Optimizing Parallel Itineraries for KNN Query Processing in Wireless Sensor Networks

Tao-Young Fu, Wen-Chih Peng
National Chiao Tung University
Hsinchu, Taiwan, ROC
csiesheep.csie91@nctu.edu.tw,
wcpeng@cs.nctu.edu.tw

Wang-Chien Lee
The Pennsylvania State University
State College, PA 16801, USA
wlee@cse.psu.edu

ABSTRACT

Spatial queries for extracting data from wireless sensor networks are important for many applications, such as environmental monitoring and military surveillance. One such query is K Nearest Neighbor (KNN) query that facilitates sampling of monitored sensor data in correspondence with a given query location. Recently, itinerary-based KNN query processing techniques, that propagate queries and collect data along a pre-determined itinerary, have been developed concurrently [12][14]. These research works demonstrate that itinerary-based KNN query processing algorithms are able to achieve better energy efficiency than other existing algorithms. However, how to derive itineraries based on different performance requirements remains a challenging problem. In this paper, we propose a new itinerary-based KNN query processing technique, called *PCIKNN*, that derives different itineraries aiming at optimizing two performance criteria, *response latency* and *energy consumption*. The performance of PCIKNN is analyzed mathematically and evaluated through extensive experiments. Experimental results show that PCIKNN has better performance and scalability than the state-of-the-art.

Categories and Subject Descriptors: H.3.4 [Systems and Software]: Distributed Systems
General Terms: Design, Experimentation and Performance
Keywords: K nearest neighbor query, wireless sensor networks.

1. INTRODUCTION

We have witnessed a rapid technological advance in wireless sensor networks (WSNs) in recent years. WSNs, which consist of a large number of sensor nodes capable of sensing, computing, and communications, can be used in a variety of applications such as border detection, environmental monitoring, smart home, and security surveillance. A lot of times, WSNs are deployed over a wide geographical area to facilitate long-term monitoring and data collection tasks. Thus, spatial queries that aim at extracting sensing data from sensor nodes located in certain areas of interests are essential to many WSN applications [6]. In this paper, we focus on the processing of *K nearest neighbors* (KNN) query, a classical spatial query with importance in many application domains in WSNs. A KNN query facilitates data sampling of the sensors located in a geographical proximity via specification of a query point q and a sample size K. Using a KNN query, one could obtain sensor data (e.g., environmental measurements) near a query location of interests. Through years of research effort, efficient KNN query processing algorithms have been developed for centralized spatial databases[7][9][10]. However, due to the stringent resource constraints at sensor nodes (e.g., limited energy, computational, storage and communication capacities), traditional KNN processing techniques are infeasible for wireless sensor networks because collecting sensing data from large-scale sensor networks into centralized databases incurs high energy consumption and long latency.

To cope with the above-mentioned issues, in-network processing techniques for KNN queries have been developed [11][12][14]. In these WSN systems, a KNN query is injected to the network (usually via an *access point*) and propagated to the sensor nodes qualified by specified predicates. As a result, sensor data from these nodes are collected and returned to the access point. Existing in-network KNN query processing techniques can be categorized into two types: a) *infrastructure-based* and b) *infrastructure-free*. The former relies on a network infrastructure (e.g., based on R-trees [1] or spanning trees in WSN) for query propagation and processing [11]. Maintenance of those network infrastructures is a major issue. In the dynamic environments of wireless sensor networks, maintaining infrastructures incurs excessive communications and thus energy overhead among sensor nodes. On the other hand, the latter does not rely on any pre-established network infrastructure to process queries. Recently, two infrastructure-free KNN query processing techniques have been concurrently developed [12][14]. By propagating queries and collecting data along well-designed itineraries, these techniques avoid the overhead of maintaining a network infrastructure.

Obviously, the performance (such as the response latency and energy consumption) of these KNN query processing techniques is dependent on the itineraries. With a long itinerary, long processing latency and high energy consumption are expected due to the long journey query and data may travel. Thus, *itinerary planning* is a vitally important design issue for itinerary-based KNN query processing. Moreover, with different performance requirements (e.g., en-

CIKM '07, November 6–8, 2007, Lisboa, Portugal.

ergy efficiency and response latency) from applications, it's important to design itineraries that meet specific application requirements. In this paper, we propose a new itinerary-based KNN query processing technique that derives different itineraries aiming at optimizing two performance criteria, *response latency* and *energy consumption*.

Specifically, the proposed new technique is based on *optimized parallel concentric-circle itineraries* (thus named as *PCIKNN*). PCIKNN is designed to allow a KNN query propagated in multiple concurrent threads. Clearly, with a larger number of concurrent KNN threads propagated, both the response latency and the energy consumption are significantly reduced. Furthermore, analytical models for the latency and the energy consumption of PCIKNN are derived. By optimizing the latency latency and the energy consumptions, PCIKNN derive itineraries with two modes (i.e., *min_latency* mode and *min_energy* mode), specifically tailored to minimize response latency or energy consumption, respectively. Additionally, an important issue for itinerary-based KNN query processing is to estimate a search boundary covered by derived parallel itineraries. A technique has been developed to derive accurate boundary estimation in order to improve performance of our KNN processing. In our approach, we derive multiple itineraries to facilitate parallel processing of the query. The performance of PCIKNN is analyzed mathematically and evaluated through extensive experimentation based on simulation. Experimental results show that PCIKNN has better performance and scalability than the state-of-the-art.

The contributions of this study are summarized as follows:

- An efficient itinerary design approach for in-network itinerary-based KNN query processing has been developed. The derived itineraries are tailored to optimize on response latency and energy consumption.
- A KNN boundary estimation technique is developed to improve accuracy of KNN query in wireless sensor networks.
- Analytical models for the performance of our proposal are developed.
- An extensive performance evaluation is conducted that shows the superiority of our proposal against other existing techniques.

The rest of the paper is organized as follows. Preliminaries, including the problem definition and basic ideas of existing itinerary-based KNN query processing is introduced. The design of our PCIKNN technique and analytic models for two optimization modes are described in Section 3. A mechanism for KNN boundary estimation is presented in Section 4. The performance of PCIKNN and other existing techniques is evaluated in Section 5. Finally, Section 6 concludes this paper.

2. PRELIMINARIES

In this section, the basic assumptions made in this research and the definition of KNN query are specified. Then, the general idea of itinerary-based KNN query processing is introduced. Finally, the problem addressed in this paper is presented.

