Parallelizing Itinerary-Based KNN Query Processing in Wireless Sensor Networks

Tao-Yang Fu, *Student Member*, *IEEE Computer Society*, Wen-Chih Peng, *Member*, *IEEE*, and Wang-Chien Lee, *Member*, *IEEE*

Abstract—Wireless sensor networks have been proposed for facilitating various monitoring applications (e.g., environmental monitoring and military surveillance) over a wide geographical region. In these applications, *spatial queries* that collect data from wireless sensor networks play an important role. One such query is the *K-Nearest Neighbor (KNN) query* that facilitates collection of sensor data samples based on a given query location and the number of samples specified (i.e., K). Recently, itinerary-based KNN query processing techniques, which propagate queries and collect data along a predetermined itinerary, have been developed. Prior studies demonstrate that itinerary-based KNN query processing algorithms are able to achieve better energy efficiency than other existing algorithms developed upon tree-based network infrastructures. However, how to derive itineraries for KNN query based on different performance requirements remains a challenging problem. In this paper, we propose a Parallel Concentric-circle Itinerary-based KNN (PCIKNN) query processing technique that derives different itineraries by optimizing either *query latency* or *energy consumption*. The performance of PCIKNN is analyzed mathematically and evaluated through extensive experiments. Experimental results show that PCIKNN outperforms the state-of-the-art techniques.

Index Terms—K-Nearest neighbor query, wireless sensor networks.

1 Introduction

Recent advances in microsensing MEMS and wireless communication technologies have set off the rapid development of wireless sensor networks (WSNs) in the past few years. A WSN, which consists of a large number of sensor nodes capable of sensing, computing, and communications, has been used for a variety of applications, including border detection, ecological monitoring, and intelligent transportation. Typically, WSNs are deployed over a wide geographical area to facilitate data collection in long-term monitoring. Due to the importance of geographical features in WSN applications, *spatial queries* that aim at extracting sensed data from sensor nodes located in certain proximity of interested areas become an essential function in WSNs [11], [15]. In this paper, we focus on efficient processing of *K-Nearest Neighbor* (KNN) query, which facilitates data sampling of the sensors located in a geographical proximity specified by a given query point q and a sample size K.[1] The KNN query is one of the most well-studied spatial queries under the context of centralized databases, which has received tremendous research effort in optimizing its processing performance [16], [18], [6], [19], [3]. However, these traditional KNN processing techniques are not feasible for wireless sensor network applications due to the limited network bandwidth and energy budget in sensor nodes. Basically, collecting a massive amount of sensed data from a large-scale sensor network periodically to a centralized database for query processing is not a good choice due to a number of issues, such as data freshness, redundant transmission, and high energy consumption.

To overcome these issues, a number of in-network processing techniques for KNN queries have been developed [4], [1], [20], [21], [24], [22], [23]. In these works, a KNN query is submitted to the network via an arbitrary sensor node (referred to as the *source node*) and propagated to the other sensor nodes qualified by specified parameters in the query. As a result, sensed data from these nodes are collected and returned to the source node. Existing in-network KNN query processing techniques can be classified into two categories: 1) *infrastructure-based* and 2) *infrastructure-free*. The former relies on a network infrastructure (e.g., based on a spanning tree [12], [13]) for query propagation and processing [4], [20], [21]. Maintenance of such a network infrastructure is a major issue, especially when the sensor nodes are mobile [2], [5]. The latter does not rely on any preestablished network infrastructure to process queries but propagates a KNN query along some well-designed itineraries to collect data. Two infrastructure-free KNN query processing techniques, which are based on itinerary structures, have been proposed [24], [22], [23].

An itinerary-based KNN query processing algorithm typically consists of three phases: 1) routing phase, 2) KNN boundary estimation phase, and 3) query dissemination and data collection phase.[2] An example of itinerary-based KNN query processing is shown in Fig. 1. Initially, a KNN query,

1. The KNN query considered in this paper focuses on collecting sensor data samples. Thus, we do not consider the data aggregation issues.

2. Phase 3 is also called *query dissemination phase* in short.

• *T.-Y. Fu and W.-C. Peng are with the Department of Computer Science, National Chiao Tung University, No. 1001, University Road, Hsinchu, Taiwan 300, ROC. E-mail: csiegoat@gmail.com, wcpeng@cs.nctu.edu.tw.*
• *W.-C. Lee is with the Department of Computer Science and Engineering, Pennsylvania State University, 360D Information Science and Technology Building, State College, PA 16801. E-mail: wlee@cse.psu.edu.*

Manuscript received 16 Sept. 2008; revised 10 Apr. 2009; accepted 31 May 2009; published online 10 June 2009.
Recommended for acceptance by B.C. Ooi.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2008-09-0486.
Digital Object Identifier no. 10.1109/TKDE.2009.146.

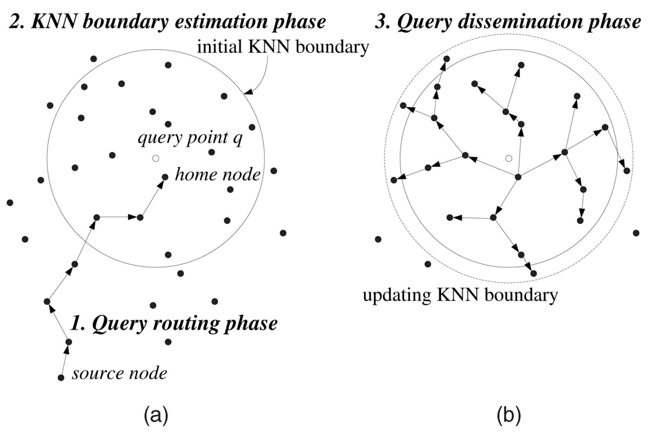


Fig. 1. An overview of itinerary-based KNN query processing.

issued at a source node, is routed to the sensor node nearest to the query point q (referred the *home node*) at the routing phase.[3] Next, in the KNN boundary estimation phase, the home node estimates an initial KNN boundary (i.e., the solid boundary line circle in Fig. 1a), which is likely to contain K nearest sensor nodes from q. Finally, in the query dissemination phase (as shown in Fig. 1b), the home node propagates the query to each node within the estimated initial KNN boundary. While the KNN query propagates along certain well-designed itineraries, query results are collected at the same time. It has been shown that, by avoiding the significant overhead of maintaining a network infrastructure, the itinerary-based KNN query processing techniques outperform the infrastructure-based KNN techniques [22], [23], [24].

Clearly, the performance (such as the *query latency* and the *energy consumption*) of itinerary-based KNN query processing techniques is dependent on the design of itineraries. With a long itinerary, long query latency and high energy consumption may be incurred due to a long itinerary traversal. On the other hand, allowing a query to run on an arbitrary number of short itineraries in parallel may result in significant collisions in the query dissemination phase [26]. Thus, *itinerary planning* is an important design issue for itinerary-based KNN query processing. Prior works in [22], [23], [24] develop several itinerary structures. However, their proposals are not optimized neither in terms of the energy consumption nor the query latency. In this paper, we propose a new itinerary-based KNN query processing technique, called *Parallel Concentric-circle Itinerary-based KNN* (*PCIKNN*) query processing technique, which is based on parallel itineraries derived for optimizing either performance criterion of query latency or energy consumption.

While prior works have tested the idea of concurrent itineraries, the number of concurrent itineraries is not optimized. In contrast, PCIKNN allows a KNN query to propagate on an optimal number of concurrent itineraries (referred to as *KNN threads* in short) in order to achieve high efficiency in query latency or energy consumption. By avoiding the collision issue, PCIKNN reduces the query latency and the energy consumption by facilitating a larger number of concurrent KNN threads. Analytical models for the query latency and the energy consumption of PCIKNN are derived. By optimizing the query latency and the energy consumption, PCIKNN provides parallel itineraries in two modes: 1) *min_latency* mode and 2) *min_energy* mode, specifically tailored to minimize the query latency and the energy consumption, respectively.

3. Several geo-routing protocols (e.g., GPSR [7], PSGR [25], and other geo-routing protocols [8], [9]) are available for this task.

Another important issue for itinerary-based KNN query processing techniques is to estimate an initial KNN boundary to decide a coverage area for itinerary planning. By exploring regression techniques, we propose a boundary estimation method that accurately determines the KNN boundary. Note that sensors tend to be spread in a wide geographical space nonuniformly, which is called *spatial irregularity* [22]. Due to the spatial irregularity, communication voids may exist, and thus, degrade the performance of KNN processing. Explicitly, the spatial irregularity results in the inaccuracy of KNN query results because the KNN query and partial results may be blocked or dropped due to the communication voids. Moreover, KNN boundary is difficult to estimate precisely under spatial irregularity. To overcome the spatial irregularity, we develop methods to bypass void areas and dynamically adjust the KNN boundary. In addition, we propose a rotated itinerary structure for PCIKNN to alleviate an energy exhaustion problem in PCIKNN. The performance of PCIKNN is analyzed mathematically and evaluated through extensive experiments based on simulation. Experimental results show that PCIKNN is superior, in terms of accuracy, energy consumption, query latency, and scalability, to the state-of-the-art techniques. The contributions of this study are summarized as follows:

- An efficient itinerary design for in-network itinerary-based KNN query processing has been developed. The derived itineraries are tailored to optimize either query latency or energy consumption.
- A KNN boundary estimation method is developed to improve the query accuracy in wireless sensor networks.
- Methods to deal with issues caused by the spatial irregularity are developed to make PCIKNN robust in a large-scale sensor network.
- A comprehensive performance evaluation is conducted. Experimental results show the superiority of our proposal methods over other existing techniques.

The rest of the paper is organized as follows: Preliminaries are presented in Section 2. The ideas and design of the PCIKNN technique are described in Section 3. A KNN boundary estimation method is developed in Section 4 and the issues of spatial irregularity are addressed in Section 5. The performance of PCIKNN is evaluated in Section 6. Finally, the paper is concluded in Section 7.

2 Preliminaries

In this section, we first state the assumptions made in this paper and define the KNN problem in wireless sensor networks. Next, we discuss the issues of query dissemination in itinerary-based KNN query processing and review some closely related works. Finally, we provide an overview of PCIKNN.

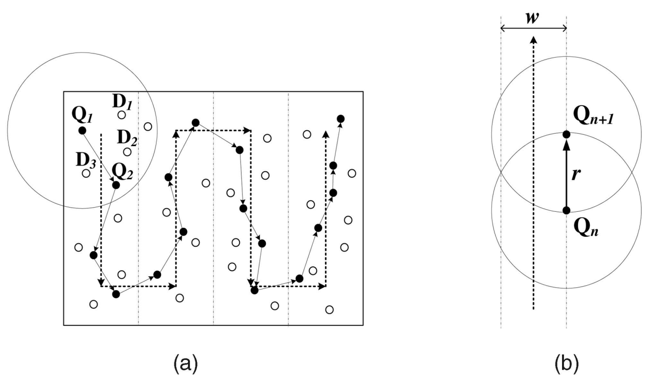


Fig. 2. Query dissemination in itinerary-based KNN query processing.

2.1 Assumptions and Problem Definition

We consider a wireless sensor network, where sensor nodes are deployed in a two-dimensional space. Each sensor is aware of its location via GPS or other localization techniques [14]. By periodically exchanging beacon information with sensor nodes nearby, a sensor node maintains a list of neighboring nodes. Moreover, the sensed data are stored locally in sensor nodes. In this network, a KNN query is issued at an arbitrary sensor node (called *source node*), which is the starting node for in-network query processing. The source node is responsible for reporting the final query result to the user. The KNN query in wireless sensor networks is formally defined as follows:

Definition (K-Nearest Neighbor query). *Given a set of sensor nodes M and a geographical location (denoted by a* query point *q), find a subset M' of M with 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 $dist(\cdot, \cdot)$ is the euclidean distance function.

Generally speaking, the "exact" result set of the KNN contains the recently sensed readings of the K sensor nodes nearest to the query point q. In this paper, however, due to the mobility of sensor nodes and dynamics of WSNs, the KNN query result may not contain the exact set of K-nearest neighbor nodes. Therefore, we measure the accuracy of the KNN query result as the *precision* of returned data, i.e., the ratio of correct KNN sensed data among the query result returned to the source node, where the correct KNN result refers to the set of sensed data from K nearest sensor nodes at the time when the query result is received at the source node.

