MULTIPLE QUERY OPTIMIZATION IN WIRELESS SENSOR NETWORKS

XIANG SHILI

(B.Eng., UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2011

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Professor Kian-Lee Tan, for his guidance, support and encouragement throughout my Ph.D. study. I am very grateful for the countless hours he has spent in nurturing me and discussing with me, when he has taught me innumerable lessons and insights on the workings of academic research in general. Moreover, I appreciate very much his generous financial support and tremendous mental support, especially when I was frustrated at times during the final stage of my Ph.D. study. His technical and editorial advice is essential to the completion of this thesis, while his kindness and wisdom have made great impact on my life.

My thanks also go to Dr. Hock-Beng Lim. Dr. Lim provided me with resources to have hands-on experience on sensor nodes, and his insights on sensor networks and encouragement were of great help for my research.

I want to express my sincere thanks to my senior Dr. Yongluan Zhou. Apart from contributing helpful discussions to refine my work, he spent much effort in updating my writings and improving my presentations. I am also indebted to Professor Karl Aberer, for the guidance and the opportunity for internship. Prof. Aberer and Dr. Zhou taught and inspired me many things when I worked with them as an internship researcher at EPFL.

I am happy that I have been a member of the database group, a big family full of joy and research spirit. I am very thankful to Dr. Anthony K. H. Tung, Dr. Mong Li Lee, Dr. Stéphane Bressan and Dr. Panagiotis Kalnis, who provided valuable feedback and suggestions to my research work and the thesis. I would also like to thank Professor Beng Chin Ooi, my mentor in the first semester, for his inspiration, guidance and care. My thanks also go to my colleagues in the group, for their encouragement, discussions and friendship. They are: Wei Ni, Ji Wu, Wei Wu, Ding Chen, Bin Liu, Chang Sheng, Yingguang Li, Yu Cao, Jianneng Cao, Wee Hyong Tok, Chenyi Xia, Xiaoyan Yang, Yueguo Chen, Yuan Ni, Weiwei Cheng, Zhifeng Bao, Liang Xu, Huayu Wu, Sai Wu, Zhenjie Zhang, Su Chen, Bingtian Dai, Nan Wang, Jingbo Zhang, Xiaohui Li and all other previous and current database group members.

I would like to thank my parents for their dedicated love, care and the many years of support during my studies. I also want to thank my husband, Gengpu, for his support and encouragement during the past few years.

Finally, I want to thank NUS for providing me the scholarship so that I can concentrate on study.

Table of Contents

Summary

Wireless sensor networks (WSNs), comprising a large number of radio-enabled programmable sensor nodes, have been increasingly deployed in many important applications to enable users to query the physical world. However, since WSNs are inherently resource constrained, when several queries are running simultaneously, existing works on optimization and execution of a single query are deficient, and multiple query optimization that enables query sharing is indispensable. Hence, the purpose of this thesis is to tackle the problem of multiple query optimization in WSNs, to make the whole network scalable and efficient.

To achieve the energy-efficiency and scalability with the number of queries in WSN, we propose a Two-Tier Multiple Query Optimization (TTMQO) scheme. It is light-weight, adaptive with query arrivals / terminations, and supports both aggregation and data acquisition queries. The first tier adopts a cost-based approach to rewrite queries into an optimized set to share the commonality and reduce redundancy among queries. In the second tier, in-network optimization is conducted to efficiently deliver query results by taking advantage of the broadcast nature of the radio channel and sharing the sensor readings among multiple queries over space and time in a distributed manner. Both tiers eliminate the redundancies incurred for similar queries, though in different ways, and their marriage

can utilize their advantages while avoiding their respective disadvantages.

To further enhance the scalability in terms of number of sensor nodes and improve the reliability and energy efficiency, we then identify the importance of an infrastructure with multiple base stations. To minimize the total data communication cost among the sensors, it is critical to intelligently allocate queries among base stations to leverage query sharing. We first examine the query allocation problem in a static context, where all the queries are known in advance. Here, we approximate the problem of allocating queries to K base stations as a Max-K-Cut problem, and adapt an existing solution to our context. In addition, considering the complexity of Max-K-Cut solution, we propose a semi-greedy allocation framework, which consists of a greedy allocation phase and an iterative refinement phase. We also investigate dynamic environments with frequent query arrivals and terminations and propose adaptive query insertion and migration algorithms.

Recently, mobile sensors have been developed and increasingly deployed to support various applications. Thus, besides optimizing multiple queries in static WSNs, we also investigate how multiple data acquisition queries can be answered quickly in sparse mobile sensor networks. Because of the sparseness and mobility, the number of sensors is limited, the connection is intermittent and the topology is unpredictable. To effectively handle the above challenges, we design distributed schemes in which the exploited mobile sensors strategically relocate themselves to proper locations to collaboratively facilitate efficient query processing and enable sharing over space and time. In addition, the most appropriate scheme is selected to adapt to the environment.

We have implemented the above approaches and conducted extensive experimental studies, which demonstrate the efficiency and effectiveness of these approaches. We believe that our research in optimizing multiple queries for WSNs significantly contributes to promoting WSN applications.

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we will describe the background of wireless sensor networks, give a general overview of query processing techniques in the current literature, and present the rationale of our study on multiple query optimization in wireless sensor networks.

1.1 Background

1.1.1 Wireless Sensor Networks (WSNs)

Recent advances in microelectronics have led to the development of micro sensors and the reduction of their size and cost, enabling the large deployment of such sensing devices. Each sensor node has one or more such sensors to sense the environment, a microprocessor to process user requests and sensory data, some memory to store data sensors sensed, a short range communication component such as radio to communicate with other sensors, and a power supply component to provide the energy so that sensor node can operate by itself. Being equipped with these data

acquisition, computation, storage and communication capacities, most of sensor nodes today (still or mobile) have programmable ability so that application-specific tasks can be built-in and collaboration protocols between sensor nodes are also possible. A Wireless Sensor Network (WSN) is a distributed system that comprises a large number of these sensor nodes.

Since sensor nodes are small, inexpensive, low power and programmable, they make it possible to sense information at previously unobtainable resolutions, and thus WSNs are very attractive to a large variety of applications [111, 6, 47, 71, 105, 57]. They can be used in resource-limited and harsh environments, such as earthquake areas, ecological contamination sites, or military battlegrounds. They can also be deployed in everyday-life environments, such as smart home environment, intelligent museum/zoo, warehouse/port, industry plants or road. They have the ability to collect many types of physical measurements, such as temperature, light, humidity, movement, seismic and noise. In this way, their deployment enables us to monitor and query the physical world anywhere, anytime. Hence, WSNs are expected to greatly improve our understanding of the world, enrich the Cyber-Physical infrastructure and provide our life with tremendous convenience.

However, these sensor nodes are inherently resource constrained due to the small size and low price constraints of sensor nodes. Take the UC Berkeley mica2 Mote that we have played with for example: it has a 7 Mhz processor, a 38.6Kbps radio with $\sim$100 foot range, 4KB of RAM and 512KB flash, runs on AA batteries and uses $\sim$15 mA in active power consumption. As a result, these sensor nodes have brought about a large number of challenges [125, 6]. Firstly, the microprocessor at each sensor node has limited

computation capacity, and hence complicated processing cannot be done at the sensor node. Secondly, the memory of sensor nodes available for programming and data is small, and thus cannot store the code for complex programs and large datasets. Thirdly, sensor nodes are communicating with each other using short range wireless radio, which usually has limited bandwidth of wireless links and results in constrained and unstable communication. Fourthly, most current sensor nodes are powered by batteries and have limited supply of energy, and therefore it is critical to conserve energy. Lastly, but by no means least, there is uncertainty in sensor readings. It is hard to guarantee that the hardware of sensor nodes is 100% accurate; and both the sensing technology and radio technology are affected by the environment, e.g., unpredictable variations, noise and the weather conditions. Thus, a large number of readings from a group of sensors over some duration are often of interest other than single-sensor-single-time readings.

From the above, we can see that the WSN can be very promising, but due to the resource constraint, its various challenges and the complex physical environment it is closely coupled to, special attention should be paid to ensure its applicability. To provide better data fidelity and robustness, a large number of sensor nodes are often densely scattered in a sensor field, as shown in Figure 1.1. Due to the short wireless communication ranges, data are typically routed back to the sink in a multi-hop way. Sinks communicate with the base station via Internet, satellites and cable etc. For simplicity, as the sink communicates with the base station directly, we shall use the term base station to refer to both the sink and base station. Special protocols are proposed to form the network topology, so that each node has at least one route to the base station/sink. Several related issues

are broadly addressed and investigated in different WSN research communities, for instance, synchronization [94, 100, 127], energy efficient data routing [39, 42, 109] and clustering [129, 126].

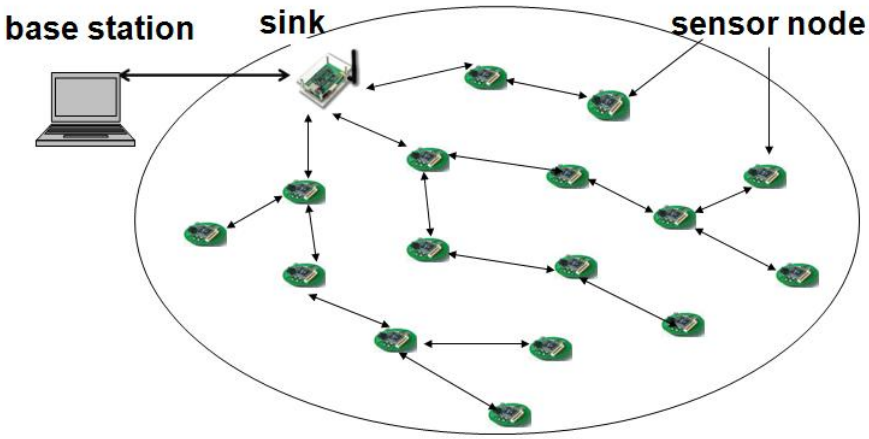


Figure 1.1: Components of sensor networks

1.1.2 Query Processing in WSNs

To ease the deployment of WSN applications, researchers have proposed techniques to treat the WSN as a database [125, 25, 34], which provides a good logical abstraction for sensor data management. A database oriented approach brings with it several advantages. First, it provides users with ease. Users can issue declarative queries based on the logical view of the data held inside the sensor networks, without having to worry about the actual implementation details of the operations on the physical network, such as storage, networking, link status, etc. Second, it provides the users with flexibility. Without reprogramming the sensor nodes as traditional approach does, users have the ability to dynamically submit queries to acquire different information with time from the WSN. Thirdly, it provides

the WSN with the opportunity for better performance. Query optimization techniques can be applied to optimize the network operations. Lastly, it provides the system with better availability and scalability. It has enabled the WSN to handle several tasks simultaneously. More specifically, with just one application such as TinyDB [69] built in the WSN, multiple users can deploy or change their interests by submitting different concurrent queries, and all of the queries can be answered at the same time.

In database oriented approach, a WSN works in the following way:

(a). A user submits a query to the base station of the whole network to request the data he or she is interested, which could be either the basic sensory data or the more advanced processed sensory data;

(b). The base station generates the query plan and propagates the query to the sensor nodes in the network;

(c). After sensor nodes receive the query, they sense the environment to collect the required data, process the data according to the query specification, and collaborate with other sensors to transmit the processed data back to the base station;

(d). The base station post-processes all the data received from individual sensors and then produces the answer to the user who issues the query.

Considering the special characteristics and hardware constraints of the WSNs, we can see that steps (b) and (c) are where the main challenges lie, in the whole process during which a WSN fulfills its task. It is, therefore, of critical importance to have effective and efficient query processing techniques for steps (b) and (c).

Some query processing systems for WSNs, such as, Cougar [125, 25], and TinyDB [68, 69], have been developed. These pioneer works establish the foundation of sensory data management and query processing. They define the query languages for WSNs, discuss the basic issues of sensor queries, and provide some solutions to query processing.

Besides these efforts on query processing systems in general, as we can see in more detail in Section 2.2, a large number of studies have focused more specifically on various aspects of sensor query processing techniques. Some studies focus on the design of routing protocols, trying to propose an energy-efficient protocol on sensor networks [63, 92]. Much work has been done on how to intelligently store data inside network and answer queries outside or inside the network efficiently [81, 27]. Many other studies have been conducted on in-network query processing techniques [97, 4], to reduce the amount of sensory data that needs to be sent to the base station and produce the query answers as soon as possible. For approximate queries, both model-driven and non model-driven techniques on sensory data have also been widely studied [26, 91], to save the energy and bandwidth while satisfying users' requirements to improve the overall performance in wireless sensor networks. In addition, several adaptive techniques have been proposed to adjust query strategies and optimize query plans over time [7, 24].

1.2 Motivation

In real life, there are many scenarios where we should allow multiple users to pose their queries to the system, and these queries run concurrently. In the multi-query situation, in order for the system to scale effectively

and efficiently with the number of simultaneous queries, special efforts are required to enable query sharing over the common operations and limited resources. We call this problem as *Multiple Query Optimization*(MQO).

1.2.1 Multiple Query Optimization (MQO) in large-scale WSNs

Since WSNs are extremely resource-constrained, MQO that enables query sharing is indispensable to make the whole network scalable and efficient. As aforementioned, existing works have mostly focused on the optimization and execution of a single long-running query. Consequently, when multiple queries are running simultaneously in a sensor network, they cannot benefit from each other by sharing their data acquisition, computation and communication cost. Moreover, running multiple queries in such an uncooperative manner will lead to bandwidth contention and even data loss as a result of transmission collisions (which may in turn require retransmission).

A good MQO scheme for WSNs should have the following set of desired properties:

- **Scalability**. As the popularity and importance of WSNs grow, so does the size of the sensor network and the number of users that are interested in the sensory data. Being able to work well for sensor networks with a large number of nodes and support a large number of queries is of critical importance to improve WSN's fidelity and availability.

- **Energy-efficiency**. Moore's law suggests that the memory density and processor speed will continue to grow at an exponential rate, so

we can predict that sensor networks will continue to be bandwidth and energy limited in the foreseeable future. Thus, it is of fundamental importance to have energy-efficient scheme for sensor nodes.

- **Adaptivity**. In WSN applications, queries are likely to be long-running continuous ones, and they may arrive and leave at any time. A good MQO scheme should be able to continuously adapt query plans to current query workload and network condition, so that these plans can always be optimal or near optimal over time.

- **Simplicity**. Sensor nodes have limited processing power, small storage space and limited bandwidth, which restrict the types of data processing algorithms that can be deployed, intermediate results that can be stored, and the size and rate of the data that can be communicated. In order to be applicable, the MQO scheme should be light weight.

However, the MQO techniques in traditional databases are not applicable to WSNs, due to different semantics of queries, different objective through optimization and the resource constraints of each sensor node. While the studies on MQO in stream databases deal with the problem of processing multiple queries efficiently at the base station, they do not tackle the problem of collecting sensory data out of the WSNs [70, 16, 48, 49]. Thus the real challenge in MQO in sensor networks remains unsolved.

There are only a few studies addressing the problem of MQO in WSNs. In Fjords architecture [66] proposed by Madden et. al. and more recently the SwissQM project [76] at ETH Zurich, a single merged query was generated to pull data from the network for all user queries. Although the

above all-to-one mapping approaches are able to eliminate redundant data access among queries, due to the semantics of continuous queries, they either seriously suffer from fetching unnecessary data or resort to provide only approximate answers by relaxing the sampling period restrictions. We believe more careful study on the relationships of queries will provide more aggressive reduction on the data that need to be fetched from the network, without sacrificing the accuracy of the answers.

The Cougar project recognized the importance of energy-efficient data dissemination and query processing in the presence of multiple continuous aggregation queries [25, 108]. Unfortunately, the complexity of dynamic programming algorithms made it hardly practical for large-scale WSNs. Moreover, the authors' focus was on data, and they did not explicitly describe how to optimize multiple queries. Later, several algorithms were proposed to optimize multiple region-based aggregation queries, which could further be classified as equivalence class approach [110, 107] and partial aggregation sharing approach [28]. However, the above approaches are not general enough. They only tackle region based aggregation queries. This is quite limited, ignores many other types of queries, and is far from being enough to satisfy the wide applications of WSNs. Moreover, they are all constrained to a tree-structure data dissemination paradigm, while a more flexible structure such as directed acyclic graph can be more advantageous to facilitate sharing among queries.

A more careful introduction of these projects and all other up to date related works together with comparison with our approaches appear in section 2.5.

1.2.2 MQO in Mobile Sensor Networks(MSNs)

With the advances of technology, more powerful sensors integrated with mobility functionality have been developed [90, 75, 112]. The mobility of sensors effectively extends the sensors' coverage, by moving around to sense a larger area than its sensing range. This makes it attractive to deploy a small number of mobile sensors to monitor a large region which would have required a large-scale WSN comprising the first generation static sensors. Moreover, with the ability to freely move to desired locations, mobile sensors can contribute flexible network topology, react to the events of the environment and adapt to the changes in the missions. Thus, Mobile Sensor Networks (MSNs) comprising such mobile sensors have been increasingly adopted to support applications in surveillance, reconnaissance, and disaster rescue [40, 132, 56, 9, 96, 64].

In this thesis, we investigate MSNs that are sparse. MSNs are likely to be sparse because of two reasons: mobile sensors are more expensive compared to stationary sensors and the mobility of sensors increases their coverage [61], so it may not be wise to densely deploy a large number of them; moreover, dense MSNs can become sparse due to node failures caused by environmental hazards or even intentional damages (e.g. by adversaries in the battlefield).

In the context of sparse MSNs, the number of sensors is limited, the locations of mobile sensors are not fixed or known to each other, and the connectivity among sensor nodes may not exist. This makes it very challenging to discover the network topology, handle intermittent connection, coordinate distributed query processing, and facilitate query sharing. Thus, an intelligent MQO scheme is crucial in order to provide timely response

for data acquisition queries from the base station.

The desired MQO scheme for sparse MSNs should take into consideration of the following opportunities and factors:

- It should take good advantage of the mobility of sensor nodes. Mobility gives the sensors the freedom to move to any desired location, and thus it is possible for them to form a dedicated routing structure to facilitate query processing.

- It has to consider the balance between centralized and decentralized control. In a sparse MSN, a purely centralized approach (the base station coordinates the whole network) is not feasible, while a purely decentralized approach (every mobile sensor independently processes queries relying on pure local information without any centralized planning) might not be sufficient.

- It should exploit the connected and encountered sensors and make them intelligently collaborate in order to gradually improve the query processing on the way. To achieve this, distributed coordination and synchronization algorithms are required to deal with the problems caused by the intermittent connections, to effectively handle various encountering and disconnecting events.

- It should enable multiple queries to cooperatively share the precious available resources over space and time. Blind competition among queries has to be avoided, and adaptive sharing in the process of distributed decision making is desired.

Several studies have been carried out for data collection using mobile elements in sparse WSNs or mobile Ad-Hoc networks (MANETs) [87, 132,

38]. In these works, the focus is on general data collection and the time taken to deliver specific information is not critical. Our problem differs from them because data delivery in our system is driven by queries, and we have to minimize the time the sensors take to process these queries.

There also exist some works on multi-task allocation and path planning for cooperating UAVs, to minimize the task completion time [43, 12, 8]. However, these works do not consider the opportunity that we will exploit in this thesis, that is, some mobile sensors could be relocated to certain locations to relay information for others, to further reduce the query processing time.

1.3 Contributions

First, we summarize the major research contributions in this thesis. The objective of our research is to propose MQO techniques for WSNs, both for large-scale static WSNs and sparse mobile WSNs, in order to support the environmental monitoring and surveillance for a large area. Only with MQO techniques, the limited resources can be effectively utilized and shared to fulfill the potential of WSNs. In order to achieve this objective, our research is mainly divided into three parts.

The first part of the thesis focuses on processing multiple queries that are distributed to a particular base station, aiming to minimize the energy consumption inside the static WSN. We design a Two-Tier Multiple Query Optimization scheme, which is light-weight, adaptive to query arrivals/terminations, and supports both aggregation and data acquisition queries. More specifically, we address the following aspects.

1. We examine the cost model of query processing inside the MSN, and provide realistic cost estimation to guide the optimization process.