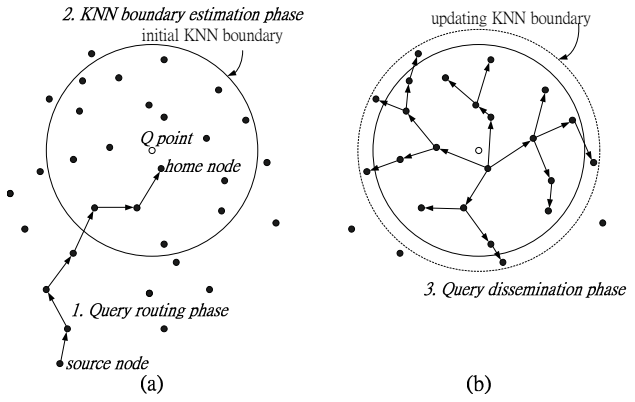

Figure 1: Overview of itinerary-based KNN query processing.

2.1 Overview of Itinerary-based KNN Query Processing

We assume that sensor nodes in wireless sensor networks are randomly distributed in a two-dimensional space. Each sensor is location-aware via GPS or other localization techniques. By periodically inter-exchanging information among sensor nodes nearby, a sensor node is able to maintain its own neighboring information. As mentioned before, KNN query processing provides a way to sample the data from sensor nodes located in the proximity of a given query location. We now define the KNN query in wireless sensor networks as follows:

Definition: (**K N**earest **N**eighbor query). Given a set of sensor nodes M and a geographical location (denoted as a query point q), find a subset M' of k nodes ($M' \subseteq M$, $|M'| = K$) such that $\forall n_1 \in M'$ and $\forall n_2 \in M - M'$, $dist(n_1, q) \leq dist(n_2, q)$, where $dist$ represents the Euclidean distance function.

KNN queries can be issued at any sensor node (called *source node*), which is the starting node for in-network query processing. The source node is also responsible for reporting the query result.

As mentioned before, itinerary-based KNN query processing techniques are infrastructure-free, thereby saving a considerable amount of maintenance overhead. Without loss of generality, an itinerary-based KNN query processing algorithm typically consists of three phases: i) routing phase; ii) KNN boundary estimation phase; and iii) query dissemination phase. Figure 1 shows an overview of the itinerary-based KNN query processing. Details of the three query processing phases are described below:

Routing phase: Figure 1(a) illustrates the routing phase. Explicitly, when a KNN query Q is issued at a source node, the query Q that specifies the query point q and the sample size K is routed to the sensor node nearest to the query point q (referred the *home node*) by using a geo-routing protocol such as GPSR [2][3]. In the routing phase, partial network information, such as the number of nodes and the area covered by communication ranges of relay messages, is collected while the query is enrouting towards the home node.

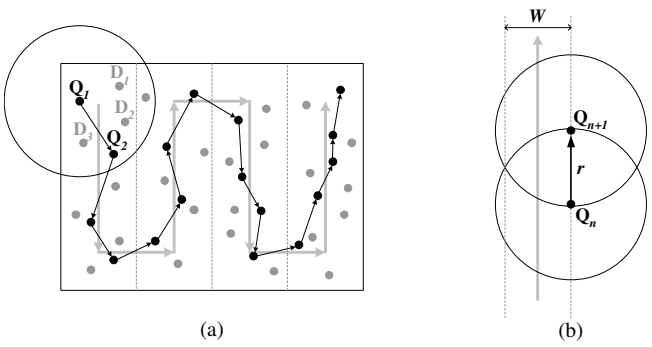

Figure 2: Itinerary-based query propagation.

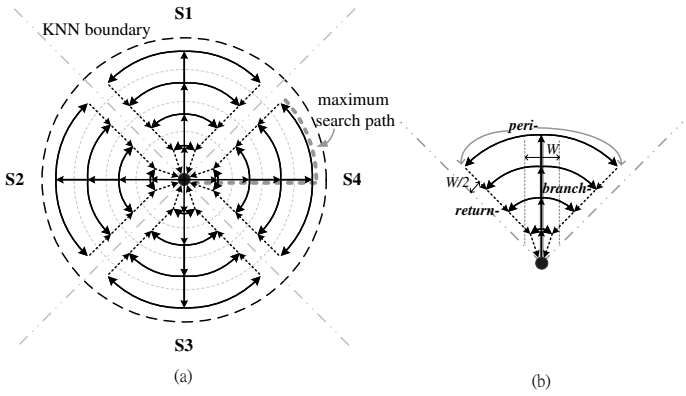

Figure 3: Parallel concentric itineraries in PCIKNN.

KNN boundary estimation phase: Upon arrival of a query at its home node, the partial network information gathered in the routing phase is used to estimate the initial KNN boundary, which is likely to contain K sensor nodes. Then the KNN query is propagated within the KNN boundary. The solid circle line in Figure 1(a) is the estimated KNN boundary. Note that the estimated KNN boundary could be dynamically updated (which is our approach) while the query is propagated within the KNN boundary.

Query dissemination phase: It can be seen in Figure 1(b) that, after estimating the initial KNN boundary, the home node propagates the query to each node within the KNN boundary. A KNN query is propagated along well-designed itineraries, while partial query results are aggregated at the same time. After reaching the end of itineraries, aggregated query results are sent back to the source node. This phase is also named as query propagation and data collection phase in this paper.

Query propagation is a critical issue for itinerary-based KNN query processing. Therefore, in the next section, we detail the itinerary-based query propagation and data collection.

2.2 Itinerary-Based Query Propagation

The detailed steps of query propagation are illustrated in Figure 2(a), where the gray line is a well-designed itinerary. As shown in the figure, sensors are categorized into *Q-node* (marked as black nodes) and *D-node* (marked as gray nodes). Upon receiving a query, a Q-node broadcasts a probe message including the query Q and the itinerary information to its neighbors. Each neighbor node (i.e., D-node) receives the probe message and then sends its sensed data to the Q-node. After collecting data from D-nodes nearby, the Q-node finds the next Q-node along the itinerary and forwards the current query result to the next Q-node. The next Q-node is determined by exploring the *maximum progress heuristic* in that the next Q-node with the farthest distance to the current Q-node along the proceeding direction of the itinerary is selected. The width of the itineraries w is set to $\frac{\sqrt{3}}{2}r$, where r is the transmission radius of a sensor node, for full coverage shown in Figure 2(b). Readers are referred to [5] [13] for detailed implementation of itinerary-based query propagation. Finally, the last Q-node forwards the aggregated query result to the source node where the KNN query is issued.

