1-2009

Localized Monitoring of kNN Queries in Wireless Sensor Networks

Yuxia YAO
Nanyang Technological University, Singapore

Xueyan TANG
Nanyang Technological University

Ee Peng LIM
Singapore Management University, eplim@smu.edu.sg

DOI: https://doi.org/10.1007/s00778-007-0089-3

Localized monitoring of *k*NN queries in wireless sensor networks

Yuxia Yao · Xueyan Tang · Ee-Peng Lim

Published in VLDB Journal, January 2009, Volume 18, Issue 1, pp 99–117
https://doi.org/10.1007/s00778-007-0089-3

Abstract Wireless sensor networks have been widely used in civilian and military applications. Primarily designed for monitoring purposes, many sensor applications require continuous collection and processing of sensed data. Due to the limited power supply for sensor nodes, energy efficiency is a major performance concern in query processing. In this paper, we focus on continuous *k*NN query processing in object tracking sensor networks. We propose a localized scheme to monitor nearest neighbors to a query point. The key idea is to establish a monitoring area for each query so that only the updates relevant to the query are collected. The monitoring area is set up when the *k*NN query is initially evaluated and is expanded and shrunk on the fly upon object movement. We analyze the optimal maintenance of the monitoring area and develop an adaptive algorithm to dynamically decide when to shrink the monitoring area. Experimental results show that establishing a monitoring area for continuous *k*NN query processing greatly reduces energy consumption and prolongs network lifetime.

1 Introduction

The development of wireless technology and sensors have enabled wide use of sensor networks. In these networks, a large number of low-powered sensor nodes are distributed in an area of interest and wirelessly connected. The sensor nodes are equipped with computation and communication capabilities [18]. Sensor networks are popular for a variety of applications, e.g., habitat monitoring, pollution monitoring, and object tracking [1,12]. They provide us with the means of interacting with the physical world. Allowing users to make queries to sensor networks is an important way to support the interactions. Most existing work has focused on non-spatial query processing in sensor networks [11,28]. To the best of our knowledge, there has been little work on spatial query processing [32,33]. In this paper, we consider monitoring *k*NN queries in object tracking sensor networks. A *k*NN query is characterized by a geographical location q called the *query point*, and a number k of requested nearest objects. The objective of a *k*NN query is to identify the k objects with the shortest distances to the query point. Consider a motivating example where a sensor network is deployed in the forest to track animals. The scientists may be interested to know the locations of the k animals nearest to a particular location over a period of time. They can issue such *k*NN queries into the sensor network and the continuous results are returned periodically.

Energy efficiency is a critical design consideration of wireless sensor networks. The sensor nodes, usually with low battery power, have to be deployed unattended for a long time. To prolong network lifetime, we need to reduce network-wide energy consumption. In wireless sensor networks, energy is mainly consumed in communication [18]. Thus, to reduce energy consumption, we need to reduce the number of message transmissions. A straightforward *centralized scheme* for monitoring *k*NN queries is to continuously send all sampled object locations to a base station. User queries are also routed to the base station for initial and continuous evaluations. However, the centralized scheme is likely to suffer from unnecessary update messages. This is because *k*NN queries are usually *localized* in that only the locations of the objects

Y. Yao (✉) · X. Tang · E.-P. Lim
School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798, Singapore
e-mail: yaoyuxia@pmail.ntu.edu.sg

X. Tang
e-mail: asxytang@ntu.edu.sg

E.-P. Lim
e-mail: aseplim@ntu.edu.sg

close to the query points are needed for query processing and the objects that are far away can be exempted from location updates. To improve energy efficiency, it is desirable to store the acquired data locally at the sensor nodes in a distributed manner and process the queries *in-network* [6,31,33]. Recent technology advances have substantially improved the capacities and energy efficiency of local storage for sensor networks [2].

Motivated by the localized property of kNN queries, in this paper, we propose a *localized scheme* to continuously evaluate kNN queries in object tracking sensor networks. Our key idea is to establish a monitoring area for each query so that only the location updates relevant to the query are collected. In this way, the network-wide energy consumption is reduced. We first propose a two-phase search mechanism to conduct the initial evaluation of a kNN query. A monitoring area is established together with the initial evaluation. Then, we develop methods to reevaluate the kNN query during query lifetime. The location updates from the sensor nodes in the monitoring area are collected for query reevaluation. Due to object movement, the monitoring area may need to be expanded and shrunk on the fly. We analyze the optimal maintenance of the monitoring area and develop an adaptive algorithm to dynamically decide when to shrink the monitoring area. Experimental results show that establishing a monitoring area for continuous kNN query processing greatly reduces energy consumption and prolongs network lifetime. It is also shown that the adaptive algorithm for maintaining the monitoring area achieves close-to-optimal performance.

The rest of the paper is organized as follows. Section 2 summarizes the related work. Section 3 introduces some preliminaries and Sect. 4 presents the localized scheme to continuously monitor kNN queries. The maintenance strategies of the monitoring area are presented in Sect. 5. Section 6 describes the experimental setup and discusses the experimental results. Finally, Sect. 7 concludes the paper.

2 Related work

With the growing needs for location-based services, continuous monitoring of kNN queries is becoming increasingly popular in the context of spatial databases [3,13,14,19,30,35]. Recently, some grid-based methods are explored in continuous monitoring of kNN queries. Yu et al. [35] proposed a method called YPK-CNN, assuming a centralized repository storing the locations of all objects which are indexed by a grid structure. Location updates are continuously sent to the centralized repository. kNN queries are reevaluated periodically according to the new locations of the objects. YPK-CNN achieves low computation time and memory usage. The CPM scheme proposed by Mouratidis et al. [13] also uses a grid structure for indexing. It further reduces the computation time by optimizing the visiting order of the grid cells to handle location updates more effectively.

Similar to YPK-CNN, the SEA-CNN scheme proposed by Xiong et al. [30] also stores the object locations in a centralized repository and indexes them by a grid structure. Each query is assigned a circle called the answer region centered at the query point and with a radius equal to the distance between the query point and the kth nearest object in the kNN result. When location updates are collected, the query is reevaluated only if the updates affect its answer region. SEA-CNN achieves high scalability in terms of computation time when there are multiple queries.

The above methods assume that there is a centralized repository to store all object locations and all location updates are simply reported to the centralized repository. However, such centralized storage is costly for object tracking sensor networks due to their energy constraints. Therefore, these methods are not appropriate for kNN monitoring in sensor networks.

Other relevant work on spatial query monitoring includes the MobiEyes algorithm proposed by Gedik et al. [3] and a threshold-based algorithm proposed by Mouratidis et al. [14]. Similar to YPK-CNN, SEA-CNN and CPM, these studies also assume a centralized server in the system. But differently, the MobiEyes and threshold-based algorithms assume *smart* objects that have some storing and processing capabilities. When an object moves away from its current position, the object can decide whether to send a location update to the server or not. Both the MobiEyes and threshold-based algorithms aim at reducing the communication cost between the objects and the server by eliminating unnecessary location updates. MobiEyes [3] focuses on monitoring range queries by assigning a safe region to each query. The objects within the safe region periodically check whether they are in the query ranges. Only the objects within the query ranges report their locations to the server. The threshold-based algorithm [14] assigns a distance range to each object in the result set of a kNN query. The distance range for the ith nearest object is delineated by the midpoint between the ith and $(i-1)$th nearest objects and the midpoint between the ith and $(i+1)$th nearest objects. Only when an object moves out of its distance range is the location update of the object sent to the server. The approaches of [3] and [14] are similar to many studies on continuous monitoring in sensor networks [22–24,28]. The general idea of these studies is to set some constraints at the sensor nodes to prevent them from reporting all sensed data to the base station. Only when the constraint is violated should the sensor node report the updates of sensed data. The detailed setting of the constraints at relevant sensor nodes depends on specific query types. However, these studies have focused on monitoring stationary phenomena (e.g., temperature and humidity). A stationary phenomenon is always captured by the same sensor node over

time. In object tracking sensor networks, however, the object location is detected by different sensor nodes as the object moves. In this paper, we make use of monitoring areas to eliminate unnecessary updates from the sensor nodes.

Spatial query processing is drawing more attention to the sensor network community. J. Winter et al. [27] and S.-H. Wu et al. [29] proposed schemes to search the k nearest sensor nodes in sensor networks. The general idea is to visit and collect information from the sensor nodes within an estimated search space. In [27], the search space is divided into several subspaces and the sensor nodes in each subspace are organized into a minimum spanning tree. The information of the sensor nodes is collected by the roots of the trees and then gathered at the nearest node to the query point where the k nearest nodes are computed. Recently, S.-H. Wu et al. [29] designed an itinerary-based approach to search for k nearest sensor nodes. Similar to the idea in [27], the search space is divided into several subspaces to enable parallel search. The sensor nodes in each subspace are visited sequentially according to a well-defined itinerary. While the above work aimed at locating the k nearest sensor nodes, we focus on searching for the k nearest objects in object tracking sensor networks.

Xu et al. [31] investigated ID-based one-shot queries in object tracking sensor networks. Xu et al. [32] studied window query processing in sensor networks. In this approach, a window query is propagated along the sensor nodes via a well-designed itinerary. Data collected by the sensor nodes are aggregated along with query propagation until all nodes in the window have been visited. In an earlier work, we proposed a grid-based scheme to process one-shot 1-NN queries in sensor networks [33]. Similar to [32], a 1-NN query is evaluated by sequentially visiting the grid cells surrounding the query point. However, these studies have focused on one-shot query processing. In contrast, in this paper, we consider continuous query processing. This paper substantially extends a preliminary report of our work presented at a conference [34].

3 Preliminaries

3.1 System model

We consider a sensor network with the sensor nodes distributed on a 2D place. The sensor nodes are aware of their locations through GPS [4] or other localization algorithms [16]. Each sensor node can communicate directly with the nodes (called neighbors) within a radio range R_t. Through message exchanges, each sensor node is aware of the geographical locations of its neighbors. We assume the network is connected, i.e., any sensor node can communicate with any other node either directly or indirectly through a routing protocol.

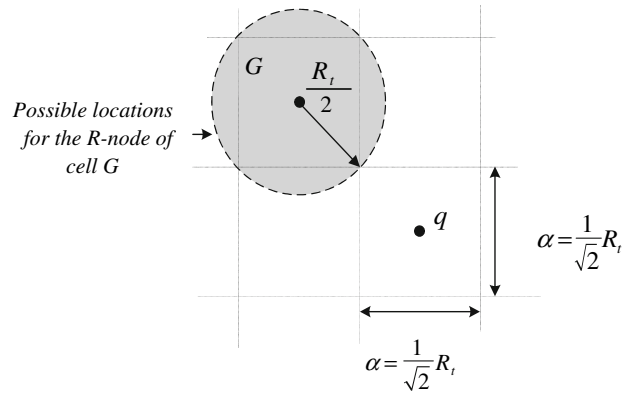


Fig. 1 Grid structure

We assume a dense sensor network in which the geographical area of interest is fully covered by the sensing ranges of the sensor nodes. At each sampling interval, the location of each object is detected by a sensor node in the network.[1] Suppose the sensing range of each sensor node is R_s. Then, the detecting sensor node of an object must be located within distance R_s to the object. Instead of sending all detected object locations to a central repository, we propose to store them locally at the detecting sensor nodes. kNN queries can be made via sensor nodes from anywhere in the network (e.g., through the hand-held devices by the users). Recall that each kNN query specifies a query point q. The sensor node receiving a query first forwards it towards q through GPSR routing [8]. Given a destination location, GPSR routes the message to the node closest to the destination location [20]. Thus, the node closest to the query point q would receive the query. We shall refer to this node as the *query sink*. Our objective is to continuously collect the kNN result at the query sink which in turn returns the result to the user.

