Disambiguating Road Names in Text Route Descriptions using Exact-All-Hop Shortest Path Algorithm

Xiao Zhang[⋆ △] and **Baojun Qiu**[⋆◇] and **Prasenjit Mitra**[⋆†] and **Sen Xu**[‡] and **Alexander Klippel**[‡] and **Alan M. MacEachren**[‡] [1]

Abstract. Automatic extraction and understanding of human-generated route descriptions have been critical to research aiming at understanding human cognition of geospatial information. Among all research issues involved, road name disambiguation is the most important, because one road name can refer to more than one road. Compared with traditional toponym (place name) disambiguation, the challenges of disambiguating road names in human-generated route description are three-fold: (1) the authors may use a wrong or obsolete road name and the gazetteer may have incomplete or out-of-date information; (2) geographic ontologies often used to disambiguate cities or counties do not exist for roads, due to their linear nature and large spatial extent; (3) knowledge of the co-occurrence of road names and other toponyms are difficult to learn due to the difficulty in automatic processing of natural language and lack of external information source of road entities. In this paper, we solve the problem of road name disambiguation in human-generated route descriptions with noise, i.e. in the presence of wrong names and incomplete gazetteer. We model the problem as an Exact-All-Hop Shortest Path problem on a semi-complete directed k-partite graph, and design an efficient algorithm to solve it. Our disambiguation algorithm successfully handles the noisy data and does not require any extra information sources other than the gazetteer. We compared our algorithm with an existing map-based method. Experiment results show that our algorithm significantly outperforms the existing method.

1 Introduction

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 pages of web sites 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 [11, 24, 8, 15]. They have also seen potential application in improving the quality of routes generated by automatic navigation systems [14, 22]. 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.

One obstacle in building such a system is the ambiguities in road names in the text. A road is a unique artificial geographic feature on the earth surface. However, their names are not unique. Multiple roads can share the same road name. In a gazetteer or geographic database, a road is often represented by a sequence of connected line segments and/or curves, such as in OpenStreetMap [13]. Searching a road name can yield more than one such sequence. Ambiguities are often seen in local road names, such as "Main Street" and "Second Street". Ambiguities even exist for interstate highways. For example, "*Interstate 405*" has three disconnected segments[2] on the west coast of the US, one bypass near Seattle, WA, one bypass near Los Angeles, CA and one loop in Portland, OR. Throughout this paper, we use the term "road" for the unique artificial geographic feature, while the term "road name" for the name, which oftentimes is ambiguous, assigned to the road. Road name disambiguation is to find the correct, unique road referred to by the road name in the context.

Road name disambiguation belongs to the scope of toponym (place name) disambiguation. Traditional toponym disambiguation focuses on point or regional geographic features such as city names. However, the unique characteristics of road names make the disambiguation task challenging. Heuristic rules used in existing work [19, 2] do not work on road names: population makes no sense for a road; location qualifiers, such as state names or abbreviations are often missing, e.g., "Atherton St." is used instead of "Atherton St. PA"; "PA 15" can be written as "15". Ontologies of toponyms have been used for disambiguation [4, 23]. 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 [17, 18]. However, it is difficult to find profile pages for all or a majority of the roads of ambiguous road names. For example, the Wikipedia page for "Main Street" only covers a limited number of roads with the name "Main Street"[3], thus limiting its power to be used for disambiguation.

In addition, human-generated route directions introduces a noisy environment. The authors of the directions may use wrong or obsolete road names and the gazetteers are often incomplete. Examples of missing roads in Google Maps can be easily seen in its help forum [4]. OpenStreetMaps [13] consists of user-contributed data and is constantly updated. It is very likely that the search results of a road name do not contain the true road. The presence of inaccurate data and incomplete gazetteer make this problem even more challenging.