2.3 Problem of Itinerary Planning

Some itinerary structures such as spiral itinerary [14] and multiple itineraries [12] are proposed. As mentioned before, itinerary structures have a great impact on the performance of itinerary-based KNN query processing techniques. Long itineraries may incur long latency and heavy energy consumption, because the query propagation and data collection are performed along the itineraries. A query may have a long journey and have to carry more collected data on long itineraries, thereby increasing both the latency and the energy consumption. In addition to the length and number of itineraries, the number of concurrent KNN query threads propagated also have an impact on performance of itinerary-based KNN query processing techniques. Clearly, with a larger number of concurrent threads propagated along itineraries, the latency of KNN query processing may be improved. However, prior works explore multiple itineraries by fixing the number of concurrent KNN query threads to the number of itineraries. Thus, in this paper, we aim at designing itineraries that allow more concurrent KNN query threads to be propagated.

3. ITINERARY PLANNING IN PCIKNN

In this section, we first describe the design of parallel concentric-circle itineraries and give a brief comparison of concurrent threads of PCIKNN and existing works. Then, corresponding to the targeted performance criteria, i.e., response time and energy consumption, we analytically derive the number of parallel itineraries to be employed. Finally, we verify our analytical results with simulation results.

3.1 Parallel Concentric-Circle Itineraries

As pointed before, the number and structure of itineraries have an impact on performance of itinerary-based KNN query processing algorithms. Intuitively, a KNN query executed concurrently through a large number of itineraries will incur small latency and energy consumptions. However, in reality, it's not feasible to use an excessive number of itineraries due to packet collisions. Here, we explore the design issue of parallel concentric-circle itineraries by assuming a boundary that contains K nearest sensor nodes to the query point is given. We will address the issue of estimating this KNN boundary later in Section 4.

Given a query point q and an estimated KNN boundary, the area within the boundary can be divided into multiple concentric-circle itineraries. Let C_i denote the ith circle with a radius $w \times i$, where w is the *itinerary width*, the distance between itineraries. Similar to [12][14], w can be set as $\frac{\sqrt{3}}{2}r$, where r is the transmission range of a sensor node. In order to propagate KNN query along concentric-circle itineraries, we partition the KNN boundary into multiple *sectors*. Figure 3(a) shows an example of concentric-circle itineraries, where the number of sectors is 4. For each sector, we have three types of itinerary segments: 1) a *branch-segment*, 2) a set of *peri-segments*, and 3) *return-segments*. As shown in Figure 3(b), a branch segment is a straight line in each sector, peri-segments are portions of concentric-circles and the return-segments are the boundary lines among sectors. With these segments of itineraries, KNN query are concurrently executed at these segments of itineraries. It is worth mentioning that the number of concurrent KNN query propagated are maximized in our PCIKNN.

In light of itinerary segments derived above, a KNN query is first propagated along with branch-segments in each sector. Along the branch-segment, a Q-node broadcasts a probe message and aggregates the partial results from D-nodes within the region width of w. Then, for each sector, when the KNN query reaches one of concentric-circles, two KNN query threads are forked to propagate along the two peri-segments, while the original KNN query continues to move along the branch-segment. This process repeats at outer concentric-circles, which increases the number of the concurrent threads for executing the KNN query. To propagate a KNN query in two peri-segments, the Q-node in the branch segment first finds two Q-nodes in peri-segments and evenly divide the partial query result collected to these two Q-nodes. Then, these two Q-nodes will start performing KNN query dissemination with peri-segments. When KNN queries propagating along peri-segments arrive the boundary lines of their sectors, KNN queries with data collected are returned back to the home node through return-segments. Once the home node receives more KNN query results, it is able to decide whether to continue KNN query propagation or not. This leads to more precise KNN query results in PCIKNN. It can be seen by exploring parallel concentric-circles, the number of concurrent KNN query propagated is maximized. Furthermore, due to the high parallelism of PCIKNN, PCIKNN achieves high performance in terms of response time and energy consumption.

3.2 Concurrent Query Threads Comparison

In the following, we will use an example to illustrate the latency and energy performances of IKNN, DIKNN and PCIKNN in terms of the number of concurrent KNN query propagated. Assume that the KNN boundary is known and all these algorithms are performed within this boundary. In our illustrative example, the radius R of KNN boundary is set to $4w$ and there are 4 concentric-circles. First, we analyze the latency performances which is directly affected by the number of concurrent KNN query propagated. Figure 4(b) shows the parallel IKNN algorithm proposed in [14], which has two itineraries. Hence, the number of concurrent KNN query propagated is 2. For DIKNN and PCIKNN, the number of sectors affects the performances directly. Assume that the number of sectors is set to 4, which is bigger than the number of itineraries in parallel IKNN. Thus, in DIKNN, the concurrent KNN query propagated is 4 (as shown in Figure 4(a)). In PCIKNN, each itinerary in the sector has a branch-segment and two peri-segments for each concentric circle for query propagation. Therefore, the maximal number of concurrent KNN query propagated is 44 (branch-segment×1, peri-segments×8 and return-segment×2 in each sector) for 4 itineraries shown in Figure 3. Comparing these three algorithms, PCIKNN has the largest number of concurrent KNN query propagated. As a result, it is expected to have better latency performance than others.

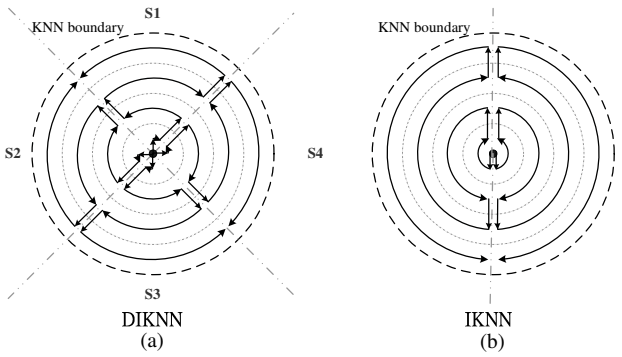

Figure 4: Itineraries of DIKNN and IKNN.

