Efficient Landmark-Based Candidate Generation for kNN Queries on Road Networks

Tenindra Abeywickrama and Muhammad Aamir Cheema

Monash University, Melbourne, Australia
`tenindra.abeywickrama,aamir.cheema@monash.edu`

Abstract. The k nearest neighbor (kNN) query on road networks finds the k closest points of interest (POIs) by network distance from a query point. A past study showed that a kNN technique using a simple Euclidean distance heuristic to generate candidate POIs significantly outperforms more complex techniques. While Euclidean distance is an effective lower bound when network distances represent physical distance, its accuracy degrades greatly for metrics such as travel time. Landmarks have been used to compute tighter lower bounds in such cases, however past attempts to use them in kNN querying failed to retrieve candidates efficiently. We present two techniques to address this problem, one using ordered Object Lists for each landmark and another using a combination of landmarks and Network Voronoi Diagrams (NVDs) to only compute lower bounds to a small subset of objects that may be kNNs. Our extensive experimental study shows these techniques (particularly NVDs) significantly improve on the previous best techniques in terms of both heuristic and query time performance.

Keywords: Road networks, nearest neighbor, landmark lower bounds

1 Introduction

The k nearest neighbor (kNN) query on road networks finds the k closest points of interest (POIs) by their shortest path distances in the road network from a query point. Incremental Euclidean Restriction (IER) [10] is a kNN method that uses a simple Euclidean distance heuristic. IER retrieves Euclidean kNNs as *candidates* and computes network distances to each one using a shortest path algorithm (e.g., Dijkstra). The kth furthest candidate implies an upper bound network distance to the kth NN. IER then iteratively retrieves further Euclidean NNs, computes network distances and updates the k candidates if closer POIs are found. Since Euclidean distance is a lower bound on network distance, IER terminates when the distance to the next Euclidean NN is larger than the network distance to the kth candidate. In a recent PVLDB experimental study [1], when IER was combined with a fast shortest path technique (instead of Dijkstra) it was found to be significantly faster than the state-of-the-art kNN methods. Inspired by the observation of a simple heuristic being so effective, our study seeks to improve on this by employing better heuristics.

Euclidean distance is a lower bound on the network distance between vertices in road network graphs with travel distance edge weights. It can also be adapted for use when edge weights represent other metrics. For example when they represent travel time, we can divide the Euclidean distance by the maximum speed for any edge to obtain the minimum possible travel time. However a lower bound obtained in this way is looser and IER is likely to retrieve far more candidates that are not real kNNs (false hits) on travel times. This was evident as IER's advantage was smaller in several travel time experiments in [1]. This identifies the need for improvement by using better heuristics.

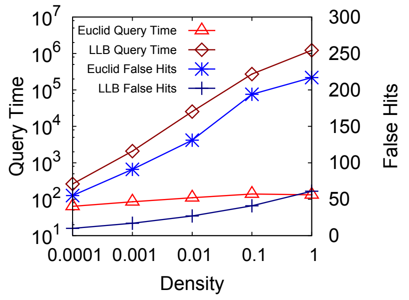
A popular alternative lower bounding technique is based on using *landmarks*, which Goldberg et al. [4] employed to improve the A* shortest path algorithm. By using distances to *landmark* vertices and the triangle inequality, they were able to compute more accurate lower bounds leading to a significant speed-up of A* search. Naturally, as the shortest path problem is closely related to the kNN problem, this raises the question whether these *Landmark Lower Bounds* (LLBs) can be used to similarly improve IER's kNN query performance. Until our study, the answer to this question has been "no".

Past attempts to use landmarks [7, 8] computed lower bounds for *all* POIs and stored them in a sorted list. This is necessary for *every* query as LLBs depend on the query vertex. Candidates with the next smallest lower bound are retrieved iteratively from the list. This approach may be reasonable for small numbers of POIs as computing lower bounds is a relatively fast operation. However it will not scale well with increasing numbers of POIs. Some POI sets number in the tens of thousands e.g., the 25,000 fast food outlets in the US [1]. In such scenarios, it is desirable to incrementally retrieve POIs in order of their LLBs without computing LLBs for all POIs. For example, Euclidean NNs can be incrementally retrieved quite efficiently using an R-tree or similar structure.

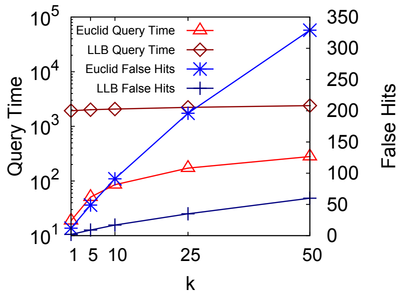
Figure 1 demonstrates this problem using the US travel time road network. In Figure 1(a) the LLB-based method is orders of magnitude worse on query time (left y-axis) despite producing fewer false hits (right y-axis). This is due to there being more POIs in total with increasing density (i.e., ratio of POIs to vertices), requiring more LLBs to be computed. The query time in Figure 1(b) is constant as it is dominated by the computation of LLBs irrespective of k. In both figures we clearly see that (1) LLB-based methods provide better lower bounds as evident from the fewer false hits and (2) LLB-based methods perform very poorly without the ability to retrieve candidates incrementally.

