>
</tr>
<tr>
<td>3</td>
<td>96.10%</td>
<td>94.99%</td>
<td>98.30%</td>
<td>97.80%</td>
</tr>
<tr>
<td>4</td>
<td>96.90%</td>
<td>94.99%</td>
<td>98.00%</td>
<td>97.80%</td>
</tr>
<tr>
<td>5</td>
<td>95.50%</td>
<td>94.89%</td>
<td>97.90%</td>
<td>97.80%</td>
</tr>
</tbody>
</table>

Table 2.6: Confusion Matrices of Naïve Bayes for Document Classification

<table>
<thead>
<tr>
<th>Actual</th>
<th>Predicted</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Predicted</td>
<td></td>
</tr>
<tr>
<td></td>
<td>direction</td>
<td>not direction</td>
</tr>
<tr>
<td>direction</td>
<td>139</td>
<td>0</td>
</tr>
<tr>
<td>not direction</td>
<td>42</td>
<td>88</td>
</tr>
</tbody>
</table>

preserved scenarios. We trained and evaluated Naïve Bayes classifier and Maximum Entropy classifier.

Table 2.5 shows the classification accuracies of the two classifiers. Each round is based on 1,000 random selected documents in our data set, with half “Direction” documents and half “Non-direction” documents. The accuracy in each round is an averaged result of the testing accuracies in 10-fold cross validation. It shows that classification with stop-word preserved has a slight decrease in accuracy than stop-word removed. Maximum Entropy with stop-word removed always gives the best performance (the bold font numbers). Table 2.6 and 2.7 give the confusion matrices of Naïve Bayes and Maximum Entropy classifiers on document classification in one training-testing cycle. It shows that although Naive Bayes classifier is good at predicting true positives, the overall accuracy is low. The performance analysis of the two classifiers in other training-testing cycles showed similar results. We use Maximum Entropy in our GeoCAM system. This result is consistent with the result reported by Nigam, et al. [67].

2.1.5.3 Sentence Extraction Evaluation

We compare the effectiveness of our proposed hybrid sentence delimitation algorithm (HYD) with two other approaches: the plain-text-based (PTB) method,
<table>
<thead>
<tr>
<th>Actual</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>direction</td>
</tr>
<tr>
<td>direction</td>
<td>131</td>
</tr>
<tr>
<td>not direction</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 2.7: Confusion Matrices of MaxEnt for Document Classification

<table>
<thead>
<tr>
<th></th>
<th>HYD</th>
<th>PTB</th>
<th>HTB</th>
</tr>
</thead>
<tbody>
<tr>
<td>correctly-extracted</td>
<td>391</td>
<td>152</td>
<td>226</td>
</tr>
<tr>
<td>over-segmented</td>
<td>7</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>under-segmented</td>
<td>5</td>
<td>246</td>
<td>145</td>
</tr>
<tr>
<td>accuracy</td>
<td>97.02%</td>
<td>37.72%</td>
<td>56.08%</td>
</tr>
</tbody>
</table>

Table 2.8: Sentence extraction results

which converts an HTML into plain text format and then used natural language knowledge to segment sentences, and the HTML-tag-based (HTB) method, which parses an HTML document into a DOM tree and extracts the text nodes as sentences.

The three algorithms are applied to the same set of HTML documents containing 403 human-identified sentences. For each algorithm, we counted the number of sentences of three types: 1) correctly extracted, 2) over-segmented sentences and 3) under-segmented. Correctly extracted sentences are the human-identified sentence. If one human-identified sentence is broken into several pieces by the algorithm, we count one over-segmented sentence. If \( n \) human-identified sentences are concatenated together by the algorithm, we count \( n \) under-segmented sentences. Table 2.8 shows the details.

2.1.5.4 Cross Validation Method

The traditional way of doing \( k \)-fold cross validation is to shuffle the data set and divide it into \( k \) equal-sized groups. In order to explore the effectiveness of models which consider the dependencies between sentences, the ordering between sentences in one document should be preserved. Therefore, we shuffle the order of documents, instead of the sentences, so that the ordering of sentences within each document can be preserved. Then the documents are divided into \( k \) equal-sized groups. In each training-testing cycle, one group is used as testing set and the remaining \( k - 1 \)
2.1.5.5 Sentence Classification

We evaluated four models: Naive Bayes (NB), Maximum Entropy (MaxEnt), CRF and MEMM. For CRF and MEMM, we changed the value of initial Gaussian variance to be 1.0, 5.0 and 0.5. In order to evaluate the impact of different feature sets, we divided the features into 5 groups: Bag-Of-Words(B), Language Patterns and surficial features matched by regular expressions(R), Window features(W), HTML visual features(H) and Dictionary(D). We add the features one by one, i.e. B, BR, BRW, BRWH and BRWHD. Besides, we tested the performance of these features without traditional IR stopwords. So each model is applied on 10 different feature sets. The 10-fold cross validation technique described above is applied on 100 HTML documents containing over 10,000 human tagged sentences. A total of 9,880 sets of experiments were conducted (Window feature is not used for NB and
MaxEnt). Due to space limits, only part of the experimental results are shown.

Figure 2.1a shows the sentence classification accuracy of different models on the full feature set (BRWHD) with stopwords. CRFs and MEMMs outperform NB and MaxEnt. After a manual examination, we found the reason is that some sentences which do not have a strong feature of a route part can be inferred by the states of adjacent sentences by CRFs and MEMMs, but are hard for NB and MaxEnt to recognize. For example, an instruction "east 7.4 mi" was not recognized by NB or MaxEnt, but was recognized correctly by CRFs and MEMMs because its previous and following sentences are both instructions.

The effects of different feature sets are shown in Figure 2.1b. We start from Bag-Of-Words (B) features only, then we add in language patterns, denoted by BH; then window features and so on. As more features are added, the performance steadily improve. We notice that among all models, language patterns give the largest improvement. Figure 2.1c shows the importance of using traditional IR stopwords in sentence classification. Stopwords give a significant improvement in route part sentence classification because most route components, especially
instructions contain many stopwords and these stopwords are characteristic. This
confirms that the concept of stopword is domain dependent. Figure 2.2 shows the
precision, recall and F1 score of each model for each class.

As can be noticed in Figure 2.2, although the classification accuracies for In-
stuction, Other and Origin are high and reasonable, the recognition of destination
is a hard problem for all the four models. This is because: 1) the position at which
a destination appears in the text is less regular compared to the other 3 classes; 2)
there lack a set of features that best characterize destinations. Although we iden-
tified some language patterns for destinations, they are frequently described in
only business names which don’t have very obvious language patterns; 3) destina-
tions are usually very short, thus making bag-of-words features perform poorly in
the recognition. A potential solution is to use geography databases to search for
business names that match the business name in the text.

2.1.5.6 System Diagram

![System Architecture of First Model of GeoCAM](image)

The research reported in this paper is part of our GeoCAM project. The first
module of the system allows users to upload an HTML document to the server.
Then the system classifies the document as either “Direction” or “Non-direction”
using a trained Maximum Entropy classifier. The system then extracts a list of sen-
tences from the HTML and feeds them into the learned MEMM sentence classifier.
The classifier assigns one of the following labels to each sentence: “Destination”,
“Origin”, “Instruction” and “Other”. Based on the classification result, the sen-
tences in the HTML document are highlighted with different colors. Figure 2.3 shows the architecture of the first module.

2.1.5.7 Discussions and Further Analysis

In this section, we discuss the experiments we conducted and approaches we explored to improve the performance of sentence classifiers, including (1) using documents which contain only one destination in them and (2) using geographical databases to generate a list of place names or points of interests (POIs), such as “Perry High School” and “LaGuardia Airport” in a bounding box covering the destination. “Place names” and “POIs” will be used interchangeably in the rest of this proposal.

The first approach we explored was to clean the data set. In the original set of 100 documents we used to train and evaluate the sentence classifiers, there were about 20 documents which deteriorated the performance greatly. Typical driving directions contain one or a few destinations and for each destination, a set of origins are associated. However, these documents contain a list of destinations and a simple driving direction is provided for each destination. For example, one document has a list of school sports fields and for each sport field a simple driving direction was associated. The destinations are names of sports fields or school names. The origin for each route direction is indicated by a highway name. Sentences in such documents are very hard to classify. Therefore we removed such documents and replaced them with documents which have only one destination in them.

We also changed the way we label “arrival information”. “Arrival information” is a type of sentences at the end of each set of instructions to help users to locate the destination, such as “Our office is on your right.” or “The gym is at the end of the road.” This type of sentences were labeled as part of the instructions. Based on our observation, these sentences usually contain the name or partial name of the destination. Thus, we tried to label these sentence as destinations and compared with labeling them as instructions.

Table 2.9 shows the sentence classification results before and after the adjustments we mentioned above. We compared the precision, recall and F1 score of the classification results for destination, origin, instructions and other. For each metric
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>precision</th>
<th>recall</th>
<th>F1 score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dest.</td>
<td>arr. as dest.</td>
<td>0.747261537</td>
<td>0.459254381</td>
<td>0.555731761</td>
</tr>
<tr>
<td></td>
<td>arr. as inst.</td>
<td>0.828676324</td>
<td>0.389285315</td>
<td>0.511855782</td>
</tr>
<tr>
<td></td>
<td>old</td>
<td>0.74261</td>
<td>0.35338</td>
<td>0.46138</td>
</tr>
<tr>
<td>Orig.</td>
<td>arr. as dest.</td>
<td>0.825049731</td>
<td>0.777841324</td>
<td>0.797495349</td>
</tr>
<tr>
<td></td>
<td>arr. as inst.</td>
<td>0.81825576</td>
<td>0.776412821</td>
<td>0.793219919</td>
</tr>
<tr>
<td></td>
<td>old</td>
<td>0.82384</td>
<td>0.72294</td>
<td>0.7625</td>
</tr>
<tr>
<td>Inst.</td>
<td>arr. as dest.</td>
<td>0.89126817</td>
<td>0.93158355</td>
<td>0.910149436</td>
</tr>
<tr>
<td></td>
<td>arr. as inst.</td>
<td>0.920628055</td>
<td>0.952078457</td>
<td>0.935687637</td>
</tr>
<tr>
<td></td>
<td>old</td>
<td>0.9128</td>
<td>0.95307</td>
<td>0.93197</td>
</tr>
<tr>
<td>Other</td>
<td>arr. as dest.</td>
<td>0.817002533</td>
<td>0.869544129</td>
<td>0.839843367</td>
</tr>
<tr>
<td></td>
<td>arr. as inst.</td>
<td>0.838055456</td>
<td>0.889239715</td>
<td>0.86056935</td>
</tr>
<tr>
<td></td>
<td>old</td>
<td>0.80417</td>
<td>0.87718</td>
<td>0.83647</td>
</tr>
</tbody>
</table>

Table 2.9: Evaluation of sentence classification after adjustments

and each class label, we compared three different settings. “arr. as dest.” means we label arrival information as destinations. Similarly, “arr. as inst.” means we label arrival information as instructions. In the table, “old” means the old setting in which we have the documents containing long lists of destinations. The results under these two settings were from the MEMM model. The result shows that removing the untypical documents from the training set indeed helped to improve the performance.

As we have analyzed before, the recognition of destination is a difficult problem, especially when the destination is only a place name. One method to tackle this problem is to use geography dictionaries. We conducted some explorative work towards using geography dictionaries to improve the recognition of destinations. We used GeoNames database [68]. It contains names of geographical objects, such as administrative regions, mountains, valleys, islands, parks, churches and etc. Given a document, after we performed sentence classification, the next module generates a bounding box around the destination based on the landmarks we identified in the document. Then the module queries the GeoNames Database and retrieve all the place names inside of the bounding box. We tried to use the list of POIs to improve the recall of destinations.

We manually examined 37 documents with 166 mentions of destinations. In all destinations, there are 119 destinations we cannot find a match or a suspicious
match in the list of POIs. For the 47 destinations, we found either an exact match, or a set of POI entries that are similar. Table 2.10 gives some examples of exact matches and potential matches. We found a total of 12 exact matches. For the rest of the destinations mentioned in the documents, we can only find similar POI entries. We observed that people occasionally use abbreviations or incomplete names. Sometimes typos can also cause the problem. In Chapter 5, we propose different ways to measure the similarity between the human generated destinations and the POI entries in the list. Then use machine learning methods to learn the weights of each similarity and make decisions.

### 2.2 Identifying Destination Names

We first define the following important terms:

**Definition 1** (Destination). A *destination in a route direction document is a physical location where the route ends.*
Definition 2 (Destination Name). A destination name in a route direction document is a named entity referring to the destination of the routes, usually an organization, company or office. It can also be the building or place name where the destination resides.

More than one name can be used to refer to the same destination in the document. For example, "Emory University Campus" is referred to as "Emory University Campus", "Emory Campus" and "Emory" in the same document. These are called "variations" of the destination name. Abbreviations or nouns or noun phrases of the type of the destination can also be used. For example, "Peters Township High School" is referred to as "PTHS" and "the school". The name of the destination, its variations, abbreviations and type nouns, are all considered to be destination names.

Problem Statement 1. Given a human-generated driving direction document, our task is to identify all destination entities: the proper names, variations, abbreviations and the type nouns referring to the destination.

Destination entities are organization or location named entities. However, only recognizing all organization and location named entities does not suffice to solve our problem because named entities other than destinations frequently appear as landmarks. For example, in "...go about one-third mile to the first traffic light, by the Home Depot.", "Home Depot" is a landmark to help travelers locate themselves and make decisions. Therefore separating the destination names from other named entities is an important task.

We solve the destination name recognition problem in two steps. First, named entities are extracted from a document. Second, each extracted named entity is classified as either "destination" or "non-destination". The choice of the distinguishing features is a critical design step and requires comprehensive study of the problem domain. Therefore, we put our effort on finding good features for recognizing destinations. As will be shown later, we identified and evaluated a set of useful features for our classification task. In addition, we analyzed the obtained experiment results. Some destination names were not recognized by the classification model. However, using the recognized destination names from the classification results, we are able to recognize some mis-classified destination names. For example,
if “Emory University campus” is classified as “destination”, then “Emory campus” is highly likely to be a “destination” because the two names are similar. Cosine Similarity captures this kind of similarity. A rule-based post-processing algorithm was developed to improve the classification accuracy. Section 2.2.2.2 gives the details.

2.2.1 Preliminaries

Among the various information present in the driving direction document, we are particularly interested in one type of named entities - destination names (defined in Section 2.2). Destination names can be presented in quite different ways: some are in enlarged fonts, highlighted with bright colors, with strong semantic clues such as “Driving directions to ...”. Some are buried in a large paragraph of text with many other landmarks and place names where no obvious visual features make them stand out from the other named entities. Some do not even have their full names explicitly mentioned and are only referred to by their types, such as “the school”. The destination names can appear multiple times and in multiple parts of the document, such as the title, the beginning of the document body, the beginning of a paragraph, a route instruction sentence, navigational link, or the footnote of the page. Various ways of presenting the destination names increase the difficulty of automatic recognition of a destination in a document.

Destination names often can be found in two types of sentences: (1) destination sentences [9] and (2) arrival information. Destination sentences explicitly specify the destination by mentioning its name, for example, “Direction to ISE Department of Rutgers” and “Visiting CASE”. Arrival information sentences are often one of the last few sentences in a set of route instructions. The destination names or the type of place, such as “the Center”, “the Church”, are often mentioned. For example, “The Center is located 0.4 miles on the left”. Obvious language patterns can be found to help us identify the destination names, such as “Directions to ...” and “... is located ...”, etc.