Second, for the energy performance, we evaluate the average length of a KNN query propagated to estimate the energy consumption. When an itinerary is longer, the energy consumed for carrying data increases faster. Thus, it has lower total energy consumption and better energy performance if the average length of a KNN query propagated is shorter. All algorithms have the total length of concentric-circle itineraries. The total length of concentric-circle itineraries is formulated as follows:

$$\sum_{i=1}^{\frac{R}{w}} 2 \times \pi \times (i - \tfrac{1}{2}) \times w$$

In our example, since R is set to $4w$, the total length of concentric-circles is thus determined as $16\pi w$ (i.e., $2\pi \times w \times (0.5 + 1.5 + 2.5 + 3.5) = 16\pi w$). In parallel IKNN, the total itinerary length is the sum of concentric-circles and additional branch-segments length which is $3.5w$ in each sector. Consequently, the average length of a concurrent KNN query propagated in IKNN is $\frac{16\pi w + 2 \times 3.5w}{2} = 28.63w$. In DIKNN, the total length of itineraries is the sum of the total length of concentric-circles and additional branch segments (i.e., 4). Since there are four KNN query propagated, the average itinerary of each KNN query is calculated as $\frac{16\pi w + 4 \times 3.5w}{4} = 16.06w$. Furthermore, with the number of sectors to be 4, the total length of itineraries in PCIKNN is estimated as $16\pi w + 12 \times 3.5w$, which consists of concentric-lengths, branch-segments and return-segment. Explicitly, a sector has a branch-segment, two return-segments with their length as $3.5w$. However, though the length of itineraries is larger, the total number of concurrent KNN query propagated is also larger. In Figure 3(a), there are 44 KNN query propagated. As a result, the average itinerary length for a KNN query propagated is $\frac{16\pi w + 12 \times 3.5w}{44} = 2.097w$. Among the three algorithms, PCIKNN has the smallest average itinerary length for a KNN query. Thus, PCIKNN should have the best energy performance. The results are illustrated in Table 1.

	Number of query	Average query length
IKNN	2	$\frac{16\pi w+3.5w}{2} = 26.87w$
DIKNN	4	$\frac{16\pi w+4\times3.5w}{4} = 16.06w$
PCIKNN	44	$\frac{16\pi w+12\times3.5w}{44} = 2.096w$

Table 1: Comparison of concurrent query threads and query lengths.

Parameter	Description
R	radius of KNN boundary (m)
d	network density ($nodes/m^2$)
r	transmission range of a node (m)
w	width of itineraries
E_r	the expected distance between hops (m)
$Delay$	time delay for processing a message (s)
$Dsize$	D-node message size ($bits$)
$Hsize$	message header size ($bits$)
$Bits$	energy to transmit one data bit per hop

Table 2: Parameters used in our analytical model.

From the above analysis, PCIKNN is able to allow as many concurrent KNN query threads as possible in itineraries derived. Thus, with a good parallelization, PCIKNN has smaller latency time and energy consumption than existing works.

3.3 Optimal Number of Sectors for PCIKNN

PCIKNN explores parallel concentric-circles itineraries to achieve better parallelism. Thus, to determine an appropriate number of sectors (denoted as **S**) is a critical issue. When the number of sectors is larger, the length of total itineraries will increase because the numbers of branch-segments and return-segments are also increased. However, when a smaller number of sectors is used, the length of peri-segments in each sector increases, thereby incurring more energy consumption for propagating KNN queries and carrying more partial KNN results. There is an obvious bound of the number of sectors that is when the length of peri-segments is zero. In this section, we derive analytical models to determine the appropriate number of sectors in accordance with the two optimization goals considered, i.e., minimum latency (referred to as *min_latency*) and the minimum energy (referred to as *min_energy*).

3.3.1 Notations and Assumptions

Given a KNN boundary with radius R and network density d, we intend to derive an appropriate number of sectors to meet the optimization objectives. Note that the network density is able to estimate while routing KNN query and will be described later. Assumptions made in our analysis are discussed as follows. We assume that sensor nodes are uniformly distributed and there is no void area in the monitored region. Message transmissions are reliable, i.e., there is no message lost. Query propagation along the branch- and return- segments are one hop for each concentric circle and each Q-node is ideally located in the itineraries. Parameters used in our analysis are summarized in Table 2

3.3.2 Minimum Latency for PCIKNN

In PCIKNN, when the number of sectors is increased, peri-segment is expected to shorten, thus reducing the latency. For each sector, once a KNN query arrives boundaries among sectors, the partial results should be transmitted to the home nodes. With a larger number of sectors, network jams are likely to happen around the home node, degrading the latency of PCIKNN. Therefore, the latency time of PCIKNN mainly depends on two values: $latency_{peri}$ and $latency_{home}$. Explicitly, $latency_{peri}$ is the latency of propagating KNN query in peri-segments at the farthest concentric-circle from the home node because the latency of query propagation along peri-segments at the inner concentric-circle is smaller. On the other hand, $latency_{home}$ is the time spent for processing partial results at the home node. Therefore, the latency time of PCIKNN is formulated as

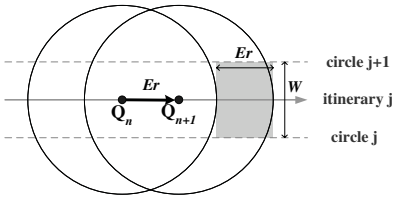

Figure 5: Information statistics in routing path.

$$latency = latency_{peri} + latency_{home}$$

To calculate $latency_{peri}$, we should take into account message delays for sending probe messages and receiving D-nodes messages at each Q-node. Thus, we could have $latency_{peri}$ as follows:

$$latency_{peri} = E^{peri}_{hop} \times (1 + E^{peri}_{D_{num}}) \times Delay$$

, where E^{peri}_{hop} is the expected number of Q-nodes in the peri-segment at the farthest concentric-circle and $E^{peri}_{D_{num}}$ is the expected number of D-nodes of a Q-nodes. The expected number of D-nodes is estimated as the number of D-nodes in the gray area in Figure 5. Consequently, we have

$$E^{peri}_{hop} = \frac{2 \times \pi \times R \times w}{2 \times S \times Er}$$
$$E^{peri}_{D_{num}} = Er \times w \times d$$

, where Er is the expected length of each hop of Q-nodes. According to [12], Er is formulated as $Er = r^2\sqrt{d}/(1+r\sqrt{d})$.

As for the latency time spent on collecting partial results at the home node, we assume that partial results are sent to the home nodes at the same time (which is the worst-case scenario). Hence, we have the following latency:

$$latency_{home} = 2 \times S \times Delay$$

According to the above derivations, the latency time for PCIKNN is able to derived as follows:

$$latency = (\tfrac{\pi \times R}{S \times Er} \times (1 + w \times Er \times d) \times Delay) + (2 \times S \times Delay)$$

In order to obtain the optimal number of sectors to achieve the minimal latency time of PCIKNN, we could differentiate the above latency formula. Therefore, the optimal number of sectors is derived as follows:

$$S = \sqrt{\frac{(\frac{\pi \times R}{Er}) \times (1 + w \times Er \times d)}{2}}$$