Inspired by the performance of Euclidean heuristics in the PVLDB study and using the observation that landmark lower bounds appreciably reduce false hits over Euclidean distance, we investigate how to efficiently employ landmarks to improve kNN query performance. To summarise our contributions:

– We present two techniques to generate kNN candidates using landmarks. The first, Object Lists, demonstrates the difficulties in using LLBs and is efficient in several scenarios. We further improve on this and present another technique which, using a novel combination of Network Voronoi Diagrams



(a) Varying Density



(b) Varying k

Fig. 1. Euclidean kNN vs. Landmark kNN (US, d=0.001, k=10, uniform)

and landmarks, provides significant improvement over existing techniques on both heuristic and running time performance.

– LLBs are expected to be more accurate than Euclidean distance, especially for edge weights such as travel times. However, to the best of our knowledge, they have not been empirically compared for kNN queries. In addition to other experimental results, we present a detailed study into the number of "false hits" (i.e., candidates which are not real kNNs). This machine independent metric is applicable to any experimental setting or shortest path technique, allowing a better understanding of the usefulness of LLBs.

2 Preliminaries

We consider a road network graph $G = (V, E)$ where V is a vertex set and E is an edge set. Edge $(u, v) \in E$ connects two adjacent vertices with weight $w(u, v) \in \mathbb{R}_{>0}$ representing any real positive metric such as distance or travel time. The shortest path $P(u, v)$ with network distance $d(u, v)$ represents the minimum sum of weights connecting any two vertices u and v. Similar to almost all existing studies we consider POIs (objects) and query points located on vertices in V. So, given a query vertex q and a set of object vertices O, a kNN query retrieves the k closest objects in O based on their network distances from q in G.

2.1 Landmark Lower Bounds

To compute a Landmark Lower Bound (LLB), firstly a set of m *landmark* vertices $L = \{l_1, \ldots, l_m\} \subseteq V$ is selected. From each landmark $l_i \in L$ we compute the distances to all vertices in V. Now given a source vertex q and destination vertex o, we can compute a lower bound LB_{l_i} on the network distance $d(q, o)$ using the distances to landmark l_i and the triangle inequality as defined in (1). We obtain the "tightest" lower bound (i.e., closest to $d(q, o)$) by choosing the maximum lower bound LB_{max} over all m landmarks as defined in (2).

$$LB_{l_i}(q,o) = |d(l_i,q) - d(l_i,o)| \leq d(q,o) \tag{1}$$

$$LB_{max}(q,o) = \max_{l_i \in L}(|d(l_i,q) - d(l_i,o)|) \tag{2}$$

First applied to road networks by Goldberg et. al [4], there now exists a large body of work utilising this concept. Two considerations arising from LLBs are (a) the vertices to select as landmarks and (b) the number of landmarks. Intuitively vertices whose shortest path trees cover longer shortest paths (e.g., those appearing at fringes of the graph [4]) have a higher probability of giving tighter lower bounds. A larger number of landmarks similarly increases this probability, but at the expense of higher space cost and computing more lower bounds to find the tightest overall. Since our study is concerned with using LLBs efficiently for kNNs rather than improving them, we refer the reader to past studies [4, 5] for discussion on these choices. Note that (1) and (2) are only applicable on undirected graphs, but the idea can easily be extended to directed graphs by computing distances to and from landmarks.

3 Techniques

As detailed in Section 1, Incremental Euclidean Restriction (IER) is a kNN technique that computes network distances to candidate objects retrieved by their Euclidean distance until the k candidates cannot be improved. This technique can be generalized to consider *any* lower bounding technique, such as Landmark Lower Bounds (LLBs) discussed earlier. Let us refer to the equivalent kNN algorithm to IER using LLBs as Incremental Lower Bound Restriction (ILBR). ILBR works in exactly the same way as IER except we retrieve the candidate with the smallest LLB. IER can incrementally retrieve candidates by Euclidean NN search on an R-tree, avoiding computing Euclidean distances to all objects. But there is no efficient analogous method for LLBs and past studies [7, 8] resort to computing LB_{max} for all objects which is not practicable. Here we present two techniques that incrementally retrieve candidates for ILBR.

3.1 ILBR by Landmark Object Lists

By computing LLBs for all objects as in past studies more LLBs are computed than necessary. We introduce the pre-computed Object List (OL) index that enables LLBs to be computed more optimistically. The OL approach solves the same underlying problem, i.e., to find the object $p \in O$ with the smallest $LB_{max}(q,o)$ as defined by (3). p is then returned to ILBR as a candidate.

$$p = \min_{p \in O}(\max_{l_i \in L}(|d(l_i,q) - d(l_i,p)|)) \tag{3}$$

Pre-Processing: Given object set O, we pre-compute an *Object List* OL_i for each landmark $l_i \in L$ as shown in Figure 2. The list OL_i contains an element

OL_1	$o_1, d(o_1, l_1)$	$o_2, d(o_2, l_1)$	$\ldots$	$o_{\lvert O \rvert}, d(o_{\lvert O \rvert}, l_1)$
			$\vdots$	
OL_m	$o_1, d(o_1, l_m)$	$o_2, d(o_2, l_m)$	$\ldots$	$o_{\lvert O \rvert}, d(o_{\lvert O \rvert}, l_m)$

Fig. 2. Unsorted Object Lists for m Landmarks

for every object $o \in O$, with each element consisting of o and its distance from the landmark $d(l_i, o)$. Finally each list is sorted on $d(l_i, o)$. Since the Object List index only depends on the object set O, which is known beforehand, it is created and sorted entirely in the pre-processing stage.