2. We design efficient and effective cost-based query rewriting algorithms for the base station tier, so that the abundant resources at the base station can be utilized and the redundant data requests that might be pushed to the wireless sensor nodes can be eliminated and filtered. Our algorithms are adaptive in the sense that queries can arrive and leave at any time, and our scheme is able to adjust to the changes automatically.

3. We propose several intelligent in-network optimization approaches, to reduce message transmissions and hence to save the total amount of energy consumption. To make it scalable to the number of sensor nodes, we adopt a distributed method where every sensor makes decision by itself instead of a centralized method where a central server computes and decides everything.

4. We evaluate the above algorithms and approaches that run at the base station and inside the sensor network, tune the necessary parameters and investigate their performance.

The second part of the thesis deals with the problem of performing multiple query optimization within a static WSN with multiple base stations, aiming to minimize the total communication cost among sensor nodes. To the best of our knowledge, this is the first piece of work to study this problem. For a large-scale sensor network, it is necessary and beneficial to have multiple base stations in the network. Firstly, it provides the sensor network with better scalability. The limited radio range of sensor nodes

leads in multi-hop routing, where the nodes nearer to the base station need to relay the messages for other nodes and hence become the bottleneck [69, 103]. Using multiple base stations can alleviate this problem. Secondly, it provides the sensor network with better reliability and efficiency [77]. The communication among sensor nodes are prone to failures, due to collision, node failure and environmental noise etc. With more base stations in the network, the average number of hops each data travels is fewer, and correspondingly the reliability and efficiency of the data transmission is better. Lastly, it extends the lifetime of the sensor network. The sensor nodes nearer to the base stations are likely to have higher load and the energy consumption there is faster than other nodes; with more bases station, the burden of nodes nearer to each base station can be relieved. Hence, to improve the scalability, efficiency and reliability of WSNs, we also make contributions in the following aspects.

1. We investigate an architecture where multiple base stations are utilized instead of a single base station, to relieve the burden of nodes nearer the base station and enhance the scalability of the whole network.

2. We design several similarity-aware query allocation schemes for region-based aggregation queries. These schemes intelligently put each query to an appropriate base station, so that multiple queries running at each base station can largely benefit from each other with underlying MQO schemes while each query is allocated to the base station incurring the least communication cost. We investigate the query allocation problem both in a static context where all the queries are known in advance and in dynamic environments with frequent query

arrivals and terminations.

3. We introduce adaptive query migration strategies to improve the query allocation on the fly, in accord with the changing patterns of the queries and the underlying wireless link.

4. We conduct an extensive performance study to evaluate the effectiveness of the above techniques in minimizing the communication cost of a large-scale WSN.

The third part of the thesis tackles the multiple query optimization problem in the context of sparse mobile wireless sensor networks (MSNs). We explore it as our third challenge in our thesis, because MSNs bring about attractive opportunities for applications while they are inherent with different challenges from its static peers. Posed by the challenges from mobility and sparseness, the problem of provide fast response to multiple data acquisition queries is handled as follows:

1. We propose distributed strategies in which the exploited mobile sensors strategically relocate themselves to proper locations to collaboratively facilitate efficient query processing and enable sharing over space and time.

2. We design intelligent algorithms to deal with the problems caused by intermittent connections and unpredictable topology, by means of effectively handling various encountering and disconnecting events among mobile sensors.

3. We define a a parameter *Coverage Ratio(CR)*, which reflects the sparseness of the network in respect to the number of queries, to guide the system to adaptively make a sound decision over the strategies.

4. We perform extensive simulation study to evaluate the proposed strategies.

The works in this thesis have resulted in a number of publications, more specifically, [118], [115] and [116], [117] and [119], and [113].

With the MQO techniques described in this thesis, the WSN should be able to reach a new level. Firstly, it should scale well to the number of concurrently running queries, and hence many users can access the network satisfactorily. Also, since the query sharing is enabled among similar queries and common work of several queries can be done only once, much unnecessary acquisition, computation, communication and retransmission due to collision can be avoided and hence the energy consumption of the whole network may be largely reduced. Moreover, with the introduction of multiple base stations and query allocation algorithms, the scalability of the sensor network should be largely extended, and the utility of the sensor network should be much better realized. Finally, in the applications where wireless sensor nodes are mobile and sparsely deployed, since our optimization strategies can effectively handle the challenges caused by intermittent connections, intelligently exploit the encountered sensor node and adaptively enable sharing among queries, the network should provide users with satisfactory data access despite of the mobility and sparseness. Hence, with our MQO techniques, many applications can be expected to be promoted by adopting wireless sensor networks.

1.4 Thesis Organization

Hereby, we outline the organization of this thesis. The rest of the thesis contains 5 chapters.

In Chapter 2, we introduce the background knowledge on WSNs and some fundamentals employed in our proposed schemes, followed by a survey on the related works.

Chapter 3 presents our solution, TTMQO, to efficiently process multiple queries that are running on a particular base station. TTMQO utilizes the base station to intelligently rewrite the queries to reduce redundancy, and then conducts in-network optimization to further enable query sharing among space and time in a distributed manner. Our experimental results indicate that our proposed TTMQO scheme offers significant performance improvements over the traditional single query optimization technique, in terms of both communication cost and scalability.

We then present our work on optimizing multiple queries in an infrastructure with multiple base stations in Chapter 4. The objective is to minimize the total data communication cost among the sensors, by intelligently allocating queries among base stations. We propose several similarity-aware query allocation schemes, both for static context and dynamic environment. Comprehensive experiments are conducted to show that our proposed schemes can effectively exploit the sharing among queries and greatly reduce the communication cost.

In Chapter 5, we tackle the problem of multiple query optimization in sparse mobile sensor networks. To provide fast response for data acquisition queries, several strategies are designed to exploit and share limited resources while dealing with the intermittent connections and unpredictable

topology. In addition, the most appropriate strategy is selected to adapt to the environment. Extensive performance studies show the effectiveness of our proposed strategies.

Finally, Chapter 6 concludes this thesis and discusses some directions for future work.

Chapter 2

Background and Related Works

In this chapter, we first introduce preliminaries and some fundamental overlay structures of wireless sensor networks, which are employed in our proposed schemes or some closely related works. Then, we focus on some related works. More specifically, we first present some representative studies conducted on important issues in single query processing over WSNs. Then, we review the most closely related works on multiple query optimization.

2.1 Background

2.1.1 Preliminaries

Queries. Two major types of queries are commonly used over WSNs: one-shot queries and continuous queries. In a one-shot query, sensors report their current data only once, and the observer gets a snapshot of current

state of sensor networks. One-shot queries are typically critical in latency, and the data need to be reported efficiently along the whole path. Our MQO work in sparse MSN, which will be presented in detail in Chapter 5 of this thesis, is processing this type of one-shot queries. In contrast, continuous queries require sensors to produce and report data periodically. As many sensor applications are interested in monitoring an environment over a long period of time, continuous queries are more frequently used. Our MQO works in WSN, which will be presented in detail in Chapters 3 and 4, are focusing on continuous queries.

To provide a declarative interface to observers, SQL-style query syntax is mostly used in current sensor network systems [125, 67, 69]. TinyDB defines queries [69] as in Figure 2.1:

```
SELECT          {agg(expression), attributes}
FROM            {sensors}
WHERE           {predicates}
EPOCH DURATION  {time interval}
```

Figure 2.1: Query format

The *SELECT*, *FROM*, and *WHERE* clauses are three fundamental clauses in a query. The *SELECT* clause specifies the observer-interested attributes or aggregates of sensor data, the *FROM* clause specifies the distributed relation of sensor type, and the *WHERE* clause specifies filters on sensor data. The *EPOCH DURATION* clause indicates the rate of sampling respectively. In continuous queries, sensor data can be viewed as a table with a single column per sensor type, and new tuples are appended to the tables when they arrive at the base station.

If a query is requesting the original sensory data, i.e., attribute specified in the *SELECT* clause, it is denoted as *data acquisition query*. On the other

hand, if a query is requesting the aggregates of sensor data, it is denoted as *data aggregation query*.

Energy. Power consumption is the major consideration of designing sensor network protocols. Hence, it is important to understand the energy consumption of various operations of sensors, to optimize the design of routing protocols and query processing strategies.

Madden et al. [69] conducted a study on the power utilization of major operations on Mica motes (a type of sensors designed in Berkeley) running TinyDB. The study shows that transmitting and processing consume the majority of energy. Processing consumes a large percentage of energy as the processor is always on in sensing, processing, and transmitting modes. However, in snoozing model, when both the processor and radio are idle, the energy cost decreases significantly. According to this study, to be energy-efficient, sensor networks should try to minimize the number of communication and sensing operations, so that the processor and radio can be idle for as long as possible. We take advantage of the above results when designing our approaches in Chapters 3 and 4.

Then, we introduce how to quantify the cost of message transmission. LEACH [39] gives a simple but recognized model on energy cost of sensor communication. To transmit a k-bit message a distance d, the energy cost of sender is:

$$E_{Tx}(k, d) = E_{elec} * k + \epsilon_{amp} * k * d^2 \tag{2.1}$$

where E_{elec} is the coefficient of running the transmitter or receiver circuit, and ϵ_{amp} is the coefficient of amplifying the signal so that the data can be received by the sensors with a distance d from the sender. To receive the

message, the energy cost of receiver is:

$$E_{Rx}(k) = E_{elec} * k \tag{2.2}$$

We adopt this model in this thesis, as reflected in section 3.3.2.

2.1.2 Overview of TinyDB

Since TinyDB [69] is one of the most popular query processing systems for sensor networks, we choose to base our multiple query optimization scheme on it. TinyDB is an application developed on top of TinyOS, an event-driven operating system developed for sensor networks [3]. TinyOS and TinyDB are designed for sensor nodes that have limited resources (e.g., 8K bytes of program memory, 512 bytes of RAM for Crossbow Motes). The TinyOS applications are implemented using a language called NesC, which is an extension to C. A sensor network has tens to hundreds of such stationary resource-constraint sensor nodes and a base station that acts as the central server and user interface.

TinyDB emphasizes on optimizing every single query [68]. Upon the arrival of a query, TinyDB parses the query and optimizes it by ordering sampling and predicates into a cost effective sequence. TinyDB then propagates the query into the network level by level down through flooding, during which a routing tree with the base station as root is constructed. After a sensor node has sampled the data at a scheduled time, it processes the data locally and may forward the data up to its parent, until the reaches the base station. For queries over constant attributes, a *Semantic Routing Tree* is utilized to direct the queries only to the nodes with the results instead of using flooding, thus saving the costs of propagation, execution

and result dissemination. For aggregation queries, detailed energy-efficient techniques such as communication scheduling and snooping have been proposed in TAG [67].

Although multiple queries can run simultaneously in TinyDB, it does not emphasize multiple query optimization. For example, although the Semantic Routing Trees are shared among multiple queries, sample acquisition is not. Thus, if two queries need a data reading even within a few milliseconds of each other, the sensor node will still acquire that reading twice. Moreover, there is no effort to optimize communication scheduling between queries, i.e. the message transmissions of one query are scheduled independently from another query. Our TTMQO scheme in Chapter 3 addresses the multiple query optimization issues not addressed in TinyDB.

2.2 Query Processing in WSN

In the recent years, there have seen a large amount of work in query processing techniques over sensor networks. Among them, in-network aggregation, data-centric storage systems, approximate techniques and adaptive techniques are the focuses in which many approaches have been proposed. In this section, we summarize and discuss these important studies conducted on query processing over sensor networks.

2.2.1 In-Network Aggregation Approaches

As data transmission between sensor nodes consumes more power than local processing, it would be very attractive if the volume of data transmitted could be reduced by local processing. In-network aggregation can

significantly reduce the data transmitted throughout the network by computing partial aggregate results at intermediate nodes. Five categories of energy-efficient in-network aggregation approaches are proposed in existing studies: cluster-based, chain-based, tree-based, multi-path-based and hybrid method.

Cluster-based approach is particularly useful for data collection in large sensor networks that require scalability to hundreds or thousands of nodes. LEACH[39] is a classic cluster-based energy-efficient protocol designed for sensor networks with continuous data delivery mechanism, with a fixed base station at a far distance. Cluster heads are elected by randomized rotation; member nodes use TDMA to send their data to the local cluster-head, cluster-heads aggregate the data from each sensor and then send this information to the observer node. By combining or aggregating the collected data at cluster heads, LEACH significantly reduces the amount of information to be transmitted. However, LEACH is not suitable for event-driven models, observer-initiated models as well as for mobile sensors. HEED [129] is an improvement on LEACH. It does not require node synchronization; moreover, it selects cluster heads according to the residual energy and node proximity to its neighbors or node degree, so that both energy and communication cost are considered.

A chain-based approach PEGASIS is proposed in [59]. A greedy TSP algorithm is used to construct a chain among the sensor nodes, and each node will receive from and transmit to a close neighbor. By exploiting the chain structure, further scheduling can be conducted so that most of nodes can be in sleep mode as long as they are not the senders or receivers at that time, and lifetime of the system can be significantly improved over cluster-

based approach. However, there are the following three main deficiencies. Firstly, the constraints of the logical chain make the nodes join in the chain in the later time expend more energy than required. Firstly, the constraints of the logical chain result in more energy expending for the nodes joining in the chain in the later time. Secondly, the latency will be large in constructing the chain and collecting the data through the chain, due to its linear property. Thirdly, it is prone to incomplete answers, for every link must be functioning in order that an integrated answer can arrive at the base station.

Tree-based approach is used in many sensor network systems such as TAG [67], Cougar [25], TinyDB [69], and PEDAP [101]. A spanning tree is constructed according to link quality [67, 69], query workload[25], node's energy level and communication cost between nodes[101], to relay and aggregate data from leaves to the root. To keep the spinning tree robust to the change in sensor networks, each sensor monitors the quality of links to its parent and neighbors. Whenever a sensor detects that the link to its parent deteriorates to some extend, it switches to a new parent with better link quality [131]. The advantage of tree-based approach is that the aggregation is computed in a straightforward way, with minimal communication cost. However, as the lost message drops the aggregation from the entire subtree, tree-based approach cannot guarantee the accuracy of aggregation, especially in high communication failure scenarios.

Ring topology is typically used in multi-path-based aggregation approaches [21, 78], in which sensors are divided into levels according to their hop count from the sink node. Sensors aggregate the data they receive from sensors in the upper level and broadcast the aggregate data to multi-

ple sensors in the next lower level, which are closer to the sink node. The ring-based multi-path topology significantly increases robustness because each reading is accounted for in many paths to the sink node. However, it also has some drawbacks: 1) For those duplicate sensitive aggregations, special techniques are required to avoid duplicate-counting; 2) Some duplications near to the edge are unnecessary, as the lost of message may not affect the accuracy of the overall aggregate result greatly.

Tributary-Delta [74] provides a hybrid method, which combines the advantages of tree-based and multi-path-based approaches by running them simultaneously in different parts of the network (Figure 2.2). This hybrid method is based on the observation that message losses close to the base station affect the accuracy of final results more than those close to the edge sensors. For this reason, tributary-delta uses tree-based topology at edge sensors, and uses multi-path-based topology at sensors close to the sink node.

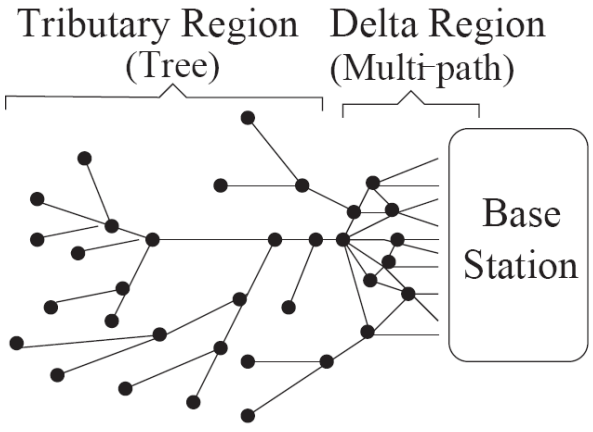


Figure 2.2: Topology in Tributary-Delta [74]

Tributary-delta [74] also presents an adaption approach for adjusting the balance between tree-based and multi-path-based aggregation topology in response to the changes of network conditions. It achieves a good trade

off between energy-efficiency and accuracy, by dynamically adjusting the aggregation topology (shrink or expand the delta region) to the current message lost rate. Tributary-delta provides two strategies to shrink or expand the delta region according to the percentage of nodes that should contribute to the aggregated results.

The topology adaptive techniques used by tributary-delta are suitable for the aggregation over the whole network. If queries are focused on some region located at the edge of network, when the network conditions change in this region, tributary-delta techniques will not be functional because they only consider the effect to the aggregation results on change of the whole network and ignore the change of some outlying regions.

2.2.2 Data-Centric Storage Mechanisms

In this thesis we are mainly dealing with the system architecture where the sensor network is composed of base station, sink and sensors. To get a full picture of how queries are being processed in the context of sensor networks, in this section, we present some techniques in peer-to-peer sensor networks.

Without a base station in sensor networks, sensed data is stored inside the network, which we call in-network storage. In-network storage techniques can be classified into two classes: local storage (LS) and data-centric storage (DCS). In LS, event information is stored locally at the detecting node upon detection of an event; queries are flooded to all nodes. This storage mechanism also works for the system architecture where there is base station. In DCS, after an event is detected the data are stored by name within the sensor networks; queries are directed to the node that

stores events of that name. In this way, query flooding is avoided, and the data only need to be aggregated once for multiple queries, which save a lot of communication cost at the query phase.

DCS is proposed by Ratnasamy et al. in Berkeley [81]. The authors develop GHT, a Geographic Hash Table system, that hashes events named with keys into geographic coordinates, and uses GPSR [44] to route the data to be stored at the sensor node geographically nearest the hash of its key as a key-value pair. This original GHT has been extended in [83] to hash to regions rather than to locations, to provide more robust service while requiring less accurate location information of sensors.

Much work especially in the domain of indexing has been done to efficiently support more complex queries, such as range query, historical data query inside DCS. DIFS [37] uses a multi-rooted quad-tree to index the spatial domain. Each child can have l parents, and each parent covers l times the area but indexes only $1/lth$ of the attribute value range of the child, and hence load balancing is achieved. A modified GHT is used to find an index node. Quad-tree is also used in DIMENSIONS [32] to support multi-resolution summarization, which is one generalization of DIFS. In many applications, data collected by a node may only be needed by other nodes that are nearby the data-originating node. In order to reduce unnecessary network traffic, this calls for hashing to locations that respect geographic proximity, and DIM satisfies such property [55]. The key of DIM is a clever construction of a locality-preserving mapping between the multidimensional attribute space and the spatial domain of sensors, in which data with values close to one another are hashed to locations nearby. It indexes multidimensional data with the help of zone tree, as in a k-d tree,

and each zone is uniquely identified by the path from the root to the zone. Fractional Cascading further improves DIM by explicitly placing the index for information close to where the information was generated [34]. It could be said as a combination of LS and DCS. Each node stores its own information locally, but also store information about data available elsewhere in the network, but in such as a way that a sensor knows only a "fraction" of the information from distant parts of the network, in an exponentially decaying fashion by distance. No nodes play a special role, thus avoiding the hot spot problems inherent with tree structure indexing as well, at the cost of some information duplication.

2.2.3 Approximate Techniques

Approximate techniques are important because they help to save a large amount of unnecessary communication while guaranteeing the accuracy and latency of query processing.

The sensory data gathered from neighbor nodes and those collected from the same node over certain time have strong similarity, which help to achieve high compression rate. The statistics on certain nodes can help to predicate the value without transmitting the sensory data. Beside these, uncertainty is an inherent property of sensory data. Study on these imprecise data with quality guarantee is also important because it affects query accuracy and communication cost greatly.

Compression Techniques

Approximate query processing techniques, such as histograms, wavelets, and linear regression, have been widely studied in database community.

However, these compression techniques need to be modified when they are applied to sensor networks, in which sensors are limited in computation ability and approximation is performed in-network. Here, we mainly focus on the compression techniques done within the sensor network, not the techniques that purely compress data over data streams at the base station.

