Available online at www.sciencedirect.com
ScienceDirect
Signal Processing 87 (2007) 2861–2881
SIGNAL PROCESSING
www.elsevier.com/locate/sigpro

# Processing $k$ nearest neighbor queries in location-aware sensor networks

Yingqi Xu[a,*], Tao-Yang Fu[b], Wang-Chien Lee[a], Julian Winter[a]

[a]*Pennsylvania State University, USA*
[b]*National Chiao Tung University, Taiwan*

Received 10 October 2006; received in revised form 25 February 2007; accepted 15 May 2007
Available online 25 May 2007

**Abstract**

Efficient search for $k$ nearest neighbors to a given location point (called a *KNN* query) is an important problem arising in a variety of sensor network applications. In this paper, we investigate in-network query processing strategies under a KNN query processing framework in location-aware wireless sensor networks. A set of algorithms, namely the *geo-routing tree*, the *KNN boundary tree* and the itinerary-based KNN algorithms, are designed in accordance with the *global infrastructure-based*, *local infrastructure-based* and *infrastructure-free* strategies, respectively. They have distinctive performance characteristics and are desirable under different contexts. We evaluate the performance of these algorithms under several sensor network scenarios and application requirements, and identify the conditions under which the various approaches are preferable.


*Keywords:* Wireless sensor networks; $k$ Nearest neighbor (KNN) query; Network infrastructure

## 1. Introduction

Sensor networks are composed of a sheer number of distributed small sensing devices. The sensor networks, facilitating detailed measurements over a wide geographical area, allow an unprecedented level of interaction with physical environments. Since sensor nodes are widely distributed across the deployed field, it may not be easy for end-users to access them. Thus, considering the frugal energy budget and very limited capabilities at sensor nodes, scalable, efficient and robust query processing mechanisms for data collection and dissemination are mandatory to fully realize the potential of sensor networks.

Spatial queries that extract sensed data from specific locations are fundamental to many sensor network applications, as the data are often geographically distributed. *K nearest neighbor* (KNN) query, a classical spatial query, is particularly relevant to sensor networks. For instance, applications usually require readings from certain number of sensor nodes closest to the location of an event [1,2]. KNN queries provide a way to sample the environment around the location of an event by specifying a sample size ($k$) in close proximity. In this paper, a *KNN query* retrieves sensor readings from the $k$ sensor nodes nearest to the given query

*Corresponding author. Tel.: +1 814 5717044.
*E-mail addresses:* yixu@cse.psu.edu (Y. Xu), csiesheep.csie91@nctu.edu.tw (T.-Y. Fu), wlee@cse.psu.edu (W.-C. Lee), jwinter@cse.psu.edu (J. Winter).

0165-1684/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.sigpro.2007.05.013

point $q$.[1] We examine a set of alternative solutions for locating and querying these $k$ nearest sensor nodes. Generally speaking, the KNN problem can be defined follows:

**Definition** (*k nearest neighbor problem*). Given a set sensor nodes $M$ (where $|M| = N$) and a geographical location (denoted as a query point $q$), find a subset $M'$ of $k$ nodes ($M' \subseteq M$, $k \leqslant N$) such that $\forall n_1 \in M'$ and $\forall n_2 \in M - M'$, $\text{dist}(n_1, q) \leqslant \text{dist}(n_2, q)$.[2]

KNN queries have been extensively studied in traditional spatial databases [3–5]. Such centralized databases utilize indices in designing efficient KNN algorithms. However, the wireless sensor network environment raises several new research challenges: (1) sensor nodes operate on an extremely frugal energy budget. Traditional centralized query processing plans which require all sensor nodes to periodically report their readings to a central node (e.g. base station), could quickly drain the energy resources of sensor nodes; (2) sensor nodes have limited computation, storage and communication abilities. Complex centralized index structures are not applicable to sensor networks; (3) sensor networks are mainly characterized by a dynamic topology due to node failures (e.g. energy depletion), unattended and untethered operations, and mobility of sensor nodes [6,7]. Therefore, a desirable KNN algorithm for wireless sensor networks should be distributed and reliable, and be able to incorporate in-network processing techniques for communication reduction. More specifically, an efficient KNN query algorithm should (1) minimize the number of sensor nodes involved with query processing since the radio operations dominate the energy consumption of sensor nodes; (2) minimize the total amount of data transmitted; (3) balance the load and the responsibility of query processing to all involved sensor nodes.

Before designing algorithms to achieve the above goals, we consider alternative query processing strategies, namely *global infrastructure-based*, *local infrastructure-based* and *infrastructure-free* query processing strategies. These strategies offer very different performance characteristics. Which one is appropriate for a particular network setting depends on the nature of the sensor network and its use. Consequently, our paper begins with a discussion of these query processing strategies for sensor networks. Correspondingly, we propose one KNN algorithm for each strategy. Each KNN algorithm is designed to achieve energy efficiency, scalability and robustness based on the underlying query processing strategy. This work does not claim a certain algorithm to be the best choice. In stead, it provides a guidance to choose preferable strategy and algorithm under various conditions. In fact, we argue that a sensor network system should embody all three KNN query processing algorithms and choose an appropriate strategy based on the given task and network conditions. The three KNN query processing algorithms proposed are *geo-routing tree* (GRT), *KNN bounded tree* (KBT), and *itinerary-based KNN* (IKNN) query processing algorithms.[3]

- GRT, inspired by R-tree [8], is a long-running, global tree infrastructure that capitalizes on the spatial proximity of sensor nodes. A KNN query propagates along the GRT, and prunes the nodes that are definitely not in the answer set. GRT involves unnecessary node visits in query processing.
- KBT is a short-lived and local tree infrastructure. To extract data from nodes of interest, a *search region* which is as small as possible while still large enough to contain the KNNs for the given query point is estimated. Inside the estimated region, a KBT is constructed, along which the query propagation and data collection are conducted.
- IKNN is an infrastructure-free approach, which processes a query along a pre-designed itinerary. Facilitated by the itinerary, IKNN stops query processing automatically soon after collecting data from the $k$ nearest nodes, which minimizes the number of nodes involved into query processing and significantly improves the algorithm efficiency.

To our best knowledge, it is the first work investigating KNN query for wireless sensor networks.[4] We evaluate the performance of the three KNN query processing algorithms with an extensive

[1] In this work, we only consider order-insensitive KNN query. We will study order-sensitive KNN query in our future work.

[2] Function $\text{dist}(\cdot,\cdot)$ denotes the Euclidean distance between two geographical locations.

[3] Without causing confusion, we use GRT and KBT to denote both the KNN query processing algorithms and the network infrastructures they are based on.

[4] The preliminary result of this work was reported in [9,10].

simulation study. Various network conditions and application requirements are examined.

The rest of the paper is organized as follows. In Section 2, we describe our assumptions as well as examine related work. Sections 3–5 present the detailed designs for the GRT, KBT and IKNN algorithms, respectively. The performance of these three approaches under different circumstances is examined in Section 6. Finally, Section 7 gives the conclusive remark and discusses the future work.

## 2. Preliminaries

In this section we first state our basic assumptions about the sensor network system under consideration. Then we give a brief review of closely related work, which ends with a discussion of fundamental query processing strategies for wireless sensor networks.

### 2.1. Assumptions

We assume that a wireless network consisting of a number of sensor nodes is deployed in a two-dimensional fixed area. Each sensor node is location-aware, i.e., each sensor node is able to obtain its location information through an equipped GPS processor [11] or some localization techniques [12]. The sensor network topology may dynamically change during operation. The dynamics of a sensor network may be introduced by (1) node mobility (sensor nodes may be mobile); (2) energy conservation sensor nodes may periodically switch to sleep mode [13–15]; (3) unreliable links and node failures.

Following the definition in Section 1, we consider KNN queries specified by: a query point $q$ specified by its geographical location and an integer value $k$ which denotes the number of sensor nodes closest to the query point need to be queried. A KNN query can be issued by any sensor node instead of by one or more stationary access points in the networks. The application expects the query results to return to the same sensor node where the query was issued. Finally, we assume that sensed data are stored locally on the sensor nodes which implies that in order to extract the data from the $k$ nearest sensor nodes, a KNN query processing algorithm has to *locate* and *retrieve* data reports from those nodes.

### 2.2. Related work

While KNN queries have not been well explored in wireless sensor networks, our work has been inspired by a wide range of research efforts on distributed sensor networks.

Traditionally, KNN query are answered by assistance of the indices on the data. The most widely used algorithm is the branch-and-bound algorithm based on R-trees, which traverses the R-tree while maintaining a list of $k$ potential nearest neighbors in a priority queue [3,8]. Our GRT algorithm adopts such an algorithm and tailored it for wireless sensor networks. There are attempts to use range queries to solve the $k$-NN search problem, such as the one proposed in [16]. The basic idea is to first find a region that guarantees to contain all $k$-NNs, and then use a range query to retrieve the potential $k$-NNs. This algorithm is further optimized by [4,17]. Our KBT shares similar design philosophy. KNN query for moving objects can be answered by a time-parameterized R-tree (TPR-tree) [18] or its variants [19]. Designed to support predictive queries, the TPR-tree extends R-tree by including the velocity vectors. The $k$-NN queries are *time*-parameterized by a time interval and can be answered by a depth-first traversal of the TPR-tree.

The above centralized algorithms focus on improving the disk access performance, and thus are not suitable for resource-constrained wireless sensor networks. Increasing number of research efforts have been made to support various sensor network applications. The proposal of constructing and maintaining a network-spanning infrastructure within a sensor network is one of them. To processing spatial query, the query is propagated toward geographical location(s) specified by the application along the infrastructure. Once the query reaches all the sensor nodes of interest, data collection takes place along the infrastructure. The sensor nodes of interest recursively report their readings to the nodes from which they received the query, such that partial results are returned level-by-level up the infrastructure until reaching the root node. An example of such network-spanning infrastructure is routing tree studied in [20,21]. Clustering is an alternative network infrastructure where a cluster head is responsible for the operations of the sensor nodes in its cluster [22,23]. With small-scale centralized control from the cluster heads, the cluster infrastructure is suitable for achieving cooperation among sensor nodes. Another well

known approach called directed diffusion represents an alternative solution for propagating queries and collecting results. Query is flooded into the network on-demand [24]. A few routes are selected adaptively by reinforcement procedure based on the route quality (e.g., latency) and maintained during the lifetime of the query. However, without considering the geographical nature of sensor networks, this end-to-end route is not beneficial for spatial queries.

Researchers also demonstrated traditional index structures (e.g., R-tree) to be useful in sensor network design. For instance, Demirbas and Ferhatosmanoglu [25] employ an R-tree as a network infrastructure and discuss various query processing mechanisms upon the modified R-tree. However their work lacks design details that are necessary to handle the unique spatial properties and limitations of sensor networks. The Geo-routing tree adopted by GRT algorithm was discussed in [26] originally. We will examine its design details shortly in Section 3.1.