OL_q	$o_6, 1$	$o_2, 2$	$o_4, 3$	$o_5, 4$	$o_7, 7$	$o_1, 8$	$o_3, 9$
Index	0	1	2	3	4	5	6

Fig. 3. Sample Query Object List OL_q

Query Algorithm: Given a query vertex q and its nearest landmark l_q, we use object list OL_q of l_q to populate a set of *potential candidates*. The first potential candidate is the object that will minimise (1) for l_q. This object can be found by binary search on OL_q for the object p whose distance $d(l_q, p)$ is closest to $d(l_q, q)$. For example Figure 3 depicts OL_q for a set of 7 objects $o_1, ..., o_7$ with distances from l_q. Let us say $d(l_q, q) = 4$, then the binary search will find the element at index 3 (shaded) as closest to 4. Therefore $p = o_5$ minimises (1) with $LB_{l_q}(q, o_5) = |4 - 4| = 0$. Finally p is inserted into a minimum priority queue Q keyed by $LB_{max}(q, p)$ computed using the ALT index [4]. For each vertex in V, ALT contains a list with its distances to each landmark. $LB_{max}(q, p)$ can be efficiently computed by iterating over the lists for q and p.

Lemma 1. *Given an object p and a landmark l_q, any object o with $LB_{l_q}(q, o) < LB_{max}(q, p)$ may also have $LB_{max}(q, o) < LB_{max}(q, p)$.*

Lemma 1 is trivially true when $LB_{max}(q, o) = LB_{l_q}(q, o)$. Now object p_n with the next smallest lower bound by l_q is immediately to the left or right of p (found above by binary search in OL_q). If $LB_{l_q}(q, p_n) < Top(Q)$ then by Lemma 1, p_n may have smaller LB_{max} than any object in Q. While $LB_{l_q}(q, p_n) < Top(Q)$, we search left or right from p. When an object satisfies the condition we compute LB_{max} and insert it into Q. When neither the next left or right object satisfies the condition, the algorithm terminates, and the top element in Q is returned as the object that minimises (3). This is correct as any object further left or right must have $LB_{l_q}(q, p_n) \geq Top(Q)$ and cannot satisfy Lemma 1.

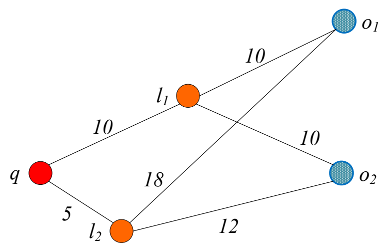


Fig. 4. Query Landmark Choices

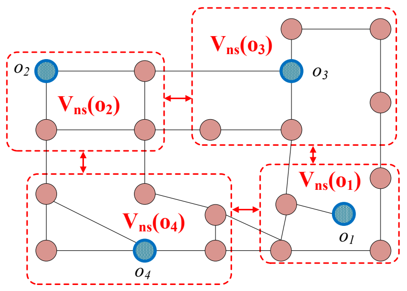


Fig. 5. Network Voronoi Diagram

Let us say in our example $LB_{max}(q, o_5) = 2$ so $Top(Q) = 2$ after inserting o_5. In Figure 3, the objects to the left and right of o_5 are o_4 and o_7, with lower bounds $LB_{l_q}(q, o_4) = |4 - 3| = 1$ and $LB_{l_q}(q, o_7) = |4 - 7| = 3$ respectively. By Lemma 1, o_4 may be a better candidate so we compute $LB_{max}(q, o_4)$ and insert it into Q. Let us say $Top(Q) = 2$ after inserting o_4. For the next element to the left, o_2, we have $LB_{l_q}(q, o_2) = |4 - 2| = 2$. In that case neither lower bounds for the object to the left or right is less than $Top(Q)$ and therefore cannot improve on the objects in Q and the search terminates. By saving Q and the indices in OL_q of the last left and right elements evaluated, we can continue to incrementally retrieve the object with the next smallest LB_{max}.

We choose l_q as the landmark closest to query vertex q. This heuristic increases the probability that objects are found further from l_q than q, thus producing a higher lower bound by (1). For example Figure 4 shows the landmark distances for two landmarks, q and two objects. q, o_1 and o_2 are equally close to l_1, so $LB_{l_1}(q, o_1) = 0$ and $LB_{l_1}(q, o_2) = 0$ even though o_1 is further away. The closer landmark l_2 gives $LB_{l_2}(q, o_1) = 13$ and $LB_{l_2}(q, o_2) = 7$, correctly distinguishing the objects. Object Lists produce fewer false hits than IER on all datasets and is more efficient on low density datasets. However, this problem still occurs on datasets with higher density as landmarks are sparse, and IER is faster on these datasets. We now present a further improved technique.

3.2 ILBR by Network Voronoi Diagrams

Our second technique employs a Network Voronoi Diagram (NVD) [9] of the object set O to improve on Object Lists. Unlike its Euclidean counterpart, NVD generators are limited to the network and shortest paths represent distances.