The destination is often the main subject of the web site where the document resides. This leads to the following three important observations: (1) the provider of the driving directions is often the owner of the web site and thus the registrant
of the web site’s domain URL. (2) The destination name, may also appear in the body and/or title of other web pages of the same site. (3) A phone number, if present, often belongs to the destination company or organization.

In Section 2.2.2, we will see that the above observations and analysis result in important features for finding destination names.

2.2.2 Methods

We solve the problem defined above in three steps: first, extract named entities from the document, using the algorithm described in previous work [9] and OpenCalais [69]; second, classify each extracted named entity to be either “destination” or “non-destination”; and finally, based on the classification results, re-classify a named entity according to its cosine similarity to a classified destination, as a post-processing step.

2.2.2.1 Destination Name Classification

We built a binary classifier and explored extensively the relevant features to classify these named entities to be either “destination” or “non-destination”. Depending on the sources of the information, we divide the 8 feature sets into two categories: feature sets 1 - 5 are extracted by analyzing the named entity itself and the document containing it; feature sets 6 - 8 are obtained by linking the information in the document with external information sources, such as online phone book search engines and domain registrant databases.

Feature Set 1 captures the “shape” of the named entity, including: whether the name has (1) all letters of all terms capitalized, or (2) all initial letters of all terms capitalized, or (3) length less than or equal to 5.

Feature Set 2 matches the sentences containing the named entity against 9 sets of pre-defined language patterns, such as “... is located on your right” and “... will be the second building on the left”.

Feature Set 3 checks if the named entity contains a type noun or a US state name. We identified a list of 129 destination type nouns, such as “school”, “university” and “hotel”. These are the type of businesses and organizations providing the driving directions.
**Feature Set 4** includes HTML visual features: route direction authors frequently use HTML visual features to emphasize route components. For example, destinations often appear in the titles and headings of the web page. Our approach checks whether a named entity is in the title, heading, or is a link.

**Feature Set 5** is the normalized count. Destinations often appear more than once in the document. We calculated the count and normalize by document length. If the normalized count is larger than a threshold \( nc \), we assign the feature to this candidate. We evaluated different values for the threshold, from 0 to 1 with step size 0.1. We used the same set of values on the following thresholds.

**Feature Set 6:** Domain registrant’s names. Given the URLs of the direction web pages, we look up the URLs to obtain the domain registrants’ names, then calculated the cosine similarity between the extracted names and the registrant names. If the similarity exceeds a predefined threshold \( d \), we assign this feature to the named entity.

**Feature Set 7:** Phone book search results. If phone numbers are present, we look them up in the online phone book search engine to get the names associated with each phone number. Then calculate the cosine similarity similar between the obtained name and the named entity. If it exceeds a pre-defined threshold \( p \), we assign the feature to the named entity.

**Feature Set 8:** We crawled the titles of other web pages in the same domain as the driving direction page, and computed the proportion of titles containing the extracted named entity. If the proportion exceeds a pre-defined threshold \( t \), we assign the feature to the named entity.

### 2.2.2.2 Post-Processing for Improving Destination Name Recognition

After training and evaluation, we found that some destination names were misclassified as “non-destination” due to lack of strong features: some names only appeared once in the document; they do not have strong language pattern features, such as “directions to ...”; they do not match any registered names from external information sources. However, they are similar to some of the correctly classified destination names. For example, “Christiana Hospital” was classified as “non-destination”; while “Christiana Hospital campus” was classified as “destination” correctly. Based on the observation, we designed a post-processing step to improve
the recognition of destination names. For each document, we take the named entities classified as “non-destination” and calculate the cosine similarities between this named entity and all the named entities classified as “destination”. If one of the similarities is larger than or equal to a pre-defined threshold \( c \), we re-classify the named entity as “destination”. Algorithm 2 gives the details.

**Algorithm 2** Post-Processing

**Input:**
(1) Pairs of named entities and predicted class labels from one document: \((e_1, l_1), (e_2, l_2), \ldots, (e_n, l_n)\), where \(l_i \in \{\text{"destination"}, \text{"non-destination"}\}\)
(2) A threshold for cosine similarity comparison \( c \)

**Output:**
Pairs of named entities and their new class labels: \((e_1, l'_1), (e_2, l'_2), \ldots, (e_n, l'_n)\)

**Procedure:**
1: for each \((e_i, l_i)\), where \(l_i \neq \text{"destination"}\) do
2: for each \((e_j, l_j)\) where \(j \neq i\) and \(l_j = \text{"destination"}\) do
3: \(sim \leftarrow \text{cosine_similarity}(e_i, e_j)\);
4: if \(sim \geq c\) then
5: \(l_i \leftarrow \text{"destination"};\)
6: break;
7: end if
8: end for
9: end for
10: return \((e_1, l'_1), (e_2, l'_2), \ldots, (e_n, l'_n)\);

### 2.2.3 Experiment Results

We manually labeled 246 destination names and 793 non-destination names from 100 documents. In order to avoid the problems caused by the imbalanced class distribution, we constructed a balanced data set by choosing all the 246 destinations and randomly selected 245 non-destinations as ground truth. We use 2-fold cross validation in our experiments. All numbers reported in this section are averaged results from the 2-fold cross validation.

Three machine-learning models are compared: Naive Bayes [18], Maximum Entropy (MaxEnt) [19] and C4.5 Decision Tree [70]. The three models show similar performance. We show the results using the MaxEnt model. The tunable parameters are: \( d \) for threshold of domain registrant name matching; \( t \) for threshold
for proportion of web page titles containing the named entity; \( p \) for phone book registrant name matching; \( c \) for threshold of cosine similarity in post-processing; and \( nc \) for threshold of normalized count feature. Note that the scopes of \( y \)-axis in Figure 2.4 and 2.5 are from 0 – 0.5 and 0.5 – 1.

### 2.2.3.1 Evaluation of External Features

We evaluate the effects of external features (Feature 6, 7 and 8) by comparing the performance of the models under 3 settings: (1) internal features only (Feature 1 - 5), no post-processing; (2) all features (Feature 1 - 8), no post-processing, and (3) all features plus post-processing. In Figure 2.4, these settings are represented by “\( \text{Base} \)”, “\( \text{Base+Ext.} \)” and “\( \text{Base+Ext.+PPS.} \)” respectively. Among the tunable parameters of external features, \( d \) has the largest impact on the performance. With the same \( d \), the effects of other parameters are similar. Therefore, in Figure 2.4, we varied \( d \) from 0.1 to 1.0, with \( \text{step} = 0.1 \).

The performance reaches the highest when \( d = 0.6 \). The external features reduced the total number of mis-classified named entities by 5.6%. After applying post-processing, the reduction reaches 16.9%, as shown in Figure 2.4a. The overall accuracies increased from 81.8% (Base) to 82.9% (Base + Ext.) and then to 84.9% (Base + Ext. + PPS.). In terms of recall, 12.2% un-recognized destination names were picked up by external features, and 37.0% were picked up after applying post-processing. As shown in Figure 2.4b, the precision dropped about 1% to 2%. With the two effects working together, the F1 score still increases from 80.4% to 81.9% (Base+Ext.) and then to 84.8%(Base+Ext.+PPS.).

Without the external features and the post-processing step, the overall classification accuracy is around 79%. External features and the post-processing step improved the accuracy to 84%.

### 2.2.3.2 Evaluation of Post-Processing

We trained and tested the MaxEnt classifier with all the features. We then applied the post-processing algorithm and gathered the overall accuracy, the precision, recall and F1 score of destinations. 10 values (from 0.1 to 1.0, \( \text{step} = 0.1 \)) are used for all thresholds. All experiment results showed similar effects of the post-
processing step. For simplicity and clarity, we show the results in Figure 2.5 under one setting: $d = 0.5$, $t = 0.1$, $p = 0.3$, and $nc = 0.04$.

Figure 2.5 shows that although post-processing decreases the precision but increases recall significantly. The overall effect is it increases the accuracy and F1 score. A threshold value of 0.7 gives us the best performance in this setting. It increases the overall accuracy and F1 score of destination by 2% and 3%, respectively.

2.2.4 Discussion on Feature Relevance

We calculate correlations and mutual information to capture the dependency between each feature set and the class label. The whole feature space is divided into 6 subsets: feature sets 1 - 5 are 5 subsets; features 6 - 8 together form one subset, called “external features”. Given a feature set, $F$, and a named entity $e$, a random variable $x$ is 1 if $e$ has a feature in $F$, 0 otherwise. A random variable $y$ is 1 if
Figure 2.5: Effects of Post-Processing on Classification Results

e is destination, 0 otherwise. Table 2.11 shows that external features have strong relevance with the destination.

Table 2.11: Correlation and Mutual Information between feature sets and class labels (sorted in decreasing order)

<table>
<thead>
<tr>
<th>Feature Sets</th>
<th>Corr</th>
<th>M.I.</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Features</td>
<td>0.565</td>
<td>0.2480</td>
</tr>
<tr>
<td>HTML features: in title or heading</td>
<td>0.559</td>
<td>0.2462</td>
</tr>
<tr>
<td>Language Patterns</td>
<td>0.504</td>
<td>0.1956</td>
</tr>
<tr>
<td>Type nouns</td>
<td>0.353</td>
<td>0.0938</td>
</tr>
<tr>
<td>Normalized Count</td>
<td>0.351</td>
<td>0.0909</td>
</tr>
<tr>
<td>Word Shape</td>
<td>−0.112</td>
<td>0.0090</td>
</tr>
</tbody>
</table>
Disambiguating and Geocoding
Road Names using Exact-All-Hop
Shortest Path Algorithm

3.1 Preliminaries

In a gazetteer, the same road name can be shared by multiple actual roads. For example, in OpenStreetMaps [28], there are four roads with the name "Atherton", one in Pennsylvania, US, one in California, US, one in Australia and one in the UK. The address or city and state of the destination, if found in the text, can be used for disambiguation. However, the address or city of the destination may be mentioned in other web pages and the authors of the direction page assume the readers can figure it out from the context; thus the address or city may be missing in the direction page. In some cases, the address found in a direction page may be misleading. For example, the address is for the headquarter of a company and the directions are for one branch office in another state.

Road names in directions are placed in a sequence because they are connected to one another. Each road name corresponds to a set of more than one road in a gazetteer. Ideally, we can find one road in each set, such that if they are ordered according to the road name sequence, they are either connected to the next one in the sequence, or have a small distance to the next one (due to the existence of
errors in the latitudes or longitudes of roads in the database). It can be done by Shortest Path algorithms, such as Dijkstra or Bellman-Ford algorithms. However, as discussed before, it is very likely that none of the roads under one road name is the road referred to by the text. This is because (1) the author of the directions used a wrong or obsolete name, thus the roads returned by querying this name do not include the true road. (2) the incompleteness of the gazetteer. The existence of such road names in the sequence will severely influence the result of the Shortest Path algorithms. Figure 3.1 gives an example.

![Figure 3.1: Simple methods fail in noisy environment](image)

For the three road names A, B and C, each yields a set of actual roads in the gazetteer: \{a_1, a_2\} for A, \{b_2\} for B and \{c_1, c_2\} for C. The true roads are \{a_1, b_1, c_1\}. Due to either wrong road names or incomplete gazetteer, b_1 is not returned by the gazetteer, therefore the algorithm will select b_2 since it is the only one for B. Once b_2 is selected, the algorithm will select c_2 since it is the closest C road to b_2, thus missing the true answer c_1.

As can be seen from the above example, the noisy environment requires the disambiguation algorithm to allow one or more road names to be missing from the answer, while still keep the number of included road names high. The path selection is therefore subject to two objectives: (1) cover as many road names as possible and (2) minimize the sum of distances to transit from one road to the next in sequence.
3.2 Related Work

Before we proceed to the solutions, we briefly review existing work on place name disambiguation, multi-constrained shortest path algorithms and route extraction from human-generated directions.

3.2.1 Geographic Term Disambiguation

Place name disambiguation has been studied extensively. Most of place name disambiguation methods can be categorized into two groups: rule-based and data-driven methods [47]. In [48], a distance-based method was introduced. The heuristic is that locations mentioned in the context (a sentence, a paragraph or a whole document) often are close together. First, for each ambiguous name in a document, all geographic locations with this name are found. Then the centroid of all locations of all names are calculated. For each ambiguous name, the location closest to the centroid is selected as its true location. In [31], each location is mapped onto an ontology [49], the locations selected are the ones yielding the maximum conceptual density, which is the correlation between the sense of a word and its density. Both methods were compared in [50]. Other heuristics include: (a) looking for qualifiers in context, e.g. “IL” immediately following “Chicago”, (b) using the location with the largest population, (c) if multiple spots of the same place name has only one disambiguated spot, its meaning delegates to others, and etc. Such rules have been used in combination [29, 30]. However, none of them is designed for the noisy environment we are facing.

Data-driven methods train statistical machine-learning models on a set of annotated data, then use the trained model for disambiguation. However, annotated data are expensive to obtain. To remedy this problem, a bootstrapping method was proposed [51] which produces accurate results while using a small set of annotated data. In addition, external information sources have also been used, such as Wikipedia entity pages, were also employed for building the disambiguation models. In [52], external knowledge was used to build a context article cosine similarity model and a Rank-SVM based on a taxonomy kernel. In [33], Wikipedia was used to learn a co-occurrence model. It models how place names occur together. In a recent work [34], the authors coupled place name extraction and place name dis-
ambiguation tasks. The location priors and location sense priors were learned from web pages, search queries and local linguistic contexts. Then algorithm iterates between the extraction and disambiguation until convergence. These methods rely on annotated data sets and/or external information sources, while our method, as will be shown later, does not depend on annotated data nor external sources.

3.2.2 Hop-constrained Shortest Path Algorithms

Traditional shortest path algorithms, such as Dijkstra and Bellman-Ford algorithms, find a path from the source to the destination, such that the weight of this path is the minimum among all paths from the source to the destination. These algorithms minimize the path weight only. Number of hops of this path is not taken into consideration. However, our problem, in addition to minimizing the path weight, has to maximize the number of hops. Our problem belongs to the multiple-constrained shortest path problems. Multiple-constrained shortest path problems arise naturally in the context of quality-of-service (QoS) routing in networks, where routes (paths) have to provide service guarantees such as delay and bandwidth, in addition to minimizing the network costs. The general problem, i.e. multiple additive constrained routing problem, is known to be NP-complete [35].

A number of special constraints, such as hop count, are amenable to tractable solutions. In [53], the authors proposed a polynomial time solution to the AHOP problem, i.e. to find the shortest path whose hop count is below any given number. In [54], not 1 but $k$ shortest paths bounded by a given maximum hop count are found. However, These paths are only bounded by a given maximum hop count, but not maximizing the hop count. In [55], the authors introduced the exact all hops shortest path (AHSP). Given a hop count, source and destination, AHSP finds a shortest path between source and destination with the exact number of hops. A polynomial time solution is theoretically proven to exist. However, no actual algorithms were given.

3.2.3 Route Extraction from Text

The problem of automatic route information extraction from text was studied in [9]. However, the authors focused on text information extraction. No actual routes
were extracted. In [24], the authors recovered routes on maps based on text route
descriptions. They extracted landmarks along the routes, then recover the routes
by connecting the landmarks. According to the authors, they “try to bypass the
important problem of ambiguity” by using IE techniques. In human-generated, as
well as machine-generated route directions, road transitions are usually described
without other landmarks, thus making this method unsuitable.