2.2 Query Dissemination

As discussed before, query dissemination is a critical phase in itinerary-based KNN query processing. The detailed steps of itinerary-based query dissemination are illustrated in Fig. 2a, where the dotted line is a predesigned itinerary [26]. As shown in the figure, sensor nodes are divided into *Q-node* (marked as black nodes) and *D-node* (marked as white nodes). Upon receiving a query, a Q-node broadcasts a probe message that includes the KNN query 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 based on the *maximum progress heuristic*, i.e., the next Q-node with the farthest distance from the current Q-node along the proceeding itinerary direction. According to a prior work [26], the width of the itineraries w is set to $\frac{\sqrt{3}}{2}r$, where r is the transmission range of a sensor node (see Fig. 2b for illustration). In the query dissemination phase, only the D-nodes within the transmission range of a Q-node will send their sensed data to the Q-node. For example, as shown in Fig. 2a, D-nodes D_1, D_2, and D_3 send their data to Q_1. Since a D-node may receive multiple data probe messages from different Q-nodes, a flag is set when the D-node first receives a probe message for a given query in order to avoid redundant data collection. Finally, the last Q-node forwards the collected query result to the source node.

2.3 Related Work

In this section, we describe two itinerary-based KNN query processing algorithms in [24] and [22]. The authors in [26] first explore the idea of itinerary-based processing for window queries. They then further employ the idea for KNN query processing [24]. Thus, the proposed algorithm is called *Itinerary-based KNN* (IKNN) processing. In IKNN, the issue of KNN boundary determination is not addressed. The authors focus on the issues of designing itinerary structures and propose both sequential and parallel itinerary processing approaches. For the sequential itinerary approach, IKNN disseminates a KNN query along a spiral itinerary and collects data during the query dissemination phase. Once the query result contains sensed data from K nearest sensors, it is returned to the source node. To reduce the query latency, the parallel approach in IKNN allows two threads to disseminate query and collect data via two itineraries (see Fig. 3a). These two threads exchange the collected query results to determine whether the KNN query should be further propagated or not.

On the contrary, the authors in [22] propose to estimate a KNN boundary that is likely to contain K nearby sensor nodes. Accordingly, the KNN query is quickly propagated within this estimated boundary. Note that the KNN boundary is determined based on the node density estimated during the routing process. Thus, the proposed algorithm is called *Density-aware Itinerary-based KNN* (DIKNN). The itinerary structure for DIKNN is shown in Fig. 3b, where the KNN boundary is divided into several cone-shaped sectors. In each sector, KNN query is propagated along an itinerary. Among itineraries, the interitinerary information exchanging is adopted when adjacent itineraries are encountered at the sector border line. When the KNN query reaches the KNN boundary, the last Q-node in each sector directly sends the partial results to the source node. Through a good estimation of the KNN boundary, DIKNN improves its query latency over IKNN. However, the accuracy of the KNN boundary estimation is very critical. Although DIKNN dynamically adjusts its estimated KNN boundary, redundancy still exists in the KNN query result of DIKNN since the partial KNN query results from all sectors are sent back to the source node without any validation. Furthermore, itinerary structures developed in IKNN and DIKNN do not explore the issues

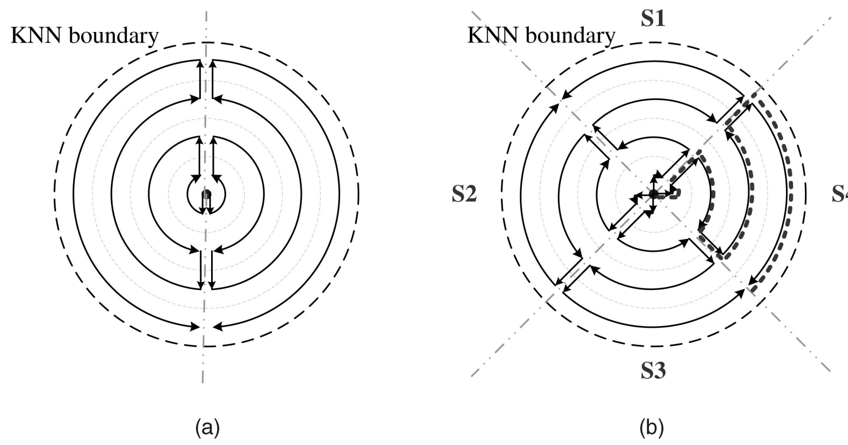


Fig. 3. Itinerary structures in (a) IKNN and (b) DIKNN.

of optimizing the number of KNN query threads. To deal with the above issues discussed, we propose a new itinerary-based query processing algorithm based on *optimized parallel concentric-circle itineraries*, namely *PCIKNN*.

2.4 Overview of PCIKNN

PCIKNN has also three processing phases: the routing phase, the KNN boundary estimation phase, and the query dissemination phase, as in the prior works [24], [22]. In addition, to improve the query accuracy of KNN query, PCIKNN employs several innovative ideas to improve the accuracy of KNN query result. Explicitly, the estimated KNN boundary is dynamically refined while the query is propagated within the KNN boundary. Moreover, to reduce redundant data transmission and improve the query accuracy, partial query results are collected at the home node and then sent back to the source node. The design of collecting partial results at the home node is able to dynamically adjust the KNN boundary, which further guarantees that the final query result contains K nearest sensor nodes.

A number of factors are considered in the design of PCIKNN, including the length and the number of itineraries. A query processed via long itineraries is expected to carry a large amount of collected data for a long way, thereby resulting in long query latency and severe energy consumption. On the other hand, by allowing a larger number of concurrent threads propagated along short itineraries, the latency of KNN query processing may be improved. The prior works [24], [22], [23], while exploring multiple itineraries, only allow the number of KNN query threads to be exactly the same as the number of itineraries. On the contrary, in PCIKNN, we aim at designing itineraries that allow more concurrent KNN query threads to be propagated. Since the routing phase of PCIKNN is the same as in [24], [22], [23], in the following sections, we only describe the itinerary structure employed in PCIKNN, our method for boundary estimation, and our proposals for improving the accuracy of KNN query result.

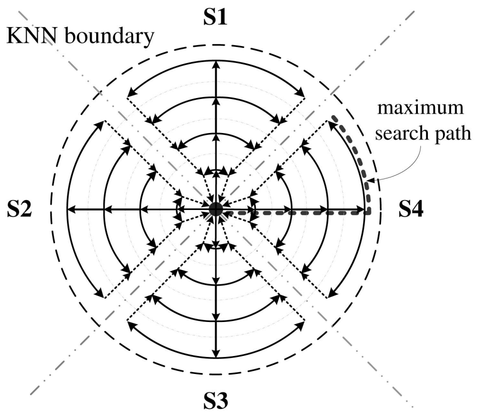
3 Itinerary Structures in PCIKNN

In this section, we first present the design of parallel concentric-circle itineraries in PCIKNN. Then, aiming at optimizing query latency or energy consumption, we analytically derive the number of parallel itineraries to be employed in PCIKNN. Analysis of itinerary structures among PCIKNN, IKNN, and DIKNN is then presented. Finally, we develop a rotated itinerary structure for PCIKNN to avoid the energy exhaustion in sensor nodes.

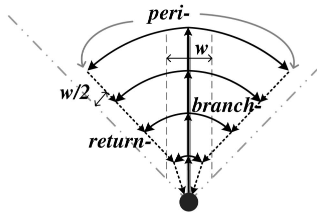
3.1 Design of Concentric-Circle Itineraries

Given a query point q and an estimated KNN boundary, the area within the boundary can be divided into multiple concentric-circle itineraries. The issue of estimating the KNN boundary will be addressed later in Section 4. Let C_i denote the ith circle with a radius $w \times i$, where w is the *itinerary width*, i.e., the distance between itineraries. As mentioned, w is set as $\frac{\sqrt{3}}{2}r$, where r is the transmission range of a sensor node. To propagate KNN query along concentric-circle itineraries, we partition the KNN boundary into multiple *sectors*. Fig. 4a 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) two *return-segments*. As shown in Fig. 4b, a branch-segment is a straight line through concentric-circles with the itinerary width w in each sector, two return-segments are the boundary lines among sectors with the itinerary width $\frac{w}{2}$, and peri-segments are portions of concentric-circles between branch-segments and return-segments. Obviously, there is no peri-segment in a concentric-circle if the regions of branch-segments and return-segments fully cover this concentric-circle. As shown in Fig. 4b, the arrows indicate the directions of query propagations. Following the itineraries in PCIKNN, a KNN query is executed concurrently.

In light of the itinerary structure derived above, a KNN query is first propagated along branch-segments in each sector. Along the branch-segment, a Q-node broadcasts a probe message and collects partial results from D-nodes within the region width of w. For each sector, when the KNN query reaches one of the 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. 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 divides the partial query result collected to these two Q-nodes. Then, these two Q-nodes will start performing KNN query dissemination and data collection along peri-segments.



(a)



(b)

Fig. 4. Parallel concentric itineraries in PCIKNN.

When the KNN threads that propagate along peri-segments arrive the boundary lines of their sectors, partial results collected by the KNN threads are sent back to the home node through return-segments. The above process of query dissemination pauses to wait for further instruction from the home node when the KNN query reaches the KNN boundary. If partial results collected at the home node do not contain K nearest sensor nodes, the KNN boundary will be extended and then the KNN query continues to propagate until the number of K nearest sensor nodes is collected. The adjustment of the KNN boundary is an important issue and will be described later. Clearly, in PCIKNN, the number of the concurrent KNN threads is greater than that in IKNN and DIKNN. Given an optimization objective in minimizing either the minimum latency or the minimum energy, we optimize the number of concurrent KNN query threads.

In PCIKNN, because the home node receives all partial results from sectors within the KNN boundary, the home node is thus able to determine whether to continue the KNN query propagation or not. In PCIKNN, if partial results collected at the home node contain K nearest sensor nodes, the home node will inform the KNN query to stop query propagation even if the KNN query has not yet reached the KNN boundary. With our design of collecting partial results at the home node, the KNN boundary is dynamically adjusted. When the KNN query arrives the KNN boundary and the number of K nearest sensors is not larger than K, the home node is able to dynamically extend the KNN boundary to discover more sensor nodes. According to the optimization goal (i.e., the minimum latency or the minimum energy), the corresponding scheme for adjusting the KNN boundary is developed in Section 5. Consequently, by collecting partial results at the home node, PCIKNN not only improves the query accuracy but also avoids unnecessary query propagation.

One may argue that our design of itineraries will result in a prolonged query latency and quick energy exhaustion in sensor nodes along the return-segments. For example, as Fig. 5 shows, returning via a path from a boundary point to the source node (indicated by dotted line) is always shorter than bypassing the home node than the source node (indicated by the solid lines). This results in a longer query latency of PCIKNN than that of IKNN and DIKNN. However, as we will show in Section 6 later, the gain in query latency obtained due to the high parallelism of PCIKNN can easily offset the delay discussed here. In addition, as shown in Section 3.4, we further propose a rotated itinerary structure in PCIKNN to alleviate the energy exhaustion of sensor nodes along the return-segments. In summary, with a proper itinerary design, PCIKNN achieves high performance in terms of both the query latency and the energy consumption.

3.2 Optimal Number of Sectors in PCIKNN

PCIKNN aims at exploring parallel concentric-circles itineraries to achieve parallelism while reducing query latency or energy consumption in KNN query processing. Thus, to determine an appropriate number of sectors (denoted by S) is a critical issue. In this section, we first discuss the trade-off between the number of sectors in PCIKNN and the itinerary length within a sector. Then we derive analytical models to determine the optimal number of sectors based on two optimization goals: 1) the minimum latency (referred to as *min_latency*) and 2) the minimum energy (referred to as *min_energy*), respectively.