We define $V_{ns}(o_i)$ as the *Voronoi node set* by (4), which identifies the vertices in V for which o_i is the nearest neighbor by their network distances to o_i.

$$V_{ns}(o_i) = \{v | v \in V, d(v, o_i) \leq d(v, o_j) \forall o_j \in O \setminus o_i\} \tag{4}$$

For any edge (u, v) where $u \in V_{ns}(o_i)$ and $v \in V_{ns}(o_j)$, then $V_{ns}(o_i)$ and $V_{ns}(o_j)$ are *adjacent*. The Network Voronoi Diagram for object set O is the

Algorithm 1: GetNearestCandidateByNVD(q,c_l,NVD,ALT,Q,H)

```
Input   : q: a query vertex, c_l: candidate returned by last call (or 1NN from
          NVD if first call), NVD: Network Voronoi Diagram for object set O,
          ALT: index to compute maximum lower bounds, Q: priority queue
          with potential candidates, H: hash-table containing IDs of all
          Voronoi node sets previously evaluated
Output  : c: candidate object, $LB_{max}(q,c)$: lower bound distance to c over L
1  $GenerateAdjacentCandidates(q, c_l, NVD, ALT, Q, H)$;
2  $(c, LB_{max}(q,c)) \leftarrow Dequeue(Q)$;
3  return $(c, LB_{max}(q,c))$;
4
5  Function $GenerateAdjacentCandidates(q, c_l, NVD, ALT, Q, H)$
6      $V_{ns}(c_l) \leftarrow GetVoronoiNodeSet(c_l, NVD)$;
7      for each $V_{ns}(p) \in AdjacentVoronoiSets(V_{ns}(c_l))$ do
8          if $!H.contains(V_{ns}(p))$ then
9              $Enqueue(Q, (p, ALT.ComputeLBMax(q,p)))$;
10             $H.insert(V_{ns}(p))$;
```

collection of Voronoi node sets for all objects in O. Figure 5 shows an example NVD for a graph with unit edge weights and four objects. Each Voronoi node set is surrounded by a dotted container and arrows indicate adjacency.

NVDs are not new in the context of kNNs [6, 15]. VN^3 [6] utilises an NVD to retrieve candidates using the observation that the next NN is contained in a Voronoi node set adjacent to the sets of NNs found so far. VN^3 also pre-computes certain network distances. For each Voronoi node set this includes the distance from each border vertex to every other border and from each border to every contained vertex in the set. This allows VN^3 to also compute the network distance to retrieved candidates, but entails huge pre-processing and query cost. We instead relax the original observation to consider candidate NNs rather than NNs, which allows us to incrementally retrieve candidates for computing cheap LLBs. Through this novel combination of the standard NVD and landmarks we are able to consider significantly fewer candidates than Object Lists and avoid the large pre-processing overhead of VN^3.

Pre-Processing: An NVD can be computed optimally in $O(|V| \log |V|)$ time with $O(|V|)$ space [3] using simultaneous Dijkstra's searches from all objects using a single priority queue. When a vertex v_d inserted by the search from o_i is dequeued and v_d is not assigned to a Voronoi node set, it is assigned to $V_{ns}(o_i)$. This is correct as v_d is the minimum element in the queue and so cannot be closer to another object. However if v_d *is* assigned to another Voronoi node set $V_{ns}(o_j)$, then $V_{ns}(o_j)$ is added to the list of adjacent sets for $V_{ns}(o_i)$ (the search from o_j creates the reciprocal entry). The search from o_i is pruned at v_d, i.e., neighbor vertices are not inserted into the queue as they cannot belong to $V_{ns}(o_i)$.

Query Algorithm: Algorithm 1 describes how to use NVDs to retrieve candidates. By their definition, an NVD can quickly return the 1NN by looking

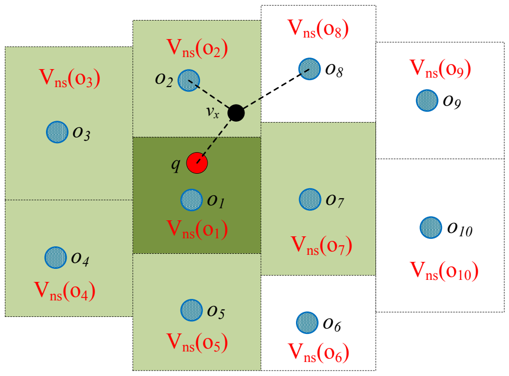


Fig. 6. Network Voronoi Diagram Query

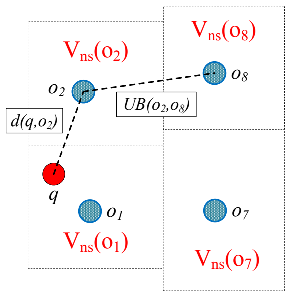


Fig. 7. LB Optimisation

up the Voronoi node set (and hence the associated object) containing the query vertex. If $k > 1$, Algorithm 1 returns the next candidate by first retrieving the adjacent Voronoi node sets of the last candidate object. Note that in the first call to Algorithm 1 the last candidate is the 1NN. Each adjacent set generates a new potential candidate, to which we compute its LB_{max} by (2) using the ALT index and insert it into priority queue Q. We use hash-table H to avoid repeated computations for previously evaluated adjacent Voronoi node sets. Once all adjacent sets are processed in this way, we return the element in Q with the minimum LB_{max} as the next candidate.

Figure 6 shows a simplified NVD, assume the dotted containers capture the Voronoi node sets of each object and when containers share an edge they are adjacent. So for query vertex q in the figure, we can retrieve the 1NN o_1 as q is contained in $V_{ns}(o_1)$. Then the adjacent Voronoi node sets of $V_{ns}(o_1)$ (lightly shaded) are used to retrieve potential candidates, which are inserted into Q by their LB_{max} values. Let us say the candidate with the minimum key is now o_7, then o_7 would be returned by the algorithm. In the next call to the algorithm, the Voronoi node sets adjacent to $V_{ns}(o_7)$ would be retrieved and, for sets not already evaluated, new potential candidates inserted into Q.

