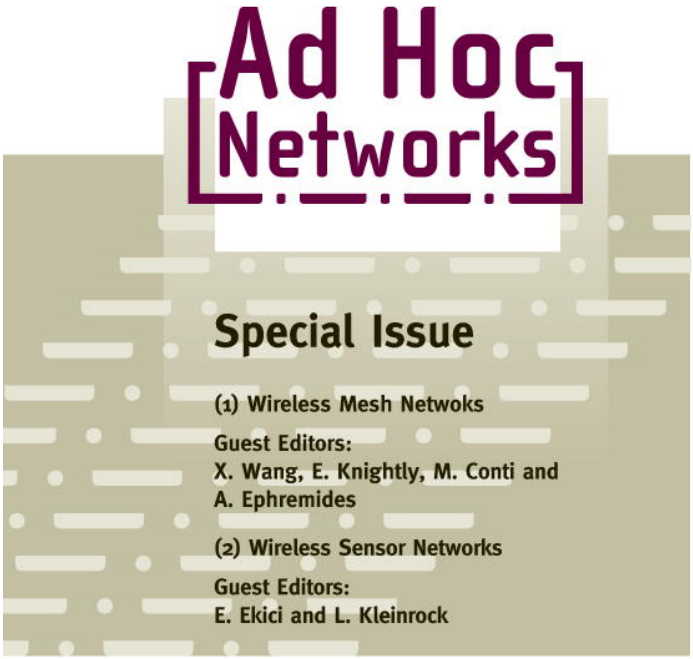
Volume 5, Issue 6 August 2007 ISSN 1570-8705



Available online at www.sciencedirect.com
ScienceDirect
Ad Hoc Networks 5 (2007) 929–942
Ad Hoc Networks
www.elsevier.com/locate/adhoc

Communication-efficient implementation of join in sensor networks

Himanshu Gupta *, Vishal Chowdhary

Department of Computer Science, Stony Brook, NY 11794, United States

Received 10 February 2007; accepted 15 February 2007
Available online 24 February 2007

Abstract

A sensor network is a multi-hop wireless network of sensor nodes cooperatively solving a sensing task. Each sensor node generates data items that are readings obtained from one or more sensors on the node. This makes a sensor network similar to a distributed database system. While this view is somewhat traditional, efficient execution of database (SQL) queries in sensor network remains a challenge, due to the unique characteristics of such networks such as limited memory and battery energy on individual nodes, multi-hop communication, unreliable infrastructure, and dynamic topology. Since the nodes are battery powered, the sensor network relies on energy-efficiency (and hence, communication efficiency) for a longer lifetime of the network.

In this article, we have addressed the problem of communication-efficient implementation of the SQL "join" operator in sensor networks. In particular, we design an optimal algorithm for implementation of a join operation in dense sensor networks that provably incurs minimum communication cost under some reasonable assumptions. Based on the optimal algorithm, we design a suboptimal heuristic that empirically delivers a near-optimal join implementation strategy and runs much faster than the optimal algorithm. Through extensive simulations on randomly generated sensor networks, we show that our techniques achieve significant energy savings compared to other simple approaches.

Keywords: Sensor network databases; Join implementation; Communication efficiency

1. Introduction

A sensor network consists of sensor nodes with a short-range radio and on-board processing capability forming a multi-hop network of an irregular topology. Each sensor node can sense certain physical phenomena like light, temperature, or vibration. There are many exciting applications of such sensor networks, including monitoring and surveillance systems in both military and civilian contexts, building smart environments and infrastructures such as intelligent transportation systems and smart homes.

Each sensor node typically generates a stream of data items that are readings obtained from one or more sensing devices on the node. This motivates visualizing sensor networks as distributed database systems [8,15,18] and the data present in a sensor

* Corresponding author. Tel.: +1 631 632 8446; fax: +1 631 632 8334.
E-mail addresses: hgupta@cs.suny.edu (H. Gupta), vishalc@microsoft.com (V. Chowdhary).

1570-8705/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.adhoc.2007.02.002

network as relational data streams. Like a database, the sensor network is queried to gather the sensed data tuples. Database queries in SQL are a very general representation of queries over data, and because of the enormous amount of data present in a typical sensor network, efficient implementation of database queries is of great significance.

The main performance criterion for distributed implementations of queries in sensor network is the total communication cost incurred, since each sensor node has limited battery power and message communication nodes is the main consumer of battery energy. Thus, distributed implementation of queries must minimize the communication cost incurred. In particular, we are interested in in-network implementation strategies since a centralized strategy of transmitting all sensor data to a central server for further computation would incur prohibitive communication costs.

In this article, we focus on designing efficient distributed implementations for the join operation in sensor networks. The join operator is essentially a cartesian product of the operand tables followed by a predicate selection. The motivation for the join operation in sensor networks comes from one of the most prominent sensor network applications viz., event detection, wherein complex events can be defined as joins over data streams [2,11]. We propose a novel path-join algorithm, which computes the join result by first distributing one of the operand tables along a predetermined path of sensors. Using path-join algorithm as the basic step, we design an *optimal* algorithm for a join operation that provably incurs minimum communication cost in dense sensor networks under some reasonable assumptions of communication cost and computation model. We also design a much faster suboptimal heuristic that empirically performs very close to the optimal algorithm, and results in significant savings over the naive approaches.

1.1. Paper organization

The rest of the paper is organized as follows. We start with modeling the sensor network as a database and motivating implementation of the join operation in the sensor network. In Section 3, we present various algorithms for in-network implementation of the join operator for static (non-streaming tables). In Section 4, we generalize our techniques to handle streaming tables and discuss relaxation of other assumptions. We present our experiment results in Section 5. Related work is discussed in Section 6, and concluding remarks presented in Section 7.

2. Sensor network databases

A sensor network consists of a large number of sensors distributed randomly in a geographical region. Each sensor has limited on-board processing capability and is equipped with sensing devices. We assume that each sensor node is aware of its geographic location (obtained using GPS or other localization techniques [5]). A sensor node also has a radio which is used to communicate directly with some of the sensors around it. Two sensor nodes can communicate with each other if and only if the distance between them is less than the *transmission radius*. We assume that each sensor node in the sensor network has a limited storage capacity of m units. As mentioned above, each sensor node has limited battery energy, which must be conserved for prolonged unattended operation. Thus, we have focused on minimization of communication cost (hence, energy cost) as the key performance criteria of the join implementation strategies.

2.1. Modeling the sensor network as a database

In a sensor network, the data generated by the sensor nodes is simply the readings obtained from the sensing devices on the node. The data records produced by a group of sensor nodes with similar capabilities and responsibility will have similar format and semantics, and thus, can be modeled as rows of the same relational table. More specifically, due to the continuous generation of data tuples in the sensor network, the sensor network data is best modeled as *data streams* [3]. The above motivates visualizing sensor networks as distributed database systems [8,15,18] of streaming tables. In a sensor network, a data stream may be partitioned horizontally across (or generated by) a set of sensors in the network. Each data stream has a corresponding generating region which could very well be the entire network region. Due to the spatial and real-time nature of the data generated, a tuple usually has `timeStamp` and `nodeLocation` as attributes, and the sensor node that generates a particular tuple is referred as its *source node*. Like traditional database systems, the sensor network database can also be queried to access and manip-

ulate the data tables, and SQL with some extensions can be used as a query language for sensor networks.