In addition, recent work has pointed out that location-awareness of sensor nodes can facilitate the design of energy-efficient and robust sensor network paradigms and protocols. For instance, Geo-routing protocols [27–30] have been adopted by many research works (e.g [31–33]), due to their efficiency. Geo-routing assumes that the location information of routing destination is known. Routing decisions at intermediate relay nodes are made based on the location information of neighbor nodes and of the destination. For example, the neighbor node that is geographically closest to the destination may be chosen as the next relay node. However, as pointed out by [32], geo-routing is not efficient for query propagation/ data collection among a set of sensor nodes. The detailed design of GPSR, a representative geo-routing protocol for wireless ad hoc and sensor networks will be examined shortly in Section 4.1.

Window query, another classical spatial query, also attracts increasing research attentions. GEAR [32] considers the issue of query diffusion for full window coverage. Even though GEAR does not require any network infrastructure for spreading the query, it does not consider data collection procedure without infrastructure support. Our preliminary work considers dynamic sensor network and studies a complete query processing plan, but lacks of technical details [7]. IWQE, an original itinerary-based window query processing algorithm, effectively integrates the query propagation and data collection into one stage without requirement of network infrastructure [33]. Our proposal of IKNN query processing algorithm follows the design principle of IWQE, which will be introduced in detail in Section 5.1.

### 2.3. Query processing strategies for wireless sensor networks

Next, we classify the various algorithms discussed above based on their fundamental design strategies, and analyze their performance characteristics.

#### 2.3.1. Global infrastructure-based strategy

A global infrastructure covers the entire sensor network field and involves all sensor nodes. To build such an infrastructure, a node is selected as the root of the infrastructure. The root node is usually the point where users interacts with the network. The root initiates an infrastructure construction by broadcasting a message that specifies its own id and other attributes (e.g., location and its level in the infrastructure). Upon receiving the message, a node joins the infrastructure and rebroadcasts the message with its own information. This process repeats until all nodes join the infrastructure. Due to the wireless broadcast medium, a node may receive duplicate message. Based on the desired infrastructure property, the node chooses one of the senders as its higher-level node and determines its own level in the infrastructure. To maintain a global infrastructure, the root node periodically broadcasts the message down to the infrastructure, so that the process of topology discovery goes on continuously. An alternative approach is to let all sensor nodes periodically notify other nodes its existence and other features (e.g., locations). The constant topology maintenance makes it relatively easy to adapt to network changes.

A global infrastructure facilitates one-to-many data communication, since the infrastructure converges at the root node. Query processing along a global infrastructure is performed in two stages. At the first stage of query propagation, the root node floods the query down to the infrastructure. Once the query reaches all sensor nodes that possess the data of interest (may not be all sensor nodes), data collection takes place. At data collection stage, the sensor readings are continually rounded up from lower-level nodes to higher-level nodes. To perform in-network data aggregation, higher-level nodes do

not propagate their query results until they receive from all their children. As such, the sensor readings are aggregated at each level.

Due to its large scale, global infrastructure incurs noticeable construction and maintenance cost. When a query only interests in part of the network with a small number of sensor nodes, the global infrastructure may not provide the best support since query processing may incur unnecessary node visits. Moreover, data aggregation, which happens at every hop on the way back to the sink node, could be inefficient when the data of interest is clustered.

#### 2.3.2. Local infrastructure-based strategy

Local infrastructure-based strategy is similar to the global one, as both process a query via an infrastructure. The difference is that the construction of a local infrastructure is usually query-driven. In other words, the local infrastructure is constructed based on the demand of queries. The location, scale, lifetime, and property of local infrastructure are customized based on the query content. To reduce its construction and maintenance cost, the local infrastructure should be as small as possible, but still large enough to cover all sensor nodes of interest. Local infrastructure-based strategy requires a node-to-node routing protocol to deliver the query message from the sink node to the root of the local infrastructure. For spatial queries, a geo-routing protocol is sufficient and desirable to delivery the query to a specified location. After the query reaches the root node, a local infrastructure is constructed along query processing. Data collection based on local infrastructure is conducted in a similar manner as based on global infrastructure. With a smaller scale and shorter lifetime, local infrastructure is more flexible and efficient than the global infrastructure. However, the flexibility of a local infrastructure raises design challenges in determining its properties (e.g., infrastructure size) for processing KNN query. We will discuss this research challenge in detail shortly in Section 4.

#### 2.3.3. Infrastructure-free strategy

Even though infrastructure-based strategies (both global and local ones) have some drawbacks, they are widely adopted by sensor networks. This is because they support in-network processing techniques, which by processing partial query results within the network, greatly reduce the total amount of network communication and energy usage, thus crucial for energy-constrained sensor networks. However, the gain achieved by in-network processing techniques is weakened by the overhead incurred for building and maintaining such infrastructures and by unnecessary node visits along these infrastructures. Therefore, infrastructure-free strategy is considered to overcome the above problems.

Flooding is a naive infrastructure-free strategy, in which the query is flooded into the network. Nodes of interest upon receiving the query send their responses directly back to the sink node. Even though removing the requirement of an infrastructure, the flooding scheme does not support in-network processing, thus incurring a significant amount of data communication. The flooding scheme reveals a big challenge faced by infrastructure-free approach, i.e., how to incorporate in-network processing techniques into query processing without the support of network infrastructure? More specifically, without an infrastructure, where and when data aggregation should take place and what nodes should take this responsibility?

Itinerary-based query processing mechanism, studied in our prior work [33], is one solution to the above research challenge. With itinerary-based query processing mechanism, the query is propagated along an itinerary which is pre-designed to ensure all nodes of interest be queried in an efficient manner. Query propagation and data collection take place concurringly at some special nodes residing on the itinerary. An example of itinerary-based algorithm developed for window query processing [33] is presented in Section 5.1.

As shown in Fig. 1, we envision the above three different query processing strategies residing at a middle layer between a sensor network and application requests (e.g., data dissemination, data collection, and event detection and monitoring). Such a middle layer interacts with location service and application layer to decide the right query processing strategy. In the following, we focus on KNN query processing and develop three KNN query processing algorithms in accordance with the global infrastructure-based, local infrastructure-based and infrastructure-free strategies, respectively.

## 3. KNN algorithm over geo-routing tree

In this section, we present our first KNN query processing algorithm based on a geo-routing tree

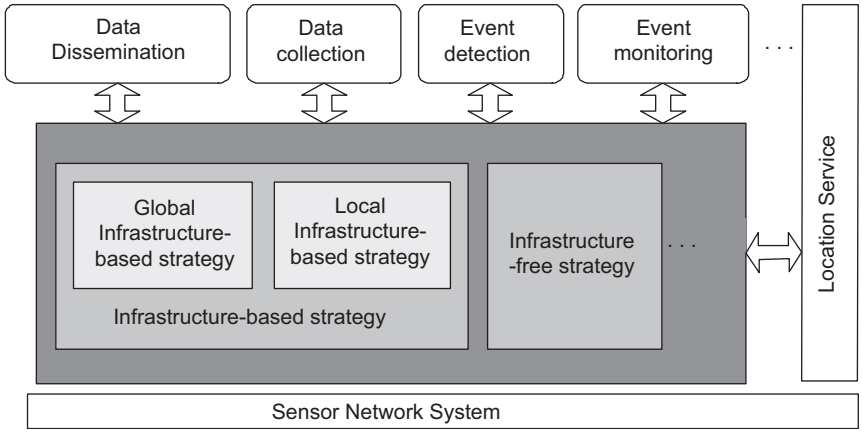


Fig. 1. Query processing framework.

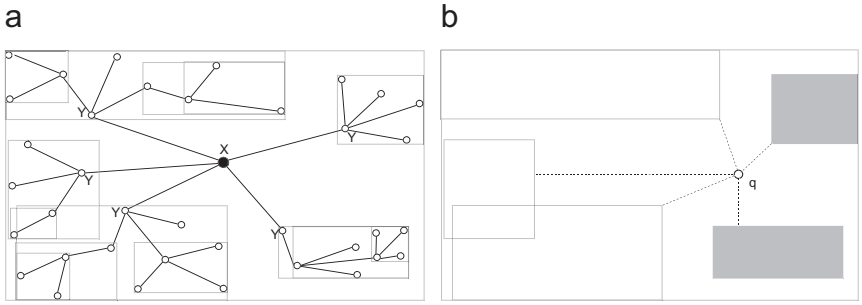


Fig. 2. Minimum bounding rectangle (MBR): (a) MBRs in a geo-routing tree; (b) MINDIST.

(GRT), which spans the entire network and remains alive during the network lifetime. For the sake of presentation, we assume queries are always inserted from the root node.[5]

### 3.1. Geo-routing tree

The GRT, a tree infrastructure similar to R-tree, was proposed for executing queries on sensor networks in [26]. In the GRT (shown in Fig. 2(a)), each sensor node $X$ maintains a minimum bounding rectangle (MBR) for its child $Y$. The MBR for node $Y$ is the smallest rectangle which encloses the geo-locations of all the sensor nodes in the tree rooted at $Y$ (including $Y$). MBRs are constructed from the leaf nodes up and MBRs for leaf nodes are geographical points. GRT captures the spatial properties of sensor nodes and does not require the minimum and maximum number nodes in one MBR.

To adapt to network topology changes, we adopt a simple beacon algorithm for GRT maintenance [27]. Periodically, each sensor node broadcasts a beacon message that includes its identifier, position and MBR. The parent $X$ recalculates $Y$'s MBR after receiving its child $Y$'s beacon. If the changes on $Y$'s MBR affects the correctness of $X$' MBR, $X$ has to update its MBR and report the update to its parent as well. If a parent node does not hear from a child after a period of time the parent deletes this child and updates the MBR. On the other hand, if a child does not hear from its parent, it selects a new parent from its neighbor table and informs the new parent which involves recomputing the MBRs as well.

Several data structures are required for GRT to process KNN query. For a given query point $q$ and a sensor node $\mathscr{S}$

- MINDIST($\mathscr{S}, q$) is the minimum distance from $q$ to $\mathscr{S}$'s MBR. The formal definition of MINDIST

[5]Queries inserted from non-root nodes have to be routed to the root node first. Other solutions, such as building multiple trees rooted at different sensor nodes or starting the GRT algorithm from any sensor nodes, are possible but outside the scope of this research.

is given by [3]. Fig. 2(b) shows an example of MINDIST denoted by dashed lines from $q$ to MBRs. Intuitively, MINDIST determines the minimum distance from $q$ to the MBR that encloses the subtree rooted at node $\mathscr{S}$.

