DIKNN: An Itinerary-based KNN Query Processing Algorithm for Mobile Sensor Networks

Shan-Hung Wu[†‡♯] Kun-Ta Chuang[†] Chung-Min Chen[‡] Ming-Syan Chen[†]

[†]Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, ROC
[‡]Telcordia Applied Research Center, Taipei, Taiwan, ROC
[♯]Industrial Technology Research Institute, Hsinchu,Taiwan, ROC
{brandonwu@arbor.ee.ntu.edu.tw, doug@arbor.ee.ntu.edu.tw, chungmin@research.telcordia.com, mschen@cc.ee.ntu.edu.tw}

Abstract

Current approaches to K Nearest Neighbor (KNN) search in mobile sensor networks require certain kind of indexing support. This index could be either a centralized spatial index or an in-network data structure that is distributed over the sensor nodes. Creation and maintenance of these index structures, to reflect the network dynamics due to sensor node mobility, may result in long query response time and low battery efficiency, thus limiting their practical use. In this paper, we propose a maintenance-free, itinerary-based approach called Density-aware Itinerary KNN query processing (DIKNN). The DIKNN divides the search area into multiple cone-shape areas centered at the query point. It then performs a query dissemination and response collection itinerary in each of the cone-shape areas in parallel. The design of the DIKNN scheme also takes into account challenging issues such as the the dynamic adjustment of the search radius (in terms of number of hops) according to spatial irregularity or mobility of sensor nodes. The simulation results show that DIKNN yields substantially better performance and scalability over previous work, both as k increases and as the sensor node mobility increases. It outperforms the second runner with up to 50% saving in energy consumption and up to 40% reduction in query response time, while rendering the same level of query result accuracy.

1 Introduction

The problem of efficient K Nearest Neighbors (KNN) search in a spatial or multi-dimensional database has been a major research topic in the literature [11, 21, 22, 29, 30]. Traditional KNN query processing techniques assume location data are available in a centralized database and focus on improving the index performance [10, 11, 12, 23]. In new applications where data sources are geographically spreaded (e.g., sensor networks [3, 15], wireless ad-hoc networks [2], *Intelligent Transportation Systems* (ITS) [26], and battlefield surveillance systems [25]), pulling data from a large number of data sources (e.g., sensor nodes, laptops or vehicles, etc.) is generally infeasible due to high energy consumption, high communication cost, or long latency [4, 31]. Recently, a number of studies have explored "in-network" KNN query processing techniques for sensor networks [7, 21, 22, 29, 30]. These techniques rely on certain in-network infrastructure—index or data structures (e.g., clustered indices or spanning trees) distributed among the sensor nodes—to select KNN candidates, propagate queries, and aggregate the result.

Although these in-network approaches avoid the overhead of periodical data gathering from a large number of sources, still they are posed some drawbacks if to be deployed in large-scale mobile sensor networks. First, the distributed indexing structures may become too costly to maintain when the number of nodes increases, due to the communication overhead among the nodes. Second, certain techniques [7, 21] require some nodes to act as supernodes, for example, as clusterheads or data aggregation points. These supernodes may easily turn into a bottleneck of the system. Furthermore, current in-network based KNN techniques [7, 21, 22, 29, 30] have all assumed a fixed network where sensor nodes are stationary and never fail. This assumption makes them inept for a network environment where sensor nodes are mobile and packet loss is the norm rather than an exception [4, 14, 16], as the maintenance overhead of the in-network indexing structure could be considerable.

In this paper, we propose a Density-aware Itinerary KNN query processing (DIKNN) for mobile sensor networks that does not rely on any sort of in-network indexing structure.

1-4244-0803-2/07/$20.00 ©2007 IEEE.

The key idea of DIKNN is to let sensor nodes collect partial results and propagate the query along a well-devised, conceptual itinerary structure. No physical maintenance of this itinerary structure is required. The DIKNN divides (conceptually) a circular search boundary centered at the query point q into multiple cone-shape areas. It then performs concurrent itinerary traversal, based on the itinerary structure, to the nodes in each of these areas. The traversal length is adjusted dynamically according to the node distribution information it collects as the traversal proceeds.

To the best of our knowledge, DIKNN is the first KNN processing technique for mobile sensor networks that does not rely on any in-network indexing structure support: no constant maintenance or fixed data aggregation point is needed. Because of this, DIKNN is able to avoid potential bottleneck and survive rapid changes of network topology. It also reduces query response time (or latency for short) by combining data collection with query propagation in a well devised itinerary.

Several challenging issues arise in the design of DIKNN, such as estimate of search radius and design of efficient itinerary. We investigate and present solutions to each of these issues. The simulation results also show that DIKNN yields substantially better performance scalability over previous work, both as k increases and as the sensor node mobility increases. In particular, it outperforms the second runner [29, 30] with up to 50% saving in energy consumption and up to 40% reduction in query response time, while rendering the same level of query result accuracy.

The rest of the paper is organized as follows. Section 2 reviews the previous studies on KNN query processing and related work to DIKNN. Section 3 presents the design of DIKNN. In Section 4, algorithms determining the KNN boundary are introduced. We also discuss interactions between DIKNN and network environments. Section 5 reports our performance evaluation based on simulation results. Section 6 concludes the paper.

2 Related Work

Previous work on KNN or window (range) query processing in sensor networks can be generally classified into two categories: the *centralized* and *in-network* approach, as shown in Figure 1. The centralized approach performs the queries in a centralized database containing locations of all the sensor nodes [10, 11, 12, 23]. These location data are usually maintained in an R-tree variant index modified to handle mobile objects. In contrast, the in-network approach does not rely on a centralized index, instead, it propagates the query directly among the sensor nodes in the network and collects relevant data to form the final result [5, 6, 7, 21, 22, 29, 30]. This approach is favored when maintenance of a centralized index is expensive or may impose high energy consumption on the sensor nodes. And it happens, for example, when the locations of the nodes change frequently, or when there is no direct wireless link between the nodes and the centralized index and thus substantial message routing/relay overhead among the nodes will incur.