3.2 One-shot kNN query processing

We first consider the processing of one-shot kNN queries, which forms the basis of initial and continuous evaluations of long running kNN queries. For each kNN query, we define a grid structure to conceptually partition the sensor network into a set of grid cells. As shown in Fig. 1, each grid cell is a square of size $\alpha \times \alpha$ and the query point q is set as the centroid of a grid cell. Each sensor node can autonomously compute the grid cell in which it is located provided that it knows the location $(q.x, q.y)$ of query point q. The centroid of the grid cell containing a sensor node located at (x, y) is given by $\left(q \cdot x + \lfloor \frac{x-(q \cdot x - \frac{\alpha}{2})}{\alpha} \rfloor \cdot \alpha, q \cdot y + \lfloor \frac{y-(q \cdot y - \frac{\alpha}{2})}{\alpha} \rfloor \cdot \alpha\right)$. As shall

[1] Although the sensor nodes may work collaboratively to determine the location of an object in their vicinity [10], we assume that for each object, only one sensor node (the sensing leader or cluster head) is responsible for storing its location at each sampling interval [9,31,37]. For simplicity, the detecting sensor node in the rest of this paper refers to this node.

be discussed later, the knowledge of q is made known to relevant sensor nodes in query processing.

Starting from the query sink, the query message is passed along the sensor nodes to search for nearby objects in the neighboring grid cells. When a grid cell is visited, the object locations detected by the sensor nodes in the cell are collected by one sensor node called the *R-node*. To do so, the R-node broadcasts a one-hop *probe* message to the sensor nodes in the cell. To guarantee that all nodes within the cell can hear the probe message, the diameter of the grid cell (i.e., $\sqrt{2}\alpha$) should be less than the transmission range R_t. Therefore, we set the cell size at $\alpha = \frac{1}{\sqrt{2}} \cdot R_t$. To cover the entire cell with one-hop broadcast, the R-node of a grid cell must be located within distance $\frac{1}{2}R_t$ to the centroid of the cell (as shown in Fig. 1).

The visit to a grid cell G is divided into two steps.

- In the first step, the query message is routed to the R-node of G. To do so, the sensor node currently holding the query message first checks whether any of its neighbors is within distance $\frac{1}{2}R_t$ to G's centroid. If such a neighbor exists, it is selected as the R-node of G and the query message is sent to it in one hop. In case multiple neighbors are within distance $\frac{1}{2}R_t$ to G's centroid, the one closest to G's centroid is selected as the R-node of G. Otherwise, if no such neighbor exists, the query message is routed towards G's centroid by GPSR routing. In this case, the sensor node closest to G's centroid would receive the query message [20]. If the receiving sensor node is within distance $\frac{1}{2}R_t$ to G's centroid, it assumes the role of G's R-node. Otherwise, G must be an empty grid cell without any sensor node in it, and hence no data needs to be collected from G. The R-node of G would be successfully selected in this step if G is non-empty.
- In the second step, the R-node broadcasts a probe message to the sensor nodes in G. The probe message contains the locations of the query point q and G's centroid. By including the query point in the probe message, each sensor node receiving the probe message can compute the centroid of the grid cell containing it. The node then compares the centroid with G's centroid to determine whether it is in G. Only the nodes in G would reply to the R-node with detected object locations if any. A number of scheduling methods exist to avoid the collisions between the replies from different nodes in G [32].

The evaluation of a one-shot kNN query proceeds in two phases: (i) *preliminary search* and (ii) *expanded search*. The purpose of the preliminary search is to find a *boundary object* and define the search space. In this step, the grid cells surrounding the query sink are visited by message passing until at least k objects are found. Among these objects, the kth nearest object to the query point is selected as the boundary object. A search space is then defined based on the location of the boundary object to guarantee the inclusion of all sensor nodes possibly detecting an object closer to the query point than the boundary object. During the expanded search, the grid cells in the search space that are not yet visited in the preliminary search are visited to locate the k nearest objects to the query point. Finally, the query result is routed back to the query sink. We now present the preliminary search and the expanded search in detail.

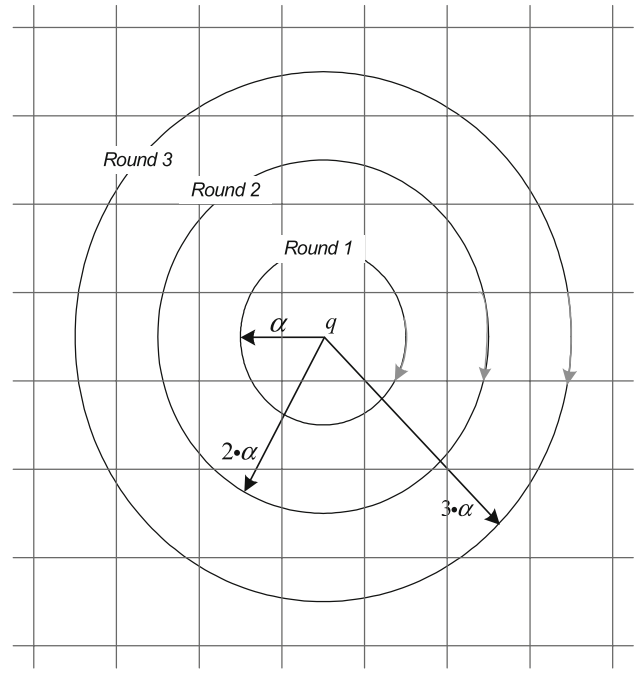


Fig. 2 Circular approach

3.3 Preliminary search

In the preliminary search, we need a rule to determine the order of the grid cells visited by the query message. Since the location of the boundary object determines the search space for the expanded search, to reduce search cost, we would like the boundary object to be as close to the query point q as possible. Thus, it is intuitive to visit the grid cells based on their distances to q. We propose a circular approach to determine the visiting order of grid cells. Specifically, the search starts from the cell centered at q and is divided into rounds as shown in Fig. 2. In each round i ($i \geq 1$), the unvisited grid cells whose minimum distances[2] to q are shorter than $i \cdot \alpha$ are visited in clockwise order. Figure 3 shows the visiting order of grid cells in the preliminary search. Initially, the query message contains the location of the query point q. When a grid cell is visited, its R-node collects the object locations detected by the sensor nodes in the cell and records them in the query message. Given the location of q, the R-node

[2] The minimum distance from a grid cell to the query point q refers to the distance between q and the point on the cell's perimeter closest to q.

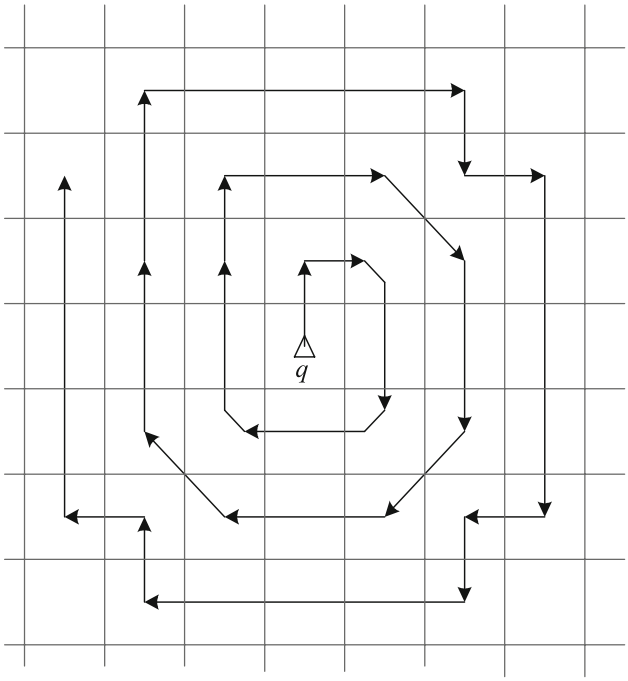


Fig. 3 Visiting order of grid cells in the preliminary search

autonomously determines the next cell to visit and routes the query message to it as discussed in Sect. 3.2.

The preliminary search completes when the number of objects recorded in the query message is no less than k. Among these objects, the kth nearest object to the query point q is chosen as the boundary object. Denote the boundary object by o_b and its distance to q by $d(o_b, q)$. The search space is then defined as the set of grid cells whose minimum distances to q are shorter than $d(o_b, q) + R_s$, where R_s is the sensing range. Intuitively, if a grid cell is in the search space, the sensor nodes in the grid cell are likely to detect objects closer to the query point q than the boundary object.

3.4 Expanded search

We refer to the grid cells to be visited in the expanded search as a *search list*. It consists of the set of grid cells in the search space that are not yet visited in the preliminary search. Note that these grid cells must be included in rounds $i, i+1, \ldots, j$ of the circular visiting order, where i is the round where the preliminary search ends and $j = \lceil \frac{d(o_b,q)+R_s}{\alpha} \rceil$. All grid cells in these rounds can be arranged in a sequence following their visiting order in the circular approach. We shall use a bit sequence of equal length to represent the search list, where a bit '1' means the corresponding grid cell is in the search list and a bit '0' means otherwise. To assist the mapping of bits to grid cells, the round number i is also included in the representation.

The query message in the expanded search contains the search list, the k recorded object locations, and the query point q. To visit a grid cell G in the search list, the query message is again routed to the R-node of G. In the expanded search, the R-node first removes G from the search list (by setting the corresponding bit in the bit sequence to 0). After the R-node collects the object locations detected by the sensor nodes in G, one of the following three cases can occur: (i) no object is detected by any node in G; (ii) all objects detected are further away from the query point q than the boundary object; (iii) at least one object detected is closer to q than the boundary object. In cases (i) and (ii), the search list and the object locations recorded in the message remain unchanged. In case (iii), the detected object locations closer to q are used to update the k object locations recorded in the message. Meanwhile, the new kth nearest object is assigned to be the boundary object o_b and the search space is shrunk accordingly. The search list is then updated by removing all grid cells outside the new search space by setting their bits to 0. After visiting grid cell G, the query message visits the next cell on the search list that is closest to G.[3] The expanded search continues until the search list becomes empty. On

Algorithm 1 Algorithm executed at R-nodes

if a query message p is received at a R-node i responsible for a grid cell G **then**
  Let $\mathcal{X}$ be the set of object locations recorded in message p;
  Collect object locations by the sensor nodes in cell G, and let $\mathcal{Y}$ be the set of object locations collected;
  if $|\mathcal{X}| < k$ and $|\mathcal{X} \cup \mathcal{Y}| < k$ **then**
    Replace $\mathcal{X}$ by $\mathcal{X} \cup \mathcal{Y}$ in message p;
    Determine the next grid cell G' to visit according to the circular approach;
    Send out message p to G' to continue the preliminary search;
  else if $|\mathcal{X}| < k$ and $|\mathcal{X} \cup \mathcal{Y}| \geq k$ **then**
    Replace $\mathcal{X}$ in message p by k object locations in $\mathcal{X} \cup \mathcal{Y}$ nearest to the query point q and let o_b be the kth nearest object to q in $\mathcal{X} \cup \mathcal{Y}$;
    Initialize the search list based on the location of o_b;
    Record the search list in p;
    Select cell G' from the search list closest to cell G;
    Send out message p to G' to start the expanded search;
  else if $|\mathcal{X}| = k$ **then**
    Remove cell G from the search list in message p;
    Let o_b be the kth nearest object to the query point q in $\mathcal{X}$;
    if $\exists o_i \in \mathcal{Y}, d(o_i, q) < d(o_b, q)$ **then**
      Replace $\mathcal{X}$ in message p by k object locations in $\mathcal{X} \cup \mathcal{Y}$ nearest to q;
    end if
    if the search list is not empty **then**
      Select cell G' from the search list closest to cell G;
      Send out message p to G' to continue the expanded search;
    else
      Send out a result message p' including $\mathcal{X}$ to the query sink;
    end if
  end if
end if

[3] In case that there are two grid cells with the same distances to G, we break the ties by choosing the grid cell closer to the query point as the next grid cell to visit.