- NTable($\mathscr{S}$) is the neighbor table for sensor node $\mathscr{S}$. A neighbor table stores the information about the nodes one-hop away from $\mathscr{S}$ including their *id* and their current geographical location. The size of the NTable is the total number of neighbors of $\mathscr{S}$ denoted by $m(\mathscr{S})$.
- KTable($\mathscr{S}, q$) is maintained by sensor node $\mathscr{S}$. From $\mathscr{S}$ and $\mathscr{S}$'s neighbor nodes, the $k$ closest sensor nodes to $q$ are selected. Their *id* and their distance to the query point (i.e., dist$(q, id)$ are stored in KTable($\mathscr{S}, q$). If $k > m(\mathscr{S})$, the remaining $(k - m(\mathscr{S}))$ entries have their *id* set as NULL and their dist$(q, id) = \infty$. The KTable is sorted based on dist$(q, id)$ in ascending order. For the simplicity of the presentation we denote $\text{dist}^i_{\mathscr{S}}(q, id)$ as the dist$(q, id)$ of $i$th entry in KTable($\mathscr{S}, q$).
- $D_{\max}$ records the maximum distance between the query point $q$ to sensor nodes from the current $k$ closest sensor nodes. $D_{\max}$ is disseminated along with the query and is updated as the query is propagated.

### 3.2. GRT algorithm

The KNN search algorithm over GRT works as follows. Once the root node receives a query from an application, it computes its MINDIST$(0, q)$, forms its KTable$(0, q)$, and initializes $D_{\max}$ as the $\text{dist}^k_0(q, id)$ which is the dist of the last entry in KTable$(0, q)$.[6] The root node then broadcasts the query along with the $D_{\max}$ value. Any child node $\mathscr{S}$ that receives the query computes its MINDIST($\mathscr{S}, q$) and compares MINDIST($\mathscr{S}, q$) with the received $D_{\max}$. If MINDIST($\mathscr{S}, q$) $> D_{\max}$ node $\mathscr{S}$ drops the query, since all sensors in $\mathscr{S}$'s subtree have a distance farther than $D_{\max}$ from the query point $q$. Otherwise, node $\mathscr{S}$ forms KTable($\mathscr{S}, q$) based on its NTable, sets $D_{\max} = \text{Min}\{D_{\max}, \text{dist}^k_{\mathscr{S}}(q, id)\}$, and broadcasts the query along with the new $D_{\max}$. The process is repeated until the query reaches the leaf nodes. Once the propagation procedure stops, the sensor node returns its KTable and query execution results to its parent. The parent at each level aggregates all the KTables received from its children and extracts the $k$ entries which have smallest dist$(q, id)$ as a new KTable as well as the query results from these $k$ nodes and reports them to its parent. This aggregation process is repeated until eventually the root receives results from all its children and forms a new KTable that contains the $k$ nearest sensor nodes to the query point along with their associated query results. Fig. 3 shows the network communication incurred by GRT algorithm. For the sake of presentation, MBRs are not drawn in Fig. 3. The lines and arrows represent the tree structure and communication incurred by query processing. The root node is bounded by a rectangle.

The above KNN algorithm over GRT, usually referred as the *branch-and-bound* technique [3], prunes the subtrees that definitely do not have nodes within $k$ closest nodes to the query point thereby reducing overall communication and conserving energy. However, GRT algorithm raises several performance issues. First, as a global infrastructure, GRT may incur excessive construction cost as well as maintenance cost when the network topology changes frequently. Second, the KNN algorithm over GRT requires substantial storage space since many data structures (e.g., parent table, children table, KTable and NTable) have to be maintained. This storage requirement would increase significantly with either increasing network density or the value of $k$.

## 4. KNN algorithm over KBT

We argue that the KNN search algorithm has to be (1) more distributed so that the responsibility of individual nodes is minimized, making the network less vulnerable to single node failure, and (2) more localized, so that the number of nodes involved in query processing is minimized. Meanwhile, we are aware of geographical routing algorithms that approach shortest-path routing that can reach any node in the network as long as its geographical location is known. These routing algorithms require a periodic beacon message to keep the list of neighbors of each node updated.

### 4.1. GPSR algorithm

KBT, our second KNN algorithm, is built upon GPSR [27], a geographical multi-hop routing algorithm. The rationale of this selection is explained shortly in Section 4.2. GPSR works in two

[6] Without loss of generality, we set the node ID for the root node to 0.

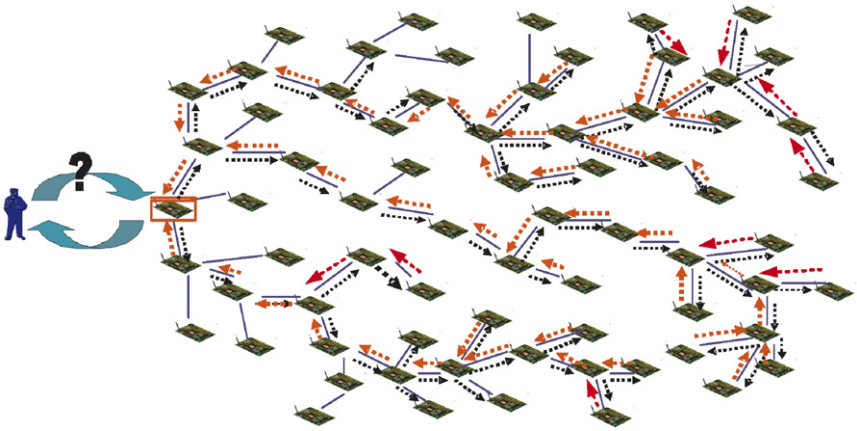


Fig. 3. GRT-based KNN query processing algorithm.

modes: *greedy* mode and *perimeter* mode. In greedy mode, the forwarding node forwards the message to the neighbor closest to the destination. If no such neighbor exists, the algorithm switches to the perimeter mode, which given a planarized graph of the network topology routes messages around voids in the network. GPSR returns to greedy mode from perimeter mode when the packet reaches a node closer to the destination than the node at which the packet entered the perimeter node. GPSR routes packets to the *nearest neighbor* (NN) of the destination which may not be associated with a specific sensor node. This property has been used in other studies; interested readers are referred to [34].

### 4.2. KBT algorithm

Our proposed KNN search algorithm over KBT is facilitated by the greedy perimeter stateless routing (GPSR) algorithm [27]. The basic idea of KBT is to find the KNN sensor nodes by searching an estimated region of the network. This region (called the *KNN search region*) is a circular region centered at the query point that is estimated to be as small as possible while still large enough to enclose the $k$ nearest sensor nodes. Clearly, the performance of the KBT algorithm heavily depends on the radius of the circular KNN search region which determines the number of nodes to be accessed for query execution. Additionally, the techniques that disseminate the query and collect data inside the KNN search region are also important for determining the query latency. In the following, we discuss the above issues in detail by breaking the KBT algorithm into several phases: (1) forwarding the query toward the query point and estimating the radius size of the KNN search region; (2) propagating the query inside the KNN search region and locating the $k$ nearest sensor nodes; (3) returning the query results to the application.

#### 4.2.1. Estimation of KNN search region

Upon receiving the KNN query from an application, the sensor node forward the query toward the *home node* (the nearest neighbor to the query point) using the GPSR routing algorithm. The query point is specified by the application and not necessarily associated with any sensor node. The home node is responsible for estimating the size of the KNN search region based on the information collected during the query propagation. In this paper, we investigate four approaches for estimating the radius of the KNN search region.

Our first approach, named SUMDIST, was previously reported in [9]. Briefly, this technique records the location information of $k$ relay nodes (i.e., nodes that forward the query toward the home node) that are closest to $q$ along the relay path. Upon receiving the query, the home node combines and sorts these $k$ relay nodes with its neighbor nodes based on the their distance to $q$. The radius of KNN search region is set as the distance of the $k$th nearest node to $q$.

The second approach, called MHD (named MHD-2 in [9]), avoids transmitting the coordinates of $k$ relay nodes toward the home node. Instead, only one maximum hop distance (MHD) value that is calculated as the maximum distance between two hops along the relay path is transmitted. Assuming the home node has $m$ neighbors and $m<k$, the KNN search region is estimated as a circle with

radius $(k-m) \times \text{MHD}$. Compared with SUMDIST, MHD dramatically reduces the transmission overhead for query propagation.

However the above two approaches are not expected to perform well when $k$ becomes large as the estimated region may quickly expand to the entire sensor network. Therefore, we propose a third approach called NeighborClass. First, the home node selects the members of its NeighborClass as the $\text{Min}\{m,k\}$ closest nodes to $q$ ($m$ denotes the number of neighbor nodes). The NeighborClass distance is computed as the furthest distance between the query point and any member of the NeighborClass. These nodes become a new NeighborClass and their IDs and the NeighborClass distance value are passed along with the query packet to the next relay node. Once a relay node receives the query packet, it again selects the $\text{Min}\{m,k\}$ closest nodes from its list of neighbors that do not belong to any NeighborClass in the query packet it received from the previous relay. These newly selected node IDs and the maximum distance become a new NeighborClass and are added to the query packet and transmitted to the next forwarding node. When the home node eventually receives the query packet, it includes its NeighborClass and then sets the radius of the KNN search region distance by iterating backward through the list of NeighborClasses and summing the number of members of each NeighborClass until the sum is larger than $k$ and the KNN search region radius is set as the NeighborClass distance of the last NeighborClass searched. By considering the neighbor nodes of all relay nodes, we expect the KNN search region estimated by NeighborClass to be much smaller than those obtained by SUMDIST and MHD and less affected by increasing $k$.

However NeighborClass still incurs a significant amount of traffic overhead along the query path since the IDs of the NeighborClass members are transmitted. Therefore, we developed a modified version of NeighborClass, called NeighborClass2 that avoids transmitting the IDs of NeighborClass members along the forwarding path. Each relay node (each having a NeighborClass) only passes the count of member nodes and the maximum distance to the query point $q$ from the furthest member node. Once the home node receives all the information, it orders those information based on the maximum distance and selects the smallest maximum distance which ensures that the total number of member nodes in the neighbor classes with shorter maximum distance than the selected maximum distance is larger or equal to $k$. This approach cannot guarantee that the radius of the KNN search region will encompass the KNN sensor nodes, but may serve as a good heuristic technique since sensors near the relay route may be double counted (i.e. sensors may be members of more than one NeighborClass). This double counting may be advantageous since it can represent sensors that are inside the KNN search region but are not members of any NeighborClass.

After the query packet has reached the candidate home node, it is transmitted around the perimeter of the query point in perimeter mode. This is necessary to verify the home node and we consider it part of the relay path in order to exploit the perimeter nodes and their neighbors for reducing the boundary radius. Fig. 4(a) depicts the routing phase of the KBT KNN algorithm.

#### 4.2.2. *Find $k-1$ nearest neighbors*

Given that the query has reached the home node, the next step is to find the remaining $k-1$ nearest neighbors (the home node is the nearest neighbor). Before we describe the details of the KBT algorithm, we first discuss a naive algorithm which simply floods within the KNN search region. Upon receiving the query, all sensor nodes inside the search region report their readings back to the home node using GPSR algorithm. Although simple, the flooding approach is expected to be efficient for small $k$ since the KNN search region is small. In this case, constructing an infrastructure for disseminating the query and collecting data inside the small KNN search region is unnecessary compared with the cost of flooding. Moreover, flooding reduces the dependence on any specific nodes inside the search region (other than the home node), thus is more tolerable to individual node failure which makes flooding likely to have good accuracy and robustness. A drawback of flooding is that the query results have to be returned back to the home node individually, thus losing the opportunity for aggregating the results at intermediate nodes.