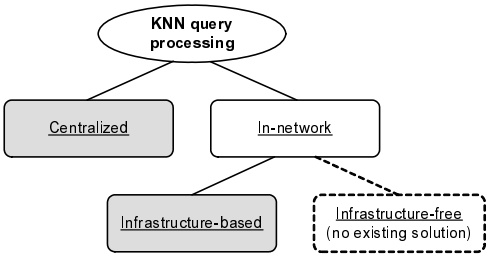

Figure 1. Categorization of previous studies.

The in-network approach can be further divided into two sub-categories: those relying on certain sort of in-network infrastructure and those that are infrastructure free. The term "infrastructure" refers to a data structure distributed among the sensor nodes that is created, either once on-the-fly or to be updated constantly, to support the query processing. Works of [7, 21, 22, 29, 30] are representatives of this kind targeting at KNN queries for fixed sensor networks. Maintenance of the in-network data structure could become costly prohibitive, if not infeasible, when the sensor nodes become mobile. To eliminate this problem, an infrastructure-free method was proposed in [31] but it applies to window query only. Recently, a number of works have addressed continuous query using in-network techniques [5, 6, 11, 23]. These methods are good for constant monitoring of queries of long-standing interest but do not suit well for on-demand queries (one time only) that is the focus of our work.

We will briefly describe the Peer-tree [7], DSI [21], and KPT [29, 30] as they are most relevant to our work. They will also be used in a competitive performance evaluation with our proposed method. The Peer-tree [7] and DSI [21] decentralize the index structures (e.g., R-tree [10]) to distributed environments. As shown in Figure 2(a), a network is partitioned into a hierarchy of Minimum Bounding Rectangles (MBRs). Each MBR covers a geographical region including all sensor nodes located inside. An MBR in the higher hierarchy (say, region A in Figure 2(a)) covers all the regions of the sub-MBRs in the child hierarchy (regions B, C, and D in Figure 2(a)). One specific node is designated as a clusterhead (i.e., distributed index) in each MBR. A clusterhead knows locations and identities (IDs) of all the other nodes within the MBR. It also knows locations and IDs of its parent and child clusterheads. To handle an NN query, the source node s routes the query message to its clusterhead (node E in Figure 2(a)). Upon receiving the message,

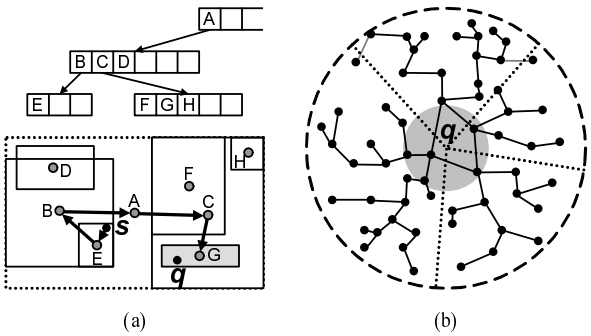

Figure 2. Related work: the decentralized R-tree approaches and KPT.

the clusterhead forwards it upward in the hierarchy until that the query point q is covered by the MBR of a clusterhead (in this case, node A in Figure 2(a)). The clusterhead then forward the message downward in the hierarchy looking for a child clusterhead (node G in Figure 2(a)) that contains q with minimal MBR. After that, location of NN of q can be determined and the NN is informed of the query message by unicast. Supporting of KNN queries is more complicated that needs multiple clusterheads to find and to propagate the query message in different MBRs. Since every query message goes through the clusterheads, the major problem of these approaches is that index nodes become system bottlenecks easily. Such approaches are vulnerable to index failure. In addition, there are many unnecessary hops from s to the KNN nodes because each query message is routed along the hierarchy of clusterheads, as depicted by the arrows in Figure 2(a). Such overhead becomes significant in the large-scale sensor networks where the distance between clusterheads is long.

To address such issues, the KPT [29, 30] is proposed to handle the KNN query without fixed indexing. This work assumes each sensor node is location-aware. After a query is issued from s, it is routed to the sensor node, named *home node*, closest to q. To avoid flooding the entire network, a conservative boundary containing at least k candidates is estimated by the home node. Multiple trees rooted at the home node are then constructed to propagate queries and to aggregate data, as shown in Figure 2(b). Upon aggregating data at the home node, it determines correct KNNs (by sorting locations) and transmits their query responses back to the source s. KPT assumes an optimal network condition where each node is stationary. It encounters two serious drawbacks in presence of mobility. First, constructing or maintaining the trees while sensor nodes are moving incur considerable overhead. Partially collected data may be forwarded again and again between new and old tree nodes. Secondly, the conservative (large) boundary grows quadratically as k increases, which leads to high energy consumption and long latency. Although such a boundary is expected to cover at least k nodes in the worst case, sensor nodes may either move in or move out the boundary during tree construction and data aggregation. KPT returns poor query result accuracy.

In light of the above problems, we propose DIKNN, which to our best knowledge, is the first infrastructure-free KNN search method for mobile sensor networks. Nevertheless, the concept of itinerary traversal is inspired by a number of research efforts in unicast routing [24], data fusion [28], network surveillance [9], and window query processing [31]. Our main contribution lies on the origination of a sophisticated itinerary structure (taking into account the important factors such as itinerary width, data collection scheme, adaptive search boundary estimation, forwarding heuristics, etc) that offers both high efficiency and high flexibility in parallel query dissemination and processing.

3 Design of DIKNN

In this section, we first give a formal definition of our problem. Then we describe the three execution phases in DIKNN.. Design of node traversal itinerary/sub-itinerary are detailed thereafter.