3.2.1 Trade-Off in PCIKNN Itinerary Design

In PCIKNN, when the number of sectors is increased, the maximal length of itineraries in each sector becomes shorter. Clearly, the query latency and the energy consumption in each sector are reduced. However, the total length of itineraries is likely to become longer as the number of sectors increases. As a result, the energy consumption may

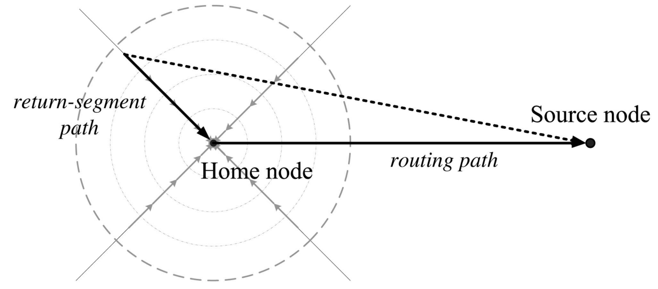


Fig. 5. An illustrative example of returning partial results to the home node.

TABLE 1
Summary of Symbols Used in the Analytical Models

Parameters	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)
D_{size}	D-node message size $(bits)$
H_{size}	message header size $(bits)$
$Bits$	energy to transmit one data bit per hop

be increased. On the other hand, when a smaller number of sectors are adopted in PCIKNN, the itinerary length within a sector may increase due to the existence of peri-segments. This leads to a higher energy consumption and a longer query latency for each sector. From the above observations, we recognize a need to strike a balanced trade-off between the number of sectors and the itinerary length in each sector in order to optimize the performance of PCIKNN.

As query latency and energy consumption are the two most critical performance metrics for KNN query processing in wireless sensor networks, we derive optimized PCIKNN itinerary structures using them as the optimization objectives.

3.2.2 Notations and Assumptions

Given a KNN boundary with radius R and the network density d, we intend to derive an optimal number of sectors to meet the optimization objectives. Note that the network density can be estimated while routing KNN query to the home node, which will be described later. Assume that sensor nodes are uniformly distributed and message transmissions are reliable. Moreover, KNN queries propagate along branch-segments and return-segments hop by hop at each concentric-circle. Each Q-node is ideally located in the itineraries. Symbols used in our analysis are summarized in Table 1.

3.2.3 Minimum Latency for PCIKNN

In PCIKNN, when the number of sectors is increased, the length of peri-segments is expected to be shortened, thus, reducing the latency. Once a KNN query along peri-segments arrives the boundaries among sectors, the partial result is transmitted to the home node along return-segments. Thus, we have the following observation:

Observation 1. Since the latency on propagating the KNN query along the branch-segments remains a constant, the primary factors for the overall latency of PCIKNN are 1) the latency of propagating KNN query along the peri-segments at the concentric-circle "farthest" from the home node and 2) the corresponding latency for returning the partial results back to the home node.

Let $latency_{peri}$ denote the latency of propagating KNN query along the peri-segments at the concentric-circle farthest from the home node and $latency_{home}$ denote the

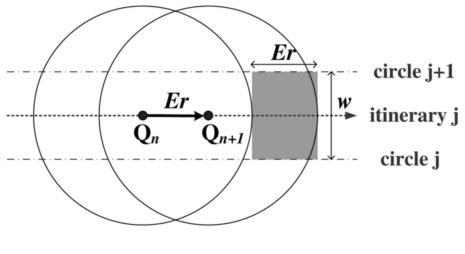


Fig. 6. Information statistics in a routing path.

latency for delivering the collected partial results to the home node. Therefore, the latency of PCIKNN is formulated as

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

To determine $latency_{peri}$, we take into account message delays for sending probe messages and receiving D-nodes' messages at each Q-node. Thus, the value of $latency_{peri}$ is formulated as follows:

$$latency_{peri} = E_{hop}^{peri} \times \left(1 + E_{D_{num}}^{peri}\right) \times Delay,$$

where E_{hop}^{peri} is the expected number of Q-nodes in the peri-segment at the farthest concentric-circle and $E_{D_{num}}^{peri}$ is the expected number of D-nodes of a Q-node.

Observation 2. The expected number of D-nodes is estimated as the number of D-nodes in the gray area in Fig. 6.

In light of Observation 2, we have

$$E_{hop}^{peri} = \frac{2 \times \pi \times R \times w}{2 \times S \times Er}; \quad E_{D_{num}}^{peri} = Er \times w \times d,$$

where Er is the expected length of each hop of Q-nodes. According to [22], [23], $Er = r^2\sqrt{d}/(1 + r\sqrt{d})$.

As for the latency spent on collecting partial results at the home node, assume that partial results are sent to the home nodes at the same time (in the worst-case scenario). The latency for collecting partial results at the home node is formulated as follows:

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

According to the above deviations, the overall query latency for PCIKNN is derived as follows:

$$latency = \left(\frac{\pi \times R \times w}{S \times Er} \times (1 + w \times Er \times d) \times Delay\right) + (2 \times S \times Delay).$$

In order to obtain the optimal number of sectors to achieve the minimum latency, we differentiate the above equation to derive the optimal number of sectors as follows:

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

3.2.4 Minimum Energy for PCIKNN

Similar to IKNN and DIKNN [22], [24], the KNN query considered in this paper facilitates sampling/collection of sensed data in correspondence with a given query location and the number of samples specified (i.e., K). As a result, we do not take any aggregation function into consideration in this work. Since the main function of the KNN query here is data collection, a longer itinerary length incurs more energy consumption.

Observation 3. A small number of sectors lead to long itineraries within a sector, incurring heavy energy consumption overhead in carrying data collected from D-nodes. On the contrary, a large number of sectors increase the number of branch-segments and return-segments, increasing the total energy consumption on the query propagation and data collection along the branch-segments, and the partial result delivery along the return-segments.

Inspired by the above observation, we derive an optimal number of sectors for minimizing energy consumption. Generally speaking, the energy consumption of PCIKNN involves two parts in each itinerary segment: 1) energy consumed for carrying data hop by hop along with the KNN query and 2) energy consumed for propagating the KNN query among Q-nodes. Without loss of generality, the energy consumption is modeled as a 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. We denote energy consumption on the *type*-segment in the ith concentric-circle itineraries 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

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

where the number of concentric-circles is R/w.

The energy consumption on data transmission 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,C_i} denote the expected number of hops in the type-segment on C_i. For example, E^{branch}_{hop,C_i} is the expected number of hops in the branch-segment on C_i. Furthermore, $E^{type}_{D_{num}}$ represents the expected number of D-nodes of a Q-node in the *type*-segment. Let the radius of C_i be $i \times w$. In the following, we derive the energy consumption along the itinerary segments.

First, the energy consumption of sensor nodes along the branch-segment to C_i is formulated as follows:

$$energy_{branch,C_i} = E^{branch}_{hop,C_i} \times \left(H_{size} + \left(E^{branch}_{D_{num}} \times D_{size}\right)\right) \times Bits,$$

where $E^{branch}_{hop,C_i} = 1$, $E^{branch}_{D_{num}} = 0$ 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.

Next, the energy consumption of sensor nodes along the peri-segments is derived as follows:

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

where

$$E^{peri}_{hop,C_i} = \frac{2 \times \pi \times i \times w}{2 \times S \times Er}$$

and

$$E^{peri}_{D_{num}} = Er \times w \times d.$$

Finally, the energy consumption of sensor nodes along two return-segments in each sector is modeled as follows:

$$energy_{return,C_i} = 2 \times E^{return}_{hop,C_i} \times \left(H_{size} + \left(E^{return}_{D_{num},C_i} \times D_{size}\right)\right) \times Bits,$$

where

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

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

$$S = \sqrt{\frac{\frac{\pi^2 w^3 d}{Er} \times \frac{(2\frac{R}{w}+1)}{6} \times D_{size}}{H_{size}}}.$$

3.2.5 Model Validation

We develop a simulator to validate our derivations. We simulate a wireless sensor network that consists of 1,000 sensor nodes randomly distributed in a 500×500 field. A total of 100 KNN queries are issued and K is set to 300. The simulation results are shown in Fig. 7. The minimum latency model determines the value of sectors to be 6 (rounded from our analytical result 6.4). As shown in Fig. 7a, we select the number of sectors to be 6, which incurs the minimum query latency. Moreover, the optimal number of sectors for the minimum energy consumption is 7 (rounded from our analytical result 7.1), which is consistent with the experimental result as shown in Fig. 7b. The above comparisons show that our analysis is very accurate. From our analytical models, we could easily determine the number of sectors based on the targeted optimization objectives. Note that the above derivations are under the assumption that sensors are uniformly distributed. To further validate the applicability of our derivations under various network conditions, we conduct experiments by varying the network density. Fig. 8 shows the experimental results under various network densities. The number of sensor nodes is from 500 to 1,500 within a fixed monitored region (i.e., $500 \times 500\ \mathrm{m}^2$). As shown in Fig. 8, when the network density is larger, the optimal numbers derived from analytical models for minimum latency and minimum energy (and thus, denoted by Minimum Latency Model and Minimum Energy Model in the figures, respectively) are very close to the optimal numbers obtained empirically (denoted by Latency and Energy in the figures, respectively). On the other hand, with a

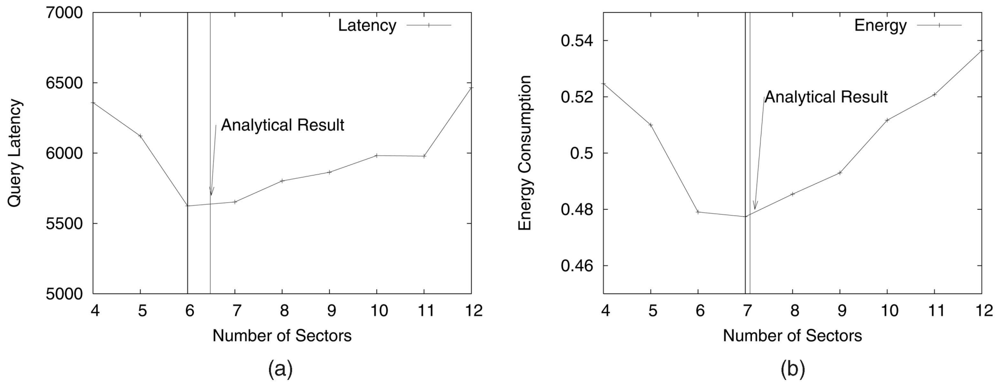


Fig. 7. Comparison of the optimal number of sectors derived by analytical model to simulation results. (a) Minimum latency model. (b) Minimum energy model.

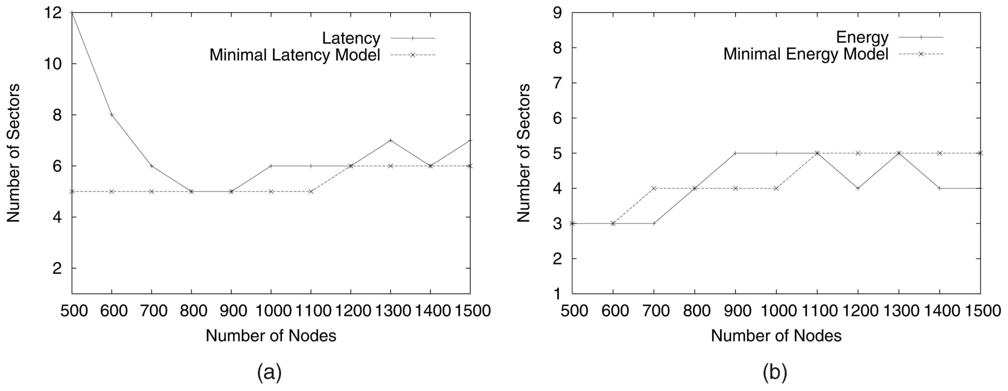


Fig. 8. Models validation with the network density varied. (a) Minimum latency model. (b) Minimum energy model.

smaller network density, the difference between analytical and experimental results is increased. This is because of the voids appearing in networks with low network density. While the analytical models may not be used directly in this situation, they do provide good insight for network planning. Moreover, we may still choose to empirically determine the optimal number of sectors in this situation.

3.3 Analysis of Itinerary Structures

In this section, we analyze the performances of IKNN, DIKNN, and PCIKNN in terms of the number of concurrent query threads, the query latency, and the energy consumption. Without loss of generality, we assume that sensor nodes are uniformly distributed in the monitored region. Moreover, the KNN boundary is known (i.e., it will be derived as shown in Section 4). The radius R of the KNN boundary is set to $c \times w$, where c is the number of concentric-circles in the KNN boundary and w is the width of itineraries for query propagation and data collection. For example, as shown in our illustrative example (see Fig. 4), the radius of the KNN boundary is $4 * w$ since there are four concentric-circles. Since the query latency and the energy consumption are proportional to the length of itineraries, we compare DIKNN and PCIKNN in terms of the length of itineraries. Note that since DIKNN outperforms IKNN, we only compare DIKNN and PCIKNN.

