University of Alberta

QUERY PROCESSING IN WIRELESS SENSOR NETWORKS

by

Alexandru Coman ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Department of Computing Science

Edmonton, Alberta
Fall 2007

Abstract

Wireless sensor networks are made of autonomous devices, called nodes, that can collect, store, process and share data with other nodes. Users can issue queries over such sensor networks to retrieve data from nodes in applications such as environmental monitoring and surveillance. Query processing is an essential research problem in sensor networks as the usefulness of this new technology depends on its capability to gather and provide data efficiently and effectively when required. In this thesis we investigate query processing techniques for queries over historical sensor observations in a peer-to-peer sensor network. We propose novel techniques for in-network processing of three types of queries: spatial range queries, both exact and approximate and join queries. Due to the limited power supply of the sensor nodes, one of the main challenges of sensor network research is extending the lifetime of the sensor network, and this is the main optimization goal for our techniques. We show, both analytically and through extensive experiments, that our techniques reduce significantly the energy cost of query processing compared to previously existing techniques.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Today's sensors are no longer just simple sensing devices wired to a central monitoring site, but they have computation, storage and wireless communication capabilities. Once deployed, these new breed of sensing devices can be organized into networks capable of distributed sensing and processing. Technological advances, decreasing production costs and increasing capabilities have made the sensor networks practical for many applications, including environmental monitoring, biological contamination detection, warehouse management, traffic organization and battlefield surveillance. Most sensing devices are battery operated, which highly constrains their lifetime, as it is often not possible to replace their power source. Energy efficient data processing and networking protocols must therefore be developed to make the long-term use of such devices possible.

While the network research community has studied energy efficient solutions in the context of wireless ad-hoc networks, the database community has been confronted mostly with time and storage constraints, but rarely with energy limitations. Therefore, the ability to apply traditional query processing techniques in sensor networks is limited, and different solutions must be found. Query processing is much more difficult in distributed environments than in centralized ones, due to the large number of parameters that affect the performance of distributed queries [OV99]. Distributed query processing has been studied in the context of distributed databases, where the total execution time or the response time of the query are considered good measures of processing cost. These measures are still relevant for the sensor network environment, but the total energy use during query processing becomes of utmost importance. Lower energy use extends the lifetime of the sensor network, making it cost effective for extended monitoring periods.

In spite of its current limitations, the potential benefits of the sensor network technology for real applications have been shown in several test-bed deployments. Intel has used a sensor network to monitor the environmental conditions in a vineyard [Mar03]. By monitoring the vineyard micro-climate, the farmers can determine when and where to irrigate the plants to grow best quality grapes for producing ice wine and other vintages. Intel has also successfully tested the technology in other scenarios, such as detecting faults in machinery at its Oregon chip-fabrication plant by monitoring vibrations [Ric05] and keeping track of visitors and exhibits at a theme park [Mar03]. The Great Duck Island project [MPS+02, GRE] studies the usage patterns of nesting burrows for storm petrels with the help of the sensor network technology, as well as the environmental parameters of burrows and surface during the breeding season. The sensor network technology allows researchers to obtain fine-grain information about the birds' breeding patterns with a much lower impact than human observation, which would cause high mortality rates among eggs and chicks. To understand the role of trees in regulating and controlling the environment, a small sensor network was deployed at Berkeley Botanical Garden [GM04]. The sensor nodes were attached to a single redwood tree to measure its micro-climate at different heights. There are several other scenarios in which the usefulness of the sensor network technology has been tested [Ric05, AFF$^+$03, NL04], a few of which are discussed in Section 1.3.

In this thesis we investigate query processing in the sensor network environment. In our research we focus on query processing in peer-to-peer sensor networks over historical sensor observations, while many of the current research efforts are directed toward sensor networks where there is some form of central administration or the sensor observations are collected in response to a query request. Query processing in peer-to-peer sensor networks is important for many applications due to its scalability and robustness, but little research has focused on query processing in this environment.

The outcome of our research consists of a set of techniques for query processing in peer-to-peer sensor networks. The techniques can be divided into three categories, based on the type of queries they are targeted for. The first category incorporates techniques for efficient processing of spatial range queries. The second category of techniques focuses on lowering the processing cost of queries with joins, where the sensor observations from two network regions must be combined to answer the query. The third category studies query processing from a different perspective, that is, query processing with approximate answering. While the approximate query answer is likely to differ from the exact answer that the techniques from the first two categories generate, the user may consider an approximate answer acceptable if, for instance, the approximate answer can be guaranteed to be close to the exact answer and the cost to obtain it is much lower than for obtaining the exact answer. Disregarding the type of processing, the ultimate goal of each technique is to reduce the energy consumed by the sensor network during query processing.

1.1 Basic Concepts

In this section we cover essential concepts for the understanding of this thesis. We first describe several characteristics of sensor networks, and we use them to divide the types of sensor networks into categories. Then we discuss data storage solutions for sensor networks. Finally, we introduce several query types that are used for extracting useful information from a sensor network. Before introducing these concepts, we define some key terms that we use in this thesis:

- *Sensor [ZG04]:* "A transducer that converts a physical phenomenon into electrical or other signals that may be further manipulated by other apparatus".
- *Sensor observation:* The state of a physical phenomenon as recorded by a sensor at a time instant.
- *Sensor node [ZG04]:* "A basic unit in a sensor network, with on-board sensors, processor, memory, wireless modem and power supply". We also refer to it as *node*.
- *Sensor network*: A set of sensor nodes organized into a network that pool their resources together for user-defined tasks (e.g.: monitoring).

- *Network topology:* A graph representation of the sensor network, where the nodes are representing the sensor nodes and the edges are representing the direct communication links between the sensor nodes.

1.1.1 Sensor Networks

Technological advances have lead to the production of a new breed of sensor devices. Besides one or more sensors, the new sensor devices have also units for data storage and processing, as well as wireless communication capabilities. The continuous reduction in production costs will soon make feasible their deployment in large numbers (from hundreds to tens of thousands). Once deployed in a region of interest and activated, the sensor devices start communicating with each other, thus forming a network of sensors nodes over the monitored region. A sensor node can communicate directly only with other nodes located within its wireless communication range. With nodes located farther away, the communication can be done through one or more intermediate nodes, a process called multi-hop[1] routing.

In spite of the relative novelty of the architecture and the small number of real-life deployments, sensor networks are considered a highly promising technology that will change the way we interact with our environment [Ric05]. Typical sensor networks will be formed by hundreds to tens of thousands of small, radio-enabled, sensing nodes. Each node is capable of observing the environment, storing the observed values, processing them and sharing them through wireless communication. While most of these capabilities are expected to rapidly grow in the near future, the energy source, be it either a battery or some sort of energy harvesting [JPC05, RKH$^+$05], will remain the main limitation of these devices due to the relatively slow progress in energy storage and harvesting technologies. Nowadays sensor nodes have a large variance in capabilities, ranging from the bulky and powerful Sensoria nodes [Sen] to the small but limited Mica Motes [Cro].

There are several characteristics that can be used for differentiating the types of sensor networks. In this thesis we differentiate the sensor networks based on the following characteristics: the availability of network information at nodes, the node mobility, and the strategy used for information collection.

When the availability of network information at nodes is considered, there are two basic types of networks:

- *Centralized sensor networks.* In this type of sensor networks, information about all sensor nodes is available at one node (called base station), or at several nodes. Many research works assume the existence of a base station, which is a network node with increased storage, battery and processing capabilities [DKR04]. Typically the base station is a computer connected to an regular power source (e.g.: power grid). The information collected by the sensor network can only be obtained through this node. An advantage of this centralized solution is that data can be extracted from the sensor network in an efficient manner as the base station knows the network topology. Thus, the best way of contacting a node can be determined off-line without accessing the sensor network. A disadvantage of this solution is that the sensor nodes located closer to the base station are prone to shorter lifetimes as they are subject to more traffic than the nodes located farther away from the base. In addition, gathering and updating the

[1]Two nodes communicating directly are considered one hop away from each other.

topology information at the base station for a large-scale sensor network can be very costly, or even unfeasible when the network is very dynamic.

- *Peer-to-peer sensor network.* Unlike the centralized sensor network, in peer-to-peer sensor networks no node has global knowledge of the network. Similar to the wired peer-to-peer networks for data/file sharing, a sensor node is only aware about a small subset of the network nodes, called peers. Different from the wired networks, a node's peers are selected based on their location rather than their identifier or stored data. Typically, the peer subset is formed by the other nodes located within the wireless communication range of a node, called neighbours. This type of organization has several advantages: a) it is flexible, with any dynamics of the nodes being dealt with only locally; b) it is more resource balanced, as typically all sensor nodes have similar capabilities and any node could be used to access the network; c) it is more robust and secure, the corruption of one node having only a reduced effect on the overall network; d) it is scalable, an increase in the number of sensor nodes producing an increase in the network's resources with typically no increase on the resource usage of a particular node. These benefits come with increased costs and complexity for information collection and processing. Peer-to-peer sensor networks may be the only way to achieve the large scale needed for some applications [ZG04].

Hybrid sensor networks which borrow some characteristics from each of the two basic types are also possible, with some sensor nodes having more resources and information than others. No sensor network organizations is best for all situations, the best type of sensor network is application specific.

A second criteria for differentiating the sensor networks is the mobility of the nodes:

- *Fixed nodes.* The sensor nodes do not change their location. Nodes do not have to update their location information, and a node's neighbours are the same over its lifetime. Transient or permanent failures among a node's neighbours are possible, and therefore the sensor nodes may still need to update their neighbours' information occasionally, or to inform the base station on the failure.

- *Moving nodes.* The nodes can change their location, which leads to possibly rapid changes in the network topology. Thus, the sensor nodes may need to regularly update their information about their neighbours. Also, node mobility in centralized sensor networks may incur high costs for updating the nodes' locations at the base station.

The strategy used for collecting the information from the sensor network can be also used for differentiating the types of sensor networks:

- *Pull-based collection.* In this type of information collection, data is retrieved from the network in response to queries from users. A user issues a query, which is introduced in the network through the base station or through one of the sensor nodes. The query is then processed by the network and its answer is returned to the user.

- *Push-based collection.* In sensor networks with push-based information collection, the sensor nodes initiate the data retrieval in response to certain events. When the sensor nodes detect an event of interest, the nodes announce the event to the user by sending a notice of the event to a predefined node (e.g., the base station) or a set of nodes. A network using push-based information collection can be seen as being pull-based, with queries for the events of interest being continuously executed.

A sensor network could also allow both strategies for information collection if some sensor observations are available in response to queries from users, while other observations are pushed to the users when they occur.

1.1.2 Data Storage in Sensor Networks

The purpose of sensor nodes is to observe the environment around them. The environmental sensing can be done either automatic or in response to queries from users. In turn, the automatic sensing can be periodical (typical for monitoring phenomena), or triggered by actions in the environment (e.g., motion for surveillance applications). Once an observation is performed at a sensor node, there are three main possibilities for data storage:

- *External storage.* When an observation is acquired, it is immediately sent for storage and processing to a location external to the sensor network, such as a base station. This solution simulates the behaviour of traditional sensing systems with nodes not capable of storage and processing, and it incurs high communication costs for transmitting wirelessly every observation to a remote location. Nevertheless, this solution should be considered when the majority of sensors' observations are required by the user. In this solution the queries are answered outside the network.

- *Local storage.* Each sensor node stores its observations locally for further processing or future retrieval. A sensor node with 1 megabyte of memory observing once every 5 minutes a phenomenon and storing its value in 4 bytes together with a 4-byte timestamp could store more than 1 year of raw data, which is likely beyond the expected lifetime of a typical sensor node. This solution has the advantage of reducing the cost of storage compared to the external storage solution, but, on the other hand, the cost of retrieving the data for answering a query is higher.

- *Network storage.* Instead of storing their own observations locally, sensor nodes send them for storage to other nodes, possibly determined with a hash function applied to the observations collected by the sensors [RKY$^+$02]. Even though the cost of sending each observation to another sensor node could be quite high, the cost of query processing is reduced. This is due to the use of the same hash function during querying, which provides the sensor nodes (or their location) that hold the observations required for answering a query. This storage solution is suitable when the number of observations stored in the network is low and many queries are issued [RKY$^+$02].

Sensor network applications require different utilizations of network resources. The number of collected observations together with the number and the type of queries used for their retrieval determine the resource usage patterns, which in turn determine the most appropriate storage solution for an application [SRK$^+$02].

1.1.3 Data Model for Sensor Networks

So far we have discussed how the observations of a sensor node are performed and where they are stored. A complementary issue is what is stored to make the collected observations useful. A sensor node can have several sensors (e.g., temperature, humidity, light), each

Sensor node location	Sensor type	Time of observation	Value of observation
N53.30567, W113.20795	temperature	11:55:06 07/05/2004	17.4
N53.30567, W113.20795	humidity	11:56:44 07/05/2004	56.5%
N53.30579, W113.20790	light	11:55:06 18/05/2004	9751
N53.30579, W113.20790	temperature	11:55:06 07/05/2004	17.6
N53.30572, W113.20787	temperature	11:56:06 07/05/2004	17.5
N53.30567, W113.20792	humidity	11:56:44 07/05/2004	56.7%

Table 1.1: A possible logical view of a sensor network database

sensor working independently from the others. Each observation has attached to it the time-stamp corresponding to the time of observation, and the type of sensor that has recorded it. In addition, the location of the sensor node that recorded the observation is added when the observation is sent to a different node than where it was recorded. For most scenarios, users are interested in the observations of several sensor nodes. If these observations must be combined (e.g., to obtain aggregations), the observations recorded at the same time by the sensor nodes must have attached the same time-stamp. In any case, some level of time synchronization is required to capture the temporal relation among the observations of the sensor nodes. Several algorithms [SY04] have been proposed for time synchronization at the sensor nodes. Another solution is the use of a GPS [Kap96] device to ensure the synchronization.

Considering the above model for storing each observation and disregarding the exact storage solution adopted, the database within a sensor network can be seen logically as a relation whose tuples describe the sensor observations (e.g., see Table 1.1). The relation is continuously updated with new sensor observations, and it is append-only. The advantage of such a logical view is that it separates the user from the details of the actual organization and implementation of data storage in the sensor network.

When the observations are stored at the sensor nodes (either local or network storage), the storage capacity is very small compared to external storage. If a node's storage space becomes full, there are two typical solutions for freeing up the storage space. When the users are only interested in the recent sensor observations, the older sensor observations can be discarded (e.g., using a FIFO policy) without affecting the answers to the user queries. When the users are interested that queries about the more distant past can be also answered in some form, the sensor nodes can store statistical summaries of the older data that it has been discarded. For instance, one could adopt the data stream storage solutions for fixed storage space such as those proposed in [ZGTS03]. The solution uses temporal aggregations over the data stream at multiple time granularities. The aggregation level for a data record is dependent on the age of the record, with only the most recent data fully stored. The aggregation levels and their granularity would be decided before the network deployment and are dependent on the observed data and the storage size. Similar to any approximation technique, this storage solution would influence the accuracy of some of the query answers as older data is aggregated, and therefore some information is lost.

1.1.4 Query Types for Sensor Networks

Sensors can generate large amounts of data, and it is typically not possible to collect all the data for external processing or storage due to the high cost of retrieval and the limited energy resources sensor nodes have. In this respect, the role of queries is important for extracting useful information from the sensor network. In the following we list several query types used for querying sensor networks:

- *Historical queries.* When local or network storage of sensor observations is used, this type of query selects for retrieval certain observations or events of interest that have been collected in the past and are stored in the network. An example of a historical query is: "What temperature observations have been collected so far?".

- *Continuous queries* (also called long-running). This type of query is useful when a user needs real-time data about the monitored region. Typically, sensor nodes start collecting sensor observations from the moment they receive the continuous query. It is common that no data is locally stored, but the collected observations are sent immediately to the user. An example of a continuous query is: "What are the temperature observations for the next 10 minutes?". Push-based data collection in sensor networks can be seen as a continuous query for an event over infinite time.

- *Temporal range queries.* A sensor can collect large amounts of data and users may be interested only in a subset of these data. Using a temporal query the user can specify a time range (in the past or future), expressing its interest in the sensor's observations recorded during that time range. Most queries used in sensor networks have a temporal component since typically the interest of users is on the sensor observations taken in a time range. An example of a temporal range query is: "What temperature observations have been collected yesterday between 3pm and 4pm?".

- *Spatial range queries.* In some applications users are interested in retrieving at different moments sensor observations taken in subregions of the entire monitored region. In this case the user specifies the region where the observations to be retrieved should belong to. An example of a spatial range query is: "What temperature observations have been collected in region 21?".

- *Join queries.* Individual observations cannot provide the desired information in some applications. Several observations from one or more sensor nodes must be combined to find the query answer. An example of a join query that could be used for tracking is: "What animals have been detected in both regions 17 and 21?".

- *Status queries.* This type of query is different from the previous types in the sense that it is not concerned with the sensor observations, but with the sensor nodes themselves. They are generally used for monitoring the status of the sensor network. A possible query is: "How many sensor nodes are active?".

When the sensor observations are not needed in the form they are generated or the cost of retrieving them is prohibitive, two solutions are typically considered:

- *Queries with aggregate answers.* Aggregation is a data summarization technique where statistics such as minimum, maximum, sum, average and count are applied to the sensor observations. The size of most aggregations is very small compared to the size of the aggregated data, and it is typically independent of the size of the data.

- *Queries with approximate answers.* Approximation is a data reduction technique where the sensor observations are replaced by approximated values. While the approximations may be used to reconstruct the original sensor observations (with a certain quality loss), this is typically not possible with aggregations. For approximation there is a trade-off between the quality of approximation and the number of values used for representing the sensor observations, where an important factor is the amount of redundancy and correlation in the sensor observations.

The query types presented above are not mutually exclusive. Typically, a query belongs to more than one query type. For instance, you can have *continuous temporal aggregate queries* such as "What is the average temperature each hour for the next 10 hours?" or *historical spatiotemporal join queries* such as "What animals have passed through both regions 17 and 21 yesterday?".

To harness the power of the sensor network technology, we believe that a sensor network should not be restricted to just one type of query, but allow several query types be executed in the sensor network. For instance, a user may use a continuous query to collect the real-time sensor observations from an area of interest and a historical query with aggregation to gather statistics of the sensor stored data from the entire network. Processing techniques designed to the specifics of each query type are essential in order to preserve the limited resources sensor nodes have.

1.1.5 Challenges and Opportunities

Sensor networks represent a challenging environment due to their limited resources and distributed nature. The ultimate goals are to maintain their scalability and robustness, while extending their lifetime through energy-efficient operation. Several important issues that must be considered when designing new query processing solutions for sensor networks are:

- *Energy conservation.* As previously discussed, this issue is essential for the success of the sensor network technology. The lifetime of the sensor nodes must be extended to allow the long-term operation of the sensor network.
- *Scalability.* As the network size increases, the costs of operating the network should scale well. In addition, the network operation should not become concentrated at any sensor node(s) with the network size increase.
- *The dynamic nature of sensor networks.* A sensor network should be able to accommodate additional re-deployments, for instance, to extend the network capability or lifetime. Sensor node failures (permanent or transient) are expected given the limited energy source, as well as the unfriendly operating environment in most applications.
- *Flexibility of topology.* Any solution for sensor networks should consider this issue. Realistic node deployments cannot guarantee any particular topologies. The dynamic nature of sensor networks may also cause changes in the network topology.
- *Fairness.* No node (or set of nodes) should be a concentration point of communication over the lifetime of the sensor network. Such nodes would deplete their energy resources much faster than most other nodes, which would lead to their early failure.

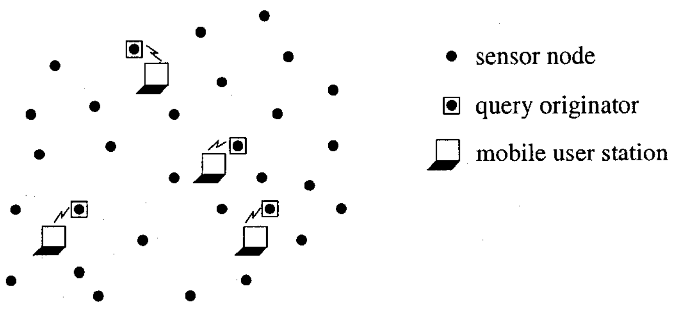


Figure 1.1: User interaction with the sensor network

- *Redundancy and correlation in sensors' observations.* As sensors observe the surrounding environment, it is natural to consider the correlation between the monitored phenomena, and the redundancy of the measured observations at sensor nodes located in the proximity of each other. Query processing solutions should take advantage of such correlation and redundancy to reduce the cost of query processing.

1.2 Our Sensor Network Environment

In this thesis we consider a *peer-to-peer sensor network* formed by a large number of *fixed* sensor nodes, where no global knowledge of the network is available. Each node has several sensing units (e.g., temperature, humidity, RFID reader), a processor, internal and flash memory, a fixed-range wireless radio and it is battery operated. These characteristics encompass a wide range of sensor node hardware, which extends the applicability of our research. Each node is aware of the nodes located within its wireless range, which form its neighbourhood. A node can address a message to one of its neighbours (unicast) or it can address the message simultaneously to all its neighbours (broadcast), and it can communicate with nodes other than its neighbours using a multi-hop routing protocol over the wireless network. Each node is aware of its location, as well as the location of its neighbours (acquired during its activation in the network). The location information is periodically refreshed to account for any network dynamics (e.g., node failures and new deployments).

Sensor nodes *collect observations periodically* and *store them locally*. The periodicity of observations could be predetermined or set through a special message sent to each node. Each sensor node has several sensors, each sensor with its own periodicity of observations. Each observation has attached to it a time-stamp corresponding to the time of measurement. Since each observation stored in a node's database has associated a time of acquisition and each node has a location, the sensor networks is, on a global view, a distributed database storing spatiotemporal data, where the data is partitioned horizontally on the location attribute, with each partition stored locally in a node's database. Note that a sensor node with 1 megabyte of memory measuring once every 5 minutes a phenomenon stored as a 4-byte value together with a 4-byte time-stamp could store more than 1 year of raw data. If the application requires high sensing rates or long-term storage is expected, data stream storage solutions for fixed storage space can be used (e.g., see [ZGTS03]).

As sensor nodes are not designed for user interaction, users access the sensor network through user stations, which connect to one of the sensor nodes in their vicinity (Figure 1.1). The sensor node communicating with the user station acts as a gateway in the sense that the node receives the user queries from the user station and returns the query answer to it, without other nodes being aware of the user station. We call this node *query originator* in the following.

Similar to other works on query processing in sensor networks [MFH02, YG02], we consider a declarative SQL-like language for the user to express queries over the sensor network, where the data collected and stored in the sensor network is represented by one virtual append-only relation, denoted as R^*. For instance, the following query can be used to find the average temperature and the lowest humidity collected each day by the sensor nodes located in the area represented by the bounding rectangle $(< x_1, y_1 >, < x_2, y_2 >)$ during the time range (t_1, t_2) when the temperature was above freezing (0 °C).

SELECT	DAY(R^*.time), avg(R^*.temperature), min(R^*.humidity)
FROM	R^*
WHERE	R^*.location IN $(< x_1, y_1 >, < x_2, y_2 >)$ AND
	R^*.temperature > 0
GROUP BY	DAY(R^*.time)
HAVING	DAY(R^*.time) IN (t_1, t_2)

As nodes store the acquired data locally, each node holds the values of the observations recorded by its sensing units and the time when each recording was performed. On a conceptual level, the schema of the relation stored at a node N_j is $R_j(loc_j, time, s_1, \ldots, s_S)$, where loc_j denotes the location of the sensor node N_j, s_i is a recorded value for sensing unit i and S is the number of sensing units. The virtual relation R^* is the union of the node relations $R^* \leftarrow \bigcup R_j$, $j = 1..N$, where N represents the number of sensors nodes. Note that an actual implementation may use a different organization. For instance, loc_j should be stored only once in N_j.

1.2.1 Radio/Energy Model

Sensor nodes are formed by several units, each with its own energy requirements. As sensor nodes spend most of their energy supply during communication [ASSC02], we are only considering the communication cost in this thesis. In addition, our work focuses mostly on reducing the amount of communication, while the operation of other sensor units is only marginally affected.

In order to compare the performance of different query processing strategies, we need a model to represent the energy used for communication by the radio unit. According to Rappaport [Rap96], the energy used to transmit a bit in wireless communication is given by:

$$E_t = \alpha + \gamma \times d^n, \tag{1.1}$$

while the energy used for receiving a bit the cost is:

$$E_r = \beta, \tag{1.2}$$

where d is the distance to which a bit is being transmitted, n is the path loss index, α and β capture the energy dissipated by the communication electronics and γ represents the energy radiated by the power amplifier. In [Hei00] the following values are used to model a sensor's radio: $\alpha = \beta = 50$ *nJ/bit*, $n = 2$ with $\gamma = 10$ *pJ/bit/m*^{2} when the wireless range is less then 87 meters, and $n = 4$ with $\gamma = 0.0013$ *pJ/bit/m*^{4} for ranges over 87 meters. We consider a

Table 1.2: Parameters of the sensor network and other notations

Notation	**Description**	**Value**
R_A	Sensor network region	1000x1000 meters
W	Wireless communication range	50 meters
N	Number of sensor nodes	varies
N_n	Average number of neighbours	varies
E_t	Energy required to transmit a bit	75 nJ/bit
E_r	Energy required to receive a bit	50 nJ/bit

Table 1.3: Energy cost of accessing common sensors [MFHH03]

Sensor	**Energy** (nJ/sample)
Light, temperature	90
Accelerometer	180
Magnetometer	1500

simple radio device that transmits all messages as far as the wireless communication range[2]. For consistency with other sensor network research [YG03], we use in our simulations a wireless range of 50 meters. Plugging the numerical values into equations (1.1) and (1.2) we obtain the following energy costs for communication: $E_t = 75$ *nJ/bit* and $E_r = 50$ *nJ/bit*.

1.2.2 Experimental Setup and Parameters

In our evaluation, we consider that the sensor nodes' placement follows a uniform distribution over a two dimensional region, denoted by R_A. The number of sensor nodes is denoted with N. The wireless range of a node is equal to 50 meters and it is denoted with W. The sensor network parameters and common notations used in our investigations are summarized in Table 1.2. Note that for some parameters the value varies in our evaluation.

In our experimental evaluation all measurements are averaged over 100 randomly generated sensor networks, with 10 random queries executed over each network. We focus on the energy efficiency of the query processing solutions only, making the measurements independent of the characteristics of the MAC layer (for instance 802.11 radios consume as much energy in idle mode as for receive mode, while other radios may switch to a low-energy state when idle). We assume that the message delivery is instantaneous, which allow us to simulate sensor networks with thousands of nodes. In reality, some techniques may incur a delay due to packet contention on the wireless communication channel. In our evaluation we do not include the cost of acquisition for the sensed values. These values are recorded periodically independently of the strategy used for query processing. Table 1.3 presents the cost of acquisition for some typical sensors (costs obtained from [MFHH03]).

[2]As typical sensor nodes do not have sophisticated communication electronics capable of varying the transmission range [DGR$^+$03], we consider that all messages are transmitted as far as the wireless communication range. This is an ideal case, as in reality the environment affects the wireless range differently on each direction for the same transmission power.

1.3 Motivating Applications

The wide range of sensor networks configurations allows their use in many applications [Ric05, Mar03, GRE]. In the context of the definitions introduced above we present several applications that require the use of queries over the historical sensor observation stored at the nodes in a peer-to-peer sensor network environment.

Environmental Monitoring. Protecting the nature reserves is the focus of many agencies. The existing monitoring techniques are typically limiting due to either cost (i.e., satellites or manpower) or scale of application. Sensor network technology could provide new ways for monitoring the environment, with potentially low-cost for large-scale monitoring. A sensor network is deployed over a nature reserve, with the task of monitoring various phenomena (e.g.: temperature and humidity) and well as observe the animals (sound, video or RFID sensing). Park rangers patrolling through the monitored region can query the network for information of interest through any sensor node in their proximity using a mobile computing device. The network's querying capabilities should allow the ranger to express several query types and query processing should be optimized for energy-efficient processing of each query type. For instance, when certain events such as vegetation diseases or small fires are observed in an area, the ranger could query the network about historical observations, which may help one understand what have caused such events or learn about other areas that are threatened by similar events. To understand what may have caused a small fire in a forest patch, the ranger could ask the query *"What were the temperature and humidity conditions in the patch for the past 72 hours?"*, which is a historical spatial range query. On the other hand, if several forest patches are affected by small fires, the ranger could ask the join query *"What were the common conditions in all the affected patches?"* to check if there were any common temperature and humidity conditions that could have triggered the small fires. Yet another query could be *"What was the temperature distribution in the last 6 hours over the western region of the reserve?"* to determine what other patches are at risk and require immediate attention. Upon finding a forest patch at risk, the ranger could ask the continuous query *"Send me all temperature reading higher that* $45\ ^\circ\mathrm{C}$ *from the patch at risk when humidity is lower than 20% in the same patch"* to continuously monitor the forest patch at risk. Similar environmental monitoring could be used in other applications as well, such as farm monitoring in agriculture.

Traffic Monitoring. Traffic monitoring is very important for large metropolitan areas, not only for controlling the traffic flow but also for enforcing traffic laws or monitoring criminal activity. As the cost of installing power and communication lines is very high, the city police and administration could use an autonomic wireless solution for detailed large scale monitoring. Each monitoring device would consists of a camera for sensing, computation and storage, specialized hardware for image processing (e.g, for detecting cars or license plates), equipment for wireless communication and a solar-rechargeable power source. The cameras could take snapshots periodically and store them locally. The collected snapshots could expire after a predefined time frame. The monitoring devices would form a distributed network. While the nodes of this sensor network have more capabilities than those used in the environmental monitoring application, they are still constrained on the energy resource available at any given time. The city administration could use a historical spatial range query such as *"How many cars have passed through intersection 29 every* *morning of last week in each direction?"* to optimize the street lights' timings and obtain a better traffic flow. The city police could ask the historical spatial range query *"What cars have passed through intersection 29 yesterday around midnight?"* to find the suspects of a nearby burglary. Then they could used a continuous query such as *"Alert when the car with license plate 4R5T3 is detected"* to find the suspects. Using a join query such as *"When did the buses with license plates in a given list have passed through intersection 29 last week"* could be used by the public transportation administration to inform the public more accurately on the bus schedule.

Warehouse Management. The condition of storage and transportation for many goods is important for guaranteeing their quality at the time of sale. When leaving the production line, each sensitive product can have attached a sensor node monitoring its temperature and, for instance, suffered shocks. Before a sale, the warehouse management may want to check if the products to be sold were handled appropriately since the last check was performed (e.g., when the product was bought by the warehouse). While the products' condition could be checked for each product manually, it is an unfeasible procedure for large warehouses with tens of thousands of products sold daily. Using the sensor network technology, the check could be performed using a historical spatial range query. The query's spatial window would correspond to the area where the products in the current sale are located in the warehouse and the temporal range would be between the last time the products' status was checked and the current time. Products would have to be handled individually only when the associated sensor nodes report problems.

Military Surveillance. Assessing the conditions of a territory in conflict zones is important for the safety of friendly units. A sensor network can be deployed over the conflict region to monitor the presence and movement of military units. When a friendly unit must move to a different area in the conflict region, a query can retrieve the observations of the sensor nodes located in the area of interest. The observations can provide information about past or current presence of unfriendly units in the area or in its neighbourhood. Past observations can also tell about the movement of units and provide insight on their possible actions. While information about the conflict areas can be obtained through satellite surveillance, sensor networks can provide more accurate information at substantially lower costs. Smart sensor node deployments can also help detect units not visible to a satellite's line of sight. In addition, the sensor network could be used for multiple tasks, such as detection of biochemical hazards and inter-unit communication.

Our query examples for the motivating applications cover several types of queries. We believe that real sensor network deployments will have multiple roles, ranging from long-term monitoring to event alerts. It is unlikely that a processing solution could perform best for all query types. As the energy efficiency is of primary concern, each type of query should be optimized differently and several processing solutions will coexist in the network.

1.4 Our Contributions

The purpose of a sensor network is to observe the environment and collect data. Nevertheless, sensor networks are useful only if users are able to extract these information from the network, which is typically done through querying. Query processing is essential to most applications of sensor networks and our research efforts have focused in this direction. There are several types of sensor networks configurations (introduced in Section 1.1.1) and several types of queries (introduced in Section 1.1.4). In our research we have studied the processing of three types of queries over historical sensor data in the peer-to-peer sensor network environment detailed in Section 1.2.

1.4.1 Processing of Spatial Range Queries

Most query processing solutions in the literature consider that each query is targeted on retrieving data from the entire monitored region (e.g., [YG02, MFH02]). While this assumption may hold for small-to-medium scale sensor networks, it is unrealistic for large scale networks. For such networks, users will rather focus their interests on subregions of the network region at a given time to answer region-related questions. Thus, the queries will be targeted on retrieving sensor data from only small subsets of the sensor network. The spatial constrain is typically expressed in the query by specifying the spatial range where the sensor data must belong to.

In this thesis we focus on energy-efficient processing of queries with spatial range predicates. We propose several query processing solutions that use the spatial range predicate for reducing the cost of query processing. Our adaptive processing solution performs substantially better for small-to-medium spatial range queries than the typical processing solution that does not take advantage of the spatial range constrains. For large spatial range queries, including queries covering the entire network region, our adaptive solution approaches the behaviour and performance of the typical solution, ultimately degenerating into it. Our preliminary results on processing spatial range queries have been published in [CNS04].

1.4.2 Processing of Join Queries

A common type of query in traditional database systems is the join query. In the case of sensor networks such a query would allow one to find correlations in data between different network regions or to filter the sensor reading using a predefined relation. Join queries are more complex to process in sensor networks than other queries due to the distributed nature of the environment. When the join operator is not processed in the network, users can use queries wihtout joins (such as spatial range queries) to extract the data participating in the join and process the join externally. Such an approach may be cost effective for joins generating much data, but may incur a substantial overhead compared to the in-network join processing when the join is highly selective.

In this thesis we investigate where and how to process join queries in sensor networks. We propose several join processing strategies and compare them with respect to the network location where the join is processed. We construct cost models for each strategy that allows a query optimizer to select the best processing strategy and location for a given join query. We also describe in detail a distributed in-network join algorithm that could be used by most join processing strategies to execute the join operation. The strategy adapts to the characteristics of the query and sensor network to minimize the energy cost of processing. Our results on processing of join queries have been published in [CNS07, CN07].

1.4.3 Processing of Queries with Approximate Answers

When sensors observe physical phenomena (e.g., temperature), the observations may not vary greatly over short periods of time and over small areas, and they are typically correlated. Retrieving all sensor observations relevant to a query may not be necessary in this case. By selectively collecting only some of the relevant sensor observations, a query answer can still be provided. While the answer will be only approximate, it may be sufficient to the user. Reducing the size of the data that is transferred by the network during query processing lowers the cost of communication, leading to increased network lifetime.

In this thesis we investigate a query that requires an approximated answer in the form of a data distribution map. Considering a user specified threshold with respect to the accuracy of the map data, we propose three query processing strategies that are able to construct the query answer using only a subset of the relevant nodes. These strategies substantially reduce the energy cost of query processing over a typical solutions while providing the user with a query answer within the required accuracy threshold. Our results on processing of queries with approximate answers have been published in [CNS05].

1.5 Thesis Organization

The thesis is organized as follows. Chapter 2 surveys existing works on query processing in sensor networks, as well as other relevant topics to our problem. Chapter 3 focuses on the processing of *spatial range queries* and introduces algorithms and models for its energy efficient processing. In Chapter 4 we investigate several processing solutions for *join queries*. We introduces cost models that allow a query optimizer to select the most energy efficient processing solution for a given join query. Chapter 5 studies a special type of an approximate query. The processing solutions proposed in the same chapter use only a subset of the relevant sensor nodes to answer the query, thus reducing the energy cost of processing. Experimental evaluation for each type of query is presented and discussed within the corresponding chapters. Chapter 6 concludes our thesis and discusses several future research directions.

Chapter 2

Related Works

Sensor network technology lays at the confluence of many disciplines, including data management, networking and distributed systems. Current research efforts address various aspects with different perspectives on the existing issues. From the plethora of publications investigating sensor networks, we present mostly research works on query processing in sensor networks, as this is the focus of our thesis. Nevertheless, data storage and data exchange affect in a great measure the processing of queries. Thus, we also discuss several works focusing on data routing and storage in wireless sensor networks.

2.1 Data Routing

In the area of networks research, much work has been done in ad-hoc wireless networks, and, more recently, in the networking aspects of sensor networks. One of the most relevant issues for efficient query processing in sensor networks is position based routing, that is, message routing when the destination node is known and addressed by means of its location. There are several surveys [MWH01, Sto02, GSB03] on techniques for position based routing in ad-hoc networks. Most position based routing algorithms cannot be readily reused in peer-to-peer sensor networks since sensor nodes are only aware of their neighbours and the position of the destination node is not known. Another related topic is geocasting [Mai04, Sto04], which is the problem of routing a message from a node to a group of nodes based on their location (typically within the same area of interest). While geocasting algorithms can disseminate a query to the relevant sensor node, they do not consider the energy cost for returning its answers, which is typically much higher. Thus, most geocasting algorithms are not suitable for efficient processing of queries.

A position based routing algorithm with guaranteed delivery is GPSR [KK00]. Each node participating in the routing only needs to know the location of its neighbours, which makes GPSR suitable for peer-to-peer sensor networks. GPSR incorporates two routing algorithms: a *greedy* algorithm that forwards the message closer to the destination location at each hop, and a *perimeter* algorithm that forwards the message as long as the greedy algorithm fails to find a suitable node for forwarding. In *greedy* forwarding, the current node forwards the message to its neighbour that is located closest to the destination, as long as its own location is not closer to the destination. When greedy forwarding fails, GPSR switches to *perimeter* routing, where the message is forwarded to a neighbour selected using the right-hand rule[1]. The perimeter routing is done on a planar subgraph of the network

[1]The right hand rule states that one can visit every wall in a maze by keeping the right hand on the wall

topology, such as the Gabriel [GS69] or the relative neighbourhood [Tou80] graphs. Once a node located closer to the destination is reached, GPSR switches back to greedy routing. While GPSR requires a node at the destination location to exist, this cannot be guaranteed in peer-to-peer sensor networks due to the lack of global information. A nice property of GPSR is that, with a slight variation presented in [RKY$^+$02], a message can be delivered to the closest node to the destination location. Frey and Stojmenovic [FS06] discuss in details several routing algorithms that combine greedy and perimeter routing. Other algorithms with guaranteed-delivery for power-aware localized routing are discussed in [SD04].

