Easiest-to-Reach Neighbor Search

Jie Shao[†], Lars Kulik[†♯], Egemen Tanin[†♯]
[†]Department of Computer Science and Software Engineering, University of Melbourne
[♯]NICTA Victoria Research Laboratory
{jsh,lars,egemen}@csse.unimelb.edu.au

ABSTRACT

Studies in cognitive science have shown that people have different optimization goals in mind for route selection: beyond shortest travel distance (or time), criteria such as smallest number of turns or straightest path are often considered. A common query that a traveller in a foreign city may ask is "where is a facility of type X". When multiple facilities of the same type are available in the nearby area, usually not the nearest neighbor but the one which is easiest to find is preferred for giving instructions by locals, especially in an unfamiliar and complex urban environment. This paper studies a novel type of neighboring object selection problem, taking cognitive complexity of navigation into account. The main difficulty arises from incorporating spatial chunking and landmark information into neighbor comparisons. We propose an algorithm based on network expansion, which uses incremental processing of graph transformation that models instruction complexity. Our approach can efficiently find the easiest-to-reach neighbor with the guaranteed smallest navigation cost. Through experimental evaluation on real road networks, the performance of the proposed algorithm is demonstrated under various settings. Our comparison results reveal that on average the travel distance of the easiest-to-reach neighbor is only 19.3% longer than that of the nearest neighbor, whereas the navigation cost can achieve a 64.8% reduction.

Categories and Subject Descriptors: H.2.8 [Database Applications]: Spatial databases and GIS

General Terms: Algorithms, Experimentation, Human Factors

Keywords: nearest neighbor, easiest-to-reach neighbor, navigation cost, spatial chunking, landmarks

1. INTRODUCTION

The design of most current navigation systems relies on the calculation of *shortest travel distance* (or time) in a network. However, numerous papers in spatial cognition (e.g., [4, 7, 9, 16, 21]) have shown that people use more than distance as the optimization goal for route selection. Other criteria such as smallest number of turns or straightest path also might play an important role in the process of route planning.

ACM GIS '10 November 2-5 2010. San Jose, CA, USA

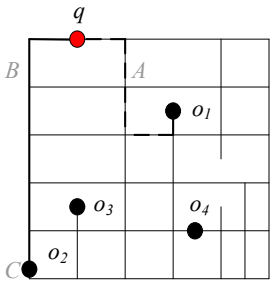


Figure 1: Nearest neighbor o_1 (reachable via the dashed thick path) and easiest-to-reach neighbor o_2 (reachable via the solid thick path) of the query location q in a network.

Imagine a mobile or Web-based location-based service that can apply cognitive principles for generating navigation instructions. The service should be able to answer a typical spatial query that a traveller asks: "where is a facility of type X?". When there are multiple facility instances available, in an unfamiliar and complex urban environment not the nearest neighbor but the *easiest-to-reach* neighbor might be preferred for giving instructions. Without yet formally defining the easiest-to-reach neighbor, we first give an example in Figure 1. q represents the query location with four data objects of the requested type o_1, o_2, o_3 and o_4 nearby. Existing nearest neighbor search algorithms such as [3, 10, 14, 18, 24] find that o_1 has the shortest network distance from q.

Navigation service communicates *how to reach a destination* to users by providing sequences of instructions as *route directions*. At each intersection, whether continue straight or to turn has to be decided. These instructions guide a user from one decision point to the next. For example, the route directions for reaching o_1 could be *turn right {onto street A}, turn left at the second intersection, and then turn first left*, while the route directions for reaching o_2 could be *turn left {onto street B}, go straight until landmark C* (here "{}" denotes additional describing elements which can be dropped). Since a smaller number of turning instructions can reduce the cognitive effort as well as possible navigational errors, o_2 is regarded as a better choice here compared to o_1 in terms of navigation complexity. To assess costs in such a setting, we need to examine two issues, namely, **spatial chunking** and **landmarks**.

In the conceptualization process of a route, instead of giving instructions at every single decision point (intersection), spatial chunking groups multiple instructions of the same type (e.g., continue straight until a turning sign appears) [13]. This is based on established principles used from a cognitive perspective for changing granularity in route directions [12]. By combining elementary route information into higher order elements, neighbor descriptions can

be simplified. In the example of Figure 1, it can be seen that the reduction of instruction complexity for describing how to reach o_1 or o_2 from q corresponds to the number and type of intersections chunked. It also implies that the total instruction complexity could be much smaller than the sum of each individual instruction.

Besides chunking consecutive decision points, landmarks can be incorporated to achieve better route directions (e.g., turn right at the 7-Eleven). Compared to turn-by-turn instructions with street names (predominantly adopted in current in-car navigation systems and online map services), a range of studies in human factors and ergonomics have empirically demonstrated the beneficial effects of incorporating landmarks into vehicle and pedestrian navigation instructions [8, 19, 23], such as improved navigation performance and confidence and reduced cognitive effort. Landmarks are also frequently referred to in human communication of routes and spatial reasoning. Furthermore, future navigation systems can be made more effective and safer, by using landmarks as key navigation cues into automatically generated route directions [6].

In contrast to previous research that solves the route selection problem of finding an optimum route in cognitive sense *for a given pair of source and destination* (e.g., [4, 9]), this paper studies a novel type of neighboring object selection problem *when there are multiple choices as the destination* in the area. In addition, we incorporate spatial chunking and landmark information which are not adequately addressed before in route directions, to resemble human interpretation of instructions. By applying a set of chunking rules to route direction elements, we can assess navigation costs from a cognitive perspective, and formulate a query that finds the easiest-to-reach neighbor.

In a database context, to search for the easiest-to-reach neighbor with the smallest navigation cost it is unnecessary to first compute the optimal descriptions of all neighbors and then compare them in order to find one as the best choice. Given a path network (e.g., the road network of a city) with pre-defined landmarks annotated on it, the principle similar to that of Incremental Network Expansion (INE) [18] can be adopted, to perform node expansion starting from query location on a locally transformed "dual" graph [25, 26]: the edges are treated as nodes, and intersections as edges where each of the weights represents the instruction complexity. Instead of precomputing the costs of all turns in a path network, we incrementally transform a graph. Our core idea is that during the greedy process of expansion, the cognitive principles of *decision point complexity*, *spatial chunking* and *reference to landmarks* can be synthetically taken into account to dynamically label the nodes. Our approach can locate promising data objects earlier, and efficiently find the choice with the guaranteed smallest navigation cost. Some extensions of the proposed algorithm are also discussed. Since the cost function is adaptive to different requirements, our approach can be tailored to user preferences. Finally, we conduct a systematic experimental evaluation on real road networks, which well represent typical urban areas. The performance of the proposed algorithm is extensively studied under various parameter settings.

The main contributions of our work are as follows:

- We introduce and solve a new type of spatial query. Finding the easiest-to-reach neighbor provides new features for advanced navigational assistance.
- We devise a model that computes instruction complexity on-the-fly. This strategy not only reduces processing costs significantly but also makes it feasible to incorporate landmarks into instructions.
- We compare our algorithm against existing work and show that, in return for slightly longer travel distances, easiest-to-reach neighbors offer considerable advantages in terms of their ease of navigation.

2. RELATED WORK

Generating navigation instructions consists of two cognitive steps [15]: first, a route is *selected* for a pair of source and destination; then, its conceptualization is transformed to a sequence of instructions to *present* the information.