3.3 Problem Formalization

In this section, we show the whole process of abstraction and generalization from
the road name disambiguation problem to the subsequence selection problem and
finally to the hop-constrained shortest path problem. We define new concepts and
show how these concepts are mapped to the concepts in the road name disam-
biguation problem. We begin with the following definition:

Problem Statement 1. Given a text route directions and a list of \( k \) road names
extracted from the text \( r_1, r_2, ..., r_k \), and a gazetteer, road name disambiguation is
to find for each \( r_i (1 \leq i \leq k) \) the true road referred to by the text, if it exists in
the gazetteer.

Each actual road returned by searching the gazetteer is a linear spatial object
representing a road. In the following discussion, a linear spatial object (road) is
abstracted to be a point or an object. The distance between two roads is defined
to be the minimum distance between them. Therefore, the distance between two
points (objects) does not satisfy the triangle inequality. Each road name yields a
set of points (objects) when searched for in the gazetteer. These road names, if
put in the order they appear in the text, form a sequence. We continue by defining
the following terms:

Definition 1. A sequence of length \( K \), \( seq = (c_1, c_2, ..., c_k) \), is an ordered list of
\( k \) sets, where each set \( c_i \) (\( 1 \leq i \leq k \)) is non-empty, i.e. \( |c_i| \geq 1 \). A subsequence
of \( seq \) is \( seq' = (c_{i_1}, c_{i_2}, ..., c_{i_l}) \), where \( 1 \leq i_1 < i_2 < ... < i_l \leq k \). Note that a
subsequence is also a sequence.

Definition 2. Given a sequence \( seq = (c_1, c_2, ..., c_k) \), a route of this sequence is
an ordered list of points \( r = (p_1, p_2, ..., p_k) \), where \( p_i \in c_i \), \( 1 \leq i \leq k \). The hop
count of the route is \( k - 1 \), since it takes \( k - 1 \) hops to reach the end of the route. The distance \( \text{dist} \) of route \( r \) is defined as:

\[
\text{dist}(r) = \sum_{i=1}^{k-1} \text{dist}(p_i, p_{i+1})
\]

where \( \text{dist}(\cdot, \cdot) \) is a distance function which takes two points (objects) as input and returns the distances between them.

Note that the distance of a route in the above definition corresponds to the total distances in transition from one road to the next, not the total traveling distance along the roads. With the above definitions, the problem can be formalized as:

**Problem Statement 2.** Given a sequence \( \text{seq} = (c_1, c_2, ..., c_k) \), a distance threshold \( d \), and a distance function \( \text{dist}(\cdot, \cdot) \), find a subsequence \( \text{seq}' = (c_{i_1}, c_{i_2}, ..., c_{i_l}) \) and a route \( r = (p_1, p_2, ..., p_l) \) of \( \text{seq}' \), where \( 1 \leq i_1 < i_2 < ... < i_l \leq k \) and \( p_j \in c_{i_j} \), such that the following two conditions are satisfied:

1. The distance of route \( r \) is below the threshold \( d \). i.e.: \( \text{dist}(r) \leq d \)

2. The hop count of the subsequence \( l \) is the maximum among all routes satisfying Condition 1.

If more than one route satisfy both conditions, select the ones with the minimum distance.

Figure 3.2 shows an example. A sequence of 4 sets is given as \( \text{seq} = (P, Q, R, S) \), where \( P = \{p_1, p_2, p_3\} \), \( Q = \{q_1, q_2\} \), \( R = \{r_1, r_2\} \) and \( S = \{s_1, s_2\} \). The distance threshold is \( d = 3 \). The distances between some pairs of points are given by the length of the lines connecting the pair of points. All points are on a 2-dimensional area. In this example, only the subsequence of points \( (p_1, q_1, s_1) \) and \( (p_3, r_1) \) satisfy Condition 1. Since \( (p_1, q_1, s_1) \) gives a hop count of 2, which is larger than the hop count of \( (p_3, r_1) \), it is selected and returned as the result.

To solve this problem, we use a \( k \)-partite graph to model it. Each point corresponds to a vertex in the graph. Each road name corresponds to a set of more than one vertices, representing the ambiguity. In order to allow skipping sets, we connect each vertex to all vertices in all other sets. Edges are directed to represent the sequential nature of the sets. Edge weights correspond to the distance
between two points (objects). Thus a sequence is converted to be a semi-complete directed \( k \)-partite graph, defined as follows:

Definition 3. A **semi-complete directed \( k \)-partite graph** is a graph \( G = (V, E) \), where \( V = \{V_1, V_2, ..., V_k\} \) are \( k \) disjoint sets of vertices. \( E = \{(u, v) : \forall u \in V_i, \forall v \in V_j, \forall 1 \leq i < j \leq k\} \) are a set of directed edges.

Figure 3.3 illustrates such a graph. There is an edge connecting any pair of vertices in different sets. If the directed edges are replaced with undirected edges, it becomes a complete \( k \)-partite graph. The direction of the edge is from the vertex in the lower-numbered set to the vertex in the higher-numbered set.

Figure 3.3: graph model

The disambiguation problem and the subsequence selection problem are then further abstracted and generalized to be the graph problem stated below:

Problem Statement 3. Given a positive number \( W \), a semi-complete directed \( k \)-partite graph \( G \), and a non-negative weight function on all edges \( w(\cdot, \cdot) \in \{0\} \cup \mathbb{R}^+ \),
find a path $p$ in $G$, such that (1) the weight of the path $p$ is smaller than or equal to $W$, (2) the number of hops on $p$ is the maximum among all paths satisfying Condition (1).

3.4 Algorithm Description

3.4.1 An Illustrated Example

Figure 3.4 illustrates how the algorithm works with a simple example. Consider the last 3 sets: $c_k$, $c_j = c_{k-1}$ and $c_i = c_{k-2}$. Starting from any points in $c_k$, only 0 hops can be made since it is already the last set; starting from any points in $c_j$, one can make either 1 or 0 hops; starting from any points in $c_i$, one can make 2, 1 or 0 hops, as shown in Figure 3.4a. Making 0 hops is trivial. Now we consider making 1 hop from points in $c_j$. In order to keep the weight minimum, simply choose the edge with a smaller weight of the two, and record the one-hop destination. We use a table for each point to store the next hop destination and minimum weight, for any possible number of hops, as shown in Figure 3.4b. Starting from points in $c_i$, two hops can be made. First hop is to either $p_{j1}$ or $p_{j2}$. In order to calculate the minimum path weight, we directly use the result in the table of each point in $c_j$. The column with 2 hops can be filled out for $p_{i1}$ and $p_{i2}$, as shown in Figure 3.4c. Similarly, we fill out the column for 1 hop. Starting from points in $c_i$ and exactly 1 hop, one can jump to either $c_j$ or $c_k$. Simply compare the 4 edges we can determine the exact-1-hop shortest path. Then the tables entries are all filled out, as shown in Figure 3.4d.

After filling out the tables for each point in all the sets, we simply scan all entries of the tables, find the entries with a weight smaller than the threshold, then choose the one with the maximum number of hops.

3.4.2 Notations

The input graph $G$ is a semi-complete directed $k$-partite graph, i.e. $G = (V, E)$, where $V = \{V_1, V_2, ..., V_k\}$ and $E = \{(u, v) : \forall u \in V_i, \forall v \in V_j, \forall 1 \leq i < j \leq k\}$. A weight function $w(\cdot, \cdot)$ takes two vertices $u$ and $v$, where $(u, v) \in E$, as input arguments, and returns the non-negative weight of the edge, i.e. $w(\cdot, \cdot) \in \{0\} \cup \mathbb{R}^+$. 
Figure 3.4: A Simple Example of the Algorithm

Our algorithm relies on two important arrays associated with each vertex $u$ to store important path information: (1) a min-weight array $D_u$ and (2) a successor array $S_u$. For a vertex $u$, the $h$-th entry of its min-weight array, i.e. $D_u[h]$, corresponds to the weight of the exact-$h$-hop shortest path starting from $u$; while the $h$-th entry of its successor array, i.e. $S_u[h]$, corresponds to the first-hop destination vertex on the exact-$h$-hop shortest path starting from $u$. For example, suppose path $p=u, v_1, v_2, \ldots$ is the exact-2-hop shortest path starting from $u$, with $v_1$ being the first-hop destination and $v_2$ being the second-hop destination. Then $D_u[2] = w_p = w(u, v_1) + w(v_1, v_2)$ is the total weight of path $p$, and $S_u[2] = v_1$ since $v_1$ is the first-hop destination on the path. For each vertex $u$, $D_u$ and $S_u$ have the same length. The minimum array index value is 0, meaning we stay at vertex $u$. 
and no hops are made. The maximum array index equals to the maximum number of hops that can be made from \( u \). For example, if \( u \in V_i, 1 \leq i \leq k \), the maximum number of hops can be made starting from \( u \) is \( k - i \), because after making \( k - i \) hops along the directed edges, one will reach a vertex in the last set \( V_k \), then no more hops can be made. Thus \( h \) ranges from 0 to \( k - i \) for both \( D_u \) and \( S_u \).

### 3.4.3 Algorithms

#### Algorithm 3 INITIALIZATION

**Input:** dag \( k \)-partite graph \( G = (V, E) \)

**Output:** initialize \( D_v \) and \( S_v \) for each \( v \in V \)

**Procedure:**

1. for \( i = 1 \rightarrow k \) do
2.   for each \( v \in V_i \) do
3.     \( D_v[0] = 0 \);
4.     if \( i < k \) then
5.       \( D_v[1 \ldots (k - i)] = +\infty \);
6.     end if
7.     \( S_v[0 \ldots (k - i)] = NIL \);
8.   end for
9. end for

Algorithm 4 gives the relaxation step on an edge \( (u, v) \). Similar to the traditional relaxation technique used by Dijkstra and Bellman-Ford algorithms, our relaxation step tests whether we can improve the shortest path by one comparison. However, unlike the traditional relaxation technique, our relaxation (1) uses \( u \) as the start of the path and \( v \) as the first-hop destination on the path, and (2) updates the min-cost and the immediate successor of the exact \( i \)-hop path starting from \( u \).

Algorithm 5 fills in \( D_u \) and \( S_u \) with proper values for each \( u \in V \). Line 2 and 3 show that the algorithm processes the vertices in high-ordered vertex sets to low-ordered vertex sets, i.e. vertices in \( V_{k-1} \) are processed first, then vertices in \( V_{k-2} \), and etc., until we finish processing vertices in \( V_1 \). Lines 4 - 10 fill in the min-weight array and successor array for a vertex \( u \in V_i \). Note that each vertex \( v \in V_k, V_{k-1}, \cdots, V_{i+1} \) can be used as the first-hop destination on a exact-1-hop path starting from \( u \). Therefore, each vertex \( v \in V_k, V_{k-1}, \cdots, V_{i+1} \) have to be examined for relaxation of the exact-1-hop shortest path from \( u \). Similarly, each
Algorithm 4 RELAX

Input: vertices $u, v$ such that $(u, v) \in E$, integer $h$ ($1 \leq h \leq k$)
Output: update $D_u$ and $S_u$ for $u$

Procedure:
1: if $D_u[h] > w(u, v) + D_v[h - 1]$ then
2: $D_u[h] = w(u, v) + D_v[h - 1]$;
3: $S_u[h] = v$;
4: end if

vertex $v \in V_{k-1}, V_{k-2}, \cdots, V_{i+1}$ can be used as the first-hop destination on an exact-2-hop path starting from $u$, thus should be examine for relaxation of the exact-2-hop shortest path from $u$. We do so for all possible number of hops of paths from $u$.

Algorithm 5 exact all-hops shortest path on semi-complete directed $k$-partite graph

Input: semi-complete directed $k$-partite graph $G = (V, E)$, weight function $w(\cdot, \cdot) \in \{0\} \cup \mathbb{R}^+$
Output: $D_v$ and $S_v$ for each $v \in V$

Procedure:
1: INITIALIZATION();
2: for $i = (k - 1) \rightarrow 1$ do
3: for each vertex $u \in V_i$ do
4: for $j = 1 \rightarrow (k - i)$ do
5: for each vertex $v \in V_{i+j}$ do
6: for $h = 1 \rightarrow (k - i - j + 1)$ do
7: RELAX$(u, v, h)$;
8: end for
9: end for
10: end for
11: end for
12: end for

After Algorithm 5 is finished, the min-weight array $D_v$ and successor array $S_v$ will be filled for each vertex $v \in V$. For each $v$, $D_v[i]$ gives the weight of the shortest path starting from $v$ with exactly $i$ hops ($0 \leq i < \text{length of } D_v$); $S_v[i]$ is the second vertex on the shortest path starting from $v$ with exact $i$ hops, whose weight is given by $D_v[i]$. Given a weight threshold $W$, we simply examine the min-
weight array $D_v$ for each $v$ to find the entries no larger than $w$. Since the index of the entry in the array gives the number of hops, we choose the largest index $i_{\text{max}}$ of the qualified entries. Suppose vertex $v$ is such a vertex: $D_v[i_{\text{max}}] \leq W$ and $i_{\text{max}}$ is the largest index among all indices of qualified array entries. The answer to our problem is a path starting from $v$. The next vertex along the path is given by $v_2 = S_v[i_{\text{max}}]$; and the vertex after next is $v_3 = S_{v_2}[i_{\text{max}} - 1]$. By using the successor arrays, we can easily recover all vertices on the path.

### 3.4.4 Proof of Correctness

Algorithm 5 is a dynamic-programming solution. The solution to a bigger problem is built upon the solution to a smaller problem. The algorithm starts with the vertices in the second highest numbered set, i.e. $v \in V_{k-1}$, which only has exact-1-hop paths. Then, for each vertex set $V_i$ with a lower number $i$, the solution is built by examining the weights of the out-going edges and the information stored in the arrays of vertices in higher numbered sets $V_{i+j}$. We do so because of the optimal substructure of the exact-$i$-hop shortest path, stated and proved as follows:

**Lemma 1.** Given a semi-complete directed $k$-partite graph $G = (V, E)$, where $V = \{V_1, V_2, \ldots, V_k\}$, with weight function $w : E \to \{0\} \cup \mathbb{R}^+$. Let $p_1 = \langle v_1, v_2, \ldots, v_h \rangle$ be the exact-$h$-hop shortest path starting from $v_1$, then $p_2 = \langle v_2, v_3, \ldots, v_h \rangle$ is the exact-$(h-1)$-hop shortest path starting from $v_2$.

**Proof.** The correctness can be shown by a proof-by-contradiction: if path $p'_2 = \langle v_2, v'_3, \ldots, v'_h \rangle \neq p_2$ is the exact-$(h-1)$-hop shortest path starting from $v_2$, we can construct another exact-$h$-hop path $p'_1 = \langle v_1, v_2, v'_3, \ldots, v'_h \rangle$ starting from $v_1$. Since $w(p_2) > w(p'_2)$, we have $w(p_1) = w(v_1, v_2) + w(p_2) > w(v_1, v_2) + w(p'_2) = w(p'_1)$. Then $p_1$ is not the exact-$h$-hop shortest path starting from $v_1$, which contradicts to the assumption. \qed

This property allows us to solve the problem efficiently. Based on this Lemma, we show the correctness of the algorithm by the following theorem:
Theorem 1. Given a semi-complete directed $k$-partite graph $G = (V, E)$, where $V = \{V_1, V_2, \cdots, V_k\}$, with weight function $w : E \rightarrow \{0\} \cup \mathbb{R}^+$. Let the proposed exact all-hops shortest path algorithm run on this graph, when the algorithm terminates, for each vertex $u \in V$, $D_u$ contains the weights of exact all-hops shortest path starting from $u$; $S_u$ contains the first-hop destinations of exact all hops shortest path starting from $u$.