Recall that ILBR (like IER) terminates when the network distance to the kth candidate is less than the lower bound distance to the next candidate. We propose Theorem 1 to show that Algorithm 1 is correct when this occurs.

Theorem 1. *When ILBR terminates the following are true (1) priority queue Q does not contain any objects with distance smaller than the kth candidate (2) there are no objects outside the Voronoi node sets visited so far (i.e., for objects in Q or returned as candidates) with distance smaller than the kth candidate.*

Proof. Let D_k be the network distance to the kth candidate. We now prove each case of Theorem 1 individually as follows:

Case 1: When ILBR terminates we have $D_k \leq Top(Q)$. We also have $Top(Q) \leq d(q, c)$ for any object c in Q as they are inserted using a lower bound distance and Q is a minimum priority queue. Thus we also have $D_k \leq d(q, c)$ and no object c in Q has a network distance smaller than the kth candidate.

Case 2: Let $C \subseteq O$ be the set of objects inserted into Q and let $S = \{V_{ns}(o) | o \in C\}$ be the set of associated Voronoi node sets. We prove Case 2 by contradiction in a similar but simpler manner to [6]. Let us assume there exists an object $p_k \notin C$ such that $d(q, p_k) < D_k$. Algorithm 1 inserts objects into Q from adjacent Voronoi node sets beginning with the set containing q, thus all Voronoi node sets in S are adjacent to at least one other set in S. So the shortest path $P(q, p_k)$ must pass through some Voronoi node set $V_{ns}(x) \in S$ since $p_k \notin C$. Thus $P(q, p_k)$ must contain at least one vertex $v_x \in V_{ns}(x)$, as illustrated in Figure 6 with $x = o_2$ and $p_k = o_8$. By the definition of an NVD we have $d(v_x, x) \leq d(v_x, p_k)$ as all vertices in $V_{ns}(x)$ are closer to x than any other object. Adding $d(q, v_x)$ to both sides results in $d(q, v_x) + d(v_x, x) \leq d(q, v_x) + d(v_x, p_k)$. This simplifies to $d(q, x) \leq d(q, p_k)$ as v_x is on the shortest path $P(q, p_k)$ and $d(q, x) \leq d(q, v_x) + d(v_x, x)$. Since x is in Q, we have $Top(Q) \leq d(q, x)$, so we must have $D_k \leq d(q, x)$. This implies $D_k \leq d(q, p_k)$, contradicting our assumption.

NVD Lower Bound Optimisation: When parallel Dijkstra's searches meet during NVD construction, we naturally compute an upper bound distance between objects of adjacent Voronoi node sets. This is an upper bound and not an exact distance because searches are pruned (e.g., shorter paths may exist through other adjacent Voronoi node sets). A lower bound distance to an adjacent object can be computed by applying the triangle inequality to the network distance (computed by ILBR) from q to the last candidate object and this upper bound distance. For example in Figure 7, let o_2 be the last candidate returned with network distance $d(q, o_2)$. While evaluating the adjacent set $V_{ns}(o_8)$, we use the pre-computed upper bound distance $UB(o_2, o_8)$ between o_2 and o_8 to compute a lower bound $LB_{nvd}(q, o_8) = d(q, o_2) - UB(o_2, o_8)$. Note that it is not an absolute value due to the upper bound. In Algorithm 1, we insert o_8 into Q keyed by $LB_{nvd}(q, o_8)$ if $LB_{nvd}(q, o_8) > LB_{max}(q, o_8)$. The new lower bound may be tighter than the one computed using ALT especially when objects are further away from q and comes at a cheap pre-processing and query time cost.

4 Experiments

4.1 Experimental Setup

Environment: All experiments were run on a 3.2GHz Intel Core i5-4570 CPU and 32GB RAM running 64-bit Linux (kernel 4.2). Code was written in single-threaded C++ and compiled using g++ 5.2 with the O3 flag. We implemented ILBR, ALT and the candidate generation techniques ourselves. We obtained implementations of existing techniques, experimental scripts and datasets from [1]. All queries were executed in-memory for fast performance.

Datasets: We use 10 travel time road networks as in Table 1 with the largest US the default. We use a combination of synthetic and real object sets. We choose

Region	$\lvert V\rvert$	$\lvert E\rvert$
DE	48,812	119,004
VT	95,672	209,288
ME	187,315	412,352
CO	435,666	1,042,400
NW-US	1,089,933	2,545,844
CA	1,890,815	4,630,444
E-US	3,598,623	8,708,058
W-US	6,262,104	15,119,284
C-US	14,081,816	33,866,826
US	23,947,347	57,708,624

Table 1. Road Networks

Type	**Size**	**Density**
Schools	160,525	0.007
Parks	69,338	0.003
Fast Food	25,069	0.0011
Post Offices	21,319	0.0009
Hospitals	11,417	0.0005
Hotels	8,742	0.0004
Universities	3,954	0.0002
Courthouses	2,161	0.00009

Table 2. Real POI Sets (US)

synthetic objects uniformly at random based on density d where d=$|O|/|V|$. In addition we use 8 real POI sets extracted from OSM[1] as in Table 2.