In this paper, we present our work in solving the problem of road name disambiguation in human-generated route directions. We

[1] ⋆Department of Computer Science and Engineering, †College of Information Sciences and Technology, ‡Department of Geography,The Pennsylvania State University, △Twitter Inc., 795 Folsom St., San Francisco, CA94107, USA, ◇eBay Inc., 2065 Hamilton Ave, San Jose, CA 95125, email: {xiazhang, bqiu}@cse.psu.edu, pmitra@ist.psu.edu, {senxu, klippel, maceachren}@psu.edu

[2] http://en.wikipedia.org/wiki/Interstate_405
[3] http://en.wikipedia.org/wiki/Main_Street_(disambiguation)
[4] http://www.google.com/support/forum/p/maps

model the problem using a semi-complete directed k-partite graph (defined formally and illustrated in Section 3). The disambiguation problem in noisy environment is then generalized to an Exact-All-Hops Shortest Path (EAHSP) problem on this graph, informally, minimizing the path weight and maximizing the number of hops on this path. Although the general multi-constrained shortest path problem is NP-complete [10], 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. The contributions of our research are listed below:

- To the best of our knowledge, ours is the first work to solve the toponym disambiguation problem given noisy data.
- Our disambiguation algorithm is a computation-based method. We achieved a high F1 score of 82% for retrieval of the correct roads of ambiguous road names, even without any auxiliary information, such as Wikipedia or ontologies, or annotated training data.
- We propose a novel approach to model the ambiguities in sequential toponyms, i.e. using semi-complete k-partite graph; and generalize the disambiguation problem to an exact-all-hop shortest path (EAHSP) problem.
- We propose an efficient algorithm which runs in polynomial time to solve the EAHSP problem on semi-complete k-partite graph.

2 Preliminaries

In human-generated route directions, a complete route description consists of origin, destination and route instructions [24]. A route instruction contains a sequence of road names. The road names, if searched in a gazetteer, often yields more than one roads. For example, in OpenStreetMaps [13], there are four roads with the name "*Atherton*", two in the 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, such information may be mentioned in other web pages and the authors of the direction page assume the readers can infer it. In some cases, the address found in a direction page may be misleading, e.g. 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 computed by existing Shortest Path algorithms. However, as discussed before, the search result of a road name may not include the true road. Such road names in the sequence will severely influence the result of Shortest Path algorithms. Figure 1 gives an example.

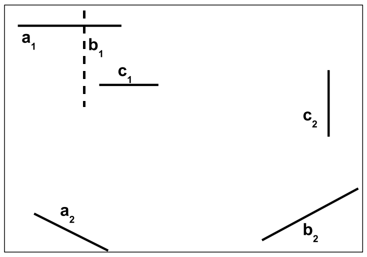


Figure 1*:* Simple methods fail in noisy environment

Three road names A, B and C each yields a set of actual roads in the gazetteer: $\{a_1, a_2\}$, $\{b_2\}$ and $\{c_1, c_2\}$ respectively. The correct roads are $\{a_1, b_1, c_1\}$. If b_1 is missing, the algorithm will select b_2 and c_2, thus missing the true answer c_1. 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 Problem Formalization

We begin with the following definition:

Problem Statement 1. *Given a text route direction and a list of k road names from it $r_1, r_2, ..., r_k$, and a gazetteer, road name disambiguation is to find for each $r_i (1 \leq i \leq k)$ the correct road referred to by r_i from the gazetteer.*

Each 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** of route r is defined as $dist(r) = \sum_{i=1}^{k-1} dist(p_i, p_{i+1})$, where $dist(\cdot,\cdot)$ 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 $seq = (c_1, c_2, ..., c_k)$, a distance threshold d, and a distance function $dist(\cdot,\cdot)$, find a subsequence $seq' = (c_{i_1}, c_{i_2}, ..., c_{i_l})$ and a route $r = (p_1, p_2, ..., p_l)$ of 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.: $dist(r) \leq d$*
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 2 shows an example. A sequence of 4 sets is given as $seq = (P, Q, R, S)$, 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.

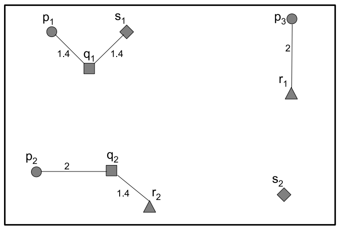


Figure 2*:* A simple example

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 corresponds to the distance between two objects. Thus a sequence is converted to be a semi-complete directed k-partite graph, defined as follows and illustrated in Figure 3:

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.*

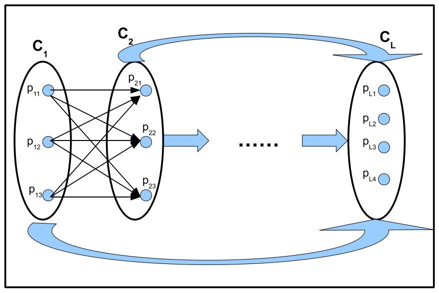


Figure 3*:* graph model

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* $w(\cdot, \cdot) \in \{0\} \cup \Re^+$*, 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).*