To address the drawbacks of flooding, we proposed the KBT tree structure which provides the opportunity for local data aggregation. For naming purposes, we refer to this technique as the *single root* (SR) KBT technique since the tree is rooted at the home node. As the query is broadcasted, nodes select their parent based on their geographical proximity. After choosing a parent the child sets a timer based on the

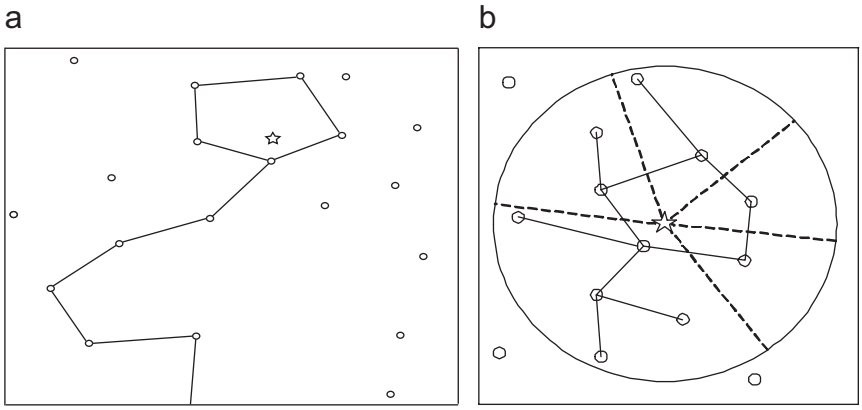


Fig. 4. KBT perimeter and perimeter tree: (a) KBT home node and perimeter; (b) KBT perimeter tree.

difference between an estimated value of the height of the tree and its level in the tree. Once the timer expires, the nodes aggregate results from their children and respond to their parents. The TreeHeight, the estimate of the height of the tree, is set before the query is issued. A counter, increased by one at every hop, is passed along with the query and recorded by the internal nodes as their level. The timer of the internal node is set as $2 \times \text{Max}\{1, \text{TreeHeight} - \text{level}\} \times \text{MessageDelay}$ where MessageDelay is the estimated message propagation delay between two neighbor nodes. The advantage of the tree approach is that results can be aggregated thereby reducing total amount of data transmitted. The drawback is that poorly set timers can either reduce the accuracy of the KNN results or unnecessarily increase the query latency.

A third approach is to use multiple trees rooted at perimeter nodes [27]. The perimeter nodes are determined by routing around the home node with GPSR perimeter mode. We refer to this technique as the *perimeter tree* (PT). The home node still serves as an overall root. The goal of this approach is to attempt to balance the tree to improve query accuracy since the timers are set based on a fixed estimate of the height of the tree. TreeHeight estimates close to the actual tree give better query accuracy. Furthermore, smaller TreeHeight estimates mean shorter latency since the timers expire sooner. When constructing the tree, the root transmits the query back around the perimeter. The circular region is divided into slices defined by the midpoints between the hops as shown in Fig. 4(b). This approach assigns more responsibilities to internal nodes but may be able to improve query accuracy for large values of $k$. This is because the height of the tree will be smaller than the SR tree since PT trees are more balanced. However, PT has more overhead than SR since an additional transmission around the perimeter is required and the midpoint data must be included in the broadcast query packet.

#### 4.2.3. Return results

After the home node has received the KNN results, they are aggregated into a single message and returned to the query point using GPSR. If a KBT tree has been constructed inside the KNN search region it is automatically dissolved after the query finishes. In other words, the lifetime of KBT is only as long as the KNN query takes to process. Fig. 5 shows an example of network communication in KBT algorithm. The bounded node acts as the root of the local infrastructure and the round circle depicts the estimated KNN search region. Comparing against Fig. 3, Fig. 5 shows that less number of nodes are involved into KBT algorithm.

## 5. KNN algorithm over itinerary

By constraining query processing within a KNN search region, KBT algorithm forms a smaller-scale infrastructure and is more efficient and scalable than GRT algorithm. However, without the knowledge about network topology, it is not easy for KBT algorithm to estimate the *right* size of a KNN search region, which has significant impact on its energy consumption and query accuracy. Another drawback of KBT algorithm is that when a query point is close to a home node, the estimation heuristics discussed in Section 4.2.1 may not be able

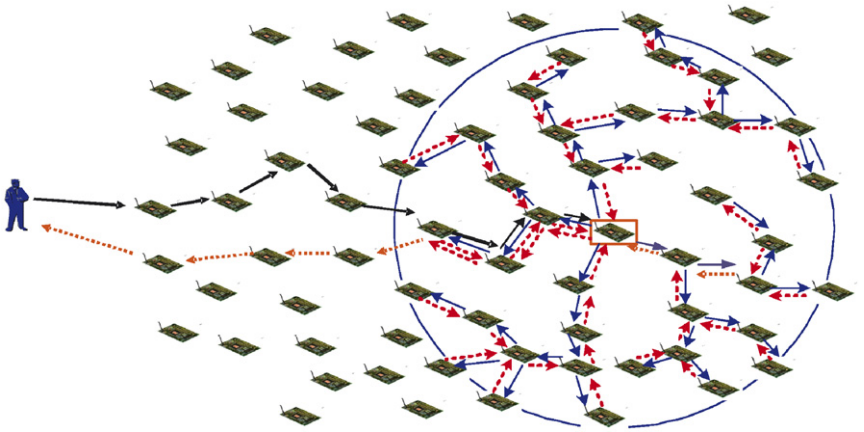

Fig. 5. KBT-based KNN query processing algorithm.

to gather enough information for estimating an appropriate KNN search region.

Motivated by the *itinerary-based window query processing* (*IWQE*) [33], we consider the infrastructure-free query processing strategy and develop the third KNN query processing algorithm, called IKNN query processing algorithm. Similar to IWQE, IKNN is executed along a pre-defined itinerary and integrates query processing and data collection into one stage. Thus, no infrastructure is required to facilitate query processing. More importantly, the query processing is able to stop very soon after collecting data from the KNN nodes, which minimizes the number of node visits. In the following, we first briefly introduce the basic idea of IWQE, which greatly inspires IKNN. Then we present IKNN algorithm.

### 5.1. IWQE algorithm

IWQE [33] is designed for processing spatial window queries for wireless sensor networks. Window query retrieves the sensed data from the sensor nodes falling within a query window (i.e., a spatial area of interest specified by the user). Distinguished from existing window query processing algorithms, IWQE integrates query dissemination and data collection into one stage, thus removing the requirement of infrastructure support. Moreover, as an infrastructure-free approach, IWQE incorporates in-network data aggregation technique which further provides opportunity for aggressive energy optimization.

IWQE algorithm is shown in Fig. 6. Once a window query (query window is marked as the rectangle in Fig. 6) is inserted into the network, IWQE forward the query toward the query window by a geo-routing protocol. After the query reaches the query window, a set of sensor nodes inside the query window, called query nodes (i.e., Q-nodes) are chosen for query dissemination. In Fig. 6, Q-nodes are the nodes connected by black arrows. For each receiving query, a Q-node broadcasts a probe message that includes the query and information about the pre-designed itinerary (shown by gray dashed lines). Upon hearing the probe message, the neighbor nodes that are qualified to reply the query, called data nodes (i.e., D-nodes), report their sensed data back to the Q-node. After aggregating the data from all D-nodes and the partial result received from the previous Q-node, the current Q-node selects the next Q-node based on the pre-designed itinerary and a query forwarding heuristic, and forward this new partial query result to the selected next Q-node. After the query traverses the entire query window, the aggregated result is returned back to the sink node, again by a geo-routing protocol. By performing data collection along with query propagation at each Q-node, IWQE does not rely on any infrastructure, thus is more robust and efficient.

Several research challenges raised by IWQE are addressed in [33]. In order to ensure the query accuracy, the *maximum itineracy width* (MIW) is defined and derived to ensure that all sensor nodes inside the query window are queried at least once, such that data from nodes falling into the query window can be collected. Three different itinerary routes, i.e., sequential itinerary, parallel itinerary and hybrid itinerary are examined aiming at energy

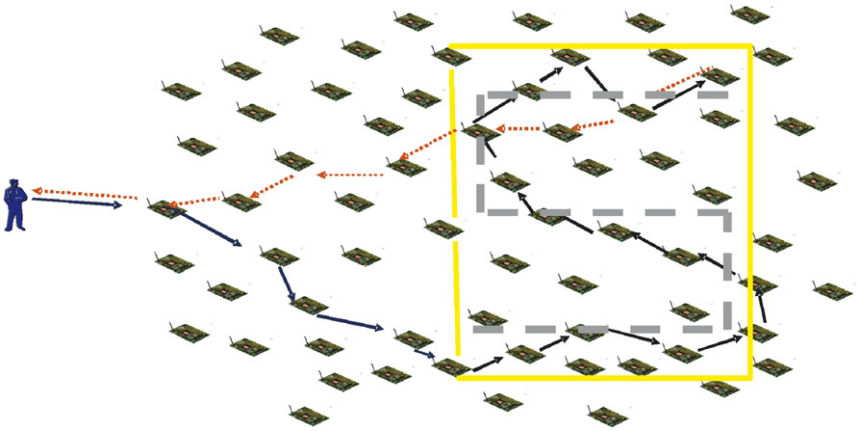

Fig. 6. Itinerary-based window query execution (IWQE) [33].

optimization, query latency reduction and a balance of those, respectively. Furthermore, itinerary traversal, including data collection heuristics and query forwarding heuristics are studied.

### 5.2. IKNN algorithm

Inspired by IWQE, we adopt itinerary-based approach for KNN query processing. One straightforward adoption is to design a similar itinerary as the one used by IWQE within a KNN search region estimated by KBT algorithm, and process KNN query by traversing the itinerary. This approach, which does not require infrastructure support, still faces the problem of KNN search region estimation as KBT does. Our design aims at stopping query processing as soon as the $k$ nearest nodes are visited to minimize the number of node visits without sacrificing query accuracy. Motivated by this goal, we consider to start the query processing from the query point (or the home node) and diffuse query outward along the itinerary to collect sensor readings. The itinerary is designed in a way such that the order of nodes being queried is the same as the order of their distance to the query point. With such itinerary, the query processing is able to stop as soon as the sensor readings from the KNN nodes are collected. In the following, we present the basic design of IKNN, followed by the discussion about the itinerary route, width and traversal.

#### 5.2.1. Basic idea of IKNN

In IKNN, the query is first forwarded toward the home node (i.e., the nearest node to the query point) using a geo-routing protocol. Upon receiving the packet, the home node, acting as query node (i.e., Q-node) broadcasts the probe message including the query and the information about the predesigned itinerary. Upon hearing the probe message, D-nodes (i.e., the neighbor nodes that are qualified to reply the query) report to the Q-node. Similar to IWQE, the current Q-node aggregates the data reports from all its D-nodes, and relay the query and partial query result to the next selected Q-node. With such itinerary-based approach, only a set of sensor nodes (i.e., Q-nodes) among the $k$ nearest neighbors need to broadcast the query.