2.2 Data Storage

The location of data in a sensor network affects the decision on how to process a query. Several data storage strategies are compared in [SRK$^+$02, RKY$^+$02]: external storage, local storage, data-centric (network) storage and combinations of them. The authors consider a sensor network scenario where the sensor observations are combined into events, and either a large portion of them or just a summarization is retrieved. They propose a data centric storage scheme based on geographic hash tables, called GHT, which performs best in their scenario. For network environments like ours where many observations are stored with only a small subset of them being retrieved, the local storage of observations is shown to perform the best [RKY$^+$02].

Gummadi et al. [GLG$^+$05] improve the data-centric storage proposed in [SRK$^+$02, RKY$^+$02] on both the storage and querying fronts. Based on analytical models they show how to vertically decompose into sub-relations the sensor relation given a query workload. The authors conclude that in most cases the sensor relation should be fully decomposed to minimize the energy costs incurred for data storage and query retrievals. To improve the performance of query processing, two techniques are employed: query planning with decentralized join ordering and query execution with optimistic join caching. Query planning comes into play when the query contains attributes from different partitions that must be combined to generate the query answer. The authors construct a query plan with an optimal join ordering based on locally stored histogram-based estimations of data distribution. During query execution the authors propose the use of a local caching technique for the results of partial join across sub-relations at each sensor node.

He et al. [HZGS05] investigate in-network storage and querying of data based on the time attribute. They propose the use of a subset of nodes (called rendezvous nodes) to store the readings of the nodes in their neighbourhood. Only these nodes participate in query processing, while other nodes can be in a low-power state. The election and rotation of the rendezvous nodes is done in a localized fashion accounting for the trade-off between energy/storage utilization and load balancing at nodes. Their storage and querying schemes are constructed around the time attribute only, making them unsuitable for efficient processing of queries with several attributes part of the query predicates.

Omotayo et al. [OHB06] study the problem of using the memory of the nodes as a shared resource. The authors consider the problem of using the memory of some nodes to store or buffer the observations of other nodes with the goal of maximizing the size of the history that is stored in the network. The base-station allocates the sensor observations to nodes using global knowledge on the available memory of each node and the nodes' probabilities of observing phenomena.

while walking forward [BMSU01].

2.3 Query Processing

We start by reviewing several works that investigate query processing solutions for retrieving the sensor observations and their aggregations. Then we discuss several works addressing the problem of processing queries with join in a sensor network. Finally, we present several research works focusing on query processing with approximate answering.

2.3.1 Processing of Queries Without Joins

Directed Diffusion [IGE$^+$03] proposes a data-centric framework for query processing in a peer-to-peer sensor network. Nodes do not store historical data and sensing is only performed in response to a query (continuous queries). In Directed Diffusion, the query originator node requests data by sending interests for specific data. The data is then collected by the source nodes and shipped to the originator node. Intermediate nodes can cache and aggregate data, as well as direct new interests based on the cached data. Directed Diffusion uses flooding to find paths from the query originator node to the data source nodes. Path reinforcements are used for selecting a small number of "good-paths" over which the sensor observations are returned. This scheme creates multiple paths for delivery, which increases the robustness of delivery at increased energy costs.

In [MF02], the authors focus on processing continuous queries in a sensor network environment where the information about the existing sensor nodes is available in a catalog at a base station. Sensor nodes do not store historical data but simply collect and transmit the raw data to resourceful proxy nodes that are in charge of further processing and routing the answers to the users. The authors aim at minimizing the used energy by adapting the sensors' sampling and data packet transmission rates. They introduce the Fjord architecture for management of multiple queries. Designed for Berkeley Mica motes and running on top of TinyOS, TinyDB [MFHH03] is an acquisitional query processor that runs on each of the sensor nodes. The acquisitional query processor has the task to decide where, when and how often data is acquired and delivered to the query processing operators [MFHH03]. The focus is on optimizing data acquisition for long-running queries, no data being stored locally at the nodes. The base station optimizes the query by selecting the ordering of sampling and selections before the query is disseminated to the sensor nodes. TinyDB supports both continuous queries and push-based data collection. To reduce the energy consumption, the TAG aggregation technique for continuous queries in networks of TinyOS motes is proposed in [MFH02]. Spatial aggregation is performed in the network by the non-leaf nodes of the query routing tree. Each node combines its data with the answers of its children as the sensors' observations are returned to the base station.

The Cougar project [YG02] investigates techniques for query processing in a centralized sensor network where the location of all sensor nodes is known. In [BGS01] the authors focus on defining a sensor database model for processing continuous queries, which are modelled as persistent views over the distributed sensor database. A central optimizer has the tasks of building a query plan and disseminating it to the relevant sensor nodes. In a similar environment but with emphasis on energy efficient query processing, the Cougar project is extended in [YG03] to address problems such as routing and crash recovery, basic query plans and in-network aggregation.

A typical assumption for calculating spatial aggregations is that the locations of the sensor nodes providing the spatially aggregated values are uniformly distributed in space. This assumption may not hold in reality. In [SS04, SS06], the authors investigate how to weight the sensor measurements in an aggregation, such as *average*, to obtain a more accurate estimation of the real phenomenon. Among the investigated spatial interpolation methods, the area of a node's Voronoi cell provides the best weight toward accurate aggregation values. They propose a local algorithm for creating and maintaining an approximated Voronoi cell of a node during the lifetime of a continuous query.

DIMENSIONS [GEH02, GGP$^+$03] investigates query processing over historical data. The authors focus on multi-resolution summarization of data using the wavelet transform and construct a sensor node hierarchy over the network. Temporal summarization is performed in each node, and each level in the sensor node hierarchy deals with another resolution of summarization. DIMENSIONS is suitable for applications where a query can first look at the data at a coarse resolution and then focus on a region of interest at a finer resolution. The hierarchical scheme is applied in a sensor network deployed in a grid layout where the nodes storing coarser resolution in the sensor node hierarchy are dynamically selected based on their location in the grid.

Redundancy removal for sensor nodes capable of range sensing is studied in [NHZ04, ZNH04]. Zou et al. [ZNH04] consider a peer-to-peer sensor network where nodes do not know their location, and sensor observations are collected at a sink node. The overlap in the sensing areas creates redundancy of observations at the sensor nodes. To save energy, the redundant observations are not returned to the sink, but discarded within the network along the data collection tree rooted at the sink. Several solutions are investigated for constructing the data collection tree. The most energy-efficient tree is obtained when each node chooses as its parent the neighbour located closest to itself among those with lower hop-distances to the sink node. However, this tree does not take into account that nodes may have different energy levels. In [ZNH05], the authors propose the ENCAST algorithm that ensures that the low energy nodes are leaves in the collection tree so that they do not participate in the data routing. Low energy nodes are only allowed to be internal tree nodes when they are needed to ensure that all sensor nodes are connected to the tree. In addition, ENCAST uses a series of energy thresholds to automatically re-adjust the data collection tree when some nodes are over-utilized in the current tree and reach critical energy levels. While the collection cost may no longer be minimal as in [ZNH04], ENCAST ensures that the lifetime of the sensor network is further extended. Another possible direction for dealing with unbalanced workload at nodes is the use of data flow splitting during data collection. However, the problem with data flow splitting is that aggregations become more difficult or inefficient to process. In [ZNH06], the authors investigate combining data aggregation for redundancy removal with data flow splitting. To allow data aggregation, the data is transmitted over one path for the first hop toward the sink, where it is aggregated with the data held by the parent node in the collection tree. Once aggregated, the data is split into and transmitted over multiple flows toward the sink to ensure a balanced workload at the routing nodes.

Many query processing algorithms (e.g.: [MFHH03, YG03]) use for data collection a routing tree rooted at the sink node, which is maintained for the duration of a continuous query. An alternative solution is the use of an itinerary based collection path/tree such as the ones proposed in [XLXM06]. Xu et al. [XLXM06] propose three itineraries for query propagation and data aggregation for processing spatial window queries. After the query message reaches the query window, it is propagated within the window based on a predefined itinerary. Once the window has been fully covered, the query answer is sent to the sink node. Data collection and aggregation are performed simultaneously with the query message propagation, which reduces the number of messages that are transmitted during query processing. Another advantage of the itinerary-based dissemination/collection is that no communication infrastructure needs to be maintained during query processing. The proposed itineraries are only efficient for data aggregation, otherwise the path length over which answers are transmitted to the sink node makes the use of itineraries inefficient.

Silberstein et al. [SBY06] focus on reducing the energy cost of collecting all sensor measurements with continuous queries by using the spatial and temporal correlations present in the monitored data. They introduce two types of constraints for eliminating redundant reports. A node will trigger a report only if its value changes. An edge triggers a report if the difference between the values at its nodes changes. By chaining these two constraint types, their system builds a network of locally maintained constraints that can provide a global view of the changes in the network with minimal energy cost. For networks prone to node failures and message losses, the authors propose the use of redundant constraints.

For applications where the main task is the continuous monitoring of extreme values (MIN and MAX), Silberstein et al. [SMY06] use the history of values to prevent nodes from transmitting. In their HAT scheme, the authors employ localized constraints (in the form of thresholds) for reducing the communication. For its threshold policy setting, HAT exploits the aggregation during different reporting rounds. Moreover, in HAT the thresholds increase monotonically from the leaf nodes toward the base station, each node's threshold acting as a threshold bound for the sub-tree rooted at the node.

Hammad et al. [HAE03] focus on sensor data processing, and propose solutions for data stream joins over the sensor data in tracking and monitoring applications. Demirbas and Ferhatosmanoglu [DF03] propose one of the first index structures for sensor networks. The solution is based on the R-tree and it seems to be energy and time efficient, but no evaluation is presented. Xu and Lee [XL03] propose a window-based query processing technique in a network of moving sensor nodes that do not store historical data but answer continuous queries. Though interesting, their solution has no evaluation.

2.3.2 Processing of Queries with Joins

Bonfils and Bonnet [BB03] consider the problem of processing a correlation operator (i.e., a special join) at a node in the network. The solution starts with a random placement of the operator at a network node. The position is progressively refined by moving the operator to the nodes with lower processing cost during the lifetime of the continuous query. Two important assumptions are that the operator can be fully processed on one node and that the lifetime of the query is sufficiently long to refine the operator position from a random location to an optimal one. An advantage of the refinement is that the operator placements adapts to the change is data during the query lifetime. For short continuous queries their solution would perform much worse than the optimal cost due to the initial random placement, while the solution is inadequate for historical queries. The authors focus on the operator's placement problem, assuming that each node that will hold the operator is able to handle the flow and processing of data alone.

Chowdhary and Gupta [CG05] propose an algorithm for performing joins in-network over a processing region with several sensor nodes participating in the join processing. The authors focus on the *self-join* problem, where two subsets of tuples of the sensor relation R^* are joined. The processing algorithm, called distribute-broadcast join, is a form of distributed block-nested loop join, where each node in the join area holds one block of the outside relation while the inside relation is broadcast over the join region. The authors do not investigate the allocation of memory at the nodes in the join region and the synchronized data flow. The authors consider a special shape for the join region and argue that this region is optimal. Along the same line, Pandit and Gupta [PG06] propose two algorithms for in-network processing of the range-join operator. One algorithm is a distributed form of a hash-join algorithm, while the other is a distributed form of index-join and uses a B-tree structure distributed at the sensor nodes. Both works [CG05, PG06] consider that the optimal join location is the weighted centroid of the triangle. The centroid has the property that it minimizes the weighted sum of the *squared* distances, and thus it is not optimal, as we will show in Chapter 4.

Yu et al. [YLZ06] investigate the processing of self-join queries with equi-joins over historical data in sensor networks. In their solution they constructs a synopsis (e.g., a histogram) of each relation involved in the join. The synopsis are transmitted to an intermediate in-network location, where they are used for finding which tuples of the two relation will certainly not join. This information is returned to the sensor nodes storing the relation, which will use it to select only the join relevant tuples to participate in the join. The join is then performed in network at a second intermediate location. The solution performs best when the join selectivity is high. The join of the synopsis is performed in a square join region whose size is determined based on the size of the synopsis, the network density and the average memory available at the join nodes. When allocating the synopsis to the join partition, they fail to consider the memory available at the individual nodes in their hash-based allocation scheme, which would cause buffer overflows and invalidate the join result. They also assume that nodes have sufficient memory when performing the final join of the filtered tuples.

The *external join* problem where the sensor relation is joined with a relation stored at the user station is studied by Abadi et al. [AML05]. The external relation is basically a relation containing filters to be applied on the sensor tuples. If the external relation is small, it is flooded in the network and the join occurs locally at each node. When the external relation is too large to be stored in the network, the authors propose three techniques (bloom filters, partial joins and cache diffusion) that help filter part of the sensor tuples. Non-filtered tuples are then join externally after reaching the base station. An intermediate situation is when the external relation fits into a group of nodes.

The join operation in the context of streaming sensor data is studied in [AAK06, SFL05]. Ali at al. [AAK06] study the use of a multi-way join operator for detecting and tracking phenomena. Nodes are grouped in clusters and the join operator is first processed at the each cluster-head for the cluster-generated tuples. The cluster-head will then send the result of the join to the other cluster-heads for inter-cluster join processing. Schmidt at al. [SFL05] focus on re-sampling the sensor streams to allow meaningful temporal joins.

2.3.3 Query Processing with Approximate Answers

Query processing with approximate answering in sensor networks has raised much interest from the research community due to its potential to substantially reduce the query processing costs.

In [DGM$^+$04] the query answers are estimated using a statistical model for the sensor observations, where the model captures the redundancy and correlation in sensor observations. The statistical model is constructed at the base station using historical sensor observations and knowledge of the network's topology. Once constructed, the model is used for answering the queries, where the queries are not introduced in the sensor network as most query processing techniques do. The sensor nodes are only interrogated to help refine the model during query answering when the uncertainty of estimates is high. This approach reduces substantially the query processing costs. The user can specify in the query its tolerance for the uncertainty of the model in its approximations.

The authors of [SBLC03, BSLC03] exploit the temporal redundancy in a sequence of sensor observations to reduce the energy cost of aggregation during query processing. Their solution, called TiNA, allows the user to specify a temporal coherency tolerance when an approximate answer is sufficient, which lowers the energy costs. TiNA is designed for a sensor network environment where the sensor nodes answer continuous queries, with no data storage involved.

To improve the fault tolerance of query processing for aggregations, duplicate insensitive sketches are used in [CLKB04] to produce accurate approximations of the aggregate answers. Sketches are bitmaps representations of the data participating in an aggregation. When the answers of the sensor nodes are returned over one routing tree, the failure of a link or node in the tree affects the result of aggregation. When multiple paths are used for returning the answers, the energy cost of computing the aggregations is high, as well as the complexity of combining the answers in the case of in-network aggregations. By employing sketches, the cost and complexity of in-network aggregation over multiple return paths is reduced, and the approximation error of aggregate answers in the case of link and node failures is low.

A solution that combines the advantages of tree-based and multi-path approaches used for data aggregation in continuous query processing is Tributary-Delta [MNG05]. The solution dynamically adapts the aggregation scheme to the loss rate in a network region: a tree-based scheme is used typically by the nodes far from the base station and the multi-path scheme is employed near the base station. The advantage of trees is their low approximation error and short messages, but they do not work well under high loss rates. Multi-path schemes are more robust under message losses, but they incur higher communication costs and introduce more approximation errors. By combining these two communication schemes, Tributary-Delta is able to outperform both approaches in networks where message losses occur at varying loss rates in different regions of a network.

In [DKR04], the authors exploit the correlation and temporal redundancy among the observations on the same sensor node to compress the short-term historical observations. Once compressed, the observations are transmitted to a base station for further processing and long-term storage. The authors use a base signal to encode the correlations among the observations of a node's sensors. They propose the Self-Based Regression (SBR) algorithm with the following tasks: construct the base signal from sensors' observations, reduce the size of the base signal, and approximate the sensors' observations using the base signal.

Caching of sensors' observations is used in [DNGS03] to reduce the cost of retrieving the sensor data, where users specify in the query their tolerance for stale data. The authors focus on XML-based sensor databases, where sensor nodes are distributed over wide spatial areas that can be grouped hierarchically (e.g., neighbourhood, city, county). Using a site naming scheme based on the hierarchy, the queries can "jump" closer to the sites holding the answers.

Kotidis [Kot05] introduces the snapshot queries as a different solution for approximate answering of queries. A snapshot query is a query over a subset of the query relevant nodes, where the nodes in the subset are representatives of their neighbours. Obtaining the answers of the representative nodes gives the user a picture of the distribution of locations and values in the network, and a quick approximate answer. Different from [DGM$^+$04] which uses a global statistical model, a statistical model is used at each sensor node to represent its correlation with the observations of its neighbour nodes. The models are used for selecting in a localized fashion the set of representative nodes.

Another work using statistical models is [SPP+06]. The models are built in a distributed fashion with the goal of detecting outlier measurements. Each node constructs a model of the distribution of its values based on a window of past measurements. The authors use a hierarchical organization of the sensor network to combine the nodes' models so that outlier values can be identified. Different from [DGM+04], the solution can capture special characteristics of the data distribution. Also, in [SPP+06] all data values need to be observed for outliers to be detected, whereas [DGM+04] observes only a few values to refine the centrally constructed model when uncertainty is high.

Chu et al. [CDHH06] propose a system that uses two probabilistic models, one at the base station and the other within the sensor network. By keeping the two models synchronized, the sensor nodes can detect within the network when the approximated sensor values reported to the user by the base station model differ (within an approximation bound) from their measured values. When this happens, the sensors send their measurements to the base station and the models are re-calibrated and re-synchronized. The models take advantage of both spatial and temporal correlations to minimize the cost of model updates. Differently from [DGM+04], the use of the in-network model allows the sensor network to detect outlier measurements.

The correlation between the measurements from different sensed attributes within a sensor node is exploited in [DGHM05] to generate conditional query plans, where low cost attributes are used to determine the best plan for acquiring the high cost sensor measurements.

Event detection based on contour maps is studied in [XLCL06]. A contour map displays the distribution of attribute values over the network. The authors assume a grid on top of the network with at most one node per grid cell. For the cells without nodes, their values in the contour map is interpolated from the values of the neighbouring cells. The system detects three types of events based on the distribution of values in the contour maps. To reduce the high cost of building the contour map for a given time snapshot, the authors propose a scheme for incremental map update during the lifetime of the continuous query.

Spatial and temporal correlation in sensor data is exploited in [YS05, YS07] to reduce the cost of in-network data aggregation. The authors propose clusters of nodes as the basic unit of data reporting, where all nodes in the cluster sense similar values and only the cluster-head reports its sensed value. For snapshot queries the spatial correlation is used for cluster formation, while for continuous queries the temporal correlation is used as well for cluster updates. In addition, for continuous queries the cluster size is used for weighting the reported value in order to improve the accuracy of data aggregates. The user must specify in the query an upper-bound error threshold, which is used for measuring the nodes' intra-cluster similarity of sensed values. The proposed algorithm, called CAG, provides an approximate aggregation result with a small and bounded error, while substantially improving the energy efficiency of in-network data aggregation compared to TAG [MFH02].

The problem of retrieving only the top-k measurements from a sensor network is studied in [SBE+06, WXTL06]. In [WXTL06], the authors propose a technique using expected value ranges at each sensor node for monitoring top-k queries. The non-overlapping ranges tell the base station which nodes are in the top-k answer. When a new measured values falls outside the expected range for a node, it is sent to the base station where the value ranges are globally re-adjusted. Since only the out-of-range values are reported to the base-station, the cost of maintaining the top-k nodes with values is reduced. In [SBE+06], the authors argue that using probabilistic models such as those proposed in [DGM$^+$04] is very expensive for top-k queries. They propose the use of models based on past samples, since such models allow inexpensive use and maintenance. The proposed algorithms construct linear programming-based query plans using the models stored at the base station, the network topology and the energy constraints. The generated plans allow approximate top-k query answering with the highest possible accuracy within a targeted energy budget.

2.4 Other Relevant Issues

A topic relevant to query processing in sensor networks is that of query processing in traditional distributed database systems. Query processing is much more difficult in distributed databases than the centralized ones, due to the large number of parameters that affect the performance of distributed queries [OV99]. Distributed query processing [Kos00] has been studied in the context of traditional distributed databases, where the query optimization criteria are the total execution time or the response time of the query. The problem of query processing can itself be decomposed into several problems: query decomposition, data localization, global query optimization and local query optimization [OV99]. For traditional distributed databases, the first three problems are addressed at a central site, with only the last solved at the local sites.

The peer-to-peer sensor network environment further increases the complexity of query processing, making the global query optimization not practical, while the other three problems require distributed solutions. Major differences from traditional distributed databases is that in a sensor network the database relations are distributed over devices with limited capabilities and hard constrains on the energy resources, that the amount of information available at each sensor node about the other partitions of the data (data location, distribution of values, etc.) is limited (if not null), and that the nodes communicate wirelessly, with all the issues this communicating environment brings into play. Nevertheless, there are also a number of similarities with respect to query processing: the query processor needs to select the partitions of the relation that must be used to construct the query answer; the query optimizer must choose the location (server, node or neighbourhood) where each operator needs to be processed; and the optimizer has to use cost models to choose the best plan and estimations affect these models. Several issues and solutions for query processing in traditional distributed database systems are presented in [LOT94].

The most common way of expressing queries for a sensor network is using a declarative language. Most works use modified versions of SQL, which allows special constructs for describing the continuous query [YG03, MFHH03] or the the queried event [XLCL06]. Chu et al. [CTH06] propose the use of SNlog language, which is a variant of Datalog. They argue that SNlog allows the rapid, flexible and efficient construction of sensor applications. For instance, using SNlog the user is able to easily compose high level events from the sensor measurements.

As more software and hardware platforms become available, application development becomes a time-consuming task. In addition, the heterogeneity of platforms makes fast deployment of new sensor networks difficult, and re-deployment of new types of nodes within an existing network likely impossible. Middleware technology is an attractive solution for dealing with the heterogeneity of sensor platforms. In [AHS06], the authors propose the Global Sensor Networks (GSN) middleware. GSN provides flexible integration of sensor platforms and provides distributed querying, filtering and combination of sensor data. The main level of abstraction used in GSN to allow these functionalities is virtual sensors. A virtual sensor abstracts from implementation the access to sensor data and corresponds to data streams either received from real sensor nodes or derived from other virtual sensors. By hiding the data sources behind the virtual sensor abstraction, GSN provides a uniform access to heterogeneous architectures. An infrastructure that provides a unified view of data provided by sensor networks with data from other types of devices (such as wired in-situ sensors and remote sensors) is GeoSWIFT [LCT05]. GeoSWIFT contains three layers, i.e., Sensor Layer, Communication Layer and Information Layer, which provide the necessary level of abstraction for the integration of multiple types of sensing.

Chapter 3

In-network Processing of Spatial Range Queries

3.1 Introduction

In this chapter we focus on energy efficient processing of spatial range queries in sensor networks. Spatial range queries are queries that specify a spatial area[1] the answers must belong to, such as "What was the lowest humidity recorded yesterday morning in the Lake Annete area?" Such a query can be processed by flooding the sensor network with the query, followed by the collection of the query answer from the sensor nodes satisfying the spatial range constraint. As the query regions are typically smaller than the whole monitored area, this is clearly an inefficient solution. Due to the spatial nature of the data collected in the sensor network, we argue that any query processing strategy should treat the spatial range predicate differently than other query predicates in order to reduce the cost of processing the query. Since the energy required by sensing and computation is three to four orders of magnitude less than the energy used for communication [MFHH03, ZG04], we focus on minimizing the energy cost of communication during query processing. We study this problem in the peer-to-peer sensor network environment introduced in Section 1.2.

We propose the SWIP (spatial window processing) framework for processing spatial range queries in peer-to-peer sensor networks. Our framework has two phases. In the first phase, a path from the query originator node to a suitable sensor node located in the query region must be located. Due to its role in the second phase, we call the located node the *query coordinator*. For the second phase, the coordinator node disseminates the query to the sensor nodes located within the query region, collects the query answers from them, and returns the answers to the originator node on the path discovered during the first phase. Within this framework, we propose the SWIF (spatial window flooding) approach formed by two algorithms, one for each phase. We use a greedy routing algorithm in the first phase, while for the second phase we propose an algorithm that uses a constrained flooding to contact the nodes located within the query region. We use an analytical model that captures the behaviour of the proposed strategy for finding the most energy efficient position for the coordinator node. Having analytical models for query processing strategies is important as they can be used to dynamically select the most energy efficient strategy for a given query, allowing a greater flexibility for the query processing. While the SWIP framework (detailed in Section 3.4) accommodates both historical and continuous queries, we focus on

[1]We refer to the spatial area covered by the query as the *query region*.

historical queries in our presentation.

Our contributions in this chapter are the following:

1. We propose the SWIP framework for processing spatial range queries in sensor networks.
2. Within the SWIP framework, we propose the SWIF processing strategy.
3. We construct analytical models to capture the energy costs of the SWIF strategy and the typical network flooding.
4. The cost model of SWIF is developed further to choose the best coordinator location for a given query.
5. We show how to select at query time the best query processing strategy using the analytical models, further reducing the energy cost of query processing.

We evaluate analytically and experimentally the performance of the investigated strategies and show how our cost-model based strategy provides significant reductions in energy costs over the typical network flooding.

The remainder of this chapter is organized as follows. Section 3.2 presents the characteristics of the spatial range query. Section 3.3 presents a typical query processing strategy for sensor networks. Section 3.4 introduces our strategy for processing spatial range queries. In Section 3.5 we construct analytical models for the investigated strategies and show how they can be used to dynamically select the most energy-efficient processing strategy for a given query. The analytical model of SWIF is developed further in Section 3.5.4 to determine the best coordinator location. We discuss several issues relevant to our techniques in Section 3.6. Section 3.7 presents the analytical and experimental evaluation of the investigated strategies. Section 3.8 discusses research works related to our contribution and Section 3.9 concludes the chapter.

3.2 The Spatial Range Query

In this chapter we focus on the energy-efficient processing of spatial range queries (without joins, which are discussed in Section 3.6), which can be expressed as SQL queries with a spatial range selection predicate in the WHERE or HAVING clauses. Along the spatial range selection predicate, we allow selection predicates on other attributes (time and sensed values), incremental aggregation operators (*min, max, sum, average, count*) and grouping. Several works [MFH02, MFHH03, YG03] have investigated energy efficient query processing, but the proposed strategies do not exploit the spatial properties of the sensor data during the query dissemination and answer collection. We detail this issue in Section 3.3 when we present the network flooding strategy, which is typically used in query processing.

Similar to other works on query processing in sensor networks [MFH02, YG02], we adopt a declarative SQL-like language for the user to express queries over the sensor network, where the data collected and stored in the sensor network is represented by one virtual append-only table, denoted R^*. For instance, the following spatial range query can be used to find the average temperature and the lowest humidity collected each day by the sensor nodes located in the area represented by the rectangle with the opposite corners $(< x_1, y_1 >, < x_2, y_2 >)$ during the time range (t_1, t_2) when the temperature was above freezing (0 °C).

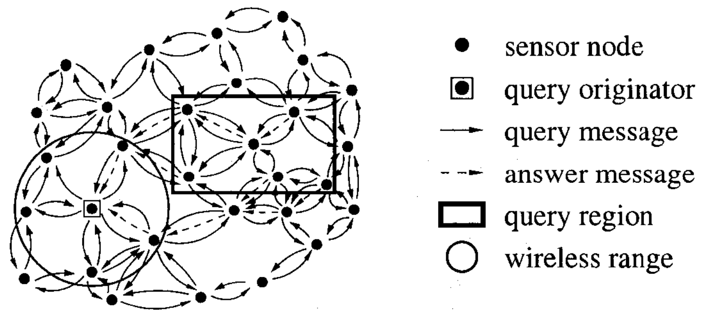


Figure 3.1: The hTAG strategy - message flow

SELECT	DAY(R^*.time), avg(R^*.temperature), min(R^*.humidity)
FROM	R^*
WHERE	R^*.location IN ($< x_1, y_1 >, < x_2, y_2 >$) AND
	R^*.temperature > 0
GROUP BY	DAY(R^*.time)
HAVING	DAY(R^*.time) IN (t_1, t_2)

3.3 Basic Query Processing Strategy

A naive implementation for query processing in sensor networks would simply collect all sensor observations at the user station and answer the queries off-line. While this strategy may be suitable when the users require all the sensors' observations and the queries are issued through the same user station, this is rarely the case in sensor networks. An efficient processing strategy should only collect the query answers at the user station, pushing most of the processing into the sensor network if it reduces the amount of data that has to be transmitted.

A typical strategy used in locating and collecting the query answers is contacting every network node. The query originator node broadcasts the query to its neighbours, which in turn broadcast the query to their neighbours, and so on, until all nodes have received the query. When a node receives a query message for the first time, it chooses the sender as its parent and re-broadcasts the message. Next, the node selects the locally stored data relevant to the query (if any), and, after waiting and receiving its neighbours' answers, it merges them with its own. Finally, it returns the answer to its parent. Once the query originator node has received answers from all its neighbours, it can answer the query. Due to the broadcast of the query message, each node will receive the same query several times. For each query, a node processes only the first message received, discarding subsequent query messages. For queries with a spatial range selection predicate, only the sensor nodes located within the query region will satisfy the query and may have a query answer to return to their parent (subject to the selectivity of other selection predicates in the query).

TAG [MFH02] uses such a processing strategy to disseminate the query and construct the routing tree. As TAG focuses on continuous queries, the routing tree is built once, with the sensors' observations flowing through their parents toward the query originator for the duration of the query. While all sensor nodes receive a query, only the nodes satisfying the spatial range selection predicate (if any) return their answers. TinyDB [MFHH03] uses the same basic strategy to build a semantic routing tree (SRT), which is used to determine which nodes need to participate in the continuous query. For historical queries, only one answer is sent from each node to its parent and, consequently, to the query originator. We consider this typical strategy as the basic processing strategy, and we will refer to it in the following as hTAG (historical TAG). Differently from TAG whose aggregations are only in the spatial domain, hTAG also allows other aggregations to be performed in-network over the historical sensor observations. The flow of the messages used by the hTAG strategy is shown in Figure 3.1.

The hTAG strategy is guaranteed to find the answer to a spatial range query for a connected sensor network, but it may have a substantial overhead for contacting all nodes. The strategy is similar to a parallel breadth first search in a network graph where sensor nodes are vertices and edges represent direct communication links between sensors. Assuming there is no communication latency, the query will reach each node on the shortest path (with respect to the number of hops) from the query originator. As query messages are broadcast along all paths, the first message reaching a node must have travelled over the shortest path. After a query is processed locally, a node returns the answer to its parent, and therefore answers are returned over the shortest path to the query originator.

3.4 Strategies for Query Processing

When there is only one node relevant to the query, the optimal strategy is contacting only the relevant node on the shortest path from the query originator and returning the query answers over the same path. However, when there are several query relevant nodes, communicating with these nodes on the shortest path between the query originator and each of them may not be optimal. Figure 3.2 shows an example. Forwarding the query over the shortest paths (routes (a) and (c)) requires 6 query messages in order to reach both relevant nodes, while route (b) requires only 5 messages. On the other hand, returning the nodes' answers over the shortest paths (routes (a) and (c)) is still optimal if no spatial aggregation on the data is performed. As the energy used in communication is proportional to the message size and the same amount of answers must be transferred over any of the possible return paths, sending the answers over the shortest path is the most energy efficient strategy. This is not true when spatial aggregation is required, in which case the best return paths for the answers depend on the size of the aggregated data. Pushing the aggregation operators as close as possible to the data sources minimizes the amount of data that must be communicated, and thus the energy use. For such queries returning the answers over the shortest path may no longer be optimal. Finding an optimal strategy would require that each query originator node obtains and stores information about the whole network, as well as performs possibly expensive local computation for finding the optimal route for processing each query. Due to the limitations of the sensor nodes, this is not feasible for large sensor networks.

A heuristic solution for query processing is contacting only the query relevant nodes, plus a few extra nodes for routing the query and the answer if the query originator is not located within the query region. A heuristic contacting only a subset of all network nodes should use a lower number of messages than hTAG, which may lead to a lower energy consumption. In addition, if only a subset of the network nodes is involved in processing each query, then several queries could be efficiently processed in different parts of the network without interfering with each other. The SWIP (spatial window processing) framework for

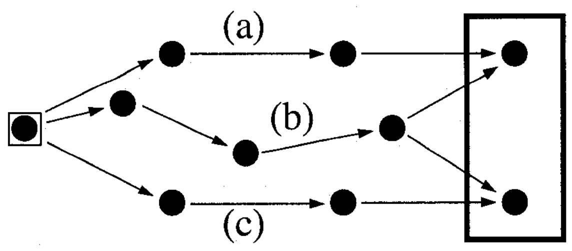


Figure 3.2: Query routing example

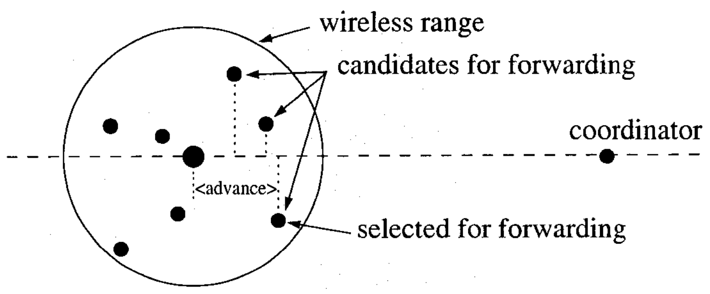


Figure 3.3: Neighbor selection in Greedy

processing spatial range queries allows such a heuristic to be implemented. In this framework, we divide the query processing into two phases, one for locating a path from the query originator node to a sensor node located inside the query region, the other for gathering the answers from the relevant nodes (and possibly performing data aggregation) and returning the answer to the query originator.

The SWIP Framework:

- **Phase 1:** Given a spatial range query at the originator node, find a path to a suitable node located in the query region and send the query over this path. The located node becomes the query coordinator for Phase 2.
- **Phase 2:** The coordinator node initiates the query processing within the query region. All relevant nodes are contacted during processing and the query relevant data is transmitted (possibly aggregated) to the coordinator node. The coordinator will then return the answer to the originator node on the routing path discovered in Phase 1.

These two phases form a general query processing framework, where various algorithms can be used in each phase. In the following we present two algorithms, one for each phase of the SWIP framework. These algorithms together form a complete processing strategy within the SWIP framework, called SWIF (spatial window flooding) in the following.

Algorithm 1: SWIF Phase 1 - Greedy Routing Algorithm

```
    Input    : Current Node $N$, NeighborList $NL$
 1  Receive query $Q$, destination $D$ from neighbour $NB$
 2  $N_{best} \leftarrow N$ /*best node for forwarding*/
 3  $advN_{best} \leftarrow 0$ /*advance to destination for best node*/
 4  foreach node $N_i$ in $NL$ do
 5      if distance($N_i$,$D$)<distance($N$,$D$) &
            advance($N_i$,$N$,$D$)> $advN_{best}$ then
 6          $N_{best} \leftarrow N_i$
 7          $advN_{best} \leftarrow$ advance ($N_i$,$N$,$D$)
 8  if $N_{best} \neq N$ then
        /*forward query to the best neighbor*/
 9      Send query $Q$, destination $D$ to $N_{best}$
10      Wait for query answer $QA$ from $N_{best}$
    else
        /*no good neighbour is found for forwarding the query*/
11      if $N.location$ in $Q.spatialRange$ then
12          Initiate Phase 2 /*N will be the coordinator node*/
13  Return answer $QA$ to $NB$
```

3.4.1 SWIF Phase 1: Greedy Routing

The *Greedy* routing uses a greedy strategy to find a routing path from the query originator node to a virtual[2] coordinator node located in the query region (finding the best location for the coordinator node is studied in Section 3.5.4). The query originator forwards the query to its neighbour with the greatest advance toward the desired coordinator location, where the advance is defined as the length of the projection of the line connecting the originator and its neighbour onto the line connecting the originator and the coordinator location (see Figure 3.3). In turn, the neighbour forwards the query to its neighbour with the greatest advance toward the coordinator, and so on. Each node considers for forwarding only its neighbours which are closer to the coordinator location than itself to ensure the query will get closer to the destination at each step. The query forwarding path will eventually reach a sensor node that is closer to the coordinator location than any of its neighbours, in which case the query is no longer forwarded. If the reached node is located in the query region, the node assumes coordinator role and initiates Phase 2, otherwise an empty answer is returned. Note that the actual coordinator location is generally different from the targeted one since the lack of knowledge about the network topology prevents one to pinpoint an actual sensor node for the coordinator role. The pseudo-code for the *Greedy* algorithm is presented in Algorithm 1.

The flow of the messages used by the *Greedy* algorithm is depicted in Figure 3.4(a). There are many greedy strategies for next hop neighbour selection (e.g., [Fin87, KSU99, TK84]) in routing algorithms, and *Greedy* is closest in spirit to [TK84]. While most such strategies are similar in performance, the presented one allows a more accurate analytical

[2]The originator node only knows its neighbours, and therefore there may be no sensor node located at the targeted location of the coordinator node.

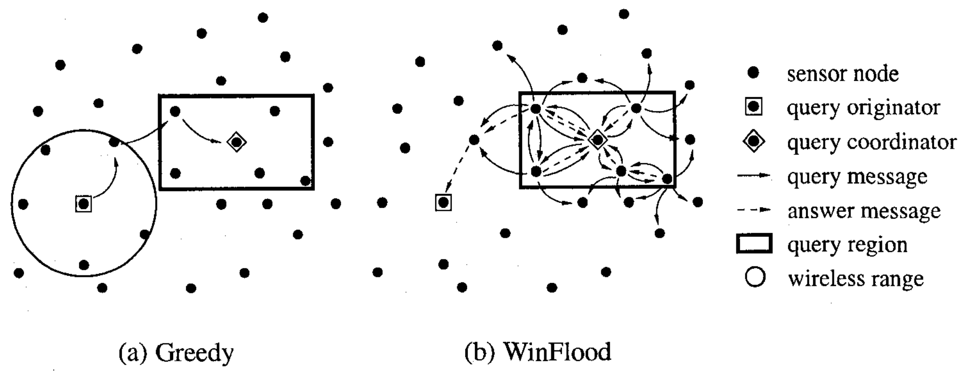


Figure 3.4: The message flow in the SWIF strategy for a given coordinator

modelling, as we shall see in Section 3.5.1. The *Greedy* algorithm uses a small number of messages, but it does not guarantee that a routing path to a node in the spatial range of the query will be found. Greedy-based routing techniques have been shown to nearly guarantee delivery for dense networks [Sto02] as it is typically the case of sensor networks. For low density sensor networks, more robust strategies that guarantee the message delivery for connected networks (e.g., based on GPSR [KK00]) could be used to improve route discovery in this phase. We discuss our choice for *Greedy* over GPSR-like techniques in Section 3.6.