Proof. We first show that Lines 4 - 10 finds the weights and immediate successors of exact all hops shortest paths starting from $u$.

Given a vertex $u \in V_i$, where $1 \leq i \leq (k - 1)$, any vertices in a vertex set with a set number higher than $i$, i.e. $v \in V_{i+j}$ where $j \in [1, (k - i)]$, can be the first-hop destination of an exact-$m$-hop path starting from $u$, where $m \in [1, (k - i - j + 1)]$. That is to say, $\forall u \in V_i$ and $v \in V_{i+j}$, $\exists p = \langle u, v, \cdots \rangle$, where the number of hops of $p$ from $u$ is $m \in [1, (k - i - j + 1)]$. Suppose path $p' = \langle v, \cdots \rangle$ is the exact-$(m - 1)$-hop shortest path starting from $v$, according to Lemma 1, $p = \langle u, v \rangle + p'$ is the exact-$m$-hop shortest path with $u$ being the starting vertex and $v$ being the first-hop destination. Lines 4 - 10 iterates through all vertices $v$ and use them to relax the exact-$m$-hop paths of $u$, for all possible values of $m$. Thus, at the end of Lines 4 - 10, the computation of the shortest paths of all possible number of hops starting from $u$ is finished.

The computation for vertices in $V_k$ is trivial and is performed by the initialization step. Line 2 and 3 iterate through vertex sets from higher-numbered to lower-numbered, i.e., $V_{k-1}$, $V_{k-2}$, ..., $V_1$. This particular order is chosen in order to guarantee that when computing the shortest paths of vertices in a particular vertex set, all shortest paths starting from any vertices in a higher-numbered vertex set have been found and ready to be used to construct the solution to the bigger problem. Therefore, it guarantees to generate the shortest paths of all possible numbers of hops starting from all vertices.

\[\square\]

3.4.5 Time Complexity Analysis

Without loss of generality, we assume that each disjoint vertex set has the same number of vertices, i.e. $|V_1| = |V_2| = \cdots = |V_k| = n$.

The lengths of min-weight array $D_v$ and successor array $S_v$ are different for
vertices in different disjoint vertex sets. In \( V_1 \), the lengths of the two arrays are \( k \) for each vertex, since if we start from each vertex in this set, we can make at least 0 hops (meaning we stay in this vertex and do not make any hops) to at most \( k - 1 \) hops. This length decreases by one for each subsequent vertex set. In \( V_{k-1} \), the length is 2, since at most one hop can be made starting from each vertex in it. In \( V_k \), these arrays only have one hop can be made starting from each vertex in it. In \( V_k \), these arrays only have one hop can be made starting from each vertex in it. In

\[
2kn + 2(k - 1)n + ... + 2n = \sum_{i=1}^{k} 2in = k(k + 1)n = O(k^2n)
\]

The initialization step fills in each entry of the two arrays for all vertices. The answer generation process after Algorithm 5 examines each entry in the arrays once. Therefore, the running time of both the initialization step and the answer generation process are bounded by \( O(k^2n) \).

Algorithm 5 calls the relaxation procedure multiple times. Since relaxation performs exactly one comparison, and the update, if necessary, takes only constant time, relaxation runs in \( O(1) \) time. We now investigate how many times the relaxation is called by Algorithm 5.

Now consider Line 4 - 10. Line 4 - 10 finds exact-all-hops shortest paths starting at \( u \in V_i \) and fills in \( D_u \) and \( S_u \) for \( u \). The for-loop on Line 6 calls relaxation \( (k - i - j + 1) \) times. Line 4 iterates through values of \( j \) from 1 to \( (k - 1) \). Line 5 iterates all \( n \) vertices in \( V_{i+j} \), thus adding a constant factor \( n \). Therefore, Line 4 - 10 calls relaxation \( \sum_{j=1}^{k-i}(k - i - j + 1) \times n \) times.

The for-loop on Line 2 iterates values of \( i \) from \( (k - i) \) to 1; Line 2 iterates through all vertices in set \( V_i \), thus the total running time of Line 2 - 12 of Algorithm 5 is:

\[
T = \sum_{i=k-1}^{1} \left( n \times \left( \sum_{j=1}^{k-i}(k - i - j + 1) \times n \right) \right)
\]

\[
= \sum_{i=k-1}^{1} \left( n \times \left( (k - i) + (k - i + 1) + \cdots + 1 \right) \times n \right)
\]
\[
\begin{align*}
&= \sum_{i=k-1}^{1} \left( \frac{(k-i)^2 + (k-i) \times n^2}{2} \right) \\
&= \sum_{m=1}^{k-1} \left( m^2 + m \right) \times \frac{n^2}{2} \\
&= \sum_{m=1}^{k-1} \left( m^2 + m \right) \times \frac{n^2}{2}
\end{align*}
\] (3.1)

Since we have the following summation of squares:

\[
\sum_{m=1}^{k-1} m^2 = \frac{(k-1) \times k \times (2k-1)}{6} = O(k^3)
\]

The running time is:

\[
T = \left( O(k^3) + O(k^2) \right) \times \frac{n^2}{2} = O(k^3 n^2)
\] (3.2)

Since the initialization and answer generation steps are both bounded by \(O(k^2 n)\), the total running time of the algorithm is bounded by \(O(k^3 n^2)\).

The weight of an edge between two vertices is the distance of two spatial objects on the earth surface. The weights can be stored in an adjacency matrix. It is done by a pre-computation step. The number of distance calculations is equal to the number of edges in the graph, i.e. \(\frac{k \times (k-1)}{2} \times n^2\).

### 3.5 Experiments

#### 3.5.1 Data Collection

We built our collection of human-generated route directions using the method described in [9]. We randomly chose 53 out of 10,000 web pages containing human-generated route directions. Each page contains one or more route descriptions. We chose the first route description from each page. From each route description, we manually extracted the road names and put them in the order they appear in the text. For each road name, we stripped the cardinal direction words, if they exist, in the road name. For example, “North Atherton Street” is converted to “Atherton Street”. Cardinal directions are also frequently used with highway names, for example, “Travel along I-80 east”. Such cardinal directions are relative to the
starting point of the route. Therefore they are also stripped. So “I-80 east” is converted to “I-80”.

### 3.5.2 Pre-Processing

We used OpenStreetMap [28] as the gazetteer. The road names are looked up in OpenStreetMap using fuzzy string match, i.e. if the name of a road in the gazetteer contains the road name we are searching for as a substring, we use this road in the database as a sense of the name. We use this search strategy because OpenStreetMap is a user-contributed database. It is very likely that the road names in the gazetteer is slightly different from the road name used in the route description, even if they refer to the same road. Using this strategy can minimize the differences and increase the chance that the correct sense of a road is returned in the results.

In OpenStreetMap, each road is stored as one or multiple road segments because OpenStreetMap requires that each database record should have no more than 2,000 points. Long roads are segmented into smaller pieces, for example, Interstate 80 has over 6,000 segments in OpenStreetMap. We find the true number of senses by stitching these small segments. If two segments are connected, they are merged. Since GPS devices may introduce errors in the latitudes and longitudes when users upload the information, we allow a 1 mile (about 1,600 meters) gap between road segments when stitching. Table 3.1 gives the statistics of our dataset.

### 3.5.3 Evaluation Results

A map-based algorithm was proposed in [48] and evaluated in [50]. We compare our disambiguation algorithm (EAHSP) with the map-based algorithm. The

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of route descriptions</td>
<td>53</td>
</tr>
<tr>
<td>number of road names</td>
<td>202</td>
</tr>
<tr>
<td>total number of roads in gazetteer</td>
<td>8464</td>
</tr>
<tr>
<td>Average number of roads per name</td>
<td>41.9</td>
</tr>
<tr>
<td>Maximum number of roads</td>
<td>704</td>
</tr>
<tr>
<td>Minimum number of roads</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.1: Statistics
map-based algorithm consists of the following procedures: let $t_1, t_2, ..., t_k$ be the $k$ toponyms in the text.

1. For each toponym $t_i$, find all its possible geographic locations $s_i$. The locations for all toponyms form a set $S$.

2. Calculate the centroid $c$ of all locations in $S$.

3. Remove from $S$ all locations $s_i$ such that the distance between $s_i$ and $c$ is larger than $2\sigma$, where $\sigma$ is the standard deviation of the set of locations. The remaining locations form a set $S'$.

4. Calculate the centroid $c'$ of all locations in $S'$.

5. For each $t_i$, select its closes location to $c'$ to represent its actual location.

Our algorithm generates two sets of results for each route description: given a maximum path weight allowed, (1) we find a path $p$ with the maximum number of hops, say $h$ hops. If multiple paths are found, we select the one with the minimum path weight. (2) After finding the first path $p$, we find an $(h - 1)$-hop path $p'$ such that its path weight is smaller than the weight of $p$ and $p'$ is not obtained by cutting off one vertex in $p$. If multiple paths are found, we select the one with the minimum path weight. We extract such two paths to evaluate the trade-off between path weights and the number of hops. The requirement that $p'$ is not obtained by simply cutting off one vertex in $p$ will enable the algorithm to find more vertices, instead of choosing a subset of vertices in the already found path. We ran the algorithm on 5 values of the maximum allowed path weight (called $max\_weight$): 0, 1600, 3200, 8000 and 16000, in meters.

We compare the two algorithm on three metrics: (1) precision, (2) recall and (3) F1 score, defined as follows:

$$precision = \frac{\text{correct senses returned by the algorithm}}{\text{all senses returned by the algorithm}}$$

$$recall = \frac{\text{correct senses returned by the algorithm}}{\text{all correct senses of all roads in the data set}}$$
Each sense of the roads in our data set is highlighted in a map and presented to a human annotator, together with the original web page where the road name is extracted, so that the human annotator can decide whether the sense is the correct sense referred to by the text or not. The annotated senses are then used for evaluation.

Figure 3.5 shows the results when the algorithm generates a path with the maximum number of hops; figure 3.6 shows the results when the algorithm generates a path with one less hop but a smaller path weight. Note the map-based algorithm remains a straight line in each figure since it is not affected by the maximum allowed path weight.

\[ F1\text{score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \]

(a) Precision  
(b) Recall  
(c) F1 Score

Figure 3.5: Paths with the Largest Number of Hops
In the setting where the algorithm finds the longest path $p$ for each route description, under all different values for $\text{max weight}$, the EAHSP algorithm achieves high precisions ranging from 79.7% to 90%, while the map-based only achieved a precision of 21.13%. Recall of EAHSP increases when $\text{max weight}$ increases. This is because of the errors in the latitudes and longitudes of the roads in the gazetteer. Two roads that are connected in the real world may have a small gap between them in the gazetteer. When $\text{max weight} = 0$, the algorithm has zero tolerance on the errors, thus will not be able to find the correct roads if they have a gap between them. When $\text{max weight}$ increases, the ability to tolerate errors increases, therefore the recall increases. The recall is 57.4% when $\text{max weight} = 0$, but increases immediately to 76.6% when $\text{max weight} = 1600$, and keeps increasing. Recall of map-based algorithm is only 31.92%. The F1 score of EAHSP when $\text{max weight} = 0$ is 69.0% because of its relatively low recall. Under all other
max_weight values, EAHSP algorithm achieves a high F1 score ranging from 81.0% to 82.8%; while the F1 score of map-based is only 25.4%.

In the setting where the algorithm finds p', the second longest but with smaller weight than p, the performance of EAHSP is sensitive to the value of max_weight. When max_weight = 0, since the path cannot be a strict subsequence of p, it is forced to select other vertices and pushed away from the correct senses. It also fails to find such a path for many files since no path can satisfy the conditions while not being part of p. Thus the precision and recall drops below the map-based algorithm. But as max_weight increases, p' has more and more overlapping vertices with p, therefore, the performance increases.

The results show that our algorithm significantly outperforms the existing map-based algorithm. Between the two strategies of generating the paths, selecting the path with the largest number of hops is better than the other strategy.

3.6 Conclusion and Future Work

Road name disambiguation is an important research issue in achieving automatic extraction, understanding and visualizing human-generated route directions. It is a difficult research topic because road names in the text display different characteristics from traditional toponyms such as city or country names. In addition, the presence of errors in the names introduced by human beings and missing data in gazetteers have further increased the difficulty in solving this problem. Although toponym disambiguation has been studied extensively and the achievements in this research topic are fruitful, existing methods do not apply well on our road name disambiguation problem in a noisy environment. In order to solve this problem, we revisited one important heuristic for disambiguating place names, i.e., places mentioned in the context are often close together geographically. Therefore, the true road of a road name is spatially close to the true road of the next road name in the sequence. We introduced a novel approach of modeling the ambiguities and noise, i.e., using a semi-complete directed k-partite graph. The disambiguation problem is then successfully converted to a hop-constrained shortest path problem. We further designed an efficient algorithm to solve this shortest path problem. The effectiveness of our algorithm has been confirmed by evaluation on real data and
comparison with an existing method.

In the future, we will incorporate spatial reasoning and natural language processing into our work, in order to accurately identify the segments of roads involved in the route description, instead of entire roads. We will use language cues to identify turns and merges of roads. Using cardinal directions, such as “north” and “south”, combined with spatial information of the roads, we can infer the directions of the route and prune uninvolved road segments. We will extract the destination names and addresses, and look up the names and addresses in gazetteers or yellow pages, to locate the destination on the last segment of the route. Our final goal is to truly recover a route description from text form to digital maps. We will continue to report our progress.
Chapter 4

Processing Transitive Nearest Neighbor Queries in Multi-channel Access Environment

4.1 Preliminaries and Related Work

4.1.1 System Model for Broadcast Environment

In this section, we first describe a system model of multiple channel wireless broadcast to introduce the system components, basic concepts and constraints of wireless broadcast environments. We then review the R-tree index, a fundamental spatial index structure which is closely related to our work, and some related spatial queries.

There are two different information access mechanisms available in wireless broadcast environments for mobile users: 1) on-demand access and 2) broadcast access. In the on-demand access scheme, a mobile client sends a query to the server through a dedicated point-to-point wireless channel. The server, which is responsible for processing the query, returns the answer directly to the mobile client. On the other hand, in broadcast access scheme, the server broadcasts the data in publicly available wireless channels in the unit of page (or packet, frame). The mobile client, after receiving a query from the user, tunes into the broadcast channels and downloads the interested pages. It then processes the query locally...
and finally returns the result to the user. Compared to the on-demand access mode, broadcast scheme conserves the limited resource of wireless bandwidth by allowing an arbitrary number of users to access the same information simultaneously. This scheme can also address the privacy issues in location-based services, since the clients can obtain location-based information without disclosing their own locations to the server.

In this dissertation, we focus on TNN query processing in the broadcast scheme. The system model of our study consists of three major components: 1) the server, 2) the mobile clients, and 3) the wireless broadcast channels. The server maintains and disseminates information (in terms of data objects) of interest to users. It is responsible for scheduling the broadcast program. The mobile client is location-aware (e.g., equipped with GPS)\textsuperscript{1}. Upon receiving a TNN query from its user, a mobile client tunes into the wireless channels and downloads data from multiple channels simultaneously to facilitate query processing. Figure 4.1 gives a high-level view of our system model.