Deligiannakis et al. [23] proposed a new technique for compressing multiple streams containing historical data from each sensor. It exploits correlation and redundancy among multiple measurements on the same sensor. The basic idea of this algorithm is to let each sensor extract and maintain a series of values called base signal as the approximate representation from real measurements, which capture prominent features of recorded data and are updated according to newly arrived data. Newly collected data are divided into several intervals, each of which is then mapped to the base signals to calculate the correlations. By recording the correlations and base signals, data are represented in a more compressed way. This algorithm is based on the observation that the values of the collected measurements have similar patterns over time, or that different measurements are naturally correlated.

SPASS, a Sharing and PAtitioning of Stream Spectrum protocol, was introduced in [7]. The main idea in SPASS is exploiting the similarity in the spectrum of nearby sensors, merging the spectra of sensors in the same group into one global spectrum, and transmitting one partition of the global spectrum by each sensor to the central sensor database. The spectrum is the range/distribution of values read by that sensor, and is collected by summary manager. Although continuous coordination between nodes are required, SPASS protocol can significantly reduce the per-sensor power

consumption by transmitting less data, and processing at the server side.

Shrivastava et al. [89] proposed q-digest, a distributed data summarization technique for approximate queries, and extended the class of queries (e.g., median) that can be answered inside the sensor network using limited memory. By accurately preserving information about high frequency values while compressing information about low frequency values, q-digest significantly saves bandwidth and power for both random and correlated data.

Lazaridis and Mehrotra [51] proposed an optimal on-line algorithm for constructing a piecewise constant approximate (PCA) of a time series which guarantees that the compressed representation satisfies an error bound. They also designed a sensor-side prediction algorithm with error guarantees, which is used to estimate the value of a time series ahead of time for some real-time applications. They combined the compression and prediction in a unified framework that avoids duplicates.

Prediction Techniques

TAG [67] provided a prediction technique, called snooping, which allows nodes to locally suppress local aggregate values by listening to the data reported by the neighbors and exploiting the semantics of aggregate functions. This technique can be applied to monotonic and exemplary aggregation such as MIN, TOP-K, etc, since a sensor can simply decide whether a particular local result may appear in the final result at the top of the network. However, snooping technique cannot be extended to some queries such as *COUNT*, *SUM* etc.

Deshpande et al. [26] proposed a model-driven data acquisition method

called BBQ which help provide answers by introducing approximations with probabilistic confidences. The models are used to optimize the acquisition of sensor readings by acquiring data only when the estimates made by the model are not sufficiently accurate to answer the query with acceptable confidence. The optimization concerns two factors: the statistical benefits of acquiring a reading and the system costs associated with wireless sensor systems. BBQ can help identify sensors that provide faulty data, and can extrapolate the values of missing sensors or sensor readings. When a particular link is not available during the execution, the topology model will be updated accordingly. To process "SELECT *" queries for sensor networks more efficiently, the authors further proposed an approximate technique called Ken [19]. Ken maintained a pair of predictive models (Gaussian Distribution) of each node's sensors, with one copy residing at the node and the other at the base station, and adopted a "push-based" approach to update the model at the base station. As data was routed towards base station, spatial correlations were further exploited by building disjoint-cliques models using greedy heuristic algorithm while optimal solution is proven to be NP-hard.

Model-driven approach presented above is effective in saving energy by avoiding acquiring data from sensors whose readings could be predicted or are unlikely to be in the result. However, it may be prohibitively expensive to answer complex queries with model-driven approach. A series of PROSPECTOR algorithms were proposed in [91] to optimize Top-K queries, by using samples of past sensor readings kept in the base station. Basically, the TAG [67] structure was assumed for routing. To reduce the numbers of messages intermediate level nodes need to transmit up, the tech-

nique *local filtering* was proposed. Correspondingly, to evaluate whether the result generated was acceptable, some additional readings were acquired and passed up to prove that others were in the top k, which was called *proof*. The problem of optimizing approximate top-k queries was thus formulated as a linear program, which complemented sampling well and was able to encode energy constraints, topology, and local filtering and proofs.

Y. Kotidis introduced the idea of snapshot queries, just requiring data from representatives to give an approximate answer to save energy [46]. Instead of forming a global model of the sensor network as BBQ [26], this framework captures localized correlations to build a local model. Each N_i maintains a data cache to maintain the values of the measurements of its neighbors, through snooping. Representatives are locally elected; N_i is said to be the representative of N_j if the difference between N_j's data value and N_i's estimated value for N_j is less than a predefined threshold. With the time series of neighbor's measurements, representatives are able to predict the current value and trend of neighbor's data, and hence significantly reduce the number of nodes to participate in the query. Moreover, with each sensor knowing about the data about their neighbors, for node failure, we can always find a node whose collected measurements is similar to the failed one to represent for it, and hence robustness is also guaranteed.

Quality Guarantees

Considine et al. [21] proposed a robust and scalable method for computing duplicate-sensitive aggregations across faulty sensor networks. This method combines duplicate insensitive FM sketches with multi-path routing techniques to produce accurate approximations to those duplicate-

sensitive aggregations such as *SUM*, *COUNT* etc., with low communication and computation overhead. It uses ring-based multi-path aggregation topology. The aggregate epoch is divided into a series of rounds, one for each level. In each round, the nodes at the corresponding level broadcast their sketches, and the nodes at the next level receive these sketches and combine them with their own sketches. In this way, the sink gets the final aggregate result from the sketches of its neighbors.

An energy-efficient spatiotemporal region queries processing scheme was introduced in [20]. since it is not practical to have a sensor node in each point of the monitored region, a query is satisfied as long as the query result is above a confidence threshold *ct*. A sensor's coverage is correspondingly defined as the area around the node in which the confidence of the sensor is above the threshold for every point in the area. They used a two-phase query processing approach. In the first step, they forwarded the query to the center of the queried region using greedy algorithms. In the second step, approximate query was processed inside the region and query result would be collected and sent back to the query initializer. By exploiting the redundancy among the sensors, they proposed two techniques to do approximate query processing: EFM and MSM. EFM is an energy-aware parallel flooding, where a node decides whether it participates based on its energy level and the status of its neighbors-open, skip or send; MSM is a depth-first method, which uses the completion of the query itself as a guide to traverse the region's nodes.

Sharaf et al. [88] proposed two novel solutions to balance the energy efficiency and quality of aggregate data in tree-based sensor networks. By assigning each sensor with a group id, they managed to construct the rout-

ing tree in such a way that the sensors need to transmit belong to the same subtree. With the introduction of a TOLERANCE clause, they imposed a hierarchy of output filters on the TAG/Cougar routing tree to suppress some data inside the network and hence achieved the reduction in the size and number of data messages sent to the central base station.

Uncertainty Management

Link failures and packet losses are very common in sensor networks because of environmental inference, packet delivery collisions, and low signal-to-noise ratios [131]. Fault sensors might also generate imprecise data. All these factors affect the degree of uncertainty between the query result and the actual data values. Uncertainty management is necessary to place more confidence in answers to the queries.

Cheng and Prabhakar [18] presented a framework to represent uncertainty of sensory data. They proposed a kind of probabilistic threshold queries, which require answers to have probabilities higher than a certain threshold value. The probabilistic threshold query can be used to eliminate data with small probability contributing to the final answer, providing a more precise mechanism in handing uncertainty. They also studied the techniques for evaluating queries under different details of uncertainty, and investigated the tradeoff between data uncertainty, answer accuracy and computation costs.

Lazaridis and Mehrotra [52] specified answer qualify requirements to queries, and showed that how the query evaluation system may do the minimal amount of work to meet these requirements. In [52], set-based uncertainty and value-based uncertainty are used to quantify data quality.

They proposed an energy-efficient processing technique for qualify-aware relational queries. Lazaridis and Mehrotra's work shows that if queries are willing to tolerate some loss of accuracy in the results they obtain from the system, then it is possible to drastically reduce the cost of query evaluation.

2.3 MQO in Traditional Databases

Since MQO is a classic problem in traditional databases and has been studied for many years, it is worthwhile to learn some techniques from traditional databases that may be adapted to our new environment. Generally speaking, as query optimization in traditional databases is shown to be NP-complete, the problem of multiple query optimization is also NP-complete [84]. Therefore, MQO is mostly studied using heuristics or probabilistic techniques, by exploiting the commonalities between queries. For example, Sellis proposed two heuristics in [85]: interleaved execution to make queries benefit from the intermediate result from others, and A* algorithm to progressively search for the expressions that can be shared with the highest probability.

More recently, some more theoretical approaches have been proposed for determining common sub-expressions from multiple queries. Chen et al. designed AND/OR DAG to convert the common subexpressions of queries into the common edges in the DAG [15]. However, AND/OR graphs specify an evaluation order, so some potential optimization choices are not considered. Moreover, the worst case complexity is proportional to the square of the number of nodes in the DAG, and hence it is quite costly to construct the AND/OR graphs. To conquer these weaknesses, Roy et al. proposed Multigraph in [82]. Being a unique, non-producural representation of mul-

tiple queries, Multigraph is not only suitable for identical and subsumption situations, but also for overlap situation. It also requires less time and space than AND/OR DAG. Using a multigraph requires much less time and space to process common subexpressions, and is adopted for MQO in mobile databases as well [72].

These approaches are meant for exploiting the commonalities of the queries from the same application. As the world-wide communication is more and more popular, inter-application sharing has also been studied. Manegold et al. designed an inter-application multi-query optimizer [73], which detected the commonalities among individually optimized queries from different applications and re-used previously computed intermediate results, by translating SQL queires into MIL programs and shrinking their Dependency Graph.

Although there have been abundant studies in MQO for traditional databases, all the above techniques are not directly applicable to WSNs, mainly due to the following three reasons:

1. WSN is a resource-constrained environment. With limited computational capacity and storage, sensor nodes cannot do complicated computation and data manipulation as the servers do in a traditional database. Besides, sensor nodes are connected by unstable wireless links, different from the stable wired LAN the traditional database resides in.

2. With data being ready to be processed, traditional MQO focuses on optimizing the physical level at the server to reduce its the computation and disk access, but the logical level optimization at the base station server is critical in our case in order to reduce the resource

consumption of wireless sensor nodes.

3. The queries for WSNs have different semantics, with one more "dimension" specified by the EPOCH DURATION clause, and hence the optimization strategies will be more challenging. In addition, with EPOCH DURATION clause, WSN queries are continuous queries which request data periodically for a long time, instead of one-time request in traditional databases.

2.4 MQO in Stream Databases

Nowadays, with the rapid increase of the number of internet users and applications, new information is coming as fast as a stream. To provide the users with real-time information they are interested in, continuous queries are popular in the context of stream databases. Since queries in WSNs are also continuous queries, we review, in this section, the studies on MQO in stream databases to see how they are dealing with the periodical needs of data from different queries.

Considering data are coming as a stream, unbounded, a general approach applied is to treat the data that come during a specific period of time as a unit (or called"window" to use a more technical term) and process the whole unit of data together. Because a query is generally composed of several operators, such as join, selection (whose conditions are called predicates), aggregation, we examine previous studies according to their emphasis on different operators.

Join, as one costly operator, is often optimized together with selections. Several studies have been done to share the selection (predicate

evaluation) in the presence of joins. In the project of NiagaraCQ [16], they represented queries as expression signature, and grouped the queries with similar signatures together. As long as the set of tuples flowing into that operator in both queries is always the same, two queries can share an operator. However, the sharing is limited: although it may be possible to share an initial selection, any operators which follow that selection must be replicated across all queries, which is quite costly. In [70], Madden et al. proposed a continuously adaptive optimization strategy to enable more sharing. They represented each query operator as one bit and annotated each tuple in the stream a bitmap and hence naturally used the linage of the tuple to explore the sharing adaptively. However, with each tuple travelled in the optimizer annotated separately, redundant annotation still exists. Most recently, TULIP project in [48] enabled sharing while reducing unnecessary work by adopting more precise check.

Stream aggregation, as one important operator that can give users a summarized view, has gained much attention recently. Arasu et al. [10] explored the problem of shared processing aggregates with varying non-periodic windows. Unfortunately, periodical sharing cannot be exploited, and on-demand results cannot be used for other queries as well. Being aware of the problem, Krishnamurthy et al. proposed on-the-fly sharing for stream aggregation in [49]. By identifying the common fragments of data results for multiple queries, much aggregation sharing can be enabled across time and space. This approach focuses on data, and hence is less constrained by query set and makes possible sharing among queries more general.

The MQO in stream database deals with processing continuous queries,

where streaming data has already been available at the base station. Bearing in mind the whole process of query processing in WSNs as described in Section 5.3.1, MQO in stream databases can be considered as a sub-problem of MQO in WSNs, that is, the part after the sensory data has been fetched to the base station. However, since MQO in stream databases does not deal with the problem of deciding which set of data to be collected at the required frequency from resource-constrained distributed-deployed sensor nodes, where the main challenge of MQO in WSNs lies, many special investigations to optimize queries inside the sensor networks are needed.

2.5 MQO in WSNs

MQO in WSNs is a relatively new research issue, and there are only a few studies mentioning MQO in WSNs. In the following, we examine all the up-to-date studies, classified by the strategies the optimization adopts: base station query rewriting and in-network result sharing.

2.5.1 Query Rewriting at the Base Station

As the base station is the interface of query processing, it has a global picture of all input queries. In addition, it has far more abundant resources and capabilities than sensor nodes. Therefore, it is very attractive to leverage the resources of base station to pre-process the queries before they are sent to the sensor nodes. As a result, several existing MQO works are focusing on query merging at the base station [66, 60, 76, 102, 130].

Madden et al. introduced the Fjords [66] architecture for the management of multiple queries. In this architecture, multiple queries over

the same set of data sources were put in a single Fjord, and thus it allowed the base station to combine multiple queries into a single plan, and explicitly handled operators with multiple inputs and outputs using grouping technique provided in NiagaraCQ [16]. However, this is not necessarily energy-efficient for sensor nodes by putting all queries into one single Fjord. In this way, although it can eliminate the redundant data acquisition and transmission among multiple queries, to satisfy all queries in the Fjord, it has to define the query conditions of the single merged query to be the loosest of all the query conditions, while the sampling period to be the Great Common Divisor of all the query sampling periods. This results in unnecessary data transmission and energy consumption, and it is sometimes even impractical because bandwidth-constrained sensor nodes could not afford such huge amount of data transmission. In particular, if the sampling periods of any two queries are relative prime, the sampling period of merged query will be 1, which is certainly undesirable. More recently, a similar approach was designed by Muller et al. [76], where a universal network query was formed to represent all user queries. However, to make the merged universal query affordable, the authors introduced a tolerant error in the sampling period up to some ε to relax the sampling period restrictions. Different from the above two pieces of work, in our approach that will be described in detail in Chapter 3, we use a many-to-many mapping instead of all-to-one mapping to make it much more flexible and energy-efficient, without sacrificing the accuracy of the answers.

Tabu search was employed to find an optimal merge order for a set of region-based data acquisition queries in an optimization scheme called TAMPA [102]. The set of neighbors was mapped from the set of adjacent

merging orders, and the gain function was derived from the cost difference between two queries that before merge and after merge. With the outputted merge order, the final set of queries was computed. Similar to our base station query rewriting, it is a many-to-many merging. However, Tabu search approach here is far more computationally complicated than ours. Moreover, apart from being able to support more types of queries, our approach is also adaptive to changing query set, while TAMPA is not.

One query aggregation framework, which focused on snapshot data acquisition queries with geographical information, was proposed in [130]. Since it was for snapshot queries, it focused more on reducing the query dissemination cost than the data delivery cost. After eliminating the queries which could be answered by the cached data, the base station would merge the remained queries to aggregated queries according to weighted overlapping zone, and assign one access node for each aggregated query. Then, each designated access node would analyze the corresponding aggregated query, decompose it and act as the local sink to locally process it. However, the framework is not applicable in our context, because different challenge lies in query merging for continuous queries.

In the above, all the query rewriting is actually query merging, and most recently, a query decomposition method has been proposed [60] for data acquisition queries. The authors proposed a cost model to estimate the similarity among queries, and designed an iteration based algorithm to rewrite the queries. In each iteration, a pair of queries with maximum similarity was chosen to be rewritten into two parts: one intermediate view for the shared part, and one new query for the remained difference for each original query. With a final integration, the answer for the original

query could be obtained. In this way, result sharing was achieved through shared intermediate views. However, although the view construction and query splitting worked well for MQO in traditional database systems, it is impractical for large-scale WSN. The resource constrained sensor nodes can only support a limited number of queries, and the increased number of queries as a result of splitting is worsening the situation. In addition, it will also incur more query dissemination messages, and engage the sensor nodes for longer time in the query/data message forwarding.

2.5.2 In-network Result Sharing among Sensor Nodes

Since the sensor nodes are suffering from bandwidth constraints and energy deficiency, it is critical to efficiently reduce the communication cost through query result sharing among sensor nodes. In this section, we review the distributed algorithms that are endeavoring to conduct in-network optimization among multiple queries.

The Cougar project recognized the importance of energy-efficient data dissemination and query processing in the presence of multiple continuous aggregation queries [25, 108]. They first examined the problem of selecting an optimal data dissemination tree to serve a set of queries [25], where they illustrated the importance by example, identified the challenge but without a concrete algorithm. Then, given a data dissemination tree, they proposed a hybrid push-pull model to minimize the number of messages for a given query workload [108]. There, each sensor node dynamically decides to actively push the data to some selected nodes (materialized views) towards the base station, or to reluctantly wait for the base station to pull the data when queries are executed, according to the query frequency and

sensor data update frequency. More specifically, dynamic programming was applied to draw an appropriate line in the data dissemination tree to divide the push-pull groups. However, due to the complexity of dynamic programming algorithms, it was hardly practical for a large-scale WSN. Moreover, their focus was on data, and they did not explicitly describe how to optimize multiple queries.

Later, a group of algorithms were proposed to optimize multiple region-based aggregation queries issued to the base station, which could further be classified as equivalence class approach [110, 107] and partial aggregation sharing approach [28, 120].

Instead of dealing with every single sensor as a data source, N. Trigoni et al. introduced the idea of Equivalence Class (EC) to reduce the complexity of the MQO problem in [110]. Moreover, they employed result encoding techniques that adopted linear reduction to send the basis instead of the full range of values to upper levels, and thus sent a minimum amount of data required to evaluate the corresponding queries. However, the routing is done separately for different EC, and hence no further optimization is allowed among different ECs in [110]. Being aware of this problem, they more recently improved their own scheme in [107], by integrating the routing together with the aggregation. In this case, more sharing can be exploited by enabling the aggregation wherever possible to reduce the message size. However, both the above two approaches do not support well the scenarios in which queries can be dynamically added in or cancelled. Because EC's representation is closely related to the number and distribution of queries, upon the update of query set, all representation of queries have to be consequently reconstructed with large overhead.

Emekci et al. designed a partial aggregation sharing paradigm [28]. More specifically, they classify edges in the query tree that intersect with query bounding boxes into *the incoming edges* and *the outing edges*, according to the flow of data aggregation. In this way, by treating the query bounding box as a black box, the final answer of a query can be obtained simply by subtracting the sum of the incoming partial aggregate values from the sum of the outgoing partial aggregate values. This approach was shown to efficiently reduce the communication cost than the approach that treated each overlap region as a separate group. However, it did not consider further partial aggregate merging opportunity in the network, and a fixed routing construction could not adapt to the query set to further enable sharing as well. An enhancement was done by Xie et al. in [120]. There, the query-based partial aggregate was revised into equivalence-class based partial aggregate, and further in-network merging at intermediate nodes was introduced to reduce the communication cost. Also, a fusion degree was defined to guide the routing path construction to be adaptive to the query set, to make good use of the proposed in-network merging.

However, all the above approaches are not general enough. They only tackle region based aggregation queries, while more types of queries need to be covered and optimized, including, for example, other types of aggregation queries (e.g., value-based aggregation queries) and data acquisition queries. Moreover, they all adopt a tree-structure data dissemination paradigm. On the other hand, the Directed Acyclic Graph adopted by our in-network optimization tier (that will be presented in Chapter 3), can be more advantageous to benefit from the broadcast nature of wireless communication.