3.3.3 Minimum Energy for PCIKNN

As mentioned above, a long itinerary length shall incur more energy consumption. However, the energy consumption of PCIKNN should consider the energy consumption on the query propagation and data collection along with branch-segments and return-segments. Explicitly, a small number of sectors leads to long itineraries in each sector, incurring heavy energy consumption overall in carrying data collected from D-nodes. On the contrary, a large number of sectors increases the number of branch-segments and return-segments, incurring more energy consumption on query propagation and data collection. Thus, an optimal number of sectors can be derived to minimize the energy consumption. Generally speaking, the energy consumption of PCIKNN involves two parts in each itinerary segment: 1) Data collected from D-nodes should be carried hop-by-hop along with KNN query. 2) KNN query is propagated along with Q-nodes. Without loss of generality, the energy consumption is modeled as the communication cost in terms of the number of bits transmitted among sensor nodes. Thus, the energy consumption of PCIKNN is the sum of energy consumption corresponding to branch-segment, peri-segment and return-segment of all sectors. Since there are multiple concentric-circle itineraries in PCIKNN, the energy consumption on the *type*-segment in the ith concentric-circle itineraries is expressed by $energy_{type,C_i}$. For example, the energy consumption of peri-segment itineraries in the first concentric-circle is represented as $energy_{peri,C_1}$. Consequently, we have

$$energy = \sum_{i=1}^{R/w}(S \times (energy_{branch,Ci} + energy_{peri,Ci} + energy_{return,Ci})),$$

where the number of concentric-circles is $\frac{R}{w}$.

The energy consumption is modeled as $E_{hop} \times Bits$, where E_{hop} is the expected number of hops and $Bits$ is the energy consumption to transmit one data bit per hop. Let $E^{type}_{hop,Ci}$ denote the expected number of hops in the type-segment on Ci. For example, $E^{branch}_{hop,Ci}$ is the expected number of hops in the branch-segment on Ci. Furthermore, $E^{type}_{D_{num}}$ represents the expected number of D-nodes of a Q-node in the *type*-segment. For computation simplification, we simplify the radius of Ci is $i \times w$. In the following, we will derive the energy consumption in each itinerary segment.

Note that the energy consumption in each segment consists of D-nodes data carrying energy and Q-nodes query propagation energy. For the energy consumption for the branch segment on Ci, we could have the following formula:

$$E^{branch}_{hop,Ci} \times (H_{size} + (E^{branch}_{D_{num}} \times D_{size})) \times Bits,$$

where $E^{branch}_{hop,Ci} = 1$, $E^{branch}_{D_{num}} = 0$ since Q-nodes are assumed to be connected hop-by-hop along with a branch-segment and D-nodes data collected are divided into peri-segments.

Similar, we could derive the energy consumption in the peri-segment as follows:

$$2 \times ((E^{peri}_{hop,Ci} \times H_{size}) + ((\sum_{j=1}^{E^{peri}_{hop,Ci}}(j \times E^{peri}_{D_{num}})) \times D_{size})) \times Bits,$$

where $E^{peri}_{hop,Ci} = \dfrac{2 \times \pi \times i \times w}{2 \times S \times Er}$ and $E^{peri}_{D_{num}} = Er \times w \times d$.

Since we have two return-segments in each sector, the energy consumption is modeled as follows:

$$2 \times E^{return}_{hop,Ci} \times (H_{size} + (E^{return}_{D_{num},Ci} \times D_{size})) \times Bits,$$

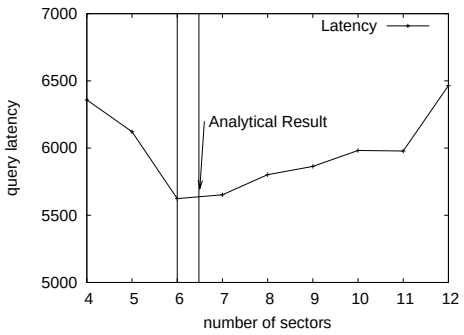

Figure 6: Minimum latency validation.

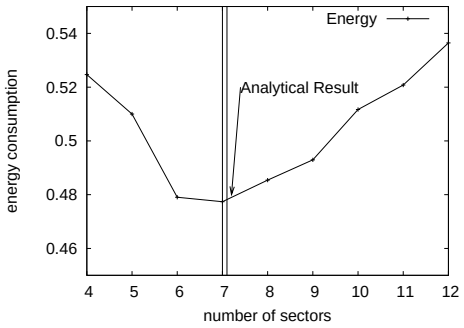

Figure 7: Minimum energy validation.

where $E^{return}_{hop,Ci} = i$, $E^{return}_{D_{num},Ci} = E^{peri}_{hop,Ci} \times E^{peri}_{D_{num}}$.

Putting the above formulas together, we could further utilize differentiation to derive the optimal number of sectors so as to minimize the energy consumption of PCIKNN. Consequently, the optimal number of sectors is derived as follows:

$$S = \sqrt{\frac{\frac{\pi^2 w^3 d}{Er} \times \frac{(2\frac{R}{w}+1)}{6} \times Dsize}{Hsize}}$$

Model Validation: The simulation environment for model validation is that there are 1000 sensor nodes randomly distributed in a 500×500 simulation field. Total 100 KNN queries are issued, where K=300. The simulation results are shown in Figure 6 and Figure 7. The minimal latency model determines $S_{latency}$ =6.4 rounded to 6 is equal to the experiment minimal latency with S=6. The minimal energy model derives S_{energy} =7.1 rounded to 7 also matched the experiment result with S=7. These comparisons show that our analysis is pretty accurate. From the formulas derived, we could easily determine the number of sectors with its quality very close to the optimized objectives.

4. KNN BOUNDARY ESTIMATION

Obviously, the precision of the KNN boundary estimation has a direct impact on the performance of itinerary-based KNN query processing. In this section, we develop a mechanism to estimate KNN boundary. Furthermore, to increase the accuracy of KNN query, we dynamically adjust KNN boundary in PCIKNN.

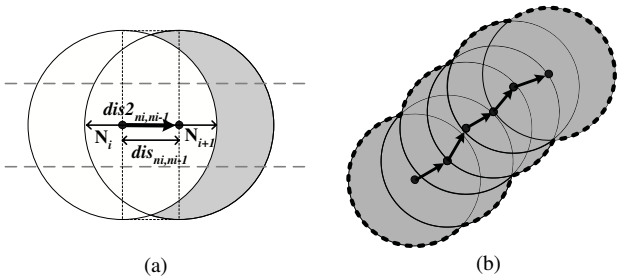

Figure 8: Estimating coverage areas of routing.