![System Model Diagram]

Figure 4.1: System Model

In broadcast channels without any index information, a mobile client has to stay active and downloads all the data objects to process a query. This approach is inefficient as it consumes a lot of energy. Energy consumption is a major concern due to the limited power source of the mobile clients. Imielinski, Viswanathan, and Badrinath proposed air indexing to tackle this issue. Air indexing interleaves index information with data objects in the wireless channels, which allows a mobile device to...\textsuperscript{1}

\textsuperscript{1}In this dissertation, we use these terms mobile devices and mobile clients interchangeably.
client to first probe the index to get the arrival time of the interested data objects, and then turn into doze mode to save energy. When the desired data is about to be broadcast, the mobile client wakes up and tunes into the broadcast channel to download the data. \((1, m)\) is a representative interleaving technique [36], which divides the dataset into \(m\) equal-sized fractions and broadcasts the entire index preceding every one of the \(m\) fractions.

We use two performance metrics: *access time* and *tune-in time* [37, 56, 36] to measure the effectiveness of our algorithms. The former is the time elapsed from the moment a query is issued till the moment the query is satisfied. Obviously, users want the answer as fast as possible; thus the access time needs to be short. The latter is the time a mobile client stays active to receive the required data for query processing, which is used to represent the energy consumption in the literature. Query processing in broadcast schemes aims at answering a query within a short time, with small energy consumption. Since *access time* and *tune-in time* are both proportional to the number of pages accessed, we measure the two metrics in the number of pages accessed during the query processing.

### 4.1.2 Related Work on Processing Spatial Queries

R-tree index, first proposed by Guttman [3], is a well-known spatial index. It recursively groups nearby objects into minimal bounding rectangles (MBRs) until the whole region is covered by one root node. Figure 4.2a depicts the MBRs of a rectangular region covering 12 objects. Assuming the fanout of the R-tree node is three, the corresponding R-tree structure is depicted in Figure 4.2b. If the objects are available a priori, packing algorithms, such as Nearest-X [57], Hilbert Sort
[58] and STR [4], can be applied to build the R-tree in order to achieve the best performance.

R-tree can support multiple queries efficiently. Take window queries as an example. The search starts at the root node, and follows those nodes with MBRs overlapping with the queried window. Often, NN searches use a branch-and-bound strategy to traverse the R-tree[59, 5, 60]. Different heuristics based on some distance metric are proposed to refine the search range and prune the search space. Among all the algorithms, Best-First (BF) algorithm [5] is most efficient. It maintains a priority queue of candidate nodes, sorted according to ascending order of mindist, the minimal distance to a query point.

![Figure 4.3: Linear Access in Broadcast Channel](image)

However, the performance of the Best-First algorithm deteriorates severely in terms of access time in the context of broadcast because we are using a physically linear media (wireless broadcast channels) to represent a tree structure of R-trees. In broadcast, data object/index information is only available when it is on the air. Once the packet containing the desired information is missed, the mobile client has to wait until the next time the object is broadcast again. In other words, random access, which is commonly supported in resident storage (e.g., disk and memory), is not allowed in wireless broadcast systems. The performance of the best-first algorithm, which employs backtracking to guarantee the effectiveness, deteriorates severely when data are broadcast. Figure 4.3 gives one example. It assumes that the R-tree is broadcast in a preorder traversal in a broadcast cycle. R2, due to a smaller mindist to q, is visited first. Then, R1 is accessed. However, it is already broadcast and is not available until the next time it is broadcast. The access time is prolonged significantly. Therefore, the search algorithms must take into consideration the linear property of broadcasting.

In [2], two algorithms were proposed for answering TNN queries in dingle-channel broadcast environments, namely, Window-Based-TNN-Search algorithm
and \textit{Approximate-TNN-Search} algorithm. The algorithms and their problems will be discussed in detail in the next section. Researchers have studied two other general versions of TNN: \textit{optimal sequenced route} (OSR) [61] query and \textit{trip planning query} (TPQ) [62]. Given a set of points $P$ where each point belongs to a specific category, a starting point $p$ and a destination $e$, OSR finds a route of minimum length starting from $p$ passing through at least one point from each category in a specified order, and ends at $e$; while TPQ finds the shortest route regardless of the order. However, these queries were studied in disk-based databases, not in the wireless broadcast environments.

Lin et al., [63] introduced several heuristics to perform ANN search on R-trees. However, the \textit{threshold} in their work, which is an important parameter in approximation, is static. Their algorithms did not take into consideration several factors, such as R-tree height, that will affect the performance. Besides, the goal of the previous work was different from ours. They did not aim to minimize the search cost.

In the mobile database literature, research on object retrieval and object organization in multi-channel broadcast environments have been reported. Sun, et al., [64] proposed two algorithms to retrieve objects given the distribution of desired pages on parallel broadcast channels using a minimum number of passes and switches (in order of priority) between the channels. Hurson et al. proposed two algorithms to distribute objects of different sizes onto parallel broadcast channels to achieve the minimum broadcast cycle length and to preserve the clustering property [65]. However, they assumed that a mobile client can only access one channel at any one time. Neither did they deal with query processing in broadcast environments.

\section*{4.2 Problem Analysis}

In this section, we first discuss the basic ideas in answering TNN queries and the two TNN algorithms proposed in [2]. A natural question arises is whether these ideas can be adapted for the multi-channel access environment. We use an example to show how they work. Then, we adapt these algorithms into our new multi-channel access environment and perform an analysis.
<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( dis(p,s) )</td>
<td>Euclidean distance between points ( p ) and ( s )</td>
</tr>
<tr>
<td>( p.NN(S) )</td>
<td>The nearest neighbor of point ( p ) in dataset ( S )</td>
</tr>
<tr>
<td>( circle(p,d) )</td>
<td>A circle with point ( p ) as the center and a radius of length ( d )</td>
</tr>
<tr>
<td>( p.TNN(S,R) )</td>
<td>The answer pair of objects in datasets ( S ) and ( R ) of the TNN query with point ( p ) as the query point</td>
</tr>
</tbody>
</table>

Table 4.1: Terminology Definition

4.2.1 Answering TNN Query

First, we use a running example to illustrate how TNN is answered in traditional broadcast environment in which a client can tune into one channel at one time. Given two datasets \( S \) and \( R \), with \( S=\{s_1, s_2, s_3, s_4\} \) and \( R=\{r_1, r_2, r_3, r_4\} \), we assume two broadcast channels co-exist on air, with one broadcasting dataset \( S \) and the other broadcasting dataset \( R \). R-trees are used to index the data points. In broadcast environment, R-tree pointers refer to the arrival time of the data pages. We assume that the broadcast program is organized according to \((1,m)\) scheme, i.e., an R-tree for the dataset (e.g., \( S \) or \( R \) ) is broadcast \( m \) times inside one broadcast cycle. In order to facilitate the following description, Table 4.1 lists the notations and their definitions.

A brute force approach can simply retrieve all the objects from datasets \( S \) and \( R \) and evaluate each pair of objects \((s,r) \in S \times R\). The pair giving the minimum transitive distance is returned as the result. However, this approach obviously is inefficient. It requires the mobile client to download all the data objects, thus results in a large storage requirement in the client as well as a large tune-in time. Since a TNN query only cares about one pair of objects with the minimal transitive distance, efficient algorithms should avoid the retrieval of any unnecessary objects. Zhen, Lee and Lee suggested an *estimate-filter* query processing paradigm [2]:

**Estimate:** find a search range from the query point \( p \) by searching the index;
**Filter:** filter unqualified data objects in the search range determined earlier to find the pair of objects with the minimum transitive distance.

After determining the final answer, the mobile client may later retrieve the desired data object and its associated attributes. An important goal for the design of TNN algorithms is to obtain a small search range in the estimate phase in order to reduce the tune-in time in the filter phase. Meanwhile, we devise optimization techniques (see Section 5) to reduce the tune-in time in the estimation phase without increasing the tune-in time in the filtering phase.

In the estimate phase, the proposed algorithms differ most significantly in how they determine the search range. In the filter phase, two range queries are issued to retrieve all objects in $S$ and $R$ in the search range. After the locations of these objects are obtained, a join is computed to find the minimum transitive distance and the corresponding pair of objects. Finally, the mobile client turns into sleep mode and wakes up when the pair of answer objects are on air to retrieve the attributes associated with the objects. Because the size of the search range has a direct impact on the number of objects retrieved, the search range should be small to reduce the tune-in time. However, it also should be large enough so that it includes the answer pair. Zheng, Lee and Lee [2] have provided the following theorem that can be used to help prune the search range.

**Theorem 1.** Given a query point $p$ and a pair of objects $(s, r) \in S \times R$, let $d = \text{dis}(p, s) + \text{dis}(s, r)$. If $s' \notin \text{circle}(p, d)$ with $s' \in S$, it is guaranteed that $s' \notin p.\text{TNN}(S,R)$. Similarly, if $r' \notin \text{circle}(p, d)$ with $r' \in R$, it is guaranteed that $r' \notin p.\text{TNN}(S,R)$.

The **Window-Based-TNN-Search** algorithm determines the search range by issuing two nearest neighbor queries. The first NN query finds $p$’s nearest neighbor $s$ in $S$, i.e. $s = p.\text{NN}(S)$. Then it issues the second nearest neighbor query to find $s$’s nearest neighbor $r$ in $R$, i.e. $r = s.\text{NN}(R)$. The transitive distance $d = \text{dis}(p, s) + \text{dis}(s, r)$ is used as the search radius. Then two window queries are issued at $p$ to retrieve objects from $S$ and $R$. Finally, a join is applied to find the minimum transitive distance and the corresponding pair of objects. The following example shows how this approach works.
Suppose a TNN query is issued at point \( p \) as shown in Figure 4.4. The Window-Based-TNN-Search algorithm first issues an NN query to retrieve the nearest neighbor \( s_1 \) of \( p \) in \( S \). Then it issues the second NN query to retrieve the nearest neighbor \( r_1 \) of \( s_1 \) in \( R \). The search range is therefore determined to be \( \text{circle}(p, d) \), where \( d = \text{dis}(p, s_1) + \text{dis}(s_1, r_1) \) (the inner circle in Figure 4.4). Then two window queries are issued and objects \( (s_1, s_2, s_3, s_4) \) and \( (r_1, r_2, r_3) \) are scheduled to be retrieved. The transitive distances between \( p \) and different combinations of object pair \( (s, r) \) are calculated and compared. The final answer \( (s_2, r_3) \) with the minimum transitive distance is then output.

The Approximate-TNN-Search algorithm performs an approximate estimate of search range at the estimate phase. This algorithm is very efficient but does not guaranteed to produce the correct answer. The window-based TNN algorithm needs two index traversals to determine the search range. However, the approximate-TNN-Search algorithm saves the two NN queries by determining the search range using the following equation [2]:

\[
r_k(S) = \ln(n) \times \sqrt{\frac{k}{(\pi \times n)}}, \text{ where } n = |S|.
\]

Given a dataset \( S \) in which the points are uniformly distributed in a unit square region, a circle with radius \( r_k(S) \) encloses at least \( k \) objects. Therefore, the radius
of the search range in a TNN query can be derived by \( d = r_k(S) + r_k(R) \), where \( k = 1 \). The radius can be easily scaled to a square of other size. This approach saves two index traversals to allow the algorithm to go directly into the filter phase and only introduces a small computational overhead. However, it does not guarantee to contain the answer in the approximated search range.

### 4.2.2 Deficiencies of Existing Algorithms

The algorithms mentioned above are sequential in nature and do not utilize the fact that the two datasets may be broadcast on two channels simultaneously. Since a mobile device may monitor and download pages from both channels simultaneously, the two NN queries can be processed in parallel. The two range queries in the filter phase can be processed in a similar fashion.

The deficiency of the two algorithms mentioned above are as follows. For the Window-Based-TNN-Search algorithm, the second NN query takes the output of the first NN query as its query point. As a result, it must wait until the first NN query is finished. For the approximate-TNN-Search algorithm, the equation mentioned above only suits datasets where the data points are uniformly distributed. For skewed datasets, this approach fails to determine a search range that guarantees to contain the final answer pair of objects (see Section 6.3). Besides, even with uniformly distributed datasets, the search range provided by the equation is unnecessarily large. More objects are enclosed in this range and more pages have to be downloaded from the broadcast channels, compared to other TNN algorithms. As a result, the tune-in time of the approximate-TNN-Search algorithm is very large even if it avoids the two NN queries in the estimate phase. In next section, we propose new algorithms, namely, *Double-NN-Search* algorithm and *Hybrid-NN-Search* algorithm, which not only guarantee correct answers but also fully exploit the parallel access ability of the mobile device.

### 4.3 New Algorithms for TNN

In this section, we introduce two new algorithms, namely the Double-NN algorithm and the Hybrid-NN algorithm. Both algorithms allow parallel access of broadcast
data in both phases to save access-time. Our algorithms find a pair of objects \((s, r)\) and use the transitive distance determined by \(p\) and \((s, r)\) as the search range.

### 4.3.1 Double-NN-Search Algorithm

The algorithm executes two nearest neighbor queries from the query point \(p\) on the two channels at the earliest opportunity, i.e., as soon as the index roots appear in the two channels. After both NN queries are completed, it then uses \(d = \text{dis}(p, s) + \text{dis}(s, r)\) as the radius of the search range, where \(s = p.\text{NN}(S)\) and \(r = p.\text{NN}(R)\).

In the running example shown in Figure 4.4, Double-NN-Search algorithm retrieves \(s_1\) (i.e. \(p.\text{NN}(\langle S \rangle)\)) and \(r_2\) (i.e. \(p.\text{NN}(\langle R \rangle)\)). Thereafter, the search range is fixed to \(\text{circle}(p, d)\), where \(d = \text{dis}(p, s_1) + \text{dis}(s_1, r_2)\) (the outer circle). Then two range queries are issued to access data in both channels in this range. Objects \((s_1, s_2, s_3, s_4)\) and \((r_1, r_2, r_3)\) are retrieved and the final answer pair \((s_2, r_3)\) is obtained by calculating and comparing the transitive distances. The pseudo code of the Double-NN-Search algorithm is given in Algorithm 6.

### 4.3.2 Hybrid-NN-Search algorithm

In Hybrid-NN-Search algorithm, when the NN search in one of the two channels is completed before the other, we can use the result to guide the search in the unfinished channel in order to find a smaller search range. Before describing the algorithm, first, we define some distance metrics that will be used in our new algorithm and then discuss the Hybrid-NN algorithm in detail.

#### 4.3.2.1 Useful Distance Metrics

The two metrics, introduced in this sub-section, act on two points \((p\) and \(r)\) and a MBR \((M_S)\) of an R-tree node. Both metrics find some transitive distance from point \(p\), to a point on the MBR \(M_S\), then to point \(r\).