#### 5.2.2. Itinerary route in IKNN

As we pointed out, to conserve energy resources without jeopardizing the query accuracy, the IKNN algorithm should stop once the KNN nodes are queried. This design goal cannot be achieved by GRT and KBT due to the nature of the infrastructure used. More specifically, partial results collected along several independent infrastructure branches are not enough to determine the stop condition of query processing. Different from IWQE, IKNN does not have a pre-specified query region (e.g., the query window for IWQE), which stops the query processing automatically when the region is fully covered. Thus, we consider a new itinerary route, called *Archimedean spiral* for IKNN to achieve the above goal.

An Archimedean spiral is a curve which in polar coordinates $(r, \theta)$ can be described by equation $r = b\theta$, with real number $b$. Fig. 7(a) shows an example of Archimedean spiral. The reason we choose Archimedean spiral counts to one of its key attribute that successive turnings of the spiral have a

constant separation distance, which equals to $2\pi b$, if $\theta$ is measured in radians. Thus, parameter $b$ controls the distance between the successive turnings, which also determines the density of the itinerary (i.e., spirals) used for query processing. IKNN sets the start of Archimedean spiral at the home node and the query is processed diffusing away from the query point. In other words, sensor nodes are queried in an order based on their distances to the query point approximately, as the distance between the successive turnings is constant. When IKNN collects data from the $k$ nearest nodes, it can stop immediately after traversing the next turning of the Archimedean spiral, since nodes residing outside the current turning are farther away from the query point than the nodes that have been queried. Fig. 8 shows an example of IKNN, in which the bounded node represents the home node; the Q-nodes are connected by solid arrows demonstrating the direction of query propagation; Dotted arrows depict the data collections. Again, comparing with GRT (shown in Fig. 3) and KBT (shown in Fig. 5), IKNN incurs less network communication represented by arrows.

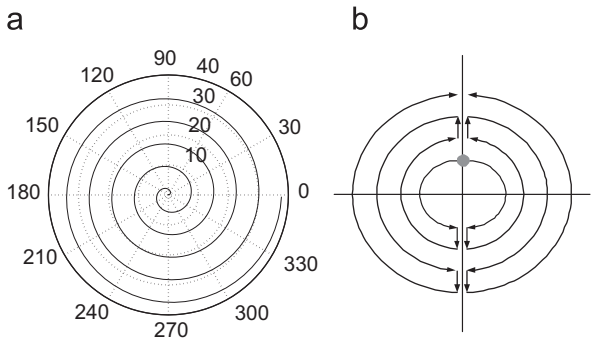


Fig. 7. Itinerary: (a) Archimedean spiral; (b) parallel itinerary.

Fig. 7(a) shows a sequential itinerary, in which only one copy of the query is processed at any time. Sequential query processing suffers from a long query latency, especially for a large $k$ (which implies a long itinerary). This long latency can be overcome by parallel itinerary, with which more than one copy of the query can be processed simultaneously. Fig. 7(b) shows an example of two threads of the query initiated by the home node (marked as the grey circle). However we want to point out that while using parallel itinerary may speed up the query processing, it is also critical to stop the query processing once the KNNs are queried to conserve energy. Therefore, along query processing, two itineraries need to meet periodically to examine the partial query result. As Fig. 7(b) shows that each itinerary only explores half of the geographical space and two itineraries encounter periodically at certain point to combine collected query results and determine the time for stopping the query processing. If $k$ or more than KNN nodes are queried, the query processing stops. Even though parallel itinerary reduces the query processing latency, it incurs potential communication collisions between two adjacent itineraries (especially where two itineraries are close to each other). Moreover, the protocol complexity is increased, as two itineraries must periodically exchange query results.

### 5.2.3. *Itinerary width in IKNN*

As [33] points out the itinerary density is critical to the query accuracy and energy usage. When an

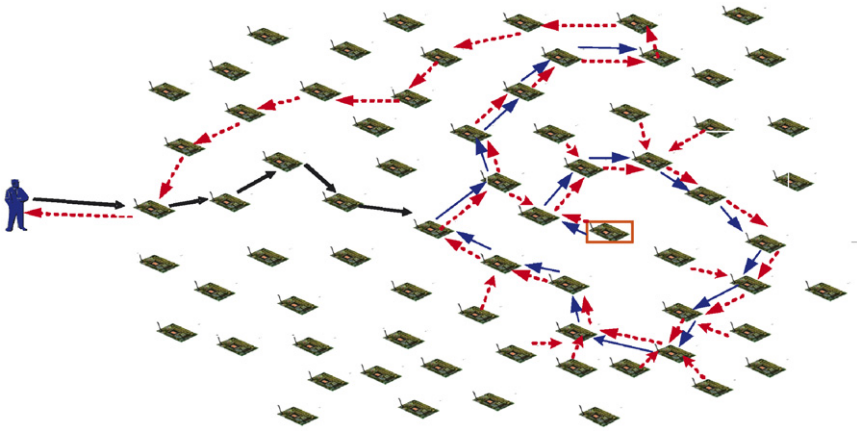

Fig. 8. Itinerary-based KNN query processing algorithm.

itinerary with high density is used, even though the query accuracy can be ensured, a sensor node may be queried for too many times, such that the energy usage increases. On the other hand, when the itinerary is too sparse, some nodes among KNNs may not receive the query, which deteriorates the query accuracy. We take Archimedean spiral as an example, and study the value of parameter $b$. Parallel itineracy can be examined in a similar way. To determine an appropriate itinerary density (i.e., the distance between the successive turnings), we use the research results presented in [33] about the maximum itinerary width (MIW), which is the upper bound of itinerary width that ensures the coverage of nodes of interest, derived in [33]. As the distance between two successive turnings on Archimedean spiral is $2\pi b$, equation $2\pi b \leqslant MWI = \sqrt{3}R/2$ needs to be satisfied to ensure the coverage of all KNN nodes. The derived $b$ from the above equation is not necessarily the optimal solution, but is the conserve settings to ensure the query accuracy.

#### 5.2.4. Itinerary traversal in IKNN

For data collection along itinerary, we consider contention-based scheme. In contention-based scheme, each D-node determines its reply precedence independently without assistance from the Q-node or knowledge about it neighbor nodes. Each D-node upon receiving the probe message that contains a reference line emanating from the current Q-node, sets a timer which can be calculated by a simple heuristic: timer $=$ max_delay $\times$ $(\alpha/2\pi)$, where $\alpha$ is the angle formed by the specified reference line and the line connecting the current Q-node and the current D-node, and max_delay is the maximum time that a Q-node is allowed to complete its data collection. A D-node does not respond to the Q-node until its timer expires. After aggregating the reports from all D-nodes, a Q-node selects the next Q-node to forward the query and partial query result. Routing along an itinerary has been well-studied in our prior work [33]. We adopt the heuristic of *most progress on itinerary* (MPI), which is effective in terms of energy consumption and query latency, and has a similar query accuracy to other forwarding heuristics. Considering the representation of Archimedean spiral $r = b\theta$, it is not easy to calculate the progress a given node made along the itinerary. As $r$ increases monopoly with variable $\theta$, we use the distance between a sensor node and the home node to evaluate the progress, as this distance is only affected by variable $\theta$.

## 6. Performance evaluation

In this section, we evaluate the performance of GRT, KBT and IKNN algorithms in terms of *energy consumption*, *query latency* and *query accuracy*. We first study the basic design of KBT and IKNN algorithms. We then compare the performance of GRT, KBT and IKNN algorithms and examine the impact of varying network conditions and application specifications (i.e., network density, the rate of node failure and the application parameter $k$), in order to test the sensitivity of designed KNN query processing algorithms to these factors.

We implemented all KNN query processing algorithms on CSIM [35] which allows customized and scaled simulation design of sensor networks. By default, 1500 sensors are deployed randomly inside a 500 m $\times$ 500 m region. Each of these sensors has a transmission radius of 40 m and has approximately 30 neighbors within its transmission radius. The query point $q$ for a given KNN query is chosen randomly with a $k$ value selected randomly from between 10 and 100. By default, nodes are stationary and do not fail. For each experiment we run five KNN queries back to back. The simulations run until all five queries have returned successfully or have been dropped due to a network failure, which typically takes between 10 and 20 s. We assume a *MessageDelay* of 30 ms. The packet size varies with different KNN processing algorithms. Moreover, given an algorithm (e.g., GRT and KBT), the packet sizes vary through query processing as more packets are disseminated along the path. The experiment results represent the average of 50 trials.

For each experiment we measure the following performance metrics:

- *Transmission energy consumption* ($J$): The total energy consumed for transmitting packets during the simulation time.
- *Query latency* ($ms$): The average elapsed time between a query being issued and results being received.
- *Query accuracy* (%): The percentage ratio of the number of nodes which are $k$ nearest nodes to $q$ reporting their results over $k$. The query accuracy for a failed query due to network failures is 0.

### 6.1. Study of KBT algorithm

This section studies the basic design of KBT algorithm. We first study the different heuristics for estimating the KNN search region, and explore an important tradeoff in KBT design by varying the TreeHeight value.

#### 6.1.1. Estimating KNN search boundary in KBT

Heuristics for estimating KNN search boundary determine the size of the estimated region, thus directly impacting the energy efficiency of KBT. For a clear comparison, we introduce the *optimal* KNN search region which contains exact KNN nodes with a *minimum* radius.

Fig. 9 shows the impact of query parameter $k$ (the driving factor on the radius size) on the different proposed boundary estimation techniques where the $Y$-axis demonstrates the radius of KNN search boundary. Fig. 9(a) shows that the SUMDIST and MHD have the best performance for very small $k$ (i.e., $k \leqslant 10$) and come very close to approaching the optimal boundary radius. However as $k$ increases, the region estimated by SUMDIST and MHD grow radically as shown in Fig. 9(b). This is because when $k$ is larger than the number of hops on the query forwarding path, the information collected by both MHD and SUMDIST approach is insufficient for making accurate estimations about the KNN search region. As we expected, the NeighborClass2 boundary method consistently gives boundary values closest to the optimal boundary size for large value of $k$ (i.e., $k > 10$). For the sake of clarity, we only simulate KNN queries with $k > 10$ and employ the NeighborClass2 boundary technique for KBT algorithm in the following experiments since it gives the best performance and has low overhead.

#### 6.1.2. Impact of KBT TreeHeight

An important design in KBT design is how to select an appropriate TreeHeight value since the internal nodes of the tree must wait long enough to obtain results from their children without unnecessarily increasing the query latency. The TreeHeight value is an estimate of the height of the KBT tree.

Fig. 10(a) shows the effect of varying TreeHeight from 2 to 12 on the KBT query accuracy. The figure shows that the accuracy of all KBT techniques increase with the tree height, as less datum are dropped due to expired timers. Fig. 10(b) shows the natural increase in the query latency as the tree height increases. Based on this experiment, we selected a tradeoff value of 6 as the default TreeHeight value.