3.1 Definitions and Network Model

In this paper, we focus on snapshot queries, which expect to obtain the query result only once during their lifetimes. The KNN problem is defined as follows:

Definition 1 (k nearest neighbor problem) *Given a set of sensor nodes S, a geographical location q (i.e., query point), and valid time T, find a subset S' of S with k nodes (i.e., $S' \subseteq S$, $|S'| = k$) such that at time T, $\forall n_1 \in S', n_2 \in S - S' : DIST(n_1, q) \leq DIST(n_2, q)$, where $DIST$ denotes the Euclidean distance function.*

Ideally, we would like to obtain the exact result set S' comprising the k nearest neighbors of q at the given time T. However, due to node mobility and efficiency considerations [4, 31], we may accept an approximate result set. Query result accuracy is measured by the percentage ratio the correct KNNs (at valid time T) are returned. Depending on different application needs, the valid time T can be defined either as the time the query is issued (snapshot results are better) or the time the result set is received (newer results are better). In our evaluation, measurements of accuracy according to these two types of valid time are called *pre-accuracy* and *post-accuracy* respectively.

We assume the network is under ad-hoc mode so that multi-hops between nodes are required to relay messages.

We assume that all sensor nodes can store data locally and answer the queries individually. In addition, the moving speed and directions of sensor nodes are arbitrary. Each sensor node is aware of its geo-location. Beacons with locations and identities (IDs) are periodically broadcasted. Every sensor node also maintains a table enrolling IDs and locations of neighbor nodes falling within its radio range, r. Note this network scenario has been assumed in [31] and complies with IEEE standard 802.15.4 [3], the *Low Rate Wireless Personal Area Network* (LR-WPAN), to achieve maximum compatibility.

3.2 Execution Phases

The execution of DIKNN consists of three phases:
1). *Routing phase*: A query message Q is geographically routed from the sink node s to the nearest neighbor (i.e., the home node n_p, where p denotes the number of hops along the routing path) around the query point q. Information of the sensor network is gathered along with the routing procedure without the aid of any infrastructure.
2). *KNN boundary estimation phase*: Upon receiving Q and the collected information from the previous phase, the home node estimates a searching boundary, named KNN boundary, with radius R by using an efficient (specifically, linear time) KNNB algorithm. The estimated boundary is not fixed and will be dynamically adjusted (by the other nodes) as long as additional information is available in the next phase.
3). *Query dissemination phase*: The home node disseminates the query message to all sensor nodes inside the KNN boundary. Dissemination follows parallel itinerary structures, and query responses are aggregated along with each itinerary traversal. At the end of dissemination, the aggregated query responses are directly routed back to the sink in a bundle.

Next, we explore the main phase of DIKNN, the query dissemination phase, by assuming that KNN boundary is given. The routing and KNN boundary estimation phases will be visited later.

3.3 Itinerary-Based Query Dissemination

Once the KNN boundary (and its radius R) is determined, the home node n_p enters the query dissemination phase aiming to inform all the sensor nodes inside the boundary of the query message Q, and to collect their responses. As the infrastructure-based technique leads to considerable overhead in dynamic environments, we turn to explore an infrastructure-free technique. One naive infrastructure-free solution is to flood the query within the boundary. Each node inside the boundary, upon receiving Q, routes its response back to s end-to-end and then broadcasts Q again. This approach, however, is extremely resource-consuming and has poor scalability because of the excessive number of independent routing paths from sensor nodes to s [31]. In addition, serious degree of collision and hidden terminal problem may also occur during the wireless transmission. To address these issues, DIKNN adopts an itinerary-based dissemination technique, which provides robust and effective query processing under transient network topologies.

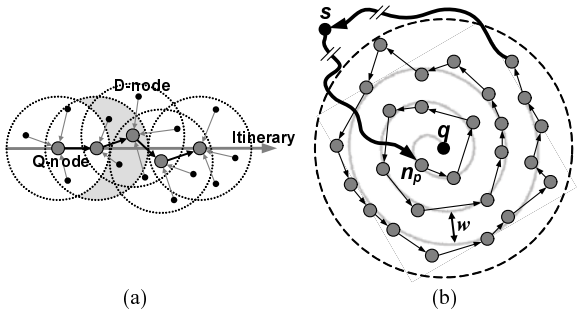

Figure 3. Itinerary-based query dissemination.

The concept of itinerary query dissemination [24, 28, 9, 31] can be best understood by the illustration in Figure 3(a). A set of *Query nodes* (Q-nodes) in the KNN boundary are chosen for query dissemination. Upon receiving a query, a Q-node broadcasts a *probe* message that includes information about Q, R, and the itinerary (e.g., itinerary width and the number of sectors, which will be discussed later). When hearing the probe message, the neighbor nodes that are qualified to reply the query, called *Data nodes* (D-nodes), report their query response back to the Q-node. After obtaining the data from all D-nodes as well as the partial result received from the previous Q-node, the current Q-node selects the next Q-node based on the itinerary information, and forwards this new partial query result to the selected next Q-node. This procedure repeats until the query traverses the entire KNN boundary along a pre-defined (say, spiral) itinerary structure, as shown in Figure 3(b). Responses of all nodes held by the last Q-node are then returned back to the sink node s in a single message.

Primitives of itinerary-based solution. Some useful primitives have been proposed in [31] to ensure correctness of itinerary execution. At first, the itinerary width w, as shown in Figure 3(b), specifies how close between segments of an itinerary. Obviously, a small w results in denser itinerary traversal which ensures the KNN boundary to be fully covered by the traversal. On the other hand, the small w incurs unnecessary transmission and long latency because of the increased itinerary length. It is shown that letting $w = \sqrt{3}r/2$ yields full coverage with minimal itinerary length, a good balance on query accuracy and energy efficiency. Secondly, data collection from multiple D-

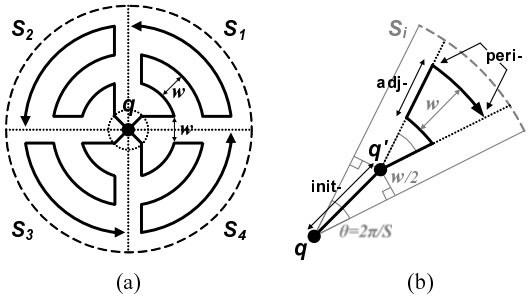

Figure 4. Concurrent query dissemination.