2.1.1. Database queries

A database query is composed of one or more database operators. The core database operators are viz. selection (selecting tuples based on a predicate), projection (selecting given attributes of a table), join (cartesian product followed by selection), grouping (partitioning a table based on a set of attribute values), aggregation (aggregating attributes for each group), outerjoins (join plus the unmatched tuples padded with NULLs), duplicate elimination, union, difference, and intersection. Union, difference, and intersection have same semantics as the corresponding set operators.

The focus of this article is communication-efficient in-network implementation of the join operator. The join operator is used to correlate data from multiple tables and is essentially a cartesian product of the operand tables followed by a selection. As selection and projection are unary operators and operate on each tuple independently, they could be implemented by computing the operation locally followed by efficiently routing to the query source. Union operation can be reduced to duplicate elimination, and the difference and intersection operations can be reduced to the join operation. Implementation of other database operators (aggregation, duplicate elimination, and outerjoins) is challenging and is part of our future work.

2.1.2. In-network implementation of SQL queries

A plausible implementation of a sensor network database query engine could be to have an external database system handle all the queries over the network. In such a realization, all the data from each sensor node in the network is sent to the external system that handles the execution of queries completely. Such an implementation would incur very high communication costs and congestion-related bottlenecks. Thus, prior research has proposed query engines that would execute the queries within the network with little external help. In particular, [9] shows that in-network implementation of database queries is fundamental to achieving energy-efficient communication in sensor networks. Moreover, due to the very limited processing memory available on a sensor nodes, it will be impossible to compute the join locally on any particular node, especially for large tables.

2.1.3. Querying and cost model in sensor networks

A query in a sensor network is initiated at a node called *query source* and the result of the query is required to be routed back to the query source for storage and/or consumption. A stream database table may be generated by a set of sensor nodes in a closed geographical region. The optimization algorithms, proposed in this article, to determine how to implement the join operation efficiently, are run at the query source. As typical sensor network queries are long running, the query source can gather all the catalogue information needed (estimated sizes and locations of the operand relations, join selectivity factor to estimate the size of the join result, density of the network) by initially sampling the operand tables. As mentioned before, we concentrate on implementations that minimize communication cost. We define the total communication cost incurred as the total data transfer between neighboring sensor nodes.

Our algorithms target the general long-running queries in the sensor network. Given a query source Q and regions $\mathscr{R}$ and $\mathscr{S}$ where a join has to be taken. Initially all the tuples of the participating tables are routed to the query source Q, which collects catalog information and estimates parameters such as locations of the region $\mathscr{R}$ and $\mathscr{S}$, sizes of R, S and $R \bowtie S$, and join selectivity factor f. Using the optimal algorithm, the query source Q calculates the optimal region P where the join should be executed in the sensor network.

2.1.4. Join in sensor networks

The SQL join operator is used to correlate data from multiple tables, and can be defined as a selection (join) predicate over the cross-product of a pair of tables; a join of R and S tables is denoted as $R \bowtie S$. One of the most popular applications of sensor networks is *event detection*, which motivates the body of our work. An *event* indicates a point in time of interest based on certain conditions over the generated sensor data. For certain applications, events may simply depend on the local value of a particular sensor reading. Higher-level events or complex events may be specified using composition operators over the primitive events. In particular, the complex events may be represented as a join of multiple data streams, involving spatial and temporal constraints and correlations.

3. In-network implementation of join

In this section, we first develop communication-efficient algorithms for implementation of a join operation over static (non-streaming) database tables stored in some sensor network region. As data in sensor network is better represented as data stream tables, we will generalize our techniques for stream database tables in the next section.

Consider a join operation, initiated by a query source node Q, involving two static (non-streaming) tables R and S distributed horizontally across some geographical regions $\mathscr{R}$ and $\mathscr{S}$ in the network. We assume that the geographic regions are disjoint and small relative to the distances between the query source and the operand table regions. We later discuss generalizing our algorithms for general query source locations and operand regions. If we do not make any assumptions about the join predicates involved, each data tuple of table R should be paired with every tuple of S and checked for the join condition. The joined tuple is then routed (if it passes the join selection condition) to the query source Q where all the tuples are accumulated or consumed. Given that each sensor node has limited memory resources, we need to find out appropriate regions in the network that would take the responsibility of computing the join. In particular, we may need to store and process the relations at some intermediate location before routing the result to the query source.

A simple nested-loop implementation of a join used in traditional databases is to generate the cross-product (all pairs of tuples), and then extract those pairs that satisfy the selection predicate of the join. More involved implementations of a join operator widely used in database systems are merge-sort and hash-join. These classical methods are unsuitable for direct implementation in sensor networks due to the limited memory resources at each node in the network. Moreover, the traditional join algorithms focus on minimizing computation cost, while in sensor networks the primary performance criteria is communication cost. Below, we discuss various techniques for efficient implementation of the join operation in sensor networks.

Naive approach. A simple way to compute $R \bowtie S$ could be to route the tuples of S from their original location $\mathscr{S}$ to the region $\mathscr{R}$, broadcast the S-tuples in the region $\mathscr{R}$, compute the join within the region $\mathscr{R}$, and then route the joined tuples to the query source Q. The breakup of the total communication cost incurred is as follows:

(1) Cost incurred in routing the table S to the region $\mathscr{R}$.
(2) Cost incurred in broadcasting the table S throughout $\mathscr{R}$.
(3) Cost incurred in routing the result (from $\mathscr{R}$) to the query source Q.

Note that in the above approach the roles of the tables R and S can be interchanged.[1]

Centroid approach. Now, we consider another approach where the region responsible for computing the join operation is a circular region around some point C in the sensor network. Let $|R|$ denote the size of the table R, m denote the memory of each sensor node, and let P_c be the smallest circular region around C such that the region P_c has more than $|R|/m$ sensor nodes to store the table R. First we route and distribute the tuples of table R in the region P_c, and then route and broadcast the tuples of table S in the region P_c. After computing the join operation in the region P_c, we route the resulting tuples of $(R \bowtie S)$ to the query source Q. The communication cost incurred consists of the following components. (i) Cost incurred in routing the tables to C. (ii) Cost incurred in distributing R and broadcasting S in the region P_c around C. (iii) Cost incurred in routing the result $(R \bowtie S)$ to the query source Q. Since the second component of the cost is independent of the choice of C, it is easy to see that the communication cost incurred in the above approach is minimized when the point C is the weighted centroid of the triangle formed by R, S, and Q (i.e., the point that minimizes the sum of the weighted distances from the three points). Here, the choice of the centroid point C is weighted by the sizes of R, S, and $(R \bowtie S)$.

3.1. Path-join algorithm

In the above two paragraphs, we described a couple of simple approaches to compute the join operation in a sensor network. However, in order to minimize communication cost, we may need to perform the join operation in a region having a non-trivial shape. In this subsection, we present a

[1] The other simple approach of computing the join at a region around Q is subsumed by the Centroid Approach discussed next.

novel *path-join* approach of performing a join operation. In the next subsection, we will extend the path-join approach to devise an optimal join algorithm that incurs minimum communication cost in dense sensor networks.