2.1 Route Selection

The traditional approach is to apply Dijkstra's algorithm (or a variant) on a graph representation of a given geometric path network to find the shortest path. The distances of nodes are used as travel costs. It is also possible to substitute the distance with time [2, 11]. However, the "cheapest" path can also be found according to some cost function using the cognitive aspects rather than travel distance or time.

Duckham and Kulik [4] proposed an algorithm to compute *simplest path* that minimizes the *complexity of instructions*. The idea of classifying different intersections was inspired by Mark [16]. Since the storage of edge-edge relations is required to allow individual weighting of each intersection for measuring the total instruction complexity, routing is not performed on the original geometric graph, but on a dual graph which models decision point complexity. Afterwards, shortest-path calculation can be applied on such a transformed graph to preferentially select a route through intersections that can be described using less complex instructions. An interesting finding of the experiments in [4] is that, simplest paths which completely rely on the measure of instruction complexity are on average only 16% longer than shortest paths.

Haque *et. al.* [9] proposed an algorithm to compute the *most reliable path*, defined as the one with the smallest *intersection ambiguities*. Analogous to the instruction complexity, the possible ambiguity of each intersection can be inferred from its *degree of connectivity*. For example, travellers are more likely to get lost at a 5-way intersection than at a 4-way intersection. Unreliability of a route is the total ambiguity of intersections that the navigator encounters. The rationale behind is that traversing a more reliable path leads to fewer navigational errors, which in turn may reduce the travel distance in practice (avoiding re-orientations).

Although these algorithms use different cost functions, their routing procedures are similar since they use only specific weights (representing the complexities/ambiguities of intersections) and there is no longer reference to geometric distance information after dual modelling. However, so far spatial chunking and landmarks have not been adequately addressed. They both have a potential to lead to much less cognitive effort required to follow route directions.

Our work emphasizes on finding the best choice for navigation in respect of cognitive complexity when multiple neighboring objects are available[1]. This *destination choice* problem complements the prior studies finding the optimum route in cognitive sense for a given pair of source and destination. To some extend, the difference here resembles the relationship between shortest-path calculation and nearest neighbor search problems in networks.

2.2 Context-Specific Route Presentation

The presentations of previously selected routes can be simplified by taking current surrounding environment into account [22]. While one-to-one relations between decision point/action pair and

[1]Or to be general, if more than one choice is needed for browsing, finding the top-k easiest-to-reach neighbors ranked in ascending order of navigation cost.

instruction represent a low granularity, a high granularity stands for a many-to-one relation expressed by one instruction covering multiple decision points of a route. The different granularity levels are produced by applying chunking rules to route direction elements [12, 13]. As we shall elaborate in Section 4.1, multiple actions at decision points can be grouped into higher order route direction elements according to numerical and structural chunking, as well as landmark information [20].

Landmarks can be broadly defined as *external reference points that are potentially useful as navigation cues*. Particularly, we distinguish between *local* and *global* landmarks. This corresponds to the distinction made between specifying a specific route (local landmark) and specifying a spot on the way to a destination but without requiring a traveller to reach this spot via a specific route (global landmark). For example, chunkings based on distant but well-recognizable landmarks (e.g., turn right at the skyscraper) provide a kind of overall guidance. [17] shows that landmarks are selected for route directions preferably at decision points. We focus on point-like landmarks located at intersections where travellers have to turn, or along route segments for confirmation. For simplicity, linear landmarks spreading along a route (e.g., follow the river) or areal landmarks (e.g., across the park) are not considered in this study.

2.3 Nearest Neighbor Search in Networks

Papadias *et al.* [18] introduced two frameworks for k nearest neighbor search in spatial networks. The Incremental Euclidean Restriction (IER) approach applies the property that the Euclidean distance between two nodes is a lower bound of their network distance for search space pruning. The Incremental Network Expansion (INE) approach performs network expansion similar to Dijkstra's algorithm from query point and examines data objects in the order they are encountered. They showed that in general INE performs better than IER. As an optimization of IER, Deng *et al.* [3] proposed incremental Lower Bound Constraint (LBC). Kolahdouzan and Shahabi [14] presented a Voronoi-based Network Nearest Neighbor (VN3) approach. VN3 divides data space into the first-order network Voronoi diagram with respect to data objects. Finding the k nearest neighbors is done by identifying the first nearest neighbor using the Voronoi diagram, and deriving the subsequent nearest neighbors from adjacent Voronoi cells. Other nearest neighbor search algorithms based on pre-computation include techniques that use pre-computed shortest-path information stored in quadtrees [24] or grid-based data structures [10].

Unlike the above existing work, we study a different problem that considers cognitive complexity of navigation. The weights of network edges representing instruction complexities are computed on-the-fly. Only the principle similar to that of INE which expands network search towards data objects most likely to be in the final solution is borrowed here. In the experimental evaluation, we show the comparisons of travel distances and navigation costs of the easiest-to-reach neighbor and the nearest neighbor.

3. INSTRUCTION COMPLEXITY AND MODELLING

We assume a network contains a set O of static data objects, all representing a particular type of facility instances. A network is traditionally represented by a (connected and simple) graph $G = (N, E)$, where N is the set of nodes, and E is the set of edges (without restriction of generality, the direction of edges is ignored). G is normally sparse due to a small number of branches at an intersection, which implies that $|E| = O(|N|)$. This weighted graph

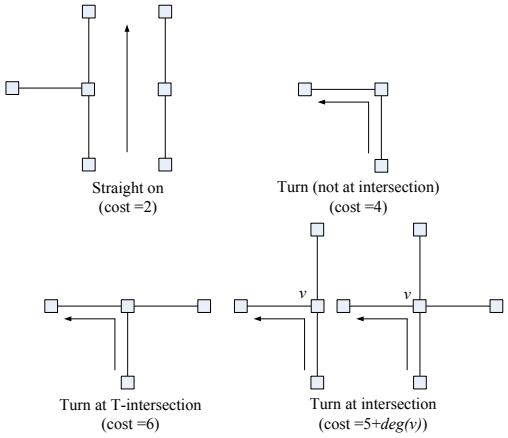


Figure 2: Weighting of decision point complexity. $deg(v)$ is the degree of a node v (i.e., the number of branches meeting at this intersection).

additionally has a cost function $\psi : E \rightarrow \mathbb{R}^+$ that maps each edge $e \in E$ to a nonnegative number. The network distance $d_G(u, v)$ between two nodes u and v is defined to be the length (i.e., sum of the costs) of the shortest path connecting them. To search for the easiest-to-reach neighbor which has the smallest navigation cost, a (cognitive) model is needed for instruction complexity.

Each neighbor description is built up from a set I of possible instructions (such as *turn left*, *turn right*, and *go straight*). Each instruction that guides a user to leave one edge e and enter the next edge e' in G has a cost associated with it using the function $\omega : I \rightarrow \mathbb{R}^+$. Potentially any meaningful cost value can be assigned as long as more complex decision points are associated with larger costs. In the experimental evaluation we adopt the weighting scheme shown in Figure 2 adapted from [4], which is based on a cognitive model to reflect the amount of information needed to successfully negotiate different decision points. In short, the cost function is not an attribute attached to individual edges or nodes in graph, but associated with *pairs of adjacent edges* (if G would be directed, the cost function should be associated with *pairs of consecutive edges* instead) to represent the **negotiation cost** of a decision point. Since in a realistic setting travellers have to reassure that they are still on track, we further assign the **traversal cost** of each route segment (or equivalently, passing a decision point) to be at least a positive number (e.g., we set this value to be two), even if just moving straight on. Thus, this model emphasizes the effect of both the overall *number* of decision points and the *type* of intersections in measuring the navigation cost of reaching a neighbor.