So far, there have also been other works aiming to share query results among multiple queries in WSNs, although in different contexts. In the similarity-aware query processing techniques proposed by Xia et al. [114], to enable the sharing among similar ad-hoc queries, query results were cached to benefit future queries in data-centric storage sensor networks. However, it is different from our work where results are shared among concurrent long-running continuous queries. In another project in Duke University, Silberstein et al. exploited the combination of multicast technique and in-network aggregation to optimize many-to-many aggregations to achieve efficient in-network control of sensors [93]. Again, their in-network control application context is different from our query-driven data collection context.

Chapter 3

Two-Tier Multiple Query Optimization for WSNs

3.1 Introduction

Wireless Sensor Networks (WSNs) have been widely deployed in many applications [47, 71, 105]. As sensor nodes are quite resource-constrained (with limited processing capacity, storage, bandwidth and power), to better realize the potential of WSNs, several query processing techniques have been specially designed to collect and process the information in an efficient way [124, 93, 103, 24]. Most of the works study how a single query can be optimized.

However, it is often necessary to process multiple user queries simultaneously. Consider a traffic monitoring application as an example. Here, traffic officers may pose their continuous queries to obtain events of speeding, the traffic jam events, or the rough log of traffic in some area for further study. Many drivers may want to know the road traffic conditions somewhere between their current position and the destination, so that they can

decide which road to take to save time. At different times of the day, the number of drivers interested in the traffic conditions in different regions may vary tremendously. Hence, in this scenario, it is not an ideal solution to have all sensors provide continuous streams of their gathered data to satisfy the possible interests. On-demand traffic monitoring of wireless sensor network is necessary. The sensor database approach, which is adopted in this thesis, enables the sensor network to concurrently handle multiple user requests through running multiple queries.

In the multi-query situation, sharing of common operations among different queries could dramatically minimize the query processing cost. Unfortunately, most existing work has focused on the optimization and execution of a single long-running query. Consequently, when multiple queries are running simultaneously in a sensor network, they cannot benefit from each other by sharing their data acquisition, computation and communication cost. Moreover, running multiple queries in such an uncooperative manner will lead to bandwidth contention and even data loss as a result of transmission collisions (which may in turn require retransmission). Thus, in the resource constrained sensor network, it is critical to perform multi-query optimization in order to share the limited communication and computational resources.

In this chapter, we focus on a typical WSN which comprises a number of sensors and a single base station. We propose a Two-Tier Multiple Query Optimization (TTMQO) scheme to minimize the number of radio messages and the average transmission time in the sensor network. TTMQO is lightweight, adaptive to query arrivals/terminations, and supports both data aggregation and data acquisition queries which are defined in Figure 2.1

in Section 2.1.1. To study the effectiveness of the TTMQO scheme, we implemented it using TinyDB [69] and evaluated it under the TOSSIM [53] emulator. Our experimental results show that the TTMQO scheme can provide significant performance improvements, in terms of the cost of radio transmission and scalability with the number of queries.

The rest of this chapter is organized as follows. In Section 3.2, we give an overview of our two-tier optimization scheme. Then, Sections 3.3 and 3.4 present each tier in detail. In Section 3.6, we discuss the methodology and results of our experimental study. Finally, we summarize this chapter in Section 3.7.

3.2 Two-tier multiple query optimization

Since sensor nodes are resource-constrained, we endeavor to design a light-weight but effective scheme to support multiple queries running inside a wireless sensor network. In this section, we will first introduce and discuss our two-tier optimization philosophy, and then give an overview of the proposed optimization strategy.

The base station is the interface of WSNs. Users send queries to the base station and get query results from the base station. Moreover, the base station is usually much more powerful than sensor nodes, with abundant processing, disk, and memory capacity. Thus, we use the base station as a filter to reduce duplicate data access to the sensor network and as a screen to hide the query dynamics from the sensor network as much as possible. The objective here should be to save the energy at sensor side instead of minimizing the response time or computation cost at the base station.

Therefore, our scheme should address the following two sub-problems:

- **base station optimization**: Given a set of queries Q that have been submitted to the base station, optimize them into a new query set Q' to be injected into the wireless sensor network, such that the redundant requests among various queries in Q can be most eliminated. The optimal situation is that data results requested by queries in Q' will be just enough to answer all the queries in Q, and the same data needed for various queries in Q will be acquired only once from the network by queries in Q'.

- **in-network optimization**: Given the set of queries Q' that has been injected into the wireless sensor network, the sensor nodes collect and disseminate the result data intelligently to minimize the number of messages, and hence achieve bandwidth and energy efficiency. The ideal situation is that each sensor node just sends data only once to satisfy all the queries that need the data.

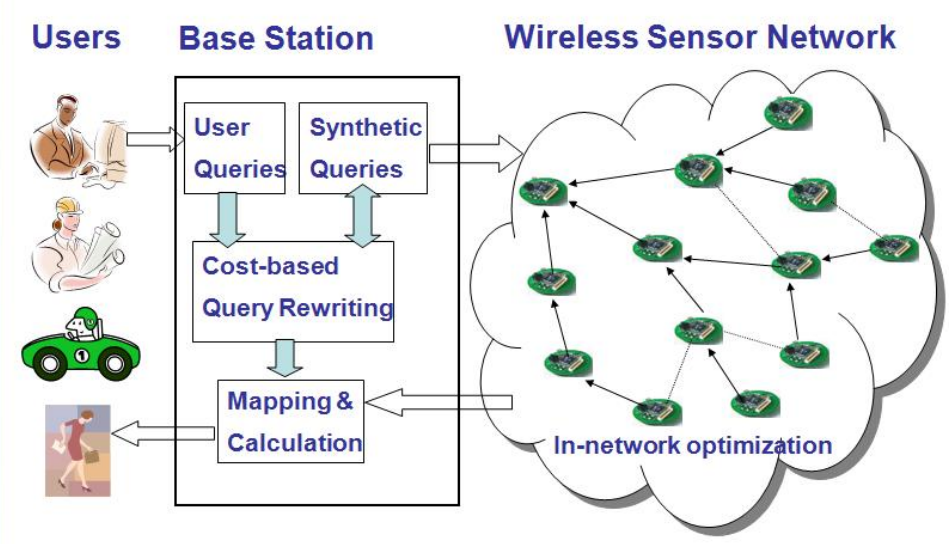


Figure 3.1: The System Architecture of Two-Tier Optimization in WSN.

Figure 3.1 shows the system architecture of our TTMQO scheme. In

our first-tier optimization performed at the base station, we adopt a cost-based approach to heuristically rewrite user queries into "synthetic" queries before injecting them into the sensor network, such that duplicate data requests from original queries can be eliminated as much as possible while guaranteeing the correctness of semantics of all queries. The optimization is also adaptive to query dynamics in that the set of running synthetic queries is continuously being updated by the arrival of new queries as well as the termination of existing queries. After the sensor network returns results for the synthetic queries, corresponding results for user queries can be easily obtained through mapping and calculation.

Our second-tier optimization is done inside the wireless sensor network. The main idea is to focus on the data required by all (synthetic) queries during specific time interval, and design a good DAG over the sensor nodes with the base station as the sink point to gather the queried data. Our algorithm further reduces the number of radio messages and saves the energy of sensor nodes in three ways. First, it schedules the communication among queries as a whole, which enables the combination of several query transmissions if these queries need data at the same time. Second, our algorithm dynamically determines the route to disseminate the query results, which enables data aggregation as soon as possible and involves fewer nodes. Finally, it tries to acquire and transmit the data to satisfy multiple queries if they need the same data, by taking advantage of the broadcast nature of the radio channel.

3.3 Base station optimization algorithm

As our base station optimization algorithm is based on query rewriting, we propose a cost model to measure the benefit of the rewriting. Then we propose a heuristic query insertion algorithm, which is guided by the cost model, to optimize the synthetic query set for each new incoming query. Finally, we look at how to re-optimize the synthetic queries when a query terminates.

3.3.1 Basic data structures

Let us first introduce some basic data structures of our *user queries* and *synthetic queries*. We store each user query in the form of $\langle qid, attribute\_list$ $|agg\_list, predicate\_list, epoch\_duration, qid'\rangle$ in a query table. qid is the unique identifier of the query. The $attribute\_list$ field contains the list of attributes that a data acquisition query qid retrieves from the wireless sensor network. agg_list is a list of $\langle operator, attribute\rangle$ that an aggregation query qid acquires. We note that for a single query, either $attribute\_list$ or agg_list will be empty. $predicate\_list$ is a list of $\langle attribute, min, max\rangle$. The qid' field is used by our algorithm to denote which *synthetic* query this query qid has been rewritten into.

As for a synthetic query, besides the above fields, a few more fields are used. (a) A *count* field is associated with the $epoch\_duration$ field as well as each entry in the various lists ($attribute\_list$, agg_list and $predicate\_list$), which denotes the number of user queries that require that piece of data. This is to facilitate the maintenance of the synthetic query when user queries terminate. (b) A $from\_list$ field contains the user queries which the synthetic query is responsible for. (c) A $flag$ field denotes the current

status of this synthetic query. (d) A $benefit$ field indicates the benefit that can be gained by the synthetic query (in comparison to processing the individual user queries). It is worthy to note that all these enhanced fields of the *synthetic* query are stored in the base station to help with query rewriting and further mapping and calculation, and they are not contained in the query propagation message.

3.3.2 Benefit estimation

Cost model. Moore's law suggests that the memory density and processor speed will continue to grow at an exponential rate. Thus, we expect sensor networks to continue to be bandwidth and energy limited. Since radio transmission is the most energy intensive operation a node performs, we use the cost of radio transmission as our performance metric.

Radio messages consist of query result transmission messages, query propagation and abortion messages, and periodical network maintenance messages. For continuous queries, result transmission messages dominate, so we only count the result message transmission in our cost model. However, to be realistic, we do include the effect of other radio message transmission into the cost of radio transmission in the experimental study.

For a query q_i, assume the length of its result message is $len(q_i)$. The transmission cost of a result message from one node to another can be estimated as $C_{start} + C_{trans} \cdot len(q_i)$, where C_{start} is the transmission startup cost and C_{trans} is the transmission cost of each unit of data. To measure the average transmission cost incurred by q_i for each unit of time, we have to estimate the number of per-unit time transmissions incurred by q_i, which is related to the number of result messages generated by the sensors as well

as the number of hops required to forward the messages back to the base station.

First, we look at the per-unit time number of result messages generated by a set of sensor nodes N_k, which is denoted as $result(q_i, N_k)$. At the end of each epoch of q_i, one result message would be generated by a sensor node whose readings satisfy the predicates of q_i. Therefore, we have

$$result(q_i, N_k) = \frac{sel(q_i, N_k) \cdot |N_k|}{epoch_i} \tag{3.1}$$

where $sel(q_i, N_k)$ is the selectivity of the query predicates over N_k, which is equal to the percentage of sensor nodes in N_k whose readings can satisfy the query predicates, $epoch_i$ is the epoch length of q_i.

Second, the forwarding hops of the result messages are determined by the message source nodes' location at the data routing tree. If a message is generated by a sensor located at the first level of the routing tree, then only one hop is required to forward the message to the base station. If the source is at the second level, then two hops are required and so on. Based on Eq. (3.1), the number of message transmission incurred by q_i can be estimated as

$$trans(q_i) = \sum_{k=1}^{max\_depth} result(q_i, N_k) \cdot k \tag{3.2}$$

where N_k is the set of sensor nodes at the kth level of the routing tree and max_depth is the maximum depth of the routing tree. Note that messages may be retransmitted due to transmission failures, such as collisions. Here we assume the number of retransmissions is proportional to $trans(q_i)$ and can be omitted in our cost model because only relative value is necessary to guide our query rewriting. Again, retransmission messages are dealt with

in our experimental study.

Eq. (3.2) provides an accurate estimation for acquisition queries where no in-network aggregation occurs. For aggregation queries, an internal node at the data routing path can forward aggregation values instead of the original detail values to reduce the number of message transmissions. Hence the actual number of transmissions would be a value within the range of $[result(q_i, N), trans(q_i)]$, where N is the whole set of sensors in the network. The lower bound value happens if each node that receives a result message also generates a result itself and can aggregate the received result with its own result, while the upper bound value occurs when no in-network aggregation can be performed at all. Unfortunately, there is no straightforward way to estimate this actual value. That is because the places where in-network aggregation occurs is hard to predict unless we make much stronger assumptions, which is undesirable. In this chapter, we just use the lower bound value. As we will see soon, this is conservative in that an aggregation query is integrated with an acquisition query only if it is guaranteed to be beneficial. Cost estimation is also not necessary for the integration of two aggregation queries.

Now we can compute the cost of a query $cost(q_i)$ as

$$cost(q_i) = trans(q_i) \cdot (C_{start} + C_{trans} \cdot len(q_i)) \tag{3.3}$$

Benefit estimation. If we integrate two queries q_1 and q_2 into one synthetic query q_{12}, to ensure correctness, all the data requested by q_1 and q_2 must be requested by q_{12}. In other words, the data requested by q_{12} is a superset of the data requested by q_1 and q_2. Semantic correctness constraints must be considered as well.

If q_1 and q_2 are aggregation queries (and hence q_{12}). In order to derive results for both q_1 and q_2 from the result of q_{12}, the two queries must have the same predicates. Hence, the integration of two aggregation queries in this way is guaranteed to be beneficial and hence we do not need to estimate their benefit.

For other integrations, we have to estimate their benefits. After integration, the requested attributes and predicates of q_{12} will be the union of those of q_1 and q_2, while the epoch duration should be the GCD(Greatest Common Divisor) of $epoch_1$ and $epoch_2$. We can estimate the cost of q_{12} by using Eq. (3.3). The benefit of the integration is estimated as $benefit(q_1, q_2) = cost(q_1) + cost(q_2) - cost(q_{12})$.

Statistics. To compute our cost function, we have to maintain some statistics. We use the reciprocal of the data rate of the sensor nodes (given by the sensor specifications) as the value of C_{trans}, while we periodically measure the actual average transmission startup time and use it as C_{start}. Another value to be estimated is $sel(q_i, N_k)$. To do so, at each level of the routing tree, we can maintain the data distribution, which is an independent problem studied in other literatures, such as [26]. In practice, to save maintenance cost, one can maintain one data distribution for multiple levels and assume the data distributions among these levels are identical. Since our focus is on multiple query optimization, in our experiments, we only use one distribution for all the levels, which actually biases against our techniques.

3.3.3 Greedy query insertion algorithm

Given a new query q_i that arrives at the base station and a list of currently running synthetic queries Q_{syn}, our greedy query insertion algorithm (shown in Algorithm 1) works as follows. If there is no synthetic query available, we directly add q_i in the synthetic query list. Otherwise, it searches for the most beneficial synthetic query q_{id} to rewrite with this q_i to produce a new synthetic query (lines 5–12). If q_{id} covers q_i (line 13), the newly added user query q_i will not have any effect on the workload in the sensor network. Otherwise, *Integrate(q_{id}, q_i)* is called to update the most beneficial query q_{id} into a new synthetic query. To identify that q_i is covered by q_{id}(as shown in line 6), we design the $Beneficial(q_i, q_j)$ function to return the benefit ratio instead of the original $benefit(q_i, q_j)$ defined in Section 3.3.2. More specifically, we divide the computed $benefit(q_i, q_j)$ by $cost(q_{ij})$. If there is no synthetic query that can be rewritten with the query q_i so that there are benefits, q_i is added into the synthetic query list. Upon the termination of the algorithm, if the synthetic query list is changed, corresponding query abortion and injection operations will be invoked to complete the whole process.

It is possible that synthetic queries can further benefit from the newly integrated synthetic query. Below shows a simple example to illustrate the situation:

q_1:select light where 280<light<600 epoch duration 2

q_2:select light where 100<light<300 epoch duration 4

q_3:select light where 150<light<500 epoch duration 4

For simplicity we assume all the sensor readings are satisfying uniform distribution and the value of $(C_{start} + C_{trans} * len(q_i))$ for any q_i is equal to

Algorithm 1: Greedy Query Insertion Algorithm

Input: one query q_i, the current running SynQueryList Q_{syn}
Output: A new running SynQueryList Q_{syn}

```
1  if Q_syn == NULL then
2  |  Q_syn.add(sqid); q_i.qid' ← sqid; UpdateCount(q_i, sqid, 1);
3  else
4  |  q_j ← Q_syn.next; max ← 0; id ← 0;
5  |  while q_j != NULL do
6  |  |  BenefitRate ← Beneficial(q_i, q_j);
7  |  |  if BenefitRate > max then
8  |  |  |  max ← BenefitRate; id ← j;
9  |  |  end
10 |  |  if max == 1 then break;
11 |  |  q_j ← q_j.next;
12 |  end
13 |  if max == 1 then
14 |  |  q_i.qid' ← q_id; UpdateCount(q_i, q_id, 1);
15 |  else if max > 0 then
16 |  |  Integrate(q_id, q_i); Insert(q_id, Q_syn);
17 |  else
18 |  |  Q_syn.add(sqid); q_i.qid' ← sqid; UpdateCount(q_i, sqid, 1);
19 |  end
20 end
```

1. Then we can compute:

$benefit(q_1, q_2) = d*(\frac{sel(p1)}{epoch_1}+\frac{sel(p2)}{epoch_2}-\frac{sel(p1\cup p2)}{GCD(epoch_1,epoch_2)})= \frac{d}{L}*(\frac{320}{2}+\frac{200}{4}-\frac{500}{2}) <$ 0, where L is the value range of light attribute and d is the average depth of a node in the routing tree (i.e. $d = \sum_k N_k \cdot k/|N|$). Under this situation, q_1 and q_2 will not be integrated, and both of them are directly added into the synthetic query list as q_1' and q_2'.

When q_3 is admitted, $benefit(q_1, q_3) = \frac{d}{L} * (\frac{320}{2} + \frac{350}{4} - \frac{350}{2}) < 0$, no integration with q_1'.

$benefit(q_2, q_3) = \frac{d}{L} * (\frac{200}{4} + \frac{350}{4} - \frac{400}{4}) > 0$, so we integrate q_3 with q_2':

q_2'':select light where 100<light<500 epoch duration 4

If we evaluate q_2'' against synthetic query q_1', $benefit(q_1, q_2) = \frac{d}{L} * (\frac{320}{2} +$

$\frac{400}{4} - \frac{500}{2}) > 0$,

so q_1' can benefit from new q_2''. The resulting query is :

q_1'': select light where 100<light<600 epoch duration 2

Hence, we need a more aggressive solution to remove the redundant data requests among user queries. To achieve this, after $Integrate(q_{id}, q_i)$ in line 16 in Algorithm 1 has updated the synthetic query q_{id} into a new one, we iteratively exploit further benefit by rewriting q_{id} with the current running synthetic querylist by calling $Insert(q_{id}, Q_{syn})$.

The *Beneficial* function first identifies whether two queries are rewritable based on semantic correctness constraints, and then computes the benefit ratio. The *Integrate* function modifies the synthetic query q_{id} so that all the data requested by q_i will be requested by the new q_{id}; it is also responsible for changing the values of the enhanced fields of the synthetic queries shown in section 3.3.1. The modification of the count fields upon insertion and termination is accomplished by an $UpdateCount$ procedure, with a flag to differentiate increment or deduction.

3.3.4 Adaptive query termination algorithm

To handle dynamic workloads where user queries may join or leave dynamically, we introduce a parameter α to adjust our query termination algorithm according to the property of application workload.

As shown in Algorithm 2, when the user terminates a query q, based on the information kept at $q.qid'$, it can easily get the synthetic query it was written into, which is denoted as sq_{old}. Query q eliminates its contribution in the synthetic query sq_{old} by *UpdateCount*. If the count of some field has been decreased to 0, it means that this query is the only query that

Algorithm 2: Adaptive Query Termination Algorithm

Input: one query q, the current running QueryList Q_{syn}
Output: A new running QueryList Q_{syn}

```
1  Find the synquery sq_old that q has been written into;
2  UpdateCount(q, sq_old, 0);
3  Remove q.qid from sq_old.fromlist;
4  if some count in sq_old has decreased to 0 then
5  |  if cost(q) > sq_old.benefit * α then
6  |  |  for All query q_i in sq_old.fromlist do
7  |  |  |  Insert (q_i,Q_syn);
8  |  |  end
9  |  end
10 end
```

requires sq_{old} to request some specific data. The termination of this query may trigger the reconstruction of the synthetic queries.