The path-join implementation of the join operation works as follows. First, all the tuples of R are distributed uniformly along an appropriately chosen path P containing $|R|/m$ sensor nodes, where $|R|$ is the size of table R and m is the memory size of each sensor node. Then, every tuple of S is routed to the path P and passed through all the sensors along P to perform the join. The resulting joined tuples computed at each sensor along the path P are then routed to the query source Q. See Fig. 1. The location of the path P is chosen to minimize the total communication cost incurred. We estimate the total communication incurred in terms of a notion of sensor length, defined below.

Definition 1 (*Sensor length $d(\mathscr{X}, y)$, and notation $|\mathscr{X}|$*). The sensor length between a region $\mathscr{X}$ and a point y in a sensor network plane is denoted as $d(\mathscr{X}, y)$ and is defined as the average weighted distance, in terms of number of hops (i.e., intermediate nodes), between the region $\mathscr{X}$ and the point y. Here, the distance between a point $x \in \mathscr{X}$ and y is weighted by the amount of data residing at x.

For a region $\mathscr{X}$ in the sensor networks, the notation $|\mathscr{X}|$ denotes the number of sensors in the region $|\mathscr{X}|$. Note that for a relational table R, we use $|R|$ to denote the size of the table R.

Let $|R|, |S|$, and $|R \bowtie S|$ be the respective sizes of the tables R, S, and the joined result $R \bowtie S$. Let Q be the query source, C_0 be an end of the path P that is closer to $\mathscr{R}$ and/or $\mathcal{S}$, and $|P|$ be the number of sensors on the path P. Note that by choice of P, $|P| = |R|/m$. Let us assume that both R and S start their broadcast and distribution phases from the same point C_0. The total communication cost incurred in the path-join algorithm consists of: cost of routing R to C_0, cost of routing S to C_0, cost of distributing the table R along the path P, cost of broadcasting the table S along the path P, and finally, the cost of routing the joined tuples from P to Q. If we assume that the resulting joined tuples are uniformly distributed along the path P, then the total communication cost incurred is

$$|R|d(\mathscr{R}, C_0) + |S|d(\mathcal{S}, C_0) + |P||R|/2 + |P||S| + |R \bowtie S|d(P, Q).$$

Note that the distribution and broadcast cost of R and S respectively is independent of the location of P. In some cases, the path-join algorithm may not be optimal, i.e., may incur more than the minimum communication cost possible.

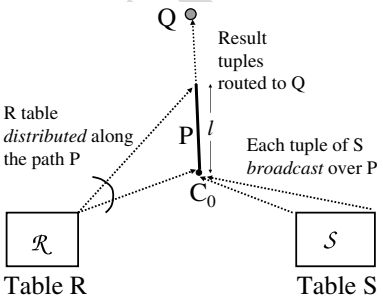


Fig. 1. Path-join implementation. Here, $l = |R|/m$.

3.2. Optimal join algorithm

In this section, we present an algorithm that uses path-join as a basic component, and constructs a region for computing the join operation using optimal communication cost. We assume that the sensor network is sufficiently dense that we can find a sensor node at any point in the region. To formally prove the claim of optimality, we need to restrict ourselves to a class of join algorithms called *Distribute-Broadcast Join Algorithms* (defined below). In effect, our claim of optimality states that the proposed join algorithm incurs less communication cost than any distribute-broadcast join algorithm.

Definition 2 (*Distribute-broadcast join algorithms*). A join algorithm to compute $R \bowtie S$ in a sensor network is a distribute-broadcast join algorithm if the join is processed by first uniformly distributing the table R in some region P (other than the region $\mathscr{R}$ storing R)[2] of the sensor network followed by broadcasting the relation S within the region P to compute the join. The joined tuples are then routed from each sensor in the region P to the query source.

As before, consider a query source Q, and regions $\mathscr{R}$ and $\mathcal{S}$ that store the static operand tables R and S in a sensor network. The key challenge in designing an optimal algorithm for implementation of a join operation is to select a region P for processing the join in such a way that the total communication cost is minimized. Note that in general, P

[2] Else, the algorithm will be identical to one of the naive approaches.

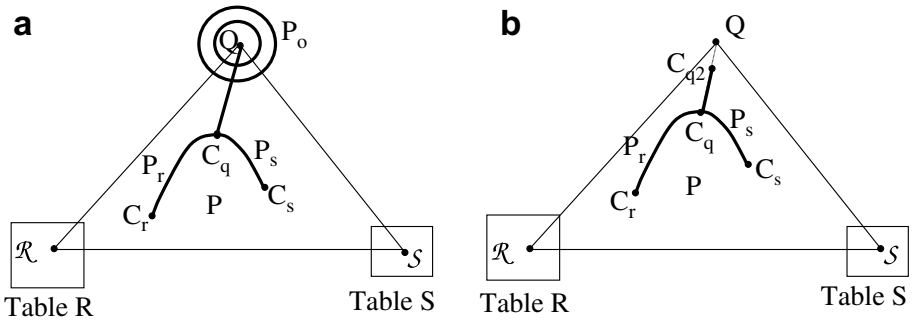


Fig. 2. Shape of an optimal join-region. (a) Shape of an optimal region P, when $Q \in P$. (b) Shape of an optimal region P, when $Q \notin P$.

may be an arbitrary *region* as opposed to just a path as in the path-join algorithm. We use the term *join-region* to refer to a region in the sensor network that is responsible for computing the join.

3.2.1. Shape of an optimal join-region

We show in Theorem 1 that the join-region P that incurs minimum communication cost has a shape as shown in Fig. 2a or b. In particular, the optimal join-region P is formed using three points C_r, C_s, and C_q in the sensor network (typically these points will lie within the $\triangle \mathcal{R}\mathcal{S}Q$). More precisely, given three points C_r, C_s, and C_q in the sensor network, the region P takes one of the following forms:

(1) Region P is formed of the paths $P_r = (C_r, C_q)$ and $P_s = (C_s, C_q)$, the line segment $\overline{C_qQ}$, and a circular region P_O of appropriate radius around Q. See Fig. 2a.
(2) Region P is formed of the paths $P_r = (C_r, C_q)$ and $P_s = (C_s, C_q)$, and a *part* of the line segment $\overline{C_qQ}$. See Fig. 2b.

Theorem 1. *In dense sensor networks, the shape of the join-region P used by a distribute-broadcast join algorithm that incurs optimal communication cost is as described above or as depicted in Fig. 2a or b.*

Proof. Let us consider an optimal distribute-broadcast implementation of join using a connected[3] join-region P. By definition of distribute-broadcast algorithms, the region P is different than $\mathcal{R}$.

Cost of distribution and broadcast. We assume that due to lack of global knowledge about the other sensors' locations and available memory capacities, the best way to distribute R in the region P is to route the tuples of R to a some point C_r in P and then, traverse the region P in a linear manner (as in the case when P is a path) to distributed the tuples evenly in P. The total cost of distributing of R in the region P using the above approach is

$$|R|d(\mathcal{R}, C_r) + |R||P|/2,$$

where C_r is the point in P where the tuples of R are first routed to and $|P|$ denotes the total number of sensors in the region P. Based on the same assumption and a similar argument, the total cost of *broadcasting* S in the region P is

$$|S|d(\mathcal{S}, C_s) + |S||P|,$$

where C_s is some point in P where the tuples of S are first routed to. Note that the above formulated cost of distribution of R and broadcast of S in the region P is independent of the shape and location of P.