**Definition 1 (MinTransDist)**. Given a starting point \(p\), a MBR \(M_S\), an ending point \(r\), \(\text{MinTransDist}(p, M_S, r)\) finds a point \(s\) on MBR \(M_S\) and returns \(\text{dis}(p, s) + \text{dis}(s, r)\).
Algorithm 6 Double-NN-Search Algorithm

\textbf{Input:} query point $p$, R-tree index $S$ for dataset $S$, R-tree index $R$ for dataset $R$

\textbf{Output:} transitive nearest neighbor $(s, r)$ to the query point $p$

\textbf{Procedure:}
1: \textit{candidate set} $S \leftarrow \emptyset$
2: \textit{candidate set} $R \leftarrow \emptyset$
3: $s \leftarrow p.NN(S)$; $r \leftarrow p.NN(R)$; \{these two nearest neighbor queries are processed in parallel\}
4: $d \leftarrow \text{dis}(p, s) + \text{dis}(s, r)$
5: let circle $w$ center at $p$ with radius length $d$
6: \textit{candidate set} $S \leftarrow p.\text{range_query}(S, w)$; \textit{candidate set} $R \leftarrow p.\text{range_query}(R, w)$; \{these two range queries are processed in parallel\}
7: \textbf{for} each object $s_i$ in \textit{candidate set} $S$ \textbf{do}
8: \quad \textbf{if} $\text{dis}(p, s_i) < d$ \textbf{then}
9: \quad \quad \textbf{for} each object $r_j$ in \textit{candidate set} $R$ \textbf{do}
10: \quad \quad \quad \textbf{if} $\text{dis}(p, s_i) + \text{dis}(s_i, r_j) < d$ \textbf{then}
11: \quad \quad \quad \quad $d \leftarrow \text{dis}(p, s_i) + \text{dis}(s_i, r_j)$
12: \quad \quad \quad $s \leftarrow s_i$
13: \quad \quad \quad $r \leftarrow r_j$
14: \quad \quad \textbf{end if}
15: \quad \textbf{end for}
16: \textbf{end if}
17: \textbf{end for}
18: return $(s, r)$;

$s) + \text{dis}(s, r)$, such that for any point $s'$ on MBR $M_S$ other than $s$, $\text{dis}(p, s') + \text{dis}(s', r) \geq \text{MinTransDist}(p, M_S, r)$.

\textit{MinTransDist} gives the minimum possible transitive distance from a point to an MBR then to another point, which can be used as the lower bound of the transitive distance if a point from the MBR is chose. We derive a method to calculate \textit{MinTransDist} as follows. There can be three cases:

1. If line segment $pr$ (line between $p$ and $r$) intersects with MBR $M_S$,
   \begin{equation*}
   \text{MinTransDist}(p, M_S, r) = \text{dis}(p, r).
   \end{equation*}

2. If not in 1, let $\ell_i$ ($1 \leq i \leq 4$) be the four sides of MBR $M_S$, if $p$ and $r$ are on the same side of $\ell_i$, find $r_i'$ which is the symmetric point of $r$ w.r.t. $\ell_i$. If
line segment \( pr_i' \) intersects with \( \ell_i \), then \( \text{MinTransDist}(p, M_S, r) = \min_i \{ \text{dis}(p, r_i') \} \).

3. If not in 1 or 2, let \( v_i \ (1 \leq i \leq 4) \) be the four vertices of MBR \( M_S \),
\[
\text{MinTransDist}(p, M_S, r) = \min_{1 \leq i \leq 4} \{ \text{dis}(p, v_i) + \text{dis}(v_i, r) \}.
\]

**Lemma 1.** The above method gives the correct \( \text{MinTransDist} \) from \( p \) to \( M_S \) to \( r \).

**Proof.** The completeness and the soundness of Case 1 and Case 2 are obvious. In Case 3, an ellipse is determined which uses \( p \) and \( r \) as two foci and the transitive distance as the major axis. Suppose \( v_1 \) gives the minimum transitive distance. \( v_2, v_3, v_4 \) are outside of the ellipse. Since Case 1 and Case 2 does not hold, any points on the four sides of \( M_S \) other than \( v_1 \) are outside of the ellipse. Then any points inside of \( M_S \) are outside of the ellipse. Thus, \( \text{dis}(p, v_1) + \text{dis}(v_1, r) < \text{dis}(p, u) + \text{dis}(u, r) \) \((u \in M_S, u \neq v_1)\). Since \( v_1 \in M_S \), this method gives a tight lower bound. \(\square\)

![Figure 4.5: Distance Metrics](image)

Figure 4.5 gives an example of three minimum transitive distances in each of the three cases.

**Definition 2** (MaxDist). *Given a starting point \( p \), a line segment \( \ell \), an ending point \( r \), \( \text{MaxDist}(p, \ell, r) \) is defined as:

\[
\text{MaxDist}(p, \ell, r) = \max_{i=1,2} \{ \text{dis}(p, v_i) + \text{dis}(v_i, r) \} \tag{4.2}
\]

where \( v_i \ (i = 1, 2) \) are the two end points of line segment \( \ell \).
Lemma 2. given a starting point $p$, a line segment $\ell$ and an ending point $r$, $\text{MaxDist}(p, \ell, r)$ gives a tight upper bound for all the transitive distances from $p$ to any points on $\ell$, to $r$.

Proof. An ellipse is determine, which uses $p$ and $r$ as the foci and $\text{MaxDist}(p, \ell, r)$ as the length of the major axis. One end point of $\ell$ is on the ellipse and the other is either on or inside of it. Therefore all other points $v \in \ell$ are inside of this ellipse, and thus making $\text{dis}(p, v) + \text{dis}(v, r) < \text{MaxDist}(p, \ell, r)$. Since one of the two endpoints provides the transitive distance equal to $\text{MaxDist}(p, \ell, r)$, this upper bound is tight. \hfill \Box

Definition 3 (MinMaxTransDist). Given a starting point $p$, an MBR $M_S$, an ending point $r$, MinMaxTransDist($p$, $M_S$, $r$) is defined as:

$$\text{MinMaxTransDist}(p, M_S, r) = \min_{1 \leq i \leq 4} \{\text{MaxDist}(p, \ell_i, r)\} \quad (4.3)$$

where $\ell_i(1 \leq i \leq 4)$ are the four sides of MBR $M_S$.

Lemma 3. given a starting point $p$, an MBR $M_S$ enclosing a point dataset $S$, an ending point $r$, $\exists s \in S, \text{dis}(p, s) + \text{dis}(s, r) \leq \text{MinMaxTransDist}(p, M_S, r)$

Proof. The definition of $\text{MinMaxTransDist}(p, M_S, r)$ uses the MBR face property[60]. This property means that every face of MBR of an R-tree node contains at least one point in the point dataset. $\forall v \in \ell$ (\ell is a side of MBR $M_S$), the transitive distance $\text{dis}(p, v) + \text{dis}(v, r)$ is bounded by $\text{MaxDist}(p, \ell, r)$, according to Lemma 2. Since $\text{MinMaxTransDist}$ chooses the smallest one among the four sides, it provides a tight upper bound. \hfill \Box

From the previous definitions and lemma, we can deduce that given a starting point $p$, a rectangle $M_S$, an ending point $r$, $\text{MinTransDist}(p, M_S, r)$ and $\text{MinMaxTransDist}(p, M_S, r)$ serve as lower and upper bounds, respectively, of transitive distance from $p$ to $M_S$ to $r$.

4.3.2.2 Algorithm Description

Since datasets of different sizes may be involved in the TNN query, the lengths of the indices for them are different. It is very likely that the two NN search
in the estimate phase will finish at different time. When the NN search in one channel generates the final result before the other one, hybrid algorithm uses this result to guide the unfinished search in the other channel to get a result that can minimize the transitive distance between the three points, in order to get a smaller search range for the next phase. Hybrid-NN algorithm achieves this goal by either switching the query point of the NN search in channel 2 or changing the distance metrics in the search in channel 1. (note that the transitive distance measure is not symmetric with respect to the two channels). In Figure 4.4 if the NN search on dataset $R$ finishes before the NN on dataset $S$, and $r_2$ is returned as the NN to $p$, Hybrid-NN returns $s_3$, which can minimize the transitive distance from $p$ to $s_3$ to $r_2$, instead of $s_1$. Three cases exist and the behavior of Hybrid-NN algorithm is described below:

1. If NN searches in both channels are not finished. Follow the algorithm as Double-NN algorithm.

2. If the NN search in Channel 1 finishes before the NN search in Channel 2, let $s$ be the result of NN search in Channel 1, i.e. $s = p.NN(S)$. Use $s$ to replace $p$ to be the new query point of the NN search in Channel 2 and find $r \in R$, which is the nearest neighbor to $s$ on the remaining portion of the R-tree for dataset $R$.

3. If NN search in Channel 2 finishes before the NN search in Channel 1, let $r$ be the result of NN search in Channel 2, i.e. $r = p.NN(R)$. Change the distance metrics in the NN search in Channel 1. Use $MinTransDist$ and $MinMaxTransDist$ to perform branch-and-bound search on the remaining part of the R-tree for dataset $S$. Find $s \in S$ which gives the minimum transitive distance from $p$ to $s$ to $r$ on the remaining portion of R-tree.

### 4.3.2.3 Updating and Pruning Strategy

Like traditional NN algorithms, Hybrid-NN algorithm keeps two upper bounds for the NN queries on each channel and updates them during the traversal of R-trees. These upper bounds are used to prune the R-tree nodes that do not contain the answer. Hybrid-NN algorithm uses two priority queues to keep track of the R-tree
nodes that should be visited in the two channels. R-tree nodes are pushed into the queues and sorted in ascending order based on their arriving time in the two broadcast channels. Initially, Hybrid-NN algorithm pushes the two R-tree roots into the queues whenever they are available on air.

In Case 2, although the Hybrid-NN algorithm switches the target (the query point) of the NN search, the updating and pruning strategy are the same as traditional NN algorithms. The difference between traditional NN algorithms and Case 2 of Hybrid-NN algorithm lies in how to obtain the initial upper bound. When the NN search in Channel 1 finishes before the NN search in Channel 2, Hybrid-NN algorithm switches the query point of the second NN search to the point returned by the first NN search. If a temporary result is generated in channel 2, the upper bound is set to be the transitive distance from query point \( p \) to its nearest neighbor in the first channel, then to this temporary result. After that, for each node in the priority queue \( MBR\_queue \) for Channel 2, Hybrid-NN computes the \( MinDist \) between the new query point and the children MBRs of this node. The smallest \( MinDist \) is used to update the upper bound if the upper bound is larger than this \( MinDist \).

The upper bound updating strategy for Case 3 uses transitive distance, \( MinMaxTransDist \) and Lemma 3. Initially, it pushes the root of R-tree for dataset \( S \) into \( MBR\_queue \). When it is detected for the first time that Hybrid-NN algorithm is in Case 3, an initial upper bound update is performed. If there is a temporary point \( s' \) given as a potential NN to query point \( p \) in dataset \( S \), Hybrid-NN algorithm uses \( dis(p, s') + dis(s', r) \) to update the old upper bound. Then, since all the MBRs that will be visited are contained in a queue \( MBR\_queue \), for all MBRs \( M_i \),
in \( MBR_{queue} \), Hybrid-NN algorithm computes \( \text{MinMaxTransDist} \) and finds the minimum one:

\[
z = \min_{M_i \in MBR_{queue}} \{\text{MinMaxTransDist}(p, M_i, r)\}
\]

If \( z \) is less than the current upper bound, then update the upper bound.

After the initial upper bound update, Hybrid-NN algorithm proceeds with query processing. When an MBR \( M_S \) of an intermediate node \( N_{inter} \) of the R-tree for dataset \( S \) is being visited, for all the children MBRs enclosed by \( M_S \) \( \{M_{Si} : 1 \leq i \leq |N_{inter}|\} \) (\(|N_{inter}| \) denotes the number of children of node \( N_{inter} \)), Hybrid-NN algorithm computes the \( \text{MinMaxTransDist} \) and finds the minimum one:

\[
z = \min_{1 \leq i \leq |N_{inter}|} \{\text{MinMaxTransDist}(p, M_{Si}, r)\}
\]

The upper bound is updated if and only if \( z \) is smaller than the current upper bound.

When an MBR \( M_S \) of a leaf node \( N_{leaf} \) is being visited, for all points \( \{O_i : 1 \leq i \leq |N_{leaf}|\} \) (\(|N_{leaf}| \) denotes the number of points in it) it encloses, Hybrid-NN algorithm computes the transitive distances and finds the minimum one:

\[
z = \min_{1 \leq i \leq |N_{leaf}|} \{\text{dis}(p, O_i) + \text{dis}(O_i, r)\}
\]

and update upper bound if and only if \( z \) is smaller than the current upper bound.

Pruning strategy uses \( \text{MinTransDist} \). If an MBR \( M_S \) has \( \text{MinTransDist}(p, M_S, r) > \text{upperbound} \), then this MBR is discarded and will not be downloaded from the broadcast channel. Figure 4.6 gives an example of updating and pruning. MBR \( M_{S3} \) has a \( \text{MinMaxTransDist} \) smaller than the old upper bound, then upper bound is updated with its value. MBR \( M_{S1} \) and \( M_{S2} \) has \( \text{MinTransDist} \) larger than the new upper bound. It is guaranteed that they do not contain the final answer, therefore they are pruned.

Algorithm 7 shows the updating and pruning of Hybrid-NN algorithm when Case 3 occurs, given \( r \) as the nearest neighbor of query point \( q \) generated in Channel 2.
Algorithm 7 Hybrid-NN-Search Algorithm (Case 3)

Procedure:
1: initial_upperbound_update();
2: while MBR_queue is not empty do
3: \( M \leftarrow MBR_queue.pop() \);
4: if \( \text{MinTransDist}(p, M, r) > \text{upperbound} \) then
5: continue;
6: else
7: wait until \( M \) is on air;
8: if \( M \) is an intermediate node then
9: for each child MBR \( M_i \) of \( M \) do
10: \( z = \min\{\text{MinMaxTransDist}(p, M_i, r)\} \);
11: if \( \text{upperbound} > z \) then
12: \( \text{upperbound} \leftarrow z \)
13: end if;
14: end for
15: prune children \( M_i \) using \( \text{MinTransDist} \)
16: sort \( M_i \) in ascending order based on arrival time;
17: MBR_queue.push(\( M_i \));
18: else
19: for each point \( s_i \) in \( M \) do
20: \( z = \min\{\text{dis}(p, s_i) + \text{dis}(s_i, r)\} \)
21: if \( \text{upperbound} > z \) then
22: \( \text{upperbound} \leftarrow z \);
23: \( s \leftarrow s_i \)
24: end if
25: end for
26: end if
27: end if
28: end while
29: return \((s, r)\);

4.3.2.4 Adjustments in Implementation

When Hybrid-NN algorithm starts with Case 1, it processes two NN queries in parallel. When an intermediate R-tree node is encountered, traditional NN algorithms first prune the children of this node and then push the remaining children into a priority queue (MBR_queue). One problem with this approach is that when Hybrid-NN algorithm changes the query point or distance metrics to form a new
query, the MBR which contains the answer to that new query may have been pruned. Figure 4.7 gives an example. MBR $M_2$ is pruned during the NN query processing which takes $p$ as the query point. However, $M_2$ may contain the NN to the new query point $r$. In order to remedy this problem, the algorithm delays the pruning process of the NN search. When Hybrid-NN algorithm visits a non-leaf R-tree node, it pushes all the children nodes into MBR_queue. When an node is popped out of the queue, either $\text{MinDist}$ (for Case 1 & 2) or $\text{MinTransDist}$ (for Case 3) is computed and compared with the upper bound. The node is pruned if its distance metric is larger than the upper bound; otherwise, it is visited.