3.3.1 Number of Concurrent Query Threads

Fig. 3a shows the itinerary structure of IKNN proposed in [24]. As shown, the number of concurrent query threads in IKNN is 2. For DIKNN and PCIKNN, the number of sectors directly affects the number of concurrent query threads. It can be seen in Fig. 3b that only one itinerary exists in a DIKNN sector, and hence, the number of concurrent query threads in DIKNN is exactly the same as the number of sectors. In each PCIKNN sector, there are one query thread along the branch-segment and two query threads along the peri-segments in each concentric-circle. Therefore, the maximal number of concurrent KNN query threads is $(2 \times c + 1) \times S$. For example, if the number of sectors is set to 4 and the number of concentric-circles is 4, we have 4 and 36 concurrent KNN query threads in DIKNN and PCIKNN, respectively. Clearly, PCIKNN has more query threads than IKNN and DIKNN.

3.3.2 Query Latency

The query latency is determined by the maximal length of itineraries in DIKNN and PCIKNN. Therefore, we derive the maximal length of itineraries shown by the dotted lines in Fig. 3b (for DIKNN) and Fig. 4a (for PCIKNN) to represent the analytical latency. As shown in Fig. 3b, the analytical query latency of DIKNN, denoted by $latency_{DIKNN}$, is the sum of the total lengths of concentric-circles in a sector and the length of a branch-segment. Hence, we have

$$latency_{DIKNN} = \sum_{i=0}^{\frac{R}{w}} (Length_{C_i}) + Length_{branch},$$

where the length of itinerary in the ith concentric-circle, denoted by $Length_{C_i}$, is formulated as $\frac{2\pi(i \times w)}{S}$, and the length of branch-segment, expressed by $Length_{branch}$, is set to $(c - \frac{1}{2}) \times w$.

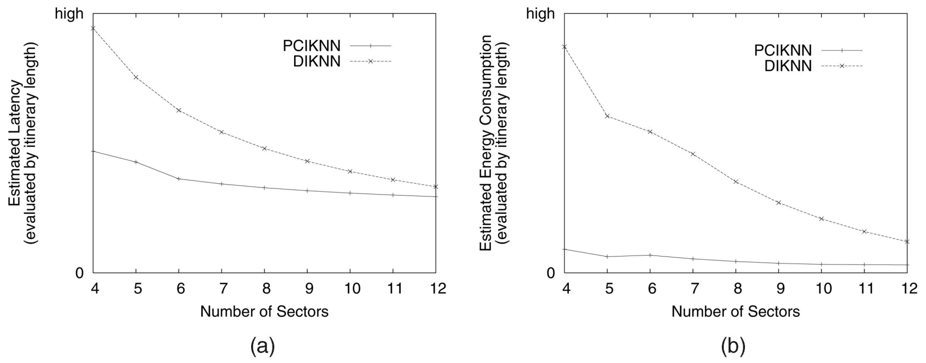


Fig. 9. Latency and energy consumption with varied number of sectors (analytical results). (a) Latency analysis. (b) Energy analysis.

Because the query latency is affected by the maximal length of itineraries, the analytical query latency of PCIKNN, denoted by $latency_{PCIKNN}$, is derived as the sum of the length of a peri-segment in the maximal concentric-circle and the length of a branch-segment. Thus, we have

$$latency_{PCIKNN} = Length_{peri} + Length_{branch},$$

where the length of the peri-segments represented as $Length_{peri}$ is $\frac{\pi \times R}{S}$ in the farthest concentric-circle and the length of a branch-segment, denoted by $Length_{branch}$, is derived as $(c - \frac{1}{2}) \times w$.

The analytical latency of DIKNN and PCIKNN is compared by varying the number of sectors S (as shown in Fig. 9a). It can be seen in Fig. 9a that the analytical latency in DIKNN and PCIKNN decreases when the number of S increases. As the number of sectors increases, the maximal length of itineraries in both DIKNN and PCIKNN decreases. As shown, the analytical latency of PCIKNN is smaller than that of DIKNN. Thus, we expect PCIKNN to achieve a better latency performance than DIKNN.

3.3.3 Energy Consumption

With a longer itinerary, the number of Q-nodes on the itinerary and the amount of data carried are increased. Thus, the analytical energy consumption is estimated as $S * energy_s$, where $energy_s$ is the energy consumption within one sector and S is the number of sectors. Intuitively, the value of $energy_s$ can be derived by integrating the length of itineraries and the amount of data carried. Denote the length of itineraries in a sector by $length_s$. Since the amount of data carried is directly proportional to the itinerary length, we could use a continuous function of itinerary lengths, denoted by $D_{amount}(*)$, to model the energy consumption. Hence, we have the following formula:

$$energy_s = \int_{x=0}^{length_s} D_{amount}(x) \times d(x) \propto (itinerary\_length)^2.$$

As shown, the analytical energy is a quadratic function of itinerary lengths within a sector. The analytical energy of DIKNN is formulated as the sum of the square of the total lengths of concentric-circles in a sector and the length of a branch-segment. Hence, we have the following:

$$energy_{DIKNN} = S \times \left(\sum_{i=0}^{\frac{R}{w}} (Length_{C_i}) + Length_{branch} \right)^2.$$

Note that the itineraries for the data collection of PCIKNN in a sector include a branch-segment and peri-segments. When a KNN query propagating along a branch-segment encounters a new concentric-circle, the KNN query is forked into two KNN query threads along the two peri-segments. Meanwhile, the data collected along the branch-segment are equally divided into two parts for two new KNN query threads. As a result, the amount of data carried along these itineraries becomes smaller, thereby reducing the energy consumption. The analytical energy of PCIKNN is formulated as the sum of the square of each subitinerary (a partial branch-segment and a peri-segment in a concentric-circle):

$$energy_{PCIKNN} = S \times \left(\sum_{i=0}^{\frac{R}{w}} (Length_{peri,C_i} + Length_{branch,C_i})^2 \right),$$

where the length of a peri-segment in the concentric-circle C_i is $Length_{peri,C_i} = Length_{peri} = \frac{\pi \times R}{S}$ and the partial branch-segment from a concentric-circle C_{i-1} to its next concentric-circle C_i is $Length_{branch,C_i}$ (i.e., w).

In light of the analytical energy formulas for DIKNN and PCIKNN, we compare the analytical energy results by varying the number of sectors. As shown in Fig. 9b, both DIKNN and PCIKNN have decreased the energy consumption as the number of sectors increases. Furthermore, PCIKNN has a smaller energy consumption than DIKNN. From the above, we observe that PCIKNN obtains a smaller query latency and the energy consumption than IKNN and DIKNN. This is because PCIKNN achieves a good parallelism by allowing as many concurrent KNN query threads as possible.

3.4 Rotated Itinerary Structures of PCIKNN

In PCIKNN, partial results are returned along the return-segments to the home node. A potential concern is that sensor nodes along the return-segments may get their energy exhausted faster than other sensor nodes. To overcome this issue, we develop a rotated itinerary structure in PCIKNN. Without loss of generality, we consider an example, where the number of concentric-circles is 4 (i.e., $C = 4$). Fig. 10 shows an example of a rotated itinerary

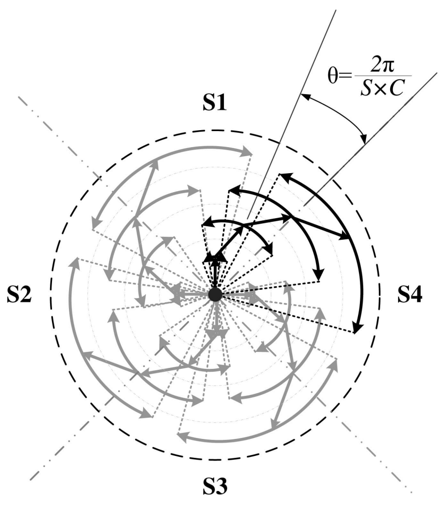


Fig. 10. A rotated itinerary structure in PCIKNN.

structure in PCIKNN, where the bold line shows an itinerary structure within a sector. As shown in Fig. 10, for each concentric-circle, the branch-segment is rotated by θ degree. Assume that S is the number of sectors and C is the number of concentric-circles. The rotation angle θ is set to $\frac{2\pi}{S \times C}$ such that the return-segments of all concentric-circles are evenly distributed within a KNN boundary. The dotted straight lines in Fig. 10 are the return-segments. For each concentric-circle, sensor nodes along the return-segments are different, and thus, the energy consumption of sensor nodes along the return-segments is balanced. While the rotated itinerary of PCIKNN deals with the energy exhaustion problem of sensor nodes along the return-segments, the query latency is slightly increased because the total length of branch-segments in one sector is increased. Experimental results to be presented later show that the rotated itinerary of PCIKNN is effective since the overhead in query latency is very low.

4 KNN Boundary Estimation

An overestimated KNN boundary leads to excessive energy consumption and long latency, whereas an underestimated KNN boundary deteriorates the accuracy of query results. Thus, boundary estimation is very critical to itinerary-based KNN query processing. Without a priori knowledge about the network density and distribution of sensor nodes, it's challenging to obtain an accurate estimation of the KNN boundary. To address this issue, DIKNN [22], [23] collects network information during the routing path of KNN query to derive the network density, which, in turn, is used to estimate a KNN boundary. Clearly, how to determine the network density precisely from the partial information gathered in the routing phase is an important research issue.

Via a geo-routing protocol, such as GPSR, a KNN query is greedily forwarded from the source node to the home node. Let A_i denote the area covered by relaying messages up to the ith hop and Num denote the total number of nodes within the coverage area of the routing path. By collecting the information for A_i and Num while the KNN query moves hop by hop toward the home node, the network density can be estimated.

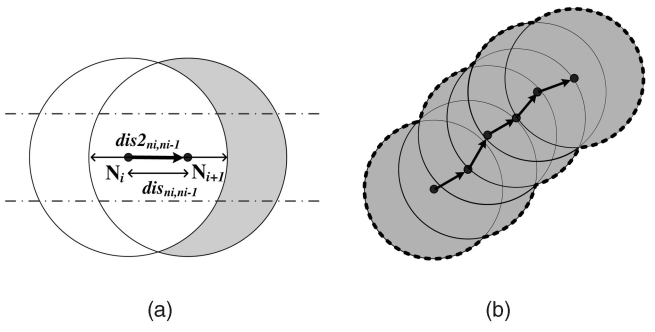


Fig. 11. Coverage areas estimated in the routing phase.

Here, we describe how PCIKNN updates these two values during the routing phase. Fig. 11a shows a message transmitting from node N_i to node N_{i+1}. In the figure, the gray area is the newly explored area, denoted by EA_i. The number of sensor nodes in EA_i is denoted by inc_{i+1}. By adding inc_{i+1} to Num, we have the 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 range of a sensor node, H(*) is a linear function, and $dist(N_i, N_{i+1})$ is the euclidean distance between N_i and N_{i+1}. From experiments shown in Fig. 12, we observed that the intersected area between two sensor nodes is almost negative correlated with $dist(N_i, N_{i+1})$. Thus, in this paper, to precisely estimate the intersection area between two sensor nodes, we employ a linear regression technique to formulate H(*) as follows:

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

where c_1 and c_2 are coefficients determined by Leon [10].

Hence, the total area covered by relaying messages up to the 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 network density D is formulated as $D = \frac{Num}{A}$, where A is the total area covered by relaying messages from the source node. As a result, the radius of KNN boundary is estimated as follows:

$$\pi R^2 \times D = K$$

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

Although DIKNN [22], [23] also explores the network density in estimating the KNN boundary region, an extra-list is used to record collected local information in each hop visited in the routing phase. This list, sent along with the KNN query, incurs extra energy overhead. Furthermore, DIKNN estimates KNN boundary by Algorithm KNNB in [22] based on the collected local information. On the contrary, PCIKNN utilizes a linear regression technique to estimate the total area covered by sensor nodes. Our proposal is validated by experiments and will be presented later.