We hide the effect of termination of query q from the sensor network by keeping the old synthetic query sq_{old} unchanged, if the following condition is satisfied:

$$\frac{|sq_{old}.benefit - sq_{old}.benefit'|}{sq_{old}.benefit} \leq \alpha$$

where $sq_{old}.benefit'$ is the new $benefit$ value of sq_{old} after the removal of q. Since $cost(q)$ is equal to $sq_{old}.benefit - sq_{old}.benefit'$ according to Section 3.1.2, the condition can also be represented as: $cost(q) <= sq_{old}.benefit * \alpha$ (line 5). If such condition is not satisfied, we re-insert the remaining user queries contained in sq_{old} in the same way as the newly arrival queries (lines 6-7). α is a system parameter to tune the aggressiveness of query rewriting upon query termination. A good α value can avoid frequent query abortion and injection to the sensor network, which are also costly operations, without incurring too much extra data communication.

Moreover, when there are considerable similarity between queries, it is very likely that the query insertion and termination can be handled at the

base station, without affecting the sensor network.

3.4 In-network Optimization Algorithm

From the above analysis, we can see that the base station optimization is able to exploit the similarity among queries and eliminate the redundancy among queries through greedy query rewriting. It can also effectively hide query dynamics from the sensor side if the dynamics of the query does not affect the data set requested by the running queries. However, the base station optimization does not support sharing of the commonality among queries at the finest granularity. Since every query from the base station has the same meaning for each sensor node all the time, the base station optimization is a "all-or-nothing" approach. Moreover, base station optimization cannot take advantage of the special properties of sensor nodes, such as the broadcast nature of sensor radio transmission. Hence, we have our second-tier optimization inside the wireless sensor network, called in-network optimization, where sensors make local decisions by themselves and behave adaptively to the query workload with time.

3.4.1 Sharing Over Time

Consider two queries q_1 and q_2, whose only difference is their epoch durations. If the epoch duration of one query can be divided by that of the other (such as 2048ms and 4096ms), these two queries can be integrated into one according to the base station algorithm in Section 3.3.2 and thus the common result transmissions are shared. Otherwise (such as 4096ms and 6144ms), these two queries are sent into the network as two indepen-

dent queries because we are not be able to construct a beneficial synthetic query by using the greatest common divisor of the two epoch durations. However, in this case, half of the data requested by q_2 are also requested by q_1, which can be saved if we can schedule these two queries properly.

Based on the above observation, we exploit more sharing by scheduling the data acquisition and transmission of all queries in a whole. After a new query is propagated to the network, we (re)set the node's clock to fire at the GCD (Greatest Common Divisor) of the epoch durations of all the queries. The epoch start time for the new query on a sensor node is set to be divisible by the epoch duration (the smallest allowed epoch duration is 2048ms, and we assume that every epoch duration is divisible by it). In this way, the latency of the first epoch may be longer; however, for a continuous query, this extra latency for the first epoch is acceptable. On the other hand, by introducing such a little delay, various queries that have the same epoch duration will start sampling at the same time in every epoch, and hence can share sample acquisition. More specifically, when the clock is fired at time t, if there exists any q_i such that $t \bmod q_i.epoch = 0$, a shared data acquisition is conducted for all such q_is.

Since queries are injected into the network one by one, the sample rate of sensors may be reconsidered after every new query comes or a query has been stopped. But it is worthy to note that dynamically injecting new query or stopping a query at running time does not require any changes on other current running queries.

3.4.2 Sharing Over Space

After the sample rate has been set at each node, data will be retrieved periodically and transmitted out of the network to the base station. During the query result collection, we use the following optimization heuristics to aggressively share data over space. Each sensor node dynamically selects a route (parent) that is aware of the query space; in the meanwhile, it tries to take advantage of the broadcast nature of the radio channel to satisfy multiple queries in one message.

In TinyDB, a parent node is associated for each node based on the link quality [33], and hence a fixed routing tree is constructed, which is ignorant of the query space. In our scheme, we focus on the data that are required by queries during specific time interval. We let the source sensor node multicast/unicast the data along a DAG (Directed Acyclic Graph) with the base station as the sink point, and dynamically form the routing trees for various queries at the same time. The scheme works as follows:

Query Propagation Phase. Queries are flooded throughout the network from the base station. For a value-based query, flooding is necessary, because the accurate set of sensors that have data for the query are not pre-known to the base station and the set of sensor nodes can vary with time as well. Moreover, the query propagation cost occurs once per sensor query, which is comparatively negligible compared to result collection cost for continuous queries. If the query is a region-based query or a node-id based query, the set of answer nodes are known in advance, flooding is not necessary here, and some more efficient techniques such as SRT [69] and location-based routing [20] have been proposed. Here, we let every sensor to decide where to propagate to based on its local information about

neighbors.

When the query is propagated from node x at level i to level $i+1$, node x checks whether it has the data the query retrieves, and piggybacks this information down. In the meanwhile, the DAG is formed by having an edge from every node to each of its upper level neighbors (If the network is too dense and not all neighbors can be maintained, preference is given to the neighbors that also have query result to transmit). If the data at node x do not satisfy any query, x switches into sleep mode and will wake up after a predefined time. When it wakes up, if it finds that its current data satisfies a query, it sends a one-hop broadcast message so that its lower level neighbors would consider the node as an option to relay its data.

Result Collection Phase. When the data of a sensor node satisfy the predicates of any query that is triggered at the current time, the node will pack the data and select routes to forward them. Data acquisition queries and aggregation queries are processed independently, and hence the way they can share their common data in the network is different. For data aggregation queries, in-network aggregation at internal nodes is applied and each aggregation operator (such as MAX) is processed with a result message. Thus, one data message can be packed to share among all of the queries whose partial aggregation value are the same. For data acquisition queries, the sensor node generates a result message that contains the requesting attributes of all the queries whose predicates are satisfied. In this way, the message transmission can be shared among multiple queries, and would be further forwarded all the way along until the base station.

After the result messages are generated, each sensor node dynamically chooses a parent for each message based on local information. To intelli-

gently select a route to transmit data, each node keeps a list of its neighbors as what is done in TinyDB, but we also maintain the information about whether its upper level neighbor has data for each query, which was achieved through piggyback mentioned above. When a node x at level k has result messages for one or more queries, it checks whether there is a neighbor node at level $k-1$ that also has data for these queries. Neighbors with data for more queries have higher priority to be chosen. Ties are broken by favoring those nodes with more stable link with x. In this case, unicast message is sent to the chosen neighbor node to further forward or aggregate. Otherwise, if multiple neighbors are chosen (each is responsible for forwarding message for a subset of queries), one multicast message is required to send out the message to all these neighbors.

When an upper level node y receives a multicast message and it is one of the destinations of the multicast message, from the packet header, it identifies the set of queries that the message is for. It may perform necessary processing on the message (e.g. aggregate with its own data for aggregation query) and choose an upper level neighbor to forward the message. This procedure repeats until the message reaches the base station.

Discussion. In real applications, sensor readings are often spatially and temporally correlated, and hence the set of sensor nodes that have data retrieved by a query are likely to be spatially connected and temporally stable. When a node has a result message for a set of queries, it is very likely that one of its neighbors would also have one for those queries. Under the dynamic route selection strategy, such result transmission cost can be shared among queries. This is especially beneficial for data aggregation queries, whose common partial aggregation will continue to be aggregated

with other partial data at the upper level nodes to further reduce the radio transmission.

By using one result message $r(q')$ to answer multiple queries (such as q_1, q_2), the length of a shared message $len(q')$ may be larger than either $len(q_1)$ or $len(q_2)$. However, $len(q') <= len(q_1) + len(q_2)$ (many common fields in result of messages of q_1 and q_2 are in fact shared in $r(q')$, hence here the "=" will never be reached). Even assume the "=" is reached, according to the *cost* function studied in Section 3.1.2, C_{start} can at least be saved by result message sharing. Hence, here saving on the number of result transmission messages are guaranteed to achieve the saving on transmission cost. As for more accurate relationship, we study them in the experiment section with realistic parameters set in TOSSIM [53]. In the example shown in Figure 3.2, which illustrates our in-network optimization, we do not especially give parameters to show the performance gain in terms of result transmission cost.

From the above, we can see that much data transmission and energy can be saved by enabling sensor nodes to make intelligent local decisions: a sensor node only needs to transmit its data once to answer all the data acquisition queries; in-network aggregation is conducted sooner for data aggregation queries; the nodes that have no data to transmit can operate in a sleep mode to save energy.

In Figure 3.2, we illustrate the algorithm by a simple example. If two nodes are connected with lines, it means that they are within the communication range of each other. The dark solid lines denote the routing tree in TinyDB. Suppose D, E, F, G, H are queried by data acquisition query q_i, and B, D, G, H are queried by data acquisition query q_j, and both

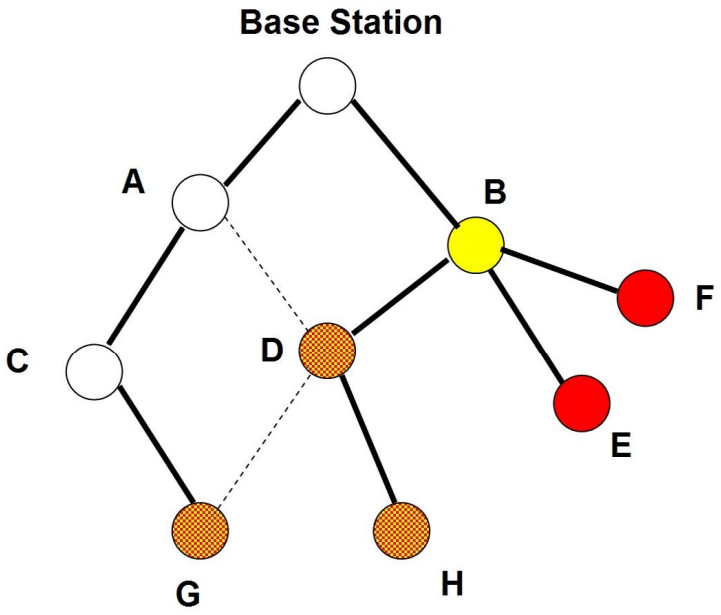


Figure 3.2: An example illustrating in-network Optimization

queries need data at time t. Using TinyDB, to answer q_i, all nodes will be involved. To answer q_j, nodes D, G, H will conduct sample acquisition again and intermediate nodes will relay their data twice. Hence, in total, 8 sensor nodes are involved, and 12+9=21 radio messages are transmitted. Using our DAG, G will choose D instead of C to relay for both q_i and q_j, and hence nodes C and A can be instructed to sleep. The data messages from nodes D, G and H can be transmitted only once to answer both of queries. Thus, a total of 6 sensor nodes are involved and 4+9=13 radio messages are transmitted.

For data aggregation queries, even more messages and energy can be saved. By dynamically choosing node D as the parent of node G, the aggregation for data at node G that is supposed to be done at base station is done sooner at D. Moreover, the aggregation from nodes G, H and D can be shared at D among q_i and q_j. Thus, even node B still needs to send one aggregated message representing q_i and q_j respectively due to the further

aggregation of data at E and F for q_i and data at B for q_j at B, only 7 out of 14 messages will be transmitted in total.

3.5 Discussion

Let q_1 and q_2 be two queries such that the epoch duration of q_1 is larger than that of q_2. Generally, base station optimization cannot effectively rewrite them to eliminate the redundancy among them under the following two cases.

1. The epoch duration of q_1 cannot be divided by that of q_2. In this case, there does not exist a beneficial synthetic query which integrates q_1 and q_2 under the constraint that its epoch duration must be the greatest common divisor of epoch durations of q_1 and q_2. So even though q_1 and q_2 request data from the same set of sensor nodes, due to the constraint of epoch duration, both queries must be injected into the sensor network.

2. The epoch duration of q_1 can be divided by that of q_2, but the selectivity of $q_2.predicate\_list$ is so much smaller than the selectivity of $q_1.predicate\_list$ that it will retrieve more unnecessary data than the common data that could be shared if integrating these two queries together.

On the other hand, in addition to the larger result message size incurred to enable sharing among multiple queries, our in-network optimization cannot effectively share the data required by two queries at the same time stamp under the following cases:

1. The queries are aggregation queries with different predicates. To guarantee semantic correctness, the node will process and transmit sensory data for each query separately.

2. One query is an aggregation query and the other is a data acquisition query, even if their epoch duration and predicates are the same. Without base station optimization, the base station is acting as a pure interface without processing the query, and it is expecting the network to do the aggregation and return the final answer, and the base station will not derive the answer for the aggregation query based on the results from other queries.

3. Both of them are data acquisition queries, but their predicates are different. Our in-network optimization algorithm dynamically designs the route to disseminate the query result, where the sensors without required data are less likely to be chosen to rely data for others, so that it affects as few nodes as possible and idle nodes may be put into sleep to save energy. However, in this way, it is possible that the same data will be forwarded by different node for different query, and hence the sharing of results are not in the finest grain.

From the above discussion, we can see that the base station optimization and in-network optimization are both similar and complementary to each other. Both of them can avoid the duplicate transmission of the same data for several data acquisition queries, although the base station optimization is somewhat more constrained by the epoch duration while the in-network optimization will result in a bigger result message size. In base station optimization, an aggregation query can benefit from both data acquisition queries or other aggregation queries; while in in-network optimization,

aggregation queries can only benefit among themselves with semantic correctness guarantee.

It is beneficial to apply in-network optimization after base station optimization. According to the complementary property, we can see that: by applying base station optimization first, the situations where in-network optimization is less effective can be partly eliminated and the size of result messages can be reduced as well; on the other hand, in-network optimization can enable the sharing among the queries where they cannot be rewritten in an effective way by base station optimization.

3.6 Experimental evaluation

3.6.1 Methodology

We have implemented our TTMQO scheme on top of TinyDB, the most popular query processing system for sensor networks. In our experiments, we used the packet-level TOSSIM [1] [53], an emulator for TinyOS-based sensor networks. For most of the system parameters, we use the default settings in TOSSIM, such as the message transmission speed, sensor reading size etc.

Our base station optimization consists of around 3,000 lines of java code; in-network optimization adds 2,200 lines of NesC code. The footprint size of our final image installed in the ROM of a node is 68KB (64KB for original TinyDB). It uses 4205 bytes RAM compared to the original 2846 bytes, including our increment of 256 bytes heap to 512 bytes to better support multiple queries. With rapid advances in memory technology, larger RAM

[1]The version of TOSSIM we adopt comes with the TinyOS package distribution TinyOS-1.1.10.

is easy to be achieved.

We assume that the sensor nodes are deployed uniformly in a $n \times n$ two-dimensional grid, with the base station node 0 at the upper left corner. The radio transmission radius is set to be 50 feet, while the grid spacing is 20 feet. In this work, we assume a lossless communication environment in which each node could transmit data to sensor nodes that are within its radio range. As a reference, we use the following strategy as the baseline for comparison: each query is optimized by TinyDB, and multiple queries that have been sent to the base station are all injected into the network to run concurrently without multi-query optimization.

As we have discussed in Section 3.3.2, the cost of radio transmission is our performance metric to minimize energy and bandwidth in the sensor network. The cost function there actually tries to measure the transmission time of the result messages. To be realistic, we count in the transmission time of all radio messages, which comprise result transmission messages, query propagation and abortion messages, network maintenance messages and retransmission messages due to transmission failure. More specifically, we report the *average transmission time* in our figures, which measures the average percentage of transmission time spent on each node for all running queries over the simulation time. The longer a node spends on waiting for the channel to be free, sending the message, and retransmitting due to collisions, the longer the transmission time will be.

3.6.2 Impact of optimization tiers

In this section, we study the performance gain we can achieve with each optimization tier, and verifies the similar and complementary relationship

between them against static workloads.

Referring to the Bisque benchmark [65], we construct three static workloads as shown in Figure 3.3. The $WORKLOAD_A$ is designed to focus on the (common) savings that can be achieved by both the base station optimization and in-network optimization; the $WORKLOAD_B$ is used to show the complementary of in-network optimization to base station optimization; the $WORKLOAD_C$ is designed to test the mutual complementary of these two optimizations. And all workloads can also evaluate the effectiveness of our two-tier multiple query optimization scheme.

Due to TOSSIM's inherent constraint in multihop routing, if the query needs data too frequently or too many queries are running in the sensor network, much data would be lost on the way. In order to make a fair comparison between our scheme and the baseline, we set the number of queries in each workload as 8, so that messages will not be dropped due to congestion and/or overflow of the communication queue. The biggest EPOCH DURATION specified in the workload is 20480ms (20s in binary time), and we set our simulation time as 3000s. Each query can be executed more than 150 rounds, which is sufficient to get stable performance results.

From the results in Figure 3.4, we can see that our optimization algorithms behave as what we have expected. For $WORKLOAD_A$, the base station optimization and in-network optimization algorithm are both supposed to eliminate the redundant data requests for similar queries, though in different ways. Compared with base station optimization, in-network optimization can more progressively share data requested over time and space, but it cannot enable aggregation queries to benefit from data acquisition queries in addition to its larger message size to support multiple

$WORKLOAD_A$:
```
SELECT temp EPOCH DURATION 10240
SELECT MIN(light) EPOCH DURATION 20480
SELECT MAX(temp) EPOCH DURATION 8192
SELECT nodeid,light,temp EPOCH DURATION 10240
SELECT MAX(temp) EPOCH DURATION 4096
SELECT nodeid,temp WHERE temp>700 EPOCH DURATION 20480
SELECT nodeid,light WHERE nodeid<10 EPOCH DURATION 8192
SELECT nodeid,light WHERE nodeid<15 EPOCH DURATION 8192
```

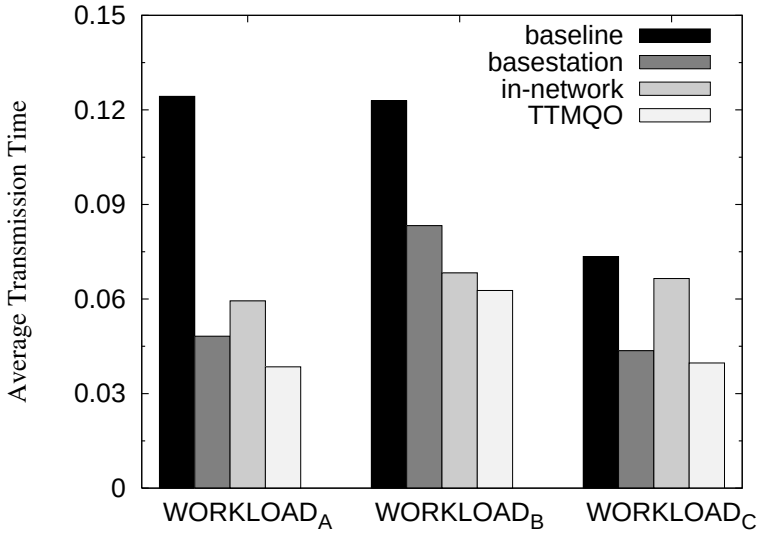
$WORKLOAD_B$:
```
SELECT temp EPOCH DURATION 8192
SELECT MIN(light) EPOCH DURATION 20480
SELECT MAX(temp) EPOCH DURATION 6144
SELECT nodeid,light,temp EPOCH DURATION 10240
SELECT MAX(temp) EPOCH DURATION 4096
SELECT nodeid,temp WHERE temp>700 EPOCH DURATION 20480
SELECT nodeid,light WHERE nodeid<7 EPOCH DURATION 6144
SELECT nodeid,light WHERE nodeid<15 EPOCH DURATION 12288
```

$WORKLOAD_C$:
```
SELECT temp EPOCH DURATION 8192
SELECT MIN(light) WHERE nodeid<15 EPOCH DURATION 20480
SELECT nodeid,light EPOCH DURATION 12288
SELECT nodeid,light,temp EPOCH DURATION 10240
SELECT nodeid,temp EPOCH DURATION 20480
SELECT nodeid,light WHERE nodeid<7 EPOCH DURATION 6144
SELECT MAX(temp) WHERE nodeid<15 EPOCH DURATION 8192
SELECT MAX(temp) WHERE nodeid<15 AND WHERE nodeid>8
EPOCH DURATION 10240
```

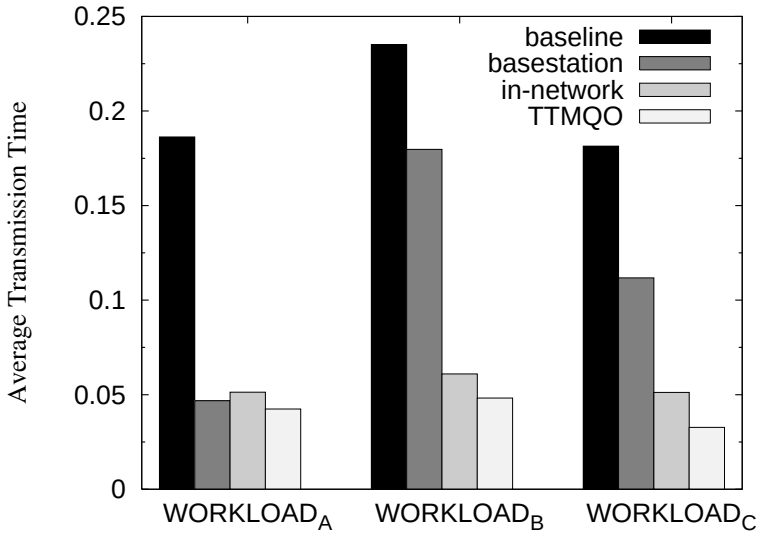
Figure 3.3: Static query workloads

queries. The average transmission time by the two tiers shown are quite similar, and have both been significantly reduced by up to around 61% and 75% compared with that of the baseline when the number of nodes is 16 and 64 respectively.

