Theoretical Computer Science 608 (2015) 146–165

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Optimal nearest neighbor queries in sensor networks ☆

Gokarna Sharma *, Costas Busch [1]

School of Electrical Engineering and Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA

ARTICLE INFO

Article history:
Received 3 March 2014
Received in revised form 14 January 2015
Accepted 6 May 2015
Available online 13 May 2015

Keywords:
Sensor networks
Mobile objects
Nearest neighbor queries
Hierarchical structure
Object tracking
Competitive ratio

ABSTRACT

Given a set of m mobile objects in a sensor network, we consider the problem of finding the nearest object among them from any node in the network at any time. These mobile objects are tracked by nearby sensors called proxy nodes. This problem requires an object tracking mechanism which typically relies on two basic operations: *query* and *update*. A query is invoked by a node each time when there is a need to find the closest object from it in the network. Updates of an object's location are initiated when the object moves from one location (proxy node) to another. We present a scalable distributed algorithm for tracking these mobile objects such that both the query cost and the update cost are small. The main idea in our algorithm is to maintain a virtual tree of downward paths pointing to the objects. Our algorithm guarantees an asymptotically optimal $\mathcal{O}(1)$ approximation for *query* cost and an $\mathcal{O}(\min\{\log n, \log D\})$ approximation for *update* cost in the constant-doubling graph model, where n and D, respectively, are the number of nodes and the diameter of the network. We also give polylogarithmic approximations for both *query* and *update* cost in the general graph model. Our algorithm requires only polylogarithmic bits of memory per node. To the best of our knowledge, this is the first algorithm that is asymptotically optimal in handling nearest neighbor queries with low update cost in a distributed setting.

1. Introduction

In sensor networks, object tracking is an important application where the presence of particular objects (animals, vehicles, etc.) can be detected by nearby sensors [2–16]. This application is interesting and arises in a setting where a system of static sensor nodes and a set of mobile objects are working in the same physical plane [12]. We consider in this paper the *nearest neighbor query* problem, denoted as NNQ, in which the objective is to track the mobile objects in such a way that any node in the network can locate the nearest object, among the set of objects available in the network, from it at any time with the minimum cost possible. By locating the nearest object we mean the finding of the closest sensor node that has (i.e., detects) the object. For an example, consider a sensor network tracking the movement of taxis (as objects). A pedestrian injects queries into the network from his location, by sending them through nearby sensor nodes using devices, such as PDAs, for the nearest taxi. The data will then be extracted from the relevant sensor nodes to respond to the

☆ A preliminary version of this paper appears in the Proceedings of ALGOSENSORS'2013 [1].

* Corresponding author. Tel.: +1 225 578 1495; fax: +1 225 578 1465.
E-mail addresses: gokarna@csc.lsu.edu (G. Sharma), busch@csc.lsu.edu (C. Busch).
[1] Tel.: +1 225 578 7510.

http://dx.doi.org/10.1016/j.tcs.2015.05.006
0304-3975/

user. Note that NNQ is along the lines of the in-network data processing problem for object tracking studied in, e.g. [17,2], and different from the target tracking problem to approximate the trajectory of moving objects studied in, e.g. [18,19].

We model the problem of tracking mobile objects by an edge weighted graph G, where graph nodes correspond to sensor nodes and graph edges correspond to communication links between the sensor nodes. Moreover, we assume that each sensor node has its own memory space where the objects information reside. A sensor node that currently detects a mobile object is called the *proxy node* for that object at that time. We consider a *nearest-sensor* model such that the sensor node that is near to the object becomes its proxy [20,3], breaking ties arbitrarily. In this model, a sensor node that receives the strongest signal from the object becomes the proxy node for that object. To accomplish this task, we can partition the sensing field into a Voronoi graph such that every point in a polygon of that graph is closer to its corresponding sensor in that polygon than to any other. This partitioning of the sensing field into a Voronoi graph can be achieved using the technique presented in Chen et al. [20]. Now if the mobile object is at any point inside that polygon, then the corresponding sensor at that polygon becomes its proxy. If the mobile object is at the boundary of the polygon in the Voronoi graph, then the sensors that share that common boundary (they are in fact neighbors) communicate and decide which one will become the proxy for that object. The communication range of sensors is assumed to be large enough so that neighboring sensors can communicate directly to each other. The proxy nodes may change over time when objects move. We sometimes refer to sensor nodes as "users" since queries are injected into the network from sensor nodes nearby to the users' current location; for example, a pedestrian injecting queries into the network for the nearest taxi. In NNQ, the objective is to find the nearest proxy node.

Object tracking in NNQ involves two basic operations: *query* and *update*. A *query* operation is invoked when a user is looking for the location of the nearest object (i.e., the nearest proxy node). An *update* operation is initiated when the object moves from one node to another, i.e., it updates the location of the objects. Location queries and updates may be done in various ways. A naive way to query an object is to flood the whole network. All proxy nodes will reply to the query and the user which issued the request can choose the nearest one among them. Clearly, this approach is inefficient due to energy consumed when the network is large or the query rate is high [3]. Alternatively, if all location information is stored at a specific node (e.g., the sink), no flooding is needed. But, whenever a movement is detected, update messages have to be sent up to that specific node all the time, which might be a major bottleneck. Moreover, when objects move frequently, abundant update messages will be generated. Therefore, we are interested in an approach for NNQ where *query* operations are not required to be flooded and *update* operations are not always required to be sent up to a sink. We assume that each node can issue *query* operations at any time, query rate is frequent, and multiple queries from different nodes can exist.

1.1. Cost model

Consider a set of m mobile objects in a sensor network G. We assume that the cost to send messages between any pair of nodes in G depends only on the distance (shortest path length) between them.

- **Query cost:** We measure the cost of a *query* operation with respect to the *communication cost*, which is the total number of messages sent in the network G by a NNQ algorithm to reach the closest proxy node from any requesting node. The optimal communication cost for any *query* operation is the distance in G between the requesting node and its nearest proxy node. We compare the communication cost of a NNQ algorithm for a *query* operation to the optimal communication cost for that operation to obtain the *query competitive ratio*, which expresses the cost approximation of the NNQ algorithm for queries.
- **Update cost:** When an object moves from one proxy node to another proxy node, the optimal communication cost to update the location of that object is at least the distance between these proxy nodes. This is because any algorithm for NNQ needs to send messages at least equivalent to this distance far to reflect the location change in its NNQ data structure. We compare the communication cost of a NNQ algorithm for a set of *update* operations to the optimal communication cost for that set of operations to obtain the *update competitive ratio*, which expresses the cost approximation of the NNQ algorithm for updates.

We look for a class of algorithms for NNQ that hold following two properties:

1. NNQ can be answered with the cost that is proportional to the shortest distance to the closest proxy node from the requesting node in G, and
2. The data structure needed for answering NNQ can be maintained with the cost that is proportional to the minimum distance the objects traverse if they followed shortest paths in G.

We present a NNQ algorithm in this work which is optimal for query cost and near-optimal for update cost.

Existing techniques focused on aggregation and join queries, e.g. [21,22], and not much work has been done on nearest neighbor queries in object tracking [23]. Previous work on nearest neighbor queries has focused only on finding the nearest sensor nodes to a specific query point [24,25]. This is different from our objective of finding the nearest mobile object from any sensor node at any time. Authors in [2–6] focused on tracking mobile objects but queries are only supported from a specific sink node. These papers [10,13,26,9,27] tried to solve the NNQ problem but with only one object; for multiple