![Figure 4.7: Updating and Pruning](image)

Note that this adjustment does not affect the correctness of the answer if no query point switching or distance metrics changing is made. One concern of this adjustment is that since Hybrid-NN algorithm is pushing more MBRs into the queue than typical NN algorithms, the size of the queue will become larger. However, the increase in queue size is not significant. Let $M$ be the maximum fanout value, and $H$ be the height of the R-tree, the worst-case queue size is $(H - 1) \times (M - 1)$. In our experiment, the R-tree for the dataset containing nearly 100,000 points has $H = 10$ and $M = 3$. Therefore, the mobile device only has to allocate memory space for $9 \times 2 = 18$ MBRs in its queue. This memory consumption is affordable.

## 4.4 Optimization of TNN Algorithms

While our algorithms use exact search to provide a precise search range in order to reduce tune-in time in the filter phase, here we explore an optimization technique by using approximate NN search on R-tree to reduce tune-in time in the estimate
phase (while not introducing much increase in tune-in time in the filter phase). The goal is to reduce the overall tune-in time in two phases.

In an exact nearest neighbor (eNN) search, traditional algorithms prune an R-tree node only when it is guaranteed that this node does not cover the answer of an NN query. When the MBR of an R-tree node is obtained, the minimum distance between this MBR and the query point is computed and compared with the current upper bound. If the former is larger, then it is guaranteed that this node does not contain the answer and thus can be pruned. Otherwise this node has to be visited and pushed into the queue. Approximate nearest neighbor search, on the other hand, aims to find a point that is not too far away from the query point. It relaxes the pruning condition so that more R-tree nodes can be pruned, compared with eNN. An R-tree node satisfying the ANN pruning condition (as will be discussed in the next subsection) is pruned, even if it is possible to contain the exact nearest neighbor to the query point.

When an ANN is used in the estimate phase of TNN query processing, the tune-in time in this phase is reduced since it visits less R-tree nodes compared to eNN. Since ANN does not guarantee to find the best/exact NN to the query point, the point ANN finds is farther to the query point than the exact NN. Thus, the search range determined may be larger than using the eNN and the tune-in time in the filtering Phase is increased. However, the ANN approach represents a new tradeoff between two phases from the previous approach. Our goal is to strive a good balance in this tradeoff in order to reduce the overall tune-in time. Also note that ANN optimization technique does not affect the final answer to the TNN query. Even though an ANN search may produce a larger search range, our algorithm assure that the TNN answer is included in this range (see Theorem 1).

4.4.1 ANN Pruning Condition

ANN estimates the probability that an R-tree node contains the real nearest neighbor to the query point \( p \). During the R-tree traversal of the ANN search, the upper bound is obtained and updated in the same way as in the exact NN search: upper bound is set to be infinity at the beginning of ANN query processing and updated using the value of either the smallest \( \text{MinDist} \) between \( p \) and an MBR or the
smallest distance between \( p \) and any real point objects. ANN uses a new heuristic to estimate the probability that a node contains a point \( q \) such that \( \text{dis}(q, p) \leq \text{current upper bound} \), where \( \text{current upper bound} \) is the latest upper bound of ANN obtained during R-tree traversal. If this probability is smaller than a threshold \( \alpha \), it is pruned; otherwise it is preserved in the queue and waits to be visited.

![Figure 4.8: Overlap Area](image)

The distribution of the children or real objects of an MBR of an R-tree node cannot be obtained unless this node is visited. Since the pruning decision has to be made before visiting a node, our heuristic assumes that the children or real objects are uniformly distributed in the MBR of this node and uses the following method to estimate the probability and do the pruning. An example is given in Figure 4.8a.

**Heuristic 1** (circle-rectangle overlap). *Given a query point \( p \), an MBR \( M \) of an R-tree node, the upper bound upper bound of an ANN query, let \( S_{\text{overlap}} \) denotes the overlap area between \( M \) and a circle which takes \( p \) as the center and upper bound as the radius; \( S_{MBR} \) denotes the area of \( M \). The R-tree node is pruned if \( S_{\text{overlap}}/S_{MBR} \leq \alpha \), where \( \alpha \) is a pre-defined pruning threshold.*

Similarly, approximate search can also be applied to Case 3 of Hybrid-NN algorithm. ANN estimates the probability that an MBR contains a point that gives a transitive distance to \( p \) and \( r \) smaller than the current upper bound. For Case 3 of Hybrid-NN algorithm, we use the following heuristic to do the pruning and Figure 4.8b gives a corresponding example.
Heuristic 2 (ellipse-rectangle overlap). Given a starting point \( p \), an ending point \( r \), an MBR \( M \) of an R-tree node, the upper bound upper_bound of an TNN query in Case 3 of Hybrid-NN algorithm, let \( S_{\text{overlap}} \) denotes the overlap area between \( M \) and an ellipse which takes \( p \) and \( r \) as two foci and upper_bound as the length of the major axis; \( S_{\text{MBR}} \) denotes the area of \( M \). The R-tree node is pruned if \( S_{\text{overlap}}/S_{\text{MBR}} \leq \alpha \), where \( \alpha \) is a pre-defined pruning threshold.

Note that the MBR which gives the latest upper bound has to be preserved and visited even if its ratio between overlap area and area of MBR is smaller than \( \alpha \). Otherwise the ANN algorithm will have a probability that it does not reach any leaf nodes of R-tree and no actual points will be retrieved. Also note that the value of threshold \( \alpha \) varies between 0 and 1. When \( \alpha \) is 0, ANN becomes eNN. A smaller \( \alpha \) introduces limited approximation and may not be helpful to reduce the tune-in time. As \( \alpha \) gets closer to 1, more R-tree nodes satisfy the pruning condition and ANN prunes out more R-tree nodes and the result of ANN becomes further to the query point \( p \); thus results in a large search range and the penalty of tune-in time increase in the filter phase will be dramatic. The value of \( \alpha \) should be chosen carefully. Different factors affecting the decision in choosing an \( \alpha \) is discussed in the following subsection.

4.4.2 Factors Affecting Approximation Quality

The depth of an R-tree node and the height of the R-tree have an impact on the approximation quality. R-tree nodes with a smaller depth(near the root) covers a larger area and contains more data points than nodes with a large depth(near the leaves). Therefore if a node near the root is pruned, the penalty for finding a point close to the query point may be large. If a node near the leaves is pruned, the penalty is small and affordable. A fixed value for \( \alpha \) may not be suitable for all R-tree nodes(as shown in our experiment). In our ANN algorithm, the value of \( \alpha \) is determined dynamically during the traversal of the R-tree. When visiting a node near the root, ANN sets \( \alpha \) close to 0 so that the search is close to eNN; while visiting a node close to leaves, ANN sets \( \alpha \) close to 1 so that it prunes out a large number of nodes to reduce the tune-in time in the estimate phase and not introducing a large increase in the search range. We use the following equation to
determine $\alpha$:

$$\alpha = \frac{Node\_depth}{Rtree\_height} \times factor$$  \quad (4.4)

where $Node\_depth$ is the number of level a node resides from the root and $Rtree\_height$ is the total number of levels an R-tree has. $factor$ is used to adjust the value of $\alpha$ to get the optimal optimization quality. It is determined by whether Double-NN, Window-Based-TNN or Hybrid-NN is used. As shown later in our experiments, for Double-NN and Window-Based-TNN algorithms, $factor = 1$, while for Hybrid-NN algorithm, $factor = 1/150$ or $1/200$.

Different densities of the two datasets involved in TNN query also have an impact on the value of $\alpha$. E.g. the density of dataset $S$ is larger than that of dataset $R$. Since $S$ and $R$ are covering the same region, points in $S$ are closer to each other, while points in $R$ are sparse. Therefore, given the same $\alpha$, ANN on $R$ contributes more to the increase in search range than the ANN on $S$. More points in $S$ will be enclosed in this search range than those in $R$. These points in $S$ will have to be retrieved in the range query in the filter phase. The tune-in time penalty in the range query on $S$ will counteract the reduction in tune-in time in query processing on dataset $R$(as shown in our experiment). Taking into consideration the difference in densities, we set the value of $\alpha$ to be close or equal to 0 for the dataset with a smaller density of the two to reduce the tune-in time penalty in the dataset with a larger density.

### 4.5 Performance Evaluation

In this section, we present the result of experiments evaluating the performance of the Double-NN, Hybrid-NN algorithms and the two algorithms adapted from [2]. Two metrics are used in these experiments - *access time* and *tune-in time*, both measured by the number of pages accessed. The access time is the larger of the access times in both channels and the tune-in time is the sum of two tune-in times in both channels. The datasets used in these experiments include synthetic and real datasets. The points in one set of the synthetic datasets are uniformly distributed in a $39,000 \times 39,000$ square region. Eight datasets are generated for
the first dataset involved in TNN query with densities $10^{-7.0}, 10^{-6.6}, 10^{-6.2}, 10^{-5.8}, 10^{-5.4}, 10^{-5.0}, 10^{-4.6}$ and $10^{-4.2}$, each having 152, 382, 960, 2, 411, 6, 055, 15, 210, 38, 206 and 95, 969 points. Another set of eight uniform datasets are generated as the second dataset, with the same density range and area, but different points from the first set. For the second set of synthetic datasets, 16 datasets having sizes ranging from 2,000 to 30,000 with 2,000 increment. For simplicity, we use $UNIF(E)$ to denote the first synthetic datasets, where $E$ represents a power of ten; and we use the number of points to denote the second synthetic datasets. The real datasets include $CITY$ and $POST$ datasets, both extracted from [71]. The former contains nearly 6,000 cities and villages of Greece in a $39,000 \times 39,000$ square region, while the latter contains more than 100,000 post offices in the northeast of the United States in a $1,000,000 \times 1,000,000$ square region. When datasets with different areas are used, they are scaled to the same area.

In the experiments, R-tree is used as the air index. We arrange the R-tree in a depth-first order in the broadcast channels. The reason is stated in section 2.1. Due to the limited memory size of the mobile client, such a queue may not be possible to maintain. As discussed in Section 2.2, previous R-tree algorithms introduce backtracking in NN query, which deteriorates the performance severely in terms of access time in the broadcast environment due to its linear delivery property. Therefore, we maintain the priority queue of the candidate R-tree nodes according to their arrival time, so that backtracking is avoided. Because the location of the points in all the datasets are known a priori, and no insertion and deletion are involved, we use STR packing algorithm to build the R-tree in order to achieve the best performance [4]. $(1, m)$ interleaving technique [36] is applied to arrange the index and data on both channels.

For each set of experiments, 1,000 query points are generated randomly in the same region as the datasets. We assume that when the mobile client first tune-in to the two broadcast channels, the roots of the two R-tree indices may not be available immediately and the mobile client has to wait to different time periods to get the two roots. Therefore, two random numbers are generated to simulate the waiting time to get the two roots. Also we assume that the computational overheads, including the computing time for the search range determination of Approximate-TNN-Search algorithm, the join step of all the algorithms to find the
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>size of an index pointer</td>
<td>2 bytes</td>
</tr>
<tr>
<td>size of a coordinate</td>
<td>4 bytes</td>
</tr>
<tr>
<td>size of a data content</td>
<td>1k bytes</td>
</tr>
<tr>
<td>page capacity</td>
<td>64 - 512 bytes</td>
</tr>
</tbody>
</table>

Table 4.2: Parameter Setting

answer and ANN pruning condition checking are small and thus can be neglected. Other parameters in our experiments are the same as in [2] and listed in Table 4.2.

### 4.5.1 Algorithms with Exact Search

We evaluate the performance of our proposed algorithms and two existing ones with exact search. For the first set of synthetic datasets, we did experiments for each combination of two datasets with different densities ($8 \times 8 = 64$ sets of experiments). Different pruning strategies are tried out and different values for $\alpha$ are examined. Only part of the results are shown below as representatives due to the space limit. Please see [72] for full sets of experiments and evaluations. The page capacity is set to be 64 bytes.

#### 4.5.1.1 Access Time

The access time of the four algorithms is only affected by the sizes of datasets and visiting orders of TNN queries. Figure 4.9 gives the results of access time of the four algorithms. Among these algorithms, the Approximate-TNN-Search always gives the best performance in terms of access time because it computationally estimates the search range. Therefore, it avoids the two index searches in the first phase and allows the algorithm to go directly into the second phase; whereas the other algorithms have to search on the indices to determine the search range. Note that the Double-NN and Hybrid-NN algorithms always have the same access time because they start receiving and end processing at exactly the same time. Double-NN and Hybrid-NN give a better access time than Window-Based-TNN-Search. Figure 4.9a and Figure 4.9b show the experiment results in which the size of either $S$ or $R$ is fixed to 10,000 points and the sizes of the other datasets varies from
2,000 to 30,000; while Figure 4.9c and Figure 4.9d show the variation of access times in a large range of densities of the two datasets.

Our experiments show that when \( \text{size}(S) \geq \frac{\text{size}(R)}{40} \) and \( \text{size}(S) \leq 1.8 \times \text{size}(R) \), Double-NN and Hybrid-NN give a better performance than Window-Based-TNN in terms of access time. The largest improvement in access time occurs when both datasets have similar sizes and Double-NN and Hybrid-NN reduce the access time by 7% to 15%. When the difference between the sizes of \( S \) and \( R \) becomes larger than the above range, access times of the three algorithms become similar.

When \( \text{size}(S) > 1.8 \times \text{size}(R) \), Window-Based-TNN has a high probability to finish the NN search on Channel 2 before the next root on Channel 1 arrives (on time point b in Figure 4.10a). Then it starts and ends the filter phase at the same time as Double-NN and Hybrid-NN. When \( \text{size}(S) < \text{size}(R)/40 \), Double-NN and Hybrid-NN give a shorter access time than Window-Based-TNN only when the
(a) \( \text{size}(S) > 1.8 \text{size}(R) \)  
(b) \( \text{size}(S) < \frac{\text{size}(R)}{40} \)

Figure 4.10: Access Time Analysis

starting time point of query processing falls in range c in Figure 4.10b. As the difference between sizes of \( R \) and \( S \) increases, this probability reduces. Thus in the above two cases, all three algorithms give similar access time.

Also note that the access time is dominated by the processing time of the larger dataset of the two. This is because access time is the longer of the access times in the two channels and larger dataset usually result in a larger processing time.

4.5.1.2 Tune-in Time

Figure 4.11 gives the tune-in time of Window-Based-TNN, Double-NN and Hybrid-NN algorithms. In each of the four sets of experiments, we fix the size of dataset \( S \) and changes the size of \( R \).

When \( 0.01 \text{size}(R) \leq \text{size}(S) \leq 0.4 \text{size}(R) \), Hybrid-NN gives the best tune-in time among the three algorithms, as shown in Figure 4.11a and 4.11b. In this case, Hybrid-NN algorithm finds a shorter search range than Double-NN and Window-Based-TNN, while taking similar tune-in time in the first phase to estimate the search range, and thus provides the smallest tune-in time.

When \( \text{size}(S) \geq 0.4 \text{size}(R) \), the decrease in tune-in time in filter phase of Hybrid-NN algorithm is countervailed by the increase in tune-in time in the estimate phase to find a short search range, thus rendering the overall tune-in time to be larger than the other two algorithms. Also, in this case, Double-NN and Window-Based-TNN algorithms have similar tune-in times, as shown in Figure 4.11a and 4.11b.