3.4.2 SWIF Phase 2: WinFlood Algorithm

The *WinFlood* algorithm consists of a constrained flooding, where a node broadcasts the query to its neighbours only if its own location is inside the query region. The flooding starts at the coordinator node and stops when the query reaches nodes located outside the query region. Once a node receives the query, it processes the query over its locally stored observations. After waiting and receiving the answers of its neighbours, its merges them with its local answer and processes the query again over the merged answers (e.g., to perform partial aggregations). Finally, it returns its answer to the neighbour it first received the query from. The pseudo-code for the *WinFlood* method is presented in Algorithm 2. Figure 3.4(b) shows the query and answer messages used during query processing. Similar to hTAG, some nodes will receive the same query from more than one neighbour. To prevent answering the same query request more than once, each node keeps a list of the recent queries it has processed, and only returns the query answer to the first neighbour that it has received the query from. To all the other neighbours, an empty (null) answer is returned. This answer policy ensures that only one copy of a node's relevant data is transferred by the network, avoiding communication redundancy.

Due to the distributed nature of the data, the query is decomposed into fragments. This is a common step in distributed query processing [OV99]. In our case the decomposition generates three fragments. The first fragment consists of the spatial range predicate of the query and it is used by nodes to check if their answer is required (Algorithm 2, line 3). The second fragment consists of the query operations that are applied on the local sensor database. Such operations are selection predicates in the WHERE clause and grouping (and selection predicates on groups) on other attributes than location, as well as non-spatial aggregation. For instance, for the example query in Section 3.2, this query fragment is:

Algorithm 2: SWIF Phase 2 - WinFlood Algorithm

```
Input    : Current Node $N$, NeighbourList $NL$, QueryList $QL$
1  Receive query $Q$ from neighbour $NB$
2  QueryAnswer $QA \leftarrow \emptyset$
3  if ($N.location$ in $Q.spatialRange$) and ($Q.id$ not in $QL$) then
4      Add $Q.id$ to $QL$ /*save query ID*/
5      Broadcast query $Q$
6      $QA \leftarrow$ processQueryLocal ($Q$, $N.data$ ) /*process local data*/
7      foreach node $N_i$ in $NL$ do
8          Wait for answer $QA_i$ from node $N_i$
9          $QA \leftarrow QA \cup QA_i$ /*collect answers from neighbours*/
10     $QA \leftarrow$ processQueryCombined($Q$, $QA$) /*process collected answers*/
11 Return answer $QA$ to $NB$
```

```
SELECT     DAY(s.time) AS day, sum(s.temperature) AS sumT,
           count(s.temperature) AS countT, min(s.humidity) AS minH
FROM       LocalSensorData s
WHERE      s.temperature > 0
GROUP BY   DAY(s.time)
HAVING     DAY(s.time) IN (t1, t2)
```

In the *WinFlood* algorithm this query fragment is executed on the local sensor observations while the node waits to receive the answers of its neighbours (Algorithm 2, line 6). The third fragment consists of spatial aggregation and grouping on sensor data from different nodes. This fragment is applied on the concatenation of the local answer (generated by the previous fragment) with the answers received from the neighbours. For the query in Section 3.2, this query fragment is:

```
SELECT     u.day AS day, sum(u.sumT) as sumT,
           sum(u.countT) as countT, min(u.minH) as minH
FROM       UnionOfAnswers u
GROUP BY   u.day
```

In the *WinFlood* algorithm the third query fragment is executed before the answer of the node is returned to the node's parent (Algorithm 2, line 9).

3.5 Analytical Models for Query Processing

Since there are several alternative strategies for processing a query, choosing the most energy-efficient processing strategy for each query becomes a critical decision with direct consequence on the lifetime of the sensor network. Having the proper information, the originator node could decide which strategy processes a query at a lower energy cost. In this section we propose analytical cost models for SWIF and hTAG that can be used by the query originator node to decide which one is the best strategy for processing a query. The models use only the information available locally at a node to estimate the processing cost.

Table 3.1: Notations used in the definitions of the analytical models

Parameters of query and sensor network	Description
R_A	Area of the monitored region
N	Number of sensor nodes
W	Wireless communication range
Q_A	Area of the query's spatial range
Q_s (in bits)	Size of the query message
$\emptyset_s$ (in bits)	Size of the empty answer message
E_u (per bit)	Energy used for unicast
E_b (per bit)	Energy used for broadcast
E_r (per bit)	Energy used for receive
Other notations	**Description**
$N^i(x_i, y_i)$	Node i located at (x_i, y_i)
$O(o_x, o_y)$	Query originator node
$C(c_x, c_y)$	Coordinator node (for SWIF)
N_r	Number of relevant nodes
N_n	Average number of neighbours per node
W_A	Area covered by a node's wireless range
R_i	Local observations stored in relevant node
A_i	A relevant node's answer based on query and R_i
d_{OC}	Distance between O and C
h_{OC}	Number of hops from O to C
h_{N^iC}	Average number of hops from N^i to C
h_{N^iO}	Average number of hops from N^i to O
a_{hop}	Average advance for one hop

In the following two sections we focus on the construction of the analytical models, while in Section 3.5.3 we discuss the dynamic selection of the best strategy.

Before proceeding to the analytical models, we introduce several notations and estimate some basic values used in the models. Three important values for the models are the number of sensor nodes, denoted by N, the number of nodes relevant to the query, denoted by N_r, and the average number of neighbours each node has, denoted by N_n. In our models we assume nodes are uniformly distributed over the monitored area. The area[3] covered by the wireless communication range W of a node is $W_A = \pi W^2$. Each point in the monitored area is covered on average by the wireless ranges of $N\frac{W_A}{R_A}$ nodes, where R_A is the area of the monitored region. Each sensor node is covered by the wireless ranges of its neighbours, therefore the average number of neighbours that each node has is $N_n = N\frac{W_A}{R_A} - 1$. Due to the uniform node distribution, the number of relevant nodes is proportional to the area covered by the query region from the monitored region and it can be estimated as $N_r = N\frac{Q_A}{R_A}$, where Q_A is the area covered by the spatial range of the query. If the sensor nodes do not know how many sensor nodes (N) are used for monitoring, each node N^i can use the number of its neighbours N_n^i (acquired as part of network activation) to estimate N as $N_{est} = (N_n^i + 1)\frac{R_A}{W_A}$. Table 3.1 summarizes the notations used in the analytical models.

[3]As discussed in 1.2, we consider that all messages are transmitted as far as the wireless communication range.

3.5.1 Estimating the Cost of SWIF

The energy cost of SWIF is determined by several costs: for sending the query from the originator node O to coordinator C, for distributing the query from C to the relevant nodes, for collecting the query answers from the relevant nodes to C, and finally for sending the answers to the originator node. The analytical model of SWIF is independent of an exact location for the coordinator C, with the constraint that $C(c_x, c_y)$ is located within the query region (in conformity with the SWIP framework). We show in Section 3.5.4 how to determine the best coordinator location for a given query. We estimate the cost of SWIF separately for its two phases:

$$E_{SWIF} = E_{Greedy} + E_{WinFlood}$$

Estimating the Cost of Greedy

The *Greedy* algorithm is responsible for finding a path between the query originator O and a coordinator node C, and sending the query over this path during its discovery. For each hop of this path, a node transmits the query, while another node receives it. Therefore, the energy used by *Greedy* is equal to

$$E_{Greedy} = (E_u + E_r)Q_s h_{OC}$$

where E_u and E_r represent the energy used to transmit and, respectively, receive one bit of information, Q_s is the size of a query message and h_{OC} represents the number of hops between O and C. Assuming a dense sensor network, one can approximate h_{OC} by the Euclidean distance d_{OC} between O and C divided by the average advance a_{hop} toward C for a hop (i.e., the advance for a neighbour selected for forwarding):

$$h_{OC} = \frac{d_{OC}}{a_{hop}}.$$

The query originator node O knows both its location and the targeted destination location of the coordinator C: $d_{OC} = \sqrt{(o_x - c_x)^2 + (o_y - c_y)^2}$. Let us denote with a_A the area of the the circular segment within the wireless range circle where the neighbour with the greatest advance must be located (see Figures 3.3 and 3.5). The size of the network area corresponding to each node is equal to $\frac{R_A}{N}$. Since a_A is the smallest circular segment such that there is exactly one node inside (assuming uniform node distribution), we have that $a_A = \frac{R_A}{N}$. The area of the circular segment a_A is also equal to $W^2 \arccos(\frac{W-h}{W}) - (W - h)\sqrt{2Wh - h^2}$, where h is the height of the arced portion. From the equality of the two expressions of a_A we can find h since the other terms are known, and thus the coordinates for the circular segment. The selected neighbour could be located anywhere within a_A. The average advance a_{hop} of the neighbour selected for forwarding toward C is the sum of the advances for all possible locations for this neighbour divided by the number of these locations (see Appendix A for details):

$$a_{hop} = \frac{\int\int_{a_A} y \, dxdy}{\int\int_{a_A} dxdy} = \frac{\frac{2}{3}(2Wh - h^2)^{\frac{3}{2}}}{a_A}.$$

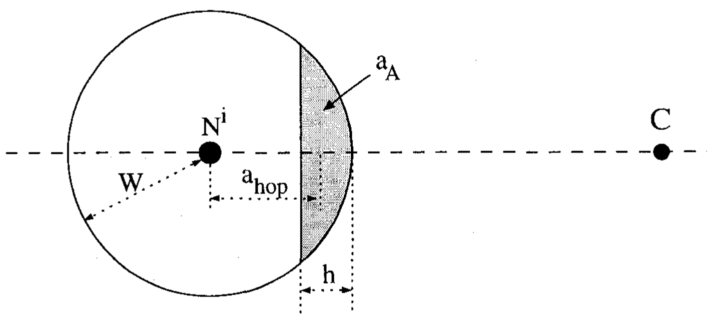


Figure 3.5: The average advance a_{hop} for a hop

Estimating the Cost of WinFlood

The energy cost of *WinFlood* has three components: for forwarding the query to the relevant nodes (E_Q), for these nodes to return their answer to the coordinator node (E_C), and for the coordinator node to return the query answer to the originator node O (E_O):

$$E_{WinFlood} = E_Q + E_C + E_O.$$

During query forwarding, each relevant node will broadcast the query once, and receive the query from all its neighbours[4]. The energy used for forwarding the query consists of the energy to broadcast the query plus the energy to receive the broadcast messages:

$$E_Q = E_b Q_s N_r + E_r Q_s N_r N_n.$$

Even though E_Q grows quadratically in N, for small query regions the slope of the increase is small, since the fractions in N_r and N_n are small.

There are two situations to consider when calculating the cost of returning the answers. The first situation is for queries without spatial aggregation. In this case, the query answers generated by each relevant node over its locally stored data are concatenated with the answers received from the other nodes and returned toward the coordinator node. Thus, the local answer A_L generated by a node is returned over the shortest path (in number of hops) to the query coordinator. The average distance between the coordinator C located in the query region and a relevant node N^i can be computed as the sum of the distances from C to all possible locations for the relevant nodes divided by the number of these locations:

$$d_{N^iC} = \frac{\int\int_{Q_A} \sqrt{(x-c_x)^2 + (y-c_y)^2} dxdy}{\int\int_{Q_A} dxdy}.$$

The solution for the analytical formula of d_{N^iC} is derived in Appendix B. The average advance toward C over a hop for any node N^i is a_{hop} (as calculated for the *Greedy* algorithm[5]). Therefore, the average number of hops that the answer A_L travels between a relevant node N^i and C is $h_{N^iC} = \frac{d_{N^iC}}{a_{hop}}$. As the coordinator is also a relevant node, only $N_r - 1$ nodes send their answer A_i to the coordinator node. Under a relational storage

[4]We do not consider the boundary effects at the query region.

[5]As messages reach the destination over the shortest path when fboding, the neighbour with the greatest advance must be used over each hop. This is similar to the neighbour selection in *Greedy*.

model, the average size of the local answer A_i of a relevant node N^i to a given query Q can be estimated as:

$$\texttt{size}(A_i) = \texttt{agg}(R_i)\ \texttt{sel}(R_i)\ \texttt{card}(R_i)\ \texttt{length}(R_i),$$

where $\texttt{agg}(R_i)$ is the query's aggregation factor over relation R_i (R_i is the local relation storing the observations of a relevant node and it is a horizontal fragment of the global virtual relation R^* over sensor network's data as discussed in Section 1.1.3), $\texttt{sel}(R_i)$ is the query's selectivity factor over relation R_i, $\texttt{card}(R_i)$ is the cardinality of R_i and $\texttt{length}(R_i)$ is the length in bits of a tuple in R_i. We discuss in Section 3.6 how one can determine the selectivity and aggregation factors. In total, the energy used for gathering the answers at the query coordinator node C for a query with no spatial aggregation (*nsa*) is:

$$E_C^{nsa} = (E_u + E_r)(N_r - 1)h_{N^iC}\texttt{size}(A_i).$$

The second situation is for queries with spatial aggregation. In this case, the local answer A_i of each node is aggregated into one answer with the other answers the node receives. In addition, each answer generated by a node is only transmitted one hop (that is, to a node's parent in the routing tree), where it is aggregated again with other answers. Thus, the energy used for gathering the answers at the query coordinator node C for a query with spatial aggregation (*sa*) is:

$$E_C^{sa} = (E_u + E_r)(N_r - 1)\texttt{size}(A_i).$$

For returning the query answer from the coordinator node to the originator, the size of the answer is also estimated differently for the two cases. If the query has no spatial aggregation, the size of the query answer can be estimated as $N_r\texttt{size}(A_i)$, which leads to the following cost:

$$E_O^{nsa} = (E_u + E_r)h_{OC}N_r\texttt{size}(A_i),$$

while for queries with spatial aggregation the coordinator node aggregates all the answers of the relevant nodes (including its own), and we have:

$$E_O^{sa} = (E_u + E_r)h_{OC}\texttt{size}(A_i).$$

Overall, the cost of *WinFlood* for queries without spatial aggregation is:

$$\begin{aligned} E_{\textit{WinFlood}}^{nsa} &= E_Q + E_C^{nsa} + E_O^{nsa} \\ &= E_bQ_sN_r + E_rQ_sN_rN_n + (E_u + E_r)(N_r - 1)h_{N^iC}\texttt{size}(A_i) + \\ &\quad (E_u + E_r)h_{OC}N_r\texttt{size}(A_i) \end{aligned}$$

and for queries with spatial aggregation we have:

$$\begin{aligned} E_{\textit{WinFlood}}^{sa} &= E_Q + E_C^{sa} + E_O^{sa} \\ &= E_bQ_sN_r + E_rQ_sN_rN_n + (E_u + E_r)(N_r - 1)\texttt{size}(A_i) + \\ &\quad (E_u + E_r)h_{OC}\texttt{size}(A_i). \end{aligned}$$

The cost of returning the query answers increases linearly in N, Q_A (due to N_r) and $\texttt{size}(A_i)$. E_Q is quadratic in N, but the fractions in N_r and N_n are typically small. Thus, the size of A_i determines which of the three costs has a larger weight in the total cost of *WinFlood*. When $Q_s \ll \texttt{size}(A_i)$ the cost of *WinFlood* is dominated by the cost to return the query answers.

3.5.2 Estimating the Cost of hTAG

To estimate the cost of the basic query processing strategy hTAG, we split the energy use into three components: for forwarding the query to all nodes (E_Q), for returning the empty answers which signal that the query has already been processed or no answers are available ($E_\emptyset$), and for returning the query answers from the relevant nodes to the query originator O (E_O):

$$E_{hTAG} = E_Q + E_\emptyset + E_O.$$

hTAG uses network flooding for query forwarding. Each node will broadcast the query once, and receive the query from all its neighbours. Thus, the cost of disseminating the query is:

$$E_Q = E_b Q_s N + E_r Q_s N N_n.$$

Once the query is received, all nodes except the relevant nodes will return an empty answer to all their neighbours, while the relevant nodes will return the query answer to one of their neighbours and the empty answer to every other neighbour:

$$E_\emptyset = (E_u + E_r)\emptyset_s(NN_n - N_r).$$

For returning the answers to the query originator node, the shortest path (in number of hops) between each relevant node and the originator O is used. The average number of hops between O and a relevant node N^i can be estimated by the average distance d_{N^iO} from a relevant node N^i to the originator O divided by the average advance toward originator over a hop a_{hop}: $h_{N^iO} = \frac{d_{N^iO}}{a_{hop}}$ (the average advance over a hop was calculated as part of the cost estimation for the *Greedy* routing). The average distance from a relevant node to the originator node is equal to the sum of the distances from O to all possible locations for the relevant nodes divided by the number of these locations:

$$d_{N^iO} = \frac{\int\int_{Q_A} \sqrt{(x-o_x)^2 + (y-o_y)^2}\,dx\,dy}{\int\int_{Q_A} dx\,dy}.$$

The solution for the analytical formula for d_{N^iO} can be derived using similar steps as for d_{N^iC} (see Appendix B). Since we consider peer-to-peer sensor networks where knowledge of the network topology is not available, we cannot estimate if any partial spatial aggregation can be performed on the paths over which the answers are collected. Differently from the *WinFlood* algorithm where each node participating in the return of the answers to the coordinator node was itself a relevant node holding answers, in hTAG a node's answers may be returned to the originator node over a path that is only used by that node. We consider that no spatial aggregation can be performed in the network over the answer return paths. Thus, our cost estimation is correct for queries without spatial aggregation and it is only an upper bound for the cost of queries with spatial aggregation. Since each node on the path between the query originator and a relevant node will receive and transmit the answer

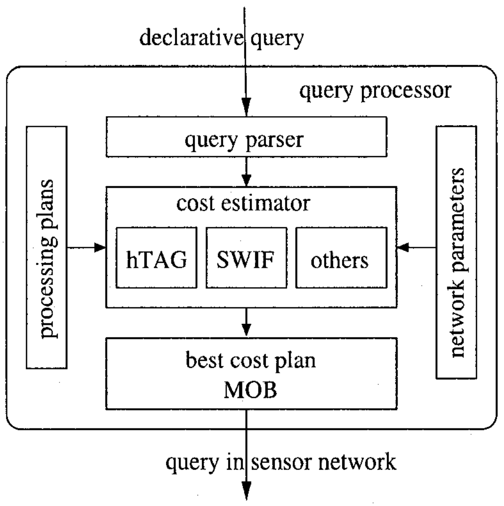


Figure 3.6: Dynamic selection of the processing strategy

of that node, the energy used for returning the answers is:

$$E_O = (E_u + E_r) h_{N^i O} N_r \mathtt{size}(A_L).$$

Both E_q and $E_\emptyset$ costs depend quadratically on N, and $E_\emptyset$ is also affected by the size of the query area Q_A (but not affected by $\mathtt{size}(A_L)$). Thus, for denser networks a large increase in these costs is expected. The E_O cost is linear in all three variables. Different from *WinFlood*, both A_L and Q_A affect the weights of the three costs of hTAG. When query area is small, the cost of hTAG is dominated by the cost of query forwarding, while for queries over large areas with no aggregation the cost of returning the answers prevails.

3.5.3 Dynamic Selection of the Query Processing strategy

In the previous sections we have developed analytical models to estimate the cost of query processing for the SWIF and hTAG processing strategies. We expect that other strategies will be proposed in the future for processing spatial range queries due to the importance of this query type and the increased interest from both academia and industry in the sensor network technology [Ric05]. As for most problems, there is no strategy that is a clear winner for all situations and application scenarios. Thus, it is important to find automatic ways to select the best strategy for a particular situation.

In the case of query processing in sensor networks, we propose to use the analytical models to estimate the cost of each available query processing strategy, and, based on the cost estimation, to select the most promising one to be used for processing each query. Assuming the cost models are accurate, this can lead to an overall lower energy consumption and extended lifetime of the network. In our case we want to find the best strategy for a given sensor network and a given query. Figure 3.6 shows the execution blocks of a query processor that dynamically selects the query processing strategy to be executed in the sensor network. We have denoted with MOB (model-based) the most energy efficient processing strategy as suggested by the analytical models.

There are several issues that are important when using the analytical models in the query

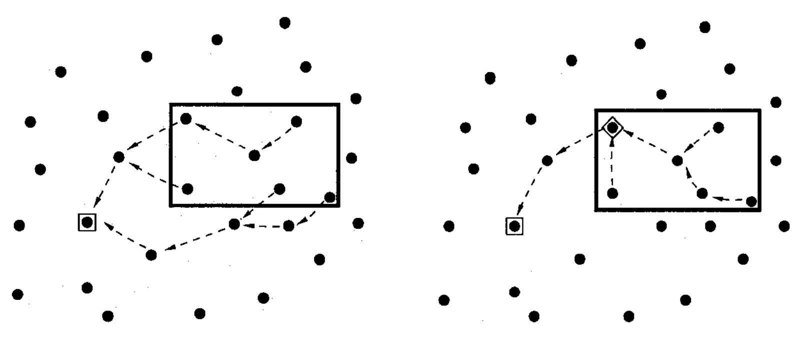

(a) hTAG (b) SWIF

Figure 3.7: Possible paths for returning the query answers

processor shown in Figure 3.6:

- *Existence of the analytical model.* In order to include a processing strategy in the dynamic decision of a query processor, that strategy must have an analytical model for estimating its cost. We have proposed analytical models for the SWIF and hTAG strategies and demonstrated how to estimate certain parts of a processing strategy. Our cost estimations may help in developing the analytical model for a new strategy as we expect certain aspects to be common among strategies.
- *Accuracy of the analytical model.* The dynamic selection of a query processing strategy is only as good as the analytical models used for the decision. When one (or more) of the analytical models differ largely from the actual query processing cost, a possibly expensive processing strategy may be selected. We will show in Section 3.7 that our models are accurate.
- *Complexity of the analytical model.* While the analytical models are computed only once, before the query is processed in the sensor network, the computational complexity of some models may be beyond the processing capabilities of sensor nodes. In such cases the models cannot be computed in the query originator node, but their calculation must be done at the mobile user station, which is typically a more resourceful machine. While the dependence of a sensor network on external, more powerful, machines is undesirable in our opinion, it may be worthwhile in some applications for the benefit of selecting the most energy efficient processing strategy for a query.

3.5.4 Finding the Best Coordinator

The SWIP framework is flexible with respect to the position of the coordinator node, requiring the coordinator only to be located within the query region without specifying a particular location. In this section we show how the best location for the coordinator node can be found, given the location of the query originator node and the parameters of the query and the sensor network.

Assuming no communication latency, hTAG forwards the query in parallel over all possible paths to nodes, and therefore each node receives the query first over the shortest path from the originator node. As the same path is also used for returning a node's answer, the query answer is returned to the originator node over the shortest path in hTAG (see Figure 3.7(a)). When considering only the cost of returning the answers to queries without spatial aggregation, hTAG must have the lowest cost for returning the answers. For queries with spatial data aggregation, hTAG performs partial[6] spatial aggregation in-network (when the paths over which the answers are returned intersect), while full spatial aggregation is guaranteed only at the originator node once all the answers of the relevant nodes have been returned. On the other hand, if we force all return paths to intersect at a coordinator node (see Figure 3.7(b)), this node can perform full spatial aggregation of the answer. The result of the aggregation would be a potentially smaller answer to be sent to the originator node, thus saving on the cost of returning the query answer. In addition, we are interested not only in the cost of collecting the query answers, but also in the cost of disseminating the query.

Finding the optimal location for the coordinator node is not trivial since several separate costs are involved. Minimizing these costs individually will indicate different good locations for the coordinator node. We take advantage of the analytical model of SWIF introduced in Section 3.5.1 to find the best coordinator location. The cost of SWIF is $E_{SWIF} = E_{Greedy} + E_{WinFlood}$. Given a sensor network, a query and an originator node, the cost of E_{SWIF} depends only on the position of the coordinator $C(c_x, c_y)$. Minimizing this cost function allows us to obtain the position of the coordinator C for which the cost of SWIF is minimal.

After eliminating the constant terms (with respect to the position of the coordinator node), we obtain the following objective functions:

$$\mathtt{min}[E^{nsa}_{SWIF}] = \mathtt{min}[(Q_s + N_r\mathtt{size}(A_L))d_{OC} + (N_r - 1)\mathtt{size}(A_L)d_{N^iC}]$$

and

$$\mathtt{min}[E^{sa}_{SWIF}] = \mathtt{min}[d_{OC}].$$

For spatial aggregation, the minimum of the function E^{sa}_{SWIF} depends only on the position of the originator node and the location of the query region (as C must be located within the region). This is expected since the answer of each relevant node is transmitted only to the node's parent in the routing tree. Therefore, finding the best coordinator location to minimize the cost of E^{sa}_{SWIF} is greatly simplified: the coordinator position that minimizes the cost of SWIF for spatial aggregation is the point of the query region closest to the originator node.

For queries without spatial aggregation, the best coordinator location is the location of C that minimizes the cost of equation $\mathtt{min}[E^{nsa}_{SWIF}]$. The best coordinator location depends on the weights of the terms containing d_{OC} and d_{N^iC} as the other factors are independent of C. If we consider only the term containing d_{OC}, the best location is the point of the query region closest to the originator node. If we consider only the term containing d_{N_iC}, the best location is the centre of the query region. In our sensor network simulator we use an iterative solution for finding the location for the coordinator node the minimizes the function E^{nsa}_{SWIF}. Finding the minimum of the cost function E^{nsa}_{SWIF} may be beyond the computational capabilities of some sensor platforms. Subject to the type of platform used for the sensor nodes, we see two possible situations: for platforms with advanced CPUs (e.g., Sensoria WINS platform [Sen]), the best coordinator location can be found at

[6]The cost savings due to partial spatial aggregation cannot be modelled accurately without knowledge of the sensor network layout, which is not available in peer-to-peer sensor networks.

the originator node; for platforms with reduced CPU capabilities (e.g., Crossbow MICA motes [Cro]), the mobile user station can help find the best coordinator location. In the latter case, the mobile user station should either be familiar in advance with the analytical model of SWIF, or accept computation tasks from the originator node.

3.6 Discussion and Extensions

We have focused so far on the core of the adaptive techniques that we have proposed for processing spatial range queries. In the following we discuss several issues related to our techniques, as well as some open problems.

Joins. The SWIF strategy does not process queries with joins. We see two possible cases for processing joins using SWIF. If the join of the R^* relation is with relations stored at the mobile user station, such joins could be processed at the station once the relevant sensor data is extracted from the network using SWIF. Processing self-joins of the R^* relation is a far more challenging problem. For instance, consider joining the sensor data collected in two non-overlapping regions. While a trivial strategy is to collect the relevant data from both regions using SWIF and perform the join operation at the user station, it may be cheaper to send the data from one region to the other, perform the join on-the-fly, and return the join result only to the user. Finding a good plan is challenging due to the lack of global network knowledge at sensor nodes in peer-to-peer sensor networks. Chapter 4 investigates the processing of join queries in sensor networks.

Greedy vs. GPSR. The *Greedy* algorithm uses a small number of messages, but it does not guarantee that a routing path to a node in the query's spatial region will be found. While properties such as guaranteed delivery are desirable, our preliminary studies have shown that the simple *Greedy* algorithm successfully reaches the query's spatial region for more than 98% of the queries in the sensor networks considered in our experiments. In addition, routing strategies with guaranteed delivery such as GPSR [KK00] may require knowledge of the network layout for accurate cost estimation, while one can find a good cost estimate for the *Greedy* algorithm (see Section 3.5.1). For instance, the cost of the GPSR algorithm could be estimated as the cost of its greedy forwarding[7], which GPSR uses as its main routing algorithm, plus the cost of the perimeter forwarding that GPSR uses to recover when the greedy forwarding fails. The number of nodes contacted during the perimeter forwarding could vary anywhere from 0, if the greedy forwarding does not fail, to the number of sensor nodes, if all nodes are on one face of the network graph. Thus one cannot find a good estimate of the GPSR cost without knowledge of the network layout, which is unavailable in our network scenario.

The selectivity and aggregation factors of a query. The query originator node must estimate the selectivity and aggregation factors of a query in order to use the analytical models developed in Section 3.5. In the case of traditional distributed databases, statistics on databases play an important role in estimating the cost of query operators [OV99]. Due to the nature of the peer-to-peer sensor network environment, nodes cannot obtain statistics about other nodes, but only general statistics about the data stored in the sensor network. Such statistics can be obtained using the locally stored data and the answers to queries that a node has seen in the past. When no statistics about certain attributes are available, nodes can use predetermined values for the reduction factors. For instance, the System R query

[7]The greedy routing in GPSR uses a different neighbour selection rule than our *Greedy* routing algorithm.

Table 3.2: Parameters of query and sensor network

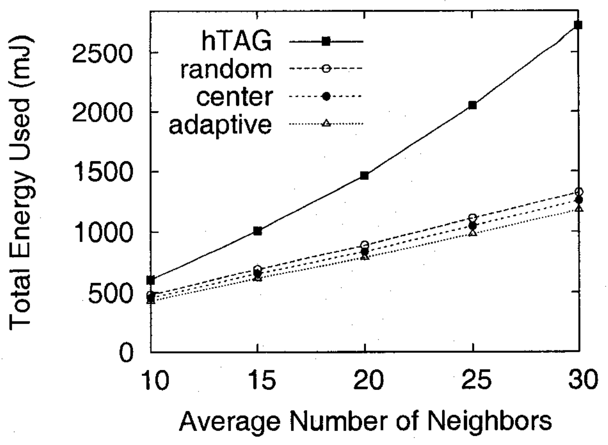
Parameter	**Default Value**
Average number of neighbours (N_n)	15 ($N = 2050$)
Tuple size <*value, time-stamp*>	8 bytes
Query message size (Q_s)	32 bytes
Size of query's spatial region (Q_A)	5% (of R_A)
Size of query's answer for a node (A_L)	60 tuples

optimizer for traditional relational database systems uses a reduction factor of 0.1 for *column=value* selection predicate when no index or statistics on *column* is available [RG00].

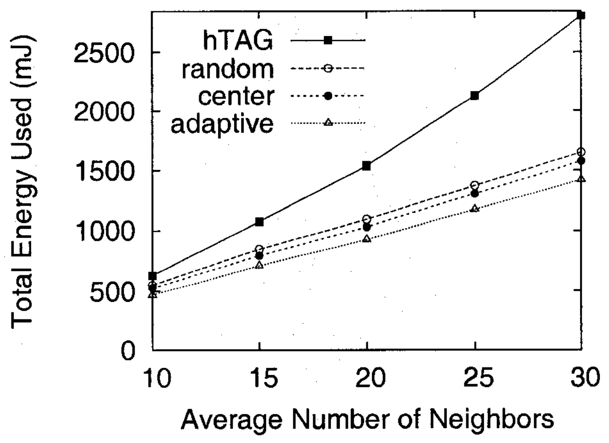
Sensor node failures. In sensor network deployments the nature of the environment and energy constraints can cause transient or permanent failures of the sensor nodes. Dealing with sensor failures is a complex problem which has not yet been thoroughly investigated in the related works. Most systems employ solutions suitable only for specific application settings. In this context let us consider the sensor node failure in the environmental application scenario introduced in Section 1.3, where the sensor network is deployed for long-term monitoring with only occasional queries introduced in the network. If one assumes that most sensor failures would occur when the network is idle, the problem is less severe as the network will adapt itself during the periodical neighbour look-up performed for maintenance. Otherwise, i.e., when a node participating in processing a query fails, a simple mechanism such as heartbeat notification messages [SBLC04] can be used to recognize the failure at the node's neighbours in the query forwarding tree. The parent node in the forwarding tree re-initiates the query processing from that point forward, while the child nodes drop the processing of the initial query. In addition, answer caching at intermediate nodes can be used to reduce the cost of re-processing the query. On the one hand, if the probability of node failure during query processing is very low, partially re-processing a query has a small impact on the lifetime of the sensor network. On the other, re-processing a query is not an efficient solution in environments where the failure rate and query load are high. Nonetheless, how to efficiently and effectively handle sensor failures is complementary to our processing strategies presented in this chapter.

3.7 Evaluation

We implemented a sensor network simulator to study the performance of the presented strategies and evaluate our cost models. As discussed in Section 1.2, in our evaluation we focus on the energy cost of communication only. The sensor nodes' placement follows a uniform distribution over a two dimensional region. We express the size of the query region as a percentage of the size of the monitored region. For the selectivity of other query operators, we use 60 tuples as the number of tuples that are selected from a node's database (A_L) for the query answer. For instance, if we consider temporal selection only, this number would correspond to an 1 hour temporal range if sensors take 1 observation every minute, or an 1 month temporal range for 1 observation taken every 12 hours. The query and sensor network parameters and their default values used in our evaluations are summarized in Tables 1.2 and 3.2. Similar to other works (e.g., [RKY$^+$02]), in our sensor network simulator the message delivery is instantaneous and error-free between nodes communicating directly.



(a) analytical models



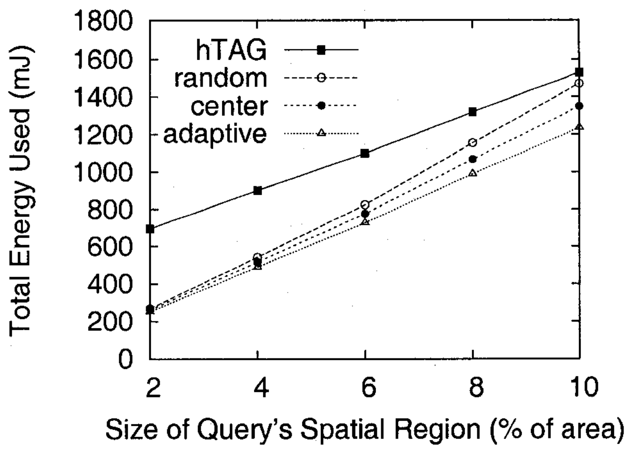
(b) experimental simulations

Figure 3.8: The impact of network density for queries without aggregation

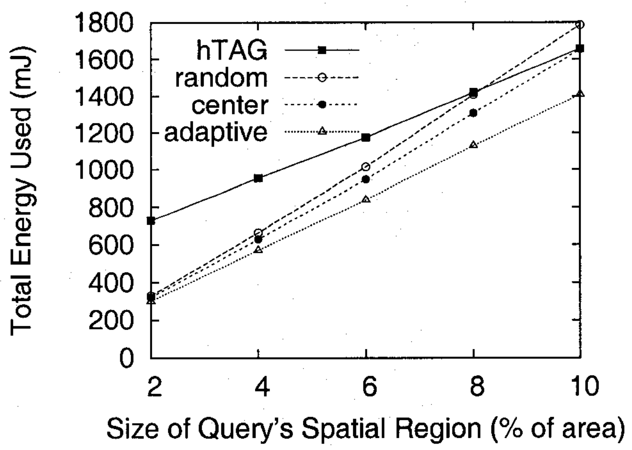
We evaluate the performance of the *adaptive* SWIF strategy against two other strategies based on SWIF: a baseline strategy (*random*), where the coordinator location is selected at random within the query region; and a preliminary strategy that we have proposed in [CNS04] (*center*), where the coordinator location is at the centre of the query region. We also compare the SWIF strategies against the hTAG, the typical strategy for query processing used in sensor networks. We denote by MOB the method that dynamically selects the most efficient of the hTAG and the adaptive SWIF strategies for a given query and originator location as determined by the analytical models. MOB is only shown in a graph when its cost differs from the most efficient strategy in the graph.

3.7.1 Impact of Network Density

We first investigate the performance of the strategies when the query does not require aggregations. Figure 3.8(a) shows the effect of node density on the strategies using the analytical models. hTAG is affected the most by the increase in the node density due to the overhead



(a) analytical models



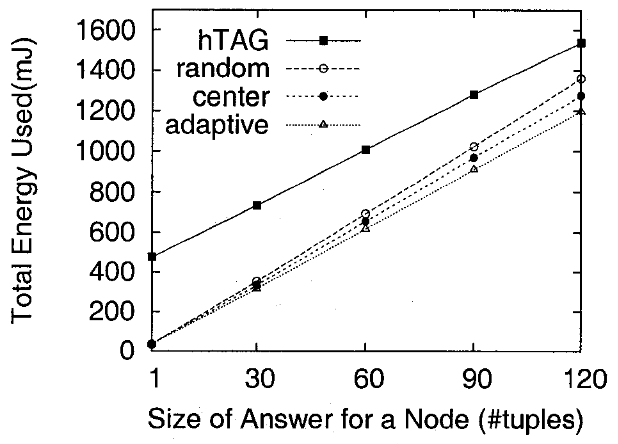
(b) experimental simulations

Figure 3.9: The impact of size of query region for queries without aggregation

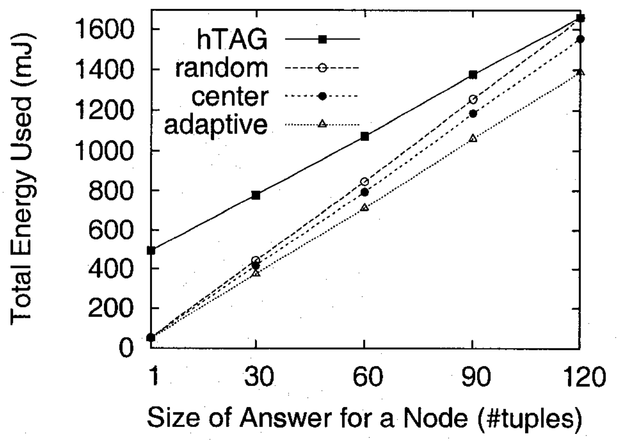
for sending the query to a larger number of non-relevant nodes and the increased query message redundancy as each network node will receive the query from more neighbours. All strategies show slightly higher processing costs in the experimental simulations (Figure 3.8(b)). This is due to an underestimation of the number of hops between two nodes in the cost models. Another effect in the experimental simulations is that there are no nodes located exactly at the targeted coordinator locations, thus all SWIF strategies use coordinator nodes located at nearby locations, leading to higher costs than predicted. By adaptively selecting the best coordinator location for each query, the *adaptive* SWIF reduces the query processing cost by up to 50% compared to hTAG, and up to 15% and 10% compared to the *random* and *center* SWIF.