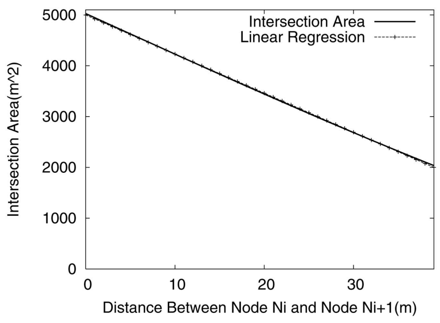


Fig. 12. Intersection covered area with various node distances.

5 Spatial Irregularity

Thus far, the analytical models and boundary estimation we derived are based on an assumption that the sensor nodes are uniformly distributed in the monitored region. However, as discussed before, spatial irregularity is likely to occur in large-scale networks, resulting in communication voids. The voids may deteriorate the accuracy of KNN boundary estimation and KNN query results. Following up the ideas previously explored in GPSR, we develop a mechanism to bypass voids during KNN query propagation. Additionally, we develop another mechanism to dynamically adjust KNN boundary.

5.1 Bypassing Void Regions in PCIKNN

In the scenario where a message (with the KNN query or partial results) reaches a void region, the message needs to find a way to bypass the void. Similar to GPSR, PCIKNN adopts two modes (i.e., the greedy mode and the perimeter mode) in the KNN query propagation. In the greedy mode, a message is forwarded by selecting the sensor node making the most progress toward the destination. When a void region is encountered, PICKNN switches to the perimeter mode. After bypassing voids, it changes back to the greedy mode and continues to move forward. Since void regions may appear in branch-segments, peri-segments, and return-segments, we develop methods to bypass void regions, correspondingly. Fig. 13 shows an example of bypassing void region along the branch-segments and peri-segments, where gray areas are void regions. In the figure, the bold and dotted lines refer to the KNN query propagation along the branch-segments andperi-segments, respectively. Also, the black and gray nodes are Q-nodes and D-nodes, respectively.

When a KNN query reaches a void region on a branch-segment, the KNN query is split into two KNN query threads. To simplify our discussion, we call them the left KNN query thread and the right KNN query thread. The left KNN query thread continues to move forward by using the left-hand rule to select the next Q-node close to the branch-segment. The right KNN query thread acts similarly based on the right-hand rule. If these two KNN query threads reach a concentric-circle, both two KNN query threads fork two additional KNN query threads and move forward along the peri-segments. Note that after bypassing void regions, both the left and the right KNN query threads will merge into one KNN query that keeps propagating along the branch-segment. If a void exists on a peri-segment, the KNN query will decide which rule to use

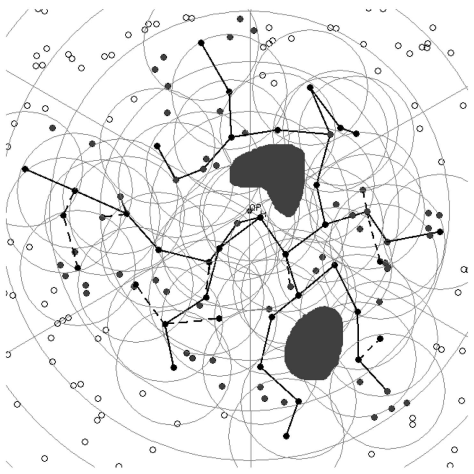


Fig. 13. An example of bypassing void regions in PCIKNN.

according to the relative position of the peri-segment. If a KNN query thread is on the left (or right, respectively) peri-segment from the perspective of the propagating direction, the KNN query thread will perform the left-hand (or right-hand, respectively) rule. Similar to the bypassing method for voids on peri-segments, a message carrying partial results along return-segments decides its bypassing rules based on the relative position of return-segments.

5.2 Adjusting Estimated KNN Boundary

The KNN boundary initially estimated at the home node may actually contain less than K sensor nodes. Thus, there is a need to adjust the KNN boundary when the KNN query reaches nodes at the KNN boundary. In PCIKNN, partial results from each sector are returned to the home node. Therefore, the home node is able to decide whether the estimated boundary should be further extended or not. Corresponding to the two optimization objectives (i.e., the minimum energy mode or the minimum latency mode), we develop two different schemes for adjusting the KNN boundary.

5.2.1 KNN Boundary Adjustment for the Min_Energy Mode

When a KNN query reaches a sensor node at the KNN boundary, the KNN query will wait at the node for a control message from the home node to signal whether the KNN query should continue to the next concentric-circle or not. Upon reception of all partial results from sectors within the initial KNN boundary, the home node checks its data collection against the query parameter K. If the number of collected sensor readings is smaller than K, the home node broadcasts a control message to inform the KNN query to further propagate to the next concentric-circle in order to collect more data. Note that the radius of the KNN boundary is extended in a step-by-step manner to conserve energy. The process of extending KNN boundary stops when the home node has collected data from the K nearest sensor nodes.

5.2.2 KNN Boundary Adjustment for the Min_Latency Mode

In this mode, the ultimate goal for PCIKNN is to minimize the query latency. Thus, KNN boundary adjustment is proceeded in an aggressive manner to reduce query latency.

The idea is to allow the KNN query to continue to propagate toward outer concentric-circles as long as the home node does not have sufficient data from the K nearest sensor nodes. Specifically, when the KNN query arrives the KNN boundary, the KNN query will hold for a period of time to wait for the control message that indicates the updated radius of KNN boundary. The holding time is set as the sum of the latency of KNN query propagation along peri-segments, the time of sending partial results along return-segments, and the transmission time of broadcasting control messages along branch-segments to the KNN query. In fact, the holding time can be determined by our derivation in Section 3.2.3. Different from the scheme for the Min_Energy Mode, the KNN query will continue to propagate outward when the holding time is expired. At the home node, some partial results are returned as the KNN query propagates along itineraries in each sector. Once the number of sensor readings collected is equal to or larger than K, the radius of the KNN boundary is updated as the distance between the query point and the Kth nearest sensor node so far. The updated KNN boundary is broadcasted along the branch-segments to the KNN query. According to the updated KNN boundary and the position of the KNN query, the nodes holding KNN query determine whether they should continue to propagate or not. If the KNN query is outside the updated KNN boundary, this KNN query stops. Otherwise, the KNN query will keep propagating.

6 Performance Evaluation

In this section, we develop a simulator to evaluate the performance of PCIKNN, IKNN, and DIKNN. The simulation model and parameter settings are presented in Section 6.1 and the experimental results are reported in Section 6.2.

6.1 Simulation Model

Our simulation is implemented in CSIM [17] and some simulation settings are the same as the prior work [24]. There are 1,000 sensor nodes randomly distributed in a $500 \times 500\ \mathrm{m}^2$ region and the transmission range of a node is 40 m. For each sensor node, the average number of neighboring nodes is 20 by default. The message delay for transmitting or receiving messages is 30 ms. In our default settings, sensor nodes are static. For each query, the location of a query point q is randomly selected. The default value of K for each KNN query is 100. A sensed datum is 4 bytes long and the query result is not aggregated. The broadcasting period of beacon messages is 3 s. A KNN query is considered as answered when the query result is returned to the source node. In each round of experiment, five queries are issued from randomly selected source nodes. Each experimental result is derived by obtaining average results from 50 rounds of experiments.

Three itinerary-based KNN algorithms (i.e., PCIKNN, IKNN, and DIKNN) are implemented. For a fair comparison, we obtain the result of DIKNN for all possible number of sectors and use the result of DIKNN with the minimum latency/energy consumption. In PCIKNN, we show the results for the minimum latency mode and the minimum energy mode, respectively. We compare these 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 sizes (i.e., K for KNN queries), the node mobility, and the failure rate of nodes. Three performance metrics are defined as follows:

- **Energy Consumption (Joules):** The total amount of energy consumed for processing a KNN query.
- **Query Latency (milliseconds):** The elapsed time between the time a query is issued and the time the query result is returned to the source node.
- **Query Accuracy (percent):** The ratio of the correct sensor readings of K nearest sensor nodes to the total number of sensor readings returned to the source node.

6.2 Experimental Results

6.2.1 Impact of Network Density

First, we investigate the impact of network density on the performance of examined algorithms. Here, the network density is measured as the number of sensor nodes deployed in a fixed monitored region (i.e., $500 \times 500\ \mathrm{m}^2$). Thus, we investigate the impact of network density by varying the number of nodes from 500 to 1,500. As a result, the average number of neighbors for each node is varied from 10 to 30. Fig. 14a 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 outperforms IKNN and DIKNN. The reason is that the itineraries in IKNN and DIKNN are longer than that in PCIKNN. Thus, the KNN query on IKNN and DIKNN may easily get dropped. Due to the high parallelism and short length in itineraries, PCIKNN is very robust. Furthermore, DIKNN does not have a way to ensure the accuracy of KNN query results since all partial results are directly sent to the source node. As a result, the query accuracy of DIKNN is significantly affected by spatial irregularity. In a dense network (i.e., the number of nodes is larger than 900 nodes), both IKNN and PCIKNN have better query accuracy. As can be seen in Fig. 14b, the latency of PCIKNN in both minimum latency and minimum energy modes is the lowest among these three algorithms, showing the strength of concurrent KNN query propagation. In a dense network, the latency of the three algorithms tends to decrease. Clearly, the KNN boundary is small in a dense network, leading to a short latency. Fig. 14c shows the energy consumption of the three algorithms. PCIKNN has the lowest energy consumption, showing the merits of itinerary design in PCIKNN. Note that when the network density is smaller, there are more voids in the monitored region. Fig. 15 depicts sensor distributions under various network density settings. As shown in Fig. 15a, void regions frequently appear in networks with low network density. From the experimental results observed above, PCIKNN has the best performance under the settings with low network density because PCIKNN has a mechanism to bypass voids.

6.2.2 Impact of the Sample Size

Next, we study the impact of the sample size K on scalability of the three examined 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 Fig. 16a. It can be

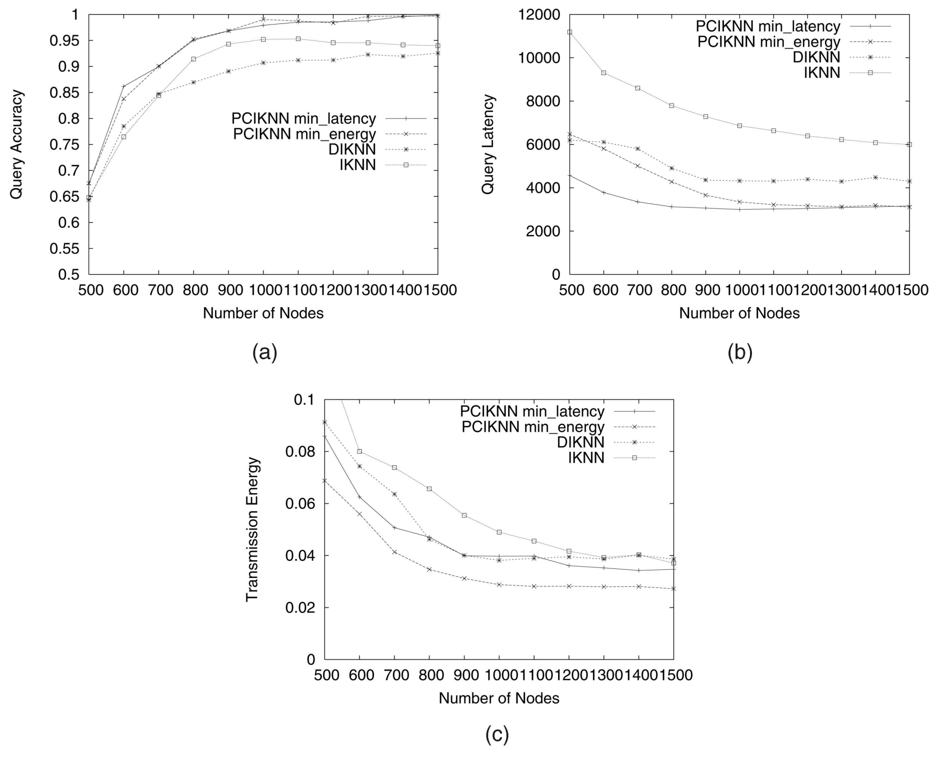


Fig. 14. Impact of network density. (a) Query accuracy. (b) Query latency. (c) Energy consumption.