In addition, as aforementioned instructions may also include references to landmarks. For instructions that refer to landmarks from a pre-defined set[2], each landmark has a value which relates to a few factors (such as distance and orientation of a traveller when approaching, and saliency of the landmark itself). Thus, each instruction including a landmark also has a cost associated with it using the function $w : I \rightarrow \mathbb{R}^+$, which models the cognitive effort of its execution. Generally, small costs are assigned to landmarks with high information content and large costs are assigned to less prominent landmarks. Here, we use the similar weighting strategy

[2]Our work is concerned with incorporating landmarks in route directions, rather than extracting landmarks from databases like some other studies such as [5]. Therefore, we assume that a set of potentially useful landmarks is given a priori with their positions and characteristics available.

described in [1] to assign such a cost value w_i of a specific landmark with respect to the node of traveller's position:

$$w_i = a \cdot Distance + b \cdot Orientation + c \cdot Salience$$

where a, b, and c are the preference parameters when looking for landmarks, $Distance$ and $Orientation$ values are given by the spatial configuration and $Saliency$ value is derived from the inherent characteristics of the landmark itself.

In essence, each common node shared by a pair of adjacent edges (e, e') in E can be labelled with a cost, which models the instruction complexity when leaving one edge e and entering the other edge e'. Note that, each instruction may be valid for different pairs of adjacent edges, and *each pair of adjacent edges could have more than one instruction associated with it*. For example, the instructions "turn right at the intersection" and "turn right at the 7-Eleven" might both encode the equivalent action at a particular decision point. Different instructions may have different costs. However, when we have several instructions (including the possibility of using landmarks in the instructions) to describe how to reach the next decision point from the current one, *the minimum navigation cost is determined by the instruction with the smallest cost*. From a cognitive perspective, the cost w_i of a turning instruction incorporating a landmark is generally smaller than the cost ω_i of a plain instruction that encodes the equivalent action. Thus, the parameters a, b, and c should be calibrated to properly assign cost values.

Modelling instruction complexity was previously treated as constructing an evaluation mapping of dual graph $G' = (E', \xi)$ from the original node-edge graph $G = (N, E)$ in [4, 25, 26]. E' is the set of edges in G and ξ is the set of pairs of adjacent edges, with each weight representing instruction complexity. However, we cannot follow such a dual modelling to weight the combination of pairs of adjacent edges for solving our problem, for two reasons:

- First, all the edge-edge relations need to be enumerated to make the dual construct of the whole graph available. In other words, the individual weighting of each pair of adjacent edges in the network has to be pre-computed. For each node n_i in G, there are $m(m-1)/2$ pairs, with m being the degree of n_i (as a rough estimation we can make a reasonable assumption about the mean degrees to be 3 in networks in the geographical domain). It incurs a high cost to compute and maintain such a large number of weights.

- Second, using landmarks in instructions will not be easily possible. When reference to landmarks is considered, the cost assignment process critically depends on the distance between the node representing a traveller's position and a landmark, as well as the orientation of the traveller with respect to the landmark when approaching. This process is intrinsically query-dependent and as a result, cannot be easily materialized.

Since easiest-to-reach neighbors are generally close to query locations, usually only a small portion of the network is relevant to query processing. In fact rather than assigning costs in advance, we can compute instruction complexity on-the-fly from geometry and topology. For real path networks (not necessarily like the rectangular block structure as shown in Figure 1), the degree of each node is first computed. Normally, a turn is perceived as an enforced deviation from the geodesic line. Thus, the set of possible instructions for decision points can be inferred from the angular deviations from the previous movement which are then matched to categories of angular intervals. For example, only the movement with angular deviation less than 12° is deemed as continuing straight. Other choices for angular intervals here are possible as well. Through such an *incremental materialization* strategy to obtain the relevant part of a transformed graph, we can follow the network from the query location q and expand further in a similar fashion to Dijkstra's algorithm (a greedy process still yielding the global optimum). Data objects are inspected in the order they are encountered. Since numerical, structural and landmark chunkings impose multiple complex issues onto the process, we introduce a set of chunking rules employed in this study before presenting easiest-to-reach neighbor search.

4. NAVIGATION COST WITH CHUNKING

The hierarchical organization of spatial information and the ability to change between different granularity levels are important characteristics of the cognitive organization of spatial knowledge. In this section, we elaborate some popular means of chunking consecutive decision points into higher order route direction elements, which have direct influences on measuring navigation costs.

4.1 Chunking Rules

Numerical Chunking: Numerical chunking characterizes the grouping of actions at decision points by counting them and summarizing them as a single instruction.
Example: *Go straight at the first and second intersections, and then turn left ⇒ Turn left at the third intersection*.
Example: *Turn first left, and then turn left again ⇒ Turn left twice*.

Structural Chunking: Salient structural characteristics of intersections or other environmental elements allow identifying these locations uniquely. Within the context of a specific route, some intersections can have highly salient features, especially when they are complex, enforce a change in the movement or even block it.
Example: *Turn left at the T-intersection* (when a T-intersection is reached at the end of a road from the "body" of the T, a turning instruction is mandatory so it marks the end of a chunk).

Landmark Chunking: Landmarks located along a route can be used to chunk certain parts of the route. Such landmarks are considered point-like if they are located at a specific spot along the route (e.g., an intersection), and are only functionally relevant for this spot. It could chunk all straight following route segments until a specific action is required at the end of chunk. The principle is similar to structural chunking.
Example: *Turn left at the church*.

Sometimes if a landmark is well known or has good visibility, it potentially allows chunking large parts of a route without the need of mentioning actions to be taken at the intermediate decision points in-between start and end of the chunk. This kind of chunkings is termed *global chunking*. By using global landmarks, a route does not always have to be fully specified and individual decision point/action pairs may be no longer identifiable. On the contrary, in the case that part of a route is chunked by local landmarks, the involved decision point/action pairs are implicitly represented but still identifiable.

4.2 Measuring Navigation Costs

In Section 3, we have mentioned that the navigation cost of reaching a neighbor is constituted of negotiation cost and traversal cost. Let $\{s = n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow n_m = d\}$ represent a path which passes through a sequence of nodes where $e(n_i, n_{i+1}) \in E$, $i = 1, \ldots, m-1$. Assume we have the relevant part of the transformed graph, which gives the complexity of each involved instruction ω_i. We introduce two policies in the measure of navigation cost $NC(s \rightsquigarrow d)$, which can assess the cognitive complexity of navigation it takes to travel along the path $(s \rightsquigarrow d)$.

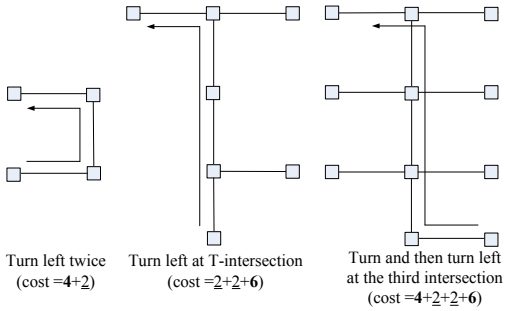


Figure 3: Examples of how navigation costs are measured when chunking rules are applied (the part in bold is negotiation cost and the part with underline is traversal cost).