nodes needs to be better scheduled to avoid collisions and delays. The *contention-based data collection scheme*[1] can be utilized to prevent serious contention and to sustain network dynamics. In this scheme, a reference line emanating form the current Q-node is included in the probe message. The probe message also contains a precedence list indicating the reply order of D-nodes. Upon receipt of the probe message, each D-node sets a timer with $timer = (\frac{\alpha}{2\pi})im$, where α is the angle formed by the specified reference line and the line connecting the current Q-node and the current D-node, i is the received precedence, and m is a time unit for the Q-node waiting for each D-node to report its data. A D-node does not respond to the Q-node until its timer expires. Discussions on the other issues such as fault tolerance, and traveling in low-connectivity areas with *itinerary voids* (i.e., situations when a Q-node cannot find the next Q-node for query forwarding) can be found in [17, 31]. ■

Clearly, the performance of dissemination solely depends on the structure of an itinerary, along which Q is propagated and responses are forwarded. Query latency can be significantly improved by considering the parallel dissemination. Nevertheless, concurrent dissemination may increase the likelihood of channel contention and collision at the data link and physical layers, causing degradation of network throughput. Parallelization should be exercised cautiously to avoid the overhead, and should satisfy the following criteria. First, the number of routing paths leading back to the sink s, after dissemination, should be controllably small to prevent high energy consumption in large-scale sensor networks. Second, as concurrent itinerary traversals may incur channel interference in wireless transmission, the chance they meet should be as small as possible. Unfortunately, the only study [31] that mentions parallelization cannot scale well to high concurrency level due to its assumption upon the query range and itinerary structure.

Concurrent itinerary structures. To fulfill the above criteria, a KNN boundary is partitioned into multiple sectors, as shown in Figure 4(a). In each sector, the query is propagated along a sub-itinerary. The distance between sub-itineraries in adjacent sectors is w to ensure full coverage of the KNN boundary when $w \leq \sqrt{3}r/2$. Each sub-itinerary consists of three segments: the init-, adj- and peri-segments, as illustrated in Figure 4(b). The init-segment is a portion of sub-itinerary which has a distance less than $w/2$ to either side of a sector's border. This segment is formed by a straight line to get rid of the interference as soon as possible. Specifically, let S be the number of sectors and l_{init} be the length of the init-segment. Then we have $\sin(\theta/2) = \sin(\pi/S) = (w/2)/l_{init}$, which gives $l_{init} = \min\{w/(2\sin(\pi/S)), R\}$. Let q' denote the end of the init-segment. The peri-segments are portions of the sub-itinerary which together form perimeters of concentric circles centered at q'. Let l_{peri} be the total length of the peri-segments, then $l_{peri} = \sum_{i=1}^{\lfloor (R-l_{init})/w \rfloor} \frac{2\pi(iw)}{S}$, where $2\pi(iw)/S$ denotes the perimeter length of the i^{th} concentric circle, and $\lfloor (R-l_{init})/w \rfloor$ denotes the number peri-segments. The adj-segments are portions of the sub-itinerary that are parallel to either side of the sector's border. It is clear that each adj-segment has the same length of w. The total length of the adj-segments l_{adj} therefore equals $l_{adj} = \lfloor (R-l_{init})/w \rfloor\, w$.

Ideally, two sub-itineraries in adjacent sectors interfere with each other only at their init-segments. An important observation is that even if these two sub-itineraries are traversed in different speeds, extra interference can only occur at adj-segments, which are relatively short as compared to peri-segments. Such a cone-shape itinerary structure is highly adaptive to various degrees of parallelism. At an extreme, the shape of a sub-itinerary degenerates into a straight line if S is large enough. This allows the best efficiency when no interference can ever occur in the sensor network (e.g., when *Contention Free Period* (CFP) is exercised in LR-WPAN).

4 KNN Boundary Estimation

To precisely estimate the KNN boundary without the aid of supernodes containing long-term monitored (and cached) information is a challenging issue because decision must be made with very limited knowledge that can only be obtained from query propagation. DIKNN adopts a simple, but effective, algorithm named KNNB tailored for sensor nodes with limited ability.

4.1 Routing Phase

In the routing phase, a query Q is routed from sink node s to the nearest neighbor n_p (i.e., the home node) around the query point q, where p denotes the number of hops along

[1] As suggested by our simulation result, the data collection scheme introduced in this paper combines both the *token ring based* and *contention based* scheme to achieve higher performance. For detailed discussion on these two schemes, please refer [31].

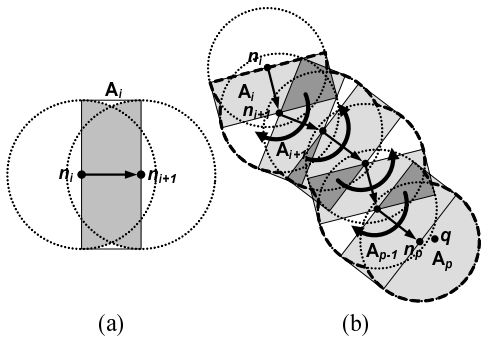

Figure 5. KNN boundary estimation.

the routing path. Any geographic face routing protocol [17, 19] is compatible with DIKNN. Ensuring correctness and efficiency of these routing protocols in the sensor network is an orthogonal issue to DIKNN, which is studied in the literature [18, 20].

By utilizing the geographic face routing protocol, information collection is performed between hops. An additional list L is sent along with Q. On the i^{th} $(1 \leq i < p)$ hop to the destination, the corresponding node (i.e., the sensor node triggering the i^{th} hop) appends its own location loc_i and the number of newly encountered neighbors enc_i to L. To avoid duplicate information, enc_i can simply be counted by checking the number of neighbors having a distance larger than r from the corresponding node of the $(i-1)^{th}$ hop. Note that in comparison with the previous studies [5, 7, 21, 6], such an information gathering technique consumes very few extra recourses since merely nodes around the route is involved.