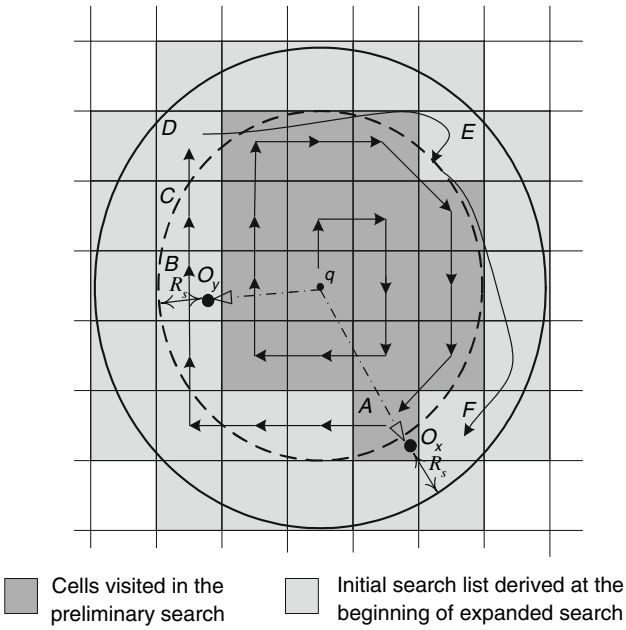


Fig. 4 Expanded search

completion of the expanded search, the message is routed to the query sink and the k recorded object locations form the kNN result.

Figure 4 shows an example of 1-NN query processing. The dark grey grid cells are visited in the preliminary search. Suppose a boundary object o_x is found when cell A is visited. A's R-node determines the search space (defined based on the solid circle in Fig. 4) and derives the search list (the light grey grid cells in Fig. 4). In the expanded search, the query message then starts visiting the grid cells on the search list. Suppose that when cell B is visited, an object o_y closer to q than o_x is found. Then, the search space is shrunk accordingly. The new search space is defined based on the dashed circle in Fig. 4. Cells C, D, E and F are now the only four grid cells left in the updated search list. The expanded search completes after visiting these cells.

Algorithm 1 summarizes the algorithm executed at each R-node when a query message is received.

4 Localized scheme for continuous kNN queries

4.1 Overview

The set of kNNs and their locations may change over time as the objects move. In this section, we propose a localized scheme to continuously derive the kNN result at the query sink. The basic idea is to collect only the relevant data from the sensor nodes near the query point for kNN monitoring. A straightforward strategy is to reevaluate the query from scratch at each sampling interval using the one-shot kNN query processing algorithm described in Sect. 3.2. However, this may incur large number of query messages. In the following, we propose to set up a *monitoring area* for a continuous kNN query in the sensor network. The sensor nodes in the monitoring area proactively report the location updates that may potentially affect the kNN result to the query sink. There are two stages in the processing of a continuous kNN query with a monitoring area setup. In the first stage, the kNN query is initially evaluated at the first sampling interval using the scheme described in Sect. 3.2. The monitoring area is established along with the initial query evaluation as will be discussed in Sect. 4.2. In the second stage, the query sink continuously collects the location updates, reevaluates the query results, and maintains the monitoring area at each subsequent sampling interval. The query reevaluation will be discussed in Sect. 4.3 and the maintenance of monitoring area will be discussed in Sect. 4.4.

4.2 Monitoring area setup

A monitoring area is defined as the set of grid cells whose minimum distances to q are shorter than $d(o_k, q)+R_s$, where $d(o_k, q)$ is the distance between the kth nearest object o_k and q, and R_s is the sensing range. The radius is set in this way to guarantee that all sensor nodes possibly detecting the objects closer to q than o_k are included in the monitoring area. To collect all object locations closer to q than o_k, the sensor nodes in the monitoring area may simply update all detected object locations with the query sink at each sampling interval. However, this may result in unnecessary location update for objects that are further away from q than o_k. To reduce location updates, we divide the grid cells in the monitoring area into two groups: all-report cells and partial-report cells. The sensor nodes in the all-report cells update all detected object locations with the query sink; the sensor nodes in the partial-report cells keep a *distance threshold* and only update with the query sink the detected object locations within the distance threshold from q. To guarantee that all object locations closer to q than o_k are reported, the distance thresholds at the sensor nodes in the partial-report cells should be set at no less than $d(o_k, q)$. Lastly, the sensor nodes in the grid cells beyond the monitoring area do not update any detected object location with the query sink.

We now show how to set up the monitoring area in the initial query evaluation. Recall that during the preliminary and expanded searches, all grid cells whose minimum distances to q are shorter than $d(o_k, q)+R_s$ are visited. When a grid cell is visited, its R-node broadcasts a probe message to all nodes in the cell. To establish the monitoring area, a new parameter is included in the probe message to indicate whether the grid cell is classified as an all-report cell or a partial-report cell. All grid cells visited during the preliminary search are classified as all-report cells. For each grid cell visited during

the expanded search, if its minimum distance to the query point q is shorter than the distance from the boundary object recorded in the query message to q, the grid cell is classified as an all-report cell. Otherwise, the grid cell is classified as a partial-report cell and the distance thresholds at the sensor nodes in the grid cell are set to the distance from the boundary object to q. Note that the boundary object recorded in the query message at any time in the expanded search cannot be closer to q than the kth nearest object o_k in the final kNN result. Therefore, the distance thresholds set must be larger than $d(o_k, q)$.

Figure 5 illustrates an example of monitoring area setup following Fig. 4 in Sect. 3.2. The dark grey grid cells are visited in the preliminary search and the light grey grid cells G_1 to G_9 are visited in the expanded search. The dark grey grid cells are classified as all-report cells. When grid cells G_1 to G_4 are visited, o_x is the boundary object and G_1 to G_4 are classified as all-report cells since their minimum distances to q are all shorter than $d(o_x, q)$. At G_5, a new boundary object o_y replaces o_x in the query message. When G_6 to G_9 are visited, the boundary object recorded in the query message is o_y. As the minimum distances from G_5 and G_6 to q are shorter than $d(o_y, q)$, G_5 and G_6 are classified as all-report cells. On the other hand, G_7 to G_9 are classified as partial-report cells because their minimum distances to q are longer than $d(o_y, q)$. As a result, the distance thresholds at the sensor nodes in G_7 to G_9 are set at $d(o_y, q)$.

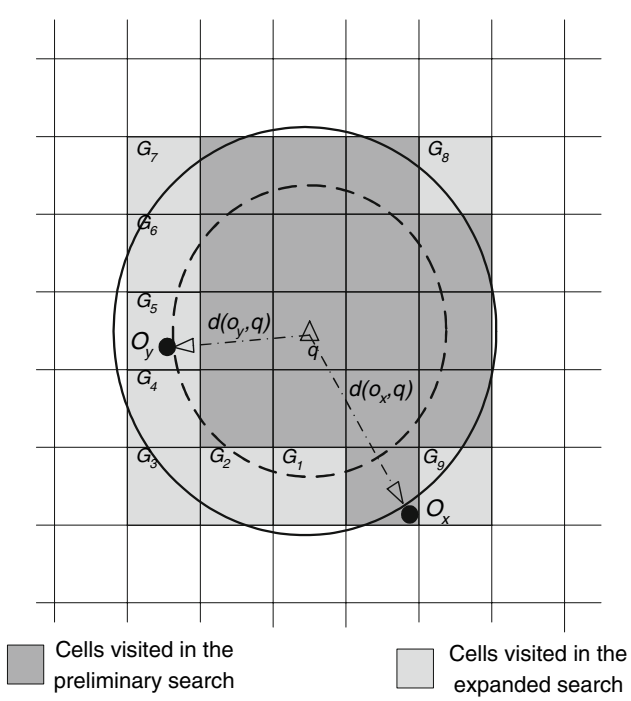


Fig. 5 Setup of monitoring area

4.3 Query reevaluation

After the initial query evaluation at the first sampling interval, the monitoring area is set up accordingly. At subsequent sampling intervals, the query sink continuously collects the location updates from the sensor nodes in the monitoring area and reevaluates the kNN result.

Consider a sampling interval i. Suppose the monitoring area at the beginning of interval i includes the set of grid cells whose minimum distances to q are shorter than $d(o_k^{i-1}, q) + R_s$, where o_k^{i-1} is the kth nearest object to q at interval $i-1$. At interval i, denote by k' the number of object locations reported by the sensor nodes in the monitoring area. If $k' < k$ (case $\mathcal{A}$), the query sink needs to search for $k - k'$ more objects. Similar to the scheme described in Section 3.2, the search involves two phases: the preliminary search and the expanded search. The only difference is that the all-report cells are exempted from the preliminary search. This is because the sensor nodes in these grid cells have already reported all detected object locations to the query sink. Note that the grid cells whose minimum distances to q are shorter than $d(o_k^{i-1}, q)$ must have been classified as all-report cells. Thus, the query message, containing the k' object locations collected, is sent from the query sink to the first grid cell in round $\lceil \frac{d(o_k^{i-1}, q)}{\alpha} \rceil$ of the circular visiting order to start the preliminary search.

If $k' \geq k$, a list of k object locations closest to q are selected from the k' object locations collected. Let o_b be the kth nearest object to q among these k objects. If $d(o_b, q) \leq d(o_k^{i-1}, q)$ (case $\mathcal{B}$), the k objects selected are the new kNN result since all object locations nearer to q than o_k^{i-1} are included in the k' object locations collected. Neither the preliminary search nor the expanded search is needed in this case. Otherwise, if $d(o_b, q) > d(o_k^{i-1}, q)$ (case $\mathcal{C}$), the expanded search is carried out to refine the new kNN result. o_b is set as the initial boundary object for the expanded search, and the search list includes the grid cells whose minimum distances to q are shorter than $d(o_b, q) + R_s$ and longer than $d(o_k^{i-1}, q)$ (note that all grid cells whose minimum distances to q are shorter than $d(o_k^{i-1}, q)$ were all-report cells).

4.4 Maintenance of monitoring area

The monitoring area, initially set up at the first sampling interval, may need to be updated later due to the change in the kNN result upon query reevaluation. Let o_k^i and o_k^{i-1} be the kth nearest objects to q at intervals i and $i-1$ respectively. Then, the monitoring area at the beginning of interval i includes the grid cells whose minimum distances to q are shorter than $d(o_k^{i-1}, q) + R_s$. If $d(o_k^i, q) > d(o_k^{i-1}, q)$, the monitoring area should be expanded at interval i to include all grid cells whose minimum distances to q are shorter than

$d(o_k^i, q) + R_s$. Thus, a set of new grid cells should be added to the monitoring area. The sensor nodes in these grid cells need to be notified to update detected object locations with the query sink starting from the next sampling interval. Note that o_k^i is further away from q than o_k^{i-1} only in cases $\mathcal{A}$ and $\mathcal{C}$ discussed in Sect. 4.3. In both cases, the preliminary search and/or expanded search are needed to reevaluate the query. The new grid cells to be added to the monitoring area would be visited in these searches. Similar to the initial query evaluation, the sensor nodes in these cells are notified along with the searches.

On the other hand, if $d(o_k^i, q) \leq d(o_k^{i-1}, q)$, the monitoring area can be shrunk to reduce the number of sensor nodes updating object locations with the query sink. As shown in Fig. 6, the old monitoring area includes the grid cells whose minimum distances to q are shorter than $r_{\text{old}} = d(o_k^{i-1}, q) + R_s$. The new monitoring area includes the set of grid cells whose minimum distances to q are shorter than $r_{\text{new}} = d(o_k^i, q) + R_s$. We divide the grid cells in the old monitoring area into three categories:

(i) The grid cells whose minimum distances to q lie in $[r_{\text{new}}, r_{\text{old}}]$ (shown by the dark grey grid cells in Fig. 6). The sensor nodes in these grid cells need to be informed to stop updating detected object locations with the query sink;

(ii) The grid cells whose minimum distances to q lie in $[d(o_k^i, q), r_{\text{new}}]$ (shown by the light grey grid cells in Fig. 6). These grid cells would be classified as partial-report cells in the new monitoring area. The sensor nodes in these grid cells need to be informed about the new distance threshold $d(o_k^i, q)$;

(iii) The grid cells whose minimum distances to q are shorter than $d(o_k^i, q)$ (shown by the white grid cells in Fig. 6). These grid cells are all-report cells in the old monitoring area and would remain all-report cells in the new monitoring area.