POLICY 1. *When numerical chunking is applied to group actions at multiple decision points, besides the first decision point the negotiation costs for other decision points in the chunk are no longer evaluated. However, since a minimum traversal cost of each route segment is enforced, the navigation cost is always increased by the value of traversal cost when passing more decision points.*

POLICY 2. *A chunk cannot be arbitrarily long unless a structural feature or a landmark unambiguously marks its end. For structural chunking or landmark chunking, the navigation cost is determined by the instruction complexity at the final decision point at the end of the chunk, plus the cumulative traversal cost that relates to how many decision points have been chunked.*

Figure 3 shows three examples of measuring the navigation costs when chunking rules are applied (the weighting scheme of decision point complexity follows Figure 2). The first example illustrates Policy 1 and the second example illustrates Policy 2. As we can see from the third example, navigation costs should be measured by properly applying the above two policies together to deal with different chunkings. Compared to simply computing the sum of individual decision points $\sum_{i=1}^{m-1} \omega_i$, these two policies enable us to better assess costs from a cognitive perspective. Note that we assume when chunking rules are applied, navigation cost is monotonically increasing when passing more decision points. This excludes the use of global chunking, in particular, the use of global landmarks in route directions.

4.3 Problem Formulation

With the measure of navigation cost, we are able to formally define our neighboring object selection problem (for generality, a version of finding the top-k choices is stated below).

DEFINITION 1 (k EASIEST-TO-REACH NEIGHBOR SEARCH). *Considering a set of data objects $O = \{o_1, o_2, \ldots, o_n\}$ and a query location q on a path network. A k easiest-to-reach neighbor search is defined as a query that finds a subset $O' \subseteq O$ of k objects with minimum navigation cost to reach from q, i.e., for any object $o' \in O'$ and $o \in O - O'$, $NC(q \rightsquigarrow o') \leq NC(q \rightsquigarrow o)$.*

5. SEARCH ALGORITHM

Given a path network annotated with additional information of landmarks, this section presents a computationally efficient algorithm for easiest-to-reach neighbor search. Our approach combines the considerations behind both route selection in cognitive sense and context-specific route presentation. The proposed algorithm favors a route with the smallest cost in terms of total instruction complexity. Meanwhile, it generates the best presentation of a route in light of the employed chunking rules. Here, the incremental processing of graph transformation is used to model instruction complexity. The algorithm is essentially a single source algorithm that performs node expansion starting from query location q, and inspects data objects in the order they are encountered. Since chunkings leave open more options to describe a neighbor which needs to be carefully considered, the challenge of the algorithm design lies in how to efficiently find a data object while guaranteeing that we minimize the navigation cost of reaching it.

To sum up, two most fundamental distinctions of our algorithm from the conventional INE for nearest neighbor search are:

- It works on a transformed graph modelling instruction complexity (where nodes are dynamically labelled by several cognitive principles), rather than geometric distance information (where edge weights are fixed), to evaluate the navigation cost of reaching a neighbor.
- The predecessor instruction of each step is recorded during the expansion, to apply chunk validity check for adapting the action to be taken to its context. This is crucial for computing the smallest cost of a node expansion.

5.1 Chunk Validity Check

Grouping of instructions performed in spatial chunking could be handled by introducing virtual edges in a graph that serve as "shortcuts" by connecting route segments which can be described by subsequent identical instructions. This method however will make the graph non-planar. In this study we adopt another method to handle this by inclusion in network expansion (propagating identical instructions forward through the graph from the edge currently being processed). First, we need to know whether edges are *chunkable*.

DEFINITION 2 (CHUNKABLE EDGE). *An edge e_t is chunkable from edge e_s with an instruction i if there is a path from e_s to e_t that can be encoded as sequence of executions of i, and such a sequence is valid according to the employed chunking rules.*

As decision points in a chunk can be covered with a single instruction, besides the decision point to reach the first edge, the negotiation costs of the following decision points in the chunk are no longer evaluated individually (recall that for the first example in Figure 3, the navigation cost of "turn left twice" is measured by 4+2, rather than 4+4). We should only add traversal costs when passing more decision points but ignore negotiation costs for those edges which are chunkable. Formally, to implement this behavior in our algorithm, we assume a *chunk validity check* function.

DEFINITION 3 (CHUNK VALIDITY CHECK FUNCTION). *We define a chunk validity check function $v : E \times E \times I \rightarrow \{true, false\}$ for a given starting edge $e_s \in E$, terminating edge $e_t \in E$, and instruction $i \in I$. $v(e_s, e_t, i)$ is true only if e_t is chunkable from e_s using i.*

5.2 Network Expansion

We first consider a scenario that excludes global chunkings (i.e., only local landmarks can be included in route directions). The pseudo-code of the module for searching k easiest-to-reach neighbors is presented in Algorithm 1 and explained in detail below. Given the relevant part of a locally transformed graph with specific weights of involved intersections and pre-defined landmarks, and the chunk validity check function v, the algorithm takes a query location q and a value k denoting the number of data objects requested as input, and returns the k easiest-to-reach neighbors together with their corresponding navigation costs from q.

Algorithm 1

Input:
Query location q, number of data objects requested k.
Output:
k easiest-to-reach neighbors with their NCs.
Description:
1: initialize $NC_{max} \leftarrow \infty$;
2: find the edge $(n_i n_j)$ that covers q;
3: $S_{cover} = FindObjects(n_i n_j)$;
//S_{cover} is the set of objects covered by $(n_i n_j)$
4: $O' = \{p_1, \ldots, p_k\}$ are the k easiest-to-reach neighbors sorted in ascending order of NC (initially empty);
5: $Q = [(n_i, NC(n_i), min\_NC(n_i), pre\_i(n_i)),$
$(n_j, NC(n_j), min\_NC(n_j), pre\_i(n_j))]$; //sorted by NC
6: dequeue the node n in Q with the smallest $NC(n)$;
7: **while** $|O'| \leq k$ and $NC(n) \leq NC_{max}$ **do**
8: **for** each adjacent node n_x of n connected by unexplored edge **do**
9: $S_{cover} = FindObjects(nn_x)$;
10: update O' from $O' \cup S_{cover}$;
11: $NC_{max} = NC(q \rightsquigarrow p_k)$;
12: enqueue or update $(n_x, NC(n_x), min\_NC(n_x), pre\_i(n_x))$;
//evaluate with the cost function and chunk validity check
13: **end for**
14: dequeue the next node n in Q;
15: **end while**

Our algorithm incrementally expands its search for data objects through the network starting from q. Whenever a node is expanded, all outgoing edges from the node are retrieved and adjacent nodes are explored. Yet-to-be-visited nodes are stored in a priority queue Q sorted in ascending order of navigation cost to reach from q (Q is assumed to be empty initially and does not allow duplicate nodes). The information of each element in Q is represented as a tuple $(n, NC(n), min\_NC(n), pre\_i(n))$, where n is the node ID, $NC(n)$ is the *currently determined* navigation cost from q to the node, $min\_NC(n)$ is the *minimum possible* navigation cost from q (by assuming traversal costs only, which is computed based on the number of route segments to reach the node), and $pre\_i(n)$ is the recorded predecessor instruction. The algorithm iteratively expands the node currently with the smallest NC and adds its adjacent nodes into Q (if a node has been visited before and hence is already in the priority queue then, if the newly determined navigation cost is smaller than its NC stored in Q, the navigation cost is updated). These operations are repeated, and terminated when the following conditions are met: we have k data objects found by S_{cover} in $\{p_1, \ldots, p_k\}$, and no other object is possible to have a smaller navigation cost than the kth one we already found (once the next node n to be expanded in Q has a navigation cost $NC(n)$ that is larger than NC_{max}). Our core idea is that local landmarks are integrated in the cost evaluation by using instructions incorporating landmarks to override plain instructions whenever appropriate, and spatial chunkings are handled by chunk validity check during each step the expansion. As we will see in Section 5.3, an additional post-verification module can be employed in the presence of global landmarks with the aid of obtained minimum possible navigation cost information, $min\_NC(n)$.