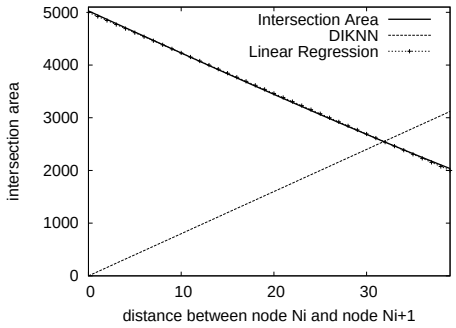

Figure 9: Intersection covered area with various node distances.

4.1 Design of KNN Boundary Estimation

An over-estimated KNN boundary will lead to excessive energy consumption and long latency, whereas an under-estimated KNN boundary may reduce the accuracy of query results. Thus, boundary estimation is very critical to the success of itinerary-based KNN query processing. Here we assume that sensor nodes do not have a priori knowledge about the density and distribution of nodes in the network. To address this issue, DIKNN [12] first collect (partial) network information in the query routing phase to derive the network density, which in turn is used to estimate a boundary that are likely to contain K sensor nodes. Clearly, how to precisely determine the network density from the partial information gathered in the routing phase is important. In the following, we will describe our approach to derive the network density in the routing phase.

By using a geo-routing protocol, e.g., GPSR, KNN query is greedily forwarded from the source node to the home node. In the routing phase, network information, such as the number of nodes and the coverage area shown the dotted area in Figure 8(b) along the routing path, is obtained. This information (i.e., coverage area of routing and the number of nodes encountered) are gathered and sent along with KNN query. Let A_i denote the area covered by relay messages up to the ith hop and Num represents the total number of nodes within the coverage area of the routing path. They are collected and updated as a KNN query moves forward hop-by-hop. Once the query reaches its home node, the collected information is used to estimate the network density.

Next, we demonstrate how to update these two values in the routing phase. A message relay from node N_i to node N_{i+1} is shown in Figure 8(a), where the gray area is the newly explored area, denoted as EA_i. The number of sensor nodes in EA_i is denoted by inc_{i+1}. By adding inc_{i+1} to Num, one could have the most updated number of nodes encountered so far. The value of EA_i is formulated as $EA_i = \pi r^2 - H(2r - dist(N_i, N_{i+1}))$, where r is the transmission radius of a sensor, H(.) is a linear function and $dist(N_i, N_{i+1})$ is the Euclidean distance between N_i and N_{i+1}. By extensive experiments, we observed that the intersection area between two sensor nodes is almost negative correlated with $dist(N_i, N_{i+1})$ shown in Figure 9. Thus, one cannot simply derive the intersection area of two sensor nodes as $2r \times dist(N_i, N_{i+1})$. As can be seen in Figure 9, the result of DIKNN is simply estimated the intersection area by $2r \times dist(N_i, N_{i+1})$, where r is transmission radius and is set to 40m, the same as in [12][14]. It can be seen in Figure 9, the estimation area by [12] does not have the same trend with the real intersection area. Thus, in this paper, to precisely estimate the intersection area between two sensor nodes, we explore linear regression techniques and H(.) is thus formulated as:

$$H(dist(N_i, N_{i+1})) = c_1 + c_2 \times dist(N_i, N_{i+1}),$$

where c_1 and c_2 are coefficients, which are determined by [4].

In fact, H function, to be empirically determined in our simulation, is used to estimate the intersection area of N_i and N_{i+1}. Hence, the total area covered by relay messages up to ith hop is as follows:

$$A_i = EA_i + A_{i-1}, \text{ for } i > 1 \text{ and } A_1 = \pi r^2.$$

When a KNN query reaches the home node, the home node computes the network density D as $D = \frac{Num}{A}$, where A is the total area covered by relay messages from the source node. As a result, the KNN boundary is estimated as follows:

$$\pi R^2 \times D = K$$
$$R = \sqrt{\frac{K}{\pi D}}.$$

Although DIKNN also explores the network density to estimate the KNN boundary region, the network density derived in DIKNN is not accurate due to poor estimation of total coverage area. Specifically, the intersection area of two sensor nodes is coarsely determined. Furthermore, an additional list is used to record local information. This list, sent along with KNN query, incurs significant more energy overhead. We compare the performance of our proposed mechanism with that of DIKNN later.

4.2 Mechanism for Spatial Irregularity

The above boundary estimation is under the assumption that sensor nodes are uniformly distributed in the monitored region. However, most of real sensor networks are spatial irregularity. To deal with the spatial irregularity, we propose one mechanism in which KNN boundary is further adjusted according to the local network information within a sector. Furthermore, due to the nature of PCIKNN, the home node is able to decide whether KNN should be propagated or not. In other words, if the number of nodes in KNN query results is not larger than K, the home node will inform Q-nodes at the farthest concentric-circle to continue KNN search for one more concentric-circle. The above procedure will be

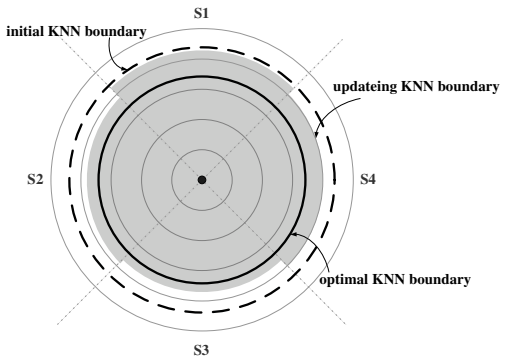

Figure 10: Updating KNN boundary dynamically.

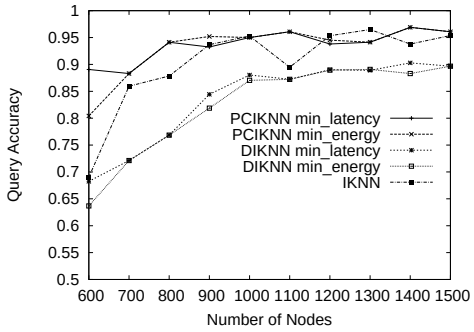

Figure 11: Query accuracy under various density.

repeated until the number of sensor nodes in KNN boundary is larger or equal to K. This guarantees the query accuracy of KNN query.

After estimating the KNN boundary with radius R , KNN query is then propagated to branch-segments. The network information obtained in the routing phase (i.e., the total coverage area A and the number of nodes Num) is sent with KNN query processing. When KNN query is propagated along with the branch-segment, local network information within the segment is accumulated via the above operation in the routing phase. A_B (respectively, Num_B) is the coverage area (respectively, number of nodes) along the branch-segment. Consequently, we could derive the network density by exploring local network information. Hence, the network density is updated as follows:

$$D_{update} = (Num+Num_B)/(A+A_B)$$
$$R_{update} = \sqrt{\frac{K}{D_{update}}}$$

By exploiting local network information in each sector, PCIKNN is able to dynamically adapt KNN boundary. For example, the gray area in Figure 10 is adapted to node densities in sectors. Even though this adapted boundary still cannot guarantee the accuracy of KNN query result, PCIKNN performs further processing the query results at the home node and adjusts KNN boundary as needed. Basically, the home node checks whether the KNN query is satisfied by the collected result or not. If the number of nodes within the KNN boundary is not larger than K, Q-nodes at the farthest concentric-circle will further extend one concentric-circle itinerary to search more number of nodes.