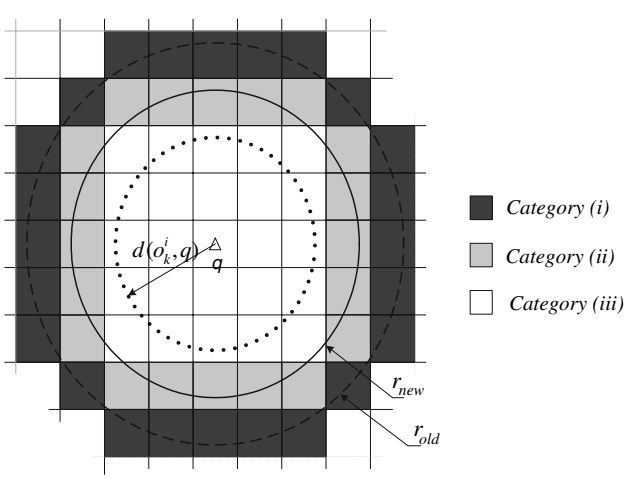


Fig. 6 Shrinking of the monitoring area

To inform the sensor nodes in the grid cells of the first two categories, a notification message is sent to them through geocast [15]. To describe the target area of geocast, the notification message contains the location of the query point q, and the new distance threshold $d(o_k^i, q)$ as well as the radius of the old monitoring area r_{old}. In the geocast, the notification message is first sent to a sensor node in the target area by unicast. Starting from this sensor node, the message is flooded to all sensor nodes in the target area. For each node in a grid cell whose minimum distance to q falls in $[r_{\text{new}}, r_{\text{old}}]$, it stops reporting location updates to the query sink upon receiving the notification message. For each node in a grid cell whose minimum distance to q falls in $[d(o_k^i, q), r_{\text{new}}]$, it records the distance threshold upon receiving the notification message.

Traditional geocast does not guarantee that all sensor nodes in the target area would receive the notification message. This is because some nodes in the target area may only be reachable via the nodes outside the target area [26]. To increase the delivery rate, in the unicast step of geocast, the notification message can be sent to a number of m nodes in the target area instead of one node only [26]. The flooding then starts from these nodes concurrently. As shown in Fig. 7, the target area in our case has a ring shape. We propose to equally divide the ring into m sub-areas. A notification message is sent to the center of each sub-area in the unicast step of geocast. In the flooding step, each sensor node receiving the notification message further rebroadcasts the message to its neighbors, unless it has received the notification message before or it is in a grid cell whose minimum distance to q is longer than r_{old} or shorter than $d(o_k^i, q)$. We shall study the impact of m through simulation experiments (Sect. 6).

There is in fact a tradeoff between the overhead of shrinking the monitoring area and the saving in location updates sent by the sensor nodes in the monitoring area. If the monitoring area is shrunk whenever the kth nearest object moves

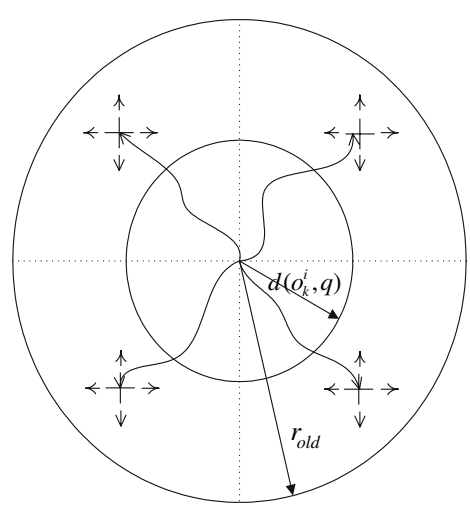


Fig. 7 Example of geocast ($m = 4$)

Algorithm 2 Algorithm executed at the query sink

1: At the first sampling interval, conduct the initial query evaluation by sending out a query message p to the first grid cell G according to the circular approach;
2: Wait for the result message p';
3: Extract from p' the k nearest object locations to the query point q and let o_k^1 be the kth nearest object location;
4: **for** each subsequent sampling interval i before the query expires **do**
5: Record in $\mathcal{U}$ the location updates received from the sensor nodes in the monitoring area;
6: **if** $|\mathcal{U}| < k$ **then**
7: Record $\mathcal{U}$ in a query message p;
8: Select the grid cell G to visit for the preliminary search;
9: Send out message p to G to start the preliminary search;
10: Wait for the result message p';
11: Extract from p' the k nearest object locations to the query point q and let o_k^i be the kth nearest object location;
12: **else**
13: Derive from $\mathcal{U}$ the k nearest object locations to q (called $\mathcal{U}'$) and let o_b be the kth nearest object location in $\mathcal{U}'$;
14: **if** $d(o_b, q) > d(o_k^{i-1}, q)$ **then**
15: Initialize the search list for the expanded search;
16: Record $\mathcal{U}'$ and the search list in a query message p;
17: Select cell G from the search list which is nearest to the query sink;
18: Send out message p to G to start the expanded search;
19: Wait for the result message p';
20: Extract from p' the k nearest object locations to the query point q and let o_k^i be the kth nearest object location;
21: **else if** $d(o_b, q) \leq d(o_k^{i-1}, q)$ **then**
22: The k nearest object locations to q are those in $\mathcal{U}'$ and let $o_k^i = o_b$;
23: **end if**
24: **end if**
25: **if** the maintenance strategy determines to shrink the monitoring area **then**
26: Send out the shrink message to a relevant ring-shaped area;
27: **end if**
28: **end for**

nearer to q (called the *AggressiveShrink* strategy), unnecessary location updates are aggressively eliminated. However, geocasting the notification message to shrink the monitoring area incurs communication overhead. Moreover, if the kth nearest object moves away from q later, the monitoring area would have to be expanded again. As a result, the overhead of updating the monitoring area may exceed the saving in location updates, thereby increasing the total communication cost. On the other hand, if the monitoring area is never shrunk (called the *NoShrink* strategy), a large number of unnecessary location updates may be sent to the query sink leading to high total communication cost. In our preliminary work [34], we designed a simple maintenance strategy that looks ahead to the location update saving and the shrinking overhead in one subsequent sampling interval only. In fact, shrinking, the monitoring area normally affects the location update costs in a series of subsequent sampling intervals. Thus, in the following section, we systematically investigate the maintenance of the monitoring area. We first analyze an offline optimal schedule to shrink the monitoring area. Then, we propose an adaptive strategy that dynamically decides when to shrink the monitoring area on the fly based on the shrinking overhead relative to the saving in location updates.

Algorithm 2 highlights the algorithm executed at the query sink including the initial query evaluation at the first time interval (steps 1–3), and query reevaluations at subsequent sampling intervals (steps 4–28).

5 Scheduling strategy to shrink the monitoring area

5.1 Optimal schedule to shrink the monitoring area

Given all object locations and their detecting sensor nodes at each sampling interval, we would like to compute the optimal schedule to shrink the monitoring area, i.e., to find a set of intervals such that the total message complexity is minimized if the monitoring area is shrunk at these intervals. The total message complexity includes those of shrinking the monitoring area, query reevaluation, and location updates at each sampling interval.

Suppose that at an interval i, the monitoring area consists of all grid cells whose minimum distances to q are shorter than $d(o_k^i, q) + R_s$, where o_k^i is the kth nearest object at interval i. We consider the message complexity at a sampling interval $v > i$ assuming that the monitoring area does not shrink at intervals $i+1, i+2, \ldots, v-1$. At the beginning of interval v, the monitoring area would include all grid cells whose minimum distances to q are shorter than $\max_{i \leq x < v} d(o_k^x, q) + R_s$, where o_k^x is the kth nearest object at interval x. We shall denote by $c_r(i, v)$ the message complexity for the sensor nodes in the monitoring area to report location updates to the query sink at interval v. Let o_k^v be the kth nearest object at interval v. If $d(o_k^v, q) > \max_{i \leq x < v} d(o_k^x, q)$, the query reevaluation at the query sink involves preliminary search and/or expanded search. As a result, the monitoring area must be expanded. Otherwise, if $d(o_k^v, q) \leq \max_{i \leq x < v} d(o_k^x, q)$, the query reevaluation does not involve any message transmission, and it is possible to shrink the monitoring area at interval v. In this case, we say that interval v is shrinkable with respect to interval i. Given i, we shall denote all intervals $v > i$ that are shrinkable with respect to interval i by a set $S(i)$. Without loss of generality, for any $v > i$, we define $c_q(i, v)$ as the message complexity for query reevaluation at interval v; for any $v \in S(i)$, we define $c_s(i, v)$ as the message complexity for shrinking the monitoring area at interval v. The complexities $c_r(i, v)$, $c_q(i, v)$, $c_s(i, v)$ can be derived from the object locations and their detecting sensor nodes at the sampling intervals.

Assume that at a sampling interval I, the monitoring area consists of all grid cells whose minimum distances to q are shorter than $d(o_k^I, q) + R_s$. We consider a period of sampling intervals $I+1, I+2, \ldots, H$. Suppose the monitoring area is shrunk at intervals $x_1, x_2, \ldots, x_m$, where $I < x_1 < x_2 < \cdots < x_m \leq H$, $x_i \in S(x_{i-1})$ for each $1 < i \leq m$, and $x_1 \in S(I)$. Then, the total message complexity over intervals $I+1, I+2, \ldots, H$ is given by

$$\begin{aligned} cost(I, H : x_1, x_2, \ldots, x_m) &= c_s(I, x_1) \\ &+ \sum_{v=I+1}^{x_1} \left(c_r(I, v) + c_q(I, v)\right) \\ &+ \sum_{i=1}^{m-1} \left(c_s(x_i, x_{i+1}) + \sum_{v=x_i+1}^{x_{i+1}} \left(c_r(x_i, v) + c_q(x_i, v)\right) \right) \\ &+ \sum_{v=x_m+1}^{H} \left(c_r(x_m, v) + c_q(x_m, v)\right). \end{aligned} \tag{1}$$

Given $c_r(i, v)$ and $c_q(i, v)$ for all $I \leq i < v \leq H$, and $c_s(i, v)$ for any i and v where $v \in S(i)$, the offline optimal shrinking schedule problem is to find a set of intervals $I < x_1 < x_2 < \cdots < x_m \leq H$ that minimize (1). For convenience, we shall call it the (I, H)-optimization problem. We show that the problem can be solved by a dynamic programming algorithm. This is because the optimal schedule to the (I, H)-optimization problem contains optimal solutions to some subproblems. Let $(x_1, x_2, \ldots, x_m)$ be an optimal shrinking schedule to the (I, H)-optimization problem. Then, $(x_2, x_3, \ldots, x_m)$ must be an optimal shrinking schedule to the (x_1, H)-optimization problem. This is because if there exists another shrinking schedule $(y_1, y_2, \ldots, y_l)$ (where $x_1 < y_1 < y_2 < \cdots < y_l \leq H$) that results in a lower message complexity than $(x_2, x_3, \ldots, x_m)$, i.e.,

$$\begin{aligned} &cost(x_1, H : y_1, y_2, \ldots, y_l) < cost \\ &\quad (x_1, H : x_2, x_3, \ldots, x_m), \end{aligned} \tag{2}$$

it follows that

$$\begin{aligned} &cost(I, H : x_1, y_1, y_2, \ldots, y_l) \\ &\quad = c_s(I, x_1) + \sum_{v=I+1}^{x_1} \left(c_r(I, v) + c_q(I, v)\right) \\ &\quad + cost(x_1, H : y_1, y_2, \ldots, y_l) \\ &\quad < c_s(I, x_1) + \sum_{v=I+1}^{x_1} \left(c_r(I, v) + c_q(I, v)\right) \\ &\quad + cost(x_1, H : x_2, x_3, \ldots, x_m) \\ &\quad = cost(I, H : x_1, x_2, ..., x_m), \end{aligned}$$

which contradicts the optimality of $(x_1, x_2, \ldots, x_m)$.