To clarify the algorithm, we explain it using an example as shown in Figure 4 in conjunction with Table 1, to find the first easiest-to-reach neighbor with respect to the query location q. A simple instruction set $I = \{Left \rightarrow L, Right \rightarrow R, Straight \rightarrow S\}$ is assumed for ease of illustration. The cost function ω follows the scheme shown in Figure 2. Next, we describe the workings of the algorithm step by step.

First, after initialization the algorithm begins with searching in the network R-Tree to locate the query point q, for finding which

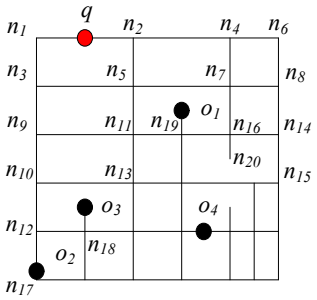


Figure 4: The sample example following Figure 1, with subscript of a node denoting the visiting order of it.

edge on the original graph it lies on. The result is (n_1, n_2). Then it checks whether there are any data objects on this edge (with the function $FindObjects$) to be added into in-memory list S_{cover}. In our example here $S_{cover} = \varnothing$. Then, n_1 and n_2 are added into the priority queue Q. Nodes in Q should be sorted by NC in ascending order. Here, we assign nodes directly connected to the query point q with zero navigation cost (i.e., no cost is applied to the initial orientation stage). A more sophisticated choice might include weights for initial orientation, perhaps preferentially selecting initial orientations that are easier to explain (e.g., orientation towards an easily visible landmark).

Next, the algorithm removes the node with smallest NC from Q and expands it. In case of a tie we pick up an arbitrary one, so the element $(n_1, 0, 0, S)$ is dequeued first and n_1 is expanded. Since the negotiation cost of the turning instruction "L" via n_1 to n_3 is 4 (according to Figure 2) and the minimum traversal cost is assigned to be 2, we enqueue a new element $(n_3, 4, 2, L)$ in Q. Every time we dequeue the element of a node n which is to be expanded, the navigation cost from q to each adjacent node n_x of n connected by an unexplored edge will be evaluated according to the cost function ω together with the chunk validity function v. This is because we have the predecessor instruction of each step recorded and when consecutive instructions are identical, the increment for the navigation cost is just the traversal cost (smaller), but not negotiation cost (larger). As discussed above, *the increment for the navigation cost for expansion to a new node should always be determined by the instruction with the smallest cost if there are multiple possibilities to encode the action*. For example, when the algorithm dequeues the element $(n_2, 0, 0, S)$ and expands the node n_2, since the instruction of the action from n_2 to its adjacent node n_4 is "S" which is identical with the previous one, the navigation cost is increased by 2, so $(n_4, 2, 2, S)$ is added into Q. At the same time, the navigation cost from q to n_5 (another adjacent node of n_2) is evaluated. Since the negotiation cost of the turning instruction "R" via n_2 to n_5 is 8 (according to Figure 2), another element $(n_5, 8, 2, R)$ is also enqueued. In this way, the algorithm iteratively expands the node with the smallest NC and adds its adjacent nodes into Q (the detailed steps are shown in Table 1). Note that, when a node n is removed from the priority queue for expansion, we have to examine all its adjacent nodes connected by unexplored edges. If an adjacent node has been visited before (by another expansion) and is already in Q, once an even smaller NC value than the old one is obtained, the tuple stored in Q will be updated. For example, when n_3 is expanded in step 6, the navigation cost to its adjacent node n_5 via the path $q \rightarrow n_1 \rightarrow n_3 \rightarrow n_5$ should be evaluated. The result is 4+8=12 which is larger than the NC value of 8 obtained from the previous step, so the original tuple $(n_5, 8, 2, R)$ is kept unchanged. On the other hand, although n_{11} is first visited from the expansion

Step	Operation	Elements in the priority queue Q (sorted in ascending order of NC)
1	Locate the edge (n_1, n_2) that covers q	$(n_1, 0, 0, S), (n_2, 0, 0, S)$
2	No object on (n_1, n_2), expand n_1	$(n_2, 0, 0, S), \underline{(n_3, 4, 2, L)}$
3	No object on (n_1, n_3), expand n_2	$\underline{(n_4, 2, 2, S)}, (n_3, 4, 2, L), \underline{(n_5, 8, 2, R)}$
4	No object on (n_2, n_4) or (n_2, n_5), expand n_4	$\underline{(n_6, 4, 4, S)}, (n_3, 4, 2, L), (n_5, 8, 2, R), \underline{(n_7, 10, 4, R)}$
5	No object on (n_4, n_6) or (n_4, n_7), expand n_6	$(n_3, 4, 2, L), \underline{(n_8, 8, 6, R)}, (n_5, 8, 2, R), (n_7, 10, 4, R)$
6	No object on (n_6, n_8), expand n_3	$\underline{(n_9, 6, 4, S)}, (n_8, 8, 6, R), (n_5, 8, 2, R), (n_7, 10, 4, R)$
7	No object on (n_3, n_9) or (n_3, n_5), expand n_9	$\underline{(n_{10}, 8, 6, S)}, (n_8, 8, 6, R), (n_5, 8, 2, R), (n_7, 10, 4, R), \underline{(n_{11}, 14, 6, L)}$
8	No object on (n_9, n_{10}) or (n_9, n_{11}), expand n_{10}	$(n_8, 8, 6, R), (n_5, 8, 2, R), \underline{(n_{12}, 10, 8, S)}, (n_7, 10, 4, R), (n_{11}, 14, 6, L), \underline{(n_{13}, 16, 8, L)}$
9	No object on (n_{10}, n_{12}) or (n_{10}, n_{13}), expand n_8	$(n_5, 8, 2, R), \underline{(n_{14}, 10, 8, S)}, (n_{12}, 10, 8, S), (n_7, 10, 4, R), (n_{11}, 14, 6, L), (n_{13}, 16, 8, L)$
10	No object on (n_8, n_{14}) or (n_8, n_7), expand n_5	$(n_{14}, 10, 8, S), (n_{12}, 10, 8, S), (n_7, 10, 4, R), \mathbf{(n_{11}, 10, 4, S)}, (n_{13}, 16, 6, L)$
11	No object on (n_5, n_7) or (n_5, n_{11}), expand n_{14}	$(n_{12}, 10, 8, S), (n_7, 10, 4, R), (n_{11}, 10, 4, S), \underline{(n_{15}, 12, 10, S)}, (n_{13}, 16, 8, L),$ $\underline{(n_{16}, 18, 10, R)}$
12	No object on (n_{14}, n_{15}) or (n_{14}, n_{16}), expand n_{12}	$(n_7, 10, 4, R), (n_{11}, 10, 4, S), \underline{(n_{17}, 12, 10, S)}, (n_{15}, 12, 10, S), (n_{13}, 16, 8, L),$ $(n_{16}, 18, 10, R), \underline{(n_{18}, 18, 10, L)}$
13	o_2 is found on (n_{12}, n_{17}) and no object on (n_{12}, n_{18}), expand n_7	$(n_{11}, 10, 4, S), \mathbf{(n_{16}, 12, 6, S)}, (n_{17}, 12, 10, S), (n_{15}, 12, 10, S), (n_{13}, 16, 8, L),$ $(n_{18}, 18, 10, L)$
14	No object on (n_7, n_{16}), expand n_{11}	$(n_{16}, 12, 6, S), (n_{17}, 12, 10, S), (n_{15}, 12, 10, S), \mathbf{(n_{13}, 12, 6, S)}, (n_{18}, 18, 10, L),$ $\underline{(n_{19}, 19, 6, L)}$
15	No object on (n_{11}, n_{13}) or (n_{11}, n_{19}), expand n_{16}	$(n_{17}, 12, 10, S), (n_{15}, 12, 10, S), (n_{13}, 12, 6, S), \underline{(n_{20}, 14, 8, S)}, (n_{18}, 18, 10, L),$ $(n_{19}, 19, 6, L)$