As designed, $WORKLOAD_B$ plays more on epoch duration to make the in-network optimization more effective than base station optimization. As shown in Figure 3.4, the average transmission time under in-network optimization is considerably smaller than that under base station optimization. Interestingly, the percentage of improvements by in-network optimization



(a) Number of Nodes=16



(b) Number of Nodes=64

Figure 3.4: Average Transmission Time

is much bigger in the network with 64 nodes than 16 nodes, compared with that of base station optimization.Since the number of radio messages for aggregation queries will not increase with network size while that for data acquisition queries will be proportional to the network size, the number of radio messages under in-network optimization grows much slower than that under the base station optimization, and consequently the per-

centage of improvement on number of radio messages increases faster. As we analyzed in Section 3.3.2, the average transmission time increases with the number of radio messages, and thus the percentage of improvement on average transmission time increases faster.

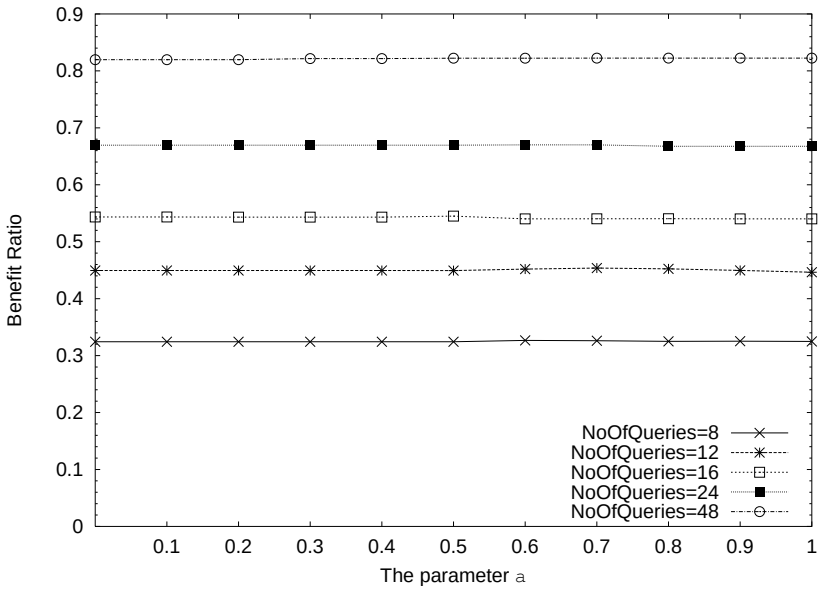
The results under $WORKLOAD_C$ (see Figure 3.4) show that the two-tier multiple query optimization performs much better than applying in-network optimization or base station optimization separately. It shows that the two tiers are mutually complementary, and it is beneficial to apply in-network optimization after base station optimization. In-network optimization does not support the similarity sharing among aggregation queries and data acquisition queries, but base station optimization can support it in the finest granularity. By applying base station optimization first, the aggregation queries whose answers can be derived from data acquisition queries are suppressed from injecting into the sensor network, so the in-network optimization will not face the problem of doing extra work to answer these aggregation queries; Moreover, with the common sharing that can be achieved both by base station optimization and in-network optimization enabled at the base station, the in-network message size will not be unnecessarily enlarged; on the other hand, the in-network optimization can effectively handle the situation where the queries cannot be effectively rewritten by base station optimization due to epoch duration constraint. It is also interesting to note that: when the number of nodes is 16, base station optimization is more effective than in-network optimization; while the contrary is true when the number of nodes has increased to 64. This is due to the same reason that applies to the scenario where there is fast increase in the percentage of improvement by in-network optimization as

network size grows which we have explained above. Our two-tier optimization scheme is shown to improve up to 82% in terms of the transmission time, which implies that it can save much bandwidth and energy.

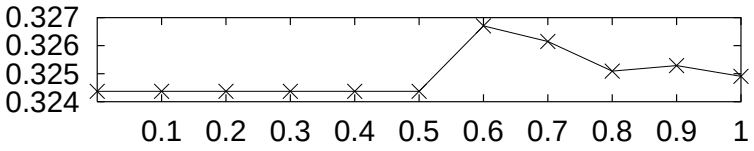
3.6.3 Performance under adaptive workloads

We evaluate the TTMQO scheme against various adaptive workloads. First, we evaluate the scalability of our TTMQO scheme with the number of queries and study the effect of parameter α with a model of queries that randomly select attributes (nodeid,light,temp), aggregations (MAX, MIN), predicates and epoch durations (from shortest 8092ms to longest 24576ms, all divisible by 4096ms). We keep the average arrival frequency at 40s per query, but we vary the average duration so that the average concurrent running queries is changing. A set of workload is complete after the termination of 500 queries. We use benefit ratio to represent the percentage of cost savings in Figure 3.5(a). Though we do not study skewed query workload, we expect the similarity to be greater among such workload, and the benefit can be even bigger.

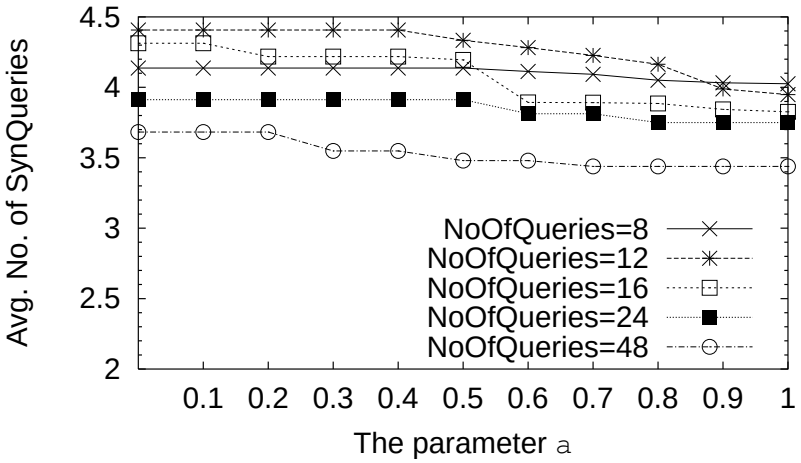
Given random queries, as we can see in Figure 3.5(a), the benefit ratio increases significantly from around 32% to 82% as the number of currently running queries increases from 8 to 48. Comparing with the effect of number of concurrently running queries, the parameter α has less effect on the benefit ratio. As shown in Figure 3.5(b) , when there are 8 simultaneous queries, the most benefit is obtained when α=0.6, which validates our analysis of Section 3.1.4. When α is too small, the significantly overlapped remaining queries may be forced to rewrite with other synthetic queries which may incur less benefit than original old synthetic query; on



(a) Benefit ratio



(b) Benefit ratio(NoOfQueries=8)



(c) The average number of synqueries

Figure 3.5: The performance against various parameter α

the other hand, when α is too big, unnecessary data fetched for previously-existed queries may incur so much overhead that it is better to rewrite the remaining queries.

Figure 3.5(c) shows that our scheme can scale pretty well with the number of concurrently running queries. The average number of synthetic queries running in the sensor network is less than 4 even when the number of concurrently running queries reaches 48. When the number of queries grows from 8 to 12, the average number of synthetic queries increases slightly. However, after the number of concurrently running queries is sufficient, such as 12 in Figure 3.5(c) given the random query sets, the average number of synthetic queries decreases instead. With sufficient queries, the probability that a piece of data is requested by multiple queries is higher, more savings are achieved by writing these queries into one synthetic query. As the value of α increases, the average number of synthetic queries slightly decreases, because bigger value of α favors keeping the old single synthetic query instead of rewriting the remained queries into a new synthetic query set whose number is generally bigger than 1.

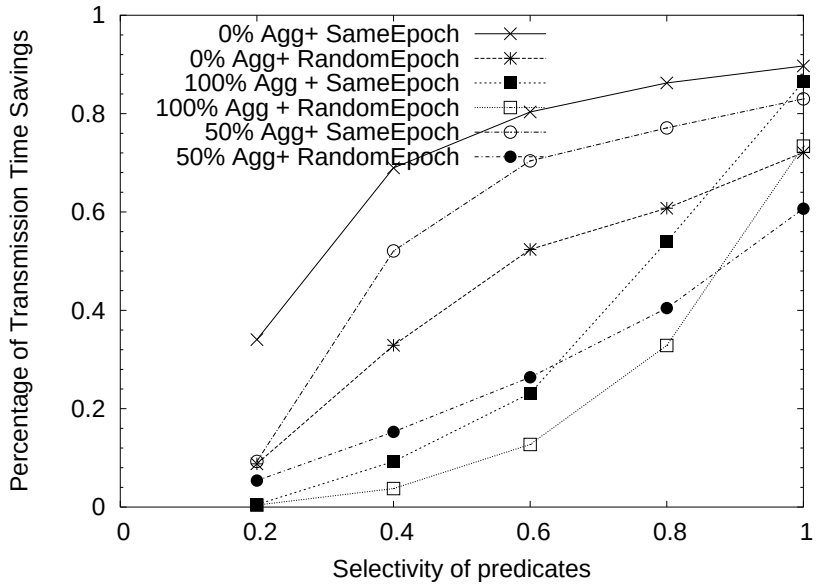


Figure 3.6: The percentage of transmission time savings against various predicate selectivity

Next, we further evaluate our TTMQO scheme against workloads with

various specific properties. More specifically, different composition of aggregation and data acquisition queries with predicates of different selectivity is utilized. In this experiment, average transmission time is computed until 500 queries have terminated and the number of concurrent queries is 8; data acquisition queries retrieve all the attributes; aggregation queries request for MAX(light); selectivity of predicates = 0.6 means that one of the attributes (nodeid, light, temp) is randomly specified in the query predicate with a range coverage as 0.6. Figure 3.6 shows that the percentage of transmission time savings grows with selectivity of predicates for all workloads, because there is higher probability that similar data can be shared among multiple queries when queries request more data, which also shows that similarity among queries with same epoch duration or different epoch durations are all exploited, and much savings are introduced by our TTMQO scheme. More carefully, we can see that the percentage of transmission time savings for data acquisition queries is generally higher than that of data aggregation queries. When the selectivity of predicates is 1, 8 data acquisition queries with the same epoch duration achieves around 89.7% message savings, which is even more significant than the theoretical value 87.5%, because less result message transmission required by TTMQO incurs less transmission failure and radio message retransmission. And, it is interesting to note that with 100% aggregation queries, there is a sharp performance improvement when the selectivity of predicates reaches 1. This is because two data aggregation queries with different predicates are processed separately due to semantic correctness constraints as discussed in section 3.1.2, and only in-network optimization scheme can reduce the number of messages by selecting proper routes to enable aggregation as soon

as possible and sharing data among queries when the value of their partial aggregation is the same. This is also the reason why the performance gain under 100% aggregation query increases quite rapid as the selectivity of predicates increases.

3.7 Summary

In this chapter, we have proposed a two-tier multiple query optimization scheme (TTMQO) to enable similar queries to share both communication and computational resources in the sensor network. It is a light-weight and general scheme, and supports both data acquisition queries and aggregation queries. We conducted intelligent optimizations at each tier which fully take the advantage of unique strength of that tier, and studied the relationships between the two tier optimizations. Our experimental results showed that the TTMQO scheme can provide significant performance improvements, with lower cost of radio transmission (average transmission time), and can scale well with the number of concurrently running queries.

Chapter 4

Query Allocation in WSNs with Multiple Base Stations

4.1 Introduction

In the previous chapters, we have seen several techniques being developed to efficiently utilize resources in WSNs, such as energy-efficient processing [39, 63, 92], in-network aggregation [125, 74, 93], approximate query processing [103, 26, 91] and multiple query optimization [110, 28, 118]. All these works focus on WSNs with a single base station.

However, for a large scale sensor network, it is necessary and beneficial to have multiple base stations. Firstly, it provides the sensor network with better coverage and scalability. The limited radio range of sensor nodes leads to multi-hop routing, where the nodes nearer to the base station need to relay the messages for other nodes and hence become the bottleneck [69, 103]. Using multiple base stations can alleviate this problem. Secondly, multiple base stations provide the sensor network with better reliability [77]. The communication among sensor nodes are prone to failures, due to

collision, node failure and environmental noise etc. With more base stations in the network, the average number of hops each datum travels is fewer, and correspondingly the reliability of the data transmission is better. Lastly, it extends the life time of the sensor network. The sensor nodes nearer to the base stations are likely to have higher load and the energy consumption there is greater than other nodes; with more base stations, the burden of nodes nearer to each base station can be relieved.

In this chapter, we study how to perform multi-query optimization within a WSN with multiple base stations. We assume that, once the queries are sent out to the WSN, the WSN can exploit the sharing of data communication among queries from the same base station to minimize the communication cost by using the existing approaches [110, 28] or our TTMQO approach presented in Chapter 3. Within this context, the allocation of queries to the base stations plays a critical role as it determines how much sharing can be exploited by the WSN.

To the best of our knowledge, this is the first piece of work to study how queries should be allocated to multiple base stations to minimize the total communication cost among sensor nodes. More specifically, we propose several similarity-aware query allocation algorithms to leverage the sharing among region-based aggregation queries. In a static environment where all the queries are known apriori, we approximate the problem of allocating queries to K base stations as a Max-K-Cut problem, and adapt a classical solution that uses Semidefinite Programming (SDP) relaxation to solve it [31]. In addition, to reduce the complexity of Max-K-Cut solution, we also propose a two-phase semi-greedy allocation framework which consists of a greedy allocation phase and an iterative refinement phase. In

many real systems, new queries may arrive and existing queries may terminate. This calls for adaptive query allocation techniques. To this end, we adopt incremental query insertion algorithms, which can quickly allocate a newly inserted query to an appropriate base station. We also introduce query migration strategies to adaptively re-allocate some existing queries to dynamically improve the query allocation on the fly. We conducted an extensive performance study and our results show that our techniques are effective in minimizing the communication cost of a large-scale WSN.

The rest of this chapter is organized as follows. In Section 4.2, we formulate our query allocation problem and point out the challenges to solve the problem. Then, we study the static query allocation problem in Section 4.3. To deal with dynamic query insertion and termination, in Section 4.4, our incremental insertion and adaptive query migration algorithms are proposed. In Section 4.5, we present our experimental results. We review some related work in Section 4.6, and finally, we conclude the paper in Section 4.7.

4.2 Problem Formulation

Consider a large scale sensor network that comprises K base stations and hundreds of (say N) sensor nodes. The base stations are powerful machines, with abundant processing, storage, and memory capacity and can be recharged easily. On the other hand, the sensor nodes are resource constrained, with limited processing capacity, storage, bandwidth and power. Thus, we focus on conserving the resources of sensor nodes. More specifically, we focus on minimizing the communication cost among sensor nodes, instead of that among base stations which is comparatively negligible.

To exploit the sharing among queries, one solution is to tightly integrate our query allocation work with a specific multiple query optimization scheme that runs at each base station, such as [118, 110]. This means the query allocation scheme has to be aware of the cost models used by the specific multi-query optimization scheme at each base station. In this chapter, to be general, we adopt a more flexible approach. To guide query allocation, we exploit the inherent sharing among queries without knowledge of the specific underlying multiple query optimization scheme.

4.2.1 Problem Statement

Our query allocation problem is defined as follows. Suppose there are K base stations and currently M queries are running in the sensor network of size N. For a query q_i running exclusively at a specific base station b_j, a number of radio messages will be transmitted to retrieve the sensory data to the base station. We refer to the number of radio messages as the communication cost. Let c_{ij} denote the communication cost incurred by a query q_i at base station b_j. We further denote the query set allocated to base station b_j as Q_j, and the amount of sharing (redundant requests) among these queries as S_j. Then the objective of the query allocation problem is to minimize the communication cost among sensor nodes in order to answer the queries. More formally, the objective function can be expressed as:

$$minimize \quad \sum_{j=1}^{K}(\sum_{q_i \in Q_j} c_{ij} - S_j)$$

If the multiple query optimization scheme at each base station does not exist, i.e., $S_j = 0, \forall j = 1...K$, the above optimal query allocation is easy to achieve in linear time. To minimize $\sum_{j=1}^{K} \sum_{q_j \in Q_j} c_{ij}$, we could just allocate

each query q_i to the base station b_j that incurs the smallest c_{ij}. That is,

$$q_i \in Q_j, \quad if \ c_{ij} = min(c_{i1}, c_{i2}, ..., c_{iK})$$

However, since multiple query optimization will be used at each base station, the introduction of S_j makes the query allocation very challenging. To get the optimal allocation, we need to get the optimal balance between minimizing
$\sum_{j=1}^{K} \sum_{q_j \in Q_j} c_{ij}$ and maximizing $\sum_{j=1}^{K} S_j$.

4.2.2 System Model

As we mentioned in Chapter 2, WSN queries are often classified as data acquisition queries and data aggregation queries. For data acquisition queries, intermediate nodes relay the result data for the nodes that are further from the base station. Without packet merging, the cost of transmitting data at a specific sensor node to a specific base station is equal to the number of hops between them. Under this case, the optimal solution for our query allocation problem is to split each query into several sub-queries according to the Voronoi Graph of the base stations, and assigns each sub-query to the base station whose Voronoi Region covers that sub-query. Hence, due to lack of research challenge, here we will not study the query allocation problem for data acquisition queries.

Thus, in this chapter, we adopt the following assumptions and simplifications of the system model.

First, we focus on region-based aggregation queries, such as SUM, MAX and AVG. More specifically, they belong to the category of *distributive* and *algebraic* aggregation queries, as defined in [67]. A Query region denotes

the geographical area of interest for the query. Thus, a region-based query specifies a fixed set of nodes that are involved in the query and hence simplifies the cost estimation and sharing estimation. Our method can be generalized to other queries as long as the sharing among queries can be estimated, e.g., by semantics exploration or maintaining a statistical model such as [26]. As it is an independent problem, we do not study it in this thesis. Also, in this chapter we assume one query region per query. If a user is interested in distant clusters of nodes, each of the clusters will be specified by a query region respectively, and correspondingly multiple queries will be issued.

Second, for each base station, a routing infrastructure (a routing tree or a routing Directed Acyclic Graph (DAG)) rooted at the base station is constructed. Each such routing infrastructure only involves those sensor nodes that are necessary to process the queries allocated to the corresponding base station.

Third, in-network aggregation [118, 110] with multi-query optimization is performed in the routing infrastructure.

Finally, it is assumed that there is a controller which maintains the query allocation information for all the base stations and optimizes the allocation of queries. Such a controller-based architecture is often used in load management within locally distributed systems which are under centralized administrations [121].