For any $I \leq j < H$, let $C[j]$ be the minimum achievable message complexity in the (j, H)-optimization problem, and $B[j]$ be the first shrinking interval in the optimal schedule to the (j, H)-optimization problem. Note that if the monitoring area does not shrink at any interval, the message complexity over intervals $j+1, j+2, ..., H$ is $\sum_{v=j+1}^{H} \left(c_r(j, v) + c_q(j, v)\right)$. Hence, the recurrences for dynamic programming are given by

$$C[j] = \begin{cases} c_r(H-1, H) + c_q(H-1, H) & \text{if } j = H-1, \\ \min\left(\min_{j<x<H, x\in S(j)} \left(c_s(j, x) + \sum_{v=j+1}^{x} \left(c_r(j, v) + c_q(j, v)\right) + C[x] \right), \sum_{v=j+1}^{H} \left(c_r(j, v) + c_q(j, v)\right) \right) & \\ \qquad\qquad \text{if } j < H-1, & \end{cases} \tag{3}$$

and

$$B[j] = \begin{cases} \emptyset \quad \text{if } j = H-1, \\ \arg \min_{j<x<H, x\in S(j)} \left(c_s(j, x) + \sum_{v=j+1}^{x} \left(c_r(j, v) + c_q(j, v)\right) + C[x] \right) \\ \quad \text{if } j < H-1 \text{ and } C[j] = \min_{j<x<H, x\in S(j)} \left(c_s(j, x) + \sum_{v=j+1}^{x} \left(c_r(j, v) + c_q(j, v)\right) + C[x] \right), \\ \emptyset \quad \text{if } j < H-1 \text{ and } C[j] = \sum_{v=j+1}^{H} \left(c_r(j, v) + c_q(j, v)\right). \end{cases} \tag{4}$$

Starting from $C[H-1]$ and $B[H-1]$, we can compute all $C[j]$'s and $B[j]$'s in decreasing order of j. On obtaining all $C[j]$'s and $B[j]$'s, the optimal shrinking intervals can be derived by tracing back the B-entries. Starting from $x_1 = B[I]$, we can obtain the shrinking intervals in the optimal schedule by setting $x_{v+1} = B[x_v]$ iteratively until $B[x_v] = \emptyset$.

Now we analyze the time complexity of the dynamic programming. Let $P = H - I$ be the length of the period under consideration. Given any w, the computation complexity of $\sum_{v=w+1}^{u} \left(c_r(w, v) + c_q(w, v)\right)$ for all different u's is $O(P)$. Hence, $\sum_{v=w+1}^{u} \left(c_r(w, v) + c_q(w, v)\right)$ for all pairs of u and w can be computed in a pre-processing stage in $O(P^2)$ time. Then, the time complexity to compute $C[j]$ is given by $O(P)$. Thus, the time complexity to compute all $C[j]$'s is given by $O(P^2)$. Therefore, the total time complexity of the dynamic programming algorithm is $O(P^2)$. The computed optimal schedule shall be referred to as the *OptimalShrink* strategy.

5.2 Adaptive schedule to shrink the monitoring area

The OptimalShrink strategy provides the minimal total cost over a designated period. However, it is computed in an offline manner where object locations at all sampling intervals are assumed known a priori. In practice, the object locations are not known beforehand. Thus, in this section, we propose an adaptive strategy (called *AdaptiveShrink*) to dynamically decide when to shrink the monitoring area on the fly. The general idea is to compare the saving in location updates with the overhead of shrinking the monitoring area.

Consider a sampling interval i. Suppose prior to interval i, the monitoring area was last updated at interval $j < i$ (see Fig. 8). It could be either shrunk by geocast or be expanded due to query reevaluation. Then, the monitoring area at the beginning of interval i includes the grid cells whose minimum distances to q are shorter than $r_{\text{old}} = d(o_k^j, q) + R_s$. The monitoring area can be shrunk at interval i only if $d(o_k^i, q) < d(o_k^j, q)$. If the monitoring area is shrunk at interval i, the new monitoring area would include the grid cells whose minimum distances to q are shorter than $r_{\text{new}} = d(o_k^i, q) + R_s$. We notice that, if the monitoring area is shrunk at interval i, it saves not only the location updates at the next sampling interval $i+1$, but also the location updates at subsequent sampling intervals as long as the sizes of the monitoring areas at these sampling intervals are less than r_{old}. Let $h > i$ be the first interval at which the size of the monitoring area first exceeds r_{old} since interval i (i.e., $d(o_k^h, q) \geq d(o_k^j, q)$). Then, the shrinking of the monitoring area at interval i would save the location updates at intervals $i+1, i+2, \ldots, h$. Denote the total saving of location updates by c_{saving}. On the other hand, the overhead of shrinking the monitoring area not only includes the cost of shrinking the monitoring area at interval i (denoted as c_{shrink}), but also includes the cost of expanding the monitoring area from size r_{new} to size r_{old} due to query reevaluation at later sampling intervals. Let $g > i$ be the first interval at which the size of the monitoring area exceeds r_{new} [i.e., $d(o_k^g, q) > d(o_k^i, q)$]. It follows that $h \geq g > i$. Then, the cost of expanding the monitoring area from size r_{new} to r_{old} is given by the total query reevaluation cost at intervals $g, g+1, g+2, \ldots, h$ (denote by c_{query}). In our adaptive strategy, the new monitoring area is kept unchanged at interval i if $c_{\text{saving}} \leq c_{\text{shrink}} + c_{\text{query}}$. Otherwise, if $c_{\text{saving}} > c_{\text{shrink}} + c_{\text{query}}$, the monitoring area is shrunk. The problem remains to predict c_{saving}, c_{shrink} and c_{query} at interval i.

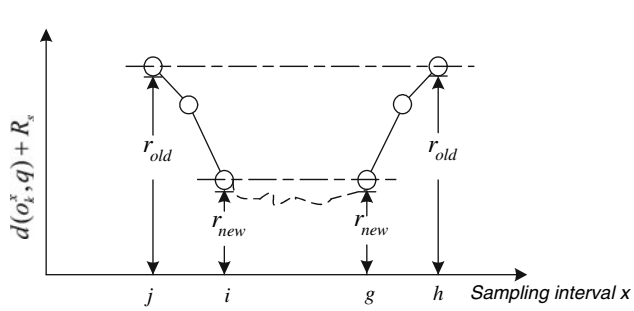


Fig. 8 Adaptive strategy

We start by estimating intervals g and h. Note that $g > i$ is the first interval since i such that $d(o_k^g, q) > d(o_k^i, q)$. For clarity of presentation, we shall replace g by $g(i)$. We estimate $g(i)$ by computing $g(i) - i$ based on the historical durations of $g(v) - v$ for each interval v prior to i. That is, $g(i) - i = \frac{1}{i-1} \sum_{v=1}^{i-1} (g(v) - v)$, i.e., $g(i) = i + \frac{1}{i-1} \sum_{v=1}^{i-1} (g(v) - v)$. To do so, the query sink keeps the history of the kth nearest object locations. For each $v < i$, $g(v)$ can be computed straightforwardly if $\max_{v<u<i} d(o_k^u, q) > d(o_k^v, q)$. Otherwise, if $\max_{v<u<i} d(o_k^u, q) \leq d(o_k^v, q)$, $g(v)$ is simply set to i. To compute h, we estimate $h - g$ by $i - j$. Thus, given g and j, h is computed as $i - j + g$.

After estimating the intervals g and h, we first compute c_{saving}, the saving in location updates. Let $\mathcal{N}$ be the number of objects whose distances to the query point q are between $r_{\text{old}} = d(o_k^j, q) + R_s$ and $r_{\text{new}} = d(o_k^i, q) + R_s$ at interval i. $\mathcal{N}$ can be derived at the query sink. For each sampling interval from $i+1$ to g, we estimate that $\mathcal{N}$ objects would be exempted from location updates if the monitoring area is shrunk at interval i. For the sampling intervals from $g+1$ to h, the monitoring area is gradually expanded from size r_{new} to r_{old}. We approximate that on average $\frac{1}{2}\mathcal{N}$ objects are exempted from location updates at each interval from $g+1$ to h. The message complexity to send one location update of an object to the query sink is approximated by $\lceil \frac{1}{2}(r_{\text{old}} + r_{\text{new}})/R_t \rceil$. Thus, the total saving in location updates from interval $i+1$ to h is estimated by

$$c_{\text{saving}} = \left(g - i + \frac{1}{2} \cdot (h - g)\right) \cdot \mathcal{N} \cdot \lceil \frac{1}{2}(r_{\text{old}} + r_{\text{new}})/R_t \rceil.$$

Next, we compute c_{shrink}, the cost of shrinking the monitoring area at interval i. Following the two steps of geocast, the cost of shrinking the monitoring area includes that of sending the unicast messages to the target area and that of flooding the messages within the target area. For simplicity, we exclude the cost of sending the unicast messages in the

estimation of c_{shrink} since it is usually much smaller than the cost of flooding. The message complexity of flooding is approximated by the number of sensor nodes in the target area. The target area includes all grid cells whose minimum distances to q lie between $d(o_k^i, q)$ and $r_{\text{old}} = d(o_k^j, q) + R_s$. According to the derivation in [33], given the size α of a grid cell, the number of grid cells whose minimum distances to q are shorter than r is given by $a \cdot r^2 + b \cdot r + c$, where $a = \frac{\pi}{\alpha^2}$, $b = \frac{4.1178}{\alpha}$, $c = 2.3241$. Thus, the number of grid cells in the target area is given by

$$\left(a \cdot (d(o_k^j, q) + R_s)^2 + b \cdot (d(o_k^j, q) + R_s) + c\right) - \left(a \cdot d(o_k^i, q)^2 + b \cdot d(o_k^i, q) + c\right).$$

Suppose f is the mean sensor node density. Then, on average, each grid cell would contain $f \cdot \alpha^2$ sensor nodes. Therefore, the expected cost of shrinking the monitoring area at interval i is estimated by

$$c_{\text{shrink}} = f \cdot \alpha^2 \cdot \Big((a \cdot (d(o_k^j, q) + R_s)^2 + b \cdot (d(o_k^j, q) + R_s) + c) - (a \cdot d(o_k^i, q)^2 + b \cdot d(o_k^i, q) + c)\Big).$$

Finally, we compute c_{query}, the cost of query reevaluations from interval g to h. We first estimate the number of grid cells visited during query reevaluation. Then, by estimating the message complexity of visiting a grid cell, we can derive the message complexity of query reevaluation. The grid cells visited in query reevaluation include those whose minimum distances to q lie between $d(o_k^i, q)$ and $r_{\text{old}} = d(o_k^j, q) + R_s$. As discussed above, the number of these grid cells is given by

$$\left(a \cdot (d(o_k^j, q) + R_s)^2 + b \cdot (d(o_k^j, q) + R_s) + c\right) - \left(a \cdot d(o_k^i, q)^2 + b \cdot d(o_k^i, q) + c\right).$$

The messages involved in visiting grid cells include the query messages transmitted between the R-nodes, the probe messages broadcast by the R-nodes, and the reply messages sent by the sensor nodes detecting objects. The number of probe messages is equivalent to the number of grid cells visited. The complexity of query messages, on the other hand, depends on the hop count between the R-nodes of two neighboring grid cells. Since the maximum distance between two neighboring grid cells is twice the transmission range, we approximate that an average of two hops are needed to transmit a query message between the R-nodes of two neighboring cells. The number of reply messages is ignored since only a small portion of sensor nodes would reply to the R-nodes. Hence, the total message complexity of query reevaluation is estimated by

$$c_{\text{query}} = 3 \cdot \left(\left(a \cdot (d(o_k^j, q) + R_s)^2 + b \cdot (d(o_k^j, q) + R_s) + c\right) - \left(a \cdot d(o_k^i, q)^2 + b \cdot d(o_k^i, q) + c\right)\right).$$

6 Performance evaluation

6.1 Experimental setup

We simulated continuous kNN query processing in sensor networks using a simulator called J-Sim [7,25]. We conducted a wide range of experiments to evaluate the performance of the localized scheme for monitoring kNN queries.

Table 1 summarizes the system parameters and their settings. We simulated a sensor network covering a 360 m × 360 m geographical area. A total of 2,500 sensor nodes were randomly deployed in the sensor network, implying that on average, there was one sensor node in an area of 50 m^2. Similar to other studies [5], the transmission range and the sensing range for each sensor node were set at 12.5 m and 10 m respectively. Each sensor node has an average of nine neighbors.