Table 1: Network expansion process for the example of Figure 4. Underline typeface means the this element is newly enqueued after this expansion, and bold typeface means the original tuple for this node which is already in Q is updated with a lower NC.

of n_9 in step 7 (an element $(n_{11}, 14, 6, L)$ is enqueued due to the expansion path $q \rightarrow n_1 \rightarrow n_3 \rightarrow n_9 \rightarrow n_{11}$), its navigation cost is lowered in step 10 (the tuple is updated to $(n_{11}, 10, 4, S)$ due to the expansion path $q \rightarrow n_2 \rightarrow n_5 \rightarrow n_{11}$). Likewise, although n_{16} is first visited from the expansion of n_{14} in step 11, its navigation cost is lowered in step 13 with the expansion of n_7. When o_2 is discovered on the edge (n_{12}, n_{17}) after the expansion of n_{12}, we set the threshold $NC_{max} = 12$ which provides a bound to restrict the search space. Finally, due to the fact that further expanding n_{16}[3] which has a same navigation cost with NC_{max} cannot find any objects with smaller navigation cost, the algorithm terminates and o_2 is returned as the answer.

For implementation, we mainly need four data structures to support the network expansion process: (i) an *adjacency component* which captures the network connectivity; (ii) an *edge component* which provides the information of each network edge (u, v), length of the edge $d_G(u, v)$, and a pair of pointers to the adjacency list for its two endpoints u and v; (iii) an *instruction complexity component* which includes the cost of each network edge pair (these costs are computed during the expansion); (iv) an *R-tree component* which indexes the MBRs of edges.

By expanding the node with the smallest NC and adding new adjacent nodes, the algorithm maximizes the chance that promising data objects are located earlier than others. This is similar in spirit to Dijkstra's algorithm to propagate a search "wavefront" for finding the shortest paths from a source node to multiple destinations. However, in our problem we need to consider that currently determined navigation costs of nodes could be lowered by different expansion paths and chunking rules. Our algorithm works on a transformed graph modelling instruction complexity, and we only materialize the relevant part of it when actually needed. The cognitive principles of spatial chunking and reference to landmarks are also carefully taken into account. At the heart of our algorithm is the chunk validity check that is employed during the expansion. Previous action at a decision point used to reach an edge from its preceding edge is recorded for each step. The algorithm applies every valid chunking rule to minimize the navigation cost. When the algorithm explores a new edge, it immediately checks whether two edges are chunkable so that the navigation cost can be lowered compared to the cost when treating them separately. If consecutive edges are chunkable, rather than evaluating the increment for the navigation cost by the regular negotiation cost, only a small traversal cost is added (the first navigation cost measuring policy). Since the navigation cost with chunking is monotonically increasing, the correctness of our algorithm is guaranteed. In effect, if m edges are chunkable by an instruction, the navigation cost is simply the negotiation cost at the starting decision point of the chunk, plus $m - 1$ times the traversal cost. For example, the navigation cost of $q \rightarrow n_1 \rightarrow n_3 \rightarrow n_9 \rightarrow n_{10} \rightarrow n_{12} \rightarrow o_2$ is measured by $4 + 4 \times 2$. By keeping track of $pre\_i(n)$, "do n times" chunkings are include in our network expansion.

5.3 Post-verification for Global Landmarks

For the purpose of testing whether there is any global landmark that could be included in route directions and consequently results in an even smaller navigation cost to reach a neighbor, an additional post-verification module can be used. Recall that for each node n, besides the navigation cost $NC(n)$ we also store the information of minimum possible navigation cost $min\_NC(n)$ during the expansion by assuming traversal costs only. $min\_NC(n)$ is measured as a *lower bound* of any possible navigation cost even if global chunking is applied. Note that, the minimum number of route segments to reach a node depends on the expansion path. For example in the process shown in Table 1, $min\_NC(n_{11})$ is first computed as 6 in step 7 (via $q \rightarrow n_1 \rightarrow n_3 \rightarrow n_9 \rightarrow n_{11}$, 3 times the traversal cost), but becomes 4 later in step 10 (via $q \rightarrow n_2 \rightarrow n_5 \rightarrow n_{11}$, only 2 times the traversal cost).

For global chunking, recall that the navigation cost is determined by the instruction referring to a global landmark at the final decision point, plus the cumulative traversal cost that relates to how many decision points have been chunked (the second navigation cost measuring policy). For example, if we have an available global landmark LM located at the intersection n_{18} as shown in Figure 5,

[3]As shown in step 15, while the similar steps of expanding n_{17}, n_{15} and n_{13} are omitted from the table for brevity.

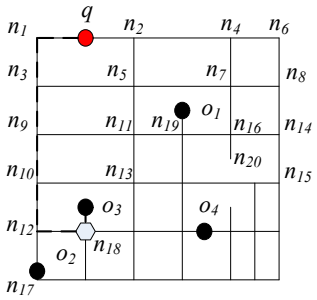


Figure 5: With a global landmark LM located at n_{18} (represented by a hexagon), the description of o_3 could be "turn right at LM" (via the dashed thick path).

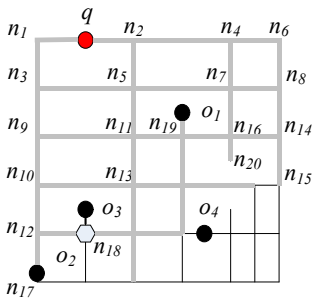


Figure 6: The grey bold subgraph ($min\_NC \leq 10$) needs to be verified whether there is any global landmark.

the route directions for reaching o_3 could be simply *turn right at LM*. Suppose the cost value of executing such a turning instruction at the landmark LM is 4, the navigation cost of reaching o_3 is 10+4=14. This is because the minimum number of route segments to reach n_{18} is 5, so the cumulative traversal cost is 10 although there is no need to mention the actions at the five intermediate decision points n_1, n_3, n_9, n_{10} and n_{12} in-between the start (q) and the end (n_{18}) of the chunk.