3.7.2 Impact of Size of Query Region

Figure 3.9 shows the behaviour of the strategies for small to medium spatial-range queries. For small query region, all SWIF strategies substantially reduce the cost of processing



(a) analytical models



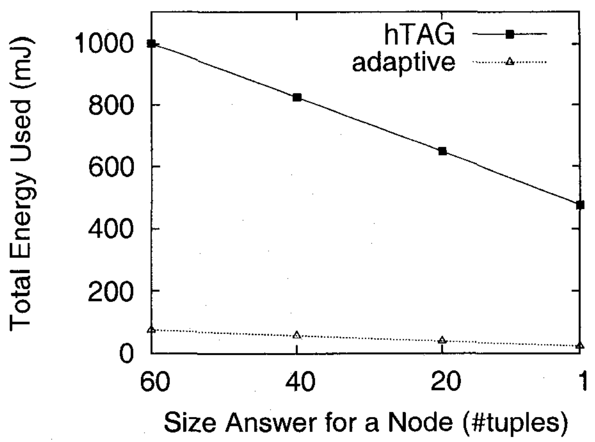
(b) experimental simulations

Figure 3.10: The impact of query selectivity for queries without aggregation

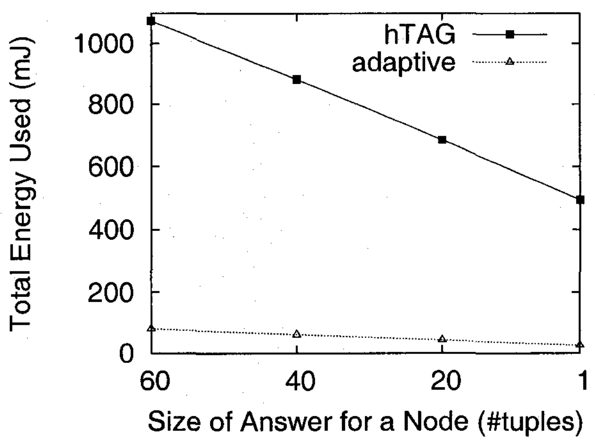
compared to hTAG, the *adaptive* SWIF reducing the cost of hTAG by 80% for a 1% region (not shown). For medium query regions, the performance of the *random* and *center* SWIF approaches that of hTAG. As the size of the query region increases, the size of the answers that must be returned to the originator node increases. This effect favours hTAG as the answers are returned over the shortest path, while the overhead payed by the SWIF strategies for returning answers through the coordinator node increases their cost. The *adaptive* SWIF selects a coordinator node located closer to the originator node compared to the coordinators used by the *random* and *center* SWIF and thus its difference in the path length from the shortest path is smaller than for the *center* and *random* SWIF.

3.7.3 Impact of Query Selectivity

Figure 3.10 shows the behaviour of the strategies when the size of a node's answer based on its local observations (A_L) is varied. As the size of the answer increases, the cost of returning the answers starts dominating the total cost of hTAG, the fixed cost payed for



(a) analytical models



(b) experimental simulations

Figure 3.11: The impact of data aggregation

query forwarding reducing its weight on the total cost. In this case hTAG benefits from returning the answer over the shortest path. When the user is interested in retrieving the observations taken at a time point (i.e., a single tuple for a node), the SWIF strategies reduce the cost of processing by 93% compared to hTAG.

3.7.4 Impact of Data Aggregation

Figure 3.11 shows the effect of the data aggregation on the *adaptive* SWIF and hTAG. We consider aggregations where observations taken at the same time (or aggregations over the same time range) are aggregated into one measurement over all relevant sensor nodes (full spatial aggregation). For the aggregation on the local sensor data, we vary the aggregation factor between no aggregation (60 tuples) and full aggregation (1 tuple), where all observations taken by the same sensor node are aggregated into one. As the main component of the query processing cost for the *adaptive* SWIF is the cost of returning the answers, aggregation drastically reduces its cost. The *random* and *center* SWIF have a much higher cost than the *adaptive* SWIF and are not shown in the graphs. Compared to the cost of processing queries with no aggregations, the *adaptive* SWIF reduces its cost by 90% for spatial-only aggregation, followed by 70% cost reduction in cost for full aggregation. The cost of processing spatial range queries is lowered by 92.5% to 95% for the *adaptive* SWIF compared to hTAG. hTAG is not able to benefit as strongly as the SWIF strategies from the query aggregation as the full spatial aggregation may only be performed at the originator node, while for SWIF the full aggregation is performed at the coordinator node. Also, the decreasing costs for returning the answers favours more the SWIF strategies than hTAG due to the higher cost payed by hTAG to disseminate the query.

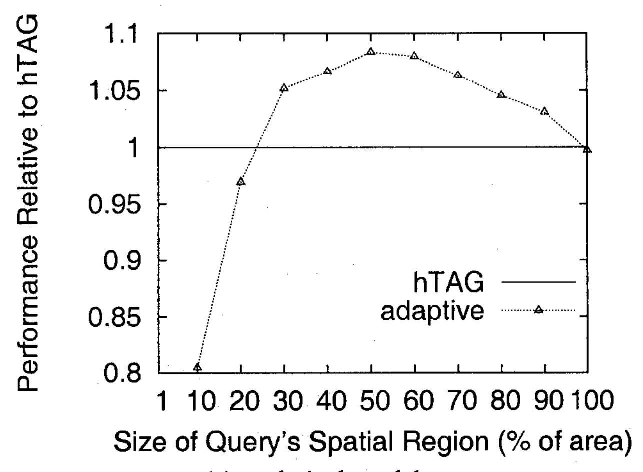
The analytical models capture well the behaviour of the strategies. The models underestimate the actual processing costs for all strategies since the network topology is unknown. For the SWIF strategies, the under-estimation is also due to the cost difference between the desired coordinator location and the location of the actual coordinator node. Due to the adaptive selection of the best coordinator position for each query, the *adaptive* SWIF performs better than the *random* and *center* SWIF for all situations. The performance of the *adaptive* SWIF is also better than that of hTAG for all scenarios investigated so far. Thus, the cost of MOB, the method that selects the best processing strategy based on the analytical models, is identical to the cost of the *adaptive* SWIF and it was not shown in the graphs.

3.7.5 Impact of Large Query Region

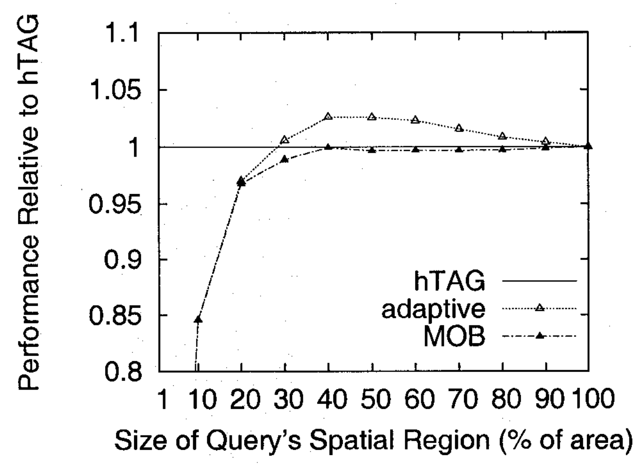
We have shown so far that the *adaptive* SWIF substantially reduces the cost of processing small to medium spatial range queries compared to hTAG. Nevertheless, some sensor network applications may require queries of all sizes. Figure 3.12 shows the performance of the *adaptive* SWIF relative to hTAG for spatial range queries covering from 1% to 100% of the monitored region. For queries larger than 30%, hTAG performs better than the *adaptive* SWIF, but the cost difference is small (up to 2.5% in the experimental evaluation). For large queries, the cost of returning the query answers dominates the total processing cost for both strategies. This works to the advantage of hTAG as the effect of its overhead for contacting every network node is compensated by the short return paths. When considering the cost of returning the answers only, the difference in cost between returning the answers over the shortest path as in hTAG and the cost of returning them through a coordinator node as in SWIF increases with the answer size. When the spatial region reaches 100%, both strategies use the same amount of energy since the *adaptive* SWIF degenerates to hTAG. For query sizes where the most efficient of the hTAG and *adaptive* SWIF depends on the location of the originator and query region, MOB processes the query using *the most efficient of the two strategies* as determined using the cost models, which leads to a lower average cost of MOB than both the *adaptive* SWIF and hTAG (Figure 3.12(b)). This behaviour shows the potential of MOB for selecting the best processing strategy for each query when the costs of the strategies are close.

Summary

Overall, the *adaptive* SWIF reduces substantially the cost of query processing compared to hTAG for small to medium query regions, while it cost is only slightly higher than for hTAG for large query regions. Thus, we also recommend the *adaptive* SWIF for reducing the cost of query processing in sensor network scenarios where the spatial range queries



(a) analytical models



(b) experimental simulations

Figure 3.12: The impact of large query region for queries without aggregation

contain a mix of region sizes and the dynamic selection of the best query processing strategies is not available.

3.8 Related Work

Directed Diffusion [IGE$^+$03] proposes a data-centric framework for query processing. Their sensor network environment is similar to ours in the sense that the query can be originated at any node, and nodes are only aware of their neighbourhood. Differently from us, nodes do not store historical data and sensing is only performed in response to a query request.

The Cougar project [YG02] investigates techniques for query processing in a centralized sensor network where the location of all sensor nodes is known. In a similar environment but with emphasis on energy efficient query processing, the Cougar project is extended in [YG03] to address problems such as routing and crash recovery, basic query plans and in-network aggregation. Madden et al. [MFHH03] propose TinyDB, which is a distributed query processor that runs on each of the sensor nodes. The authors' focus is on optimizing data acquisition for continuous queries, no data being stored locally at the nodes. To reduce the energy consumption, they also propose TAG [MFH02], an aggregation service for networks of TinyOS motes. Only spatial aggregation is performed in the network by the non-leaf nodes of the query routing tree: each node combines its data with the answers of its children as the sensors' observations are returned to the base station. All these works focus on processing continuous queries and are not suitable for the query type investigated in this chapter.

He et al. [HZGS05] investigate in-network storage and querying of data based on the time attribute. Their work complements ours by investigating the efficient processing of temporal queries. Their storage and querying schemes are constructed around the time attribute only, making them unsuitable for efficient processing of queries with several attributes part of the query predicates.

Silberstein et al. [SBY06] focus on reducing the energy cost of collecting all sensor measurements for continuous queries by using the spatial and temporal correlations present in the monitored data. They introduce two types of constraints for eliminating redundant reports. By chaining the constraints at a network scale, their system can provide a global view of the changes in the network with local energy cost. The system is not capable of handling query predicates as it is directed toward collecting all sensor measurements.

Xu et al. [XLXM06] propose the use of itineraries for covering the spatial window of the query. For queries with aggregations, their solution could be used in the second stage of SWIP. However, for queries without aggregations, the use of itineraries substantially increases the number of hops over which the query answers will reach the originator node, which, in turn, would cause a substantial increase in the energy cost of query processing.

In [CSN05] we have constructed analytical models for capturing the *average* performance of hTAG and a preliminary SWIF strategy [CNS04] (using a fixed, centrally located coordinator) given a query size and a sensor network. The analytical models presented in Section 3.5 capture the performance of the investigated strategies for *each* individual query, allowing us to select at query time the most energy efficient processing strategy.

3.9 Summary

In this chapter we have investigated the energy efficient processing of spatial range queries in a peer-to-peer sensor network. We proposed the SWIP query processing framework and, within this framework, the SWIF strategy. We constructed an analytical model to estimate the query processing cost of SWIF. We used the model to select the best coordinator location at query time, improving the energy efficiency of SWIF compared to a preliminary version [CNS04] that was using a fixed, centrally located, coordinator.

We showed both analytically and experimentally that the cost of query processing is reduced up to 10-times when the adaptive SWIF is used for small to medium spatial range queries compared to hTAG, the typical processing strategy. Only for large query regions hTAG uses less energy than SWIF (up to 2.5%). We have also shown that the analytical models can be used to dynamically select at query time the most energy efficient processing for a given query and originator node.

Chapter 4

In-network Processing of Join Queries

4.1 Introduction

The sensor network queries investigated in the literature typically have one or more of the following operators [ZG04]: selection, projection, union, grouping and aggregations. Continuous queries also allow special operators that specify the duration of the query [YG02, MFHH03] and, sometimes, the sensing frequency. The join operator has been mostly neglected in the literature. Recently, a few works tackled some aspects of the join processing problem. Bonfils and Bonnet [BB03] consider the problem of placing a correlation operator (i.e., a special join) at a node in the network. Pandit and Gupta [PG06] propose two algorithms for processing a range-join operator in the network and Yu et al. [YLZ06] propose an algorithm for processing equi-joins. These works study the *self join* problem where subsets of the sensor relation are joined. Abadi et al. [AML05] propose several solutions for the *external join* problem, where the sensor relation is joined with a relation stored at the user station. A third type of join is the *internal join* where the sensor relation is joined with relations stored locally at the nodes, such as historical statistics or preloaded relations.

In this chapter we focus on the processing of the join operator in sensor networks. In Section 1.3 we have discussed some applications where joins are an important operation for satisfying the users' information need. We are interested in minimizing the energy cost of communication during the processing of the join query, and we study this problem in the peer-to-peer sensor network environment introduced in Section 1.2.

There are two important questions that come to mind when considering the processing of join queries: *"Where should the join be processed?"* and *"How to process the join at a given location?"*. In this chapter we try to address both questions. We start by investigating several strategies for processing join queries with respect to the location where the join operator is processed. Our investigation is constructed in such a way as to be decoupled (within reasonable limits) from how the join operator is actually processed at the join locations under discussion. This is reasonable as different types of join operators (e.g., theta-, range- or equi-join) would employ different algorithms (i.e., *"how"*) for their processing, while the location where such algorithms are executed would be the same in most cases for all join operator types. To answer the *"how"* questions, we propose and discuss in details an algorithm for in-network processing of theta-join operators given a location where to process the operator. We then analyze the relation between the *"where"* and *"how"* problems and suggest how to combine the solutions to the two problems to process a join query.

Our contributions in this chapter are the following:

1. We analyze several strategies for processing join queries with respect to the location where the join operator should be processed. We investigate their suitability for a given scenario (i.e., a combination of query and network characteristics) and their performance under various conditions.

2. We develop cost models to estimate the processing cost of each strategy. We use these models in a query optimizer to dynamically select the most energy efficient processing strategy for a given query and sensor network.

3. In an extensive experimental evaluation we show that each processing strategy performs best under certain conditions. We also show that dynamic strategy selection by the model-based query optimizer outperforms any processing strategy investigated in this chapter if it is used all the time. Moreover, the optimizer makes close to optimal strategy selections in most cases.

4. We develop and analyze in detail a distributed algorithm for in-network processing of theta-join operators. We construct a model that captures the cost of the algorithm and allows one to determine the join order (i.e., $A \bowtie B$ or $B \bowtie A$) of lowest cost.

The remainder of this chapter is organized as follows. Section 4.2 presents the characteristics of the join query investigated in our work. Section 4.3 details the problem statement and presents four solutions for processing the join operator in the sensor network. We also build cost models for each of the presented solutions in this section. The evaluation of the investigated solutions is presented in Section 4.4. Section 4.5 introduces our algorithm for in-network processing of the theta-join operator. We discuss in Section 4.6 the relation between *"where"* and *"how"* to process the join operator and how the solutions to the two problems should be combined. Section 4.7 describes some of the research work related to ours and Section 4.8 concludes the chapter.

4.2 The Spatial Join Query

In this chapter we analyze the join processing problem in sensor networks for join queries having the sensor relation R^* as one of the join relations, while the other relation could be R^* (*self join*), a relation stored at the user station (*external join*) or a relation stored in the network (*internal join*). For clarity of presentation, in the following we consider *self join* queries only . In Section 4.3.6 we show that our analysis of the *self join* problem applies to the *external* and *internal joins* as well.

We impose no restrictions on the join conditions, that is, any tuple from a relation could match any tuple of the other relation. For each occurrence of the R^* relation we consider that the query also contains a spatial selection predicate constraining the tuples of the relation to belong to a region. For instance, the query *"What animals have been in both region R_A and R_B around the times of interest?"* that can appear in the context of an environmental monitoring application such as the one introduced in Section 1.3 can be expressed in an SQL-like language as:

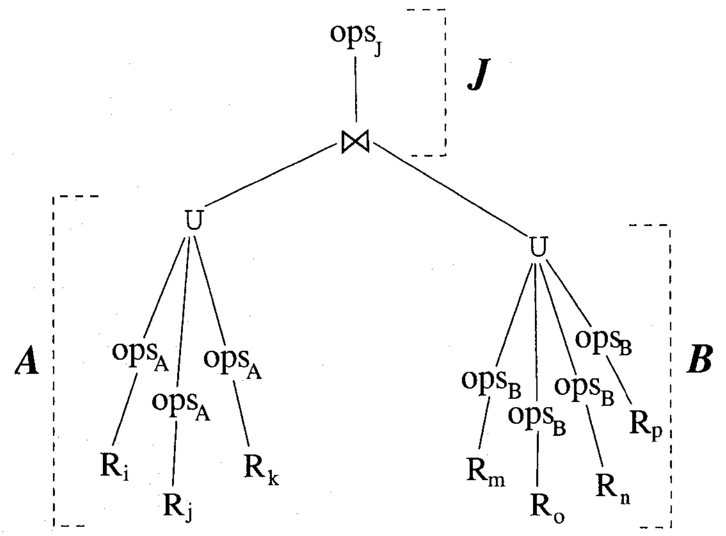


Figure 4.1: Query tree and notations

```
SELECT    S.animalID
FROM      R* as S, R* as T
WHERE     S.location IN Region R_A
    AND   T.location IN Region R_B
    AND   S.time IN TimeRange T_A
    AND   T.time IN TimeRange T_B
    AND   S.animalID = T.animalID
```

Let us denote with A the restriction of R^* to the sensor nodes in region R_A and with B the restriction of R^* to the sensor nodes in region R_B. In our presentation we will refer to the joined relations as A and B, but they are in fact restrictions of the R^* relation to the respective areas as specified in the query. The query may also contain other operators (selection, projection, etc.) on each tuple of R^* or the result of the join. Since our focus is on join processing, we consider the relations A and B as the resulting relations after the operators that can be applied on each node's relation have been applied. We consider operators that can be processed locally by each sensor node N_j on its stored relation R_j and thus they do not involve any communication. We denote with J the result of the join of relations A and B, including any operators on the join result required by the query: $J = ops_J(A \bowtie B)$. We consider operators on the join result can be processed in a pipeline fashion immediately following the join of two tuples. Figure 4.1 shows a general query tree and the notations we use.

4.3 Strategies for Processing Join Queries

4.3.1 Problem Statement

We are interested in evaluating the merits of several join processing strategies and deriving the conditions under which a strategy performs best. In our analysis we categorize the strategies based on the location where the join operation is carried and if a semi-join is used. We analyze the following processing strategies:

- **External Join**: the join is processed externally;
- **Internal Join**: the join is processed at the location of one of the join relations;
- **Mediated Join**: the join is processed in the network at a location other than the locations of either join relations;
- **Local Semi-Join**: the join is processed using a semi-join at the location of one of the join relations;
- **Mediated Semi-Join**: the join is processed using a semi-join at a location other than the locations of one of the join relations.

Before we proceed on discussing the join processing strategies, let us model the energy cost E of exchanging data in the sensor networks. As done before in this thesis, these costs will be used as building blocks for modelling the cost of each strategy. Let us first introduce the following definitions:

Definition 1: The distance between sensor nodes N_i and N_j in the sensor network is the Euclidean distance between their locations L_i and L_j. We denote this distance with $d_{N_iN_j}$.•

Definition 2: The location L_A of a relation A distributed over the sensor nodes in a region R_A is the centroid C_A of the region R_A.•

Definition 3: The distance between a sensor node N_i and a region R_A is the average distance between N_i and a node N_j in R_A: $d_{N_iR_A} = \frac{\int_{R_A} d_{N_iN_j}\, dR_A}{Area(R_A)}$. For the large scale sensor networks considered in our work, most queries will involve the sensor relation constrained to relatively small regions from the network. We approximate the location of a a query region R_A by the location of its centroid C_A. Thus, we have $d_{N_iR_A} = d_{N_iC_A}$.•

Definition 4: The distance between two relations A and B in the sensor network area is the Euclidean distance between their locations L_A and L_B. We denote this distance with d_{AB}.•

The cost of transmitting data D from node N_i to node N_j using unicast multi-hop routing is directly proportional to the size of the data s_D, the energy cost to transmit (E_t) and receive (E_r) one bit of information over one hop and the number of hops between the two nodes $h_{N_iN_j}$:

$$E(N_i, N_j, s_D) = (E_t + E_r)\; s_D\; h_{N_iN_j}. \tag{4.1}$$

The number of hops is equal to the distance $d_{N_iN_j}$ between nodes N_i and N_j divided by the average advance towards destination over one hop a_{hop}. We denote with k_u the terms independent of N_i and N_j, i.e. $k_u = (E_t + E_r)/a_{hop}$. Note that k_u is independent of the query and it is network specific. We have:

$$E(N_i, N_j, s_D) = k_u\; s_D\; d_{N_iN_j}. \tag{4.2}$$

To transmit relation A distributed over R_A to node N_j, we transmit the subset of A stored at each node in R_A to N_j. We have:

$$E(A, N_j, s_A) = \sum_{i=1}^{N_A} k_u\; s_{A_i}\; d_{N_iN_j}, \tag{4.3}$$

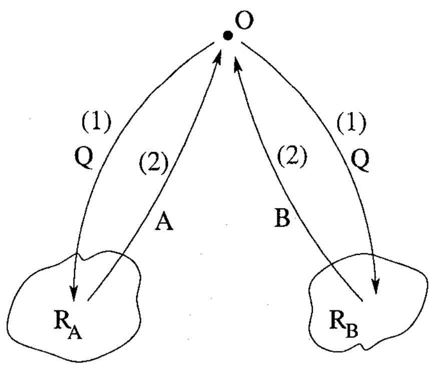


Figure 4.2: Join processing at the user station (external) - data flow and steps

where N_A is the number of sensor nodes in region R_A and A_i is the partition of A stored at the node N_i. We approximate $d_{N_iN_j}$ with d_{AN_j} (see *Definitions 2* and *3*) and we obtain:

$$E(A, N_j, s_A) = k_u \sum_{i=1}^{N_A} s_{A_i}\, d_{AN_j} = k_u\, s_A\, d_{AN_j}. \tag{4.4}$$

Finally, to transmit relation A to the nodes in R_B, we multicast relation A to region R_B. A is unicast to the centroid of the region R_B and distributed from there over R_B using broadcasting. Each node in R_B transmits A once (possibly using several messages) and receives it from every neighbor during broadcasting. Let N_n be the average number of neighbors and N_B the number of nodes in R_B. Thus, we have:

$$E(A, B, s_A) = E(A, C_B, s_A) + (E_t + N_n E_r)\, s_A\, N_B. \tag{4.5}$$

Let $k_b = (E_t + N_n E_r)$, which is independent of A and B. Note that k_b is independent of the query and it is network specific. From Equations 4.1-4.5 we obtain:

$$E(A, B, s_A) = k_u\, s_A\, d_{AB} + k_b\, s_A\, N_B. \tag{4.6}$$

We will use this notations when estimating the cost of each join processing strategy.

4.3.2 External Join

Most query processing solutions focus on processing efficiently the selection, projection and aggregation operators in the network, with the resulting data collected at the user station. For these solutions we can process a join by separately processing the query over the two relations, collecting the results (i.e., A and B) at the user station and performing the join externally. Figure 4.2 shows the data flow (query Q and relations A and B) and the processing steps for this solution. The solution is a straightforward way for extending the current query processors to handle joins and it would require no (or very little) modifications to existing algorithms. We denote with O the location of the query originator node. The cost E_{Ext} of processing the join is equal to the sum of the costs of processing the two queries:

$$E_{Ext} = E(A, O, s_A) + E(B, O, s_B) = k_u s_A d_{AO} + k_u s_B d_{BO}.$$

The external join is advantageous when the size of the relation J resulting from the join is larger than the sum of the two relations A and B or the data extracted from the sensor network is re-used for other tasks, such as external storing, other joins or building a model of the monitored environment. However, if the join selectivity factor is low (highly selective join), the network would waste energy for transmitting unnecessary tuples to the originator node and user station.

4.3.3 Local Join

An alternative for processing the join is transmitting one of the relations to the location of the other relation, performing the join locally at the sensor nodes holding this relation and returning the join result to the originator node. At first, it may seem that it is advantageous to move the smaller relation to the location of the larger one. However, as discussed next, this may not be the most efficient case due to the cost of returning the join result to the originator node. Figure 4.3(a) shows the flow of data (query Q, relation B and join result J) and the steps of the *Local Join* solution. The energy cost is:

$$E_{Loc}(A) = E(B, A, s_B) + E(A, O, s_J) = k_u s_B d_{AB} + k_b s_B N_A + k_u s_J d_{AO}.$$

As each node in R_A receives all of relation B in this strategy, each node can process locally the join between its local partition of A and B. Thus, in the *Local Join* all join types (e.g., theta- or equi-joins) can be processed this way. Note that the joins processed within each node in R_A can be performed in a distributed, pipeline fashion; as soon as a node in R_A receives a packet containing a subset of B's tuples it joins them with the local partition of A and it can send the join tuples to the originator node. Thus, each node requires only two buffers for the received and outgoing packets to process the join. We do not include the cost of the local processing in the total cost as it does not involve communication.

Similarly, when performing the join at the location of relation B we have:

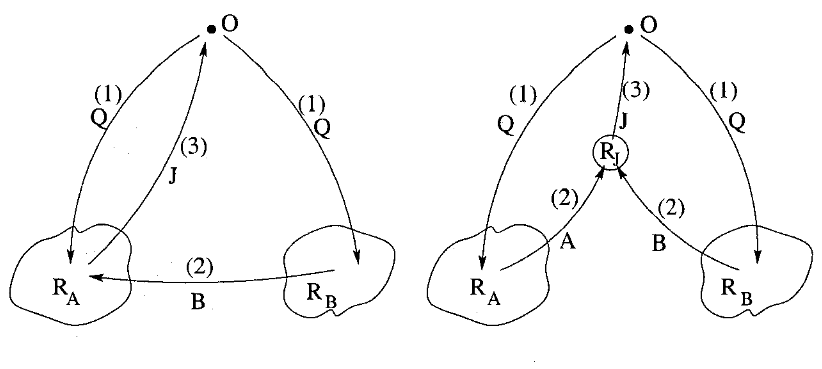
$$E_{Loc}(B) = E(A, B, s_A) + E(B, O, s_J) = k_u s_A d_{AB} + k_b s_A N_B + k_u s_J d_{BO}.$$

By comparing the costs $E_{Loc}(A)$ and $E_{Loc}(B)$ we obtain that when the relation located closer to the originator node is also larger in size, it is more efficient to process the join at that location to minimize the cost. Otherwise, the two costs must be estimated and compared to decide the best join location. We discuss in Section 4.3.6 how the originator node (or user station) may estimate the parameters used in the cost models and how the accuracy of the estimates affects the decision. For now, we assume these estimates are available.

4.3.4 Mediated Join

A third alternative for processing the join is performing the join at a location different than the location of the originator node and the locations of the join relations. To process the join, relations A and B are collected at location R_J where they are joined and the resulting relation J is transmitted to the originator node. Figure 4.3(b) shows the data flow (query Q, relations A, B and J) and the order of the processing steps. The cost of processing the join at the intermediate location R_J is:

$$E_{Med} = E(A, J, s_A) + E(B, J, s_B) + E(J, O, s_J)$$



(a) Local Join (b) Mediated Join

Figure 4.3: In-network join processing (w/o semi-join) - data flow and steps

$$= k_u s_A d_{AJ} + k_u s_B d_{BJ} + k_u s_J d_{JO}.$$

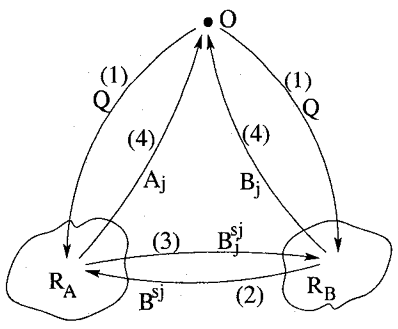
Note that the *External Join* is in fact an instance of the *Mediated Join* where locations R_J and O coincide. In the general case, the challenge is to find the optimal position for the join location such that the cost of processing the join is minimized. We need to locate the optimal R_J such that it minimizes the weighted sum of the distances from R_J to A, B and O, where the weights are the sizes of the data involved in the join. This problem is known as the the *weighted Fermat*[1] *problem*, where one wants to find the point with the property that the weighted sum of the distances from the point to the vertexes of a triangle is minimized. To find the optimal join location, we use the solution proposed by Greenberg and Robertello [GR65]. The main points of the solution are:

- The locations of A, B and O form a triangle where each location has assigned a weight equal to the amount of data it sends (s_A or s_B) or receives (s_J).
- If the weight at a location is greater than the sum of the weights at the other locations, then the join should be processed at that location.
- If the weights are equal and one of the angles of the triangle is larger than $2\pi/3$, the join location is at the vertex where the angle occurs.
- Otherwise, the location R_J lies in the triangle. The derivation of the optimal location involves non-trivial trigonometry and analytical geometry, but the terms expressing the optimal join location are computationally inexpensive. For further details see Appendix C and [GR65].

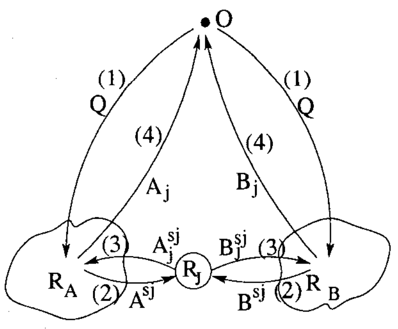
In [CG05, PG06] the authors also investigate finding the optimal join location for this scenario. They consider that the optimal join location is the *weighted centroid* of the triangle formed by A, B and O. The centroid has the property that it minimizes the weighted sum of the *squared* distances, and thus it is not optimal.

As we have previously discussed, in this section our analysis and cost models have focused on *where* to perform the join, that is, where should region R_J be located in the network. We do not discuss here what shape and size region R_J should have and how would

[1]This problem is also know as the three factory problem, the three villages problem and the weighted Steiner problem. Steiner has analyzed it in a general context involving three or more locations.



(a) Local Semi-Join



(b) Mediated Semi-Join

Figure 4.4: Join alternatives with semi-join - data flow

the join operator be performed in this region. There are three reasons that motivates this approach. First, different types of join operators (e.g.: theta- or equi-joins) would be processed very differently and our current analysis is meant to be independent of a particular join operator and associated processing algorithm, and focused on the choice of location for the processing of the join. A second reason is that the cost of processing the join over the region R_J depends on the algorithm used for its processing and dynamic sensor parameters such as the memory available at nodes at the time of processing. Finally, the cost of processing the join over R_J should be low, under our assumption for the query and network environment, compared to the cost of moving the relations between network locations. In fact, the communication cost of processing the join operator in R_J for many join queries could be as low as zero if the join operator is processed in just one node. However, when the cost of a particular algorithm used for processing the join operator in R_J is substantial relative to the cost E_{Med}, this cost should be added to the overall cost of the *Mediated Join*. In Section 4.5 we discuss a distributed algorithm for processing a theta-join operator at a given network location. et al. propose an index-based and a

4.3.5 Join Processing with Semi-Joins

For highly selective join conditions it is often the case that many tuples of one relation will not match any tuple of the other relation. Since transporting the tuples over the network is costly, one wants to avoid transporting tuples that do not join. A technique commonly used in distributed databases for reducing the cost of moving non-matching tuples over the network is the semi-join [Kos00]. In a semi-join, for each tuple only the attributes appearing in the join condition together with a tuple identifier are used for evaluating the join. Only this subset of a relation must be transported over the network to evaluate the join. Once the join is evaluated, the tuple identifiers for the joined tuples are returned to the location of the original relation. The full tuples for the joined tuples are then transmitted to the join location or the query result destination. The semi-join technique assumes that the size of the subset of attributes transmitted plus the size of the identifiers for the joined tuples is much smaller than the size of the original relation.

As most sensor nodes have several sensing units, sensor tuples tend to have a large number of attributes. If the join condition involves only some of these attributes, it may be more cost efficient to employ the semi-join technique when processing joins over the sensor data. In the following we discuss how semi-joins can be used with local and mediated join processing.

Local Semi-Join

We consider first the case when the join is performed locally at the relation A. If the join condition is highly selective, sending the entire relation B at R_A may be unnecessary and expensive. When using semi-joins, part of each tuple of B is sent to R_A, where it is joined locally at each node in R_A with the local partition of A as for the *Local Join*. For each semi-tuple of B matching a tuple of A, its identifier is returned to B. To obtain the query result at the query originator, the entire tuples of the matching semi-tuple of B must reach the originator. It is more efficient to send these entire tuples directly to the originator node than through A. After the semi-join has been fully processed at R_A, the tuples of A that have joined one or more of the semi-tuples of B are also sent to the query originator. Once the joined tuples from both A and B have reached the query originator, they can be joined externally at the user station. Figure 4.4(a) shows the data flow and the processing steps. The cost of the processing is:

$$\begin{aligned} E_{sjLoc}(A) &= E(B,A,s_{B^{sj}}) + E(A,B,s_{B_j^{sj}}) + E(A,O,s_{A_j}) + E(B,O,s_{B_j}) \\ &= k_u s_{B^{sj}} d_{AB} + k_b s_{B^{sj}} N_A + k_u s_{B_j^{sj}} d_{AB} + k_b s_{B_j^{sj}} N_B + \\ &\quad k_u s_{A_j} d_{AO} + k_u s_{B_j} d_{BO} \end{aligned}$$

where B^{sj} represents the vertical partition of B required for the semi-join, B_j^{sj} represents the tuple identifiers for the tuples of B^{sj} joined with tuples of A, and A_j and B_j are the tuples of A, respectively B, that joined during the semi-join. Similarly, if the join is performed at B, the processing cost is:

$$\begin{aligned} E_{sjLoc}(B) &= E(A,B,s_{A^{sj}}) + E(B,A,s_{A_j^{sj}}) + E(B,O,s_{B_j}) + E(A,O,s_{A_j}) \\ &= k_u s_{A^{sj}} d_{AB} + k_b s_{A^{sj}} N_B + k_u s_{A_j^{sj}} d_{AB} + k_b s_{A_j^{sj}} N_A + \\ &\quad k_u s_{B_j} d_{BO} + k_u s_{A_j} d_{AO} \end{aligned}$$

Note that the difference in cost between $E_{sjLoc}(A)$ and $E_{sjLoc}(B)$ is given by the semi-join part of the cost, as the cost of sending matching tuples to the query originator is the same. As the size of the tuple identifiers for the semi-joined tuples is much smaller than the semi-join partitions, the cost difference is determined mostly by A^{sj} and B^{sj}. If A^{sj} is larger than B^{sj}, then R_A should be the semi-join region, otherwise R_B should be the semi-join region.

Mediated Semi-Join

In this approach both relations A and B send semi-tuples to an intermediate location R_J where these tuples are joined. Once the semi-tuples are joined, the identifiers of the tuples participating in the join result are returned to the locations of the joined relations. Then the nodes send the tuples contributing to the join result to the query originator, where they are joined again to generate the query result. Figure 4.4(b) shows the data flow and the processing steps. The cost of the processing is:

$$\begin{aligned} E_{sjMed} &= E(A,J,s_{A^{sj}}) + E(B,J,s_{B^{sj}}) + E(J,A,s_{A_j^{sj}}) + \\ &\quad E(J,B,s_{B_j^{sj}}) + E(A,O,s_{A_j}) + E(B,O,s_{B_j}) \\ &= k_u s_{A^{sj}} d_{AJ} + k_u s_{B^{sj}} d_{BJ} + k_u s_{A_j^{sj}} d_{AJ} + \\ &\quad k_u s_{B_j^{sj}} d_{BJ} + k_u s_{B_j} d_{BO} + k_u s_{A_j} d_{AO}. \end{aligned}$$

To obtain the optimal join location that minimizes the cost of processing we need to minimize E_{sjMed}. The costs of sending the joined tuples from A and B to O is independent of the join location. Thus, the cost we need to minimize is:

$$\mathtt{min}(E_{sjMed}) = \mathtt{min}(s_{A^{sj}} d_{AJ} + s_{B^{sj}} d_{BJ} + s_{A_j^{sj}} d_{AJ} + s_{B_j^{sj}} d_{BJ})$$

If $s_{A^{sj}} + s_{A_j^{sj}} \geq s_{B^{sj}} + s_{B_j^{sj}}$, we re-write this equation as:

$$\mathtt{min}(E_{sjMed}) = \mathtt{min}\left(\frac{s_{B^{sj}} + s_{B_j^{sj}}}{s_{A^{sj}} + s_{A_j^{sj}}}(d_{AJ} + d_{BJ}) + (1 - \frac{s_{B^{sj}} + s_{B_j^{sj}}}{s_{A^{sj}} + s_{A_j^{sj}}})d_{AJ} \right), \qquad (4.7)$$

and if $s_{A^{sj}} + s_{A_j^{sj}} < s_{B^{sj}} + s_{B_j^{sj}}$, we re-write it as

$$\mathtt{min}(E_{sjMed}) = \mathtt{min}\left(\frac{s_{A^{sj}} + s_{A_j^{sj}}}{s_{B^{sj}} + s_{B_j^{sj}}}(d_{AJ} + d_{BJ}) + (1 - \frac{s_{A^{sj}} + s_{A_j^{sj}}}{s_{B^{sj}} + s_{B_j^{sj}}})d_{BJ} \right). \qquad (4.8)$$

Note that all terms have positive values. Since A, B and J form a triangle, we have the triangle inequality $d_{AJ} + d_{BJ} \geq d_{AB}$. To minimize Equations 4.7 and 4.8, we obtain that the optimal join location is on the segment AB, in which case the first term of Equations 4.7 and 4.8 does not depend on the location of J. Considering the remaining terms in Equations 4.7 and 4.8, we have that the join location should be at A (R_A) if $s_{A^{sj}} + s_{A_j^{sj}} \geq s_{B^{sj}} + s_{B_j^{sj}}$ and the join location should be B (R_B) otherwise. Since the optimal join location is either R_A or R_B, this approach is similar to the *Local Semi-Join* approach.