Parameters: We vary object set density from 0.0001 to 1 and k from 1 to 50. We use the same default parameters as [1] with default density $d = 0.001$ and $k = 10$. We generate 25 uniform object sets and execute methods for 200 randomly selected query vertices, averaging running time over 5000 queries.

Techniques: Like IER, ILBR uses a different road network index to compute network distances. We combine ILBR with Pruned Highway Labelling (PHL) [2] as it is one of the fastest techniques. We use an ALT [4] index with 16 random landmarks to compute lower bounds and construct Object Lists. Finally we compare our techniques against the current fastest state-of-the-art technique, IER (similarly using PHL) [1]. For real-world object sets we also compare against two other techniques G-tree [16] and INE [10] for comparison with [1].

Road Network		**PHL**	**G-tree**	**ALT** (m=16)	**OL** (d=0.1%)	**NVD** (d=0.1%)	**R-tree** (d=0.1%)
NW	Time	16s	47s	2s	0.8ms	264ms	0.2MS
	Space	325MB	104MB	67MB	136KB	4.2MB	44KB
US	Time	30m	71m	60s	15ms	12s	4ms
	Space	15.8GB	2.9GB	1.43GB	1.8MB	92MB	0.9MB

Table 3. Index Statistics

4.2 Index Pre-Processing

Table 3 details the index pre-processing time and space. PHL and ALT are the road network indexes employed by ILBR and IER. While PHL is faster to construct for travel time road networks, G-tree consumes less space making it more appropriate with limited memory. The index size of ALT is small, but this is dependent on m the number of landmarks used (16 in our case). It can be reduced by using fewer landmarks at the expense of looser lower bounds. We also observe the performance of object indexes used by ILBR (Object Lists and Network Voronoi Diagrams) and IER (R-trees) for the default density $d = 0.001$.

[1] http://www.openstreetmap.org

Object Lists and R-trees are fast to construct and their index sizes are small. However since the space cost is a function of object set size we expect it to increase with density. NVDs take longer and occupy more space as the time and space complexity are functions of $|V|$. But both costs are still significantly smaller than road network indexes making it feasible to compute an NVD for each object set. NVDs may also be compressed using the geometric area of Voronoi node sets to capture vertex containment. For example it may be stored as a polygon in an R-tree [6] or as merged cells in a Quadtree [12].

4.3 Query Performance

We evaluate query performance of each technique on two metrics, namely running time and false hits per query. A *false hit* occurs when a candidate NN is not a real kNN. The greater the number of false hits, the more unnecessary network distance computations ILBR must perform. Thus false hits are an indication of a heuristic's performance irrespective of the experimental setting (disk based or main memory) or the network distance technique used. We refer to the two ILBR methods as NVD-X and OL-X as variants employing Network Voronoi Diagrams and Object Lists respectively (and X is the road network index used).

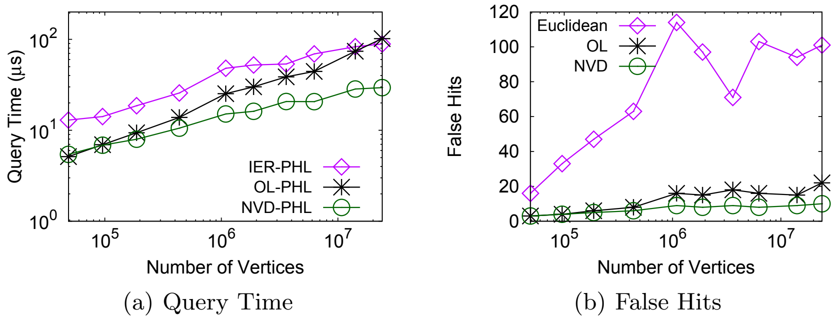
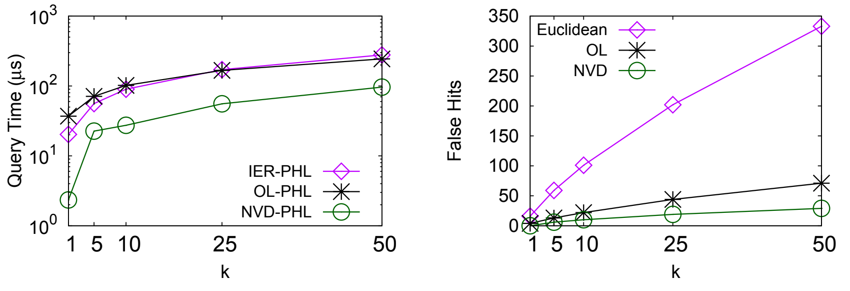


Fig. 8. Effect of Network Size $|V|$ (d=0.001, k=10, uniform)

Effect of Network Size: Figure 8 shows query performance as the number of road network vertices $|V|$ increases. In Figure 8(a), NVD-PHL is consistently the best performing method and is 2-3× faster than IER-PHL. OL-PHL is comparable to NVD-PHL for the first few datasets after which its advantage over IER-PHL narrows until being on par with it for the largest dataset. With increasing $|V|$ the total number of objects increases for the same density causing Object Lists to become larger. E.g., we expect there to be more fast food outlets in larger regions. OL is susceptible to objects that appear close when they are similar distances from the landmark as the query vertex. When $|V|$ increases, landmarks become more distant from query vertices on average (as the number