### 6.2. Itinerary in IKNN

In this section, we study the performance of IKNN using sequential and parallel itinerary, respectively. We consider two network settings, i.e., 1000 and 1500 nodes deployed within the network. The impact of network density on GRT and KBT algorithms will be examined in Section 6.3.

Fig. 11 shows the performance of IKNN algorithm with and without data aggregation (denoted by agg and non-agg, respectively) along sequential

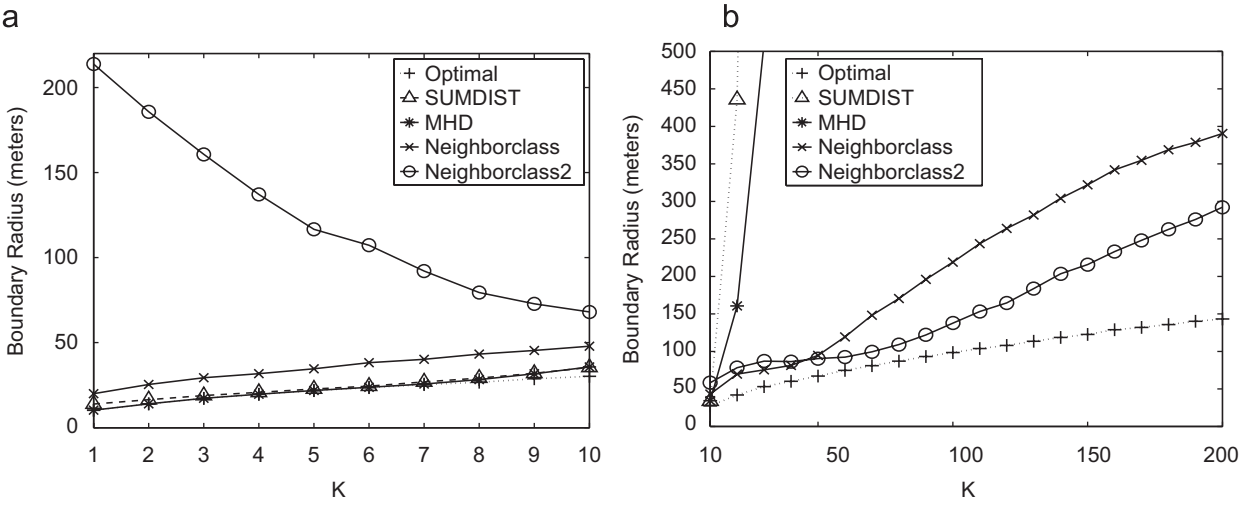


Fig. 9. KBT boundary techniques: (a) $k < 10$; (b) $k > 10$.

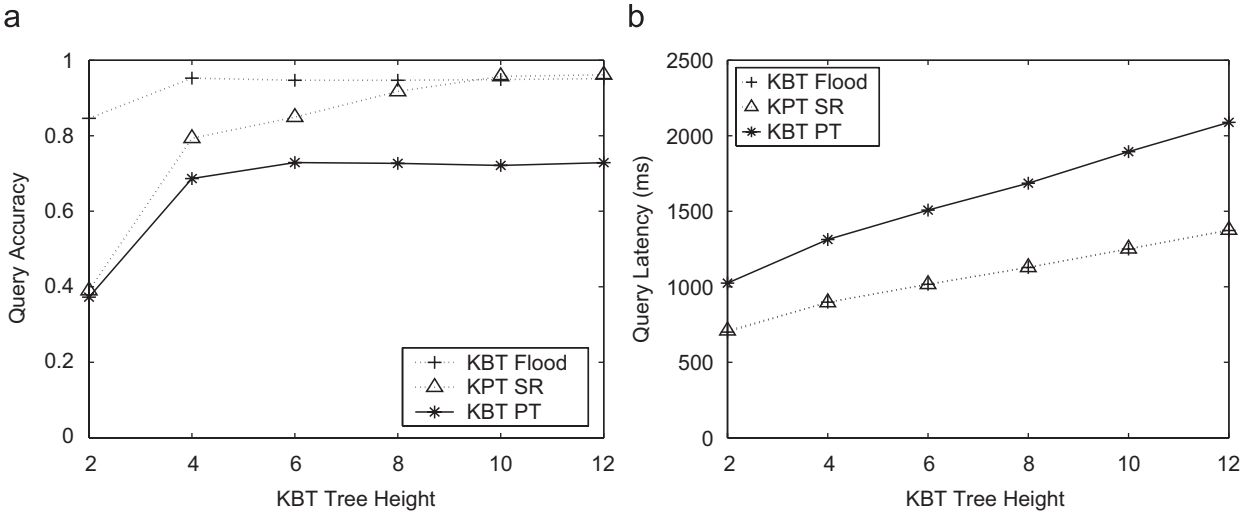


Fig. 10. Impact of KBT tree height: (a) query accuracy; (b) query latency.

and parallel itinerary (denoted by $S$ and $P$, respectively). For non-aggregation case, we assume that even though data from multiple nodes could be combined into one packet to reduce the overhead incurred by packet headers, sensor readings cannot be compressed. On the other hand, data aggregation (i.e., agg) refers to the case where sensor readings can be aggregated and compressed such that the data packet size remains the same along query processing in IKNN. Among three compared algorithms (i.e., GRT, KBT and IKNN), only IKNN supports such in-network data aggregation. GRT and KBT do not support in-network data aggregation as when data is propagated back to the application, the intermediate nodes relaying the data do not have enough information to avoid unnecessary transmission, as they cannot tell whether the collected data are from KNN nodes. In both GRT and KBT, the root node has to select the readings of the $k$ nearest nodes from all reported data based on the location where it is collected. Since GRT and KBT do not support in-network processing techniques, they are much less energy-efficient than IKNN when data aggregation is allowed, which will be further studied shortly.

As shown in Fig. 11(a), IKNN algorithm with sequential and parallel itinerary have close energy consumption. IKNN with data aggregation incurs significantly less energy cost than that without aggregation, since the total amount of collected data is drastically reduced by in-network processing. Since in-network processing techniques do not have significant impacts on query accuracy and query latency, we only show those of IKNN with sequential and parallel itinerary in Figs. 11(b) and (c), respectively. As we expected, a query accuracy of using sequential itinerary is close to that of using parallel itinerary. However, sequential itinerary causes more than 50% longer of query latency than parallel itinerary. Our experimental results show that this is due to the small $k$ value used in the experiment. When $k$ increases, the latency ratio between parallel and sequential itinerary approaches to $\frac{1}{2}$. Due to the space constrains, the results are not shown here.

### 6.3. Impact of network density

In this experiment, we measure the effect of sensor node density on the performance of GRT, KBT and IKNN algorithms by varying the number of sensor nodes inside the fixed region from 500 to 1500 (the average number of neighbors varies from 10 to 30), which is demonstrated in Fig. 12. For the clarity of the presentation, we only show the performance of IKNN using parallel itinerary with and without data aggregation, since experimental results in Section 6.2 show that the performance of sequential itinerary is close to that of parallel itinerary. Meanwhile, we depict the performance of KBT Flood and KBT SR, as KBT PT has worse performance than these two schemes.

Fig. 12(a) shows an increasing energy consumption in both GRT and KBT algorithms, as the

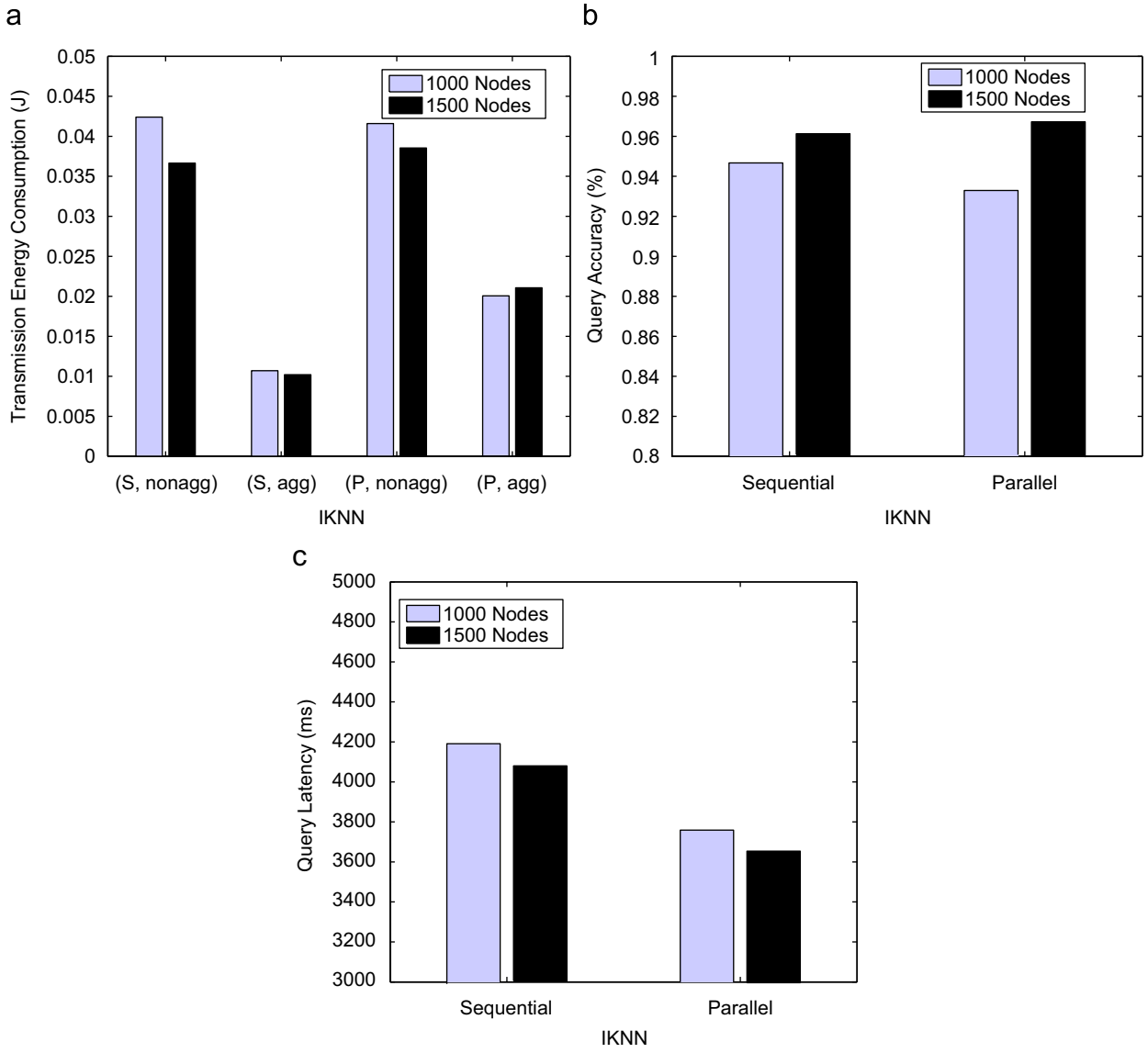


Fig. 11. Study of IKNN: (a) transmission energy consumption (J); (b) query accuracy; (c) query latency.