Total communication cost. Given the join-region P and the points $C_r, C_s \in P$, where R and S are routed for distribution and broadcast respectively in the region P, the total communication cost $T(C_r, C_s, P)$ incurred in computing the join can be formulated as below.

$$T(C_r, C_s, P) = |R|d(\mathcal{R}, C_r) + |S|d(\mathcal{S}, C_s) + |R||P|/2 + |S||P| + |R \bowtie S|d(P, Q) \quad (1)$$

The term $|R \bowtie S|d(P, Q)$ is the communication cost incurred in routing the result tuples from P to the query source Q. Note that for a fixed pair of points C_r and C_s, the only component of $T(C_r, C_s, P)$ that depends on the shape of P is $d(P, Q)$.

Proof plan. We prove the theorem by contradiction. In particular, we show that if P is not of a shape depicted in Fig. 2a or b, then we can alter the shape of region P without changing $|P|$, C_r, or C_s, such that the cost component $d(P, Q)$ is further reduced. The above change will result in a reduction

[3] The generalization to disconnected join-regions can be easily made by applying the proof to each connected subregion independently.

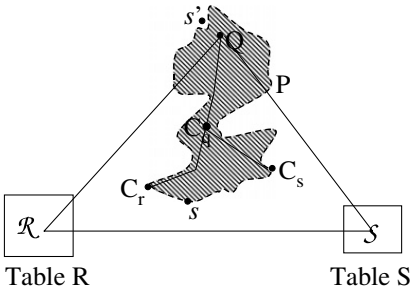


Fig. 3. An arbitrary join-region P containing Q, and the shortest paths (C_r, Q) and (C_s, Q) in P. Here, s is the point in $P - (C_r, Q) - (C_s, Q)$ that is farthest from Q, and s' is the point not in P that is closest to Q. If P is not of the form Fig. 2a, then $d(P, Q)$ can be reduced (without changing C_r, C_s, or $|P|$) by replacing s by s', and thus, reducing the total cost $T(C_r, C_s, P)$.

of the total cost $T(C_r, C_s, P)$, since we keep $|P|$, C_r, and C_s fixed.

Reducing $d(P, Q)$, when $Q \in P$. Let us consider the case when Q is in P, but the region P is not of the form Fig. 2a. Let C_r and C_s be the points in P to where R and S are routed for distribution and broadcast respectively in the region P. See Fig. 3. Consider paths (not necessarily disjoint) (C_r, Q) and (C_s, Q) *contained* in P that connect C_r and C_s respectively to Q using minimum number of sensor nodes. Since P is connected, such paths exist. Consider a point s in P such that s is neither on path (C_r, Q) nor on path (C_s, Q), and is farthest away from Q. Such a point s much exist, else P would be comprised entirely of paths (C_r, Q) and (C_s, Q) and hence, of the form Fig. 2a.[4]

Now, consider a point $s' \notin P$ that is closest to Q. See Fig. 3. If Q is closer to s than s', i.e., if $d(s, Q) \leqslant d(s', Q)$, then P is just comprised of the paths (C_r, Q), (C_s, Q), and a fully packed circular region around Q, and thus, of the form Fig. 2a. Since we assumed to the contrary, Q must be closer to s' than s, i.e., $d(s', Q) < d(s, Q)$. Now, in such a case, $d(P, Q)$ can be reduced as follows. Since $d(P, Q) = 1/|P| \sum_{p \in P} d(p, Q)$ (since R is uniformly distributed in P), the value $d(P, Q)$ can be reduced by changing P to $P - \{s\} \cup \{s'\}$, i.e., replacing s by s'. Note that such a point s' will be directly connected to P, and hence, addition of s' maintains the connectivity of P. Moreover, since s is neither in (C_r, Q) nor in (C_s, Q), and is farthest such point from Q, removal of s from P maintains the connectivity of P. Finally, the above replacement keeps C_r, C_s, and $|P|$ fixed.

Final arguments. Thus, we can reduce $d(P, Q)$, while keeping C_r, C_s, and $|P|$ fixed, and thus, reduce the total cost $T(C_r, C_s, P)$, when $Q \in P$ and P is not of the form Fig. 2a. Thus, by contradiction, the optimal join-region P must be of the form depicted in Fig. 2a if $Q \in P$. Using similar arguments, we can show that if $Q \notin P$, the optimal join-region must be of the shape depicted in Fig. 2b. □

Note that the assumptions made (viz. restricted class of algorithms, distributing and broadcasting in a linear fashion) in proving the above theorem do not restrict the applicability of our developed techniques. The assumptions were made solely to prove optimality, and more importantly, to develop an algorithm that could form the *basis* of a communication-efficient implementation of the join operation in general sensor networks without any restrictions on the communication/computation model.

Note that the above theorem only restricts the shape of an optimal join-region; there are still an infinite number of possible join-regions of shapes depicted in Fig. 2. Thus, we now further restrict the shape of an optimal join-region. by characterizing the equations of the paths P_r and P_s that connect C_r and C_s respectively to C_q.

3.2.2. Optimizing paths P_r and P_s in the join-region

Consider an optimal join-region P that implements a join operation using minimum communication cost. By Theorem 1, we know that the region P is of the shape depicted in Fig. 2a or b. As derived in Eq. (1), the total communication cost $T(C_r, C_s, P)$ incurred in processing of a join using the region P is $|R|d(\mathcal{R}, C_r) + |S|d(\mathcal{S}, C_s) + |R||P|/2 + |S||P| + |R \bowtie S|d(P, Q)$. Let $P' = P - P_r - P_s$, i.e., the region P without the paths P_r and P_s. Since the result $|R \bowtie S|$ is uniformly spread along the entire region P, we have

$$d(P, Q) = \frac{1}{|P|}|P'|d(P', Q) + |P_r|d(P_r, Q) + |P_s|d(P_s, Q).$$

For a *given* $|P|$ and a given set of points C_r, C_s, and C_q, the total communication cost T is minimized when the path P_r is constructed such that $|P_r|d(P_r, Q)$ is minimized. Otherwise, we could reconstruct P_r with a smaller $|P_r|d(P_r, Q)$, and

[4] Note that since paths (C_r, Q) and (C_s, Q) are shortest in P, they intersect at only one point C_q and have the same subpaths (C_q, Q).

remove/add sensors nodes from the end[5] of the region P' to maintain $|P|$. Removal of sensor nodes from P' will always reduce T, and it can be shown that addition of sensor nodes to the end of the region P' will not increase the cost more than the reduction achieved by optimizing P_r. Similarly, the path P_s could be optimized independently.

We now derive the equation of the path P_r that minimizes $|P_r|d(P_r, Q)$ for a given C_r and C_q. Consider an arbitrary point $R(x, y)$ along the optimal path P_r. The length of an infinitesimally small segment of the path P_r beginning at $R(x, y)$ is $\sqrt{(\mathrm{d}x)^2 + (\mathrm{d}y)^2}$, and the average distance of this segment from Q is $\sqrt{x^2 + y^2}$, if the coordinates of Q are $(0, 0)$. Sum of all these distances over the path P_r is: $F = \int_{C_r}^{C_q} \sqrt{x^2 + y^2}\sqrt{(\mathrm{d}x)^2 + (\mathrm{d}y)^2} == \int_{C_r}^{C_q} \sqrt{x^2 + y^2}\sqrt{1 + (y')^2}\mathrm{d}x$. To get the equation for the path P_r, we would need to determine the extremals of the above function F. Using the technique of calculus of variations [7], we can show that the extremal values of F satisfy the Euler–Lagrange differential equation. The equation of the path P_r can thus be computed as (we omit the details): $\beta = x^2 \cos\alpha + 2xy \sin\alpha - y^2 \cos\alpha$ where the constants α and β are evaluated by substituting for coordinates of C_r and C_q in the equation.