Other Strategies with Semi-Joins

For both semi-join techniques discussed above, the resulting relations after the semi-join (A_j and B_j) are joined at the user station. Another option is to perform the final join between A_j and B_j in the network using one of the join solutions without semi-join (e.g., the *Mediated Join*). The decision on which join strategy to use for the final join would depend on A_j and B_j rather then A and B. Our goal is to identify the best join strategies with respect to the join location. Since the performance of such hybrid strategies is due to the locations where the semi-join and the final join are performed, we do not analyze such hybrid strategies in this work.

4.3.6 Discussion

In this section we discuss some issues that are relevant to our work, namely: performing the join at a mediated location; the estimation of the join selectivity; reliability of routing;

region approximation; and processing external and internal joins.

Mediated Join. For the *Mediated Join* solution we have presented the problem in terms of finding the best location for performing the join. A first issue is that nodes are located at discrete location and there may be no node located at the best location. This issue is trivially solved by assigning the task of processing the join operator to the closest node to the best location. The nodes located in regions R_A and R_B do not actually need to know the location of this node as the geographic routing algorithm [KK00, BMSU01] used for packet routing will route the packets destined for the best join location to the nearest located node. A second issue is the amount of storage the node performing the join has available. If the relations to be joined are small (or at least one of them), the node may store locally the smaller relation and perform a block-nested loop join [OV99, RG00] in a pipelined fashion, in which case only very little buffer space is required for the second relation and the join result. However, if the relations to be joined are large, more nodes must participate in the join processing. We thoroughly analyze these issues in Section 4.5 where we propose a distributed algorithm for processing theta-joins. We also refer the reader to [PG06] for two distributed algorithms for processing queries with range-joins.

Estimation of Join Selectivity. Accurate estimation of join selectivity is important for any query processor as the query optimizer uses the estimate to choose the most cost-effective processing plan. For our problem in particular, estimation errors may lead to using an expensive solution for processing the query, which, in turn, would reduce the network lifetime. The cost of obtaining the estimation itself must also be considered, and it is typically a trade-off of estimation accuracy. In our context, a communication-free solution is using past query answers to estimate the join cardinality for new queries, but it may not be very accurate. A more accurate solution is using samples of the query relevant data, at the added cost of transferring these samples to the query originator node or user station. End-biased samples [EN06] is a particularly attractive solution as it provides highly accurate estimations with small sample sizes for correlated data, a typical characteristic of sensor data. In any case, the cost of estimating the join selectivity is very low compared to the cost of query processing, considering that very few data must be communicated for the estimation.

Reliability. In this work no routing tree is built and maintained, but rather geographic routing is used for routing data. This effectively means that every data packet is sent to a destination location rather than a specific node and data stops at the nearest located node. In addition, data sent from node N_i to node N_j could follow a different route than data send back from N_j to N_i as no routes are maintained. This approach alleviates the network reliability issue in part as a node on the route from N_i to a destination could die, but another route to the same destination will be discovered and used when the next data packet is sent. In our evaluation we use GPSR/GFG [KK00, BMSU01] for geographic routing which guarantee packet delivery if a route exist. A heartbeat technique [MFH02] ensures that the neighbour lists are updated regularly to account for transient or permanent node failures.

External and Internal Joins. In our presentation we have focused on processing the join between the data located at two regions in the network area. Nevertheless, we analyzed five general techniques that should apply equally well if only one of the relations is a subset of the sensor relation R^*. The other relation could be located *externally* at the user station or *internally* at one or more of the sensor nodes. Let us consider that relation A is the subset of the sensor relation R^* and B is the external or internal relation. If B is an external relation, we have that $d_{BO} = 0$. Fitting this case into the cost models it is easy to see that the external relation B should be moved in the network and the join should be performed

Parameter	**Default Value**
Average number of neighbours (N_n)	12 ($N = 1655$)
Link quality (%)	100
Size of each query region	0.5% (of R_A)
Number of tuples per node	100
Join selectivity factor(JSF)	0.001
JSF estimation error	0
Number of attributes per join tuple	6
Number of attributes per semi-join tuple	2

Table 4.1: Parameters of query and sensor network

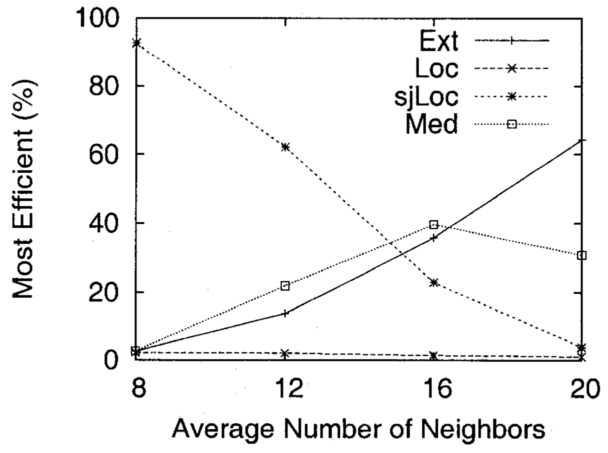
at the location of relation A if the size of A is larger than the size of the external relation B plus the join result J. REED [AML05] discusses several situations for joining the sensor relation with an external relation. In the case of the join with an internal relation stored at the sensor nodes (different than the sensor relation R^*), we have two cases. If the subset A of the sensor relation and the internal relation are located at the same set of nodes, the join can be performed in the common region. Even more, if the join involves equality conditions on the spatial attribute, the join is trivially performed locally at each node and no data needs to be communicated during the processing (except for the join result). If the subset A of the sensor relation and the internal relation are located in different regions, the join processing problem reduces to a setting similar to the one analyzed in this chapter.

4.4 Evaluation

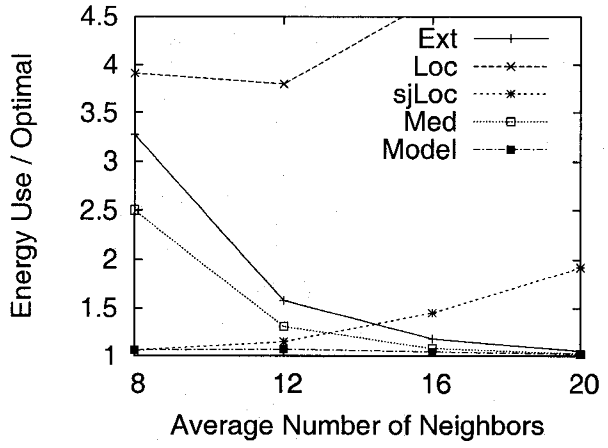
We implemented a sensor network simulator in C++ to study the performance of the solutions and evaluate the cost models. The sensor nodes' placement follows a uniform distribution over a two dimensional region. The query consists of a join operation over the data from two query regions from the network area. We express the size of the query regions as a percentage of the size of the monitored region. The query originator is a sensor node selected at random and the query regions are distributed at random in the network area. A summary of query and sensor network parameters and their default values used in our evaluations are presented in Tables 1.2 and 4.1.

A parameter particularly important in the evaluation is the ratio of the sizes of relations A, B and J. We consider that the query selects a constant number of tuples from each relevant node's relation (e.g., a temporal selection for a constant size interval). At first it may seem that this setting results in relations A and B having the same size due to the uniform node distribution. This is true only in average. In our evaluation, for a particular query with default sizes of query regions, the ratio of the sizes of the relations A and B is up to 3 due to the small size of the query regions and the sparseness of the nodes in the network area. The default value for the selectivity of the join operator is 0.001 which results in the size of the join relation J being close to the sum of the sizes of relation A and B for the default query parameters. We further detail this aspect when we discuss the impact of the join selectivity factor on the performance of the solutions.

We compare the solutions in terms of the average energy used per network node for communication while processing a query. We evaluate the performance of the *External Join* (Ext), *Local Join* (Loc), *Mediated Join* (Med) and *Local Semi-Join* (sjLoc) solutions.



(a) Most efficient (% of queries)



(b) Relative energy use vs. *Optimal*

Figure 4.5: The impact of network density

In addition, we evaluate the cost of the *Model-Based Join* solution (Model) that uses our cost models to choose and execute the most cost-efficient solution among the four on a per query basis. We compare the cost of the investigated solutions against an *Optimal Join* solution (Optimal) that would process every query using the most efficient of the four solutions.

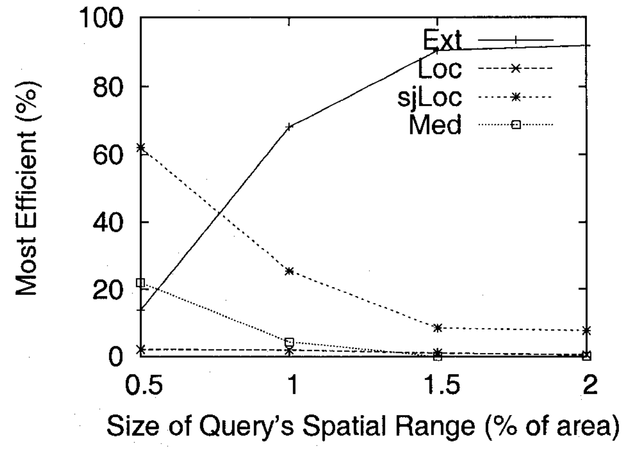
We evaluate the impact of several parameters on the performance of the solutions. Two of the parameters are characteristics of the sensor network: the network density and the packet delivery success rate. We also investigate the effect of two query characteristics on the performance of the algorithms: the size of the query's spatial range and the selectivity of the join operator.

4.4.1 Impact of Network Density

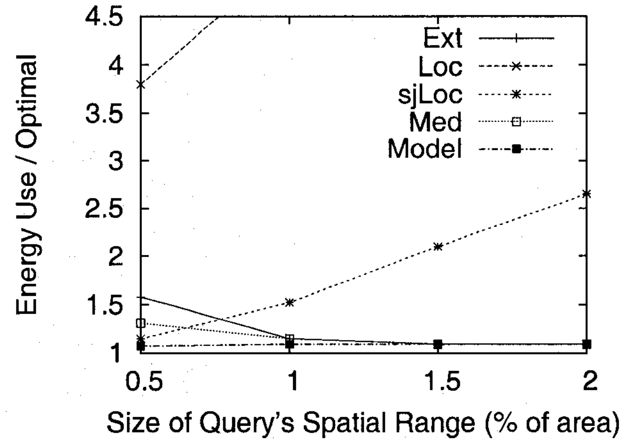
We investigate first the effect of network density on the performance of the join processing solutions. Figure 4.5(a) shows the percentage of queries for which each solution performs best. The *Local Semi-Join* processing performs substantially better than the other solutions when the network density is small. When the network density increases, the *External Join* and *Mediated Join* perform better than the *Local Semi-Join*. For very dense networks with 20 or more neighbors per node, the *External Join* becomes the most efficient one for a majority of queries. This is due to the larger number of nodes in the query regions. As more data participates in the join and more join tuples are generated, it becomes more efficient to send the relevant data to the user station and process the join there. In addition, in the case of the *Local Join* and *Local Semi-Join* solutions, the cost of distributing the semi-join tuples to the nodes in the query regions increases substantially for higher network density. This effect can be better seen in Figure 4.5(b), where we show the cost ratio of each processing solution against the cost of the *Optimal Join*. In spite of the simple cost models, the *Model-Based Join* performs best for all network densities, choosing a solution close or equal to the most efficient one for processing the join. In fact, the cost of the *Most-Based Join* solution is within 7% of the cost of the *Optimal Join* for all network densities, while the *External Join* performs up to 327% worse, the *Median Join* up to 250% worse and the *Local Semi-Join* up to 192% worse than the cost of the *Optimal Join*. The *Local Join* performs poorly for all network densities, in average between 380% and 602% worse than the cost of the *Optimal Join*, performing best for less than 2% of the queries (Figure 4.5(a)). There are two reasons for this poor behaviour: large relations are transferred over longer paths (e.g., A from R_A to R_B and J from R_B to O) than for other solutions; and broadcasting an entire relation over a region (e.g., A over R_B) is much more costly than broadcasting just a partition of it (e.g., A^{sj} over R_B as the *Local Semi-Join* does).

4.4.2 Impact of Size of Query Regions

We varied the size of the query regions between 0.5% and 2% of the network area for each query region (Figure 4.6). The increase in the size of query's range has a strong effect on the performance of the algorithms. For queries with regions larger than 1% of the network area the *External Join* performs best for a majority of queries (Figure 4.6(a)). The *Model-Based Join* solution outperforms all solutions for all query sizes (Figure 4.6(b)), but for query regions of 1.5% or larger its cost is very close to the cost of the *External Join* solution. As the *External Join* performs best for most queries with query regions larger than 1.5% of the network area and the *Model-Based Join* solution captures this behaviour, the performance of the two solutions converge. The cost of the *Mediated Join* converges as well to the same value, since the best mediator position approaches or matches the position of the originator node for large query regions. The increase in the size of the query regions causes more nodes to be relevant to the query, and thus the amount of data that participates in and is generated by the join increases. Therefore it becomes more efficient to send the data to the user station over the shortest path (as in the *External Join*) compared to moving it in the network over longer paths. In addition, as we do not vary the the join selectivity factor in this experiment, the query reaches a point where the size of the data resulting after the join becomes larger than the participating relations. For our default join selectivity ratio (0.001), this effect occurs when each query region covers 20 or more nodes. It is easy to see that in this situation sending the data to the user station is the best solution, behaviour well captured by the *Mediated Join* which moves the join location to the originator node. *Local Join* performs poorly again for the reasons discussed in the previous experiment. In addition, its cost increases sharply with the increase in the size of query regions as a larger relation is broadcast over a larger region.



(a) Most efficient (% of queries)

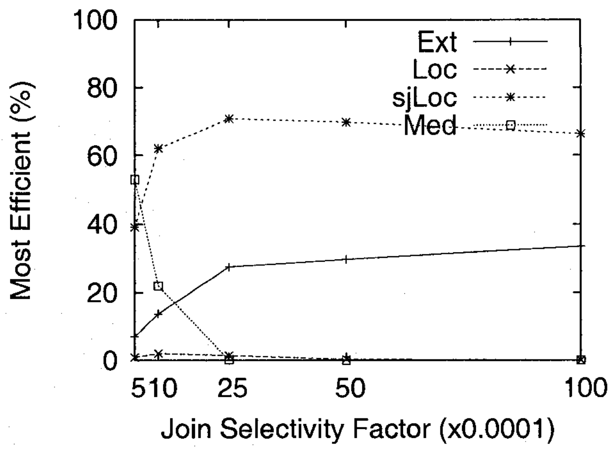


(b) Relative energy use vs. *Optimal*

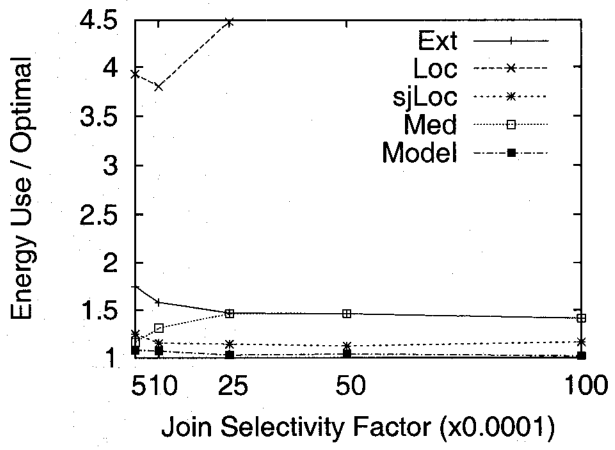
Figure 4.6: The impact of the size of the query regions

4.4.3 Impact of Join Selectivity Factor

To evaluate the effect of the join selectivity factor (JSF) on the performance of the solutions, we varied the factor between 0.0005 and 0.01. The lower range corresponds to 5 tuples being generated by the join of the data from each pair of nodes from the two query regions, while the higher range corresponds to 100 tuples being generated for each pair of nodes. Considering our default size of the query regions and network density, this effectively translates overall into the size of the join result being smaller than the joined data for the lower join selectivity factor (highly selective join condition), and larger for the higher selectivity factor. While we do not evaluate directly the impact of joins followed by aggregations, the behaviour of the solutions for small JSFs is similar to that of aggregations applied on top of joins with larger JSFs. What matters in both cases is the size of relation J compared to the sizes of A and B. Also note that the join selectivity factor has no effect on the cost of the *External Join* as the join is performed at the user station, and only a small effect on the cost of the *Local Semi-Join* as the joining tuples from each query region are joined at



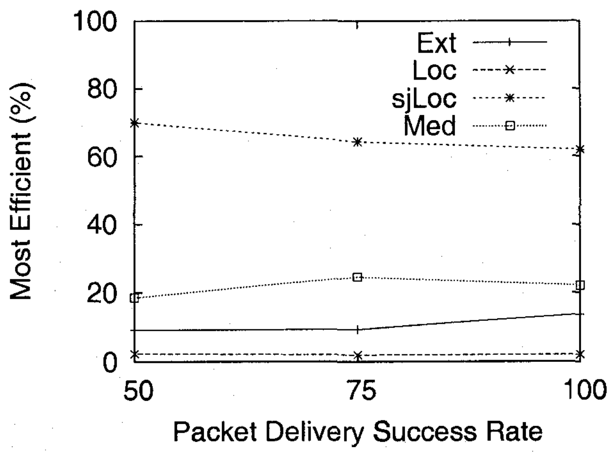
(a) Most efficient (% of queries)



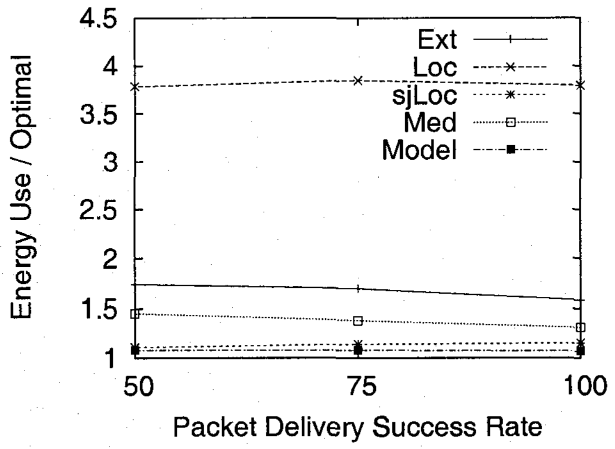
(b) Relative energy use vs. *Optimal*

Figure 4.7: The impact of join selectivity factor (JSF)

the user station as well. The solution most affected by the size of the join selectivity factor is the *Local Join* since the join is performed at one of the query regions and the join data is then transferred to the user station. While the *Mediated Join* also performs the join at a location in the network, the mediated location is dynamically selected at query time and it coincides for some queries with the location of the query originator when the size of the join data grows larger. When this effect occurs the *Mediated Join* and the *External Join* behave similarly (Figure 4.7(b)), and in the efficiency graph (Figure 4.7(a)) we consider that the *External Join* is the most efficient solution between the two. The experimental evaluation also shows that the *Mediated Join* performs best for small selectivity factors, closely followed by the *Local Semi-Join*. With the increase in the join selectivity factor (less selective join condition), the *Mediated Join* approached the *External Join* in behaviour and performance, and the *Local Semi-Join* is only slightly affected. The *Model-Based Join* solution is able to pick these effects on the solutions, performing best and within 8% of the cost of the *Optimal Join* for all join selectivity factor sizes. The performance of the *Local Semi-Join* decreases slowly for increasing selectivity sizes, and in our setup it is the most efficient of



(a) Most efficient (% of queries)



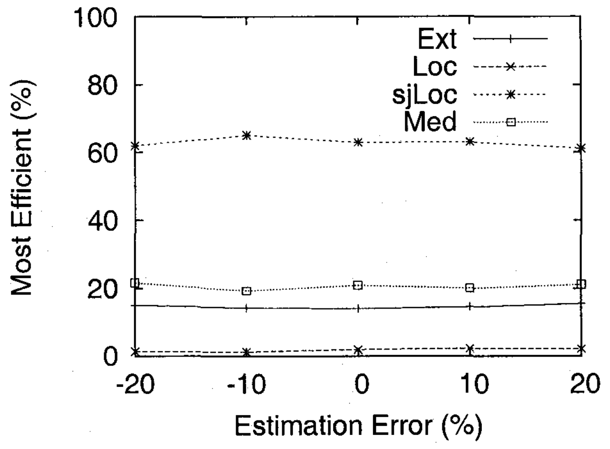
(b) Relative energy use vs. *Optimal*

Figure 4.8: The impact of packet delivery success rate

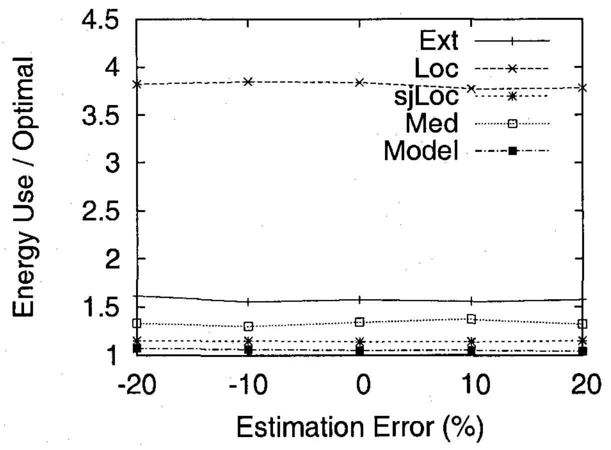
the four solutions for a majority of queries when the size of the join selectivity factor is larger than 0.001.

4.4.4 Impact of Realistic Communications

Up to this point in our experiments we have considered a reliable communication environment, where no messages (packets) are lost during transmission. This assumption allowed us to investigate the performance of the solutions independent of the communication environment. Unfortunately, the typical environments where sensor networks operate affect the quality of transmission negatively, with packet delivery failing at times. To capture this unreliable conditions, we have set each communication link between two nodes with a packet delivery success rate. In addition, the quality of communication links is typically not symmetric, so we varied the delivery rate with up to 10% for the two directions of each communication link. When a packet does not reach its destination in a one hop transmission, the source node re-transmits it until it receives acknowledgement of receival.



(a) Most efficient (% of queries)



(b) Relative energy use vs. *Optimal*

Figure 4.9: The impact of the estimation error of JSF

Figure 4.8 represents the performance of each solution under three packet delivery success rates. Each success rate represents the lower bound in terms of success rate for the links between two nodes located within the wireless communication range of each other, while the higher bound is the 100% delivery (no packet loss). The delivery success rates are randomly distributed in this interval. While the energy cost of each solution increases for decreasing delivery rates, the *relative* performance of the solutions remains unchanged. This suggests that the *relative* performance of the solutions is not affected by unreliable communication mediums and the cost model can be used to capture the *relative* performance of the solutions regardless of the reliability of the communication environments.

4.4.5 Impact of the Estimation Accuracy for the Join Selectivity Factor

The cost models used in all solutions, except the *External Join*, use the join selectivity factor to estimate the size of the resulting join relation. In our previous experiments we have considered that this factor is estimated correctly to reduce the influence of its error (if any) on the effects of the other parameters. In this experiment we consider that the estimation of the factor is not accurate, and we investigate the effect that the estimation error has on the performance of the solutions. We consider both the underestimation and overestimation errors, varying the error of the estimated join selectivity factor from -20% to $+20\%$ from the actual factor. Figure 4.9 shows the effect of this variation on the investigated solutions. While quantitatively the cost of all solutions with JSF estimation error varies from their cost with accurate JSF estimation, the *relative* performance of the solutions is very stable with respect to the JSF estimation error.

4.4.6 Impact of the Location of the Query Regions and Originator Node

Our previous experiments have considered a setup where the query originator could be located anywhere in the network and there is no restriction on the location of the query regions. We have investigated the performance of the solutions on three more setups. In one of these setups the query originator is located in the upper-left corner of the network area and the query regions can be anywhere (a typical setup when using one fixed base-station). In the other two setups considered, one with a randomly distributed originator and the other with a corner originator, we restricted the locations of the query regions so that they are far away from each other and the originator. The experimental evaluation on these three setups has shown qualitatively similar behaviours of the solutions as for the setup discusses in details above. This suggests that the *relative* performance of the solutions is not affected by the relative locations of the query originator and query regions and that the cost models, while simple, are sufficient for correctly predicting the *relative* performance of the solutions regardless of the application scenario.

4.4.7 Summary

Overall, the evaluation shows that no join processing solution performs best for all queries. The *Local Semi-Join* is especially suitable when the query regions are small and the network density is low. The *External Join* performs best for dense networks and large query regions, while the *Local Join* does not perform well under any investigated condition. The *Mediated Join* adapts well to the query characteristics and it is a good alternative to the *External Join* and *Local Semi-Join* solutions if performing the join at the user station is not acceptable. In any case, as energy is a vital resource of the network, one should not settle to use only one solution for processing all queries, but rather select for each query the best solution. We have shown that using simple cost models to capture the relative performance of the investigated solutions is an effective way of selecting an efficient solution for each query, and it outperforms by a large margin processing all queries with the same solution.

4.5 DIJ: A Distributed Algorithm for Theta Joins

In this section we focus on *how* to process the join between two relations A and B at a given network location, where the relations are distributed over subsets of network nodes. In Section 4.3 we have investigated several join processing solutions with respect to the location of the network region *where* the join is performed.

Most join processing strategies presented in the literature (including those discussed in Section 4.3) either assume that nodes have sufficient memory to buffer the partition of the join relations assigned to them for processing, or that the amount of memory available at each node is known in advance and the assigned data partitions can be set accordingly. This assumptions are unrealistic for most scenarios. It is well known that sensor networks are very constrained on main memory and the energy cost of using their flash storage (for those devices that have it) is rather prohibitive for data buffering during query processing. In addition, in large scale sensor networks, it is not feasible for the sensor nodes or the user station to be aware of up-to-date information on memory availability of all network nodes.

4.5.1 DIJ: A Distributed Join Processing Algorithm

Join processing in sensor networks is a highly complex operation due to the distributed nature of the processing and the limited memory available at nodes. We discuss some of the requirements of an effective and efficient join processing algorithm for sensor networks, namely: distributed processing, memory management and synchronized communication.

- **Distributed processing.** In large scale sensor networks the join operator must be processed in a distributed manner using localized knowledge. For some queries no single node can buffer all the data required for the join. In addition, no node (or user station) has global network knowledge to find the optimal join strategy. As nodes have information only about their neighbourhood, the challenge is to take correct and consistent decisions among nodes with respect to processing the join operator. For instance, when the join operator is evaluated over a group of nodes, each node in the group must route and buffer tuples such that each pair of join tuples is evaluated exactly once in the join.

- **Memory management.** Each node participating in the processing of the join operator must have sufficient memory to buffer the tuples that it joins and the resulting tuples. For some join queries the join relations are larger than the available memory of a single node. Typically, several nodes must collaborate to process the join operator, pooling their memory and processing resources together. A join processing algorithm should pool these resources together and allocate tasks and data among the participating nodes such that the efficiency of the processing is maximized.

- **Synchronized data flow.** Inter-node communication must be synchronized such that a node does not receive new tuples to process when its memory is full. Otherwise, the node would have to drop some of the buffered or new tuples, which is unacceptable as it may invalidate the result of the join. Thus, each node must fully process the join tuples it holds before receiving any new tuples. A similar problem occurs also for the nodes routing the data. A parent node routing data for multiple children may not be able to buffer all received data before it can forward it. Thus, a join processing algorithm should carefully consider the flow of data during its execution.

In this section we propose and detail a distributed join processing algorithm which considers the above requirements. In our presentation we focus on the join between two restrictions (A and B) of the R^* relation, where the join condition is general (theta-join). Thus, every pair of tuples from relations A and B must be verified against the join condition. Relations A and B are located within regions R_A and R_B and they are joined in the network in a *join region*, denoted by R_J. This setup is similar to the one used in Section 4.3. Again, the problem of finding *where* (i.e., the network location) to process the join operator is orthogonal to our problem of *how* to process the join operator and it has been investigated in Section 4.3 and elsewhere [CG05, YLZ06]. In fact, our algorithm is general with respect to the join relations and their locations and could be used within the core of other previously proposed join solutions (e.g., the *Mediated Join*), including solutions for processing semi-joins (e.g., [YLZ06]). For clarity of presentation we describe our algorithm for processing the join operator in the context of the *Mediated Join* strategy introduced in Section 4.3.4.

Let us briefly review the *Mediated Join* strategy: relations A and B are sent to the join region (R_J) where they are joined and the resulting relation J is transmitted to the query originator node. Figure 4.3(b) shows in overview the query processing steps and the data flow. The *Mediated Join* strategy seems straightforward based on this high-level description, but there are several issues that must be carefully addressed in the low-level sensor implementation to ensure the correctness of the query result:

- How to ensure that both relation A and B are transmitted to the same region R_J?
- How large should region R_J be to have sufficient resources, i.e., memory at nodes, to process the join?
- How should A and B be transmitted such that the join is processed correctly at the nodes in R_J?
- How to process the join in R_J such that the join is processed correctly using minimum resources?

A typical algorithm for processing theta-joins in traditional database systems is the block-nested loop join (*BNLJ*). If A and B are the join relations, the steps of *BNLJ* are:

```
FOREACH block $B_A$ of tuples in $A$ DO
    FOREACH block $B_B$ of tuples in $B$ DO
        Join tuples in $B_A$ with tuples in $B_B$
```

Since this strategy goes over relation B for each block B_A of A, relation B must be "read" multiple times. In a distributed environment such as sensor networks, "read" multiple times translates into communicated multiple times. Such a strategy would increase dramatically the communication cost. Our join processing technique is inspired by this strategy, with the major difference that each join relation is communicated only once.

We now describe in details *DIJ*, our join processing technique, in the context of the *Mediated Join* strategy. The steps of the *DIJ* technique are:

1. *Multicast the query from originator node O to nodes in R_A and R_B.* Designate the nodes closest to the centres C_A and C_B of the regions R_A, respectively R_B, as regional coordinators. Designate the coordinator location C_J for join region R_J. Disseminate the information about the coordinators along with the query.

2. *Construct routing trees in regions R_A and R_B rooted at their respective coordinators C_A and C_B.*

3. *Collect information on the number of query relevant tuples for each region at the corresponding coordinators.* Each coordinator sends this information to coordinator C_J of the join region R_J.

4. *Construct the join region.* C_J constructs R_J so that it has sufficient memory space at its nodes to buffer A.

5. *Distribute* A *over* R_J.

   (a) C_J asks C_A to start sending packets with tuples. Once C_J receives A's tuples (in packets), it forwards them to a node in R_J with available memory.

   (b) Upon receiving a request for data from C_J, C_A asks for relevant tuples from its children in the routing tree. The process is repeated by all internal tree nodes until all relevant tuples have been forwarded up in the tree.

6. *Broadcast* B *over* R_J

   (a) Once C_J receives a signal from C_A that it has no more packets (i.e., tuples) to send, C_J asks for one packet with tuples from C_B. When the packet is received, it is broadcast to nodes in R_J.

   (b) Each node in R_J joins the tuples in the packet received from B with its local partition of A, sending the resulting tuples to O. Once the join is complete, each node asks for another packet of B's tuples from C_J.

   (c) Upon receiving a request for tuples from C_J, C_B asks for a number of join tuples from its children in the routing tree. The process is repeated by internal tree nodes if they cannot satisfy the request alone.

   (d) Once C_J receives requests for B's tuples from all nodes in R_J, Step 6 is repeated unless C_B signals that it has no more packets (i.e., tuples) to send.

In the steps above we chose, only for the sake of presentation, that relation A is distributed over the nodes in R_J and relation B is broadcast over the nodes in R_J. Although the steps above are symmetric if the roles of A and B are switched, the order *does* matter. In Section 4.5.2 we explore this issue and show how to determine which relation should be distributed and which should be broadcast in order to minimize the cost of the processing the join operator.

Let us draw a parallel between *DIJ* and *BNLJ* techniques. In *BNLJ*, relation A is "read" once block by block into memory. In *DIJ*, each block of relation A is communicated once. Differently from *BNLJ*, each block of relation A is not joined immediately with B, but it is stored at a node in region R_J. In *BNLJ*, each block of relation B is "read" several times, once for each block on A. In *DIJ*, each block of tuples in B is transmitted only once from R_B to R_J. Then, each block of B is broadcast over the nodes holding A's blocks and the join is performed in parallel at the nodes. One may see *DIJ* as an extreme case of *BNLJ*, where the external loop of *BNLJ* is replaced by a distribution of tuples at the nodes in R_J. The great advantage of the distribution is that B is transmitted only once and the join executes in parallel at the nodes in R_J.

Steps 1-3 of *DIJ* are typical to in-network query processing and do not present particular challenges. In Step 4, the join coordinator C_J must request and pool together the memory of other nodes in its vicinity for allocating relation A to these nodes (in Step 5a). This is a non-trivial task as C_J does not have information about the nodes in its vicinity (except its 1-hop neighbours). Steps 5 and 6 also pose a challenge, that is, how to control the flow of tuples efficiently without buffer overflows, ensuring correct execution of the join. We detail these steps in the following.

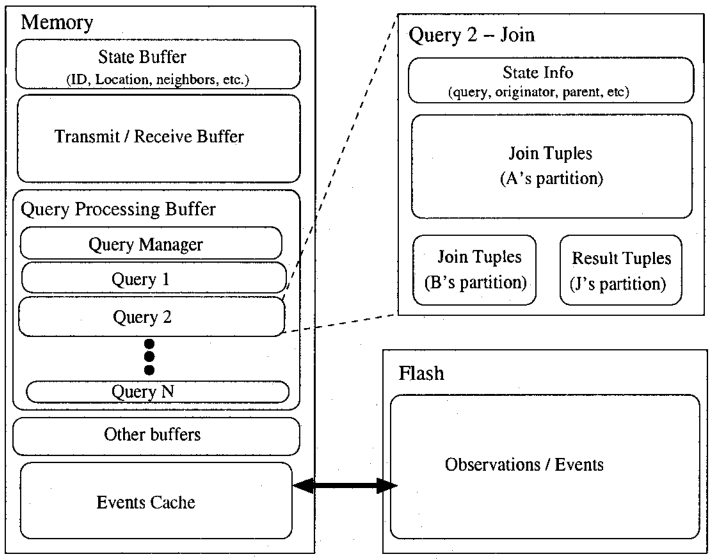


Figure 4.10: Memory allocation scheme

Constructing the join region (Step 4)

Once node C_J receives the size of the join relations A and B from C_A and C_B (in Step 1), it must find the nodes in its vicinity where to buffer relation A. *DIJ* uses the following heuristic for this task, which we refer to as **k-hop-pooling**:

> **k-hop-pooling.** If C_J alone does not have sufficient memory to buffer relation A, C_J asks its *1-hop* neighbours to report how much memory they have available for processing the query. If relation A is smaller than the total memory available at the *1-hop* neighbours, C_J stops the memory search. Otherwise, C_J asks its *2-hop* neighbours to report their available memory. This process is repeated for *k-hops*, where k represents the number of hops such that the total memory available at the nodes up to k hops away from C_J plus the memory available at C_J is sufficient to buffer relation A.

An interesting question is how much memory should a node allocate for processing a particular query. If the sensor network processes only one join query at a time (e.g., there is a central point that controls the insertion of join queries in the network), then nodes can allocate all the memory they have available for processing the join. However, if nodes allocate all their memory for a query, but several join queries are processed simultaneously in the network, it may happen that a coordinator C_J will not find any nodes with available memory in its immediate vicinity, forcing it to use farther away nodes during processing, and, thus, consuming more energy. For networks where multiple queries may coexist in the network, nodes should allocate only a part of their available memory for a certain query, reserving the rest for other queries. How to actually best allocate the memory of an individual node is orthogonal to our problem. In this work we assume that nodes report as available only the memory they are willing to use for processing the query that the memory was requested for. Figure 4.10 shows a possible memory allocation scheme at a node.

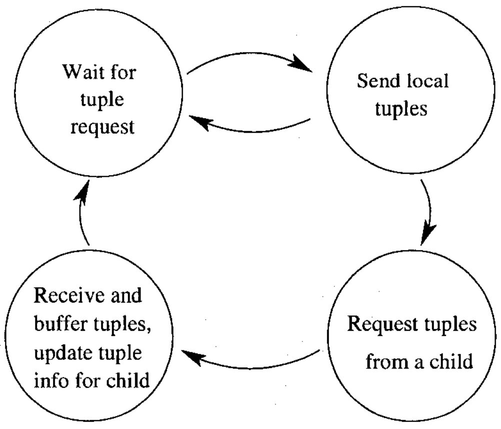

Wait for tuple request
Send local tuples
Request tuples from a child
Receive and buffer tuples, update tuple info for child

Figure 4.11: States of a node during tuple routing

Distributing A over R_J (Step 5)

In this step two tasks are carried out concurrently: C_A requests and gathers relevant tuples (grouped in data packets) from R_A, and C_J distributes the packets received from C_A over R_J.

Once the set of k-hop neighbours that will buffer A has been constructed, C_J asks for relation A from C_A, packet by packet, and distributes each packet of A's tuples in a round-robin fashion to its neighbours, ordered by their hop distance to C_J. When deciding to which node to send a new packet with A's tuples, a straightforward packet allocation strategy would be for C_J to pick a node from its list and send to it all new packets with A's tuples until its allocated memory is full. This strategy has two disadvantages. As all packets use the same route (for most routing algorithms) to get to their destination node, their delivery will be delayed if there is a delay on one of the links in the route. Also, consecutive packets may contain tuples with values such that they all (or many of them) will join with the same tuple in B. In this case, the node holding all these tuples will generate many result tuples that have to be transmitted, delaying the processing of the join. The hop-based round-robin allocation also ensures that all k-hop neighbours have a fair chance of having some free memory at the end of the allocation process, memory that can be used to process other queries.