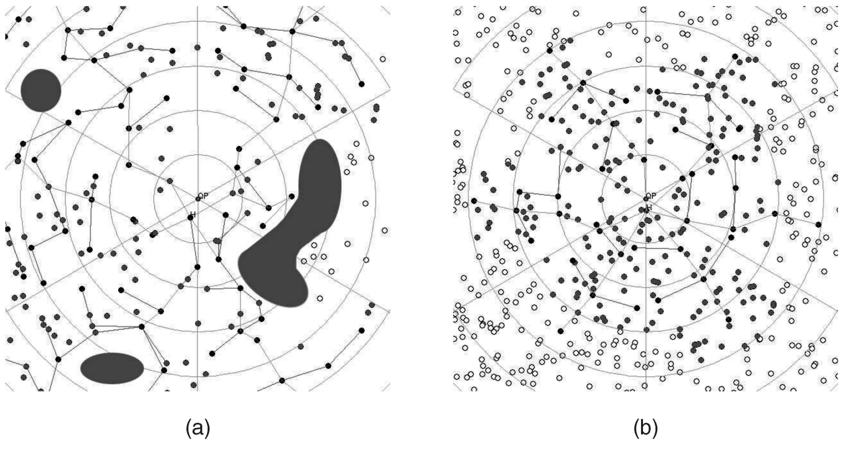
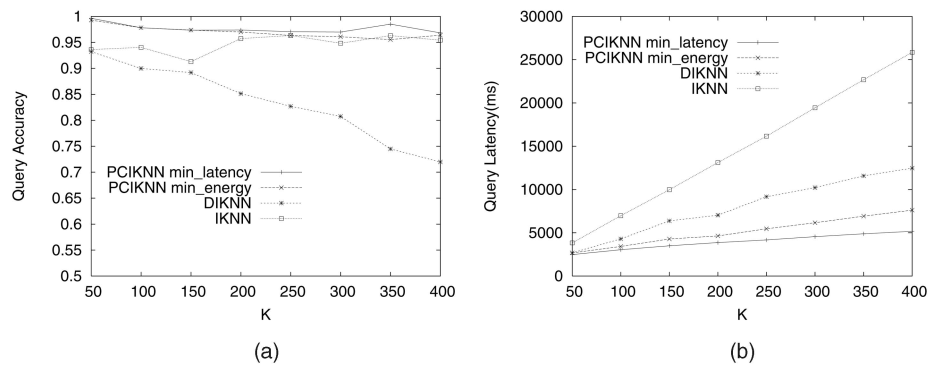


Fig. 15. Visualization of sensor distributions under various network density settings. (a) Low density. (b) High density.

observed that the query accuracy of DIKNN is drastically decreased as K increases. With a larger value of K, DIKNN seriously suffers from spatial irregularity. This is because the KNN query result obtained via each itinerary in DIKNN is directly sent back to the source. On the other hand, PCIKNN and IKNN have good query accuracy under varied K because both IKNN and PCIKNN have built-in mechanisms at the home node to ensure result accuracy. The latency of the three algorithms tends to increase as K increases. As can be seen in Fig. 16b, PCIKNN has the smallest latency, validating our analytical model for minimum latency of PCIKNN. Meanwhile, the energy consumption of all algorithms tends to increase as K increases. This is because the number of sensor nodes involved in KNN query is increased. Still, PCIKNN has the smallest energy consumption. Furthermore, PCIKNN with the minimum energy mode (i.e., PCIKNN min_energy) indeed has the minimal energy consumption, validating the correctness of our optimization. Note that when$K$ is small, the performance of DIKNN is very close to that of PCIKNN although PCIKNN is still better. From the experimental results, we could observe that when K is small, the performance of DIKNN is almost the same as the PCIKNN in terms of the query latency and the total energy consumption. On the other hand, PCIKNN is significantly better than DIKNN when the value of K is larger.

6.2.3 Impact of Node Mobility

We now conduct experiments with some mobile sensor nodes. Clearly, sensor node movements have an impact on the query accuracy. The moving speed of mobile sensor nodes is varied from 1 to 15 m/s. The accuracy of query results is shown in Fig. 17a. From Fig. 17a, the accuracy of query results obtained by the three algorithms drastically decreases as the moving speed increases. Clearly, with faster moving speeds of mobile sensor nodes, the query



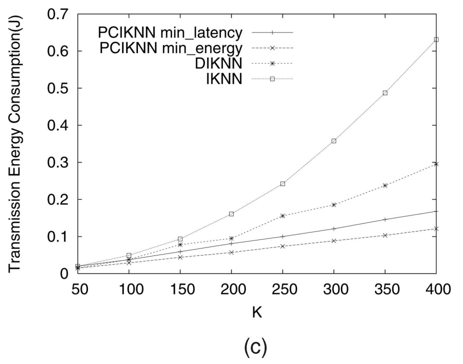
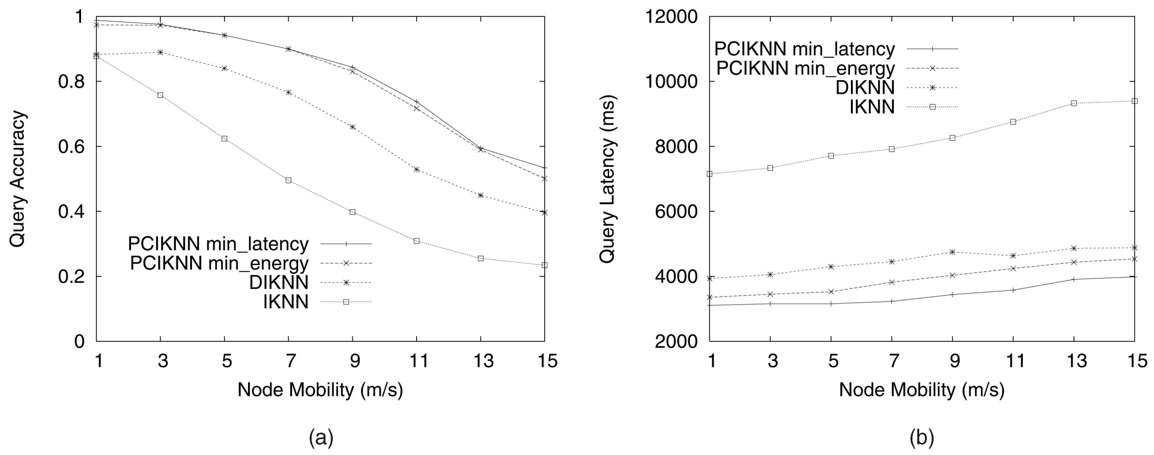


Fig. 16. Impact of K. (a) Query accuracy. (b) Query latency. (c) Energy consumption.



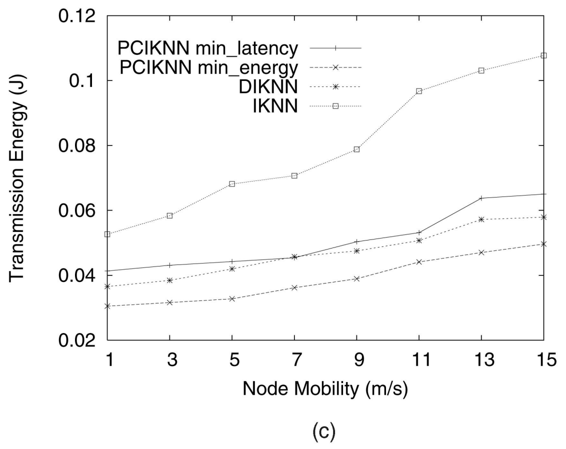


Fig. 17. Impact of node mobility. (a) Query accuracy. (b) Query latency. (c) Energy consumption.

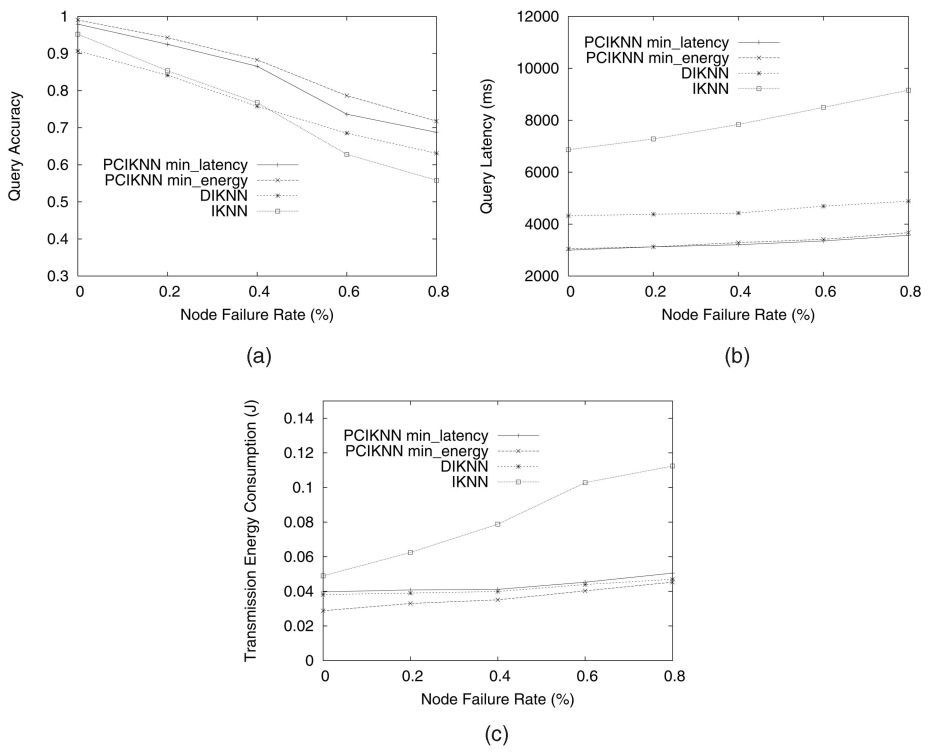


Fig. 18. Impact of node failure. (a) Query accuracy. (b) Query latency. (c) Energy consumption.

accuracy is likely to decrease. The reasons are 1) the packet loss of messages, including queries and partial results increases, 2) the movements of sensor nodes influence the KNN query result. Among the three algorithms, PCIKNN has the best query accuracy. Equipped with high parallelism in itinerary design, PCIKNN has the smallest query latency. Hence, PCIKNN quickly captures the snapshot of sensor nodes, leading to a better query accuracy. Fig. 17b demonstrates the query latency under the three algorithms. With faster moving speeds, the packet loss problem is more serious, resulting in dropping of the KNN query. Thus, all of the three algorithms need to wait for a longer time for the query processing to complete due to the packet loss problem. The energy consumption of the three algorithms is shown in Fig. 17c. As can be seen in Fig. 17c, high moving speeds of mobile sensor nodes result in more packet loss, thereby increasing more energy consumption of sensor nodes for resending packets. Similarly, with a smaller length of itineraries in PCIKNN, PCIKNN with the min_energy mode incurs a much smaller energy consumption as well.

6.2.4 Impact of Node Failure

In this experiment, we investigate the impact of node failure to the performance of three algorithms. The node failure rate of sensors clearly affects the performance of KNN query processing. Once nodes are failed, message droppings occur, thereby affecting KNN query processing. For example, a Q-node failure may result in the drops of KNN query threads, decreasing the query accuracy. The node failure rate is varied from 0 to 0.8 percent. Performance study of these three algorithms is shown in Fig. 18. It can be seen in Fig. 18a that PCIKNN has the best query accuracy, which is still larger than 70 percent even when the node failure rate is high (i.e., 0.8 percent). As shown in Figs. 18b and 18c, the latency and energy consumption results of each algorithm tend to increase when the node failure rate increases. PCIKNN has a better performance since parallelizing itineraries leads to a number of concurrent KNN query threads and a smaller itinerary length for each KNN query thread. Consequently, the risk of message dropping is decreased, showing the strength of the parallelized itineraries in PCIKNN. In addition, the design of collecting partial results at the home node in PCIKNN increases the KNN query accuracy.