3.2.3. Computing communication cost

Given $|P|$ and the three points C_r, C_s, and C_q, we now derive the total communication cost $T_{\mathrm{opt}}(C_r, C_s, C_q, |P|)$ incurred by using the optimal join-region of size $|P|$ constructed over C_r, C_s and C_q. We will use the formulation of $T_{\mathrm{opt}}(C_r, C_s, C_q, |P|)$ to design an optimal algorithm by consider all possible combinations of values of $|P|$, C_r, C_s and C_q and picking the quartet that results in minimum $T_{\mathrm{opt}}(C_r, C_s, C_q, |P|)$.

Given $|P|$ and points C_r, C_s, C_q, let P_r and P_s be the paths as obtained in the previous paragraph. Let

$$l_Y = |P_r| + |P_s| + |\overline{C_qQ}|.$$

If $l_Y > |P|$, then the optimal join-region P cannot contain the point Q, and hence, by Theorem 1, the region P is comprised of the optimized paths P_r, P_s, and the line segment $\overline{C_qC_{q2}}$, where $C_{q2} \in \overline{C_qQ}$ is such that $|\overline{C_qC_{q2}}| = |P| - (|P_r| + |P_s|)$. See Fig. 2b. For the case when $l_Y \leqslant |P|$, the $l_Y/|P|$ fraction of the join is processed on the curves P_r, P_s, and the line segment $\overline{C_qQ}$, while the remaining fraction of the join is processed on a circular region P_O of appropriate radius around Q. See Fig. 2a. From Theorem 1, the above choice of P minimizes the value $d(P, Q)$ for a given combination of C_r, C_s, C_q, and $|P|$. Thus, we have

$$P = P_r \cup P_s \cup \overline{C_qC_{q2}} \quad \text{if } l_Y > |P|, \tag{2}$$

$$P = P_r \cup P_s \cup \overline{C_qQ} \cup P_O \quad \text{if } l_Y \leqslant |P|. \tag{3}$$

As mentioned before, the point C_{q2} is such that $\overline{C_qC_{q2}} = |P| - (|P_r| + |P_s|)$, and P_O is a circular region of sufficient radius around Q such that $|P_O| = |P| - (|\overline{C_q, Q}| + |P_r| + |P_s|)$. For a given quartet of values $(C_r, C_s, C_q, |P|)$, let $T_{\mathrm{opt}}(C_r, C_s, C_q, |P|)$ denote the total communication cost incurred when the join-region P is optimally constructed as suggested by Eqs. (2) and (3). In other words, $T_{\mathrm{opt}}(C_r, C_s, C_q, |P|)$ is equal to $|R|d(\mathscr{R}, C_r) + |S|d(\mathscr{S}, C_s) + |R \bowtie S|d(P, Q) + |R||P|/2 + |S||P|$, where P is the optimally constructed join-region as suggested by Eqs. (2) and (3).

3.2.4. Optimal join algorithm

Based on the above discussion, we construct an optimal join-region to compute a join operation for tables R and S and the query source Q, by considering all possible triples of points C_r, C_s, and C_q in the sensor network and values of $|P|$, and pick the quartet $(C_r, C_s, C_q, |P|)$ that minimizes the value $T_{\mathrm{opt}}(C_r, C_s, C_q, |P|)$. For such an optimal quartet $(C_r, C_s, C_q, |P|)$, we construct the optimal join-region P as suggested by Eqs. (2) and (3) in the previous paragraph. If n is the total number of network nodes, then there are at most n^4 combinations of $(C_r, C_s, C_q, |P|)$. Thus, the time complexity of the above algorithm which constructs an optimal join-region is $O(n^4)$.

3.2.5. Suboptimal heuristic

The high time complexity of the optimal algorithm described above makes it impractical for large sensor networks. Here, we design a suboptimal heuristic that has a much lower time complexity and performs very well in practice (see Fig. 4). In particular, we reduce the complexity of our designed algorithm from $O(n^4)$ to $O(n^{3/2})$ using the following five steps. (i) We choose the minimum value of $|P|$, i.e., $|P| = |R|/m$, where $|R|$ is the size of the table R to be distributed and m is the memory at each sensor node. (ii) We look at all possible values for C_r in

[5] Here, by the end of the region P', we mean either the circular part P_O or the line segment $\overline{C_qC_{q2}}$ depending on the shape.

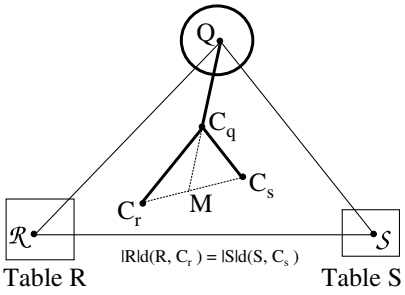


Fig. 4. Suboptimal heuristic for join implementation.

the region. (iii) For each C_r, we stipulate that C_s should be "symmetrically" located ($|R|d(\mathcal{R}, C_r) = |S|d(\mathcal{S}, C_s)$) in the $\triangle\mathcal{R}Q\mathcal{S}$. Thus, the location of C_s is fixed for a given C_r. (iv) We approximate paths P_r and P_s to be straight line segments $\overline{C_rC_q}$ and $\overline{C_sC_q}$ respectively. (v) We further stipulate that the point C_q should lie on the median of the $\triangle C_rC_sQ$. Thus, for each point as C_r in the sensor network, we determine C_s and search for the best C_q on the median of $\triangle C_rC_sQ$. The above reduces the time complexity to construct a join-region to $O(n^{3/2})$, where n is the network size.

4. Generalizations to stream tables and general sensor networks

In this section, we extend the our proposed algorithms to real sensor networks and relax the assumptions made in the previous section. We start with generalizing our technique for stream database tables. Then, we present the overall working of our approach in general sensor networks. Finally, we discuss a few other generalizations.

4.1. Implementation for stream database tables

In the previous section, we discussed implementation of the join operation in a sensor network for static database tables. Since, sensor network data is better represented as stream database tables, we now generalize the algorithms to handle stream database tables. First, we start with presenting our model of stream database tables in sensor networks.

4.1.1. Data streams in sensor networks

As for the case of static tables, a stream database table R corresponding to a data stream in a sensor network is associated with a region $\mathcal{R}$, where each node in $\mathcal{R}$ is continually generating tuples for the table R. To deal with the unbounded size of stream database tables, the tables are usually restricted to a finite set of tuples called the *sliding window* [1,6,16]. In effect, we expire or archive tuples from the data stream based on some criteria so that the total number of stored tuples does not exceed the bounded window size. We use W_R to denote the sliding window for a stream database table R.