A given number of n objects were randomly distributed and tracked in the sensing field. The object locations were sampled by the sensor nodes at every 30 s. In our experiments, we assumed that the detecting sensor node of an object is the one closest to the object [36]. The objects were initially placed at random in the network. We simulated two different object mobility models: random waypoint and random walk. In the random waypoint model (abbreviated as RWP), each object repeatedly picks a random destination in the network and moves to the destination at a speed randomly chosen from a range $(0, V_{\max}]$. $V_{\max}$ varies from 1 to 10 m/s in our experiments. After reaching a destination, the object immediately chooses the next destination and moves towards it. In the random walk model (abbreviated as RW), each object periodically changes its moving speed by randomly choosing the velocity from a range $(0, V_{\max}]$ and the moving direction from a range $[0, 2\pi]$. The changing period in random walk model was set at 30 s.

We simulated the processing of continuous kNN queries in the network. The query points of the kNN queries were

Table 1 System parameters and settings

Parameters	Description	Value
N	Number of sensor nodes	2,500
R_t	Communication range	12.5 m
R_s	Sensing range	10 m
f	Sensor node density	1 node/50 m^2
$s \times s$	Size of sensor network	360 m × 360 m
n	Number of objects tracked	[50, 100, 200, 400, 600]
T	Sampling interval	30 s
$V_{\max}$	Maximum object moving velocity	[1, 5, 10 m/s]
k	Number of NNs required by a query	[1, 2, 4, 6, 8, 10]

Table 2 Message types and contents

Category	Type	Content
Query-related	Query	Query point q + k + search list + (0 to k) object locations
	Query result	k object locations
	Probe	Centroid of a grid cell + q + distance threshold
	Probe reply	Object location
	Shrink	Parameters of the target area + q
Location-update	Location update	Object location(s)

randomly generated in the sensing field. Each simulation run was performed for 2,000 sampling intervals. When visiting a grid cell, we used the contention-based scheduling scheme [32] to avoid collisions between the probe reply messages from the sensor nodes detecting objects. The time for each R-node to complete its data collection in the contention-based scheme was set at 0.018 s [29]. The performance is evaluated by four metrics: energy consumption, message complexity, query latency and query accuracy. Specifically, *energy consumption* is the average amount of energy consumed by all sensor nodes in the simulation. We focus on the energy consumption of message exchanges.[4] The power for sending and receiving messages was set at 60 and 45 mW, respectively [17]. *Message complexity* refers to the total number of messages transmitted in the network. There are six types of messages in the proposed localized scheme for kNN monitoring. As shown in Table 2, these messages can be divided into two categories: query-related messages and location-update messages. In our experiments, we assumed that each integer data value takes up 4 bytes in the message and each floating-point value takes up 8 bytes. *Query latency* measures the average time duration between the time when the query evaluation or reevaluation is initiated and the time when the query results are returned. Also, due to the query latency, the reported results may not be accurate because some objects may have moved from their last tracked location. *Query accuracy* is employed to measure the impact of object mobility on the accuracy of the query result. It is defined by $\frac{|S \cap S'|}{k}$, where S is the kNN result set obtained by the query sink, and S' is the actual kNN set at the time when the query results are returned.

We studied the localized scheme in depth by simulating five different strategies: *No-Monitoring-Area*, *OptimalShrink, AdaptiveShrink, NoShrink* and *AggressiveShrink*. In the No-Monitoring-Area strategy, no monitoring area is established in the network. Hence, no location update is sent to the query sink. The object locations are all stored locally at the detecting sensor nodes. At each sampling interval, the query is reevaluated from scratch using the two-phase method as described in Sects. 3.2–3.4. On the other hand, in the OptimalShrink, AdaptiveShrink, NoShrink and AggressiveShrink strategies, a monitoring area is established in the network. These four strategies use the same methods of query evaluation, monitoring area setup and query reevaluation as described in Sect. 4. The difference lies in the maintenance of monitoring area. The OptimalShrink strategy derives the optimal shrinking schedule in an offline manner as described in Sect. 5.1. The AdaptiveShrink strategy dynamically determines when to shrink the monitoring area on the fly as described in Sect. 5.2. The NoShrink strategy only expands the monitoring area at query reevaluation and never shrinks the monitoring area. The AggressiveShrink strategy, on the other hand, shrinks the monitoring area whenever the kth nearest object computed in the current sampling interval is closer to the query point than the kth nearest object in the previous sampling interval.

6.2 Impact of m in shrinking the monitoring area

Recall that in the first step of geocast, the notification message is sent to a number of m sensor nodes in the target area. The flooding then starts from these nodes. In this section, we investigate the impact of m on the *delivery rate* of geocast, where delivery rate is defined as the number of sensor nodes successfully receiving the message over the total number of nodes in the target area. We tested a wide range of target areas that are rings centered at the query point with inner and outer radius set at $i \cdot R_s$ and $(i + 1) \cdot R_s$ respectively ($1 \leq i \leq \lfloor \frac{s}{2\alpha} \rfloor$). The width of the target area was set at R_s since it is the narrowest possible width of target areas (representing the worst-case delivery rate in our application). Figure 9 shows the average delivery rate as a function of m. As can be seen, the delivery rate generally increases with m. It becomes rather steady when m increases beyond 4. Hence, the default value of m was set at four in our experiments.

6.3 Performance comparison of five strategies

In this section, we compare the performance of the five strategies OptimalShrink, AdaptiveShrink, NoShrink, AggressiveShrink and No-Monitoring-Area under different mobility models. We first simulated the processing of a single continuous kNN query. In this set of experiments, the number of objects was set at 200 and k was set at 8. The maximum moving speed for the objects was set at 5 m/s. For the OptimalShrink strategy, we first ran a set of separate

[4] Since this paper focuses on query processing, we do not include the energy consumed in tracking objects. As is the practice in other studies [21,32], we do not measure the routing protocol load placed by GPSR since GPSR generates a constant volume of routing protocol traffic (beacon messages) that is independent of the query processing scheme and is usually of lower order than the application data traffic.

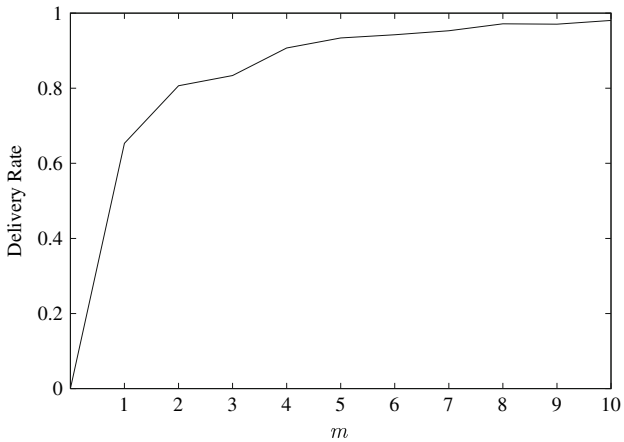


Fig. 9 Delivery rate versus number of unicast messages

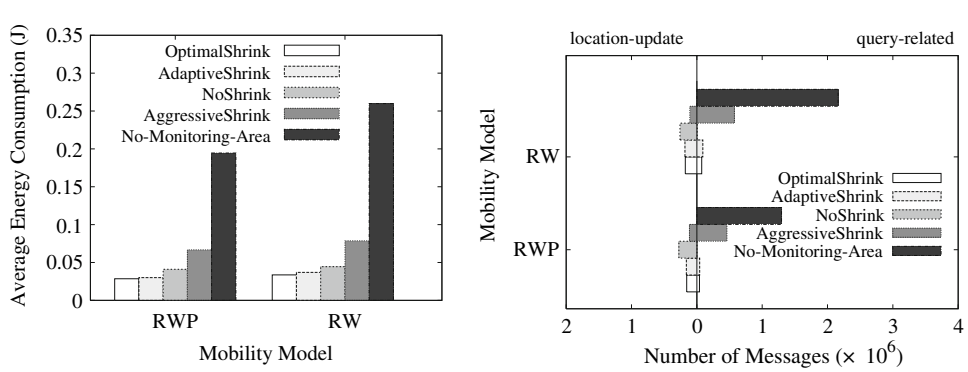
simulations to compute the costs $c_r(i, v)$, $c_q(i, v)$ and $c_s(i, v)$ for all pairs of i and v. These costs were then used to derive the optimal shrinking schedule as described in Sect. 5.1. Figure 10a shows the average energy consumption of the five strategies. It is seen that: (1) the energy consumption in the No-Monitoring-Area strategy is much higher than those strategies establishing a monitoring area in the network; (2) the NoShrink and AggressiveShrink strategies consume much more energy than the AdaptiveShrink strategy; (3) the average energy consumption of the AdaptiveShrink strategy is close to the OptimalShrink strategy.

To better understand these observations, we present the message breakdown for the five strategies in Fig. 10b. We observe that the No-Monitoring-Area strategy has the highest number of query-related messages and involves no location-update message. This is because in the No-Monitoring-Area strategy, no monitoring area is established to monitor the location updates. Hence, the query has to be reevaluated from scratch at each interval. For the other four strategies, a monitoring area is set up to monitor the location updates that contribute to query reevaluation. Thus, the number of query-related messages is greatly reduced.

Among the four strategies establishing monitoring areas, the AggressiveShrink strategy has the lowest number of location-update messages but the highest number of query-related messages. This is because the monitoring area in AggressiveShrink is always shrunk to a minimum circle covering the k nearest objects and their detecting sensor nodes. In this way, the location updates are kept at minimum. However, aggressive shrinking of monitoring area may lead to frequent query reevaluation when the objects move away from the query point. Thus, the AggressiveShrink strategy has the highest number of query-related messages among the four strategies. In contrast, the NoShrink strategy has the lowest number of query-related messages. This is because the monitoring area is never shrunk in NoShrink. Thus, the possibility of carrying out preliminary and expanded searches in query reevaluation is low. However, the NoShrink strategy leads to a large number of unnecessary location updates. Hence, it has the highest number of location-update messages among the four strategies. The AdaptiveShrink strategy makes a good balance between query-related and location-update messages by considering the tradeoff between the shrinking overhead and the saving in location updates. It shrinks the monitoring area only when it is beneficial. As a result, the AdaptiveShrink strategy achieves lower total message complexity than AggressiveShrink and NoShrink. Figure 10 shows that its performance is close to the offline OptimalShrink strategy.

Figure 11 shows the distribution of energy consumption among all sensor nodes in the network for processing a single continuous kNN query. A point (x, y) on the curve means that a fraction x of all sensor nodes consume more than y J energy each. The energy consumption for the top 10% nodes in the No-Monitoring-Area strategy is much higher than the top 10% nodes in the other strategies. This is because many nodes are involved in transmitting query-related messages at each interval. Compared to the AggressiveShrink, OptimalShrink and AdaptiveShrink strategies, the energy consumption in the NoShrink strategy is highly unbalanced. This is due to the large number of location updates sent to the query sink. The sensor nodes surrounding the query sink have to relay

Fig. 10 Performance of localized scheme (single query)



(a) Average Energy Consumption

(b) Message Breakdown

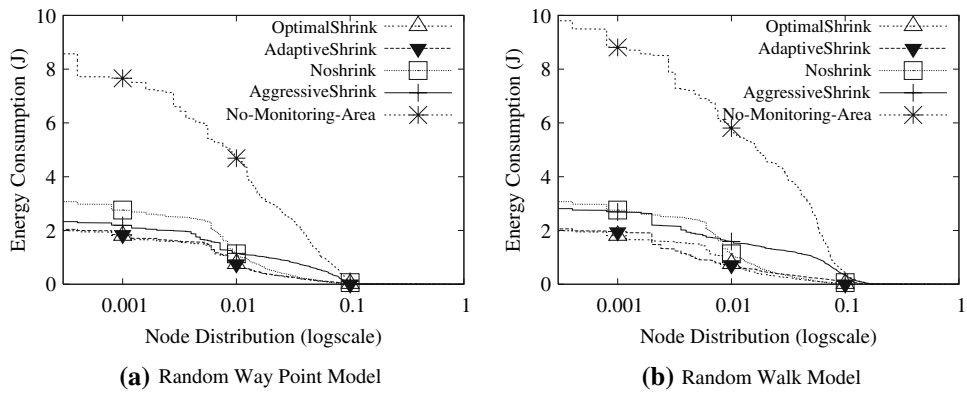


Fig. 11 Distribution of energy consumption of localized scheme with single continuous query