network becomes denser. This is because that more sensor nodes are involved in query processing. IKNN has much less energy consumption than KBT and GRT algorithms by reducing the number of node visits. The query accuracy of all algorithms depicted in Fig. 12(b) improves as more sensor nodes are deployed, since fewer routing failures are likely to happen in a denser network. This is especially clear for IKNN, because the Q-node selection is constrained within a relatively small region (i.e., within the itinerary width), which is more likely to fail comparing with tree-based routing in GRT and KBT algorithms. The problem worsens when the network is sparse. In a fairly dense network (e.g., 1000 nodes), IKNN has comparable query accuracy as KBT algorithms. Fig. 12(c) demonstrates the query latency for different algorithms. Query latency of IKNN and KBT is only slightly affected by sensor density. The reason that IKNN has a longer query latency than KBT algorithm is because KBT collect sensor readings through multiple tree branches, which is more time-efficient than IKNN algorithm using two itinerary for query processing. We believe by using more parallel itinerary, the query latency of IKNN algorithm can be further reduced. This paper focuses

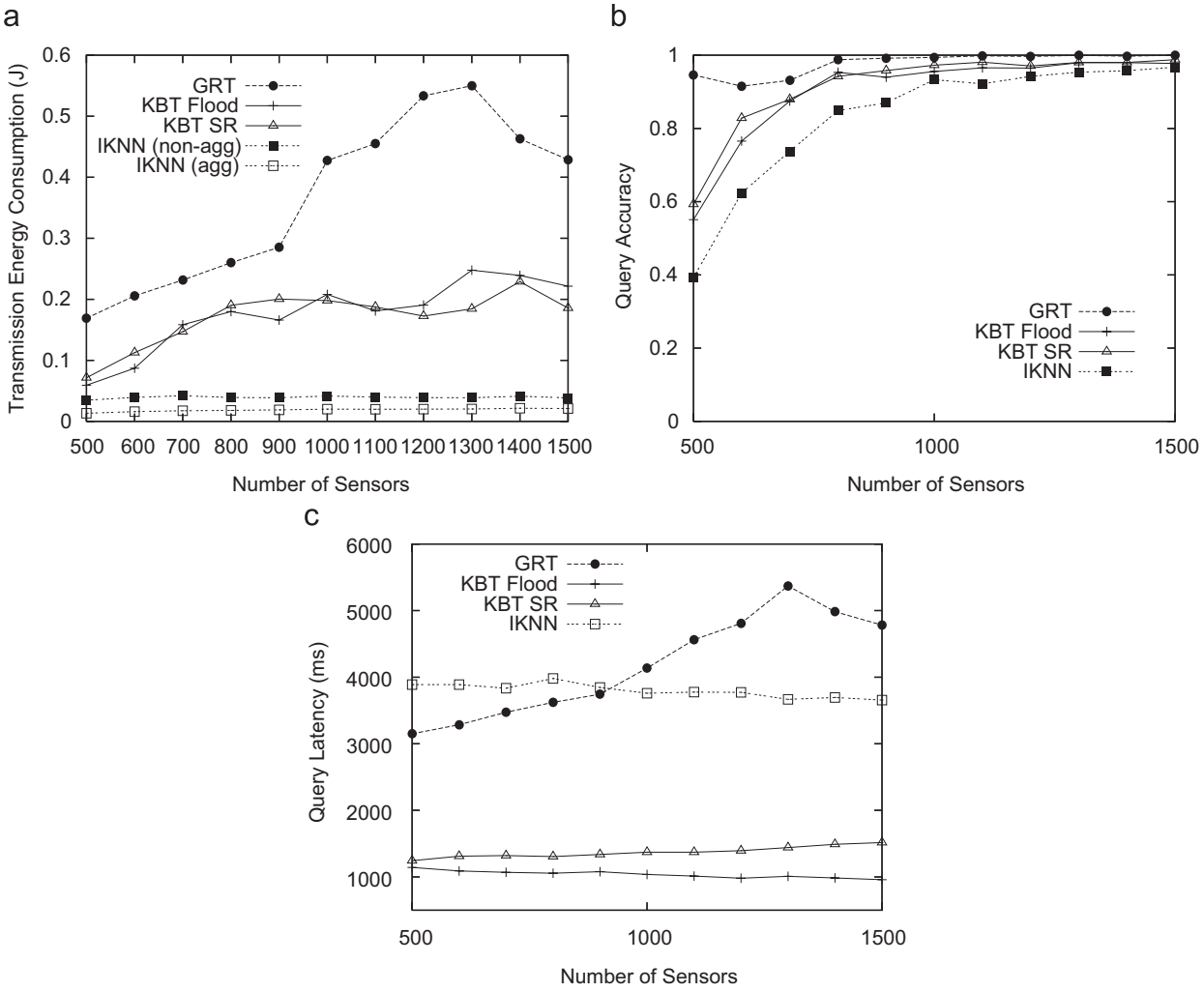


Fig. 12. Impact of sensor density: (a) transmission energy consumption; (b) query accuracy; (c) query latency.

on the performance comparison of different KNN query processing algorithms and will study the impact of parallel itinerary on IKNN in our future work.

### 6.4. Impact of $k$

In this section we investigate the application requirement of $k$ that directly affects the number of nodes involved with the query processing. Fig. 13 varies $k$ from 50 to 400. As KBT Flood and KBT SR have similar performance, we only show the performance of KBT Flood in this section.

Fig. 13(a) shows that with small and medium $k$ ($k<250$), the transmission energy consumed by IKNN is lower than GRT and KBT algorithms. However, IKNN without data aggregation has dramatically increasing energy consumption as $k$ further increases. This is because that all collected sensor readings are carried along a longer itinerary when $k$ becomes larger. The transmission energy consumed by the GRT algorithm stays fairly constant. It is because that when $k$ is larger than the average number of neighbors (i.e., $m$) a node has, the size of KTable transmitted from children node to their parent node is determined by $m$ that is constant in this experiment and that of KBT algorithm increases since the size of estimated KNN search region grows along with $k$.

Fig. 13(b) shows the effect of varying $k$ on query accuracy. The query accuracy of comparing algorithms is close, but all declines slowly as $k$ increases. Therefore, we conclude that when $k$ is

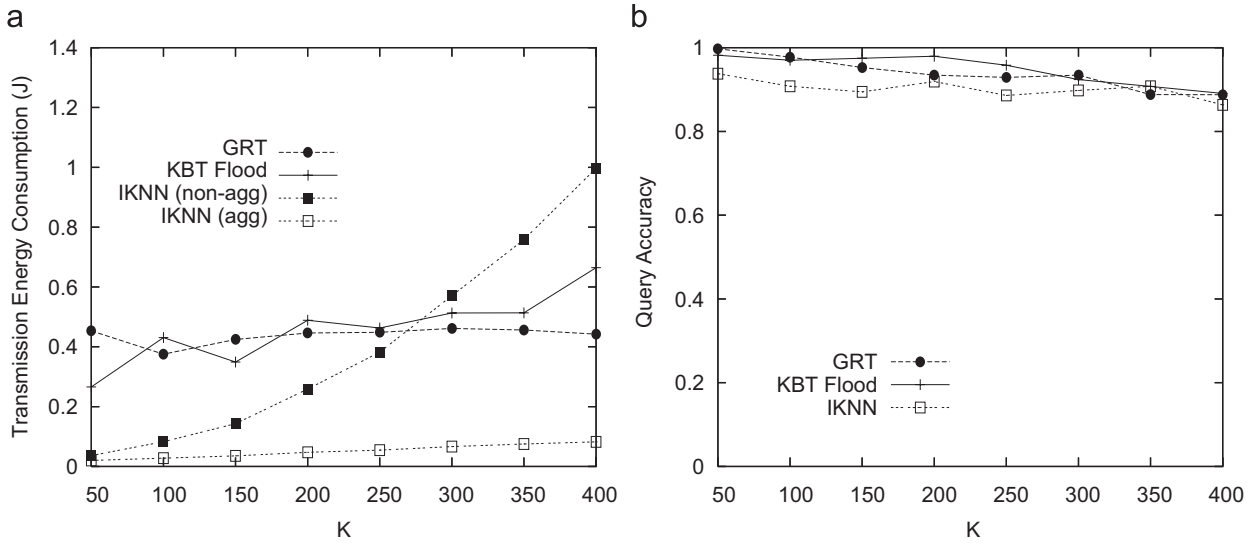


Fig. 13. Impact of $k$ (number of expected answers): (a) transmission energy consumption; (b) query accuracy.

small (e.g., $k<250$), IKNN is a better choice than KBT and GRT. On the other hand, when $k$ becomes larger and in-network data aggregation is not applicable, both GRT and KBT perform better than IKNN in terms of transmission energy consumption.

### 6.5. Impact of sensor failure

Another property of sensor networks that can affect query processing performance is the rate of failure of sensor nodes. Fig. 14 shows the effect of node failure on energy consumption and query accuracy by varying the percentage rate of node failure per second from 0% to 0.3%. A beacon period (the duration between two beacon messages) of 3 s is used for this experiment. As all algorithms require such a beacon mechanism, we do not count the energy consumption for maintaining neighbor states. The transmission energy consumption depicted in Fig. 14(a) decreases for KBT algorithm since less number of nodes within the estimated KNN region report back to the root node. In Fig. 14(b), the query accuracy of IKNN drops when more nodes fail, since it is more difficult to select Q-nodes when the network density drops with less nodes active within the network.

The results presented in this section show that when $k$ is small or medium, IKNN achieves comparable query accuracy at a much less energy cost compared with GRT and KBT algorithms. Considering the relative long query latency of IKNN, KBT algorithm is more appropriate for data collection with tight time constraints. With a larger $k$, if data aggregation is allowed, IKNN by supporting in-network processing significantly improves energy efficiency and outperforms GRT and KBT algorithms. Otherwise, GRT and KBT algorithms have comparable performance and both can be adopted for KNN query processing. IKNN using parallel itinerary has comparable energy efficiency and query accuracy as that using sequential itinerary. However, parallel itinerary is promising for shortening the query latency experienced by the sequential itinerary.