4.2 Linear KNNB Algorithm

Upon receipt of the query message and the list L from the previous phase, n_q starts estimating the KNN boundary by determining its radius length R. As described previously, too large a boundary (as the one given by KPT [29]) incurs great energy consumption and long latency. In contrast, a small boundary loses the query accuracy. Determination of R must balance two conflicting factors: (1) increasing R to enclose correct KNN points as many as possible; (2) decreasing R to reduce the energy consumption.

In most of the previous work, sensor nodes are thought to be uniformly distributed in the network and sometimes even to form a grid. However, the recent investigation [8] argues that spatial irregularity happens in majority of the cases. Regarding to the scarce resource that prohibits complex analysis of the sensor distribution, for now we content ourselves that KNNB adopts a weaker assumption: sensor nodes are uniformly distributed only *within the optimal KNN boundary* (i.e., the boundary containing exactly k nearest neighbors). A novel and cost-efficient solution will be discussed later to remit the bias of estimation.

Let $\widetilde{R}$ be the radius of the optimal KNN boundary and n_i be the corresponding node of the i^{th} hop in the routing path which locates inside the optimal KNN boundary. We have $\overline{loc_i q} \leq \widetilde{R}$. Consider a circle centered at q with radius $\overline{loc_i q}$. From assumption of uniform distribution, we can estimate the number of nodes est_k locating inside the circle by using the equation:

$$D \approx \frac{est\_k}{\pi\left(\overline{loc_i q}\right)^2} \approx \frac{\sum_{j=i}^{p} L.enc_j}{\textit{Area covered from } n_i \textit{ to } n_p}, \quad (1)$$

where D denotes density of nodes $(nodes/m^2)$ within the optimal KNN boundary. Thus we have

$$est\_k \approx \frac{\pi(\overline{loc_i q})^2 \sum_{j=i}^{p} L.enc_j}{\textit{Area covered from } n_i \textit{ to } n_p},$$

while leaving the coverage area (the long-dotted line shown in Figure 5(b)) along the routing path from n_i to n_p undetermined. Since correct evaluation of this area is too complicated to be executed on a sensor node given its limited computing power, we need an easy approximation. Observing that a sensor node will always find the next hop within its radio coverage, thus advance of a single hop must be less than r. As depicted in Figure 5(a), the coverage area between two sensor nodes of successive hops is large enough to be approximated by using a rectangle A_i. Putting rectangles A_i, A_{i+1}, ..., A_{p-1} from n_i to n_{p-1} together, the symmetric property shown by the arrows in Figure 5(b) helps the doubly-counted regions (heavily-shaded in Figure 5(b)) to complement the opposite areas against the routing path. Thus summing these rectangles with an additional semicircular region A_p yields an easily calculated but close approximation to the coverage area from n_i to n_p.

From the above, we now have est_k. Back to our problem of estimating the KNN boundary, applying Eq. (1) we have

$$D = \frac{k}{\pi\widetilde{R}^2} \approx \frac{est\_k}{\pi\left(\overline{loc_i q}\right)^2}.$$

While $\widetilde{R}$ is unknown, approaching est_k to k yields $R = \overline{loc_i q} \approx \widetilde{R}$. Algorithm 1 below shows the detail steps of KNNB.

The list L is indexed from one. In lines 4-13, KNNB iteratively approaches est_k to k by examining L from the tail. The function $DIST(a,b)$ at line 5 simply returns the distance between the given two positions a and b, and the function $APPROX(a,b)$ at line 11 gives the approximative rectangle by returning $r \cdot DIST(a,b)$. The complexity of KNNB is $O(n)$, where n is the number of hops in the routing path.

Note the experimental results reveal that radius lengths returned by KNNB are generally $1/\sqrt{k\pi}$ of the previous

Algorithm 1 The KNNB algorithm: KNNB(L, q, r, k)

Require: information list L gathered from the first phase of DIKNN, the query point q, radius r of a sensor node, and number k of nearest neighbors to be found.

Ensure: returning radius length R of the KNN boundary.

1: $i = L.length - 1$;
2: $neighbors = L.enc_i$;
3: $approx\_area = \pi r^2/2$;
4: **while** $i \geq 0$ **do**
5: d =DIST($L.loc_i$, q);
6: $est\_k = \pi d^2(neighbors/approx\_area)$;
7: **if** $est\_k \geq k$ **then**
8: return d;
9: **end if**
10: $neighbors$ += $L.enc_{i-1}$;
11: $approx\_area$ += APPROX($L.loc_i$, $L.loc_{i-1}$);
12: $i = i - 1$;
13: **end while**

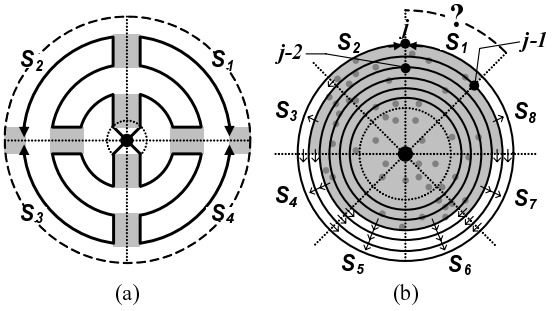

Figure 6. Dynamic search boundary adjustment according to the spatial irregularity.

work KPT [29, 30] under the same level of accuracy. An example finding for five nearest neighbors ($k = 5$) is depicted in Figure 2(b). The shaded KNN boundary determined by KNNB is much smaller than the long-dotted one estimated by KPT.

4.3 Interaction with Environments

Next, we discuss some mechanisms adopted by DIKNN to face spatial irregularity and mobility in real-world sensor environments.