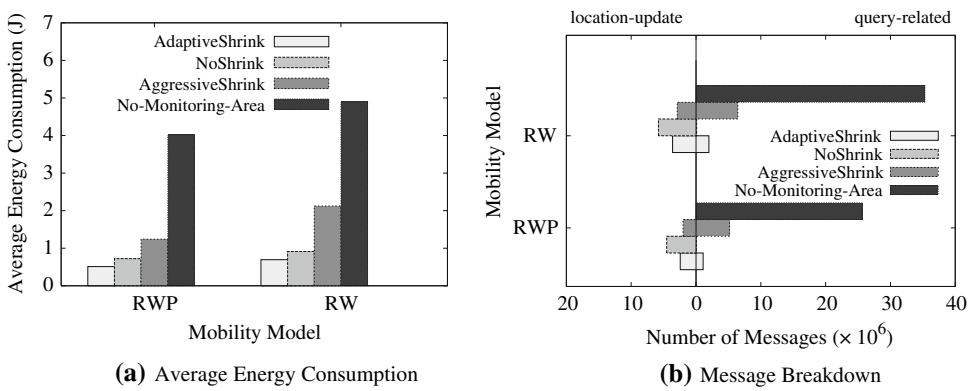


Fig. 12 Performance of localized scheme (multiple queries)

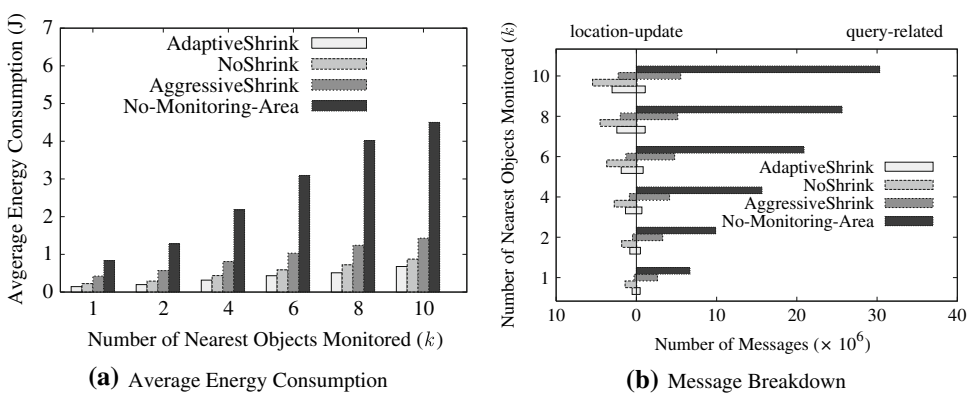


Fig. 13 Impact of k (random waypoint model)

many location updates. Similar to the No-Monitoring-Area strategy, the energy consumption for the AggressiveShrink strategy is also high among the top 10% sensor nodes. This is because query reevaluation and shrinking the monitoring area both increase the workload of all sensor nodes close to the query sink. The AdaptiveShrink strategy achieves a similar energy distribution to the OptimalShrink strategy. The energy consumption is lower and more balanced in these two strategies than in the AggressiveShrink and NoShrink strategies.

We also simulated the scenario where 20 kNN queries were monitored concurrently. Figure 12 shows the performance results of the four strategies AdaptiveShrink, Noshrink, AggressiveShrink and No-Monitoring-Area. It is seen that the performance trends of the four strategies are similar to those of monitoring a single kNN query in the network (Fig. 10).

6.4 Impact of the number of nearest objects monitored

Figures 13 and 14 show the average energy consumption and the message breakdown for different numbers of nearest objects monitored. In this set of experiments, the number of objects was set at 200 and $V_{\max}$ was set at 5 m/s. A total

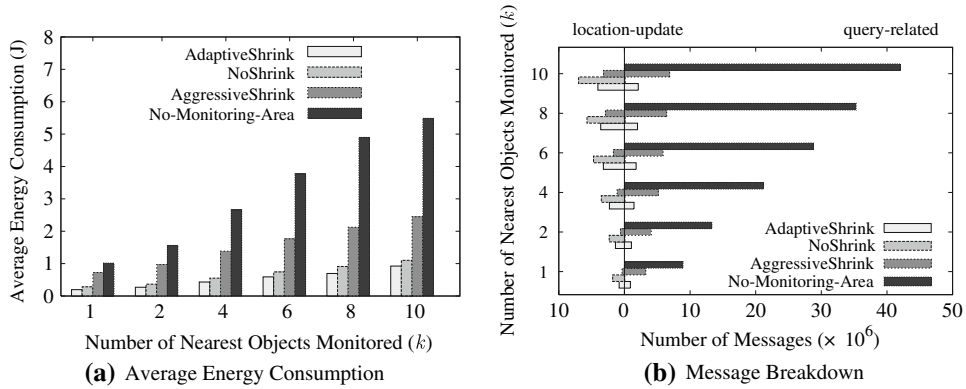


Fig. 14 Impact of k (random walk model)

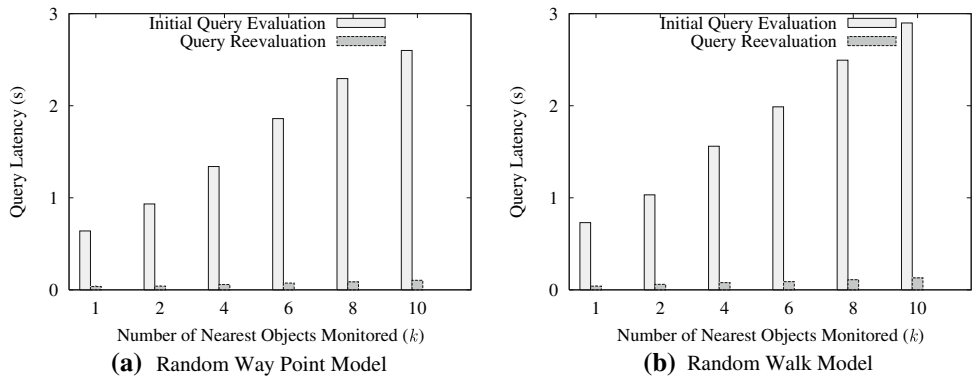
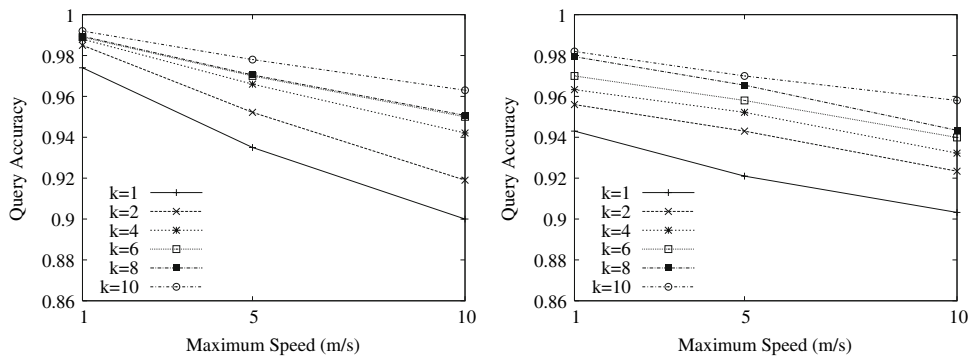


Fig. 15 Query latency of adaptive shrink strategy

number of 20 kNN queries were concurrently monitored in the network. It is seen from Figs. 13a and 14a that the average energy consumption increases with k for all four strategies. This is because when the requested number of nearest objects increases, more sensor nodes are visited in query evaluation and reevaluation. Thus, as shown in Figs. 13b and 14b, the number of query-related messages increases with k. Moreover, for the AdaptiveShrink, NoShrink and AggressiveShrink strategies, a larger monitoring area is established when more nearest objects are requested. As a result, more location updates are sent to the query sink at each sampling interval. Figs. 13b and 14b show that the number of location-update messages in these four strategies also increases with k. As k increases, the energy consumption of NoShrink grows rapidly due to large number of location updates. Similarly, with increasing k, the energy consumption of AggressiveShrink rises quickly due to high cost of query reevaluation. The AdaptiveShrink strategy balances the costs in query reevaluation and location updates. Hence, its energy consumption increases mildly when k becomes larger.

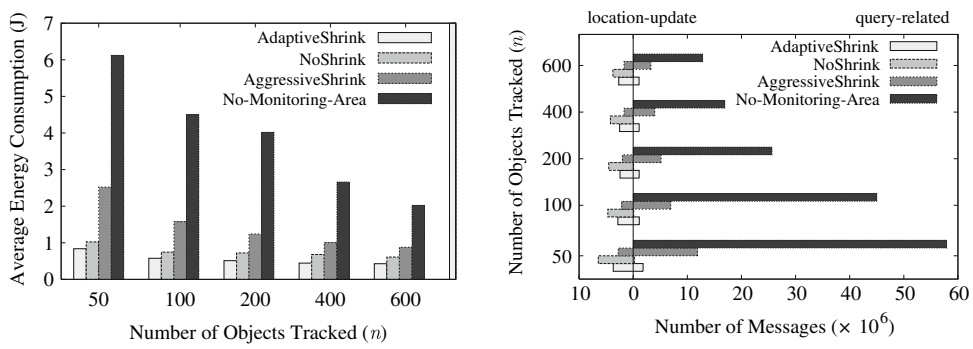
Figure 15 shows the latency of initial query evaluation and query reevaluation for the AdaptiveShrink strategy as a function of different numbers of nearest objects monitored. It is shown that the query latency increases when more nearest objects are monitored in the network. This is because more grid cells are visited to derive the k nearest objects. As shown in Fig. 15, the latency of initial query evaluation in this set of experiments is less than 3 s and the latency of query reevaluation is lower than 0.15 s. The query reevaluation is much faster than initial query evaluation because the location updates received from the sensor nodes in the monitoring area are used to reevaluate the query. Hence, the visit to the grid cells in the monitoring area are exempted.

Figure 16 shows the query accuracy of the AdaptiveShrink strategy for different numbers of nearest objects monitored as a function of $V_{\max}$. As expected, query accuracy generally decreases when the objects move faster. This is because the object moves a longer distance during the query evaluation or reevaluation. It is also seen from Fig. 16 that under the same object moving speed, query accuracy improves when more nearest objects are monitored. This is because query accuracy is normalized by the size of the kNN result set. When the number of nearest objects monitored is large, an error in the kNN result set has less impact on query accuracy. Figure 16 shows that under both mobility models, the AdaptiveShrink strategy achieves higher than 90% query accuracy.