6.2.5 Impact of Link Failure

In this experiment, we examine the impact of link failures to the performance of IKNN, DIKNN, and PCIKNN. A link failure may occur due to the mobility of sensor nodes or environmental interferences in wireless sensor networks. Clearly, a link failure results in the packet losses in wireless sensor networks. Thus, we vary the packet loss rate, i.e., the probability of packet losses during wireless transmissions, to investigate the impact of link failures. With a higher packet loss rate, the links become more unstable, leading to more packet losses in transmissions. To deal with packet losses, a sensor node replies an ACK message upon reception of a message from another sensor node. If a sender does not receive an ACK message from the receiver after a predefined time-out period, the sender will resend the same message until the ACK message is received. We implement the above operation in IKNN, DIKNN, and PCIKNN for comparison. The query accuracy of the three algorithms is shown in Fig. 19a. As the packet loss rate increases, the query accuracy of all the algorithms tends to decrease. As shown, since PCIKNN collects partial results at the home node, it achieves a better query accuracy than IKNN and DIKNN. Furthermore, due to the multiple

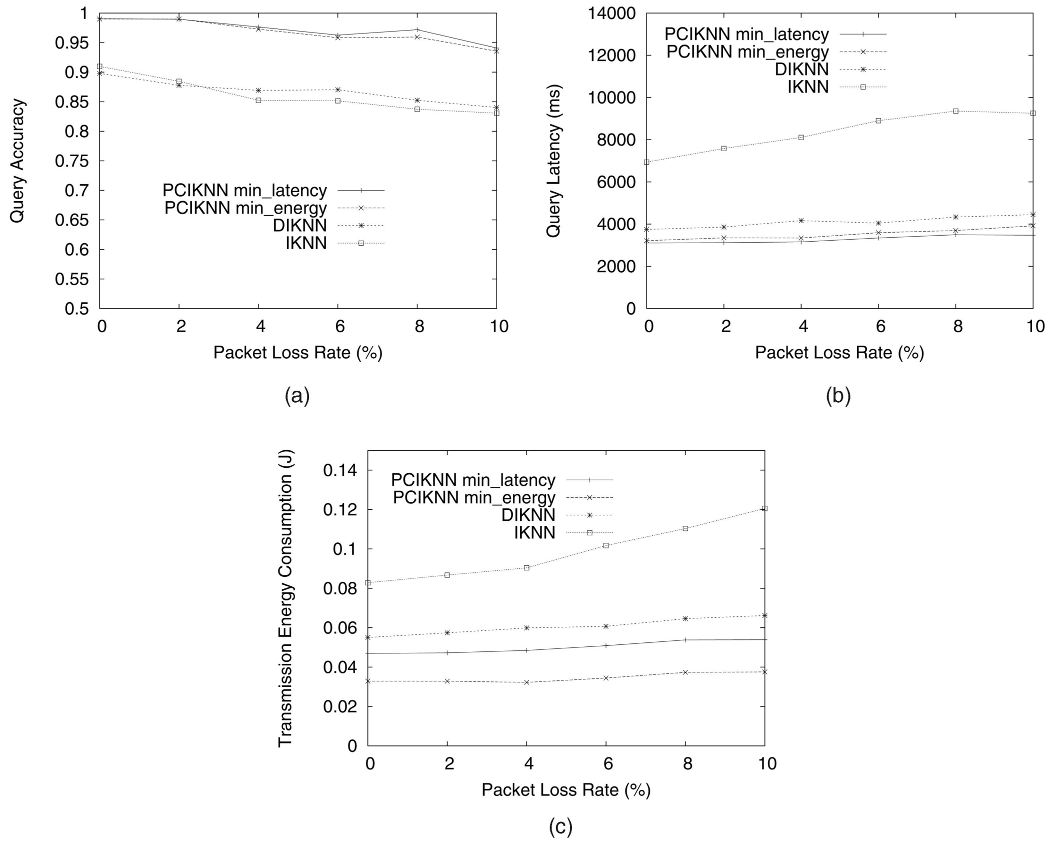


Fig. 19. Impact of link failure. (a) Query accuracy. (b) Query latency. (c) Energy consumption.

concurrent query threads in PCIKNN, the amount of data transmission in each itinerary of PCIKNN is smaller than that of IKNN and DIKNN. Thus, as shown in Fig. 19b, the query latency of PCIKNN is better. The energy consumption of the examined three algorithms is shown in Fig. 19c. PCIKNN with the minimum latency mode may have more KNN query threads, which increases the number of re-sent packets. On the other hand, PCIKNN with the minimum energy mode still has the best performance in terms of energy consumption.

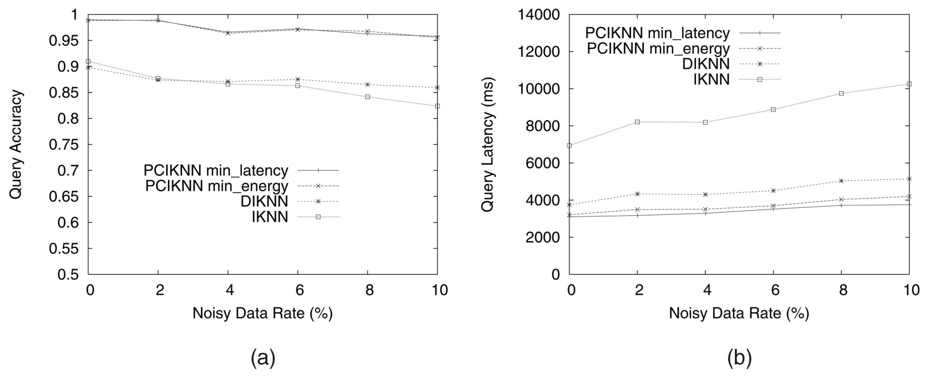
6.2.6 Impact of Noisy Data

In wireless sensor networks, some noisy data may occur in transmissions. When a noise occurs, the data are likely to contain some errors. To detect such errors occurred in wireless transmissions, one could utilize some existing error detection methods. In this paper, we utilize *Cyclic Redundancy Check* (CRC), a well-known error detection method in telecommunication, to detect possible errors occurred in transmissions. Each message includes a corresponding CRC code determined based on its message content. Upon receiving a message, a sensor node will check the CRC code to verify the correctness of messages received. If this sensor node finds an error by the evaluation of CRC codes, a retransmission is required. To simulate the noisy environment, a noisy data rate, which is the probability of having noisy data during message transmissions, is varied to study its impact on IKNN, DIKNN, and PCIKNN. With a higher noisy data rate, messages are likely to have more noisy data during message transmissions. The accuracy result of the examined algorithms is shown in Fig. 20a. PCIKNN has the best accuracy in both modes due to the design of collecting partial results at the home node. The latency and energy results are shown in Figs. 20b and 20c. It can be seen in Fig. 20b that the latency of IKNN, DIKNN, and PCIKNN is increased. This is due to the retransmissions incurred in a noisy environment. Furthermore, the energy consumption of all the three algorithms is slightly increased along with the increased noisy data rate. Clearly, with a higher noisy data rate, the more message transmissions are incurred.

6.2.7 KNN Boundary Estimation and Maintenances

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., $100\ \mathrm{m} \times 100\ \mathrm{m}$) are selected. The linear regression function of PCIKNN is set to $H(dist(N_i, N_{i+1})) = (-76.9166 \times dist(N_i, N_{i+1}) + 4{,}999.0903)$ by linear regression technique in [10]. The optimal KNN boundary is the average distance of the Kth distant nodes of all queries derived by the experiments. As shown in Fig. 21, PCIKNN is very close to the optimal KNN boundary under various network density. However, the boundary estimated in DIKNN does not fit well with the trend of the optimal KNN boundary. Moreover, in Fig. 21, 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.

In addition, we further evaluate the performance of the two schemes we proposed for adjusting the KNN boundary



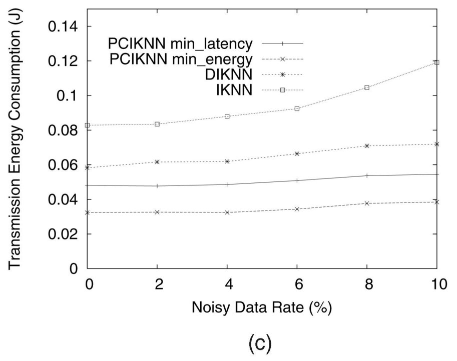


Fig. 20. Impact of noisy data. (a) Query accuracy. (b) Query latency. (c) Energy consumption.

in PCIKNN. We set the value of K to be 300, and the result is shown in Fig. 22. It can be seen in Fig. 22 that, when the value of K increases, the query accuracy is worsen if the KNN boundary is not adjusted by the home node. This is because whenK increases, the gap between the real KNN boundary and the estimated boundary increases, and thus, affecting the accuracy of KNN query significantly.

6.2.8 Rotated Itinerary Structures

As mentioned before, when K is larger, the energy of sensor nodes along return-segments is likely to drain out. To deal with this problem, we propose rotated itineraries. In this experiment, we evaluate this proposal by varying the value of K from 50 to 400. As shown in Fig. 23a, PCIKNN with rotated itineraries has a high query accuracy. Fig. 23b depicts the query latency under PCIKNN and PCIKNN with rotated itineraries. As shown, the latency of PCIKNN with rotated itineraries slightly increases because the length of branch-segments is increased. It can be seen in Fig. 23c that the total energy consumption of PCIKNN and PCIKNN with rotated itineraries is almost the same. Thus, both PCIKNN and PCIKNN with rotated itineraries consume a similar amount of energy since all sensor nodes within the same KNN boundary are involved. However, from the standard deviation of energy consumption shown in Fig. 23d, it can be seen that PCIKNN with the rotated itineraries has a more balance energy consumption distribution in both the minimum latency and the minimum energy mode.

7 Conclusion

In this paper, we propose an efficient itinerary-based KNN algorithm, namely *PCIKNN*, for KNN query processing in a

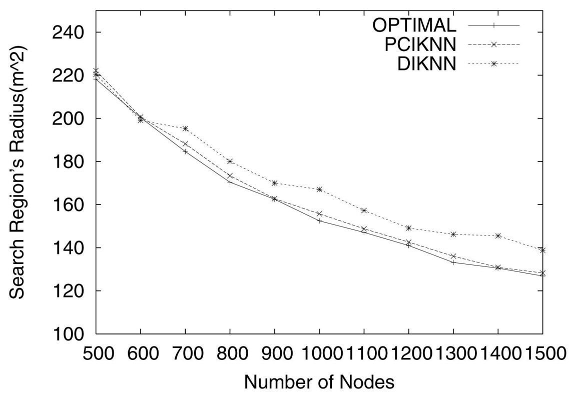


Fig. 21. KNN boundary estimation of DIKNN and PCIKNN.

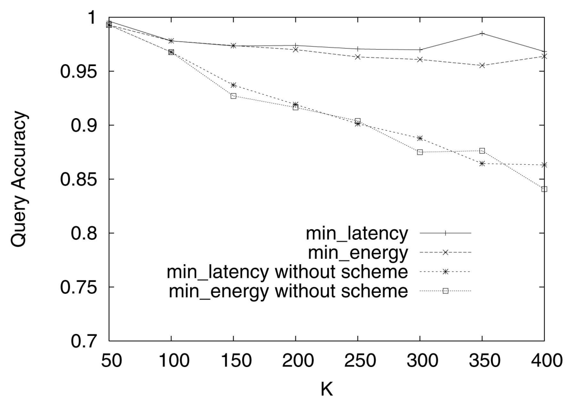


Fig. 22. Performance of KNN boundary adjustments in PCIKNN.