With the above assumptions, we can compute the cost function as follows. c_{ij} is computed by counting the number of sensors involved in processing query q_i (including those in the query region and those used to relay the message to the base station). This is because each sensor node only

has to send (or relay) one message (due to in-network aggregation).

To estimate the value of S_j, we keep bitmap m_j of size N maintained for base station b_j, whose default values are zero. If a sensor node x is queried by q_i, where $q_i \in Q_j$, we check the value of $m_j[x]$. If it is 0, we set it to 1. Otherwise, some other queries have already requested data from a sensor node x at base station b_j, and this cost is shared and correspondingly we add 1 to S_j.

Note that, if different parts of the network have different reliability in transmission, a weight factor should be assigned to each queried node to represent its reliability during the computation of c_{ij} and S_j.

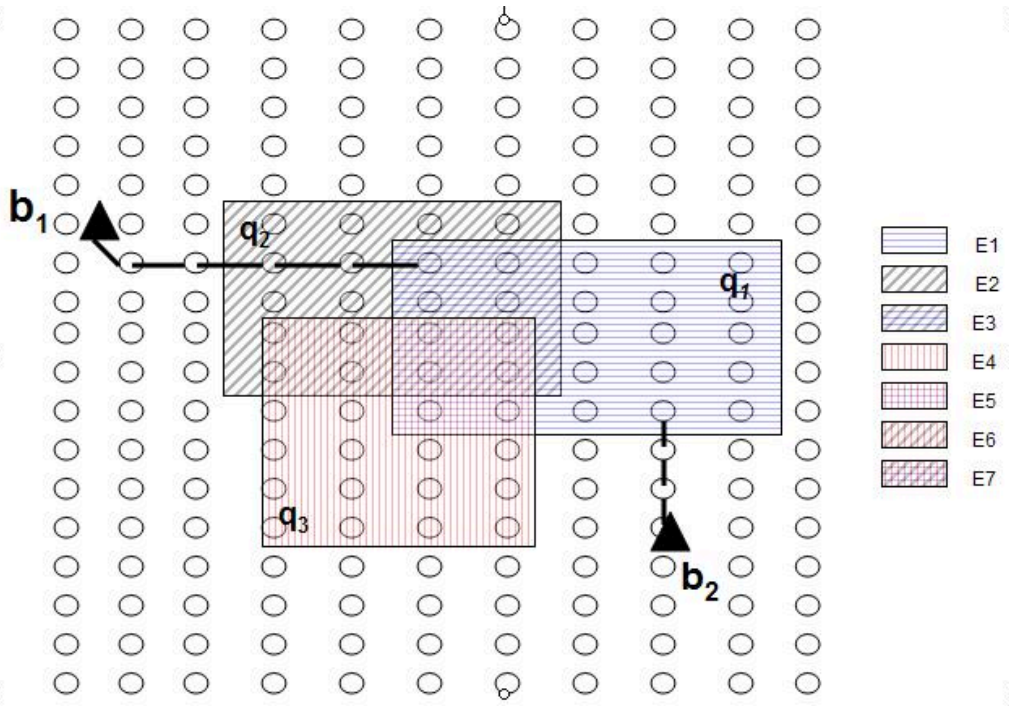


Figure 4.1: A scenario with multiple base stations and queries

Figure 4.1 shows an example of how the query cost c and sharing S are computed in our model. Each of the small circles denotes one sensor, while each rectangular region represents one query and each triangle denotes a

base station. For an aggregation query q_i assigned to a particular base station b_j, c_{ij} is computed as the sum of the area size of the query region and the extra cost incurred by relaying the aggregated result back to the base station. For example, as illustrated in Figure 4.1, q_1 covers 25 sensors and its minimal distance to b_1 is 5, we denote c_{11} as 30. Similarly, $c_{12} = 28$. If both q_1 and q_3 are allocated to b_1, the regions E_5 and E_7 can be shared, hence $S_1 = |E_5| + |E_7|$. It is worth noting that when q_1, q_2 and q_3 are all allocated to b_1, since E_7 has been shared twice, $S_1 = |E_3| + |E_5| + |E_6| + 2 * |E_7|$.

As a side note, since the objective is to minimize the resource consumption of the sensor nodes and the communication cost among base stations is negligible, one straightforward idea is to let the sensor nodes perform partial aggregation and leave the final aggregation to the base stations, instead of performing the whole calculation within the sensor network. To realize this approach, one can simply divide the network into non-overlapping regions (e.g., according to the voronoi graph for the base stations) and get each base station to take care of the region that is closest to it. Then, each aggregation query is divided into sub-queries based on the partitioning of the sensor network, and each sub-query is sent to its respective base station and correspondingly to its relevant sub-region. Finally, the partially aggregated results for sub-queries are further aggregated through communication among base stations to get the final result of the original aggregation query, without consuming energy among sensor nodes. From the above, we can see that partitioning approach forces the sharing within the subregions among different queries, and thus it maximizes $\sum_{j=1}^{K} S_j$. However, this partitioning approach is suboptimal for aggregation queries discussed in this chapter.

This is because partitioning a query may reduce the opportunities of in-network aggregation for the query and introduce extra cost incurred by relaying the partially aggregated results back to the base stations. Again, take q_1 in Figure 4.1 for example. If q_1 is partitioned, the relaying cost for both b_1 and b_2 cannot be avoided, and hence the total cost to execute q_1 will be 25+5+3=33, which is larger than the cost of allocating q_1 to either b_1 or b_2. Hence, the partitioning approach would result in a plan whose cost is higher than the similarity-aware non-partitioning approach and it is not ideal. The detailed performance of the partitioning approach is shown in Section 4.5.

4.3 Static Query Allocation

In this section, we examine the problem where all the queries are known beforehand, and they will be executed in a static context. That is, a set of queries need to be allocated to base stations, and there is no termination of existing queries and insertion of new queries. In this context, base stations can cooperate to generate a good query allocation plan based on the information from all the queries, with less consideration of the time complexity and flexibility.

4.3.1 Max-K-Cut approximation

From Section 4.2, it can be seen that, to minimize the communication cost among sensor nodes, the queries should be allocated to achieve the followings. First, similar queries should be assigned to the same base station so that overlapping data need not be transmitted multiple times. This can

reduce the energy consumption. Second, each query should be assigned to the base station that incurs the least communication cost. In other words, we could restate the problem as avoiding the following allocation as much as possible. First, similar queries are allocated onto different base stations. Second, a query is assigned to a base station that results in high communication cost.

To achieve both goals, we approximate the query allocation problem as a classical Max-K-Cut problem as follows.

An undirected graph G = (V, E, W) is constructed such that each vertex v_i represents either a base station or a query. There is one edge e_{ij} between each pair of vertices v_i and v_j. The weight of an edge e_{ij} is given by w_{ij}:

$$w_{ij} = \begin{cases} c_{ij} & \text{if} \quad v_i \in Q \quad and \quad v_j \in B; \\ -s_{ij} & \text{if} \quad v_i \in Q \quad and \quad v_j \in Q; \\ \infty & \text{if} \quad v_i \in B \quad and \quad v_j \in B. \end{cases}$$

where c_{ij} denotes the cost of executing query q_i on base station b_j, s_{ij} denotes the amount of common requests between query q_i and q_j, Q denotes the set of queries and B denotes the set of base stations.

The query allocation problem can then be expressed as a Max-K-Cut problem. That is, we partition V into K (the number of base stations) subsets, with each vertex allocated to exactly one subset. Formally, a partition P: P_1, P_2,...P_K defines an edge cut: $EC = \{e_{ij} | v_i \in P_l \wedge v_j \in P_r \wedge l \neq r \wedge 1 \leq l, r \leq K\}$. The Max-K-Cut problem is:

$$maximize \;\; w(p) = \sum_{e_{ij} \in EC} w_{ij}$$

One can see that, in the Max-K-Cut solution, if the value of c_{ij} is high,

then edge e_{ij} is very likely to be in EC and hence q_i is unlikely to be allocated to b_j.

We use $-s_{ij}$ instead of s_{ij} to denote the edge weight between two queries so that more similar queries will less likely be separated into different partitions. In addition, if both q_i and q_j are assigned to the same base station b_k, the cost of the common data s_{ij} is counted both in w_{ik} and w_{jk}. Since the sharing among the queries can be exploited by the sensor network, this over-counts the actual cost. By setting w_{ij} as $-s_{ij}$, the over-counted cost can be eliminated.

Finally, the edge weight of each pair of base stations is set to ∞, and hence different base stations are cut into different partitions.

SDP-K-Cut

Max Cut is a well-known NP hard problem, and Max-K-Cut is even more complicated. According to [11], there can be no polynomial-time approximation scheme for Max-K-cut, for any $k >= 2$, unless P = NP. Goemans and Williamson [36] significantly improved the approximation rate from 0.5 to 0.878 for Max Cut problem by using SemiDefinite Programming (SDP) as a relaxation. In [31], Frieze and Jerrum extended the work to solve the Max-K-Cut problem, and achieves the expected approximation rate of α_K, where $\alpha_K - (1 - K^{-1}) \sim 2K^{-2}lnK$. We apply the algorithm in [31] to solve our problem and denote the algorithm as SDP-K-Cut.

SDP-K-Cut is in fact a randomized heuristic algorithm using semidefinite programming relaxation that produces a K-partition which is on average provably better than the one produced by oblivious random partition. The challenge lies in how to model the variables which take one of

K values. This is done by allowing y_i to be one of K vectors $a_1, a_2, ..., a_K$ defined as follows: take an equilateral simplex Σ_K in R^{K-1} with vertices $b_1, b_2, ..., b_K$. Let $c_K = \frac{\sum_{i=1}^{K} b_i}{K}$ be the centroid of Σ_K and let $a_i = b_i - c_K$ for $1 \leq i \leq K$, with scaled Σ_k so that $|a_i| = 1$ for $1 \leq i \leq K$. By proving that $a_i \cdot a_j = -1/(K-1)$ for $i \neq j$, the Max-K-Cut problem can be formulated as follows:

$$\begin{aligned} IP_K: \quad & maximize \quad \tfrac{K-1}{K} \textstyle\sum_{i<j} w_{ij}(1 - y_i \cdot y_j) \\ & subject \;\; to \quad y_j \in \{a_1, a_2, ..., a_K\}, \quad \forall j. \end{aligned}$$

By replacing y_i by v_i, where v_i can now be any vector in S_{n-1} so that there are more freedom to partition the space, the max-K-cut problem can be relaxed into a semidefinite programming problem:

$$\begin{aligned} SDP_K: \quad & maximize \quad \tfrac{K-1}{K} \textstyle\sum_{i<j} w_{ij}(1 - v_i \cdot v_j) \\ & subject \;\; to \quad v_j \in S_{n-1}, \quad \forall j \\ & \qquad\qquad\;\; v_i \cdot v_j \geq \tfrac{-1}{K-1}, \quad \forall i \neq j. \end{aligned}$$

The K partitions can now be obtained after the following two steps:

1. Solve the problem SDP_k to obtain vectors $v_1, v_2, ..., v_n \in S_{n-1}$

2. Choose K random vectors $z_1, z_2, ..., z_K$. Partition V into $P_1, P_2, ..., P_K$ according to which of $z_1, z_2, ..., z_k$ is closest to each v_j. That is,

$$P_i = \{j : v_j \cdot z_i \geq v_j \cdot z_{i'}, for\ all\ i' \neq i\},\ for\ 1 \leq i \leq K$$

Implementation of SDP-K-Cut

To implement the SDP-K-Cut algorithm, the challenge lies in solving the problem SDP_K (Step 1). We adopt the convex programming form of SDP_K and use the SDPT3 [104], a solver for semidefinite-quadratic-linear programming developed by Toh et. al to obtain the result vectors.

More specifically, we denote $x_{ij} = v_i \cdot v_j$, and hence the SDP_K problem can be represented as:

$$\begin{aligned} CP_K: \quad & minimize \quad && \textstyle\sum_{i<j} w_{ij} * x_{ij} \\ & subject \ \ to \quad && x_{jj} = 1, \quad \forall j \\ & && x_{ij} \geq \tfrac{-1}{K-1}, \quad \forall i \neq j. \end{aligned}$$

Due to the constraint $x_{ij} \geq \frac{-1}{K-1}$ for $i \neq j$ instead of normal constraint $x_{ij} > 0$, we need to solve another linear programming problem together with the semidefinite programming problem. Since X is symmetric, we set $|V| * |V-1|/2$ linear constraints to especially deal with the situation for off-diagonal. Then, we can use SDPT3 to solve this problem. After we get result x_{ij}, since X is a symmetric positive semidefinite matrix, we can get vectors v by Cholesky factorization.

4.3.2 Semi-Greedy Allocation Framework

The above Max-K-Cut approximation is not ideal. Firstly, the high complexity of SDP-K-Cut solution in terms of time and space makes it impractical to be deployed if the number of queries to be allocated is huge. Secondly, although Max-K-Cut approximation well captures the general trend of the query allocation as can be seen from Section 4.3.1, it actually slightly biases towards assigning similar queries to the same base station.

This is because its underlying weighted graph representation is not sufficient to reflect the detailed dependencies among different edge weights, and thus cannot detect the same overlap that is recorded by several edge weights. For example, if some set of data is shared by three queries, and these three queries are all allocated to the same base station, by adding together all the weights of all the edges in the partition, the cost of the set of data will be deducted by three times with Max-K-Cut model while actually only two times should be deducted.

Hence, in this section, we will introduce a semi-greedy allocation framework for the static query allocation problem. The framework comprises two phases: *Greedy Insertion* and *Iterative Refinement*. The main idea is to make greedy local decisions to generate an initial query allocation plan, followed by a refinement step to iteratively adjust the whole plan.

Greedy Insertion

Firstly, we heuristically order the batch of queries before they are inserted in the sensor network one by one. The logic behind is: for the queries that are inserted earlier, we emphasize more on minimizing their own cost (less on maximizing the sharing) , while those that are inserted later, we take more advantage of their sharing with the earlier ones. We study the following two orderings:

- **Area**: Queries are ordered in descending order of the areas of their query regions. Hence, "BIG" queries are inserted first.

  In this way, queries with bigger regions will have the opportunity to be allocated to their smallest-cost base stations; smaller queries that are inserted later are more likely to benefit from such "big queries" since

the probability of finding an existing query that shares overlapping regions becomes higher.

- **Diff**: Queries are ordered in descending order of the values of function $\text{diff}(q_i) = c_{im} - c_{ij}$, where c_{ij} is the cost of the minimum allocation of q_i and c_{im} is the cost of the sub-minimum allocation. More formally, $c_{ij} = \min(c_{i1}, c_{i2}...c_{iK})$, and $c_{im} = \min(c_{i1}, ...c_{ij-1}, c_{ij+1}, ...c_{iK})$.

  The intuition is if the sub-minimum allocation of a query has a much higher cost than its minimum allocation, it implies that the query favors one base station much more than the others and hence we hope to allocate it first to achieve its minimum allocation.

Next, the queries are allocated to the base stations one by one in the sorted order as follows:

1. Estimate the additional cost ac_{ij} incurred by q_i at b_j, which is equal to c_{ij} subtracted by the amount of sharing between q_i and the other queries at b_j. This can be easily achieved with the bitmap m_j maintained for each base station b_j as we mentioned in Section 2.

2. Put q_i to the b_j with the smallest ac_{ij}.

Here we use the additional cost ac_{ij} because it reflects not only the amount of non-sharing region, but also the cost of executing q_i at b_j by itself.

Now, we have efficiently generated the initial query allocation plan. Note that this is only a plan and all queries have not been physically disseminated into the sensor network. So far, each to-be-inserted query tries to benefit from the previous queries. However, the previous inserted

queries cannot benefit from the later inserted ones if they are not allocated to the same base station, which we call "side effect". Although the order heuristics are supposed to relieve this "side effect", better solutions are expected.

Iterative Refinement

In this section, we introduce an iterative refinement phase to further optimize the query allocation plan generated by the greedy insertion phase.

Algorithm 3: Iterative Refinement Algorithm

Input: The initial query allocation plan $QB[0..M-1]$
Output: The refined query allocation plan $QB[0..M-1]$

```
1  SmallestCost ← CompCost ();
2  while True do
3      Count ← 0;
4      Changed ← 0;
5      while Count < M do
6          Qnode ← FindNextQuery (QList);
           /*Qnode =(qid, bid, costdiff);*/
7          reallocate q_qid from b_QB[qid] to b_bid;
8          TmpCost ← CompCost ();
9          if costdiff > 0 and TmpCost < SmallestCost then
10             SmallestCost ← TmpCost;
11             TempQB[0..M-1] ← QB[0..M-1];
12             Changed ← 1;
13         Count++;
14         Remove q_qid from QList;
15     if Changed == 1 then
16         QB[0..M-1] ← TempQB[0..M-1];
17         Restore QList to contain all queries;
18     else
19         QB[0..M-1] ← TempQB[0..M-1];
20         Break;
21 return QB[0..M-1];
```

The algorithm is shown in Algorithm 3. It runs in multiple iterations.

Within each iteration, it tries to refine the current allocation plan (lines 5-14). If a plan better than the current one is found, it will restore the current plan to be the best plan in this iteration and continue the refinement process by restarting another iteration (lines 15-17), otherwise it will stop as no more refinement can be found (lines 18-20).

In each iteration, one chance is given for each of the M queries in the whole query list $QList$ to reallocate itself, and hence it takes M *round*s. As shown in line 6, in each *round*, the function *FindNextQuery (QList)* examines all the choices of reallocating a query in the current $QList$ to another base station and returns the query q_{qid} whose reallocation results in the largest cost reduction $costdiff$. Note that the largest cost reduction $costdiff$ could be a negative value here. In line 7, we update the bitmaps of the base stations affected by the reallocation of query q_{qid}, and update the current allocation plan by setting QB[qid] to bid. Then, in line 8, we recompute the cost of the current plan QB[0..M-1] and store it in *Tmpcost*. If the current plan has a cost smaller than *SmallestCost*, the cost of the best plan we have visited, then it caches the current plan at *TempQB[0..M-1]* and set *SmallestCost* as the current cost (lines 9-12). Before we continue to start the next *round* to reallocate the next query, we remove the current query q_{qid} from the $QList$(lines 14). Note that if extra gain can be further achieved through reallocating q_{qid} again after the reallocation of other queries, it will be exploited in the next iteration.

It is worthy to note that, the algorithm still continues the refinement (lines 5-14) even if the current cost $TmpCost$ is larger than the one before the refinement. This is to capture the opportunities where performance gain can be achieved through the relocation of *multiple* queries altogether,

and allows the algorithm to jump out of the local optima.

4.4 Adaptive Query Allocation

In many of the real sensor applications, new users may issue new requests and existing users' demands may have been satisfied and their running queries terminate as well. This calls for incremental algorithms that are able to adjust to the dynamic context.

4.4.1 Incremental Insertion Algorithm

The SDP-K-Cut solution, which is designed to solve the Max-K-cut problem for a static graph, is not an incremental algorithm. Upon the insertion of each query, the SDP-K-Cut algorithm has to recompute from scratch instead of incrementally optimizing the new query set based on the previous status. Hence, it is computationally impractical to deploy the SDP-K-Cut in a dynamic context. For our greedy insertion algorithm presented in Section 4.3.2, the heuristic ordering is also impractical to be maintained in a dynamic context, since removing and reinserting the running queries upon the change of the ordering is too costly.

We choose to modify our greedy insertion algorithm to make it incremental and suitable for dynamic context. More precisely, we just keep the part that greedily inserts the arrival query into the base station which results in the smallest additional cost. In this way, our solution is able to efficiently find the best allocation for the new query, which incurs the least communication cost inside the sensor network, and does not affect other running queries.

4.4.2 Adaptive Migration Algorithm

With our current incremental insertion algorithms, when a query is inserted, we just do the best for the newly inserted query, but do not consider any re-allocation of running queries that already existed in the system to benefit from the newly inserted query. To deal with this problem and also to deal with the effect of termination of queries that often happen in real systems, existing queries may need to be re-allocated if necessary.