Once node C_A receives a request for tuples from C_J, it has to gather relevant tuples from R_A. If C_A would simply broadcast the tuple request in the routing tree constructed over R_A, nodes in R_A will start sending these tuples toward C_A. As each internal tree node has (likely) several children, it should receive and buffer many packets before being able to send these packets out. Some nodes may not be able to handle such a data flow due to lack of buffer space, possibly dropping some of the packets. To ensure that no packets are lost due to lack of buffer space, we propose a flow synchronization scheme where each node will only buffer one packet. In this scheme, the request for A's tuples is transmitted one link at a time. Each node in the routing tree is in one of the following states during the synchronized tuple flow (Figure 4.11):

- Wait for a tuple request from the parent node (or C_J in the case of C_A) in the routing tree constructed in *Step 2*.
- Send local tuples (from the local storage or receive buffer) to the parent node.

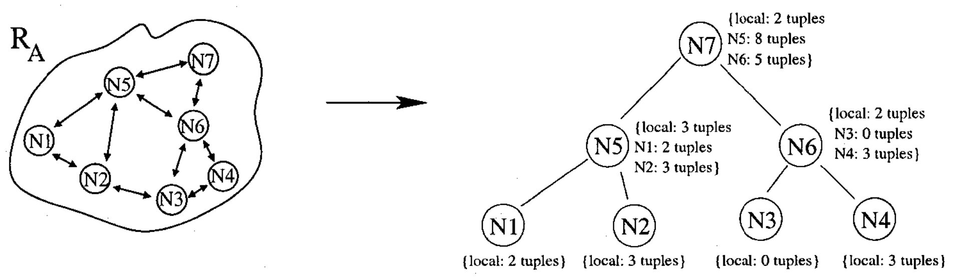


Figure 4.12: Join tuples information at nodes

- If buffer space has been freed and there are relevant tuples available at the children nodes in the routing tree, request tuples from a child node that still has tuples to send. Figure 4.12 shows the routing tree for a region and the information maintained in each node of the tree as tuples are routed from either R_A or R_B to R_J. Note that the number of tuples that each child node will provide has been collected in *Step 3*.
- Receive tuples from the child, buffer the tuples and update the number of tuples that child still has available.

Once a node has forwarded to its parent all of A's tuples from its routing sub-tree, it can free all buffers used for processing the query.

Broadcasting B over R_J (Step 6)

The collection of B's tuples proceeds much like the collection of A's tuples, with one important difference. Whereas C_A gathers and sends *all* of the relevant tuples of A as a a result of a single tuple request from C_J, C_B only sends *one* packet with tuples for each request it receives from C_J. This way, C_J can broadcast such a packet of tuples to all nodes in R_J, wait until all nodes fully process the local joins and send the results, and then request a new packet of tuples from R_B when each node in the join region R_J is ready to receive and join a new set of tuples.

4.5.2 Selecting the Relation to Be Distributed

In the previous discussions we have assumed for clarity of presentation that relation A is distributed over the nodes in region R_J and B is broadcast over the nodes in the region. An interesting question is which of the two join relation should be distributed and if the choice makes a major difference in cost.

Let us focus first on which of the two join relation should be distributed and, subsequently, which should be incrementally broadcast. To decide on this matter, the query optimizer has to estimate the cost of the two options (i.e., distribute A or B) and compare their costs to decide which alternative is more energy efficient. For generality, we derive in the following a cost model for processing the join by distributing relation R_d and broadcasting relation R_b. The actual relations A and B can then be substituted into R_d and R_b (or vice-versa) to estimate the processing costs.

Considering the steps of *DIJ*, the cost of query processing can be decomposed into a sum of components, with one component associated to each step. Several of these components are independent of the choice of the relation that is distributed. Thus, they do not affect the decision of which relation to distribute and do not need to be derived. For instance, we have the cost for disseminating the query in regions A and B (Step 1) and the cost for constructing the routing tree over regions R_A and R_B (Step 2). These costs are identical when processing the join by distributing A or B and do not affect the decision. The steps that have different costs when A or B are the distributed relation R_d are the construction of the join region R_J (Step 4), the distribution of the relation R_d (Step 5a) and the broadcast of the relation R_b (Step 6a). Note that we are only interested in differences in the communication cost between the two alternatives.

Constructing the join region (Step 4)

As discussed in Section 4.5.1, we use the **k-hop-pooling** strategy to construct the join region R_J. In each round of memory allocation, C_J broadcasts its request for memory in a hop-wise increasing fashion, until sufficient nodes with the required buffer space are located.

During a round h, each node within h-hops from C_J broadcast the memory request and its 1-hop neighbours receive the request message. Thus, the total energy cost is:

$$E_4^{memreq} = \sum_{h=0}^{k-1}(E_t N_n^h M_r + E_r N_n^h N_n^1 M_r),$$

where N_n^h represents the average number of nodes within h hops from a node, E_t and E_r represents the energy required to transmit, respectively receive, one bit of information and M_r represents the size of the memory request message (in bits). N_n^h is a network-dependent value independent of our technique and it is derived in the Appendix D.

When a node receives a memory request message for the first time, it allocates buffer space in its memory and sends the memory information to C_J. The nodes located h-hops away from C_J perform two tasks: they send their own memory information to the nodes located $h-1$ hops away; and they forward the information they have received from the nodes located between $h+1$ and k hops away from C_J. If we denote by M_i the size of the memory information for one node, the total energy cost of collecting the information on available memory is:

$$\begin{aligned} E_4^{meminfo} &= \sum_{h=1}^{k}((E_t + E_r)(N_n^h - N_n^{h-1})M_i \\ &\quad +(E_t + E_r)(N_n^k - N_n^h)M_i) \\ &= (E_t + E_r)(kN_n^k - \sum_{h=1}^{k-1} N_n^h)M_i \end{aligned}$$

Note that $(N_n^h - N_n^{h-1})$ represents the number of nodes h-hops away and $(N_n^k - N_n^{h-1})$ represents the number of nodes located more than h and up to k hops away from C_J. The total energy cost of the fourth step of *DIJ* is:

$$E_4 = E_4^{memreq} + E_4^{meminfo}.$$

Note that the costs of the Step 4 do not depend on the join relations directly, but through k which determines the size of the join region R_J and it is determined by the size of the join relation R_d.

Let B_s be the average size (in bits) of the buffer space that each node in R_J can allocate for processing the query. The minimum number of nodes that must be used to store relation R_d in region R_J is $\frac{||R_d||}{B_s}$, where $||R||$ denotes the size (in bits) of relation R. Since nodes are added to R_J in groups based on their hop distance, k is the lowest number of hops such that the nodes within k hops from C_J have sufficient buffer space to buffer R_d:

$$k = \{\mathtt{min}\, h \mid N_n^h B_s \geq ||R_d||\}.$$

Distributing R_d over R_J (Step 5a)

In Step 5a of *DIJ*, C_J receives and distributes relation R_d at the nodes in R_J. Nodes located h hops away from C_J receive from the nodes located $h-1$ hops away from C_J partitions of R_d of size B_s for buffering. They also route toward their destination the partitions B_s allocated to the nodes between $h+1$ and $k-1$ hops away from C_J, as well as the partitions allocated to the nodes k hops away. Note that nodes located k hops away will only buffer whatever is left of R_d instead of B_s as the other nodes do. Therefore, the total energy cost of distributing R_d at the nodes in R_J is:

$$\begin{aligned} E_{5a} = \quad & \sum_{h=1}^{k-1} (\, (E_t + E_r)(N_n^h - N_n^{h-1})B_s \\ & + (E_t + E_r)(N_n^{k-1} - N_n^h)B_s \\ & + (E_t + E_r)(||R_d|| - N_n^{k-1}B_s)\,) \\ = \quad & (E_t + E_r)((k-1)\,||R_d|| - B_s \sum_{h=0}^{k-2} N_n^h). \end{aligned}$$

Broadcasting R_b over R_J (Step 6a)

In Step 6a of *DIJ*, C_J broadcasts relation R_b (packet by packet) over the nodes in R_J, where it is joined with the buffered partitions of R_d. Note that only the nodes in R_J up to $k-1$ hops away from C_J need to broadcast R_b so that all nodes participating in the join receive it. The total energy cost of the broadcast is:

$$E_{6a} = E_t N_n^{k-1} ||R_b|| + E_r N_n^{k-1} N_n^1 ||R_b||. \tag{4.9}$$

Discussion

Using the cost models for Steps 4, 5a and 6a of *DIJ*, C_J can determine which of the two join relation should be R_d and which should be R_b. To calculate the energy costs, C_J need to know the value of the parameters used in the models. C_J learns the size of the join relation A and B in the Step 3 of *DIJ*. C_J can estimate B_s based on the size of the available memory at itself and its 1-hop neighbours. We show in the Appendix D how N_n^h can be estimated. The other parameters used in the cost model are network or algorithm constants.

4.5.3 Cost Model Evaluation

The cost model developed in Section 4.5.2 allows C_J to choose which of the join relation should be distributed (R_d) in the the join region R_J and which should be broadcast (R_b)

Table 4.2: Cost model parameters

Parameter	Default Value
Average number of neighbours (N_n^1)	12 ($N = 1655$)
Number of tuples in A	500
Number of tuples in B	500
Number of tuples per node in R_J (B_s/T_s)	25
Size of a tuple (T_s)	192 bits
Size of a memory request message (M_r)	8 bits
Size of a memory information record (M_i)	80 bits

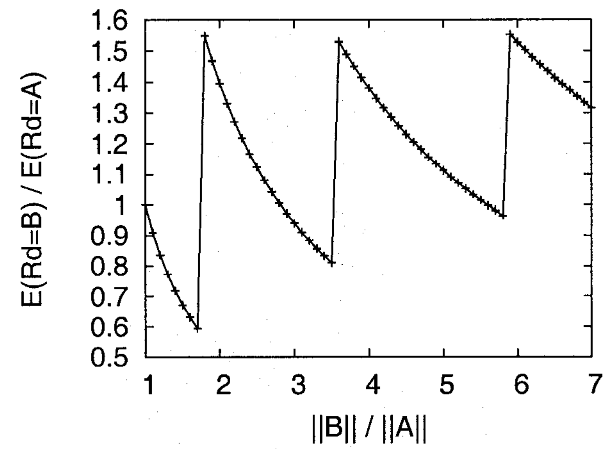


Figure 4.13: Energy cost ratio when $R_d = A$ and $R_d = B$ for variations in the relative size of the join relations

to minimize the cost of processing the join operator. In this section we further investigate the behaviour of *DIJ* based on the cost model. In our evaluation we consider a sensor network with nodes uniformly distributed over a two dimensional region. We are interested in evaluating the relative performance of two alternatives: distributing relation A in R_J and broadcasting relation B over R_J (denoted by $R_d = A$); and distributing relation B in R_J and broadcasting A over R_J (denoted by $R_d = B$). Our measure of efficiency is the energy used for communication while processing the join operator. Thus, we compare the energy cost $E(R_d = A)$ of *DIJ* when A is the distributed relation with the cost $E(R_d = B)$ when B is distributed. We only consider the energy costs of the Steps 4, 5a and 6a as they are the ones that determine the difference between the two options of *DIJ* for which relation to distribute and which to broadcast. Figures 4.13, 4.15 and 4.16 evaluate the relative cost of *DIJ* when $R_d = B$ compared to the cost when $R_d = A$: $E(R_d = B)/E(R_d = A)$. When the cost ratio is equal to 1, both alternatives for which relation to distribute have the same cost. When the cost ratio is lower than 1 it is more efficient to distribute relation B ($R_d = B$) and broadcast relation A, while for cost ratios higher than 1 relation A should be the distributed relation ($R_d = A$). The cost model parameters and their default values used in our evaluation are presented in Table 4.2.

In the first experiment, we evaluate the relative costs of the Steps 4, 5a and 6a for different ratios between the sizes of the join relations. We keep relation A fixed and we vary

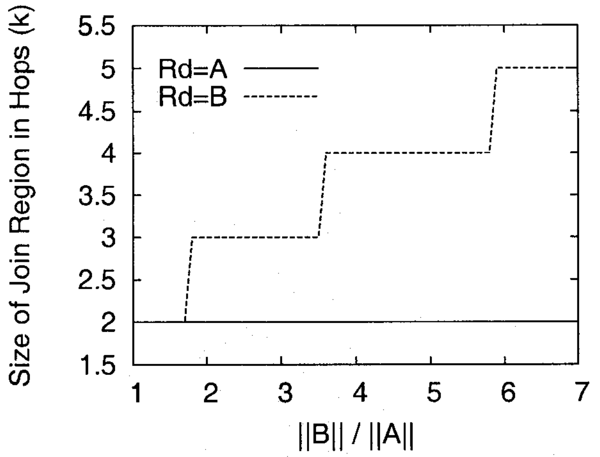


Figure 4.14: Size of join region (R_J) in number of hops when $R_d = A$ and $R_d = B$ for variations in the relative size of the join relations

B such that it is between 1 to 7 times larger than A. We evaluate the join processing costs when relation A, respectively B, is the relation distributed over the join region ($R_d = A$ or $R_d = B$). Figure 4.13 shows the relative performance of the two join alternatives. When both relations have the same size ($||B||/||A|| = 1$), the cost of the processing the join operator is the same for both alternatives, as one would expect. For rations $||B||/||A||$ up to 6, the best relation to distribute changes with variations in the ratio. For ratios higher than 6 (we only show ratios up to 7 in the graph), the smaller relation (A) is always the relation that should be distributed for lower join processing costs. The sharp changes in the cost ratio are caused by the increase in the number of hops (k) that are required so that R_J is sufficiently large to store R_d. When k increases, the cost of broadcasting relation R_b in Step 6a increases substantially as another set of nodes are added to R_J. Note that k does not vary for $R_d = A$ as the size of A does not change, but it does vary between 2 and 5 for $R_d = B$ as shown in Figure 4.14. For instance, when B is between 1.8 to 3.5 times larger than A, nodes up to $k = 3$ hops away from C_J are required to buffer relation B when distributed over R_J. As k stays constant for these ratios, the cost of broadcasting relation A over R_J stays constant as well. At the same time, as the size of B increases, the cost of distributing A over R_J and broadcasting B increases. Thus, the cost ratio $E(R_d = B)/E(R_d = A)$ decreases and it becomes more efficient to distribute B when $||B||/||A||$ is between 2.9 and 3.5. When $||B||/||A|$ reaches 3.6, C_J must contact another "hop" of nodes ($k = 4$) so that R_J is sufficiently large to buffer B. Not only that the cost of distributing B increases with the addition of new nodes, but the cost of broadcasting A over the 4-hop neighbourhood is substantially higher than the cost of broadcasting it over the 3-hop neighbourhood. When this happens, it becomes more efficient to distribute A over the smaller R_J ($k = 2$) required to buffer it and broadcast the larger relation B over the smaller region. In general, the cost of Step 6a (broadcasting) dominates by a large margin the costs for Steps 4 and 5a. Thus, distributing the smaller relation and broadcasting the larger one over the join region R_J performs better for most ratios than broadcasting the smaller relation. The reason is that the cost of broadcasting increases quadratically as the size of R_J increases (see Equation 4.9). It is only when the size of R_J must be increased to accommodate a larger relation that the processing cost increases drastically, causing a sharp change in the cost ratio.

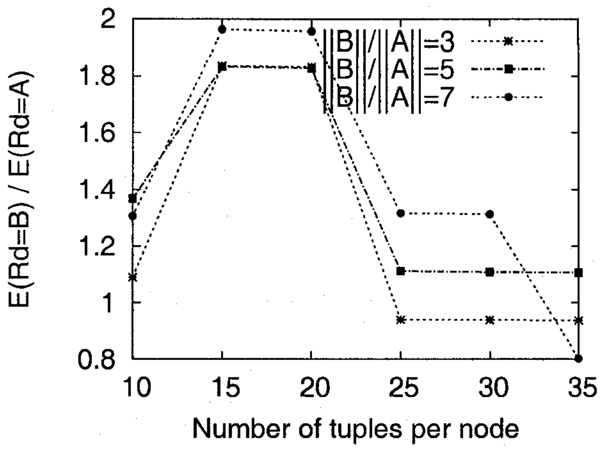


Figure 4.15: Energy cost ratio when $R_d = A$ and $R_d = B$ for variations in the size of available memory at the nodes in R_J

Figure 4.15 shows the relative performance of the two alternatives for distributing the join relations when the size of allocated buffer space at nodes (B_s) varies. The variation of B_s affects the size of the join region R_J (through the number of hops k required to reach sufficient nodes) and thus the performance of the two processing alternatives. We show the relative costs for three ratios between the sizes of the join relations. Note that when the two relations are equal in size ($||B||/||A|| = 1$), the processing costs of the two alternatives are equal as well. The relative performance of the alternatives has a similar trend for the three ratios of the relation sizes and, thus, we discuss in detail the behaviour for $||B||/||A|| = 3$. Consistent to the results shown in Figure 4.13, distributing the smaller relation A and broadcasting the larger relation B is most efficient for more buffer sizes (B_s) due to the large weight of the cost of broadcasting in the total cost. The exception is, again, when the size of R_J is modified (through k). When the number of tuples that can be stored at a node (B_s/T_s) increases from 20 to 25, the number of hops k required for distributing relation B over the nodes in R_J decreases from 4 to 3, while it stays constant for distributing relation A ($k = 2$). Thus, the cost of the alternative that distributes B decreases substantially due to the much reduced cost of broadcasting A over the smaller number of nodes. At the same time, the cost of the alternative distributing A over R_J varies only slightly as more of A's tuples can be stored closer to C_J (but still up to $k = 2$ hops away from C_J), causing a sharp change in the relative performance of the two solutions. When B_s/T_s varies from 15 to 20 and 25 to 35, the number of hops required for distributing R_d (be it A or B) does not change, and, thus, there is only a small variation in the relative performance of the two alternatives caused by the slightly lower cost of distributing R_d. Similar trends can be observed for $||B||/||A|| = 5$ and $||B||/||A|| = 7$.

Another parameter affecting the relative performance of the two alternatives is the network density. The variation of network density directly affects the size of the join area R_J as nodes farther (in terms of hop-count), or closer, to C_J are required for storing the relation that is distributed when the density decreases, respectively increases. Figure 4.16 shows the effect of the network density on the relative performance of the two solutions. We evaluate again the relative cost for three ratios of join relation sizes. As the relative performance of the two alternatives shows similar trends for all three ratios, we focus our discussion on

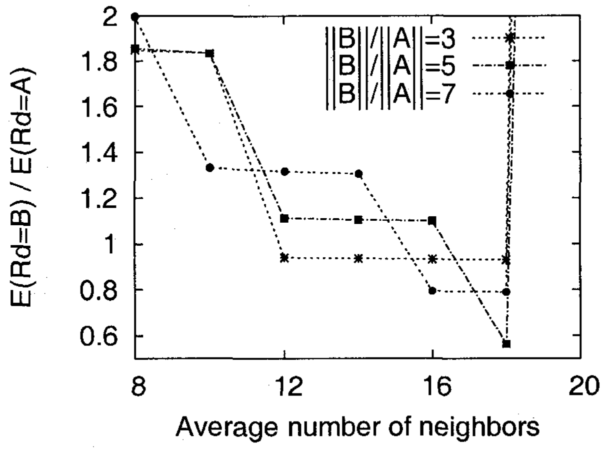


Figure 4.16: Energy cost ratio when $R_d = A$ and $R_d = B$ for variations in the network density

$||B||/||A|| = 3$. When the average number of neighbours (N_n^1) is between 8 and 18, only nodes up to $k = 2$ hops are required to buffer relation A in R_J. This effectively means that the cost of the alternative distributing A in R_J varies only slightly, while the alternative that distributes B decreases its cost at a faster rate as k decreases from 4 to 3 when the N_n^1 increases from 10 to 12. When the number of neighbours is 20, the network is sufficiently dense so that C_J uses only its 1-hop neighbours ($k = 1$) to store A. As such, the cost of broadcasting B decreases sharply, causing a similar decrease in the overall processing cost. Note that for $k = 1$, only one node (C_J) need to broadcast relation R_b. This is a sharp decrease in the number of broadcasts since for $N_n^1 = 18$ and $k = 2$ the number of nodes broadcasting R_b is 19 (C_J plus 18 1-hop neighbours). On the other hand, the cost of the alternative distributing B varies only slightly when the N_n^1 increases from 18 to 20. Therefore, there is a sharp change in the relative performance of the two solutions, and it becomes substantially more efficient to distribute A than to distribute B when $N_n^1 = 20$.

4.5.4 Summary

In this section we have discussed in details a technique (DIJ) for processing the theta-join operator in a sensor network. The strength of the technique is that we take into account the memory available at the sensors nodes and the synchronization of the data flow. Both issues have been overlooked or simplified in the existing literature as we shall discuss in Section 4.7. We have also developed a cost model that allows our technique to be optimized with respect to the size of the join relations and the amount of available memory at the nodes processing the join. Another important aspect is that our technique is general in the sense that it can be re-used in the core of other previously proposed join solutions for relaxing their assumptions on memory availability at nodes. Finally, we studied the technique's behaviour through the cost models under several combinations of query and network parameters. We have shown that the size of the region over which the join is processed (represented by k) has a strong impact on the cost of the processing.

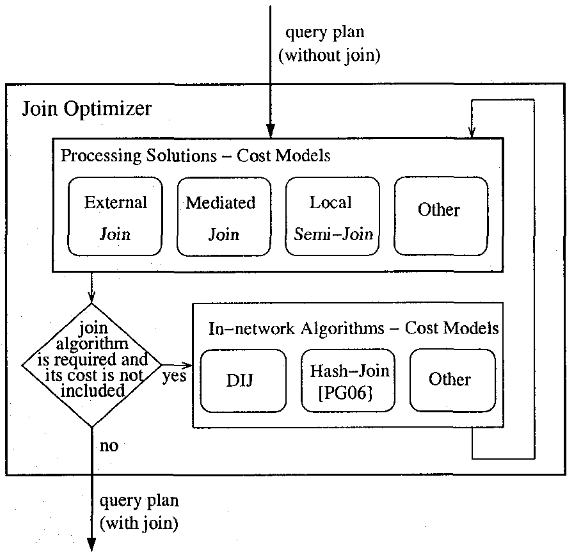


Figure 4.17: A join optimizer for queries in sensor networks

4.6 Selecting a Join Location and a Join Algorithm

In the previous sections we have discussed two problems relevant to processing a join query in a sensor network: *where* should the join be performed (Section 4.3) and *how* to process the join operator in a distributed manner at a network location (Section 4.5). Since these two problems are complementary to each other, we have analyzed them separately. However, processing a join query involves both these issues simultaneously. Thus, we discuss in this section how to combine the solutions to these two issues for efficient processing of join queries.

Given a join query, the query optimizer must find first the location where the join operator should be processed. For that, the optimizer should evaluate the cost of each processing solutions using, for instance, our cost models derived in Section 4.3. If the most efficient processing solution as suggested by the cost models requires in-network processing, the optimizer should find the type of join operator (e.g.: theta- or equi-join) involved in the query. Assuming that the optimizer is aware of several algorithms (such as *DIJ*) for processing each operator type, the next step is for the optimizer to estimate the cost. Once the lowest cost algorithm has been determined, its cost should be added to the cost of the selected processing solution and the best solution should be re-evaluated. The reason is that the cost of the join algorithm plus the cost of the processing solution may prove to be larger than other solutions that do not require any join algorithm (such as the *External Join*). Once the most efficient processing solution is determined, the query optimizer informs the query processor, which initiates the processing. An overview of this process is shown in Figure 4.17.

4.7 Related Work

Research on query processing in sensor networks has mostly focused on processing of selection, unions, grouping and aggregation operators [GM04]. Recently, a few works addressed the processing of join queries.

Bonfils and Bonnet [BB03] consider the problem of processing a correlation operator (i.e., a special join) at a node in the network. The solution starts with a random placement of the operator at a network node. The position is progressively refined by moving the operator to the nodes with lower processing cost during the lifetime of the continuous query. Two important assumptions are that the operator can be fully processed on one node and that the lifetime of the query is sufficiently long to refine the operator position from a random location to an optimal one. An advantage of the refinement is that the operator placements adapts to the change in data during the query lifetime. For short continuous queries their solution would perform much worse than the optimal cost due to the initial random placement and the solution is inadequate for historical queries. The authors focus on the operator's placement problem, assuming that each node that will hold the operator is able to handle the flow and processing of data alone.

Chowdhary and Gupta [CG05] propose an algorithm for performing the *self-join* in-network over a processing region with several sensor nodes participating in the join. The processing algorithm, called distribute-broadcast join, is a form of distributed block-nested loop join, where each node in the join area holds one block of the outside relation while the inside relation is broadcast over the join region. The algorithm is similar in spirit to *DIJ*, but the authors do not investigate the allocation of memory at the nodes in the join region and the synchronized data flow. Different from us, the authors consider a special shape for the join region and argue that this region is optimal. Along the same line, Pandit and Gupta [PG06] propose two algorithms for in-network processing of the range-join operator. One algorithm is a distributed form of a hash-join algorithm, while the other is a distributed form of index-join and uses a B-tree structure distributed at the sensor nodes. Both works [CG05, PG06] consider that the optimal join location is the weighted centroid of the triangle formed by the originator node and the query regions. The centroid has the property that it minimizes the weighted sum of the *squared* distances, and thus it is not optimal.

Yu et al. [YLZ06] investigate the processing of *self-join* queries with *equi-joins* over historical data in sensor networks. In their solution they constructs a synopsis (e.g., a histogram) of each relation involved in the join. The synopsis are transmitted to an intermediate location, where they are used for finding which tuples of the two relations will certainly not join. This information is returned to the sensor nodes storing the relation, which will use it to select only the join relevant tuples to participate in the join. The join is then performed in network at a second intermediate location. The solution performs best when the join selectivity is high and it is closest to our *Mediated Semi-Join*. The join of the synopsis is performed in a square join region whose size is determined based on the size of the synopsis, the network density and the average memory available at the join nodes, which we also use in our approach. When allocating the synopsis to the join partition, they fail to consider the memory available at the individual nodes in their hash-based allocation scheme, which would cause buffer overflows and invalidate the join result. They also assume that nodes have sufficient memory when performing the final join of the filtered tuples.

The *external join* problem where the sensor relation is joined with a relation stored at the user station is studied by Abadi et al. [AML05]. The external relation is basically a relation containing filters to be applied on the sensor tuples. If the external relation is small, it is flooded in the network and the join occurs locally at each node. When the external relation is too large to be stored in the network, the authors propose three techniques (bloom filters, partial joins and cache diffusion) that help filter part of the sensor tuples. Non-filtered tuples are then joined externally after reaching the base station. An intermediate situation is when the external relation fits into a group of nodes. While multiple groups of nodes are formed in REED, this solution is closest to our *Local Join*.

4.8 Summary

Many works have focused on processing queries over the sensor network, but they limited their focus on processing the selection and aggregation operators over the sensor relation R^*. Such queries were mostly filtering the sensor data in-network, with any further processing done off-line. In this chapter we have argued that the join operator allows one to pose important queries on the sensor data. We have also shown that the join operator can be pushed in-network together with the other operators previously studied. We have analyzed several solutions for in-network processing of the join between two relations with respect to the location where the join operation should be performed. We have shown empirically that no join processing solution performs best for all queries. Using our cost models to choose at query time the most efficient processing solution, we are able to reduce the cost of join processing with up to 83% compared to processing every query with the same solution, and also perform within 7% of the optimal processing cost. We have also proposed a technique for processing the theta-join operator in a sensor network. Our technique takes into account the memory available at the sensors nodes and the synchronization of the data flow. Both issues have been overlooked or simplified in the literature. We have developed a cost model that allows our technique to be optimized with respect to the size of the join relations and the amount of available memory at the nodes processing the join.

In this chapter we have investigated the processing of queries with one join operation. An interesting open problem is in-network processing of queries joining multiple sensor relations. The challenge is finding the best order for joining the relations and, for each join, the most energy efficient processing solution. Any decision on what solution to use for processing a particular join operator in the query tree should consider its effect on processing of the other join operators in the tree. In our investigations we have assumed that the query regions are much smaller than the distance between the regions and the originator. As such, we approximated each region by the location of its centroid when building the cost models. For larger query regions or small distances to the originator node such an approximation may not be sufficient. Using more realistic approximations of the query regions and their effect on join processing and the cost model is an open problem.

Chapter 5

In-network Processing of Approximate Queries

5.1 Introduction

In domains such as GIS [BC99], a typical query involves building a map of values for a given area, e.g., "find the temperature for each point of lot X12 at 2pm yesterday." Since it is not practical to have a sensor in each point of the monitored region, one has to settle for some approximated value for those points where a sensor is not present. This leads to the notion of an approximate answer for a query where a map is built with values for each point in the map, but also with a confidence level in the accuracy of the value associated to a map point. In fact, this leads to two maps, one with the requested values and another one with the confidence associated with those values. In a more general case, the values of interest could form a set of maps, for different time granularities, e.g., "find the approximate hourly temperature values for the past 12 hours of lot X12 so long as the reported value has a confidence above 40%". We call this type of query a SpatioTemporal Data Map (STDMAP) query. An important fact to note is that in the same way some points of the queried map are "covered" by one or more sensors, sensor themselves can also be covered by other sensors. In effect this means that some nodes may take advantage of such coverage redundancy and not participate in the query's processing without loss of the answer's quality. We study this problem in the peer-to-peer sensor network environment introduced in Section 1.2.

Our contribution in this chapter is the proposal of three techniques to address the problem of approximate query processing in sensor networks, namely:

1. the AFM technique, which employs parallel flooding of the queried region where each node decides, based on sound criteria, whether it should participate or not in the query's answer;

2. the EFM technique, which is a energy-aware parallel flooding, where nodes decide about their participation considering also the amount of energy they have;

3. the MSM technique, which uses the completion of the queried map itself as a guide to traverse the region's nodes.

All three techniques aim at taking advantage of the redundancy mentioned earlier, i.e., process the query so that not all relevant node need to participate (depending on the required level of confidence). Through extensive experimental evaluation, we show that in-network

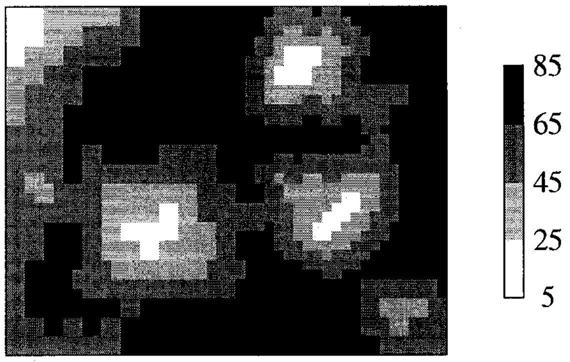


Figure 5.1: Example of a map answer

processing of STDMAP queries reduces by up to twenty times the energy use compared to the typical solution that retrieves the raw sensor measurements and assembles the STDMAP answer off-line.

The remainder of the chapter is organized as follows. Section 5.2 details the STDMAP query. Section 5.3 presents our three algorithms for processing STDMAP queries in the peer-to-peer sensor network environment. The performance of the proposed processing solutions is evaluated in Section 5.4. Section 5.5 discusses research works related to our contribution and Section 5.6 concludes the paper.

5.2 The STDMAP Query

A common representation of information in environmental remote sensing [BC99] is in the form of a map capturing the spatial distribution of data (Figure 5.1), where each map point represents a spatial area and its associated value represents the state of the observed phenomenon in the area corresponding to the point. Query support for such a representation is important for applications where the spatial distribution of data is more important than individual data values. The map representation for sensor network data can be constructed by first collecting the sensor measurements from all sensors located in the region of interest, followed by the construction of the map off-line. However, collecting the measurements from all these sensors (called relevant nodes) may not be necessary, as the measurements from only a *subset* of the relevant nodes may be sufficient to construct the map. This is possible if the answers of some nodes can be approximated by the answers of other nodes. There are two reasons for considering answer approximation. First, it is not practical to have a sensor in each point of the monitored region, and therefore the values used for most of the map points must be approximated[1] using the answers of the sensors located nearby. Second, due to the inherent correlation among the states of physical phenomena at close locations, the measurement of any sensor can be approximated by the measurement of other sensors located nearby with a certain degree of confidence.

Let us consider two locations where we are interested in the state of a monitored phenomenon: location s where a sensor node S exists and location l where there is no sensor.

[1]In this work we consider sensors that observe the state of a monitored phenomenon at the sensor location only. This is different from range sensing (e.g., movement sensing used in tracking [FZG02]), which measures the state of an entity not necessarily located at a sensor's position, but in its vicinity.

S can provide a measurement for the monitored phenomenon only at location s. For location l, we can approximate a state with the measurement for location s, with some degree of confidence. We model the confidence of a node as a function $C(s,l)$, which represents the confidence of the sensor S that a measurement taken at location s is the same at location l. Depending on the monitored phenomenon and the capabilities of the sensing unit, the function $C(s,l)$ could be very simple or highly complex, constant over the lifetime of the sensor node or adaptive to various conditions (e.g., time of day). Naturally, the confidence of S for a location decreases with the distance to that location. The concept of confidence is also used in [DGM$^+$04] to capture uncertainty, but, differently from us, their confidence represents the uncertainty with respect to approximated sensor readings, while ours captures the uncertainty in the validity of an actual sensor reading for a different spatial location than where it was acquired.

The SpatioTemporal Data Map (STDMAP) query supports the map representation of the sensor network data. Assuming a confidence function $C(s,l)$, which is dependent of the sensor network setting but it is query-independent, we denote the query by STDMAP*(sw,tw,ct)*, with the following characteristics:

- The answer of a STDMAP query is a set of map layers representing the approximations of the sensor measurements with confidence above the minimum confidence threshold *ct* for the spatial region *sw*, with each layer corresponding to one time point within the query's time window *tw*. The confidence threshold *ct* represents the minimum confidence that a user is willing to accept in the query answer for the approximation of a value at a map location.

- Each point of a map layer corresponds to an area within the query's spatial window, with its value equal to the approximated state of the monitored phenomenon in the corresponding area. Figure 5.1 shows an example of a map layer.

Note that each sensor node has a confidence in approximating the state of the monitored phenomenon with its measurement for every possible location. Unfortunately, interpolating the measurements of all sensors for each location using their confidences is very difficult, possibly requiring information from every sensor node. Distributed regression is used in [GBT$^+$04] to model spatiotemporal redundancy in sensors' measurements, where the user is responsible for providing the location of kernels and the set of basis functions. This is not feasible for large sensor network deployments. In this chapter we associate with a map point the measurement with the highest confidence among all approximations obtained during query processing, reserving the problem of interpolating the sensors' approximations for future work.

Before going further let us introduce the following definitions which are necessary for the remainder of the chapter:

Definition 1. *Given a confidence function and a confidence threshold, the coverage area of a sensor node is the area around the sensor in which the sensor's data is relevant to the query (i.e., the confidence of the sensor is above the threshold for every point in the area).*

Definition 2. *Assuming the confidence function of a sensor is isotropic*[2] *and stationary, the coverage range c_r of a sensor is the radius of the circle centred at the sensor which forms its coverage area.*

[2]While the variation of physical phenomena may be different on each direction due to the environment, sensor nodes may not able to detect this variation and adjust the confi dence function accordingly.

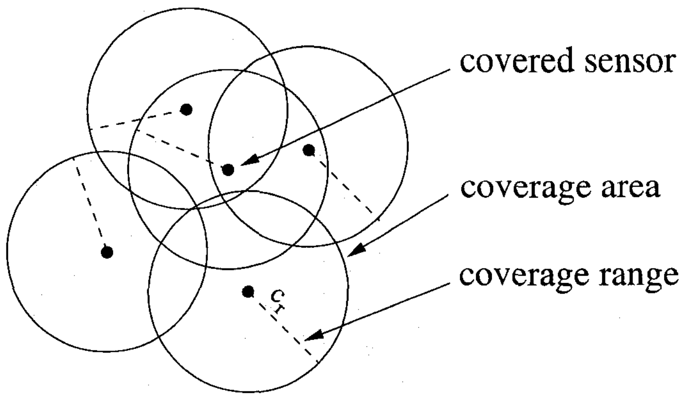


Figure 5.2: The coverage of sensors

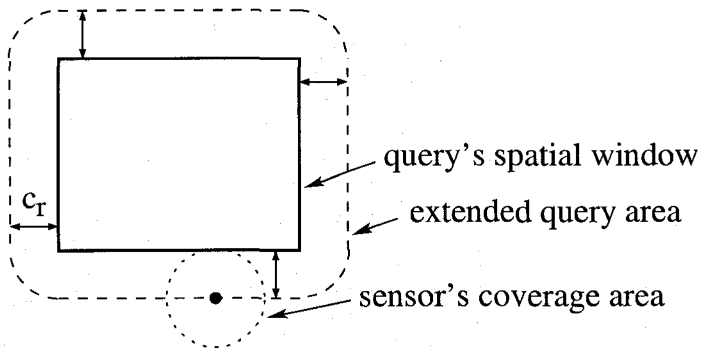


Figure 5.3: Extended query area

As sensors' coverage areas can overlap (dependent on the inter-sensor distance, the confidence function and the confidence threshold), the coverage areas of some sensors may be covered by other sensors (see Figure 5.2 for an example). In this situation, the STDMAP query can be answered using a subset of the relevant sensor nodes, thus saving communication and processing costs. This is possible as the STDMAP query does not require in its answer the measurement with the highest possible confidence, but with a confidence higher than or equal to the confidence threshold *ct*. Note that a sensor's coverage area depends on both the confidence function $C(s, l)$ and the query (via the user specified confidence threshold *ct*). This is different from the sensing area that some range sensor types (i.e. motion sensors) provide, where the sensed area is a characteristic of the sensing device.

5.3 Strategies for Query Processing

Since the coverage areas of the sensors located in the proximity of the query's spatial window may intersect the query window, finding the query answer for a location inside the query window may require contacting nodes located outside the query window.

Definition 3. *The extended query area is the area where the sensor nodes whose coverage areas intersect the query's spatial window can be located.*

Algorithms for processing STDMAP queries must be able to contact the nodes located in the extended query area. Given a confidence function $C(s,l)$ and a confidence threshold *ct*, the extended query area is formed by the extension of the query's spatial window in every direction with the coverage range, as shown in Figure 5.3.

Due to the nature of the environment where the sensors are deployed, it is possible that the assumed measurements of two neighbouring sensors for the same location are inconsistent. This suggests possible obstacles affecting the expected variation of the monitored phenomenon. In this situations nodes could mark the affected areas as not covered by their neighbour that provides the inconsistent approximation. We leave the problem of inconsistent approximations for future work, and we assume in this work that the nodes covering a certain location provide consistent approximations for the location.