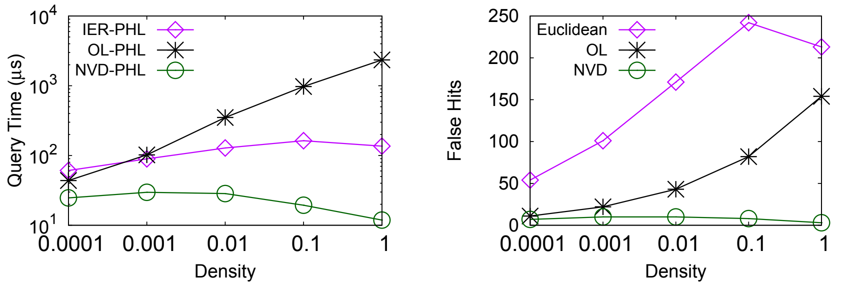


(a) Query Time (b) False Hits

Fig. 9. Effect of k (US, d=0.001, uniform)

of landmarks m is constant), so the probability of such objects appearing increases. While these are only *potential candidates* and are not reflected in false hits, OL must still compute their LB_{max} values. This is evident in Figure 8(b) as the number of false hits for OL is still lower than Euclidean distance.

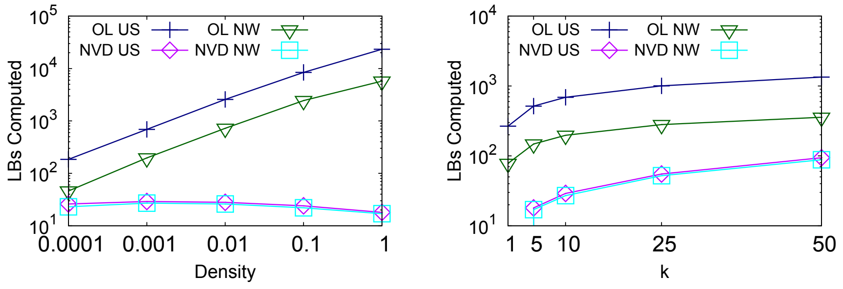
Effect of k: Figure 9 shows the query performance with increasing k. For k=1, NVD-based methods are essentially optimal as only a single look-up operation is needed. NVD-PHL once again outperforms all other methods, being at least 2-3× faster than IER-PHL over all k, again showing that it *is* possible to efficiently use landmarks for kNNs. Landmarks display significant improvement on false hits over Euclidean distance in Figure 9(b). But earlier trends are also seen here and OL-PHL's query time does not improve on IER-PHL.



(a) Query Time (b) False Hits

Fig. 10. Effect of Density (US, k=10, uniform)

Effect of Density: We observe query performance with increasing object set density in Figure 10. As density increases, the average distance between objects decreases. This makes kNNs appear closer to the query vertex and they should be easier to find. IER-PHL is an exception, as objects become closer and more numerous they become more difficult to differentiate using Euclidean distance. NVD-PHL shows this problem can be remedied using landmarks as it is an order of magnitude better than IER-PHL in Figure 10(a). OL-PHL however



(a) Varying Density (b) Varying k

Fig. 11. No. of Lower Bounds Computed (US, d=0.001, k=10, uniform)

degrades with increasing density to the point that its running time is an order of magnitude worse than IER-PHL. With more objects, more of them will produce inaccurate lower bounds similar to the scenario depicted in Figure 4, making distant objects appear close to the query vertex. NVD-PHL does not suffer from this as using adjacent Voronoi node sets acts as a filter avoiding objects that "seem" close by inaccurate lower bounds. As a result NVD-PHL experiences far fewer false hits in Figure 10(b).

Lower Bounds Computed: Figure 10(b) showed that with increasing density, OL experiences fewer false hits than Euclidean distance even at its worst. This suggests that the poor running time of OL for high densities in Figure 10(a) is not caused by ILBR making additional network distance computations due to false hits. It is actually due to the number of lower bounds computed by OL, which increases with density, as illustrated in Figure 11(a). NVD computes very few lower bounds thanks to its filtering property. The final evidence of this is the behaviour of OL on the two datasets in Figure 11. The US road network with 24 million vertices requires more lower bounds to be computed than the smaller NW dataset with 1 million vertices. The US has more objects for the same density, resulting in a larger Object List and hence a larger search space to find the best object. We note however, computing all lower bounds would require significantly more computations than OL. While OL is a substantial improvement, its utility is still dependent on the number of objects.

Real-World Object Sets: We verify our observations on real-world POIs in Figure 12 with increasing object set sizes from left to right. Trends seen in previous figures are also observed for real-world POIs. NVD-PHL is the overall best performing method, while OL-PHL is competitive except on larger object sets like parks and schools. For small object sets like courts, IER-PHL remains competitive as there are so few objects that Euclidean distance has a smaller probability of making a false hit. A more typical object set such as fast food outlets demonstrates the significant superiority of NVD-PHL.

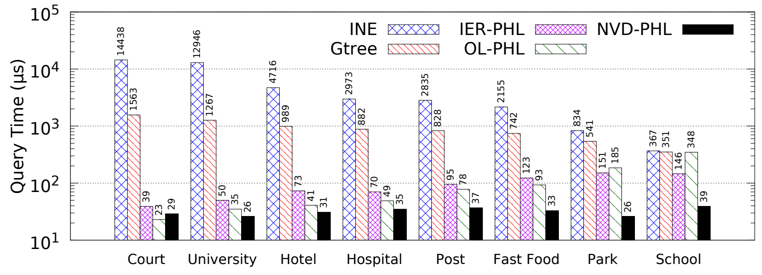


Fig. 12. Real-World POIs

5 Related Work

A recent experimental study [1] on the kNN problem provided an in-depth review of the state-of-the-art. The main outcomes of this study were the surprising performance of IER and the implications this had on heuristics used in kNN. We refer the reader to this paper for a detailed review of kNN techniques, while in this section we discuss the work most relevant to our study.