When \( \text{size}(S) < 0.01 \text{size}(R) \), Window-Based-TNN gives the best tune-in time because the search range determined by Window-Based-TNN is smaller than those of the other two algorithms, while they uses similar tune-in times to determine the search range. Also note that as \( \text{size}(S) \) keeps increasing, the tune-in time of Hybrid-NN gets closer to that of Window-Based-TNN. This is because the NN search in Channel 1 has a high probability to finish before the NN search in Channel 2.
Hybrid-NN, in this case, acts similarly to Window-Based-TNN algorithm.

For Approximate-TNN algorithm, although it never fails to generate the correct answer for TNN queries in our experiments, the tune-in time is significantly larger than all the other algorithms. One example is shown in Figure 4.11d. Approximate-TNN algorithm generates an unnecessarily large search range by the equation and thus results in a dramatic increase in the tune-in time in the filter phase. This is especially severe when one of the two datasets is very sparse. A large amount of points on the dense dataset have to be retrieved and the penalty in tune-in time is large.
4.5.2 Optimization

4.5.2.1 ANN vs. eNN

Our optimization technique with approximate-NN search reduces the tune-in time of all the three algorithms. Figure 4.12a gives the result of tune-in time improvement of Window-Based-TNN and Double-NN algorithms, with approximation factor $\text{factor} = 1$. ANN reduces tune-in time in all the four page capacity settings. Due to the space limit, only page capacity of 64 bytes is given in Figure 4.12a. Improvement in tune-in time ranges from 11% to 20%. The two datasets involved have the same size. The impact of difference between sizes are discussed in the following subsection, as well as the Hybrid-NN algorithm, which mainly works on datasets of different sizes.
4.5.2.2 Impact of Sizes of Datasets

If two datasets with different densities use the same strategy (using equation 4.4 with $factor = 1$) to determine the value of threshold $\alpha$, the penalty on the denser dataset of the two will counteract the reduced tune-in time in the estimate phase and thus increases the total tune-in time. For datasets with different densities, we set $\alpha$ of the sparse dataset to be 0 (meaning to do exact NN search) and $\alpha$ of the denser dataset using equation 4.4 with $factor = 1$. Figure 4.12b and 4.12c give the performance of using this strategy to determine the values of $\alpha$. The tune-in time of both Window-Based-TNN and Double-NN algorithms are reduced, no matter which visiting order is taken. Figure 4.12d shows that this strategy also reduces the tune-in time of TNN queries over real datasets. In this set of experiments, $S = CITY$ and $R = POST$, all the four page capacity settings are checked.

For Hybrid-NN algorithm, our experiments show that when $factor = 1/150$ or 1/200, ANN optimization reduces the tune-in time. Figure 4.13 gives the results.

4.5.3 Distribution

The distribution has an important impact on the performance of the existing Approximate-TNN-Search algorithm. Approximate-TNN-Search estimates a correct search range only for uniformly distributed datasets. However, as skewed datasets are involved, the Approximate-NN-Search algorithm does not guarantee to provide an effective search range and therefore fails the TNN query. Table 4.3
<table>
<thead>
<tr>
<th>Density Combination</th>
<th>Average Fail Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>uni-uni</td>
<td>0%</td>
</tr>
<tr>
<td>uni-real</td>
<td>9.08%</td>
</tr>
<tr>
<td>real-uni</td>
<td>9.08%</td>
</tr>
<tr>
<td>real-real</td>
<td>43.2%</td>
</tr>
</tbody>
</table>

Table 4.3: Fail Rate

gives the combination of distributions of datasets and the average fail rate. In *uni-real* and *real-uni* combinations, we use *CITY* dataset and change the eight uniform ones. For *real-real* combination, we use *CITY* and *POST*. We also uses three page capacities - 64 bytes, 128 bytes 256 bytes and 512 bytes, and average fail rates are calculated. Note that Double-NN and Hybrid-NN never fails to provide the correct answer to TNN queries.
Chapter 5

Conclusions and Future Work

In this chapter, we conclude this dissertation and discuss several topics for future research.

5.1 Conclusions

It has been easy and common for users to create and share contents containing geospatial information. A plentiful of such documents are available and accessible from the Internet. They provide a good data source for researches and applications. Meanwhile, the demand for querying spatial information is increasing. However, there remains a huge gap between utilizing the data sources and fulfilling the increasing demand for information. In order to bridge the gap, we must be able to (1) accurately identify, extract and disambiguate geospatial information from user-generated contents; (2) retrieve spatial information/objects efficiently so that meaningful spatial information can be served to users. The first task is very challenging because processing natural language is a difficult problem and user-generated contents usually contain noise. The second task in general is challenging as well since query processing has to take into consideration the specific constraints and query types. On a mobile device, power source, computational power and storage space are all limited, which makes the second problem even worse. In this dissertation, we provide our contributions towards bridging this gap. We focused on the following research problems and provided efficient and effective solutions:
1. Extracting useful information from human-generated route direction documents, namely destinations, origins and route instructions, is difficult. We used both language features and HTML structural and visual features to successfully identified route components and separated them from information irrelevant to route directions.

2. In addition to the sentence level route component extraction, we further studied the problem of destination name extraction. We explored various of machine learning features and compared different models. By using an important route information - arrival information, we improved the accuracy of destination name recognition. Our approach successfully identifies the mentions of destinations from route direction documents.

3. When geo-tagging a road name, a search for a road name in a gazetteer often yields more than one actual roads. In order to disambiguate road names, we annotatively modeled the disambiguation problem to be a hop-constrained shortest path problem on a semi-complete directed $k$-partite graph, in which vertices on the graph are roads, edge weights correspond to the traveling distances to transit from one road to another. In order to handle the noisy environment, we connected each vertex with all vertices in other sets and maximize the number of hops while minimizing the total distance of a path. The algorithm is computationally efficient and robust to wrong names and incomplete gazetteers.

4. Given the multi-channel broadcast environment, we studied a particular spatial query - transitive nearest neighbor query, and developed an query processing algorithm and its optimized version. The algorithm utilizes the parallel signal receiving feature of the mobile device and minimizes the energy cost. It is efficient in terms of both energy consumption and processing time.

The effectiveness of the algorithms and methods proposed above has been proven by experimental results.

### 5.2 Future Work

Our research provides solutions to elemental yet important problems in geospatial information extraction from text and spatial query processing. As for the next steps, one can proceed with the following research problems:
1. Identify road segments in route directions.

Road name extraction and disambiguation produces a sequence of actual roads mentioned in the route directions. However, for each actual road, only a segment belongs to the route. In order to truly recover a route from text form to digital form on a digital map, accurate segmentation on the roads is required. Based on our knowledge, this is a challenging problem because of the following two reasons: (1) Due to the noisy environment, the disambiguation step may still contain wrong roads, or miss some of the roads. Such errors will propagate to the segmentation step. (2) In human-generated route directions, some road names are used as references. For example, a route direction may contain a sentence like this: "If you have seen Road A, you have gone too far" or "...and you will cross over Pearl Road". These roads are not part of the route but rather references for the travelers to locate themselves. Such roads must be identified and separated from the roads which are part of the route.

2. Spatial query processing and route planning using human knowledge.

Human-generated route directions encode the human preferences over different routes. Such route directions may favor one route which is longer but has good road condition and easy to drive, over another route which is shorter but trickier. In addition, user-generated contents, such as Tweets and status updates may contain information on accidents and road condition changes. Such information sources should be utilized when processing spatial queries and planning routes.
A.1 An Example of Sentence Classification

In Table A.1, we show an example of the sentence classification result of a document. The model in this classification task is Maximum Entropy Markov Model. There are a total of 39 sentences in the document and 36 were classified correctly. The wrongly classified sentences are: Sentence Number 32, 35 and 36.

Sentence number 32 is a place name. The appearance of such pattern usually indicates a destination or origin. However, in this case, the previous sentence was labeled as “Origin”, according to the patterns learned before, the sentences following an origin are more likely to be “instruction” than “origin”. Thus, the model gave a higher score to “instruction” and made a wrong classification. Sentence number 35 is an interesting case. Such information does not directly tell users where to go but instead indirectly help users to locate themselves and the destination. They do not appear very often in the documents. Such classification errors can be corrected by adding rules as post-processing steps. Sentence number 36 was classified wrongly for a similar reason as Sentence 32 since “Other” sentences are usually grouped together.
<table>
<thead>
<tr>
<th>number</th>
<th>label</th>
<th>sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Orig.</td>
<td>From points North of Washington, DC</td>
</tr>
<tr>
<td>2</td>
<td>Orig.</td>
<td>From points North of Washington, DC</td>
</tr>
<tr>
<td>3</td>
<td>Inst.</td>
<td>Merge onto CAPITAL BELTWAY/I-495 S.</td>
</tr>
<tr>
<td>4</td>
<td>Inst.</td>
<td>Take the VA-193 exit- EXIT 43-44- toward GEORGE WASHINGTON MEMORIAL PKWY/GEORGETOWN PIKE.</td>
</tr>
<tr>
<td>5</td>
<td>Inst.</td>
<td>Merge onto GEORGE WASHINGTON MEMORIAL PKWY S via EXIT 43.</td>
</tr>
<tr>
<td>6</td>
<td>Inst.</td>
<td>Take the US-29 N exit toward KEY BRIDGE/WASHINGTON.</td>
</tr>
<tr>
<td>7</td>
<td>Inst.</td>
<td>Turn SLIGHT RIGHT onto N LYNN ST/US-29 N.</td>
</tr>
<tr>
<td>8</td>
<td>Inst.</td>
<td>Continue to follow US-29 N.</td>
</tr>
<tr>
<td>9</td>
<td>Inst.</td>
<td>Turn SLIGHT RIGHT onto ramp.</td>
</tr>
<tr>
<td>10</td>
<td>Inst.</td>
<td>Merge onto US-29 N/WHITEHURST FWY.</td>
</tr>
<tr>
<td>11</td>
<td>Inst.</td>
<td>Turn SLIGHT RIGHT onto K ST NW.</td>
</tr>
<tr>
<td>12</td>
<td>Inst.</td>
<td>Turn RIGHT to stay on K ST NW.</td>
</tr>
<tr>
<td>13</td>
<td>Inst.</td>
<td>Turn LEFT to stay on K ST NW.</td>
</tr>
<tr>
<td>14</td>
<td>Inst.</td>
<td>Turn LEFT onto VERMONT AVE NW.</td>
</tr>
<tr>
<td>15</td>
<td>Orig.</td>
<td>From points South of Washington, DC</td>
</tr>
<tr>
<td>16</td>
<td>Inst.</td>
<td>I-95 N toward WASHINGTON.</td>
</tr>
<tr>
<td>17</td>
<td>Inst.</td>
<td>Merge onto I-395 N via EXIT 170A on the LEFT toward WASHINGTON.</td>
</tr>
<tr>
<td>18</td>
<td>Inst.</td>
<td>Merge onto US-1 N via the exit on the LEFT.</td>
</tr>
<tr>
<td>19</td>
<td>Inst.</td>
<td>Stay straight to go onto 14TH ST NW.</td>
</tr>
<tr>
<td>20</td>
<td>Inst.</td>
<td>Turn LEFT onto K ST NW.</td>
</tr>
<tr>
<td>21</td>
<td>Inst.</td>
<td>Turn RIGHT onto VERMONT AVE NW.</td>
</tr>
<tr>
<td>22</td>
<td>Orig.</td>
<td>From points West of Washington, DC</td>
</tr>
<tr>
<td>23</td>
<td>Inst.</td>
<td>I-66 E toward WASHINGTON.</td>
</tr>
<tr>
<td>24</td>
<td>Inst.</td>
<td>Take the E STREET exit on the LEFT toward I-66 E.</td>
</tr>
<tr>
<td>25</td>
<td>Inst.</td>
<td>Take the E STREET ramp.</td>
</tr>
<tr>
<td>26</td>
<td>Inst.</td>
<td>Stay straight to go onto E ST EXPY.</td>
</tr>
<tr>
<td>27</td>
<td>Inst.</td>
<td>Turn SLIGHT LEFT onto E ST NW.</td>
</tr>
<tr>
<td>28</td>
<td>Inst.</td>
<td>Turn LEFT onto 17TH ST NW.</td>
</tr>
<tr>
<td>30</td>
<td>Inst.</td>
<td>Turn LEFT onto VERMONT AVE NW.</td>
</tr>
<tr>
<td>31</td>
<td>Orig.</td>
<td>From the Metro</td>
</tr>
<tr>
<td>32</td>
<td>Inst.</td>
<td>McPherson Square</td>
</tr>
<tr>
<td>33</td>
<td>Inst.</td>
<td>Take the Vermont Avenue exit, walk diagonally through the park, cross K, and pick up Vermont Avenue.</td>
</tr>
<tr>
<td>34</td>
<td>Inst.</td>
<td>Our building will be two blocks down on your left (look for American flags).</td>
</tr>
<tr>
<td>35</td>
<td>Other</td>
<td>If you reach the circle you ve gone too far.</td>
</tr>
<tr>
<td>36</td>
<td>Other</td>
<td>Farragut North</td>
</tr>
<tr>
<td>37</td>
<td>Inst.</td>
<td>Take the L Street exit and turn left from the escalator onto L.</td>
</tr>
<tr>
<td>38</td>
<td>Inst.</td>
<td>Walk down L until you reach Vermont.</td>
</tr>
<tr>
<td>39</td>
<td>Inst.</td>
<td>Our office building will be to your left on that block</td>
</tr>
</tbody>
</table>

Table A.1: MEMM model on sentence classification
Bibliography


Xiao Zhang

Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802

E-mail: xiazhang@cse.psu.edu
Homepage: www.cse.psu.edu/~xiazhang

Education:

- **Ph.D. Degree** (Aug. 06 – Aug. 12) GPA: 3.75/4.0
  Department of Computer Science & Engineering, the Pennsylvania State University, USA
  Dissertation: Route Extraction, Road Name Disambiguation and Efficient Spatial Query Processing under Location Constraints
- **Bachelor’s Degree** (Aug. 02 – July. 06) GPA: 89.3/100, Rank: top 5/280
  Department of Computer Science and Engineering
  Harbin Institute of Technology, Harbin, China

Research Experiences:

- **Penn State University** (Sep. 07 – Sep. 11)
  Research Assistant in GeoCAM project. Doing research in geo-spatial information extraction, geo-spatial disambiguation, text data mining and machine learning.
- **Penn State University** (Aug. 06 – Sep. 07)
  Research Assistant. Doing research in mobile/wireless databases and location based services.
- **Harbin Institute of Technology** (Oct. 05 – July 06)
  Doing research on \( k \)-anonymity and privacy preservation.

Internship Experiences:

- **Google** (May, 10 – Aug., 10) Search Infrastructure Team
- **Facebook** (May, 09 – Aug., 09) Search Team
- **Microsoft Research Asia** (May, 08 – Aug., 08) Web Search and Data Mining group

Teaching Experiences:

- **Penn State University** (Jun. 07 – Aug. 07) Instructor and Teaching Assistant for C++ Matlab
- **Penn State University** (Aug. 07 – Dec. 07) Teaching Assistant for Computer Networks

Selected Publications:

- Xiao Zhang, Baojun Qiu, Prasenjit Mitra, Sen Xu, Alexander Klippel, Alan MacEachren, Disambiguating Road Names in Text Route Descriptions using Exact-All-Hop Shortest Path Algorithm, In *proceedings of European Conference on Artificial Intelligence (ECAI 2012)*
- Xiao Zhang, Prasenjit Mitra, Sen Xu, Anuj R. Jaiswal, Alex Klippel, Alan MacEachren, Extracting Route Directions from Web Pages, *WebDB 2009*