Using $min\_NC(n)$, we can test the possibility whether some other data objects which have inclusion of global landmarks in their neighbor descriptions can result in even smaller NC. To this end, the tentative easiest-to-reach neighbor we have found in the network expansion process needs to be further verified to ensure that our eventual choice has the smallest navigation cost. The pruning condition can be summarized as: if the cost of instruction referring to global landmark is defined to be at least c_{min}, the area reachable from q with minimum possible navigation cost under $NC_{max} - c_{min}$ should be searched for global landmarks. For example, if we define $c_{min} = 2$, in Figure 6 the grey bold subgraph (reachable from q with min_NC under 12-2=10) needs to be verified whether there is any global landmark that can be included in a neighbor description.

6. TAILORING TO USER PREFERENCES

It is often a desirable feature of navigation services to be adaptive to user preferences. The easiest-to-reach neighbor search algorithm tends to be reluctant to choose a turn, so in some extreme cases it could produce routes of considerable length. When there are a variety of choices of the same facility type available, sometimes navigators being guided in unfamiliar geographic environments would like to achieve certain trade-off between nearest and easiest-to-reach neighbors, or prefer major roads to minor roads, etc. With the proposed algorithm as a basis, we introduce some possible extensions to support these more sophisticated behaviors.

The problem of nearest neighbor search has been studied extensively in spatial networks [3, 10, 14, 18, 24]. If travellers want to achieve some balance between *travel distance* and *navigation cost*, we can introduce a parameter λ to determine the cost used in the network expansion process described above. The hybrid of these two criteria can be reflected by a modified cost function

$$\lambda \cdot Cost_{distance} + (1 - \lambda) \cdot Cost_{instruction}$$

where $Cost_{distance}$ is derived from the cost function $\psi : E \rightarrow \mathbb{R}^+$ regarding network distance of traversing the edges, $Cost_{instruction}$ derived from the function $\omega : I \rightarrow \mathbb{R}^+$ modelling instruction complexity of turning onto the edges, and $\lambda \in [0, 1]$ is a heuristic parameter used in the weighted sum (in order to produce dimensional similitude, the value of λ has to be calibrated for specific road networks to scale $Cost_{distance}$ and $Cost_{instruction}$ to be in the same units). Thus, a node shared by a network edge pair can be labelled with some new cost value embedded with both travel distance and cognitive complexity of navigation. These costs are also to be computed on-the-fly from geometry and topology during the expansion.

The easiest-to-reach neighbor algorithm sometimes selects major roads simply by virtue of their straighter geometry and less connected topology (i.e., fewer intersections). A modification of the cost function could easily be implemented to explicitly prefer certain types of road. For example, major roads could be preferred by making the costs for turns onto a major road smaller and turns off a major road larger, whereas minor roads could be avoided by making the costs of turns onto a minor road larger and turns off a minor road smaller.

7. EXPERIMENTAL EVALUATION

In this section, we report the experiment results of the proposed search algorithm.

7.1 Setup

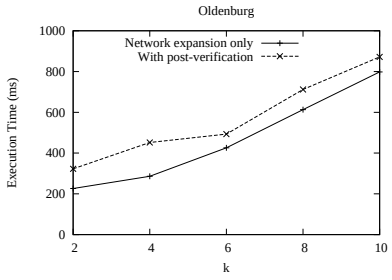
We use two real-world road network datasets for the experiments. They are the road networks for the city of Oldenburg (6,105 nodes, 7,034 edges) and San Joaquin County (18,263 nodes, 23,874 edges), respectively. The reason for using these two networks is that they well represent typical urban configurations. The degrees of connectivity in the both road networks are mostly between 2 to 4. All angular deviations less than 12° per side (within a 24° sector) are perceived as "go straight". In addition, we annotate 500 and 2,000 nodes as point-like landmarks located at intersections on Oldenburg and San Joaquin County road networks respectively, with the location and saliency information specified. 20% of them are annotated as global landmarks.

Data objects are synthetically generated on the networks with different densities (defined as *the percentages of the number of data objects over the number of nodes in the network*) varying from 1% to 10%. These data objects are distributed uniformly over the whole network space so that the data object distribution follows the network distribution. They are indexed by an R-tree with the maximum of 10 entries in each node. According to our analysis of real Point of Interest (POI) information available for another road network for California[4], 71.4% types of facilities have a distribution

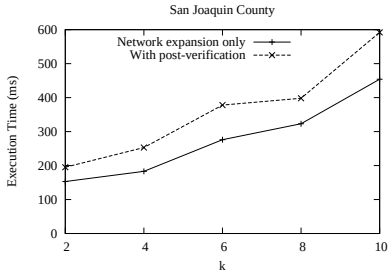
[4]This is the only dataset currently we have access to the real categories of POI information. However, since it represents the road network of a whole state, we prefer to use the above two datasets which include a denser downtown grid road network region and sparser suburban road network regions for test purpose.

Parameter	Setting
Data object density	1% to 10%, increment 1%, default: 5%
k	2 to 10, increment 2, default: 4

Table 2: Parameter settings in the experiments.



(a) Oldenburg road network.



(b) San Joaquin County road network.

Figure 7: Effect of k.

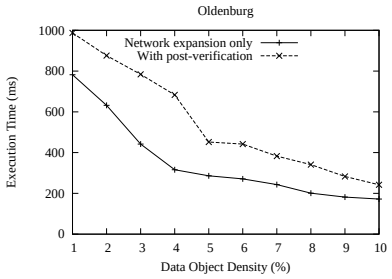
density less than 5%. The default values and ranges of the experimental parameters are summarized in Table 2.

The proposed algorithm is implemented in Java. For k easiest-to-reach neighbor search, we execute workloads of 50 queries (random pick of 50 nodes in the networks as the test query locations), also following the network distribution. In the experimental evaluation, we study the efficiency of the proposed search algorithm, by varying the number of data objects requested and the distribution density. For each set of experiments, we only vary one parameter and fix the other to its default value. In the experiments we measure *execution time*, which indicates the total running time for the query processing. Our experiments are conducted on Windows XP platform with Intel Core 2 Duo CPU (2.66 GHz) and 4.0 GB RAM. The results reported are the average of 50 individual queries.

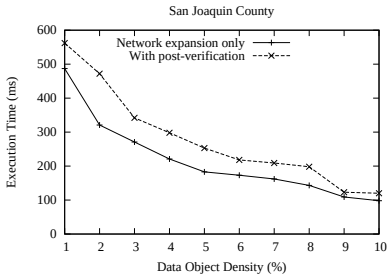
7.2 Results

First, we study the performance of the proposed search algorithm with regard to k, the number of easiest-to-reach neighbors to be searched. Figure 7 shows the query execution time versus k ranging from 2 to 10, with or without running the post-verification module for global landmarks. The results of both networks show that the running time of the network expansion module increases almost linearly with k. As can be expected, for a given road network, a large value of k corresponds to a larger area to be searched and more network expansion steps. However, the increase of overall number of network nodes accessed is not proportional to the increase of k. This is because the network search area for candidate data objects overlaps.

Next, we study the performance of the proposed search algorithm by varying the cardinality of data objects. Figure 8 shows the



(a) Oldenburg road network.



(b) San Joaquin County road network.