Spatial Irregularity. The KNNB algorithm estimates the KNN boundary by assuming that sensor nodes are uniformly distributed around the query point q. This assumption is valid for a small region due to the spatial locality; however, when k is large, the sensor nodes tend to irregularly spread and their spatial density becomes unpredictable [8]. This effect, called *spatial irregularity*, may degrade the query accuracy. To handle this problem, we let the Q-nodes in different sectors adjust its own R during dissemination. Specifically, we inverse the direction of peri-segments in every interseptal sector. In such a configuration, the face-to-face adj-segments of different sub-itineraries together form *rendezvous* (as shown by the shaded area in Figure 6(a)), in which two Q-nodes from adjacent sub-itineraries can, with little cost of latency, meet with each other and exchange the latest statistics (e.g., total number of nodes explored so far). By repeating this procedure, the j^{th} rendezvous in a sub-itinerary can obtain information from 2, 4, ..., $\min\{2j, S\}$ nearby sectors at the j^{th}, $(j-1)^{th}$, ..., 1^{st} level of the peri-segments respectively (as depicted by the shaded area in Figure 6(b)). Each sector, say S_1 in Figure 6(b), can then infer how many nodes around q are explored so far (totally) and dynamically adjust R to stop or to continue the dissemination. With rendezvous, itinerary traversals can stop immediately if k nearest neighbors are discovered before reaching the perimeter of the KNN boundary, or continue if fewer nodes are found. Note a simple bilinear interpolation can be used to complement the not-yet-exchanged information from the other sectors (as shown by the small arrows in Figure 6(b)) when S_1 is doing the inference.

Mobility Concern. Mobility of sensor nodes degrades query accuracy because nodes may move in or move out the KNN boundary during dissemination. For applications to which discovering correct KNNs are the most important concern, support of flexible expansion of R to include more candidates is critical. One naive approach is to modify the KNNB algorithm so that $R' = c \cdot R$ is returned, where R' denotes the adjusted radius of the KNN boundary and c, $c > 1$, denotes a constant. Obviously, the larger c, the more correct KNNs to be guaranteed by R'; however, more energy is consumed as well. This makes it very difficult for an application to determine a good value of c feeding its own need. In DIKNN, we address this issue in the query dissemination phase, where the last Q-node is obligated to determine how much farther a sub-itinerary should continue. Specifically, each application is allowed to specify an attribute, named *assurance gain* g, $0 \leq g \leq 1$, at the time when a KNN query is issued. By acquiring the moving speed of each sensor node along with data collection, each sub-itinerary can maintain a record μ specifying the fastest moving speed traced so far. Upon receiving this record, the last Q-node is able to measure the maximum shift of sensor nodes by $(t_s - t_e)\mu$, where t_s and t_e denote timestamps for the start and end of the query dissemination respectively. Thus, an appropriate expansion of R can be obtained by $R' = R + g(t_e - t_s)\mu$.

5 Performance Evaluation

In this section, we explore performance of DIKNN in terms of query accuracy, query latency, and energy effi-

ciency. The simulation environment is developed based on ns-2 [1]. We study the impact of varying application specifications and network conditions, such as k, the query load, the mobility of sensor nodes, and the packet loss rate.

5.1 Settings and Performance Metrics

We simulate the LR-WPAN environment at $2.4\ GHz$ by disabling the RTS/CTS mechanism and setting the channel rate $250\ kbps$. The geo-routing protocol GPSR [17] and DIKNN are implemented above the ns-2 802.11 MAC layer. Mechanisms (i.e., rendezvous and mobility assurance) coping with network dynamics are enabled. Initially, the sensor nodes are randomly distributed in the simulated field. The mobility of sensor nodes is modeled by the *random waypoint* (RWP) model, in which each sensor node selects an arbitrary destination and moves to the destination at a random speed ranging from 0 to $\mu_{\max}$. Upon arrival, the node selects a new destination and walks again. In our configuration, the mobility of sensor nodes is controlled by varying the maximum moving speed $\mu_{\max}$. By default, $\mu_{\max} = 10\ m/s$. There are 200 sensor nodes in the simulation field, and each with radio range $20\ m$ [3, 13]. By fixing the number of sensor nodes and varying the simulated field from 200×200 to $115 \times 115\ m^2$, the node degree (i.e., neighbor count of each sensor node) ranges from 5 to 20. The time unit for data collection is $0.018\ s$, and the query response size of each sensor node is $10\ bytes$. Every simulation run lasts for 100 seconds of simulated time. The performances are obtained by averaging the result over 20 simulation runs. The following summarizes the default parameters.

Parameters	Value	Parameters	Value
Node number	200	r	20 m
Network size	115×115 m^2	Sector number	8
Node degree	20	$\mu_{\max}$	10 m/s
Response size	10 bytes	Beacon interval	0.5 s
Channel rate	250 kbps	RTS/CTS	off
m	0.018 s	Query interval	4 s
Rendezvous	enabled	Assurance gain	0.1

To evaluate the simulation result, three performance metrics are employed:

Query Latency: The elapsed time (in second) between the time a query is issued by the sink and the time the query responses are returned.

Energy Consumption: Amount of energy (in $Joule$) consumed in a simulation run.

Query Accuracy: As discussed in Section 3.1, the *pre-accuracy* and *post-accuracy* are measured separately in our experiments.

We focus on comparison between DIKNN and the in-network executions: the naive approaches (KPT) [29, 30] and the Peer-tree approach [7]. Note the KNN boundary estimation techniques proposed in [29, 30] lead to quadratic growth of the boundary area as k increases. The query execution can easily flood the entire network. For example, when $k = 20$ and $MHD = 15$, the returned radius length $R = 20 \cdot 15 = 300$ exceeds twice the field edge, resulting that the boundary area is six times larger than the network size. For fair comparisons, we simulate KPT in which the KNNB algorithm is adopted for boundary estimation and a spanning tree is constructed for data collection after the boundary is determined. In Peer-tree, a global index structure, R-tree [10], is built to preserve the Minimum Bounding Rectangle (MBR) hierarchy as described in Section 2. To avoid a skew indexing, we partition the network into a 5×5 grid. Every cell represents an MBR within which a stationary clusterhead is pre-located and its address is known by every sensor node. Each sensor node periodically sends a notification of existence to its closest clusterhead. If a clusterhead does not hear from a child after a period of time, it deletes the node and updates the MBR record.