4.1.2. Naive approach for stream tables

In the naive approach, we use the region $\mathcal{R}$ (or $\mathcal{S}$) to store the windows W_R and W_S of the stream tables R and S.[6] Each sensor node in the region $\mathcal{R}$ uses $W_R/(|W_R| + |W_S|)$ fraction of its local memory to store tuples of W_R, and the remaining fraction of the memory to store tuples of W_S.[7] We need to store W_S also in the region $\mathcal{R}$ to find matches for a newly generated tuple of R. To perform the join operation, each newly generated tuple (of R or S) is broadcast to all the nodes in the region $\mathcal{R}$, and is also stored in some node of $\mathcal{R}$ with available memory. Note that the generated data tuples of S need to be first routed from the region $\mathcal{S}$ to the region $\mathcal{R}$. The resulting joined tuples are routed from $\mathcal{R}$ to the query source Q.

4.1.3. Generalizing other approaches

The other approaches viz. centroid approach, optimal algorithm, and suboptimal heuristic, use a join-region that is separate from the regions $\mathcal{R}$ and $\mathcal{S}$. These algorithms are generalized to handle stream database tables as follows. First, the strategy to choose the join-region P remains the same as before for static tables, except for the size of the join-region. For stream database tables, the chosen join-region is used to store W_R as well as W_S, with each sensor node in the join-region using $W_R/|W_R| + |W_S|$ fraction of its memory to store tuples of W_R, and the rest to store tuples of W_S. We need to store W_S as well in the join-region in order to find matches for the newly generated tuples of R. Now, each newly generated tuple (of R or S) is routed from its source node in $\mathcal{R}$ or $\mathcal{S}$ to the join-region P, and broadcast to all the nodes in P. The resulting joined tuples are then routed to Q. As part

[6] If the total memory of the nodes in $\mathcal{R}$ is not sufficient to store W_R and W_S, then the region $\mathcal{R}$ is expanded to include more sensor nodes.

[7] An alternate naive strategy could be to store W_R and W_S in $\mathcal{R}$ and $\mathcal{S}$ respectively, but route each new tuple of R to $\mathcal{S}$ and each new tuple of S to $\mathcal{R}$. Such a strategy uses more number of nodes for storages, but incurs more routing communication cost.

of the broadcast process (without incurring any additional communication cost), each generated tuple of R (or S) is also stored at some node in P with available memory.

4.2. Overall implementation in real sensor networks

In this subsection, we consider overall working of our approaches in general sensor networks. We start with discussing the construction of join-region and details of the underlying routing protocols appropriate for our developed techniques.

4.2.1. Join-regions and routing protocols in general networks

Till now, we have assumed "geometric" sensor networks, and looked at the problem of finding an optimal join-region in a geometric sense. In other words, we assumed that the sensor network is very dense so that we can find a sensor node at any desirable point in the region. In case of non-geometric (i.e., not sufficiently dense) networks, we define the join-region based on the paths traversed by appropriate routing protocols. In particular, we use GPSR [10] and TBF (trajectory based forwarding [17]) routing protocols to traverse appropriate parts of the intended join-region. More specifically, we use the paths traversed by GPSR protocol as the paths for the line-segment parts of the join-region, i.e., $\overline{C_qQ}$ (or $\overline{C_qC_{q2}}$), and the paths P_r and P_s in the suboptimal heuristic. However, for the curved (non-straight) parts of the join-region (i.e., the paths P_r and P_s in the optimal algorithm), we need to use the TBF technique, which works by forwarding packets to nodes closest to the intended path/trajectory. For reasonably dense sensor networks, the above approach yields a join-region that is very close to the originally intended optimal geometric join-region.

4.2.2. Overall working of our approaches

Recall that the algorithms to construct the join-regions are run at the query source. As typical sensor network queries are long running, the query source can gather all the catalogue information needed (estimated sizes and locations of the operand relations, join selectivity factor, network density) by initially sampling the operand tables. When the query source Q needs to issue a join query, it determines the join-region based on the catalogue information, and passes the constructed join-region (represented by the paths P_r, P_s, and $\overline{C_qC_{q2}}$ (or $\overline{C_qQ}$ and radius around Q)) to all the nodes in the regions $\mathscr{R}$ and $\mathscr{S}$. Each generated tuple r of stream R is routed from its source node (in region $\mathscr{R}$) to the node nearest to C_r using GPSR protocol. On reaching C_r, we use GPSR/TBF protocol to route the tuple r through the path P_r to reach the node nearest to C_q, and then use GPSR to route r to the node nearest to C_{q2} or Q depending on the join-region. Finally, if needed, the tuple is broadcast in a region around Q of appropriate radius. In addition, during the above traversal, the tuple is joined with tuples of W_s (the sliding window of S) stored locally at each node of the join-region. Also, the tuple r is stored at the first encountered node with available memory in the join-region.

4.2.2.1. Effect of node failures. As described above, our proposed implementations do not use any specific destination nodes for traversing the constructed join-region. That is, even though the join-region is originally represented by certain geographic locations and paths, the actual join-region traversed is based on the paths traversed by GPSR/TBF protocols to nodes nearest to geometric locations. Thus, our overall techniques automatically adapt to node failures just as the underlying routing protocols.

5. Performance evaluation

In this section, we present our simulation results comparing performance of various algorithms designed in this article. In particular, we compare the performance of Naive Approach, Centroid Algorithm, Optimal Algorithm, and Suboptimal Heuristic. Each algorithm is generalized for stream database tables and non-geometric general sensor network. We refer to the generalized algorithms as Naive, Centroid, *OptBased*, and Suboptimal Heuristic respectively. Our simulations demonstrate the effectiveness of our developed techniques. We start with defining join-selectivity factor which is used to characterize the size of the join result.

Definition 3 (*Join-selectivity factor*). Given instances of relations R and S and a join predicate, the join-selectivity factor is the probability that a random pair of tuples from R and S will satisfy the given join predicate. In other words, the join selectivity factor is the ratio of the size of $R \bowtie S$ to the size of the cartesian product, i.e., $|R \bowtie S|/(|R||S|)$.

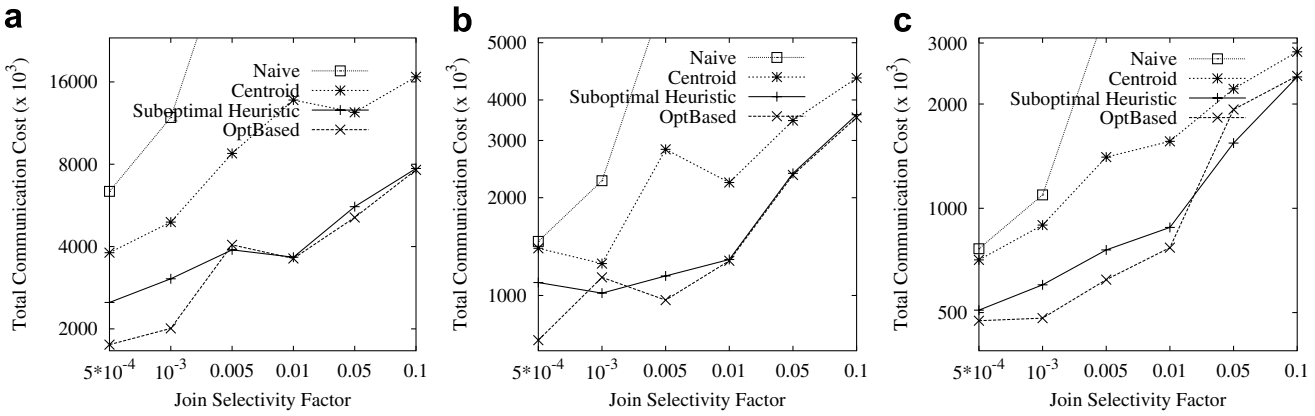


Fig. 5. Performance for a fixed $\triangle\mathscr{R}SQ$ with varying join-selectivity factor for three different transmission radii: (a) 0.13 units, (b) 0.15 units, (c) 0.18 units.