5.3.1 Processing Solutions for STDMap Queries

This section presents three algorithms for processing STDMap*(sw,tw,ct)* queries. In large scale sensor deployments, the user is typically interested at a given time in a spatial window (*sw*) of the whole monitored region and a temporal window (*tw*) of the measurements collected by a sensor. We have shown in Chapter 3 that for spatial range queries a two-phase query processing approach is more efficient than the typical network flooding. By forwarding the query to all network nodes, the network flooding contacts many nodes in addition to those that hold the query answers, which increases substantially the energy cost of processing.

Due to its lower cost, we use a similar two-phase approach for processing STDMap queries. We break each processing algorithm into two phases: one for finding a routing path from the query originator node to the query window *sw*, the other for collecting the query answers from the relevant nodes and returning the answers to the originator node. For the first phase, we use a simple greedy approach (presented in Section 3.4.1) to discover a routing path from the query originator node to a node located near the centre of the query's spatial window, called coordinator node. As the proposed algorithms use the same routing algorithm in their first phase, we detail in the following only the second phase for each query processing algorithm.

5.3.2 Aggressive Flood Strategy (AFM)

For its second phase, the Aggressive Flood for STDMap (AFM) algorithm uses parallel flooding to distribute the query to all nodes located within the extended query area. However, not all nodes will contribute to the answer. Each node decides locally if its answer is required by the coordinator node to assemble the STDMap answer. The decision is based on the coverage areas and the states of its neighbour nodes. Each one of a node's neighbours can be in any of three states: OPEN, if the node has no information about its neighbour's state; SEND, if the neighbour has decided that its answer is required; or SKIP, if the neighbour has decided that its answer is not required. The OPEN state is the state assigned by a node to itself and all its neighbours upon receiving the query for the first time. A node decides its final state between SEND or SKIP after receiving broadcasts of the query from one or more of its neighbours, but before sending its own query broadcast. Nodes send their state (SEND or SKIP) along with the query broadcast. Once a node makes a decision, it can filter out the other broadcasts of the same query based on their header. A node decides to send its answer and changes its state to SEND if its coverage area is not fully covered by its neighbours in OPEN or SEND state. Otherwise, the node does not send its answer and changes its state to SKIP, as other nodes will cover its area. A node that decides its answer is required (SEND state) returns its answer to the neighbour it first received the query from. The second phase of the AFM algorithm running in each relevant node is presented in Algorithm 3.

Algorithm 3: AFM Algorithm - Phase 2

```
Input    : Current Node N, N's Neighbour List NB
1  Receive query Q, PNB.state from parent neighbour PNB
2  if N.location not in Q.extArea then STOP
3  Initialize status of all nodes in NB to OPEN state
4  Update status of node PNB in NB to PNB.state
5  Construct coverage N.coverArea using N.location, Q.extArea, Q.ct
6  Check if nodes in NB with SEND or OPEN states cover N.coverArea
7  while N.coverArea is not covered & new broadcasts received do
8      Update status in NB for broadcasting neighbours
9      Check if nodes in NB with SEND or OPEN states cover N.coverArea
10 if N.coverArea is covered then
11     N.state ← SKIP /* N's answers are not required */
12     Broadcast Q, N.state
   else
13     N.state ← SEND /* N's answers are required */
14     Broadcast Q, N.state
15     Return (N.data in Q.tw) to PNB /* return answer */
```

Using AFM, each node receives the query message from several neighbours and sends one broadcast (Algorithm 3, line 12). Depending on the overlap of the coverage areas, only a subset of the relevant nodes will return their answers to the query coordinator node. After the query coordinator gathers the nodes' answers, it constructs the STDMAP answer and returns it to the query originator node over the path discovered in the first phase of query processing. Due to the limited information each node has, if a node is covered with the help of other nodes than its neighbourhood, the overlap is not detected. As the query is forwarded in parallel over all paths, each node is reached over the shortest path from the coordinator. Thus, the answers are returned to the coordinator node over the shortest path by the nodes that decide their answer is required.

When a node changes its state to SKIP, it may force its neighbours in OPEN state to cover its area. Assuming no communication delay latency, these neighbours must have smaller or equal hop-count distances from the coordinator node, and therefore the algorithm allows nodes closer to the coordinator node to skip answering and force their neighbours farther away to cover their area. This behaviour may create in unbalance in nodes' energy levels, leading to the early death of some nodes. The AFM algorithm is an aggressive strategy, in the sense that a node decides to skip answering with only partial knowledge of its neighbours decision, thus forcing other nodes to answer on its behalf. For the AFM algorithm to function correctly, it must be implemented on top of a wireless protocol using control frames and virtual channel sensing (such as those presented in [BDSZ94]). Such protocols ensure that neighbouring nodes do not broadcast their messages at the same time[3].

[3]Before a node is able to reserve the communication channel to broadcast the query and its state, the channel

If such protocol is not used, two neighbours that count on each others coverage could broadcast the query and their decision simultaneously. It would be too late for the two nodes to change and re-broadcast their decisions, as their non-common neighbours reached by the first broadcast may have already processed the original decisions. The correctness of the AFM algorithm is showed in the following lemma:

Lemma 1. *Any point from the query's spatial window covered by sensors will be covered in the final answer using the* AFM *algorithm.*

Proof. We assume the network within the extended query area is connected and every point within the query's spatial window it covered by the coverage areas of one or more sensor nodes. Let us assume there is an area A from the query's spatial window that is covered by each node in the set of nodes $\{N_1, ..., N_k\}$ and only those, and A is not covered in the answer, i.e., none of the nodes $N_1, ..., N_k$ is sending its answer to the coordinator node. Thus all nodes $N_1, ..., N_k$ are in SKIP state. However, there is a node N_j ($j \in 1...k$) such that N_j broadcasts the query and its status the last among its neighbours covering A. Node N_j must have received broadcasts from all its neighbours covering A before deciding its status. Thus node N_j knows that its neighbours covering A are in SKIP state (all neighbours covering A must be among $N_1, ..., N_k$), so node N_j cannot find any neighbour to count on for the coverage of A. Therefore, node N_j must be in SEND state, and area A is covered in the final answer. □

5.3.3 Energy-aware Flood Strategy (EFM)

The Energy-aware Flood for STDMAP (EFM) algorithm uses also a flooding strategy in its second phase, but, differently from the AFM algorithm, its correct execution does not require any specific support from the wireless network protocol used in the sensor network. While AFM is an aggressive strategy, EFM is a passive strategy, where a node makes a decision aware of the state of its neighbours. For the state information of all neighbours to be available, a node must receive all the broadcasts of its neighbours and send two broadcast messages to provide additional information about its status. While all three proposed algorithms try to reduce the energy used during query processing, EFM uses information about the amount of energy nodes have in order to decide which nodes should participate in query answering. Thus, EFM is an energy-aware processing strategy.

Query processing starts when the coordinator node broadcasts the query to its neighbours. A node receiving a query for the first time checks if its coverage area is covered by its neighbours. If it is not covered, the node decides that its answer is required and sets its state to SEND. If the area is covered, the node does not have yet sufficient information about its neighbours to safely decide to skip answering, and therefore sets its own state to OPEN. Next, the node broadcasts the query and its current state.

Once a node has received the query broadcast from all its neighbours, it checks if its neighbours that have decided to answer are covering its area, i.e., if its area is covered by nodes in SEND state. If its area is covered, the node can safely skip answering and it sets its state to SKIP. After this check, each node broadcasts a state update message to its neighbours and waits for the state updates from its neighbours. If a neighbour B of a node A has changed its state from OPEN to SKIP, it means that B is fully covered by its neighbours

may be used by some of its neighbours for their broadcasts, which the node will receive and consider in its decision.

in SEND state, and thus any overlap it has with A is also covered. Consequently, a node can safely skip answering if its area is fully covered by its neighbours in SEND or SKIP states. Such a node changes its state to SKIP, but this information is not exchanged. Not exchanging the information on this status update is safe, as any node that is counting on the coverage of its neighbour in SKIP state remains covered (through transitivity) by the nodes covering its neighbour.

At this point, a node can be in any of the three possible states (SEND, SKIP or OPEN). If a node's state is SEND, the node returns its answer to the neighbour it first received the query from. If its state is SKIP, nothing has to be done as the node's coverage area is covered by other nodes that will answer. If its state is OPEN, the node must decide based on the information about its neighbours whether to send or to skip. To determine correctly whether it has to send its answer, however, it needs to know whether information about its own area that is covered by neighbours that are also in OPEN state will be sent (by the neighbours directly or by other nodes covering the neighbours). This is a potential problem as the covering relation is symmetric[4]. Two nodes in OPEN state which partially cover each others areas have to independently make a consistent decision. In particular we must avoid that two nodes decide to skip and no other node will send information about their overlapping area. The information about the current state of its neighbours is not sufficient to take a consistent decision.

We use the amount of energy left in nodes to determine which one of a pair of nodes in OPEN state with overlapping coverage areas will ensure the coverage of the overlap (the energy information can be exchanged during the broadcasting of the state update message). For each pair, the node with more energy will be responsible for the coverage of the overlapping area. Thus, a node A in OPEN state with more energy will not count on a neighbour B in OPEN state with less energy to cover its area, and vice versa, B will count on the coverage by A, i.e., B will consider A to be in SEND state. This policy guarantees that for each pair of neighbouring nodes in OPEN state with overlapping areas only one node will assume that the overlap is covered by the other node. If two nodes have the same amount of energy left, they can use their unique node identifier as a tie-breaker.

After each node updates its local representation of the state of the OPEN neighbours according to the above policy, it checks again if its area is covered by neighbours in SEND or SKIP states. If its area is not covered, the node will assume its answer is required and sends it to the coordinator node. Otherwise, if its area is covered, it skips answering. Note that if a node A skips answering, this does not affect a neighbour B with less energy that counts on A for covering their overlap, since other nodes will answer for A's area. The pseudo-code for the second phase of the EFM algorithm is listed in Algorithm 4, and the correctness is shown in Lemma 2.

Lemma 2. *Any point from the query's spatial window covered by sensors will be covered in the final answer using the* EFM *algorithm.*

Proof. We assume the network within the extended query area is connected and every point within the query's spatial window it covered by the coverage areas of one or more sensor nodes. Let us assume there is an area A from the query's spatial window that is covered by each node in the set of nodes $\{N_1, ..., N_k\}$ and only those, and A is not covered in the answer, i.e., none of the nodes $N_1, ..., N_k$ is sending its answer to the coordinator node.

[4]If the coverage areas of two neighbours overlap, each node may consider the other node covering the overlapping area.

Algorithm 4: EFM Algorithm - Phase 2

```
   Input    : Current Node N, N's NeighbourList NB
1  Receive query Q, PNB.state from ParentNeighbour PNB
2  if N.location not in Q.extArea then STOP
3  NBE ← $\emptyset$ /* list of neighbours in Q's extended area */
4  foreach node $N_i$ in NB do
5      if $N_i$ in Q.extArea then
6          Add $N_i$ to NBE
7  Initialize status of all NBE nodes to OPEN state
8  Update status of node PNB in NBE to PNB.state
9  Construct coverage N.coverArea using N.location, Q.extArea, Q.ct
10 Check if all NBE nodes cover N.coverArea
11 if N.coverArea is covered then
12     N.state ← OPEN
   else
13     N.state ← SEND
14 Broadcast Q, N.state /* SEND|OPEN state */
15 Wait for broadcasts of Q, status from all NBE nodes
16 Update status for all NBE nodes based on broadcasts
17 Check if NBE nodes with SEND state cover N.coverArea
18 if N.coverArea is covered then
19     N.state ← SKIP
20 Broadcast N.state, N.energy /* SEND|OPEN|SKIP state */
21 Wait for broadcasts of status, energy from all NBE nodes
22 Update status, energy for all NBE nodes
23 foreach node $N_i$ in NBE do
24     if $N_i$.state = OPEN & $N_i$.energy > N.energy then
25         Change status of $N_i$ in NBE to SEND state
26 Check if NBE nodes in SEND or SKIP state cover N.coverArea
27 if N.coverArea is not covered then
28     Return (N.data in Q.tw) to PNB
```

Thus nodes $N_1, ..., N_k$ are in SKIP state. However, there is $j \in 1...k$ such that N_j has maximal energy among $N_1, ..., N_k$. The node N_j cannot skip. There is no neighbour N^* of N_j covering A having higher energy so that node N_j could count on N^* to cover A (if such a node would exist, it would be among $N_1, ..., N_k$, and N_j would not be the node with the highest energy in that set). Therefore, node N_j must be in SEND state, and area A is covered in the final answer. □

5.3.4 Map-guided Search Strategy (MSM)

The third solution we propose for processing the STDMAP queries is the Map-guided Search (MSM) algorithm. Differently from the previous two algorithms, MSM uses a partial query answer for guiding the query processing. The algorithm finds the STDMAP answer by forwarding the query using the current partial STDMAP answer. At each step, the current sensor node selects for query forwarding its neighbour that can provide values with confidence above the confidence threshold *ct* for the largest map area without assigned values. If a neighbour of the current node has a coverage area where every location has already been covered with confidence higher than *ct*, then the neighbour is not contacted at all during query processing.

The query message received by a node contains both the query and the partial STDMAP coverage map as obtained so far. The node first processes the query by adding its answers to the STDMAP coverage map. Before forwarding the query to any neighbour, the neighbours that are not located within the extended query area are discarded from the list of candidate neighbours. The algorithm goes iteratively through the candidate neighbours and forwards the new partial STDMAP coverage map to the best of them until either the query area is fully covered or there are no more candidate neighbours. The best candidate neighbour is the one that covers the largest area that does not have associated approximated measurement values. The query answering is considered complete when STDMAP's map layers have associated approximated values for every location. The correctness of MSM algorithm is shown in the following lemma:

Lemma 3. *Any area from the query's spatial window covered by sensors will be covered in the final answer using the* MSM *algorithm.*

Proof. By construction of the MSM algorithm. We assume the network within the extended query area is connected and every area within the query's spatial window it covered by the coverage areas of one or more sensor nodes. The MSM algorithm uses sequential depth-first forwarding to contact nodes and therefore all nodes within the query extended area are contacted until all of the query area is covered. □

Since the query forwarding is done in a sequential depth-first-like manner, only one node is processing the query at each point in time. This process ensures that only one copy of the partial STDMAP coverage map is available in the network, and each node processing the query is aware of the contribution to the STDMAP coverage map of the nodes previously involved in the query answering. While this strategy is likely to result in slower query answering for each individual query than the previous two algorithms, it facilitates several queries being processed simultaneously by the same group of relevant sensors. The second phase of MSM algorithms is listed in Algorithm 5, where the *gain* function evaluates the size of the map area that is covered by a node and it doesn't have any approximated measurement associated yet.

5.3.5 Coping with Sensor Failure

Quite often sensor networks operate in harsh environments, where permanent or transient sensor failures are expected. In addition, the energy source of sensor nodes is limited, leading to their failure after a period of operation. Regardless of the reason, node failures should be rare events during most of the network lifetime, otherwise the use of sensor network technology is not practical. In the following we discuss solutions for dealing with node failures during the processing of STDMAP queries.

We first consider the MSM algorithm, where the communication takes place only between two nodes at a time. When a node tries to send the query to one of its neighbours, the neighbour should acknowledge receiving the query message as part of the network protocol.

Algorithm 5: MSM Algorithm - Phase 2

```
   Input    : Current Node N, N's NeighbourList NB
1  Receive query $Q$, map $M$ from ParentNeighbour $PNB$
2  Update $M$ using $N. location, N.data and Q$
3  $CNB = \emptyset$ /* candidate neighbours for forwarding */
4  foreach $node\ N_i$ in $NB$ do
5      if $N_i.location$ in $Q.extArea$ then
6          Add $N_i$ to $CNB$
7  while $CNB$ not empty & $M$ not full do
8      $BN = \emptyset$ /* best neighbour */
9      foreach $node\ N_i$ in $CNB$ do
10         if $gain(N_i, M) = 0$ then
11             Remove $N_i$ from $CNB$
12         if $gain(N_i, M) > gain(BN,M)$ then
13             $BN \leftarrow N_i$
14     if $BN \neq \emptyset$ then
15         Send $Q$, $M$ to $BN$
16         Wait for $M$ from $BN$
17         Remove $BN$ from $CNB$
18 Return map $M$ to $PNB$
```

If the neighbour does not confirm receiving the query, the node assumes that its neighbour has failed and selects another neighbour for query forwarding. If the neighbour acknowledges receiving the query, the node periodically checks the availability of its neighbour until an answer is returned. If the neighbour appears unavailable during several successive checks, the node assumes its neighbour has failed and selects another neighbour for query forwarding. This solution is consistent with the sensor network environment we assumed (in the sense that nodes are only aware of their neighbours) and it guarantees that the query answer will be found[5]. If a node fails during the first phase of query processing, the failure can be treated similarly, since the greedy solution we use in the first phase of all three algorithms also uses communication between two nodes at a time.

When flooding is used for forwarding the query (in the second phase of AFM and EFM algorithms), node failures cannot be addressed locally in the neighbourhood of the failed nodes due the the parallel nature of the query processing. Node failures could be detected by using time-outs when waiting for replies from the neighbours in the *neighbour list* or periodic heartbeats notification messages [SBLC04]. In this case, nodes detecting a failure should report the location of the failed node to the coordinator node. If the coordinator node does not receive any answers to the STDMAP query for the regions where the failed nodes were located, it is possible that some of the neighbours of the failed nodes may have counted on the coverage of the failed nodes. Thus, the coordinator will restart the query processing in the query area. An optimization is for the coordinator node to restart the

[5]We assume that the network density remains high despite node failures. If the sensor network density decreases substantially, the network graph may become disconnected, making the network unusable.

Parameter	Default Value
Average number of neighbours (N_n)	15 ($N = 2050$)
Spatial window (*sw*)	4% (of region)
Temporal window (*tw*)	60 measurements
Confidence threshold (*ct*)	0.40 and 0.80
Size of query message	256 bits
Size of a measurement tuple	64 bits

Table 5.1: Parameters of query and sensor network

query processing only for the affected query area. coordinator node will learn about other

In both situations discussed above, it may happen that a neighbour has forwarded the query before failing and the query is already being processed in the network. Forwarding the query to a different neighbour or restarting the query processing at the coordinator node would cause the query to be processed twice, or even multiple times if several nodes fail after forwarding the query. This may be not desirable given the limited energy resources that sensors have. Coping with nodes failure may require trading energy efficiency for increased robustness of query processing against failures. Finding techniques to cope with sensor failure and integrating them with the proposed algorithms is an open problem.

5.4 Experimental Evaluation

We implemented a sensor network simulator to study the performance of the presented algorithms. The placement of the sensor nodes follows a uniform distribution over a two dimensional region. We represent a STDMAP query by the coordinates of a spatial window (*sw*), a temporal window (*tw*), a confidence threshold (*ct*) and a query identifier (*qID*). The query's spatial window covers 4% of the monitored region (that is 20% on each spatial coordinate), unless otherwise noted. The query's temporal range covers 60 measurements (one hour of sensor measurements for a measurement rate of one per minute, or two months of measurements for one daily measurement). For the answer of the STDMAP query, each cell (i.e., pixel) of a map layer represents 1 m^2 square area in the query's spatial window. If a different map granularity is required, the granularity could be added as a parameter to the STDMAP query. The query originator and the centre of the query's spatial window are uniformly distributed over the monitored region. The query and sensor network parameters and their default values used in our experimental evaluation are summarized in Tables 1.2 and 5.1. Similar to other works (e.g., [RKY$^+$02]), in our sensor network simulator the message delivery is instantaneous and error-free between nodes communicating directly.

While our algorithms are general with respect to the confidence function $C(s,l)$ used by the sensor nodes, an explicit confidence function is required in the evaluation. We use the Gaussian distribution function $C(s,l) = e^{\frac{-d(s,l)}{2*\sigma^2}}$ to model the confidence function, where $d(s,l)$ represents the Euclidean distance between the sensor node located at s and a location l in the network area and σ^2 is the variance of the function. Gaussian functions have been used before [DGM$^+$04] to capture the behaviour of physical phenomena and the correlation in sensor measurements. Using this function, the confidence of a sensor is high in its proximity and decreases rapidly with increasing distance. Given a confidence function $C(s,l)$ and a confidence threshold *ct*, a sensor's coverage area is determined by the confidence range c_r (see Section 5.2). When c_r is smaller than half of the wireless range

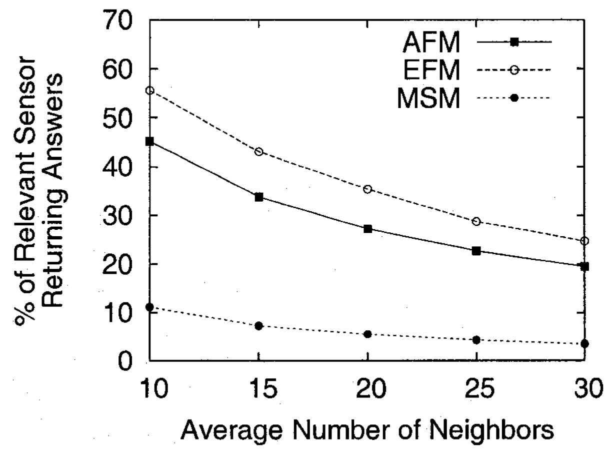
W, the sensors that cover a sensor's area are among its neighbours. Thus, the coverage can be checked easily at the node. When c_r is larger than half of the wireless range W, the sensors that cover a sensor's area may be located outside its wireless range. Thus, a node cannot find out if it is covered by checking only its neighbours. To capture both situations in our evaluation, we use $\sigma = 50$ for the function $C(s, l)$ in combination with two confidence thresholds: $ct = 0.4$ and $ct = 0.8$. When $ct = 0.4$ we have that $c_r \approx 44\ m > \frac{W}{2}$, and when $ct = 0.8$ we have that $c_r \approx 22\ m < \frac{W}{2}$.

We compare the performance of the algorithms using two metrics. The first metric evaluates the algorithms for their capability to reduce the number of relevant sensors needed to answer the STDMAP query. Finding a smaller number of nodes to answer the query may increase the energy cost of processing due to an increase in the communication cost for the control messages. Thus, we use the total energy used during query processing as our second metric. Since we are interested in energy-efficient query processing, we are mainly interested in the energy metric, but the first metric allows us to better understand the behaviour and trade-offs of the algorithms. In our experiments we only measure the energy used to transmit and receive messages during query processing, which includes the messages used for query forwarding, for returning the answers and for status updates.

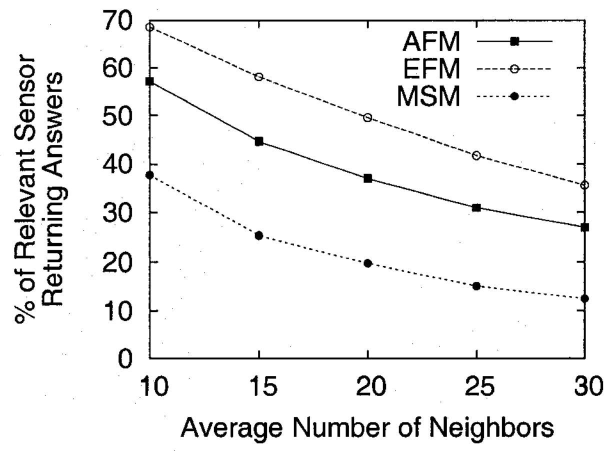
To capture the relative performance of our algorithms against the typical query processing solution, we implemented a simple network flooding with spatiotemporal constrains, called STF. In STF, the query originator node sends the query to all the sensor nodes, but only those located in the extended query area answer the query, returning the raw sensor measurements collected within the query's time window to the originator. Once the measurements reach the originator node (or the user station), the map answer to the STDMAP query can be constructed off-line. Note that in STF all query answers are returned to the originator node over the shortest paths. In the case of our solutions, there are two alternatives for storing the partial answers to the STDMAP query. One alternative is to use the raw measurements as for STF, and construct the STDMAP answer off-line. Another alternative is to take advantage of the map representation in the network and store the partial answers as image maps. Storing the partial answers as images may be more size-efficient (and thus energy-efficient due to the smaller message sizes) than storing the raw measurements if the images use a compressed format as most image formats do. Investigating the trade-off between these two storage formats is an open problem. In our evaluation our algorithms also store the map layers as the raw measurements from which the layers can be constructed for consistency with STF.

5.4.1 Impact of Network Density

Figure 5.4 shows the percentage of relevant nodes that answered the query for each algorithm. Note that for the STDMAP query the relevant nodes are the nodes located within the extended query area. The STF method retrieves the raw measurements from all the relevant nodes (i.e., 100%), and therefore it is not shown in the graphs. The AFM and EFM methods forward the query to all relevant nodes, with only some of these nodes answering the query. The MSM algorithm contacts only the nodes that are used in answering the query. As the number of neighbours increases, all algorithms except STF select a smaller percentage of the relevant nodes to cover the query window. In the case of MSM, each node has a larger set of neighbours to choose from for the next step of the algorithm, which leads to a better selection of the nodes used for covering the query area. For the AFM and EFM algorithms, a larger set of neighbours increases the probability that a node's confidence area is



(a) ct = 0.40

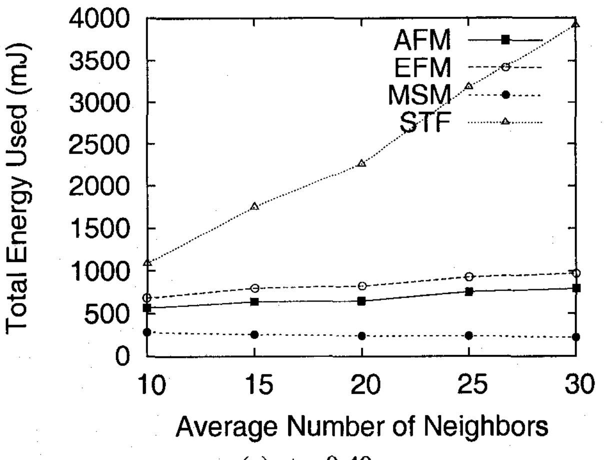


(b) ct = 0.80

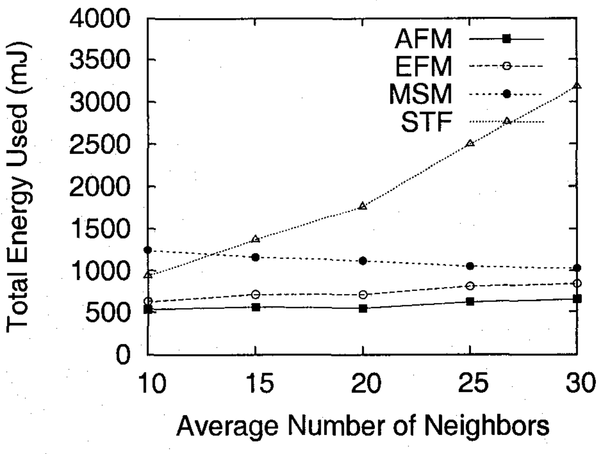
Figure 5.4: The impact of network density on the percentage of relevant nodes that answer the query (from the total number of relevant nodes)

covered, which reflects on the lower percentage of nodes that answer the query. The aggressive strategy of the AFM algorithm leads to a smaller percentage of nodes that answer the query compared to the energy-aware solution used in EFM. The increase in the confidence threshold reduces the coverage range, which has a double effect on the query processing performance: both the extended query area and each sensor's confidence area are smaller. A smaller extended query area contains a smaller number of relevant nodes, but, on the other hand, the smaller coverage areas force more sensors to answer the query in order to cover the query area. Overall, the increase in confidence threshold forces a larger portion of the relevant sensors to answer the query in all our algorithms. This effect is amplified in the MSM algorithm because of the reduced overlap among the sensors that MSM uses for query processing. For the AFM and EFM algorithms, the overlap of sensors' coverage areas is high for $ct = 0.4$, which allows these methods to have a smaller increase than MSM in the number of relevant sensors used for answering the query when $ct = 0.8$.

Figure 5.5 shows the variation of the total energy used for processing a query when the



(a) ct = 0.40

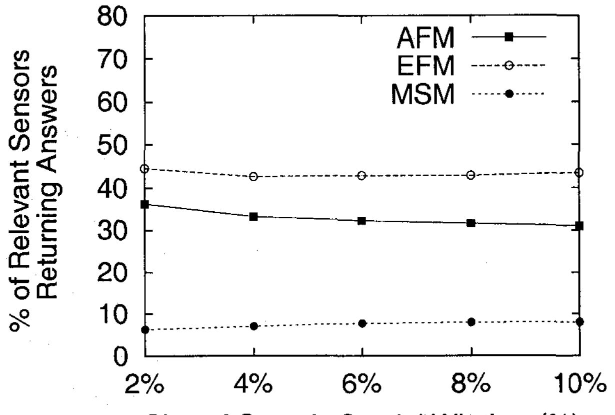


(b) ct = 0.80

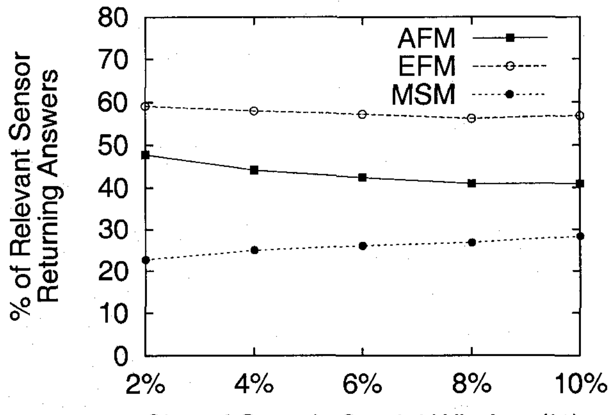
Figure 5.5: The impact of network density on the total energy used for processing the query

network density increases. The STF algorithm is the most affected due to the linear increase in the number of relevant sensors that return answers with the increase in density. Our algorithms use only a few more relevant sensors to answer the STDMAP query for denser networks, which reflects in their slow increase in the energy usage. The EFM algorithm uses two floods of the query window, which causes a slightly higher energy increase than for the AFM algorithm. On the other hand, the AFM algorithm forces nodes located farther away from the coordinator node to answer, which increases its energy cost for gathering the answers. Nevertheless, AFM has the lowest energy cost among the investigated algorithms.

The MSM algorithm uses less nodes than the AFM and EFM algorithms to answer the query, which is reflected on the its lower energy costs when $ct = 0.4$. When nodes' coverage areas are small ($ct = 0.8$) and the network density is low, MSM contacts a large percentage of the relevant nodes. This leads to a high energy cost for MSM, even higher than STF. There are two reasons: first, the cost of transferring the STDMAP partial answer to every contacted node in order to keep track of the already covered area is not negligible; second, MSM uses a depth-first strategy for query forwarding resulting in a very tall and



(a) ct = 0.40



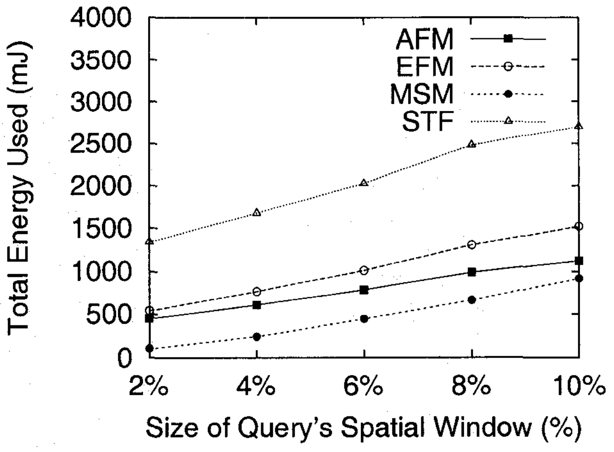
(b) ct = 0.80

Figure 5.6: The impact of the query region size on the percentage of relevant nodes that answer the query (from the total number of relevant nodes)

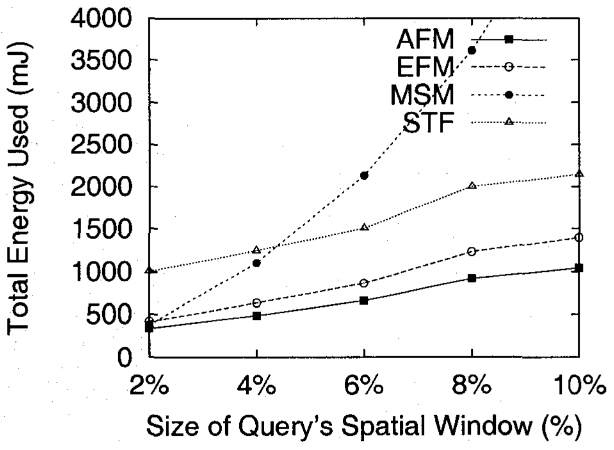
narrow tree over which the nodes communicate, and thus the STDMAP's partial answers are returned to the coordinator node over longer paths than in AFM and AFM. Overall, the AFM and EFM algorithms are the least affected by the increase in network density. While MSM uses the least energy for low confidence thresholds (up to twenty times less than STF) and the least number of sensors to answer the STDMAP query, it is more sensitive to the network density, using more energy than STF for a combination of low density and high confidence threshold.

5.4.2 Impact of Size of Query Region

In the second set of experiments, we kept all other parameters constant, while varying the size of the query's spatial window between 2 and 10 percent of the size of the monitored region. Figure 5.6 shows the effect of query size on the percentage of relevant sensors that answer the STDMAP query. The AFM algorithm takes advantage of the increase in the



(a) ct = 0.40



(b) ct = 0.80

Figure 5.7: The impact of query region size on the total energy used for processing the query

query size and forces the nodes farther away to cover the nodes closer to the query, which leads to a slight decrease in the percentage of relevant nodes that answer the query. The EFM algorithm uses each node's neighbourhood to decide which nodes should answer, and therefore it is not affected by the size of the query area and uses about the same percentage of relevant nodes to answer the query for all query sizes. In the case of MSM, larger query sizes are more difficult to cover efficiently, and thus the percentage of relevant nodes that answer the query increases with the query size. When the confidence threshold is higher, all algorithms except STF use a larger set of sensors to cover the query area since the area that each sensor covers is smaller. Consistent with our observation when investigating the effect of network density, the higher confidence threshold leads to a larger relative increase in the percentage of relevant nodes used by the MSM algorithms compared to the AFM and EFM algorithms.

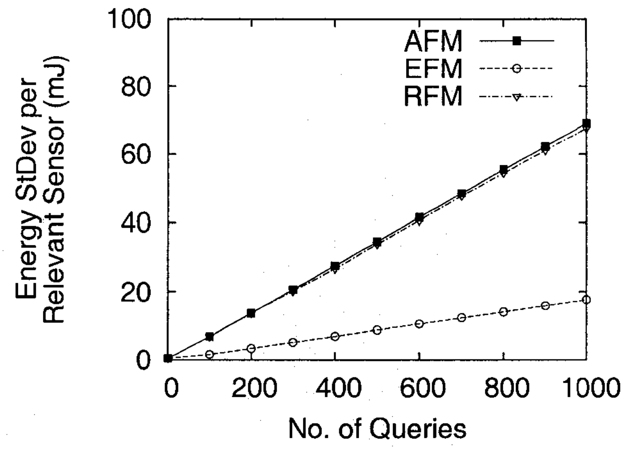
Figure 5.7 shows the total energy used during query processing for the investigated algorithms when the size of the query's spatial window is varied. The MSM algorithm is affected the most by the increase in query size due to the the larger number of nodes that it contacts and the increased height of the tree over whose paths these nodes are contacted and the answers are returned. The increase in the query size also leads to an increase in the energy used by the flood-based algorithms as the number of relevant nodes increases and more nodes must answer the query to cover the larger query area. In addition, the EFM algorithm uses two floods over the relevant sensors for updating the sensors' status, which causes a larger increase in its energy usage compared to the AFM algorithm for larger query areas. AFM and EFM are the least sensitive among the four algorithms to the confidence threshold *ct*. That reason is that they do not take full advantage of large coverage range when $ct = 0.4$ as MSM does. STF is more sensitive to the variation of *ct* as the variation in the extended query area combined with the variation of the query region has a substantial impact on the number of nodes that will answer the query.

Overall, the MSM uses the least energy for a combination of low confidence threshold and small query region, but its costs increases sharply with the increase in the size of the query region (Figure 5.7(b)). The AFM and EFM algorithms behave better than MSM, the increase in the query area causing a smaller increase in their energy costs. In addition, they use two to five times less energy than STF for processing the STDMAP queries.

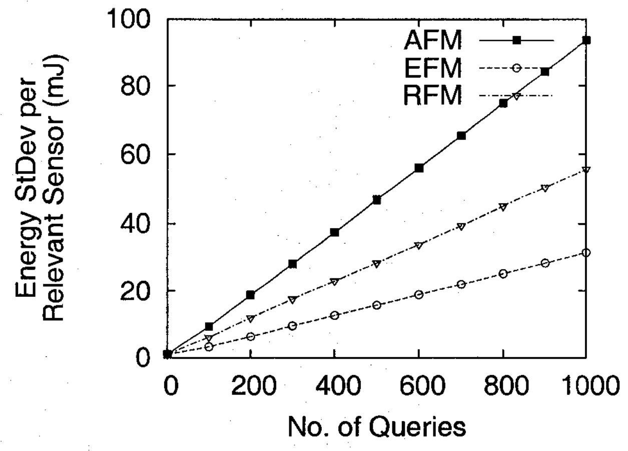
5.4.3 Impact of Non-Uniform Query Distribution

To test out intuition that the EFM algorithm produces a more balanced energy use at the nodes within the extended query region than the AFM algorithm, we compared the effect of the two methods on the nodes' energy levels over several executions of the same query. Further on, to check the effect of the energy-based tie-braking used in EFM, we also compared it against a similar technique, called RFM, that uses a random tie-braking. We fixed the position and size of the query's spatial window while we allowed the originator node to be randomly selected among the network's nodes. We only measured the energy used by the three algorithms for collecting the answers in their second phase. Before initiation the query processing, we set all nodes with similar energy levels. After each set of 100 executions of the query, we calculated the average energy left in the nodes within the extended query area, and the standard deviation of energy from the average for the same nodes. We use the standard deviation to evaluate the energy balance among the sensor nodes.