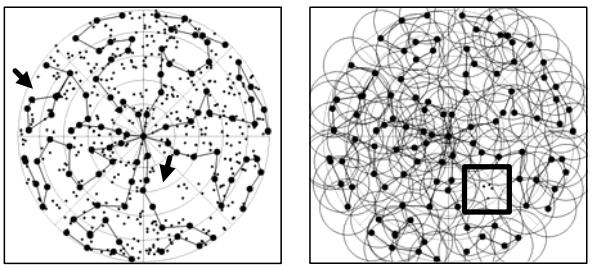

Figure 7. Visualization of DIKNN execution over the real-world sensor distributions [27].

5.2 Observations

Before studying the DIKNN performance, we discuss some observations from our simulation result that are worthy to be mentioned. We apply DIKNN to some large-scale sensor distributions obtained from [27]. The trace format of ns-2 is modified so that the query execution can be visualized. Figure 7 demonstrates a scenario for finding $k = 500$ caribous around an arbitrary query point. The concurrent itinerary traversals are illustrated in Figure 7(a), where successive hops between Q-nodes are connected with lines. As pointed out by the arrows, we can see that the itinerary void appears occasionally. When a void is encountered, the itinerary traversal switches to the perimeter forwarding mode, which bypasses the vacancy by walking into the nearby segments or sectors. During the perimeter forwarding, some sensor nodes, as surrounded by the rectangle in Figure 7(b), may not be informed of the query message be-

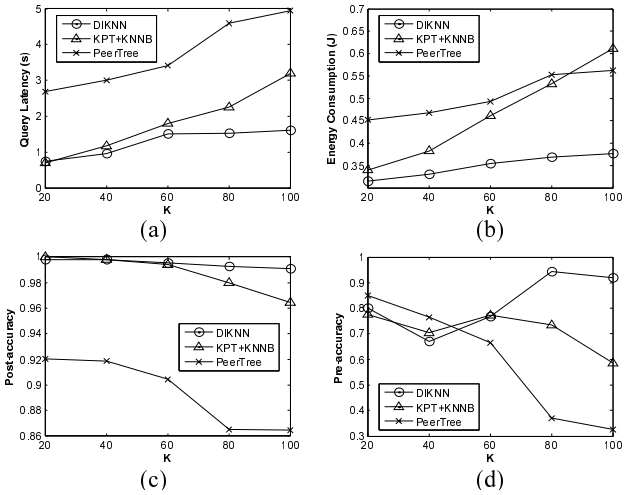

Figure 8. Scalability of DIKNN.

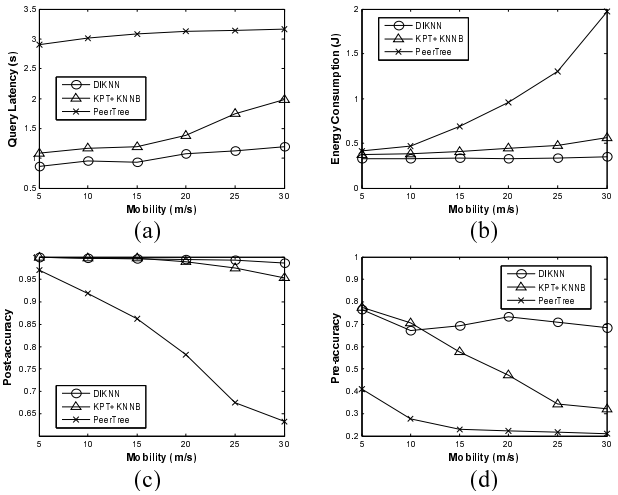

Figure 9. Impact of mobility.

cause they are isolated within a sector. Such a phenomenon can, empirically, cause 0.2% to 1% degradation of both the post- and pre-query accuracy. Fortunately, from mobility of sensor nodes, this effect can be alleviated by issuing a serious of queries within a time interval. Our demonstration verifies the applicability of DIKNN to real deployments.

5.3 Scalability

The application specified k directly affects the number of nodes involved with the query. In this section, we investigate the impact of k by varying k from 20 to 100. The parameter $\mu_{\max}$ is set to 10 m/s, and the query interval is exponentially distributed with mean 4 s. Figure 8(a) shows that both Peer-tree and KPT grow faster than DIKNN as k increases. This is due to the overhead of Peer-tree to route query messages between different levels of the R-tree hierarchy that incurs many unnecessary node visits, and the overhead of KPT to construct and maintain the spanning tree during data collection. Such overhead also leads to the higher energy consumption, as shown in Figure 8(b). KPT consumes more energy than the others when $k = 100$ due to a serious degree of collision and large retransmissions of data in the tree. Figure 8(c) shows that Peer-tree has the post-accuracy below average since the clusterheads may not obtain the most current position of each sensor node. The post-accuracy of KPT also degrades when k is large because of the long latency in data collection. The pre-accuracy of DIKNN and KPT, as depicted in Figure 8(d), varies when $k \leq 60$. This can be resulted from the error of KNN boundary estimation, since when k (correspondingly, R) is small, the error rate is relatively large. However, when $k > 60$, boundary error shrinks and DIKNN becomes precise; while the others continuously degrade due to their long latency. Among these results, DIKNN exhibits superior improvements in query latency and energy efficiency while preserving a high level of accuracy.