5.1. Parameter values and experiments

We generated random sensor networks by randomly placing 10,000 sensors in an area of 10×10 units. Each sensor has a uniform transmission radius, and two sensors can communicate with each other if they are located within each other's transmission radius. For the purposes of comparing the performance of our algorithms, varying the number of sensors is tantamount to varying the transmission radius. Thus, we fix the number of sensors to be 10,000 and measure performance for different transmission radii. Memory size of a sensor node is 300 tuples, and the size of each of the sliding windows W_R and W_S of stream tables R and S is 8000 tuples. For simplicity, we chose uniform data generation rates for R and S streams. In each of the experiments, we measure communication cost incurred in processing 8000 newly generated tuples of R and S each, after the join-region is already filled with previously generated tuples. We use the GPSR [10] algorithm to route tuples. Catalogue information is gathered for non-Naive approaches by collecting a small sample of data streams at the query source.

We ran three sets of experiments on randomly generated sensor networks. In the first set of experiments, we consider a fixed $\triangle\mathscr{R}SQ$ and calculate the total communication cost for various transmission radii and join-selectivity factors. Next, we fix the transmission radius and calculate the total communication cost for various join-selectivity factors and various shapes/sizes of the $\triangle\mathscr{R}SQ$. Finally, we plot of performance of various algorithms in terms of the network lifetime. Below, we discuss our simulation results in detail.

5.2. Fixed triangle $\mathscr{R}SQ$

In this set of experiments, we fix the locations of regions R, S, and query source Q and measure the performance of our algorithms for various values of transmission radii and join-selectivity factors. In particular, we choose coordinates (0, 0), (5, 9.5), and (9.5, 0) for R, Q, and S respectively. The total communication cost incurred by various algorithms for 8000 newly generated tuples of R and S is shown in Fig. 5a–c. We have looked at three transmission radii viz. 0.13, 0.15, and 0.18 units. Lower transmission radii left the sensor network disconnected, and the trend observed for these three transmission radii values was sufficient to infer behavior for larger transmission radii. From Fig. 5a–c, we can see that the Suboptimal Heuristic performs very close to the OptBased Algorithm, and significantly outperforms (upto 100%) the Naive and Centroid Approaches for most parameter values. Sometimes the Suboptimal Heuristic even outperforms the OptBased Algorithm by a small margin.[8] The performance of the Naive approach worsens drastically with the increase in the join-selectivity factor, since the routing cost of the joined tuples from the join region ($\mathscr{R}$ or $\mathscr{S}$) to the query source Q becomes more dominant. For sake of clarity, we have not shown the Naive Approach data points for high join-selectivity factors. Also, note that with the increase in transmission radius and/or selectivity factor, the relative benefit of Suboptimal Heuristic over the Centroid Approach reduces. In particular, for extremely large

[8] Note that this does not contradict the optimality of the Optimal Algorithm, since the OptBased is only *based* on the Optimal Algorithm for real sensor networks.

transmission radius, all algorithms will have similar performance. Also, for very large selectivity factors, all non-Naive approaches would yield similar implementations, and the savings with respect to the Naive approach would remain relatively constant after a certain (depending on other parameter values) join-selectivity factor.

5.3. Fixed transmission radius (0.15 units)

We also observe the performance of various algorithms for different size and shapes of $\triangle\mathscr{R}SQ$. In particular, we fix the transmission radius of each sensor node in the network to be 0.15 units, and generate various $\triangle\mathscr{R}SQ$'s as follows. We fix locations of regions $\mathscr{R}$ and $\mathcal{S}$, and select many locations of the query source Q with the constraint that the area of the $\triangle\mathscr{R}SQ$ is between 10% and 50% of the total sensor network area. For each such generated $\triangle\mathscr{R}SQ$, we run all the four algorithms for three representative join-selectivity factor values viz. $10^{-4}, 5\times10^{-3}$, and 10^{-2}. See Fig. 6. Again we observe that the Suboptimal Heuristic performs very close to the OptBased Algorithm, and incurs much less communication cost than the Naive and Centroid Approaches for all join-selectivity factor values.

5.4. Network lifetime

Finally, we demonstrate the effectiveness of our algorithms in prolonging the lifetime of the network. For each algorithm, we start with the same randomly generated sensor network, and equip each sensor node with a uniform battery power capable of transmitting 50,000 tuples. Queries are issued in the network by randomly selecting the locations of the regions $\mathscr{R}$ and $\mathcal{S}$, and the query source Q. We only execute those queries that can be successfully completed is all the three networks. As more and more queries are answered, sensor nodes start getting depleted of their battery power. In Fig. 7, we

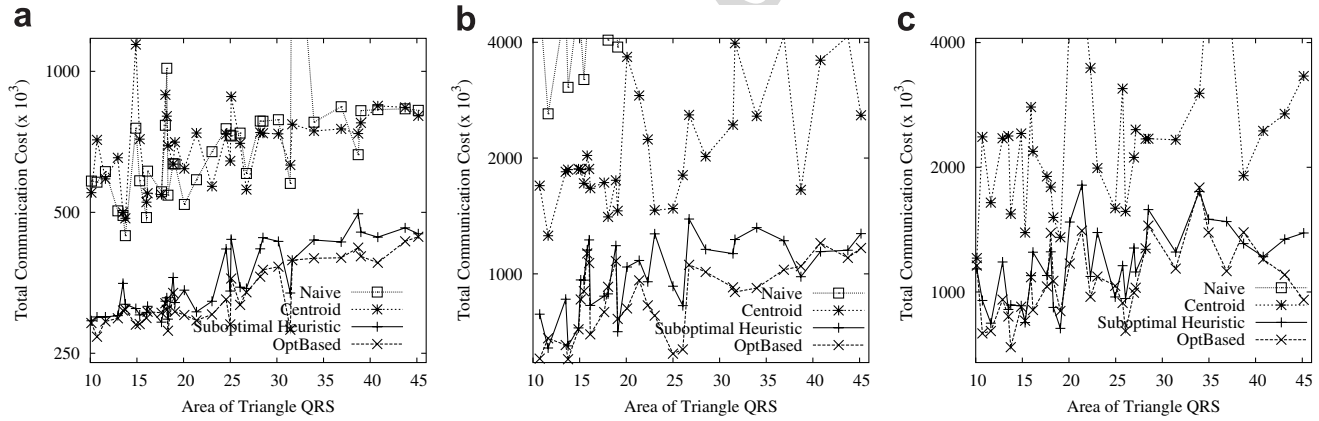


Fig. 6. Performance for different triangles $\mathscr{R}SQ$ (ordered by their area) for three different join-selectivity factors: (a) 10^{-4}, (b) 5×10^{-3}, (c) 10^{-2}. Here, the transmission radius is 0.15 units.