Figure 8: Effect of data object density.

average query efficiency versus distribution density, when searching the default number of easiest-to-reach neighbors with or without running the post-verification module. From this set of experiments, we can conclude that in general when data object density increases, the execution time of the network expansion module decreases. This is because data objects are sparsely distributed in networks when density is relatively small. The number of steps to encounter a data object will increase, since more network nodes need to be expanded to get the results. In addition, by comparing the results from the two networks, we observe that the performance of our algorithm is insensitive to different network sizes. This can be explained as easiest-to-reach neighbors are generally close to query locations, so normally only a small portion of the network is relevant to query processing.

Generally, our algorithm performs well if the distribution density is not very small, and k is not very large (both assumptions are reasonable in practice). If we investigate the execution time of the 50×2 runs individually, it is observed that the performance of query processing is poorer for query locations in a denser downtown grid than a sparser suburban region for the both road networks. Usually query points far away from the central grid requires relatively smaller number of expansion steps to find a neighboring object.

Furthermore, in the experimental evaluation we also examine the answers returned by the existing nearest neighbor search algorithm INE. We compare the travel distance and navigation cost of the first easiest-to-reach neighbor of each query with the first nearest neighbor (k=1, under the default density of data objects). The results reveal that the network distance to get to an easiest-to-reach neighbor has an average increase of 19.3% compared to that of a nearest neighbor, but at the same time the navigation cost is only about 35.2% compared to that of the nearest neighbor. Thus, in return for slightly longer travel distances, the easiest-to-reach neighbors offer considerable advantages over the nearest neighbors in terms of their ease of navigation (on average a saving of 64.8%), in particular for travellers unfamiliar with a foreign city. Therefore, the algorithm for this new type of query could be used as an alternative to the ex-

isting nearest neighbor search algorithms for advanced navigational assistance, to benefit users by reporting best candidates in terms of navigation complexity.

8. CONCLUSION AND FUTURE WORK

Finding a facility nearby for navigation is an increasingly popular and economically important application area for geographic information systems. Navigation services for people in unfamiliar environments should select route directions which are easy to follow, even if they are not the shortest ones. This paper introduces and solves such spatial queries which have not been considered before. As a result, we bridge the gap between findings from cognitive science and spatial databases. Our contribution is an incremental network expansion algorithm for easiest-to-reach neighbor search. We use cognitive complexity of navigation to dynamically label each node during a greedy search. More importantly, the algorithmic correctness and efficiency are ensured through carefully taking spatial chunking and reference to landmarks into account. Our algorithm is flexible with respect to the cost function used, and thus can be tailored to user preferences. Finally, a systematic experimental evaluation has been conducted on different road networks under various parameter settings, to evaluate its performance.

Currently we assume that for each required turn an instruction is given and discrete weights are used in the cost function to model instruction complexity. However, the proposed search algorithm does not rely on any particular weighting scheme, as long as some model of instruction complexity can be derived for decision points. In some contexts, continuous weights (e.g., costs are attributed with the angle in a turn) can be used if human travellers find them more appropriate. In addition, knowledge about one's environment can influence the amount of information provided in a neighbor description and the cognitive effort put on the navigator to reach it. We plan to further investigate tracking user movements by GPS devices to allow for establishing patterns of frequently visited places which can be assumed to be known. This enables profile-based destination choices which can better account for specific user characteristics.

Acknowledgments: The work reported in this paper is supported under the Australian Research Council's Discovery funding scheme (project number DP0880215).

9. REFERENCES

[1] D. Caduff, S. Timpf, and A. Klippel. The landmark spider: Representing landmark knowledge for wayfinding tasks. In *Reasoning with Mental and External Diagrams: Computational Modeling and Spatial Assistance, AAAI Spring Symposium*, pages 30–35, 2005.

[2] U. Demiryurek, F. B. Kashani, and C. Shahabi. Towards k-nearest neighbor search in time-dependent spatial network databases. In *International Workshop on Databases in Networked Information Systems (DNIS)*, pages 296–310, 2010.

[3] K. Deng, X. Zhou, H. T. Shen, S. W. Sadiq, and X. Li. Instance optimal query processing in spatial networks. *VLDB J.*, 18(3):675–693, 2009.

[4] M. Duckham and L. Kulik. "Simplest" paths: Automated route selection for navigation. In *International Conference On Spatial Information Theory (COSIT)*, pages 169–185, 2003.

[5] B. Elias. Extracting landmarks with data mining methods. In *International Conference On Spatial Information Theory (COSIT)*, pages 375–389, 2003.

[6] B. Elias and M. Sester. Incorporating landmarks with quality measures in routing procedures. In *GIScience*, pages 65–80, 2006.

[7] R. G. Golledge. Path selection and route preference in human navigation: A progress report. In *International Conference On Spatial Information Theory (COSIT)*, pages 207–222, 1995.

[8] J. Goodman, P. D. Gray, K. Khammampad, and S. A. Brewster. Using landmarks to support older people in navigation. In *Mobile HCI*, pages 38–48, 2004.

[9] S. Haque, L. Kulik, and A. Klippel. Algorithms for reliable navigation and wayfinding. In *Spatial Cognition*, pages 308–326, 2006.

[10] X. Huang, C. S. Jensen, H. Lu, and S. Saltenis. S-grid: A versatile approach to efficient query processing in spatial networks. In *SSTD*, pages 93–111, 2007.

[11] E. Kanoulas, Y. Du, T. Xia, and D. Zhang. Finding fastest paths on a road network with speed patterns. In *ICDE*, page 10, 2006.

[12] A. Klippel, S. Hansen, K.-F. Richter, and S. Winter. Urban granularities - a data structure for cognitively ergonomic route directions. *GeoInformatica*, 13(2):223–247, 2009.

[13] A. Klippel, H. Tappe, and C. Habel. Pictorial representations of routes: Chunking route segments during comprehension. In *Spatial Cognition*, pages 11–33, 2003.

[14] M. R. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB*, pages 840–851, 2004.

[15] K. L. Lovelace, M. Hegarty, and D. R. Montello. Elements of good route directions in familiar and unfamiliar environments. In *International Conference On Spatial Information Theory (COSIT)*, pages 65–82, 1999.

[16] D. M. Mark. Automated route selection for navigation. *IEEE Aerospace and Electronic Systems Magazine*, 1(9):2–5, 1986.

[17] P.-E. Michon and M. Denis. When and why are visual landmarks used in giving directions? In *International Conference On Spatial Information Theory (COSIT)*, pages 292–305, 2001.

[18] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB*, pages 802–813, 2003.

[19] M. Raubal and S. Winter. Enriching wayfinding instructions with local landmarks. In *GIScience*, pages 243–259, 2002.

[20] K.-F. Richter. A uniform handling of different landmark types in route directions. In *International Conference On Spatial Information Theory (COSIT)*, pages 373–389, 2007.

[21] K.-F. Richter and M. Duckham. Simplest instructions: Finding easy-to-describe routes for navigation. In *GIScience*, pages 274–289, 2008.

[22] K.-F. Richter and A. Klippel. A model for context-specific route directions. In *Spatial Cognition*, pages 58–78, 2004.

[23] T. Ross, A. J. May, and S. Thompson. The use of landmarks in pedestrian navigation instructions and the effects of context. In *Mobile HCI*, pages 300–304, 2004.

[24] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD*, pages 43–54, 2008.

[25] S. Winter. Weighting the path continuation in route planning. In *GIS*, pages 173–176, 2001.

[26] S. Winter. Modeling costs of turns in route planning. *GeoInformatica*, 6(4):363–380, 2002.