4 Related Work

4.1 Geographic Term Disambiguation

Place name disambiguation has been studied extensively. Most methods can be categorized into two groups: rule-based and data-driven [16]. In [20], 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 (details will be shown in Section 6.2). In [4], each place name is mapped onto an ontology [1], the places selected are the ones yielding the maximum conceptual density. Other heuristics include: (a)looking for qualifiers in context, (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 [19, 2]. 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 [21] 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 employed for building the disambiguation models, such as [3, 17, 18]. These methods rely on annotated data sets and/or external information sources. However, such information is extremely difficult to obtain for road entities.

4.2 Hop-constrained Shortest Path Algorithms

Traditional shortest path algorithms, such as Dijkstra and Bellman-Ford algorithms, minimize the path weight only, but not taking into consideration the number of hops. 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, which is known to be NP-complete [10].

A number of special constraints, such as hop count, are amenable to tractable solutions. In [12], 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 [6], 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 [7], the authors introduced the exact all hops shortest path (AHSP) problem. 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.**

4.3 Route Extraction from Text

The problem of automatic route information extraction from text was studied in [24]. However, the authors focused on text information extraction. No actual routes were extracted. In [9], 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.

5 Algorithm Description

5.1 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 $w(\cdot, \cdot)$, where $(u, v) \in E$, returns the non-negative weight of the edge, i.e. $w(\cdot, \cdot) \in \{0\} \cup \Re^+$.

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¿ 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 D_u and S_u.

5.2 Algorithms

Algorithm 1 INITIALIZATION

Input: dag k-partite graph $G = (V, E)$
Output: initialize D_v and S_v for each $v \in V$
Procedure:

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

Algorithm 2 gives the relaxation step on an edge (u, v). It tests whether we can improve the shortest path by one comparison. However, unlike the traditional relaxation technique used by Dijkstra and Bellman-Ford algorithms, 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 2 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
```

Algorithm 3 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 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.

After Algorithm 3 is finished, the min-weight array D_v and successor array S_v are 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 $(0 \leq i < length\_of\_D_v)$; $S_v[i]$ 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_{max} of the qualified entries. Suppose vertex v is such a vertex: $D_v[i_{max}] \leq W$ and i_{max} is the largest index among all indices of qualified entries. The answer to our problem is a path starting from v. The second vertex along the path is given by $v_2 = S_v[i_{max}]$; and the third vertex is $v_3 = S_{v_2}[i_{max} - 1]$. By using the successor arrays, we can easily recover all vertices on the path.

Algorithm 3 exact all-hops shortest path on semi-complete directed k-partite graph

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

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

5.3 Proof of Correctness

Algorithm 3 is a dynamic-programming solution. The algorithm starts with $v \in V_{k-1}$, which only has exact-1-hop paths. Then, for each vertex set V_i with $i < k-1$, 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 now prove the optimal substructure of the exact-i-hop shortest path:

Lemma 1. *Given a semi-complete directed k-partite graph $G = (V, E)$, where $V = \{V_1, V_2, \cdots, V_k\}$, with weight function $w : E \to \{0\} \cup \Re^+$. Let $p_1 =¡v_1, v_2, \cdots, v_h¿$ be the exact-h-hop shortest path starting from v_1, then $p_2 =¡v_2, v_3, \cdots, v_h¿$ 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 $p_2' =< v_2, v_3', \cdots, v_h' > \neq p_2$ is the exact-$(h-1)$-hop shortest path starting from v_2, we can construct another exact-h-hop path $p_1' =< v_1, v_2, v_3', \cdots, v_h' >$ 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. □

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 $w : E \to \{0\} \cup \Re^+$. 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 $\forall v \in V_{i+j}$, $\exists p =¡u, v, \cdots¿$, where the number of hops of p from u

is $m \in [1, (k-i-j+1)]$. Suppose path $p' =¡v, \cdots ¿$ is the exact-$(m-1)$-hop shortest path starting from v, according to Lemma 1, $p = (u,v) + p'$ 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. □

5.4 Time Complexity Analysis

Due to the space limitation, we provide a brief analysis of time complexity. Without loss of generality, we assume that each disjoint vertex set has the same number of vertices, i.e. $|V_1| = |V_2| = ... = |V_k| = n$. The lengths of min-weight array D_v and successor array S_v decreases as vertex set number increases. In V_1, the lengths are k. In V_k, the lengths are 1. The total number of entries in all these arrays is $2kn+2(k-1)n+...+2n = \sum_{i=1}^{k} 2in = k(k+1)n = O(k^2n)$. The initialization step fills in each entry of the two arrays for all vertices. The answer generation process after Algorithm 3 examines each entry in the arrays once. The running time of both are $O(k^2n)$.

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. One relaxation procedure takes only constant time. 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 3 is:

$$\begin{aligned} T &= \sum_{i=k-1}^{1}\left(n \times \left(\sum_{j=1}^{k-i}\big(k-i-j+1\big) \times n\right)\right) \\ &= \sum_{i=k-1}^{1}\left(\frac{(k-i)^2+(k-i)}{2} \times n^2\right) \\ &\overset{(m=k-i)}{=} \sum_{m=1}^{k-1}\left(m^2+m\right) \times \frac{n^2}{2} = O(k^3n^2) \end{aligned} \tag{1}$$

6 Experiments

6.1 Data Collection

The collection of human-generated route directions is built using the method described in [24]. We randomly chose 53 out of 10,000 direction documents, one route description from each document. For each route, we manually extracted the road names in order. We used OpenStreetMap [13] as the gazetteer and search for road names. Table 1 gives the statistics.

6.2 Evaluation Results

A map-based algorithm was proposed in [20] and evaluated in [5]. We compare our disambiguation algorithm (EAHSP) with the map-based algorithm. The map-based algorithm consists of the following procedures: let $t_1, t_2, ..., t_k$ be the k toponyms in the text.

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

number of route descriptions	53
number of road names	202
total number of roads in gazetteer	8464
Average number of roads per name	41.9
Maximum number of roads	704
Minimum number of roads	1

Table 1*:* Statistics

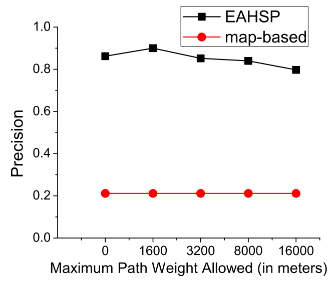
- Calculate the centroid c of all locations in S.
- Remove from S all locations s_i such that the distance between s_i and c is larger than 2σ, where σ is the standard deviation of the set of locations. The remaining locations form a set S'.
- Calculate the centroid c' of all locations in S'.
- For each 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.