Figure 5.8 shows the standard deviation of energy for the nodes within the extended query area. As more queries are processed, EFM produces a more balanced energy use compared to AFM and RFM. This benefit of EFM is likely to extend the quality of the query processing in the long-run since more nodes will be available for a longer period. AFM forces some nodes to use more energy than their neighbours, which leads to their early failure, likely creating gaps in the network's coverage. When RFM is used, both nodes in each pair of nodes in the OPEN state covering each other have the same chance of answering the query. While the random tie-braking policy may ensure fairness in terms of what node is responsible of covering a certain area, it does not ensure fairness in terms of overall lifetime as EFM does since one of the nodes may participate more in other processing tasks as well such as answer routing. With the increase in the confidence threshold *ct*, the difference in the nodes' energy levels increases due to the increase in the extended query area, which affects both EFM and AFM. On the other hand, the increase diminishes the effect of the tie-braking policy used in EFM and RFM on the nodes' overall energy levels, leading to a smaller difference between the energy variation at nodes produced by EFM and RFM. Overall, the better energy balance of EFM is likely to extend the quality of



(a) ct = 0.40



(b) ct = 0.80

Figure 5.8: The impact of the distribution of the queries on the energy variation among relevant nodes

query processing in the long-term since more nodes will be available, as well as leading to an increased network lifetime.

5.5 Related Work

Query processing with approximate answering in sensor networks has raised much interest from the research community due to its potential to substantially reduce the query processing costs. In [DGM$^+$04] the query answers are estimated using a statistical model for the sensors' readings, where the model captures the redundancy and correlation in sensor measurements. The sensors are interrogated just to help refine the model when the uncertainty is high, which reduces substantially the query processing costs. Sharaf et al. [SBLC03] exploit the temporal redundancy in a sequence of sensor readings to reduce the energy cost of aggregation during query processing. To improve the fault tolerance of query processing for aggregations, duplicate insensitive sketches are used in [CLKB04] to produce accurate approximations of the aggregate answers. In [DKR04], the authors exploit the correlation and temporal redundancy among the readings of each sensor to compress the short-term historical measurements. Once compressed, the measurements are transmitted to a base station for long-term storage. Caching the sensors' readings is used in [DNGS03] to reduce the cost of retrieving the sensor data, with the users specifying their tolerance for stale data in the query. This chapter complements the previous works in three directions: (1) we investigate the processing of STDMAP queries over the historical sensor data stored at the nodes, (2) we study this problem in a peer-to-peer sensor network, (3) we exploit the spatial redundancy of sensor readings created by dense sensor network deployments, which allows us to use only a subset of nodes to answer the query.

Recently, Xue et al. [XLCL06] study event detection based on contour maps. Similar to our map answer to the STDMAP query, a contour map displays the distribution of attribute values over the network. Differently from us, the map does not have associated a layer to express the confidence in the validity of a map value. In their work, the authors assume a grid on top of the network with at most one node per grid cell. For the cells without nodes, their values in the contour map is interpolated from the values of the neighbouring cells. Differently from us, the proposed scheme applies to continuous queries and takes advantage of the temporal correlation in sensors' measurements by using an incremental map update during the lifetime of the query.

5.6 Summary

In this chapter we have introduced the STDMAP query which exploits the redundancy of sensor measurements on the spatial dimension. We proposed three strategies for processing STDMAP queries (MSM, AFM and EFM) in a peer-to-peer sensor network environment. We showed that the STDMAP query can be effectively answered by only a subset of the relevant nodes, using in some cases less than 10% of the relevant nodes to answer the query.

In an extensive experimental evaluation, we studied the performance of the proposed algorithms under several conditions. The MSM algorithm is best suited for queries where the coverage range c_r is larger than half the wireless range. In this case MSM answers the STDMAP query using a smaller subset of nodes than both AFM and EFM and has a lower processing cost. The AFM algorithm is the most energy-efficient for most scenarios, closely followed by EFM. While the AFM algorithm requires support from the protocol layer, the EFM algorithm is independent of the underlying protocol, which recommends it for most applications. A secondary benefit of the EFM algorithm is that it provides a balanced energy use at the sensor nodes, an important advantage in applications where the queries are not uniformly distributed. In all our experiments EFM has shown low energy usage and consistent performance with respect to various network and query parameters, and therefore we also recommend it for processing queries with similar characteristics to the STDMAP query.

Our investigations have revealed several issues that require further analysis. The harsh environment where sensor networks typically operate raises the issue of sensors' transient or permanent failure. The trade-off between the robustness and energy-efficiency of query processing in the case of sensor failures requires further attention. The natural medium may also cause measurement inconsistencies among sensors and solutions for handling them are in our focus. In this paper we considered as the most reliable approximation the one with the highest confidence. Combining the approximations of several sensors based on their confidence is an open problem that has potential for improving the quality of approximations. A possible solution is the use of a weighted scheme based on a node's Voronoi cell such as the one proposed in [SS06] for spatial aggregations.

Chapter 6

Conclusions

While the technological advances have lead to sensors with reduced sizes and increased capabilities, the sensor data management is still in its incipient stages. Different from the traditional databases, the time efficiency of query processing is no longer the main optimization goal, the focus moving toward energy efficiency or a combination of both time and energy. The challenges are multiple due to the distributed nature of query processing and the limited resources available at the sensor nodes. In this thesis we have focused on energy-efficient query processing over historical observations stored at the sensor nodes. In particular, we have investigated three types of queries: spatial range queries, queries with joins, and a special type of approximate queries, that is, the spatiotemporal data distribution map queries.

Energy-efficient processing of spatial range queries in peer-to-peer sensor networks has been the topic of Chapter 3. We have introduced a general framework for processing such queries and, within the framework, the SWIF processing solution. In its first phase, SWIF finds a path from the query originator node to a coordinator node in the query region. In the second phase, the coordinator node contacts all relevant nodes using a constrained flooding strategy, collects the query relevant data and constructs the query answer, and returns this answer to the originator node. SWIF uses cost models to adapt its behaviour to the specifics of the query and sensor network. That is, the cost models are used to determine the most cost-effective location for the coordinator node. Empirical evaluation shows that SWIF reduces by up to 10-times the cost of processing small to medium spatial range queries when compared to hTAG, the typical processing solution. We have also shown that the cost models we have constructed for SWIF and hTAG can be used to select at query time the most energy efficient processing solution for a given query and originator node. Preliminary results from this chapter have been published in [CNS04].

In Chapter 4 we have analyzed several solutions for in-network processing of the join between two relations with respect to the location where the join operation should be performed. The investigated strategies perform the join at the user station (*External Join*), at the location of one of the join relations (*Local Join*), at an intermediate network location (*Mediated Join*) and in-network using the semi-join technique (*Local Semi-Join* and *Mediated Semi-Join*). We have shown empirically that no join processing strategy performs best for all queries. The *Local Semi-Join* is especially suitable when the query regions are small and the network density is low. The *External Join* performs best for dense networks and large query regions, while the *Local Join* does not perform well under any investigated scenario. The *Mediated Join* adapts well to the query characteristics and it is a good alternative to the *External Join* and *Local Semi-Join* if performing the join at the user station is not acceptable. Using our cost models to choose at query time the most efficient processing strategy, we are able to reduce the cost of join processing with up to 83% compared to processing every query with the same solution and perform within 7% of the optimal processing cost (i.e., if we would know beforehand which strategy will perform best). We have also analyzed in detail an algorithm for processing the theta-join operator in a sensor network. Our algorithm takes into account the memory available at the sensors nodes and the synchronization of the data flow. We have developed a cost model that allows our algorithm to be optimized with respect to the size of the join relations and the amount of available memory at the nodes processing the join. Our proposed algorithms can be combined to generate a full-fledged solution for join processing in sensor networks and their cost models can be used in a join query optimizer. Results from this chapter have been published in [CNS07, CN07].

Chapter 5 has introduced the spatiotemporal data distribution map (STDMAP) query which exploits the redundancy of sensor measurements on the spatial dimension. We have shown that the STDMAP query can be answered using only a subset of the relevant nodes and we have proposed three strategies (MSM, AFM and EFM) for its processing. The MSM strategy uses a depth first technique for query forwarding. This approach allows MSM to use the information on what nodes have already answered the query to select what other nodes should be contacted. The goal is to contact as few nodes as possible while proving a correct answer to the query. The AFM strategy uses a constrained flooding techniques to contact all relevant sensor nodes. The contacted nodes will decide to participate or not in the query answering based on partial information regarding the answering decision of their neighbours, which makes AFM an aggressive strategy. On the other hand, in the EFM strategy, nodes are aware of the decisions their neighbours take with respect to answering the query and base their decision on this information. In addition, the nodes' energy is used to decide which one in a pair of nodes covering each other should answer, which makes EFM an energy-aware strategy. In our empirical evaluation we have shown that the AFM algorithm is the most energy-efficient for most scenarios, closely followed by EFM. A secondary benefit of the EFM algorithm is that it provides a balanced energy use at the sensor nodes, an important advantage in applications where the queries are not uniformly distributed. The MSM algorithm is best suited for queries where the coverage range is larger than half the wireless range. In all our experiments EFM has shown low energy usage and consistent performance with respect to various network and query parameters, and therefore we recommend it for processing queries with similar characteristics to the STDMAP query. Results from this chapter have been published in [CNS05].

We have discussed several future research directions closely related to our investigations for each type of query in the respective chapters. In addition, we see several other related topics for future work. In our work we have focused on minimizing the cost of each individual query. A challenging problem is how to optimize query processing when multiple queries are processed in the network. Caching techniques (at the nodes or user-stations) could be used to reduce the cost of processing queries over historical data that has been queried before. When multiple queries are processed simultaneously, query rewriting combined with caching may help avoid retrieving the same data multiple times. In our investigations we have made a few steps toward building cost-based query optimizers. It would be interesting to see how to construct a general optimizer capable of combining multiple strategies for processing complex queries. The algorithms and their cost models should be created or modified to allow an incremental construction of a processing strategy for a complex query from basic processing blocks. Since query optimizers rely heavily on estimates, much work needs to be conducted on the efficient gathering of estimates in sensor networks, as well as on the trade-off between their accuracy and collection costs. Samples, data summaries and the use of past query answers are a few of the possible strategies for generating estimates. Sensor network applications are very dynamic by nature and query processing should be able to adapt to the change in conditions (e.g., selectivity, node availability, network load, energy levels). Interesting opportunities for research exist when investigating the adaptivity of operator ordering and placing, data rates, and granularity of aggregation. One possible solution may be the re-evaluation of the query plan (or strategy) as the query is forwarded into the network and more information or better estimates become available. A tighter integration of techniques for addressing sensor failures with our proposed techniques is another avenue of future research.

Bibliography

[AAK06] M.H. Ali, W.G. Aref, and I. Kamel. Scalability management in sensor-network phenomenabases. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 91–100, 2006.

[AFF$^+$03] A. Ailamaki, C. Faloutsos, P.S. Fischbeck, M.J. Small, and J. VanBriessen. An environmental sensor network to determine drinking water quality and security. *SIGMOD Record*, 32(4):47–52, 2003.

[AHS06] K. Aberer, M. Hauswirth, and A. Salehi. A middleware for fast and flexible sensor network deployment. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 1199–1202, 2006.

[AML05] D. Abadi, S. Madden, and W. Lindner. REED: robust, efficient filtering and event detection in sensor networks. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 769–780, 2005.

[ASSC02] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38(4):392–422, 2002.

[BB03] B.J. Bonfils and P. Bonnet. Adaptive and decentralized operator placement for in-network query processing. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*, pages 47–62, 2003.

[BC99] E.C. Barret and L.F. Curtis. *Introduction to Environmental Remote Sensing*. Stanley Thornes, 1999.

[BDSZ94] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. MACAW: a media access protocol for wireless LAN's. *SIGCOMM Computer Communications Review*, pages 212–225, 1994.

[BGS01] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proceedings of the International Conference on Mobile Data Management (MDM)*, pages 3–14, 2001.

[BMSU01] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad-hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[BSLC03] J. Beaver, M.A. Sharaf, A. Labrinidis, and P.K. Chrysanthis. Power-aware in-network query processing for sensor data. In *Proceedings of the Hellenic Data Management Symposium*, pages 1–17, 2003.

[CDHH06] D. Chu, A. Deshpande, J.M. Hellerstein, and W. Hong. Approximate data collection in sensor networks using probabilistic models. In *Proceedings of the International Conference on Data Engineering (ICDE)*, page 48, 2006.

[CG05] V. Chowdhary and H. Gupta. Communication-efficient implementation of join in sensor networks. In *Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 447–460, 2005.

[CLKB04] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 449–460, 2004.

[CN07] A. Coman and M.A. Nascimento. A distributed algorithm for joins in sensor networks. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, 2007. (To Appear).

[CNS04] A. Coman, M.A. Nascimento, and J. Sander. A framework for spatio-temporal query processing over wireless sensor networks. In *Proceedings of the International Workshop on Data Management for Sensor Networks (DMSN) (with VLDB)*, pages 104–110, 2004.

[CNS05] A. Coman, M.A. Nascimento, and J. Sander. Exploiting redundancy in sensor networks for energy efficient processing of spatiotemporal region queries. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, pages 187–194, 2005.

[CNS07] A. Coman, M.A. Nascimento, and J. Sander. On join location in sensor networks. In *Proceedings of International Conference on Mobile Data Management (MDM)*, 2007. (To Appear).

[Cro] Crossbow Technology Inc. MICA sensors. www.xbow.com.

[CSN05] A. Coman, J. Sander, and M.A. Nascimento. An analysis of spatio-temporal query processing in sensor networks. In *Proceedings of the International Workshop on Networking Meets Databases (NetDB) (with ICDE)*, pages 45–50, 2005.

[CTH06] D. Chu, A. Tavakoli, L. Popa 0002, and J.M. Hellerstein. Entirely declarative sensor network systems. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 1203–1206, 2006.

[DF03] M. Demirbas and H. Ferhatosmanoglu. Peer-to-peer spatial queries in sensor networks. In *Proceedings of the International Conference on Peer-to-Peer Computing (P2P2003)*, pages 32–39, 2003.

[DGHM05] A. Deshpande, C. Guestrin, W. Hong, and S. Madden. Exploiting correlated attributes in acquisitional query processing. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 143–154, 2005.

[DGM$^+$04] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 588–599, 2004.

[DGR$^+$03] A. Demers, J. Gehrke, R. Rajaraman, N. Trigoni, and Y. Yao. Energy-efficient data management for sensor networks. In *Proceedings of the Upstate New York Workshop on Sensor Networks*, 2003.

[DKR04] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing historical information in sensor networks. In *Proceedings of the SIGMOD Conference on Management of Data (SIGMOD)*, pages 527–538, 2004.

[DNGS03] A. Deshpande, S. Nath, P.B. Gibbons, and S. Seshan. Cache-and-query for wide area sensor databases. In *Proceedings of the SIGMOD Conference on Management of Data (SIGMOD)*, pages 503–514, 2003.

[EN06] C. Estan and J.F. Naughton. End-biased samples for join cardinality estimation. In *Proceedings of the International Conference on Data Engineering (ICDE)*, page 20, 2006.

[Fin87] G.G. Finn. Routing and addressing problems in large metropolitan-scale internetworks. ISI Res. Rep. ISU/RR-87-180, University of Southern California, 1987.

[FS06] H. Frey and I. Stojmenovic. On delivery guarantees of face and combined greedy-face routing algorithms in ad hoc and sensor networks. In *Proceedings of the ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 390–401, 2006.

[FZG02] Q. Fang, F. Zhao, and L. Guibas. Counting targets: Building and managing aggregates in wireless sensor networks. Technical Report P2002-10298, Palo-Alto Research Center, 2002.

[GBT$^+$04] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden. Distributed regression: an efficient framework for modelling sensor network data. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*, pages 1–10, 2004.

[GEH02] D. Ganesan, D. Estrin, and J. Heidemann. DIMENSIONS: Why do we need a new data handling architecture for sensor networks. In *Proceedings of the Workshop on Hot Topics in Networks (HotNets)*, 2002.

[GGP$^+$03] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann. An evaluation of multi-resolution storage for sensor networks. In *Proceedings of the Conference on Embedded Networked Sensor Systems (SenSys)*, pages 89–102, 2003.

[GLG$^+$05] R. Gummadi, X. Li, R. Govindan, C. Shahabi, and W. Hong. Energy-efficient data organization and query processing in sensor networks. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 157–158, 2005.

[GM04] J. Gehrke and S. Madden. Query processing in sensor networks. *Pervasive Computing*, Jan. 2004.

[GR65] I. Greenberg and R.A. Robertello. The three factory problem. *Mathematics Magazine*, 38(2):67–72, 1965.

[GRE] The Great Duck Island Project. www.greatduckisland.net.

[GS69] HK.R. Gabriel and R.R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18:259278, 1969.

[GSB03] S. Giordano, I Stojmenovic, and L. Blazevic. Position based routing algorithms for ad-hoc networks: A taxonomy. In *Ad-Hoc Wireless Networking*. Kluwer, 2003.

[HAE03] M.A. Hammad, W.G. Aref, and A.K. Elmagarmid. Stream window join: Tracking moving objects in sensor-network databases. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 75–84, 2003.

[Hei00] W. Heinzelman. *Application-Specific Protocol Architectures for Wireless Networks*. PhD thesis, MIT, 2000.

[HZGS05] G. He, R. Zheng, I. Gupta, and L. Sha. A framework for time indexing in sensor networks. *ACM Transactions on Sensor Networks*, 1(1):101–133, 2005.

[IGE$^+$03] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE Transactions on Networking*, 11(1):2–16, 2003.

[JPC05] X. Jiang, J. Polastre, and D. Culler. Perpetual environmentally powered sensor networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*, pages 463–468, 2005.

[Kap96] E.D. Kaplan, editor. *Understanding GPS: Principles and applications*. Artech House Publishers, 1996.

[KK00] B. Karp and H.T. Kung. Greedy perimeter stateless routing for wireless networks. In *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*, pages 243–254, 2000.

[Kos00] D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):442–469, 2000.

[Kot05] Y. Kotidis. Snapshot queries: towards data-centric sensor networks. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 131–142, 2005.

[KSU99] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proceedings of the 11th Canadian Conference on Computational Geometry*, pages 51–54, 1999.

[LCT05] S.H.L. Liang, A. Coritoru, and C.V. Tao. A distributed geo-spatial infrastructure for smart sensor webs. *Computers and Geosciences*, 31(2):221–231, 2005.

[LOT94] H. Lu, B.C. Ooi, and K.-L. Tan, editors. *Query processing in parallel relational database systems*. IEEE Computer Society Press, 1994.

[Mai04] C. Maihofer. A survey of geocast routing protocols. *IEEE Communications Surveys*, 6(2):32–42, 2004.

[Mar03] K. Marron. Ice wine and cool technology. *The Globe and Mail*, May 21, 2003.

[MF02] S. Madden and M.J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 555–566, 2002.

[MFH02] S. Madden, M.J. Franklin, and J.M. Hellerstein. TAG: a tiny aggregation service for ad-hoc sensor networks. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, pages 131–146, 2002.

[MFHH03] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the SIGMOD Conference on Management of Data (SIGMOD)*, pages 491–502, 2003.

[MNG05] A. Manjhi, S. Nath, and P.B. Gibbons. Tributaries and deltas: Efficient and robust aggregation in sensor network streams. In *Proceedings of the SIGMOD Conference on Management of Data*, pages 287–298, 2005.

[MPS$^+$02] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the International Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 88–97, 2002.

[MWH01] M. Mauve, J. Widmer, and H. Hartenstein. A survey on position based routing in mobile ad-hoc networks. *IEEE Network Magazine*, 15(6):30–39, 2001.

[NHZ04] I. Nikolaidis, J. Harms, and S. Zou. On sensor data aggregation with redundancy removal. In *Proceedings of the Queen's Biennial Symposium on Communications (QBSC)*, 2004.

[NL04] B.G. Nickerson and J. Lu. A language for wireless sensor webs. In *Proceedings of the Conference on Communication Networks and Services Research (CNSR)*, 2004.

[OHB06] A. Omotayo, M.A. Hammad, and K. Barker. Efficient data harvesting for tracing phenomena in sensor networks. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 59–70, 2006.

[OV99] M.T. Ozsu and P. Valduriez. *Principles of distributed database systems.* Prentice-Hall, Inc., 1999.

[PG06] A. Pandit and H. Gupta. Communication-efficient implementation of range-join in sensor networks. In *Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 859–869, 2006.

[Rap96] T. Rappaport. *Wireless Communications: Principles and Practice.* Prentice-Hall Inc., 1996.

[RG00] R. Ramakrishnan and J. Gehrke. *Database Management Systems - 2nd ed.* McGraw-Hill, 2000.

[Ric05] A. Ricadela. Sensors everywhere. *Information Week*, Jan. 24, 2005.

[RKH$^+$05] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*, pages 457–462, 2005.

[RKY$^+$02] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A geographic hash table for data-centric storage. In *Proceedings of the International Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2002.

[SBE$^+$06] A. Silberstein, R. Braynard, C. Schlatter Ellis, K. Munagala, and J. Yang. A sampling-based approach to optimizing top-k queries in sensor networks. In *Proceedings of the International Conference on Data Engineering (ICDE)*, page 68, 2006.

[SBLC03] M.A. Sharaf, J. Beaver, A. Labrinidis, and P.K. Chrysanthis. TiNA: A scheme for temporal coherency-aware in-network aggregation. In *Proceedings of the International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE)*, pages 69–76, 2003.

[SBLC04] M.A. Sharaf, J. Beaver, A. Labrinidis, and P.K. Chrysanthis. Balancing energy efficiency and quality of aggregate data in sensor networks. *VLDB Journal*, 13(4):384–403, 2004.

[SBY06] A. Silberstein, R. Braynard, and J. Yang. Constraint chaining: on energy-efficient continuous monitoring in sensor networks. In *Proceedings of the SIGMOD Conference on Management of Data*, pages 157–168, 2006.

[SD04] I. Stojmenovic and S. Datta. Power and cost aware localized routing with guaranteed delivery in unit graph based ad-hoc networks. *Wireless Communications and Mobile Computing*, 4(2):175–188, 2004.

[Sen] Sensoria Corp. WINS sensor platform. www.sensoria.com.

[SFL05] S. Schmidt, M. Fiedler, and W. Lehner. Source-aware join strategies of sensor data streams. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 123–132, 2005.

[SMY06] A. Silberstein, K. Munagala, and J. Yang. Energy-efficient monitoring of extreme values in sensor networks. In *Proceedings of the SIGMOD Conference on Management of Data*, pages 169–180, 2006.

[SPP$^+$06] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 187–198, 2006.

[SRK$^+$02] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin. Data-centric storage in sensornets. In *Proceedings of the Workshop on Hot Topics in Networks (HotNets)*, 2002.

[SS04] M. Sharifzadeh and C. Shahabi. Supporting spatial aggregation in sensor network databases. In *Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, pages 166–175, 2004.

[SS06] M. Sharifzadeh and C. Shahabi. Utilizing voronoi cells of location data streams for accurate computation of aggregate functions in sensor networks. *GeoInformatica*, 10(1):9–36, 2006.

[Sto02] I. Stojmenovic. Position based routing in ad hoc networks. *IEEE Communications Magazine*, 40(7):128–134, 2002.

[Sto04] I. Stojmenovic. Geocasting with guaranteed delivery in sensor networks. In *Proceedings of the Workshop on Theoretical and Algorithmic Aspects of Sensor, AD-HOC Wireless and P2P Networks*, 2004.

[SY04] F. Sivrikaya and B. Yener. Time synchronization in sensor networks: a survey. *IEEE Network*, 18(4):45–50, 2004.

[TK84] H. Takagi and L. Kleinrock. Optimal transmission ranges for randomly distributed packet radio terminals. *IEEE Transactions on Communications*, 32(3):247–256, 1984.

[Tou80] G. Toussaint. The relative neighborhood graph of a finite planar set. *Pattern Recognition*, 12(4):261268, 1980.

[WXTL06] M. Wu, J. Xu, X. Tang, and W.C. Lee. Monitoring top-k query inwireless sensor networks. In *Proceedings of the International Conference on Data Engineering (ICDE)*, page 143, 2006.

[XL03] Y. Xu and W.C. Lee. Window query processing in highly dynamic sensor networks: Issues and solutions. In *Proceedings of the Workshop on GeoSensor Networks*, 2003.

[XLCL06] W. Xue, Q. Luo, L. Chen, and Y. Liu. Contour map matching for event detection in sensor networks. In *Proceedings of the SIGMOD Conference on Management of Data*, pages 145–156, 2006.

[XLXM06] Y. Xu, W.C. Lee, J. Xu, and G. Mitchell. Processing window queries in wireless sensor networks. In *Proceedings of the International Conference on Data Engineering (ICDE)*, page 70, 2006.

[YG02] Y. Yao and J. Gehrke. The Cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31(3):9–18, 2002.

[YG03] Y. Yao and J. Gehrke. Query processing in sensor networks. In *Proceedings of the International Conference on Innovative Data Systems Research (CIDR)*, 2003.

[YLZ06] H. Yu, E.P. Lim, and J. Zhang. On in-network synopsis join processing for sensor networks. In *Proceedings of International Conference on Mobile Data Management (MDM)*, pages 32–39, 2006.

[YS05] S.H. Yoon and C. Shahabi. Exploiting spatial correlation towards an energy efficient clustered aggregation technique (CAG). In *Proceedings of IEEE International Conference on Communications (ICC)*, pages 3307– 3313, 2005.

[YS07] S.H. Yoon and C. Shahabi. The clustered aggregation (CAG) technique leveraging spatial and temporal correlations in wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 3(1), 2007.

[ZG04] F. Zhao and L. Guibas. *Wireless sensor networks: an information processing approach.* Morgan Kaufmann, 2004.

[ZGTS03] D. Zhang, D. Gunopulos, V.J. Tsotras, and B. Seeger. Temporal and spatio-temporal aggregations over data streams using multiple time granularities. *Information Systems*, 28:61–84, 2003.

[ZNH04] S. Zou, I. Nikolaidis, and J. Harms. Efficient data collection trees in sensor networks with redundancy removal. In *Proceedings of the International Conference on Ad-Hoc Networks & Wireless (ADHOC-NOW)*, pages 252–265, 2004.

[ZNH05] S. Zou, I. Nikolaidis, and J. Harms. ENCAST: Energy-critical node aware spanning tree for sensor networks. In *Proceedings of the Annual Communication Networks and Services Research Conference (CNSR)*, pages 249–254, 2005.

[ZNH06] S. Zou, I. Nikolaidis, and J. Harms. Extending sensor network lifetime via first hop data aggregation. In *Proceedings of the IEEE International Performance Computing and Communications Conference (IPCCC)*, pages 397–405, 2006.

Appendix A

The Average Advance Towards Destination over a Hop

We assume the nodes are uniformly distributed in the monitored region, and therefore a node's neighbours are also uniformly distributed within its wireless range. Let us denote with a_A the area of the circular segment within the wireless range circle where the neighbour with the longest advance must be located (see Figures 3.3 and A.1). The size of the network area corresponding to each node is equal to $\frac{R_A}{N}$ under uniform distribution. Since a_A is the smallest circular segment such that there is exactly one node inside (from a probability perspective considering the uniform node distribution), we have that $a_A = \frac{R_A}{N}$. From basic geometry we also have that the area of the circular segment a_A is equal to $W^2 \arccos(\frac{W-h}{W}) - (W - h)\sqrt{2Wh - h^2}$, where h is the height of the arced portion. From the equality of the two expressions of a_A we can find h since the other terms are known, and thus the coordinates for the circular segment. The neighbour located in a_A could be located anywhere within a_A, and, thus, we need to find the advance (in average) that such a neighbour would produce. The average advance a_{hop} of the neighbour selected

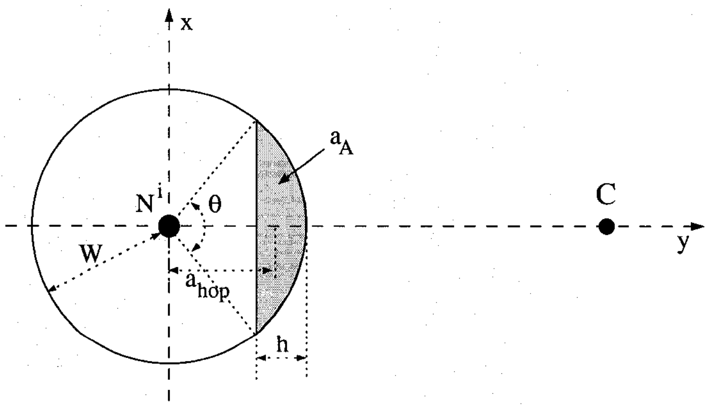


Figure A.1: The average advance a_{hop} for a hop

for forwarding toward C is equal to the sum of the advances for all possible locations for the selected neighbour divided by the number of these locations (see Figure A.1):

$$a_{hop} = \frac{\int\int_{a_A} y \; dx\, dy}{\int\int_{a_A} dx\, dy}$$

The integral $\int\int_{a_A} dx\, dy$ is equal to the size of the area where the points in a_A are and it is equal to a_A. For the upper term we have:

$$\begin{aligned}
\int\int_{a_A} y \; dx\, dy &= \int_{-\sqrt{2Wh-h^2}}^{\sqrt{2Wh-h^2}} \int_{W\cos(\frac{\theta}{2})}^{\sqrt{W^2-x^2}} y \; dy\, dx \\
&= \frac{1}{2}\int_{-\sqrt{2Wh-h^2}}^{\sqrt{2Wh-h^2}} (W^2 - x^2 - W^2\cos^2(\frac{\theta}{2})) \; dx \\
&= \frac{2}{3}(2Wh - h^2)^{\frac{3}{2}}
\end{aligned}$$

Appendix B

The Average Distance From a Relevant Node to the Coordinator Node

B.1 General Formula

We assume the nodes are uniformly distributed in the monitored region, and therefore the relevant nodes are also uniformly distributed within the query region. We consider that the spatial range of the query forms a rectangular area (Q_A), specified by its opposite corners ($(< x_1, y_1 >, < x_2, y_2 >)$). The average distance from a relevant node to the coordinator can be computed as the sum of the distances from the coordinator C to all possible locations of the relevant nodes divided to the number of these locations:

$$
\begin{aligned}
d_{N^iC} &= \frac{\int_{x_1}^{x_2}\int_{y_1}^{y_2}\sqrt{(x-c_x)^2+(y-c_y)^2}dydx}{\int_{x_1}^{x_2}\int_{y_1}^{y_2}dydx} \\
&= \frac{\int_{x_1}^{x_2}\int_{y_1}^{y_2}\sqrt{(x-c_x)^2+(y-c_y)^2}dydx}{(x_2-x_1)(y_2-y_1)}.
\end{aligned}
$$

B.2 Solving the Integral

Finding a solution for the integral $\int_{x_1}^{x_2}\int_{y_1}^{y_2}\sqrt{(x-c_x)^2+(y-c_y)^2}dydx$ is non-trivial and cannot be obtained directly with integration solving software, and therefore we will show its solution here. To solve the integral using standard anti-derivatives, we need $x - c_x > 0$ and $y - c_y > 0$ over the integrating area. Since the coordinator node $C(c_x, c_y)$ is located in the query region, we have $x_1 < c_x < x_2$ and $y_1 < c_y < y_2$. Therefore we brake the integral into subintervals. We have:

$$\begin{aligned} d^{sum}_{N^iC} &= \int_{c_x}^{x_2}\int_{c_y}^{y_2}\sqrt{(x-c_x)^2+(y-c_y)^2}dydx \\ &+ \int_{x_1}^{c_x}\int_{c_y}^{y_2}\sqrt{(c_x-x)^2+(y-c_y)^2}dydx \\ &+ \int_{c_x}^{x_2}\int_{y_1}^{c_y}\sqrt{(x-c_x)^2+(c_y-y)^2}dydx \\ &+ \int_{x_1}^{c_x}\int_{y_1}^{c_y}\sqrt{(c_x-x)^2+(c_y-y)^2}dydx. \end{aligned}$$

Using simple substitutions, we can express $d^{sum}_{N^iC}$ by a sum of the same integral *I(u,v)* for different values of u and v, with $u, v > 0$. For simplicity of presentation, we solve the indefinite integral *I(u,v)*:

$$\begin{aligned} I(u,v) &= \int\int\sqrt{u^2+v^2}dvdu \\ &= \int\left(\frac{1}{2}v\sqrt{u^2+v^2}+\frac{1}{2}u^2\ln(v+\sqrt{u^2+v^2})\right)du \\ &= \frac{1}{4}uv\sqrt{u^2+v^2}+\frac{1}{4}v^3\ln(u+\sqrt{u^2+v^2})+\frac{1}{2}\int u^2\ln(v+\sqrt{u^2+v^2})du \end{aligned}$$

For clarity of solution, let us denote the remaining integral by $I_1(u,v)$ and solve it separately:

$$\begin{aligned} I_1(u,v) &= \int u^2\ln(v+\sqrt{u^2+v^2})du \\ &= \frac{1}{3}u^3\ln(v+\sqrt{u^2+v^2})-\frac{1}{3}\int\frac{u^4}{(v+\sqrt{u^2+v^2})\sqrt{u^2+v^2}}du \end{aligned}$$

We denote the remaining integral by $I_2(u,v)$ and solve it separately:

$$\begin{aligned} I_2(u,v) &= \int\frac{u^4}{(v+\sqrt{u^2+v^2})\sqrt{u^2+v^2}}du \\ &\quad \text{substitute } \sqrt{u^2+v^2}=s \\ I_2(u,v) &= \int(s-v)\sqrt{s^2-v^2}ds \\ &= \frac{1}{3}(s^2-v^2)^{\frac{3}{2}}-v[\frac{s\sqrt{s^2-v^2}}{2}-\frac{v^2}{2}\ln(s+\sqrt{s^2-v^2})] \\ &\quad \text{substitute back } s=\sqrt{u^2+v^2} \\ I_2(u,v) &= \frac{1}{3}u^3-\frac{1}{2}uv\sqrt{u^2+v^2}+\frac{1}{2}v^3ln(u+\sqrt{u^2+v^2}) \end{aligned}$$

Substituting $I_2(u,v)$ in $I_1(u,v)$ and $I_1(u,v)$ in $I(u,v)$ and reducing terms, we obtain:

$$I(u,v) \;=\; \frac{1}{3}uv\sqrt{u^2+v^2} + \frac{1}{6}v^3 ln(u+\sqrt{u^2+v^2}) + \frac{1}{6}u^3 ln(v+\sqrt{u^2+v^2}) - \frac{1}{18}u^3$$

The final analytical formulas for $d^{sum}_{N^iC}$ and d_{N^iC} contain numerous terms and are not shown.

Appendix C

Finding the Optimal Join Location for the Mediated Join

In Section 4.3.4 we have introduced the Mediated Join solution for in-network processing of join queries. To find the optimal location for the join location, we have used the solution proposed by Greenberg and Robertello [GR65] for finding the weighted Fermat point. We have outlined the solution for several trivial cases in Section 4.3.4. We detail the solution (according to [GR65]) when the optimal join location J falls inside the triangle formed by the locations of the relations A and B and the query originator O. (see Figure C.1). Let us denote with a, b and j the size of relations A, B and J and with d_{XY} the Euclidean distance between locations X and Y. We want to find the optimal join location J such that:

$$ad_{AJ} + bd_{BJ} + jd_{JO} \quad \text{is minimized.}$$

The interior angles of the triangle ΔABO sum to 2π and are all less than π. Thus, at least two of the central angles must be obtuse. Let us assume $\angle AJB$ is obtuse. Since J lies in the triangle, we can determine its location from the locations of A, B and O and the

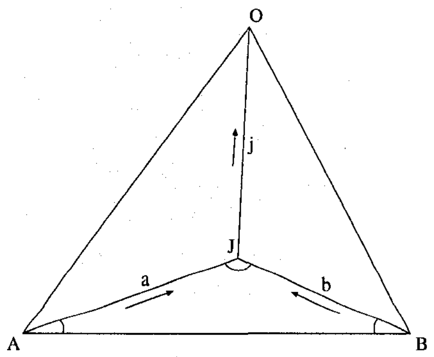


Figure C.1: Finding the optimal join location

angles $\angle JAB$ and $\angle JBA$ (note that both these angles are acute). According to [GR65], we have:

$$\tan(\angle JAB) = \frac{d_{BO}\cos(\alpha)\sin(\alpha+\beta+2\tan^{-1}(\frac{T}{S-d_{AO}}))}{d_{AB}\cos(\beta) - d_{BO}\cos(\alpha)\cos(\alpha+\beta+2\tan^{-1}(\frac{T}{S-d_{AO}}))}$$

and

$$\angle JBA = \frac{\pi}{2} - \alpha - JAB$$

where α, β, S and T are defined as:

$$\sin(\alpha) = \frac{a^2+b^2-j^2}{2ab},$$

$$\sin(\beta) = \frac{j^2+b^2-a^2}{2jb},$$

$$S = \frac{1}{2}(d_{AB}+d_{AO}+d_{BO}) \quad \text{and}$$

$$T = \sqrt{\frac{(S-d_{AB})(S-d_{AO})(S-d_{BO})}{S}}.$$

Appendix D

Estimating the number of nodes within *h-hops* from a node

In this section we estimate the average number of nodes located up to h hops away from a network node. We estimate this value here as it is a network dependent value and independent of our algorithm.

Let A_N be the area of the network, N the number of sensor nodes uniformly distributed over the network area and S the node whose h hop neighbors we try to determine. The number of nodes located up to h-hops away from S is equal to the number of sensor nodes located in an area equal in size to the area where these nodes are located. Let us denote this area with A_h. We have that $N_n^h = N\frac{A_h}{A_N}$ for $h \geq 1$. For $h = 0$ we have $N_n^0 = 1$ to account for S.

We need to find the size of the area A_h. For the 1-hop neighbours, A_1 is equal to the circle of wireless range radius W and we have $N_n^1 = N\frac{\pi W^2}{A_N}$. The average distance from S to its 1-hop neighbours is $d_{1hop} = \int\int_{A_1} d_{SN_i} dA_1 = \frac{2}{3}W$, where d_{SN_i} represents the distance between S and a 1-hop neighbour N_i. Since the 1-hop neighbours are located in average at distance d_{1hop} away from S, and the neighbours of the 1-hop neighbours could be located as far as W, we have that the 2-hop neighbours of S are located in average within a circle of radius $d_{1hop} + W = \frac{5}{3}W$. Generalizing this result for h-hop neighbours, we have that they are located within a circle of $(h-1)d_{1hop} + W = \frac{2h+1}{3}W$ radius from S. Therefore, we have:

$$N_n^h = \frac{\pi(2h+1)^2W^2}{9A_N}N \quad \text{for} \quad h \geq 1.$$