5.4 Impact of Network Dynamics

In this section, we study the impact of sensor movements by varying $\mu_{\max}$ from 5 to 30 m/s. We set $k = 40$ and the query interval is exponentially distributed with mean 4 s. Figure 9(a) shows that Peer-tree has high latency in all moving speeds because of the routes in hierarchy, as stated in the previous section. The latency of KPT grows because of tree maintenance overhead. Data at child levels of the tree will have to wait to be relayed to the root until the structures of the parent layers are settled. In Figure 9(b), we can see that the energy consumption of Peer-tree increases rapidly, because that there are more sensor nodes move across MBRs, which results in excessive information updates. The post- and pre-accuracy of Peer-tree shown in Figure 9(c)(d) degrade dramatically because the latest position of each sensor node can hardly be traced by the clusterheads under high mobility. A clusterhead simply drops packets (i.e., the queries) if they can not be routed to the destinations in the MBR record. The pre-accuracy of KPT also degrades due to its latency. Benefiting from the novel itinerary traversals that maintain no infrastructure, DIKNN has stable performance under various mobility conditions. Again, DIKNN offers prominent advantages over energy efficiency and query latency in mobile sensor networks while rendering a high level of accuracy.

6 Conclusion

In this paper, we proposed a cost effective solution, DIKNN, for handling the KNN queries in mobile sensor networks. DIKNN integrates query propagation with data collection along a well-designed itinerary traversal, which requires no infrastructures and is able to sustain rapid change of the network topology. A simple and effective KNNB algorithm has been proposed to estimate the KNN boundary under the trade-off between query accuracy and energy efficiency. Dynamic adjustment of the KNN boundary has also been addressed to cope with spatial irregularity and mobility of sensor nodes. From extensive simulation results, DIKNN exhibits a superior performance in terms of energy efficiency, query latency, and accuracy in various network conditions.

7 Acknowledgements

We are grateful to Professor Wang-Chien Lee for his suggestions, and Yingqi Xu for discussions over the ns-2 simulation.

References

[1] The network simulator. *http://www.isi.edu/nsnam/ns*.

[2] I. S. 802.11-1997. *IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1997.

[3] I. S. 802.15.4-2003. *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks*, 2003.

[4] M. Bawa, A. Gionis, H.G.-Molina, and R. Motwani. The price of validity in dynamic networks. In *Proc. of SIGMOD*, 2004.

[5] R. Cheng, B. Kao, S. Prabhakar, A. Kwan, and Y. Tu. Adaptive stream filters for entity-based queries with non-value tolerance. In *Proc. of VLDB*, 2005.

[6] H. Cho and C. Chung. An efficient and scalable approach to cnn queries in a road network. In *Proc. of VLDB*, 2005.

[7] M. Demirbas and H. Ferhatosmanoglu. Peer-to-peer spatial queries in sensor networks. In *Proc. of ICP2PC*, 2003.

[8] D. Ganesan, S. Ratnasamy, H. Wang, and D. Estrin. Coping with irregular spatio-temporal sampling in sensor networks. *ACM SIGCOMM Computer Communication Review*, 34(1), 2004.

[9] C. Gui and P. Mohapatra. Virtual patrol: A new power conservation design for surveillance using sensor networks. In *Proc. of IPSN*, 2005.

[10] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. of SIGMOD*, 1984.

[11] A. A. H.D. Chon, D. Agrawal. Range and knn query processing for moving objects in grid model. *Mobile Networks and Applications*, 8(4), 2003.

[12] G. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM TODS*, 24(2), 1999.

[13] I. Howitt and J. Gutierrez. Ieee 802.15.4 low rate - wireless personal area network coexistence issues. *Wireless Communications and Networking*, 3(16-20), 2003.

[14] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. of MOBICOM*, 2000.

[15] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In *Proc. of ASPLOS-X*, 2002.

[16] J. Kahn, R. Katz, and K. Pister. Next century challenges: Mobile networking for smart dust. In *Proc. of MOBICOM*, 1999.

[17] B. Karp and H. Kung. Gpsr: Greedy perimeter stateless routing for wireless networks. In *Proc. of MOBICOM*, 2000.

[18] Y. Kim, R. Govindan, B. Karp, and S. Shenker. On the pitfalls of geographic face routing. In *Proc. of DIALM*, 2005.

[19] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric ad-hoc routing: Of theory and practice. In *Proc. of PODC*, 2003.

[20] F. Kuhn, R. Wattenhofer, and A. Zollinger. Worst-case optimal and average-case efficient geometric ad-hoc routing. In *Proc. of MobiHoc*, 2003.

[21] W. Lee and B. Zheng. Dsi: A fully distributed spatial index for location-based wireless broadcast services. In *Proc. of ICDCS*, 2005.

[22] B. Liu, W. Lee, and D. Lee. Distributed caching of multidimensional data in mobile environments. In *Proc. of MDM*, 2005.

[23] M. Mokbel, X. Xiong, and W. Aref. Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In *Proc. of SIGMOD*, 2004.

[24] D. Niculescu and B. Nath. Trajectory based forwarding and its applications. In *Proc. of MOBICOM*, 2003.

[25] F. of American Scientists. Remote battlefield sensor system (rembass). *http://www.fas.org*, 2000.

[26] U. D. of Transportation. Intelligent transportation system joint program office home. *http://www.its.dot.gov*, 2006.

[27] G. M. N. Park. Map: Caribou population distribution in gros morne national park greater ecosystem. *http://www.pc.gc.ca*, 2003.

[28] S. Patil, S. Das, and A. Nasipuri. Serial data fusion using space-filling curves in wireless sensor networks. In *Proc. of SECON*, 2004.

[29] J. Winter and W. Lee. Kpt: A dynamic knn query processing algorithm for location-aware sensor networks. In *Proc. of DMSN*, 2004.

[30] J. Winter, Y. Xu, and W. Lee. Energy efficient processing of k nearest neighbor queries in location-aware sensor networks. In *Proc. of MobiQuitous*, 2005.

[31] Y. Xu, W. Lee, J. Xu, and G. Mitchell. Processing window queries in wireless sensor networks. In *Proc. of ICDE*, 2006.