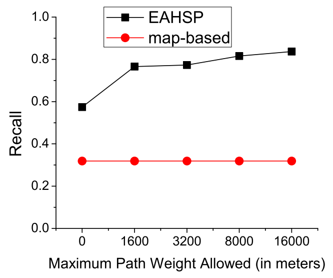
Figure 4 shows the results when the algorithm generates a path with the maximum number of hops; figure 5 shows the results when the algorithm generates a path with one less hop but a smaller path weight. Note the the map-based algorithm remains a straight line in each figure since it is not affected by the maximum allowed path weight.

In the setting where the algorithm finds the longest path p for each route description, under all different values for 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 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 in the gazetteer. When max_weight increases, the ability to tolerate errors increases, therefore the recall increases. The recall is 57.4% when $max\_weight = 0$, but increases immediately to 76.6% when $max\_weight = 1600$, and keeps increasing. Recall of map-based algorithm is only 31.92%. The F1 score of EAHSP when $max\_weight = 0$ is 69.0%, while under other max_weight values, EAHSP algorithm achieves a high F1 score from 81.0% to 82.8%; while the f1 score of map-based is only 25.4%.

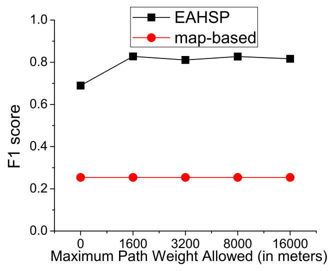
In the setting where the algorithm finds p', the second longest but with smaller weight than p, the performance of EAHSP is sensitive to the value of max_weight. When $max\_weight = 0$, since the path cannot be a strict subsequence of p, it is forced to select other vertices and pushed away from the correct roads. 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. However, as max_weight increases, p' has more overlapping vertices with p, therefore, the performance improves.



(a) Precision

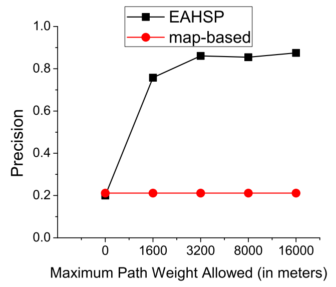


(b) Recall

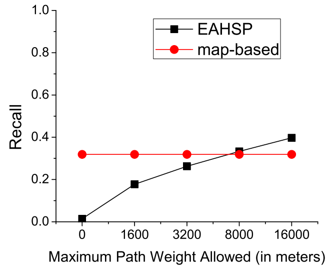


(c) F1 Score

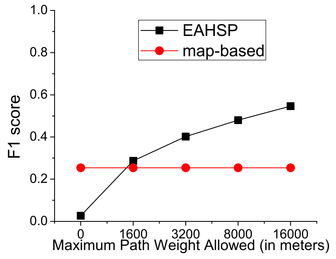
Figure 4*:* Paths with the Largest Number of Hops



(a) Precision



(b) Recall



(c) F1 Score

Figure 5*:* Paths with the Second Largest Number of Hops

7 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 problem in a noisy environment. Based on the heuristic that the correct road of a road name is spatially close to the 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 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. 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. Our final goal is to truly recover a route description from text form to digital maps.

REFERENCES

[1] *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[2] E. Amitay, N. Har'El, R. Sivan, and A. Soffer. Web-a-where: geotagging web content. SIGIR '04, 2004.

[3] R. Bunescu and M. Pasca. Using Encyclopedic Knowledge for Named Entity Disambiguation. In *Proceedings of EACL-06*.

[4] D. Buscaldi and P. Rosso. A conceptual density-based approach for the disambiguation of toponyms. *Int. J. Geogr. Inf. Sci.*, 2008.

[5] D. Buscaldi and P. Rosso. Map-based vs. knowledge-based toponym disambiguation. GIR '08, 2008.

[6] G. Cheng and N. Ansari. Finding all hops k-shortest paths. In *PACRIM*, 2003.

[7] G. Cheng and N. Ansari. Finding all hops shortest paths. *Communications Letters, IEEE*, 2004.

[8] M. Denis, F. Pazzaglia, C. Cornoldi, and L. Bertolo. Spatial discourse and navigation: an analysis of route directions in the city of Venice. *Applied Cognitive Psychology*, 1999.

[9] E. Drymonas and D. Pfoser. Geospatial route extraction from texts. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Data Mining for Geoinformatics*, 2010.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.

[11] R. G. Golledge. Human wayfinding and cognitive maps. 1999.

[12] R. Guerin and A. Orda. Computing shortest paths for any number of hops. *Networking, IEEE/ACM Transactions on*, 2002.

[13] M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE*, 2008.

[14] S. Hirtle, K.-F. Richter, S. Srinivas, and R. Firth. This is the tricky part: When directions become difficult. 2010.

[15] K. S. Hornsby and N. Li. Conceptual Framework for Modeling Dynamic Paths from Natural Language Expressions. *GIS*, 2009.

[16] S. Overell. *Geographic Information Retrieval:Classification, Disambiguation and Modelling*. PhD thesis, 2009.

[17] S. Overell and S. Rüger. Using co-occurrence models for placename disambiguation. *Int. J. Geogr. Inf. Sci.*, 2008.

[18] T. Qin, R. Xiao, L. Fang, X. Xie, and L. Zhang. An efficient location extraction algorithm by leveraging web contextual information. GIS, 2010.

[19] E. Rauch, M. Bukatin, and K. Baker. A confidence-based framework for disambiguating geographic terms. HLT-NAACL-GEOREF '03, 2003.

[20] D. A. Smith and G. Crane. Disambiguating geographic names in a historical digital library. ECDL '01, 2001.

[21] D. A. Smith and G. S. Mann. Bootstrapping toponym classifiers. HLT-NAACL-GEOREF '03, 2003.

[22] T. Tenbrink and S. Winter. Variable granularity in route directions. *Spatial Cognition and Computation: An Interdisciplinary Journal*, 2009.

[23] R. Volz, J. Kleb, and W. Mueller. Towards ontology-based disambiguation of geographical identifiers. In *WWW2007 Workshop I3*.

[24] X. Zhang, P. Mitra, A. Klippel, and A. MacEachren. Automatic extraction of destinations, origins and route parts from human generated route directions. GIScience, 2010.