VN^3 [6] uses Network Voronoi Diagrams to answer kNN queries as explained in Section 3.2. Landmarks have been used to answer kNN queries by Kriegal et al. [7, 8] based on the multi-step kNN paradigm [13] that we refer to as ILBR. These studies propose interesting improvements to using landmarks. But in both cases the kNN algorithms require creating a ranking by computing landmark lower bounds to all objects. As discussed this approach is not scalable with object set size and not competitive with existing approaches in practice.

Road Network Embedding (RNE) [14] involves transforming the road network into higher dimensional space and using Minkowski metrics to estimate network distance. However the proposed kNN method is approximate. Qiao et al. also propose an approximate technique [11] using shortest path trees to compute distance estimates based on tree distance, but their solution applies to the k nearest *keyword* problem where objects are not split into sets.

There is a wide body of work on utilising landmarks for shortest path queries. As previously discussed, ALT [4] was among the first. [8] proposes a hierarchical landmark scheme to reduce index space cost. Other work [5] has focussed on improving lower bounds for example through better landmark selection. These compliment our work, e.g., better lower bounds reduce the number of false hits.

6 Conclusion

In this paper we present two techniques to efficiently retrieve kNN candidates by landmark-based lower bounds in an incremental manner for effective integration with the ILBR framework. We empirically compare the heuristic performance of landmark lower bounds and Euclidean distance on kNN search for the first time. We show that both methods significantly improve on the number of false hits (by up to an order of magnitude) incurred in candidate generation than the Euclidean distance heuristic used by IER. In our experimental investigation on travel time road networks, the Object List technique demonstrates the difficulties in using landmarks but outperforms IER for smaller object sets. The second technique employing a Network Voronoi Diagram outperforms IER by at least 2-3× on query time across all datasets and parameters. Thus we show that it is indeed possible to use landmark-based lower bounds to improve kNN search.

Acknowledgements. We sincerely thank the anonymous reviewers for their feedback which helped improve our work. The research of Muhammad Aamir Cheema is supported by ARC DE130101002 and DP130103405. Tenindra Abeywickrama is supported by an Australian Government RTP Scholarship.

References

1. Abeywickrama, T., Cheema, M.A., Taniar, D.: K-nearest neighbors on road networks: A journey in experimentation and in-memory implementation. PVLDB 9(6), 492–503 (2016)
2. Akiba, T., Iwata, Y., Kawarabayashi, K.i., Kawata, Y.: Fast shortest-path distance queries on road networks by pruned highway labeling. In: ALENEX. pp. 147–154 (2014)
3. Erwig, M., Hagen, F.: The graph voronoi diagram with applications. Networks 36, 156–163 (2000)
4. Goldberg, A.V., Harrelson, C.: Computing the shortest path: A search meets graph theory. In: SODA. pp. 156–165 (2005)
5. Goldberg, A.V., Werneck, R.F.F.: Computing point-to-point shortest paths from external memory. In: ALENEX. pp. 26–40 (2005)
6. Kolahdouzan, M., Shahabi, C.: Voronoi-based k nearest neighbor search for spatial network databases. In: VLDB. pp. 840–851 (2004)
7. Kriegel, H.P., Kröger, P., Kunath, P., Renz, M.: Generalizing the optimality of multi-step k-nearest neighbor query processing. In: SSTD. pp. 75–92 (2007)
8. Kriegel, H.P., Kröger, P., Renz, M., Schmidt, T.: Hierarchical graph embedding for efficient query processing in very large traffic networks. In: SSDBM. pp. 150–167 (2008)
9. Okabe, A., Boots, B., Sugihara, K.: Spatial Tessellations: Concepts and Applications of Voronoi Diagrams. John Wiley & Sons, Inc., 2nd edn. (2000)
10. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: VLDB. pp. 802–813 (2003)
11. Qiao, M., Qin, L., Cheng, H., Yu, J.X., Tian, W.: Top-k nearest keyword search on large graphs. PVLDB 6(10), 901–912 (2013)
12. Samet, H., Sankaranarayanan, J., Alborzi, H.: Scalable network distance browsing in spatial databases. In: SIGMOD. pp. 43–54 (2008)
13. Seidl, T., Kriegel, H.P.: Optimal multi-step k-nearest neighbor search. In: SIGMOD. pp. 154–165 (1998)
14. Shahabi, C., Kolahdouzan, M., Sharifzadeh, M.: A road network embedding technique for k-nearest neighbor search in moving object databases. GeoInformatica 7(3), 255–273 (2003)
15. Zheng, B., Zheng, K., Xiao, X., Su, H., Yin, H., Zhou, X., Li, G.: Keyword-aware continuous knn query on road networks. In: ICDE. pp. 871–882 (2016)
16. Zhong, R., Li, G., Tan, K., Zhou, L., Gong, Z.: G-tree: An efficient and scalable index for spatial search on road networks. TKDE 27(8), 2175–2189 (2015)