5. PERFORMANCE EVALUATION

We develop a simulation to evaluate the performance of PCIKNN and other closely related works, i.e., IKNN and DIKNN. The simulation model and parameter settings are presented in Section 5.1. The experimental results are reported in Section 5.2.

5.1 Simulation Model

Our simulation is implemented in CSIM[8]. There are 1000 sensor nodes randomly distributed in a $500\times500m^2$ region. The transmission radius of a node is $40m$. For each sensor node, the average number of neighboring nodes is 20. The message delay for transmitting or receiving messages is $30ms$. To simplify our experiments, we assume sensor nodes to be static. For each query, the location of a query point q is randomly selected. The value of K for each KNN query is drawn randomly. A KNN query is answered when the query result is returned to the source node. For each round of experiment, 5 queries are issued from randomly selected source nodes. Each experimental result is obtained by average of twenty rounds of experiments. Three itinerary-based KNN algorithms are implemented. Algorithm IKNN [14] exploring one itinerary is adopted while Algorithm DIKNN [12] with minimum latency is utilized. For fair comparison, we derive the result of DIKNN with the minimum energy by selecting the minimum energy from all possible numbers of sectors in DIKNN. We compare three algorithms in terms of energy consumption, query latency and query accuracy under various environment factors such as the network density, the number of sample size (i,e., K for KNN queries). These performance metrics are summarized as follows:

Energy Consumption (Joules): The total amount of energy consumed for processing a KNN query in a simulation run.

Query Latency (ms): The elapsed time between the time a query is issued and the time the query result is returned to the source node.

Query Accuracy(%): The percentage ratio of the number of sensor nodes that are exactly the K nearest sensor nodes to query point q over the number of sensor nodes in KNN query results collected.

5.2 Experimental Results

In this section, we first investigate the impact of network density on the three examined algorithms. Then, we study the scalability of these three algorithms to the value of K. Finally, we examine the accuracy of boundary estimation.

5.2.1 The Impact of Network Density

First, we investigate the impact of network density to the performance of the three examined algorithms. Here, the network density is measured as the number of sensor nodes deployed in a fixed monitored region (i.e., $500\times500m^2$). We varied the number of sensor nodes from 600 to 1500. As a result, the average number of neighbors for each node is varied from 10 to 30. Figure 11 shows that all three algorithms have better query accuracy when the network density is increased. However, when the network is sparse (i.e., the number of nodes is smaller than 800 nodes), PCIKNN

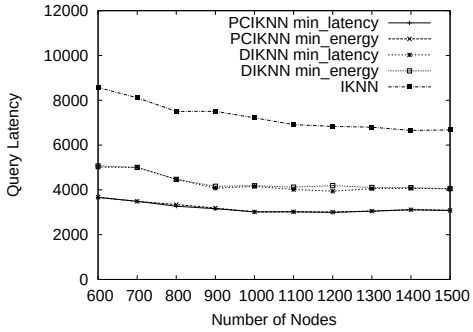

Figure 12: Query latency under various density.

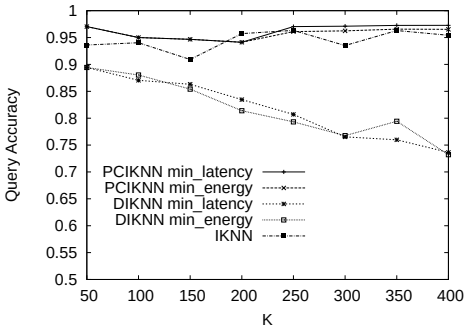

Figure 14: Query accuracy under various K.

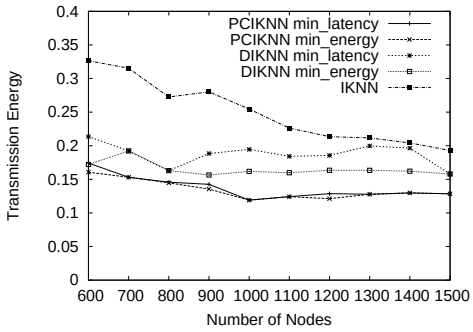

Figure 13: Energy consumption under various Density.

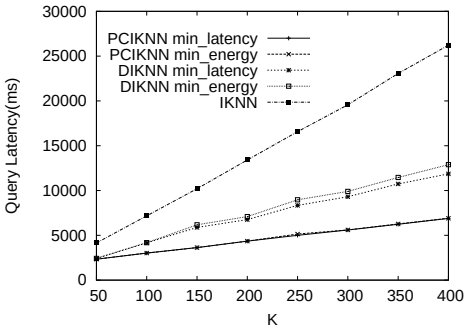

Figure 15: Query latency under various K.

outperforms IKNN and DIKNN. This is because a KNN query propagating along itineraries in IKNN and DIKNN may easily get dropped. Due to the high parallelism in itineraries, PCIKNN is robust. Only a few D-nodes are lost if a KNN query is dropped. Furthermore, DIKNN utilizes the estimated KNN boundary to decide whether KNN query should be stopped or not (i.e., without dynamic adaptation like PCIKNN). Consequently, the query accuracy of DIKNN is significantly reduced. This phenomenon is worsen when the network density is low. In a fairly dense network, both IKNN and PCIKNN have better query accuracy. As can be seen in Figure 12, the latency of PCIKNN is the lowest among three algorithms, showing the strength of concurrent KNN query propagation. The latency is lower when the number of nodes is increased for each algorithm. In the dense network, the latency of three algorithms tends to decrease. Clearly, in a dense network, KNN boundary is small, leading to a short latency. Figure 13 shows the energy consumption of three algorithms. PCIKNN has the lowest energy consumption, showing the merits of itinerary designed in PCIKNN.