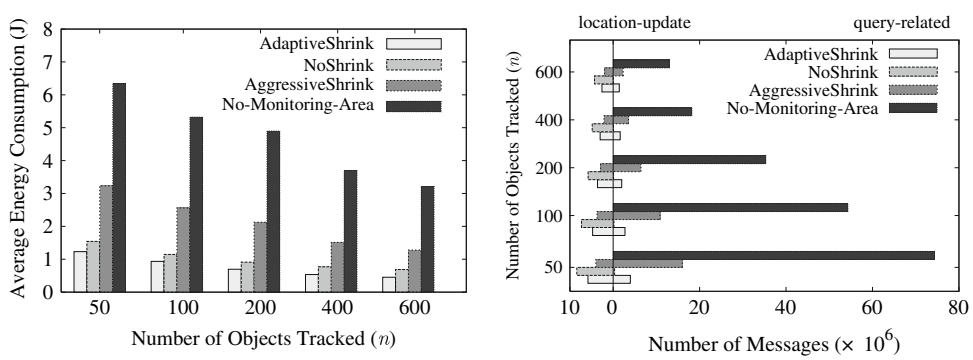
(a) Random WayPoint Model **(b)** Random Walk Model

Fig. 16 Query accuracy of adaptive shrink strategy



(a) Average Energy Consumption **(b)** Message Breakdown

Fig. 17 Impact of number of objects (random waypoint model)



(a) Average Energy Consumption **(b)** Message Breakdown

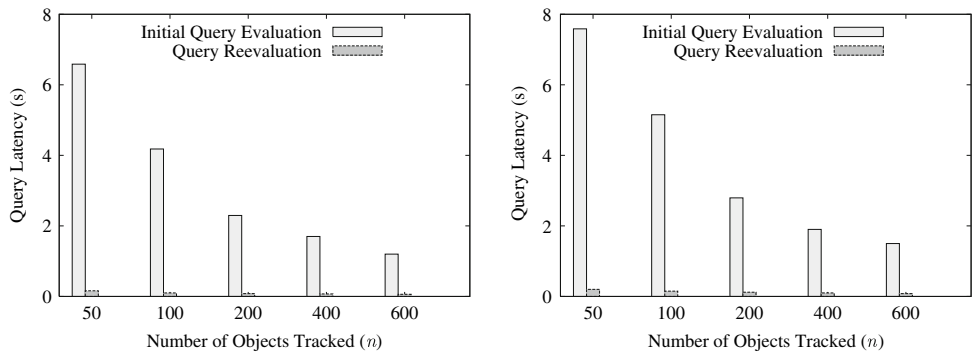
Fig. 18 Impact of Number of Objects (random walk model)

6.5 Impact of number of objects tracked

Figures 17 and 18 show the average energy consumption and the message breakdown for different numbers of objects tracked in the network. In this set of experiments, $V_{\max}$ was set at 5 m/s and k was set at 8. Again, a total number of 20 kNN queries were concurrently monitored in the network. It is observed that the average energy consumption decreases with increasing number of objects in the network. This is because when there are more objects in the network, the kth nearest object becomes closer to the query point. Hence, the message complexity of query reevaluation decreases (see Figs. 17b, 18b). The size of the monitoring area in the AdaptiveShrink, NoShrink and AggressiveShrink strategies also reduces with increasing number of objects tracked. The relative performance of the four strategies remains similar over a wide range of object numbers. Under both mobility models, the AdaptiveShrink strategy outperforms the AggressiveShrink and NoShrink strategies.

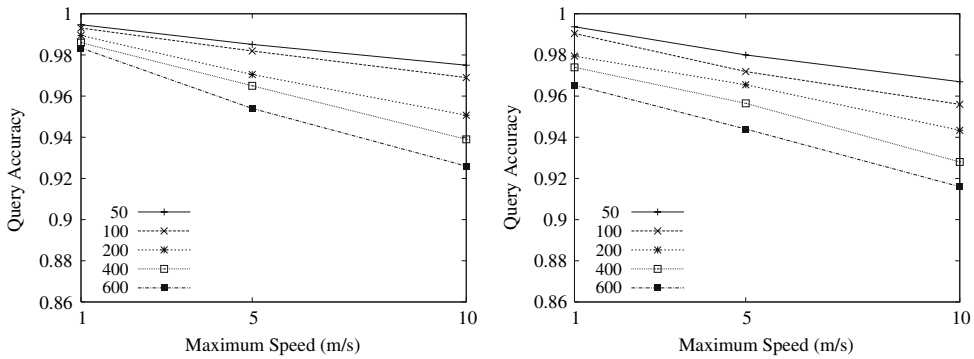
Figure 19 shows the latency of initial query evaluation and query reevaluation for the AdaptiveShrink strategy as a function of the number of objects tracked in the network. It is shown that the query latency decreases when more objects are tracked in the network. This is because fewer grid cells are visited to derive the k nearest objects. As shown in Fig. 19, the latency of initial query evaluation in this set of experiments

Fig. 19 Query latency of adaptive shrink strategy



(a) Random WayPoint Model **(b)** Random Walk Model

Fig. 20 Query accuracy of adaptive shrink strategy



(a) Random WayPoint Model **(b)** Random Walk Model

is less than 8 s and the latency of query reevaluation is lower than 0.2 s.

Figure 20 shows the query accuracy of the AdaptiveShrink strategy for different numbers of objects tracked as a function of $V_{\max}$. Similar to Figs. 16 and 20 shows that query accuracy decreases when the objects move faster. It is also seen that under the same object moving speed, query accuracy decreases with increasing the number of objects tracked. This is because when more objects are tracked in the sensing field, a change in the object location has a greater impact on the kNN result. Under both mobility models, the AdaptiveShrink strategy achieves higher than 90% query accuracy.

7 Conclusion

In this paper, we have proposed a localized scheme for continuous kNN query processing in object tracking sensor networks. In the localized scheme, the object locations are stored locally at the detecting sensor nodes. A monitoring area is set up when the kNN query is initially evaluated. Only the location updates from sensor nodes in the monitoring area are collected to reevaluate the query. The monitoring area is expanded and shrunk on the fly upon object movement. Experimental results show that the cost of query reevaluation is greatly reduced by establishing the monitoring area in the localized scheme. We have further studied the maintenance of the monitoring area. We analyze the optimal maintenance and develop an adaptive algorithm to dynamically decide when to shrink the monitoring area. Experimental results show that the AdaptiveShrink strategy achieves close-to-optimal performance and significantly outperforms the naive NoShrink and AggressiveShrink strategies in terms of energy consumption and message complexity. The AdaptiveShrink strategy also yields a good performance in query latency and query accuracy under various experimental settings.

References

1. Aslam, J., Butler, Z., Constantin, F., Crespi, V., Cybenko, G., Rus, D.: Tracking a moving object with a binary sensor network. In: Proceedings of Sensys (2003)
2. Diao, Y., Ganesan, D., Mathur, G., Shenoy, P.: Rethinking data management for storage-centric sensor networks. In: Proceedings of CIDR 2007, January 2007
3. Gedik, B., Liu, L.: Mobieyes: distributed processing of continuously moving queries on moving objects in a mobile system. In: Proceedings of EDBT'04. Heraklion, March 2004
4. Hoffmann-Wellenhof, B., Lichtenegger, H., Collins, J.: GPS Theory and Practice. Springer, New York (1997)
5. He, T., Vicaire, P.A., Yan, T., Luo, L., Gu, L., Zhou, G., Stoleru, R., Cao, Q., Stankovic, J.A., Abdelzaher, T.: Achieving real-time target tracking using wireless sensor networks. ACM Trans. Embed. Comput. Syst. (TECS) (2007)

6. Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed diffusion: a scalable and robust communication paradigm for sensor networks. In: Proceedings of Mobicom (2000)
7. J-sim homepage. http://www.j-sim.org
8. Karp, B., Kung, H.T.: GPSR: Greey perimeter stateless routing for wireless networks. In: Proceedings of Mobicom, Boston, August 2000
9. Lee, W.-C., Xu, Y., Winter, J.: Prediction-based strategies for energy saving in object tracking sensor networks. In: Proceedings of MDM'04, Berkeley, January 2004
10. Li, D., Wong, K.D., Hu, Y.H., Sayeed, A.M.: Detection, classification and tracking of targets in distributed sensor networks. IEEE Signal Process. Mag. **19**(2), March (2002)
11. Madden, S., Franklin, M.J., Hellestein, J.M., Hong, W.: TAG: a tiny aggregation service for ad-hoc sensor networks. In: Proceedings of OSDI'02, Boston, December 2002
12. Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D.: Wireless sensor networks for habitat monitoring. In: Proceedings of the 1st ACM Workshop on Sensor Networks and Applications, pp. 88–97, Atlanta, September 2002
13. Mouratidis, K., Hadjieleftheriou, M., Papadias, D.: Conceptual partitioning: an efficient method for continous nearest neighbor monitoring. In: Proceedings SIGMOD'05, Baltimore, June 2005
14. Mouratidis, K., Papadias, D., Bakiras, S., Tao, Y.: A threshold-based algorithm for continuous monitoring of k nearest neighbors. IEEE Trans. Knowl. Data Eng. (TKDE) **17**(11), 1451–1464 (2005)
15. Navas, J.C., Imielinski, T.: GeoCast — geographic addressing and routing. In: ACM/IEEE MobiCom (1997)
16. Niculescu, D., Nathi, B.: Ad hoc positioning system (aps) using aoa. In: Proceedings of INFOCOM'03, San Francisco, 2003
17. Polastre, J., Hill, J., Culler, D.: Versatile low power media access for wireless sensor networks. In: SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor System, pp. 95–107 (2004)
18. Pottie, G.: Wireless integrated network sensors. Commun. ACM **43**(5), 51–58 (2000)
19. Prabhakar, S., Xia, Y., Kalashnikov, D., Aref, W., Hambrusch, S.: Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. IEEE Trans. Comput. **51**(10), 1124–1140 (2002)
20. Ratnasamy, S., Karp, B., Yin, L., Yu, F., Estrin, D., Govindan, R., Shenker, S., Ght: a geographic hash table for data-centric storage. In: Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications, Atlanta, September 2002
21. Ratnasamy, S., Karp, B., Shenker, S., Estrin, D., Govindan, R., Yin, L., Yu, F.: Data-centric storage in sensornets with ght, a geographic hash table. Mobile Netw. Appl.(MONET) **8**(4), 427–442 (2003)
22. Sharaf, M.A., Beaver, J., Labrinidis, A., Chrysanthis, P.K.: TiNA: a scheme for temporal coherency-aware in-network aggregation. In: Proceedings of the 3rd International ACM Workshop on Data Engineering for Wireless and Mobile Access, pp. 69–76, San Diego, September 2003
23. Silberstein, A., Braynard, R., Yang, J.: Constraint chaining: on energy-efficient continuous monitoring in sensor networks. In: Proceedings of SIGMOD'06, Chicago, June 2006
24. Silberstein, A., Munagala, K., Yang, J.: Energy-efficient monitoring of extreme values in sensor networks. In: Proceedings of SIGMOD'06, Chicago, June 2006
25. Sobeih, A., Hou, J.C., Kung, L.-C., Li, N., Zhang, H., Chen, W.-P., Tyan, H.-Y., Lim, H.: J-sim: a simulation and emulation environment for wireless sensor networks. IEEE Wirel. Commun. **13**(4), 104–119 (2006)
26. Stojmenovic, I.: Geocasting with guaranteed delivery in sensor networks. IEEE Wirel. Commun. **11**(6), 29–37 (2004)
27. Winter, J., Xu, Y., Lee, W.-C.: Energy efficient processing of k nearest neighbor queries in location-aware sensor networks. In: Proceedings of the Second International Conference on Mobile and Ubiquitous Systems: Networking and Services (Mobiquitous'05). San Diego, July 2005
28. Wu, M., Xu, J., Tang, X., Lee, W.-C.: Monitoring top-k query in wireless sensor networks. In: Proceedings of ICDE'06, Atlanta, April 2006
29. Wu, S.-H., Chuang, K.-T., Chen, C.-M., Chen, M.-S.: Diknn: an itinerary-based knn query processing algorithm for mobile sensor networks. In: Proceedings of ICDE'07, Istanbul, April 2007
30. Xiong, X., Mokbel, M., Aref, W.: SEA-CNN: scalable incremental processing of continuous queries in spatio-temporal databases. In: Proceedings of ICDE'05, Tokyo, April 2005
31. Xu, J., Tang, X., Lee, W.-C.: EASE: an energy-efficient in- network storage scheme for object tracking in sensor networks. In: Proceedings of IEEE SECON'05, Santa Clara, September 2005
32. Xu, Y., Lee, W.-C., Xu, J., Mitchell, G.: Processing window queries in wireless sensor networks. In: Proceedings of ICDE'06, Atlanta, April 2006
33. Yao, Y., Tang, X., Lim, E.-P.: In-network processing of nearest neighbor queries for wireless sensor networks. In: Proceedings of DASFAA'06, Singapore, April 2006
34. Yao, Y., Tang, X., Lim, E.-P.: Continuous monitoring of kNN queries in wireless sensor networks. In: Proceedings of the 2nd International Conference on Mobile Ad-hoc and Sensor Networks (MSN 2006), Hong Kong, December 2006
35. Yu, X., Pu, K., Koudas, N.: Monitoring k-nearest neighbor queries over moving objects. In: Proceedings of ICDE'05, Tokyo, April 2005
36. Zhang, W., Cao, G.: Optimizing tree reconfiguration for mobile target tracking in sensor networks. In: Proceedings of INFOCOM'04, Hong Kong, March 2004
37. Zhao, F., Shin, J., Reich, J.: Information-driven dynamic sensor collaboration for tracking applications. IEEE Signal Process. **19**(2), 61–72 (2002)