## 7. Conclusion and future work

In this paper, we present various challenges in processing KNN queries in sensor networks, and studied three query processing strategies, including global infrastructure-based, local infrastructure-based and infrastructure-free. These strategies demonstrate very distinctive performance characteristics. We designed three KNN query processing algorithms, namely, GRT, KBT and IKNN algorithms in accordance with different design philosophies. GRT prunes nodes along a tree infrastructure spanning over the network, thus incurring noticeable infrastructure maintenance overhead and unnecessary node visits. KBT reduces such an overhead by constraining the tree infrastructure within an estimated KNN search region, which directly determines the energy cost and accuracy of

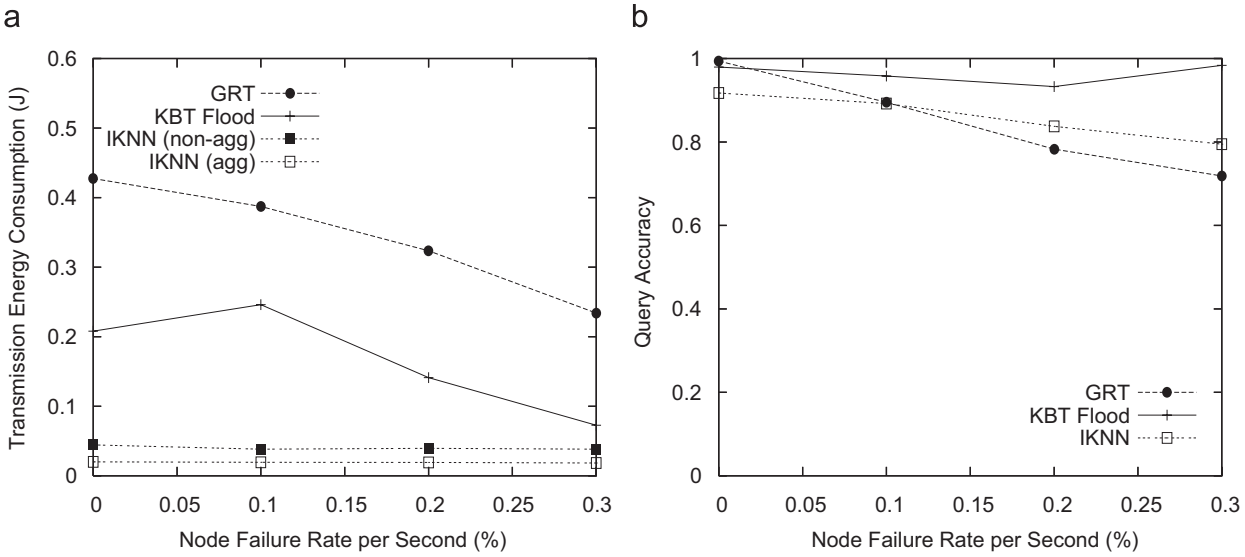


Fig. 14. Impact of sensor failure: (a) transmission energy consumption; (b) query accuracy.

KBT algorithm. Without prior knowledge about network topology, estimation of such KNN search region is difficult. IKNN takes a different approach by propagating query and collecting data along a pre-designed itinerary. Different from GRT and KBT algorithms, IKNN is able to stop query processing soon after collecting from the $k$ nearest nodes, thus effectively reducing the number of unnecessary node visits. To our best knowledge, this research represents the first study investigating KNN query processing in wireless sensor networks. Our research results reveal a profound guidance for designing and selecting appropriate KNN query processing algorithms corresponding to given network conditions and application requirements. An extensive performance evaluation of all three designed algorithms shows that when $k$ is small or medium, IKNN achieves better energy efficiency and comparable query accuracy at cost of prolonged query latency. When $k$ becomes large, both KBT and GRT are more efficient than IKNN algorithm. Moreover, among three algorithms, only IKNN is able to support in-network processing technique, which significantly optimizes energy consumption.

In our future work, we plan to adapt IKNN algorithm for order-sensitive KNN query and optimize KBT algorithm by determining Tree-Height based on the value of $k$. In addition, we will investigate the impact of parallelism in IKNN algorithm on reducing its query latency without sacrificing the energy efficiency and query accuracy. Moreover, we plan to implement the developed algorithms in a prototype.

## Acknowledgment

This research was supported in part by the National Science Foundation under Grant no. IIS-0328881, IIS-0534343 and CNS-0626709.

## References

[1] D. Li, K. Wong, Y. Hu, A. Sayeed, Detection, classification and tracking of targets in distributed sensor networks, IEEE Signal Process. Mag. 19 (2) (2002).

[2] H. Qi, X. Wang, S.S. Iyengar, K. Chakrabarty, High performance sensor integration in distributed sensor networks using mobile agents, Internat. J. High Performance Comput. Appl. 16 (3) (2002) 325–335.

[3] N. Roussopoulos, S. Kelley, F. Vincent, Nearest neighbor queries, in: Proceedings of ACM SIGMOD, San Jose, CA, 1995, pp. 71–79.

[4] T. Seidl, H.P. Kriegel, Optimal multi-step $k$-nearest neighbor search, in: Proceedings of ACM SIGMOD, New York, NY, 1998, pp. 154–165.

[5] Z. Song, N. Roussopoulos, $K$-nearest neighbor search for moving query point, in: Proceedings of International Symposium on Spatial and Temporal Databases, Los Angeles, CA, July 2001, pp. 79–96.

[6] Y. Xu, W.-C. Lee, J. Xu, G. Mitchell, PSGR: priority-based stateless geo-routing in highly dynamic sensor networks, Technical Report CSE 04-021, Pennsylvania State University, September 2004, ⟨http://www.cse.psu.edu/pda/SDB/pubs/PSGR_TR.pdf⟩.

[7] Y. Xu, W. Lee, Window query processing in highly dynamic geo-sensor networks: issues and solutions, in: Proceedings

of NSF Workshop on GeoSensor Networks, Portland, ME, October 2003.
[8] A. Guttman, R-trees: a dynamic index structure for spatial searching, in: Proceedings of ACM SIGMOD, Boston, MA, 1984, pp. 47–57.
[9] J. Winter, W. Lee, KPT: a dynamic KNN query processing algorithm for location-aware sensor networks, in: Proceedings of International Workshop on Data Management for Sensor Networks, Toronto, Canada, 2004, pp. 119–125.
[10] J. Winter, Y. Xu, W. Lee, Energy efficient processing of $k$ nearest neighbor queries in location-aware sensor networks, in: Proceedings of International Conference on Mobile and Ubiquitous Systems: Networking and Services, San Diego, CA, 2005, pp. 281–292.
[11] N. Patwari, A.O. Hero III, M. Perkins, N. Correal, R. O'Dea, Relative location estimation in wireless sensor networks, IEEE Trans. Signal Proc. (Special Issue Signal Process. Networking) 51 (9) (2002) 2137–2148.
[12] C. Wang, Survey on sensor network localization, Technical Report MSU-CSE-05-2, Department of Computer Science, Michigan State University, January 2005.
[13] B. Chen, K. Jamieson, H. Balakrishnan, R. Morris, SPAN: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks, Wireless Network 8 (5) (2002) 481–494.
[14] C. Schurgers, V. Tsiatsis, S. Ganeriwal, M. Srivastava, Topology management for sensor networks: exploiting latency and density, in: Proceedings of the International Symposium on Mobile Ad Hoc Networking and Computing, Lausanne, Switzerland, June 2002, pp. 135–145.
[15] Y. Xu, J. Heidemann, D. Estrin, Geography-informed energy conservation for ad hoc routing, in: Proceedings of ACM MobiCom, Rome, Italy, July 2001, pp. 70–84.
[16] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, Z. Protopapas, Fast nearest neighbor search in medical image databases, in: Proceedings of VLDB, San Francisco, CA, 1996, pp. 215–226.
[17] S. Chaudhuri, L. Gravano, Evaluating top-$k$ selection queries, in: Proceedings of VLDB, San Francisco, CA, 1999, pp. 397–410.
[18] S. Saltenis, C.S. Jensen, S.T. Leutenegger, M.A. Lopez, Indexing the positions of continuously moving objects, in: Proceedings of ACM SIGMOD, Dallas, TX, 2000, pp. 331–342.
[19] R. Benetis, C.S. Jensen, G. Karciauskas, S. Saltenis, Nearest neighbor and reverse nearest neighbor queries for moving objects, in: Proceedings of the 2002 International Symposium on Database Engineering & Applications, Washington, DC, 2002, pp. 44–53.
[20] S.R. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, The design of an acquisitional query processor for sensor networks, in: Proceedings of the ACM SIGMOD, San Diego, CA, June 2003.
[21] S. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, TAG: a tiny aggregation service for ad-hoc sensor networks, ACM SIGOPS Operating Systems Rev. 36 (SI) (2002) 131–146.
[22] D. Estrin, R. Govindan, J. Heidemann, S. Kumar, Next century challenges: scalable coordination in sensor networks, in: Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking, Seattle, WA, August 1999, pp. 263–270.
[23] W.R. Heinzelman, A. Chandrakasan, H. Balakrishnan, Energy-efficient communication protocol for wireless microsensor networks, in: Proceedings of the Hawaii International Conference on System Sciences, Maui, HI, January 2000, pp. 8020–8029.
[24] C. Intanagonwiwat, R. Govindan, D. Estrin, Directed diffusion: a scalable and robust communication paradigm for sensor networks, in: Proceedings of the ACM Annual International Conference on Mobile Computing and Networking, Boston, MA, August 2000, pp. 56–67.
[25] M. Demirbas, H. Ferhatosmanoglu, Peer-to-peer spatial queries in sensor networks, in: Proceedings of IEEE International Conference on Peer-to-Peer Computing, Linkping, Sweden, September 2003.
[26] G. Goldin, M. Song, A. Kutlu, H. Gao, H. Dave, Georouting and delta-gathering: efficient data propagation techniques for geosensor networks, in: Proceedings of NSF Workshop on GeoSensor Networks, Portland, ME, October 2003.
[27] B. Karp, H. Kung, GPSR: greedy perimeter stateless routing for wireless networks, in: Proceedings of the ACM Annual International Conference on Mobile Computing and Networking, Boston, MA, August 2000, pp. 243–254.
[28] Y.-B. Ko, N.H. Vaidya, Location-aided routing (LAR) in mobile ad hoc networks, Wireless Network 6 (4) (2000) 307–321.
[29] F. Kuhn, R. Wattenhofer, A. Zollinger, Worst-case optimal and average-case efficient geometric ad-hoc routing, in: Proceedings of ACM International Symposium on Mobile Ad Hoc Networking and Computing, Annapolis, MD, June 2003, pp. 267–278.
[30] Y. Xu, W.C. Lee, J. Xu, G. Mitchell, Psgr: priority-based stateless geo-routing in wireless sensor networks, in: Proceedings of IEEE Conference Mobile Ad-hoc and Sensor Systems, Atlanta, Georgia, April 2006.
[31] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, F. Yu, Data-centric storage in sensornets with GHT, a geographic hash table, ACM/Kluwer J. Mobile Networks Appl. 8 (4) (2003) 427–442.
[32] Y. Yu, R. Govindan, D. Estrin, Geographical and energy aware routing: a recursive data dissemination protocol for wireless sensor networks, Technical Report UCLA/CSD-TR-01-0023, UCLA Computer Science Department, May 2001.
[33] J. Xu, Y. Xu, W.C. Lee, G. Mitchell, Processing window queries in wireless sensor networks, in: Proceedings of IEEE ICDE, Atlanta, GA, April 2006.
[34] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, D. Estrin, Data-centric storage in sensornets, ACM SIGCOMM Comput. Commun. Rev. 33 (1) (2003) 137–142.
[35] H. Schwetman, CSIM user's guide (version 18), Mesquite Software, Inc., ⟨http://www.mesquite.com⟩, 1998.