5.2.2 Scalability of IKNN, DIKNN and PCIKNN

Next, we investigate the impact of sample size K to scalability of the three algorithms. Clearly, K has a direct impact on the number of nodes involved in query processing. In this experiment, the value of K is varied from 50 to 400. The query accuracy of IKNN, DIKNN and PCIKNN is shown in Figure 14. It can be seen in Figure 14 that the query accuracy of DIKNN is significantly reduced as K increases, because the KNN query result of each itinerary in DIKNN is directly sent back to the source without further processing even though DIKNN adjusts KNN boundary and exchanges information among itineraries. On the other hand, PCIKNN and IKNN have good query accuracy under the varied K. The latency increases as K increases since more sensor nodes are discovered. As seen in Figure 15, PCIKNN has the smallest latency, validating our analytical model for minimum latency of PCIKNN. As seen in Figure 16, energy consumption of all algorithms tends to increase as K increases because the number of sensor nodes involved is increased as well. PCIKNN has the smallest energy consumption. Furthermore, compared with PCIKNN with the minimum latency mode (i.e., $PCIKNN_{min_latency}$), PCIKNN with the minimum energy (i.e., $PCIKNN_{min_energy}$) indeed has the minimal energy consumption, showing the correctness of our optimized derivation.

5.2.3 KNN Boundary Estimation Simulation

To evaluate the proposed KNN boundary estimation technique, we set the value of K to be 300. To avoid the effect of network boundary, query points in the middle region (i.e., $100m \times 100m$) are selected. The liner regression function of PCIKNN is set to $H(dist(N_i, N_{i+1})) = (-76.9166 \times dist(N_i, N_{i+1}) + 4999.0903)$ by liner regression technique in [4]. The optimal KNN boundary is the average distance of kth distant nodes of all queries derived by the experiments. As shown in Figure 17, PCIKNN is very close to the optimal KNN boundary under various network density. How-

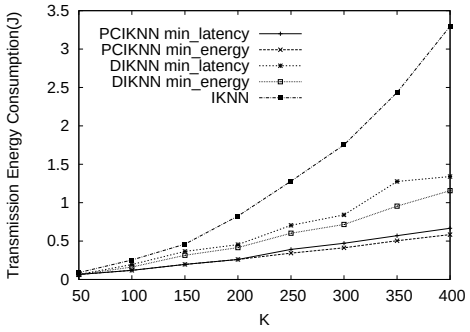

Figure 16: Energy consumption under various K.

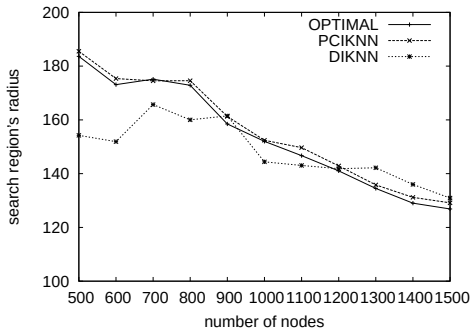

Figure 17: KNN boundary estimation of DIKNN and PCIKNN.

ever, the boundary estimated in DIKNN does not fit well with the trend of the optimal KNN boundary. Moreover, in Figure 17, when the number of nodes is smaller than 800, the KNN boundary estimated in DIKNN is smaller than the optimal value because the KNN boundary is large and the routing path from the source node to the home node is not long enough to estimate a region contained K sensor nodes. On the other hand, even though the routing path is long enough to estimate the KNN boundary, the result is not precise due to the inaccuracy of coverage areas in routing path. Figure 17 demonstrates the correctness and the accuracy of our KNN boundary estimation.

6. CONCLUSIONS

In this paper, we proposed an efficient itinerary-based KNN algorithm, PCIKNN, for KNN query processing in the sensor network. PCIKNN disseminates queries and collects data along pre-designed itineraries with high parallelism. We derived the latency and the energy consumption of PCIKNN and then by optimizing the derived formulas, we are able to determine the appropriate number of sectors for PCIKNN. Furthermore, by exploring linear regression, the KNN boundary estimated is as close as the optimal one. In addition, PCIKNN is able to dynamically adjust KNN boundary by considering local network information within sectors. Furthermore, KNN query is guaranteed to be answered since the home node will decide whether to further extend the boundary or not based on collected KNN query result. Extensive experiments have been conducted. Experimental results show that PCIKNN significantly outperforms others in terms of energy consumption, query latency and query accuracy.

Acknowledgement

Wen-Chih Peng was supported in part by the National Science Council, Project No. NSC 95-2221-E-009-061-MY3 and by Taiwan MoE ATU Program. Wang-Chien Lee was supported in part by the National Science Foundation under Grant no. IIS-0328881, IIS-0534343 and CNS-0626709.

7. REFERENCES

[1] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *ACM SIGMOD*, pages 47–57, 1984.

[2] B. Karp and T. H. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *ACM MobiCom*, pages 243–254, 2000.

[3] F. Kuhn, R. Wattenhofer, and A. Zollinger. Worst-Case Optimal and Average-Case Efficient Geometric Ad-Hoc Routing. In *ACM Mobihoc*, pages 267–278, 2003.

[4] S. J. Leon. *Linear Algebra with Applications*. Prentice Hall, 2002.

[5] D. Niculescu and B. Nath. Trajectory Based Forwarding and Its Applications. In *ACM MobiCom*, 2003.

[6] H. Qi, X. Wang, S. Iyengar, and K. Chakrabarty. High Performance Sensor Integration in Distributed Sensor Networks Using Mobile Agents. *International Journal of High Performance Computer Applications*, 16(3):325–335, 2002.

[7] N. Roussopoulos, S. Keeley, and F. Vicent. Nearest Neighbor Queries *. In *ACM SIGMOD*, pages 71–79, 1995.

[8] H. Schwetman. *CSIM user's guide (version 18)*. Mesquite Software, Inc., http://www.mesquite.com.

[9] T. Seidl and H. Kriegel. Optimal Multi-Step k-Nearest Neighbor Search. In *ACM SIGMOD*, pages 154–165, 1998.

[10] Z. Song and N. Roussopoulos. K -Nearest Neighbor Search for Moving Query Point. In *International Symposium on Spatial and Temporal Databases*, pages 79–96, 2001.

[11] J. Winter, Y. Xu, and W. C. Lee. Energy Efficient Processing of K Nearest Neighbor Queries in Location-aware Sensor Networks. In *Mobiquitous*, pages 281–292, 2005.

[12] B. Wu, K. T. Chuang, C. M. Chen, and M. S. Chen. DIKNN: An Itinerary-based KNN Query Processing Algorithm for Mobile Sensor Networks. In *ICDE*, 2007.

[13] J. Xu, Y. Xu, W. C. Lee, and G. Mitchell. Processing Window Queries in Wireless Sensor Networks. In *IEEE ICDE*, 2006.

[14] Y. Xu, T. Y. Fu, W. C. Lee, and J. Winter. Itinerary-based Techniques for Processing K Nearest Neighbor Queries in Location-aware Sensor Networks. *Signal Processing*, 2007.