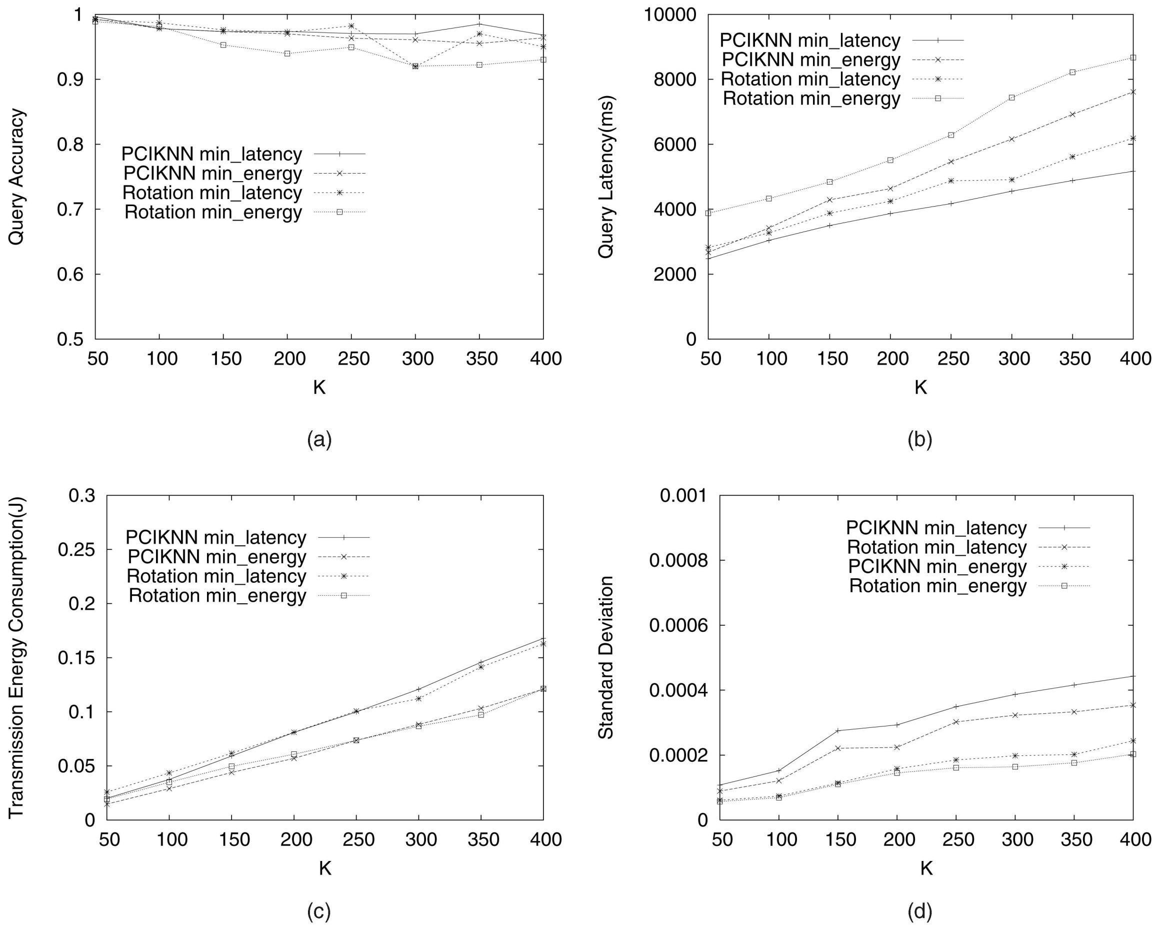


Fig. 23. Performance of PCIKNN with rotated itineraries. (a) Query accuracy. (b) Query latency. (c) Energy consumption. (d) Energy consumption standard deviation.

wireless sensor network. The basic idea of PCIKNN is to disseminate a KNN query and collect data along predesigned itineraries with "high parallelism." We derive analytical models in terms of the query latency and the energy consumption for PCIKNN. Accordingly, PCIKNN is able to determine the number of sectors for a given KNN query. To deal with the spatial irregularity, we developed methods to bypass voids and to dynamically adjust the KNN boundary. In addition, via linear regression, PCIKNN obtains an estimated KNN boundary very close to the optimal. Due to the design of returning all partial results to the home node in PCIKNN, we further proposed a rotated itinerary structure for PCIKNN to alleviate the energy exhaustion issue along the return-segments. Extensive experiments have been conducted and impact analysis on several important factors, including network density, node failure, and mobility of sensors, is conducted. Experimental results show that PCIKNN significantly outperforms the existing itinerary-based KNN query processing techniques in terms of energy consumption, query latency, query accuracy, and scalability.

ACKNOWLEDGMENTS

Wen-Chih Peng was supported in part by the MoE ATU Plan, the National Science Council under grant nos. 97-2221-E-009-053-MY3 and 95-2221-E-009-061-MY3, Taiwan, and Microsoft Research Asia. Wang-Chien Lee was supported in part by the US National Science Foundation under grant nos. CNS-0626709 and IIS-0534343.

REFERENCES

[1] M. Demirbas and H. Ferhatosmanoglu, "Peer-to-Peer Spatial Queries in Sensor Networks," *Proc. Third Int'l Conf. Peer-to-Peer Computing*, pp. 32-39, 2003.

[2] D. Estrin, R. Govindan, and J. Heidemann, "Next Century Challenges: Scalable Coordination in Sensor Networks," *Proc. ACM/IEEE MobiCom*, pp. 263-270, 1999.

[3] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A.E. Abbadi, "Approximate Nearest Neighbor Searching in Multimedia Databases," *Proc. 17th IEEE Int'l Conf. Data Eng. (ICDE)*, pp. 503-511, 2001.

[4] D. Goldin, M. Song, A. Kutlu, H. Gao, and H. Dave, "Georouting and Delta-Gathering: Efficient Data Propagation Techniques for Geosensor Networks," *Proc. NSF Worshop GeoSensor Networks*, 2003.

[5] W.R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks," *Proc. 33rd Hawaii Int'l Conf. System Sciences*, pp. 8020-8029, 2000.

[6] G.R. Hjaltason and H. Samet, "Distance Browsing in Spatial Databases," *ACM Trans. Database Systems*, vol. 24, no. 2, pp. 265-318, 1999.

[7] B. Karp and T.H. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," *Proc. ACM/IEEE MobiCom*, pp. 243-254, 2000.

[8] Y.B. Ko and N.H. Vaidya, "Location-Aided Routing (LAR) in Mobile Ad Hoc Networks," *Wireless Networks*, vol. 6, no. 4, pp. 307-321, 2000.

[9] F. Kuhn, R. Wattenhofer, and A. Zollinger, "Worst-Case Optimal and Average-Case Efficient Geometric Ad-Hoc Routing," *Proc. ACM MobiHoc,* pp. 267-278, 2003.
[10] S.J. Leon, *Linear Algebra with Applications.* Prentice Hall, 2002.
[11] D. Li, K. Wong, Y. Hu, and A. Sayeed, "Detection, Classification and Tracking of Targets in Distributed Sensor Networks," *IEEE Signal Processing Magazine,* vol. 19, no. 2, pp. 17-29, Mar. 2002.
[12] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong, "TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks," *Proc. Fifth Symp. Operating System Design and Implementation (OSDI '02),* pp. 131-146, 2002.
[13] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong, "The Design of an Acquisitional Query Processor for Sensor Networks," *Proc. ACM SIGMOD,* pp. 491-502, 2003.
[14] N. Patwari, A.M. Perkins, III, N. Correal, and R. O'Dea, "Relative Location Estimation in Wireless Sensor Networks," *IEEE Trans. Signal Processing,* vol. 51, no. 8, pp. 2137-2148, Aug. 2003.
[15] H. Qi, X. Wang, S. Iyengar, and K. Chakrabarty, "High Performance Sensor Integration in Distributed Sensor Networks Using Mobile Agents," *Int'l J. High Performance Computer Applications,* vol. 16, no. 3, pp. 325-335, 2002.
[16] N. Roussopoulos, S. Keeley, and F. Vicent, "Nearest Neighbor Queries," *Proc. ACM SIGMOD,* pp. 71-79, 1995.
[17] H. Schwetman, *CSIM User's Guide (Version 18),* Mesquite Software, Inc., http://www.mesquite.com, 2009.
[18] T. Seidl and H. Kriegel, "Optimal Multi-Step K-Nearest Neighbor Search," *Proc. ACM SIGMOD,* pp. 154-165, 1998.
[19] Z. Song and N. Roussopoulos, "K-Nearest Neighbor Search for Moving Query Point," *Proc. Seventh Int'l Symp. Spatial and Temporal Databases (SSTD),* pp. 79-96, 2001.
[20] J. Winter and W.-C. Lee, "KPT: A Dynamic KNN Query Processing Algorithm for Location-Aware Sensor Networks," *Proc. First Int'l Workshop Data Management for Sensor Networks (DMSN),* pp. 119-125, 2004.
[21] J. Winter, Y. Xu, and W.-C. Lee, "Energy Efficient Processing of K Nearest Neighbor Queries in Location-Aware Sensor Networks," *Proc. Second Int'l Conf. Mobile and Ubiquitous Systems: Networks and Services (MobiQuitous),* pp. 281-292, 2005.
[22] S.H. Wu, K.T. Chuang, C.M. Chen, and M.S. Chen, "DIKNN: An Itinerary-Based KNN Query Processing Algorithm for Mobile Sensor Networks," *Proc. 23rd IEEE Int'l Conf. Data Eng. (ICDE),* pp. 456-457, 2007.
[23] S.H. Wu, K.T. Chuang, C.M. Chen, and M.S. Chen, "Toward the Optimal Itinerary-Based KNN Query Processing in Mobile Sensor Network," *IEEE Trans. Knowledge and Data Eng.,* vol. 20, no. 12, pp. 1655-1668, Dec. 2008.
[24] 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,* vol. 87, no. 12, pp. 2861-2881, 2007.
[25] Y. Xu, W.-C. Lee, J. Xu, and G. Mitchell, "PSGR: Priority-Based Stateless Geo-Routing in Wireless Sensor Networks," *Proc. Second IEEE Int'l Mobile Ad Hoc and Sensor Systems Conf. (MASS),* pp. 7-10, 2005.
[26] Y. Xu, W.-C. Lee, J. Xu, and G. Mitchell, "Processing Window Queries in Wireless Sensor Networks," *Proc. 22nd IEEE Int'l Conf. Data Eng. (ICDE),* pp. 70-80, 2006.

Tao-Yang Fu received the BS and MS degrees from the National Chiao Tung University, Taiwan, in 2006 and 2008, respectively. Currently, he is a researcher in the Industrial Technology Research Institute, Taiwan. His research interests include mobile computing, network data management, and data mining. He is a student member of the IEEE Computer Society.

Wen-Chih Peng received the BS and MS degrees from the National Chiao Tung University, Taiwan, in 1995 and 1997, respectively, and the PhD degree in electrical engineering from the National Taiwan University, ROC, in 2001. Currently, he is an assistant professor in the Department of Computer Science, National Chiao Tung University, Taiwan. Prior to joining the Department of Computer Science and Information Engineering at the National Chiao Tung University, he was mainly involved in the projects related to mobile computing, data broadcasting, and network data management. He serves as PC members in several prestigious conferences, such as the IEEE International Conference on Data Engineering (ICDE), Pacific Asia Knowledge Discovering and Mining (PAKDD), and Mobile Data Management (MDM). His research interests include mobile computing, network data management, and data mining. He is a member of the IEEE.

Wang-Chien Lee received the BS degree from the Information Science Department, National Chiao Tung University, Taiwan, the MS degree from the Computer Science Department, Indiana University, and the PhD degree from the Computer and Information Science Department, the Ohio State University. He is an associate professor of computer science and engineering at the Pennsylvania State University (Penn State). Prior to joining the Penn State, he was a principal member of the technical staff at Verizon/GTE Laboratories, Inc. He leads the Pervasive Data Access (PDA) Research Group at Penn State to pursue cross-area research in database systems, pervasive/mobile computing, and networking. He is particularly interested in developing data management techniques (including accessing, indexing, caching, aggregation, dissemination, and query processing) for supporting complex queries in a wide spectrum of networking and mobile environments such as peer-to-peer networks, mobile ad hoc networks, wireless sensor networks, and wireless broadcast systems. Meanwhile, he has worked on XML, security, information integration/retrieval, and object-oriented databases. He has published more than 160 technical papers on these topics. His research has been supported by multiple US National Science Foundation (NSF) grants. Most of his research result has been published in prestigious journals and conferences in the fields of databases, mobile computing, and networking. He has been active in various IEEE/ACM conferences and has given tutorials for many major conferences. He was the founding program cochair of the International Conference on Mobile Data Management (MDM). He has also served as a guest editor for several journal special issues (e.g., the *IEEE Transactions on Computers*) on mobile database-related topics. He has served as the TPC chair or general chair for a number of conferences, including the Second International Conference on Scalable Information Systems (Infoscale '07), the Sixth International ACM Workshop on Data Engineering for Wireless and Mobile Access (MobiDE '07), the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC '08). He is currently serving as the TPC chair for the 10th International Conference on Mobile Data Management (MDM '09). He is a member of the IEEE and the IEEE Computer Society.