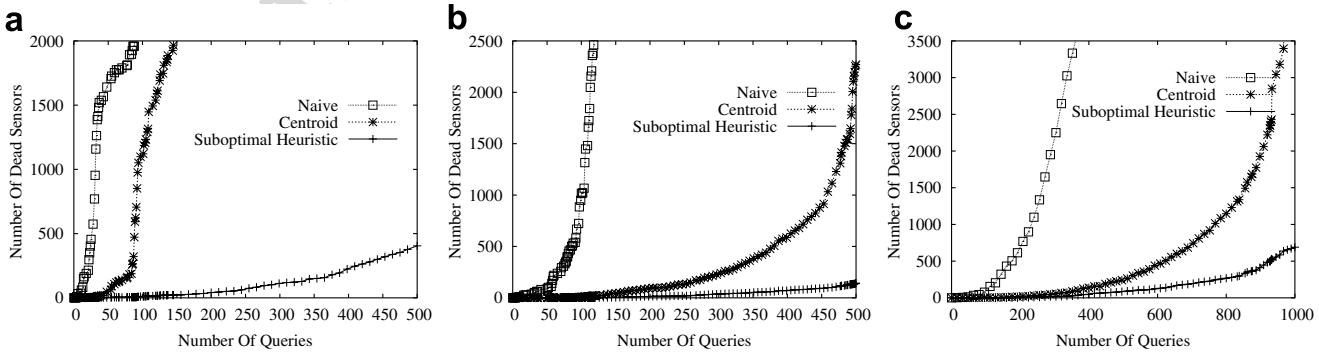


Fig. 7. Effect on network lifetime for three different transmission radii: (a) 0.13 units, (b) 0.15 units, (c) 0.18 units.

plot number of dead sensors against number of queries answered for each of the three algorithms viz. Naive Approach, Centroid Approach, and Suboptimal Approach. Since the time-complexity of OptBased Heuristic is very high and the Suboptimal Heuristic has been observed to perform very close to the OptBased Algorithm, we did not run the OptBased Algorithm for this set of experiment. As expected, higher transmission radius causes less depletion of batter power for each approach. We can easily observe from the graphs in Fig. 7 that the Suboptimal Heuristic cause much less depletion of battery power compared to the Naive and Centroid Approaches.

6. Related work

The vision of sensor network as a database has been proposed by many works [8,15,18], and simple query engines such as TinyDB [15] have been built for sensor networks. In particular, the COUGAR project [18] at Cornell University is one of the first attempts to model a sensor network as a database system. The TinyDB Project [15] at Berkeley also investigates query processing techniques for sensor networks. However, TinyDB implements very limited functionality [14] of the traditional database language SQL. A plausible implementation of an SQL query engine for sensor networks could be to ship all sensor nodes' data to an external server that handles the execution of queries completely [12]. Such an implementation would incur high communication costs and congestion-related bottlenecks. In particular, [9] shows that in-network implementation of database queries is fundamental to conserving energy in sensor networks. Thus, recent research has focussed on in-network implementation of database queries. However, prior research has only addressed limited SQL functionality – single queries involving simple aggregations [13] and/or selections over single tables [14], or local joins [18]. So far, it has been considered that correlations such as median computation or joins should be computed on a single node [4,14,18]. In particular, [4] address the problem of operator placement for in-network query processing, assuming that each operator is executed locally and fully on a single sensor node. In a recent work [2], authors consider a combination of localized and centralized implementation for a join operation wherein one of the operands is a relatively small static table which is used to flood the network. However, the problem of distributed and communication-efficient implementation for general join operation has not been addressed yet in the context of sensor networks.

7. Conclusions

Sensor networks are capable of generating large amounts of data. Hence, efficient query processing in sensor networks is of great importance. Since sensor nodes have limited battery power and memory resources, designing communication-efficient distributed implementation of database queries is a key research challenge. In this article, we have focussed on implementation of the join operator, which is one of the core operators of database query language. In particular, we have designed an Optimal Algorithm that incurs minimum communication cost for implementation of join in sensor networks under certain reasonable assumptions. Moreover, we reduced the time complexity of the Optimal Algorithm to design a Suboptimal Heuristic, and showed through extensive simulations that the generalization (for non-geometric real sensor networks) of Suboptimal Heuristic perform very close to that of the Optimal Algorithm. Techniques developed in this article are shown to result in substantial energy savings over simpler approaches for a wide range of sensor network parameters.

References

[1] D.J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik, Aurora: a new model and architecture for data stream management, The VLDB Journal 12 (2) (2003) 120–139.
[2] Daniel J. Abadi, Samuel Madden, Wolfgang Lindner, REED: robust, efficient filtering and event detection in sensor networks, in: VLDB, 2005.
[3] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: PODS, 2002.
[4] B. Bonfils, P. Bonnet, Adaptive and decentralized operator placement for in-network query processing, in: IPSN, 2003.
[5] N. Bulusu, J. Heidemann, D. Estrin, GPS-less low cost outdoor localization for very small devices, IEEE Personal Communications Magazine 7 (5) (2000) 28–34.
[6] L. Ding, N. Mehta, E. Rundensteiner, G.T. Heineman, Joining punctuated streams, in: EDBT, 2004.
[7] I. Gelfand, S. Fomin, Calculus of Variations, Dover Publications, 2000.
[8] R. Govindan, J. Hellerstein, W. Hong, S. Madden, M. Franklin, S. Shenker, The sensor network as a database, Technical report, University of Southern California, Computer Science Department, 2002.

[9] J.S. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, D. Ganesan, Building efficient wireless sensor networks with low-level naming, in: SOSP, 2001.
[10] B. Karp, H. Kung, GPSR: greedy perimeter stateless routing for wireless networks, in: MobiCom, 2000.
[11] S. Li, Y. Lin, S. Son, J. Stankovic, Y. Wei, Event detection using data service middleware in distributed sensor networks, Special Issue on Wireless Sensor Networks of Telecomm. Systems, 2004.
[12] S. Madden, M. Franklin, Fjording the stream: an architecture for queries over streaming sensor data, in: ICDE, 2002.
[13] S. Madden, M. Franklin, J. Hellerstein, W. Hong, TAG: a tiny aggregation service for ad-hoc sensor networks, in: OSDI, 2002.
[14] S.R. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, The design of an acquisitional query processor for sensor networks, in: SIGMOD, 2003, pp. 491–502.
[15] S.R. Madden, J.M. Hellerstein, W. Hong, D.B. Tiny, In-network query processing in tinyos. <http://telegraph.cs.berkeley.edu/tinydb>, 2003.
[16] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, R. Varma, Query processing, approximation, and resource management in a data stream management system, in: CIDR, 2003.
[17] B. Nath, D. Niculescu, Routing on a curve, in: Proceedings of the Workshop on Hot Topics in Networks, 2002.
[18] Y. Yao, J. Gehrke, Query processing for sensor networks, in: CIDR, 2003.

Vishal Chowdhary obtained his M.S. in Computer Science from Stony Brook University in 2004, and his B.E. in Mechanical Engineering from Bombay University in 2000. He is currently working at Microsoft Corporation in Seattle, WA.

Himanshu Gupta joined the faculty of the Department of Computer Science at Stony Brook University in Fall 2002. His recent research activities focus on theoretical issues in wireless networking. In particular, he is interested in sensor networks and sensor databases. His other research interests are in database systems and theory, wherein, he is interested in materialized views, (multiple) query optimization, and data analysis. He received a B.Tech. (1992) in Computer Science and Engineering from IIT, Bombay, and an M.S. and Ph.D. in Computer Science from Stanford University in 1999.