We note that migration during running time incurs overhead for the communication inside the sensor network. Taking the detailed query message dissemination mechanism into consideration [69, 118], for a specific query q_i to migrate from base station b_j to b_k, b_j needs to send its query abort message to the sensor nodes that are within the query region of q_i, which incurs cost c_{ij}; similarly, b_k needs to send query insert message for q_i at cost c_{ik}. That is, a one time cost of $(c_{ij} + c_{ik})$ will be incurred for the migration. If this particular migration improves the cost by Δc at each epoch, it takes at least $(c_{ij} + c_{ik})/\Delta c$ epochs for the migration gain to compensate for the migration cost. Also, reallocation of earlier queries later could create a lot of problems, such as unacceptable delay (by the query user), timeliness. Therefore, migration needs to consider the trade-off between the possible gain and the migration overhead.

It is also worthy to note that it is possible to cause temporary result missing or delay during query migration process if the epoch duration of the migrated query is less than the time to disseminate query abort and insert messages due to network congestion etc. For continuous queries, temporary result missing or delay is generally acceptable, and it can be alleviated by result interpolation or prediction as well. In case every piece of result data

is critical and timely delivery is required, this can also be achieved by instructing the sensors to delay the effect time of query abortion message until the arrival of the corresponding query insert message. Thus, while beneficial migration helps to improve the performance of the overall system, it will not incur severe problems to individual queries, such as unacceptable delay, timeliness and even starvation.

Below, we present the adaptive query migration techniques, which include the following two parts:

- A migration detection algorithm detects when it is beneficial to perform query migration. It considers the trade-off between migration gain and its overhead mentioned above.
- A migration algorithm selects which queries to be migrated. Basically, it greedily selects the most beneficial migrations.

Migration Detection

To detect when to perform the migration, the controller maintains some information of the current queries at the system. Recall that the controller maintains bitmap m_j (j=0,...,K-1) for base station b_j to denote whether a sensor is involved in the queries from b_j. Here, we extend m_j to be a countmap, which denotes the number of queries from b_j that request data from each sensor. Furthermore, the controller also dynamically keeps a M*K two dimensional matrix $a[\ ][\ ]$ to record the additional cost of allocating each query to each base station. For instance, $a[i][j]$ keeps the additional cost of allocating q_i to b_j (i.e. the value of c_{ij} subtracted by the sharing between q_i and the queries running at b_j).

To detect whether a query should be migrated, we associate a *pressure*

value p_i with each query q_i. In general, a higher p_i value represents a higher benefit to migrate query q_i. In particular, p_i is defined as $a[i][bid] - a[i][j]$, where bid is the id of the base station that q_i is currently allocated to, and j is the id of another base station b_j which satisfies $a[i][j] == MIN(a[i][0], ..., a[i][bid-1], a[i][bid+1], ...a[i][K-1])$. One can note that p_i is essentially the gain (of migrating q_i) that can be achieved at each epoch.

The migration detection algorithm is presented from line 1 to line 11 in Algorithm 4. It considers two factors. First, if the gain of a migration is high, the migration should tend to be performed. Second, if there is too frequent query arrival/termination, where the benefit for migration is not stable, migration should tend to be suppressed to avoid the thrashing effect and migration overhead.

We provide more details here. If there is a gain through migration (p_j is positive), the algorithm accumulates the gain over the epochs (lines 3-5). Otherwise, if p_j is negative and its accumulated value is positive, it means that other query insertion/termination has reduced the additional cost for q_j on the current base station or increased the additional cost for q_j to be reinserted into other base stations, during a short period of time, before q_j triggers migration. This suggests that there is frequent insert/termination of queries in the system to adjust the query allocation by itself, and hence we discourage migration by resetting the accumulated value of p_j to restart the accumulation and increasing parameter f_j to increase the migration threshold (lines 9-11). When the accumulated value has reached the adaptive threshold $f_j * p_j$, which suggests either the extra gain in each epoch p_j is big and/or the dynamics of queries is not frequent,

Algorithm 4: Migration Detection Algorithm

```
   Migration detection:
 1 for each data fetching epoch do
 2     for j=0; j<M; j++ do
 3         if p[j] > 0 then
 4             if f_j == 0 then f_j ← q_j.area/p[j];
 5             Accumulate[j] ← Accumulate[j] + p[j];
 6             if Accumulate[j] >= p[j] * f_j then
 7                 Migrate (); f_j=0; Accumulate[j]=0;
 8         else
 9             if Accumulate[j] > 0 then
10                 Accumulate[j] ← 0;
11                 f_j ++;

   Upon New Arrival of Query q_i:
12 for j =0; j<K; j++ do
13     a[i][j] = c_ij;
14     for all (x,y) in q_i.area do
15         if (x,y) of m_j >= 1 then  a[i][j] ← a[i][j]-1;
16 if a[i][bid]==MIN(a[i][0],...a[i][K-1]) then
17 Allocate q_i to b_bid;
18 Update m_bid and QB[i] accordingly; Compute p[i];
19 for all q_j overlaps with q_i do
20     for all (x,y) in q_i.area ⋂ q_j.area do
21         if QB[j] == bid AND (x,y)of m_bid == 2 then
22         a[j][bid] ← a[j][bid] − 1;
23         p[j] ← p[j] − 1;
24         if QB[j] ≠ bid AND (x,y)of m_bid == 1 then
25         a[j][bid] ← a[j][bid] − 1;
26         Recompute p[j];

   Upon Termination of Query q_i:
27 Update m_QB[i] accordingly;
28 for all q_j overlaps with q_i do
29     for all (x,y) in q_i.area ⋂ q_j.area do
30         if QB[j] == QB[i] AND (x,y)of m_QB[i] == 1 then
           a[j][QB[i]] ← a[j][QB[i]] + 1;
31         p[j] ← p[j] + 1;
32         if QB[j] ≠ QB[i] AND (x,y)of m_QB[i] == 0 then
           a[j][QB[i]] ← a[j][QB[i]] + 1;
33         Recompute p[j];
```

under the assumption that query workload and patterns in the past is similar to that in the future under most cases, we choose to trigger the migration (lines 6-7).

Now we present how to maintain the parameters that are required to implement the above migration detection algorithm. As shown in Algorithm 4, when a new query q_i arrives, we record the additional cost of allocating it to each base station (lines 12 to 15).

Furthermore, the q_i will also affect the optimal allocation decision of other existing queries. First, for another query q_j allocated to the same base station as q_i, if the countmap value for a sensor at (x,y) in their overlapped area is equal to two (line 21), it means that data at (x,y) which was exclusively requested by q_j before is now shared by both q_i and q_j. Hence, with q_i, the additional cost of q_j on its assigned base station b_{bid} decreases, the probability that other base station is better for q_j reduces, and we correspondingly reduce the pressure value p_j.

Second, for a query q_j at another base station that overlaps with q_i, if the overlapped area is exclusively requested by q_i at base station b_{bid} (line 24), the additional cost of q_j on b_{bid} (i.e. $a[j][bid]$) should be decreased. However, p_j may not increase as $a[j][bid]$ may not be the smallest among all the $a[j][\,]$ values. Therefore, we recompute p_j instead of directly increasing p_j (line 26).

Symmetrically, when existing query q_i terminates, the parameters are adjusted in a similar way, as shown in "Upon Termination of q_i" part of Algorithm 4.

Below, we illustrate by example the process of keeping track of information needed for migration detection, such as migration pressure $p[\,]$ and additional cost matrix $a[\,][\,]$. As shown in Figure 4.2, suppose queries q_1, q_2 and q_3 are allocated to base station b_2. Now:

1. Query q_4 arrives at the system. $a[4][1] = 12 + 1 = 13$, $a[4][2] =$

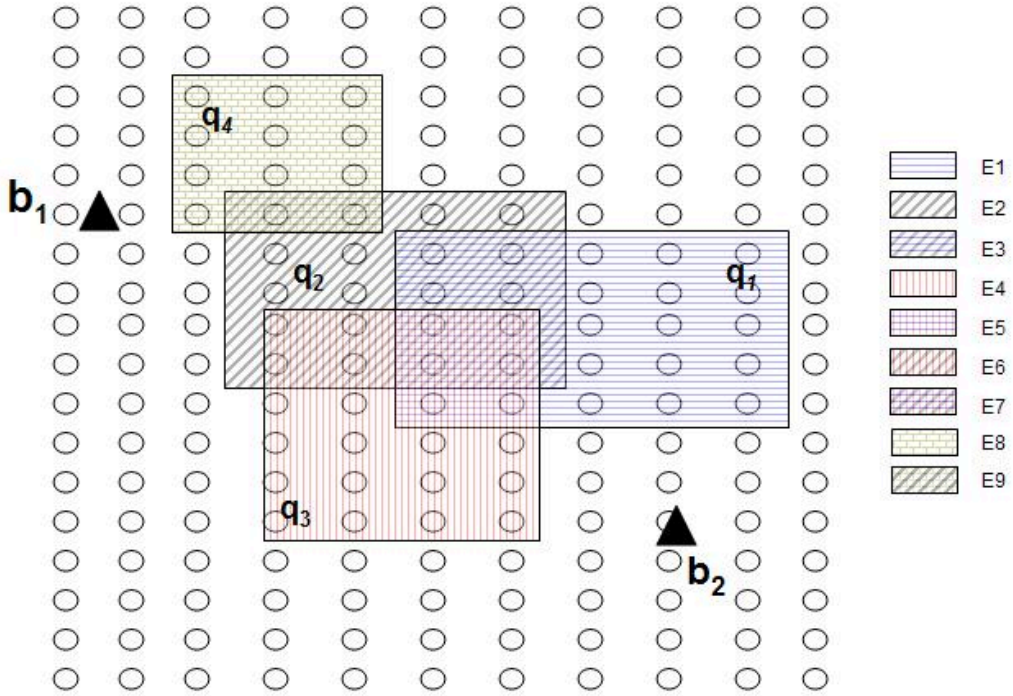


Figure 4.2: A Scenario to illustrate migration detection algorithm

$12 + 12 - |E9| = 22$, while $a[4][1] = MIN(a[4][1], a[4][2])$, hence q_4 is assigned to b_1, $QB[4] = 1$ and $p[4] = a[4][1] - a[4][2] = -9$. Since q_4 overlaps with q_2, $a[2][1] = a[2][1] - |E9|$, and $p[2] = p[2] + |E9|$.

2. Query q_2 terminates from the system. q_1 overlaps with q_2, and their previous sharing E3 is now exclusively for q_1, so $a[1][2] = a[1][2]+|E3|$, $p[2] = p[2] + |E3|$; similarly, for q_3, $a[3][2] = a[3][2] + |E6|$, $p[3] = p[3] + |E6|$. For q_4, $a[4][2] = a[4][2] + |E9|$ and $p[2] = p[2] - |E9|$.

Migration Algorithm

Once the above migration detection issues request to perform migration, the migration algorithm shown in Algorithm 5 will be run. Each round, through function *FindNextQuery* as we introduced in Section 4.3.2 we pick

the query that will result in the largest cost improvement to migrate. It is worthy to note that the migration here only modifies the query allocation plan by changing the information kept at the controller, such as *countmap* etc, and the intermediate plan is not disseminated into the sensor network. This migration process repeats until no beneficial query migration exists any more, and the final migration plan is disseminated into the network. In this way, local optimum can be achieved.

Algorithm 5: Migration Algorithm

Input: The initial query allocation plan $QB[0..M-1]$
Output: The query allocation plan after migration $QB[0..M-1]$

```
1 while True do
2     Qnode ← FindNextQuery (QList);
3     if costdiff > 0 then
4         migrate q_qid from b_QB[qid] to b_bid;
          /*Through changing the information kept at the
          coordinator, such as countmap etc.*/
5     else
6         Break;
```

4.5 Experimental Study

In this section, we shall present experimental results to show the performance of our schemes. We evaluate the algorithms by varying the number of base stations and queries, the average size of query regions and the average query arrival interval.

In the experiments, we assume N sensor nodes are deployed uniformly in a two-dimensional grid square. For every 100 sensor nodes, there is one base station at the center. Each query is expressed as a rectangular region $((x_1, x_2), (y_1, y_2))$, where (x_1, y_1) is randomly selected from any point in

the network, and the lengths on the x-axis ($x_2 - x_1$) and y-axis ($y_2 - y_1$) satisfy the uniform distribution distribution. We assume lossless communications among the sensor nodes, to evaluate the actual gain brought by our similarity-aware query allocation algorithms.

It is worthy to note that our query allocation method is general, and it is not constrained to distribution of sensors/base stations, the region shape of the queries and the assumption of lossless communication in the experiments. Through the cost estimation function, the properties of the network can be captured and the process of query allocation decision is the same.

4.5.1 Importance of leveraging query sharing

Firstly, we evaluate the importance of leveraging query sharing. We compare the performance of the following four strategies:

Random: Each query is randomly allocated to a base station.

Nearest: This is a naive strategy that leverages sharing among base stations upon query allocation. For each query q_i, we put it onto its nearest base station b_j, where b_j has the smallest minimal distance to the query region of q_i. In this way, since the number of sensors in the query region is fixed, it suggests that b_j also incurs the smallest allocation cost for q_i. As we discussed in Section 4.2.1, this is the best possible allocation without being aware of the similarity among other queries that have been allocated to a base station. Although it did not purposely take advantage of the sharing among queries, fortunately the region-based aggregation queries whose nearest base station is the same are inherently likely to have overlap with each other, and hence it inherently leverages query sharing.

Partition: This is the partitioning approach discussed in the last part of Section 4.2. The sensor network is partitioned into subregions and each base station takes care of the subregion that is closest to it. Each query is partitioned into sub-queries according to the partitioned subregions, and each sub-query is allocated to its respective base station. In this way, all the sharing within each subregion can be automatically captured.

Collection: it is a data collection approach, instead of a query-driven approach. Each sensor sends its raw data to its nearest base station.

Note that even though *Random*, *Nearest* and *Partition* are oblivious of the query similarity during allocation, queries allocated to the same base station may still benefit from data sharing through the multiple query optimization at the base station.

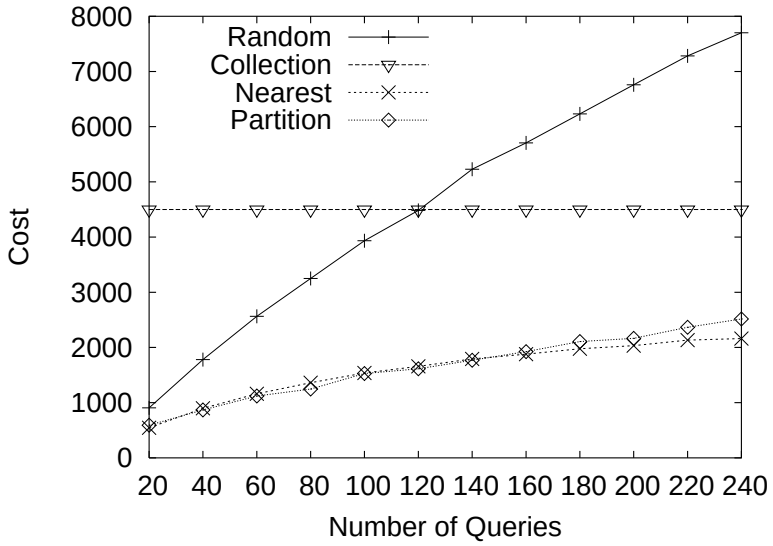


Figure 4.3: Communication cost over random queries with average QR=5*5, N=900

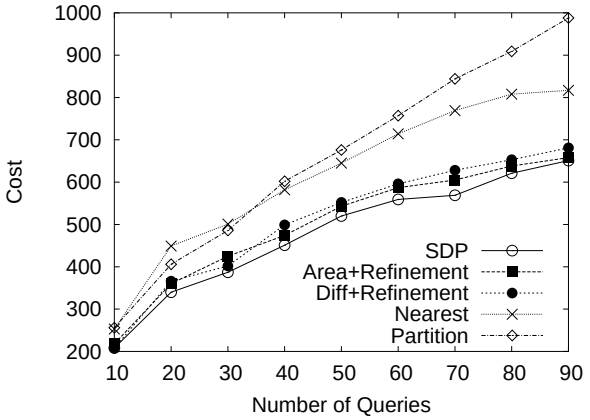
As shown in Figure 4.3, *Nearest* and *Partition* perform much better than *Random* and *Collection*, in terms of the communication cost (i.e., the number of radio messages). *Collection* scales well with the number of

queries, but its communication cost is generally much higher than necessary, because it is always fetching all the data from the network to satisfy possible queries (some of the fetching is not necessary for the current set of queries) and it cannot take advantage of in-network aggregation to reduce the transmission cost. Without cooperation among base stations nor correspondingly a good query allocation, *Random* cannot effectively make use of the underlying multiple query optimization scheme, and hence its cost grows linearly with the number of queries, which makes it unattractive for large scale networks. On the other hand, *Nearest* and *Partition* both scale well with the number of queries and incur low communication cost. From the above, we can see that it is necessary to leverage query sharing upon query allocation in large-scale sensor networks.

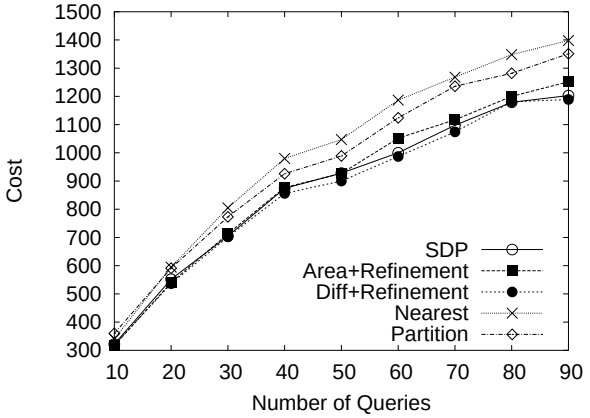
4.5.2 Performance in the Static Context

In this section, we compare the effectiveness of our similarity-aware strategies (SDP-K-Cut and semi-greedy query allocation algorithms) against the naive *Nearest* and *Partition*, in a static environment. Considering that SDP-K-Cut is not a scalable approach, our experiments are done in two parts: the first part is for small-scale scenario, with fewer queries and base stations, where we examine all the strategies in general; the second part includes large-scale scenario, where we show the effectiveness of each of the heuristics/phases in our semi-greedy query allocation algorithms. Each of the results shown below is the average result after 10 runs.

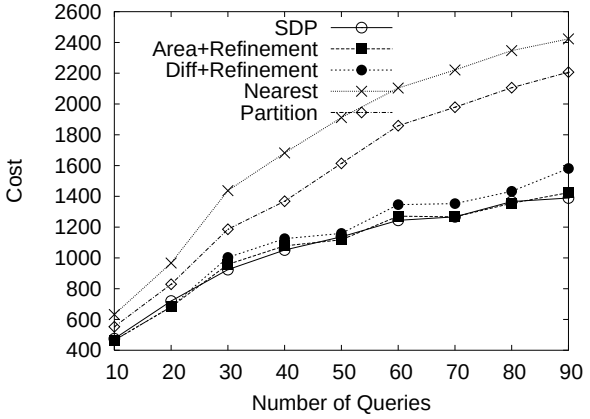
From the experimental results in Figures 4.4, we observe that neither *Nearest* nor *Partition* always outperforms the other but both strategies perform worse than our similarity-aware schemes. This is expected be-



(a)N=400, QR=5*5



(b)N=900, QR=5*5



(c)N=900,QR=8*8

Figure 4.4: Communication Cost over Random Queries in small network

cause both of them excel in one aspect but neglect the other aspect as explained here: for *Nearest* strategy, although it minimizes the relaying

cost of each query and has the tendency to assign similar queries to the same base station, it does not purposely take advantage of the sharing that can be exploited among queries; for *Partition* strategy, on the the other hand, although it maximizes the sharing among sub-queries, it sacrifices the benefit of in-network aggregation and introduces considerable relaying overhead for each partitioned query. Hence, as shown in Figure 4.3 and 4.4, when the number of queries is small (such as 10), *Nearest* outperforms *Partition*, because there is not enough sharing among the limited number of queries for *Partition* to counteract the overhead of partitioning. As the number of queries grows and overlaps among queries increase, *Partition*'s strength in enabling sharing among queries is revealed and hence *Partition* slowly outperforms *Nearest*. But when the number of queries continues to grow, *Nearest* can benefit more from the inherent overlap among queries allocated to the same base station. Furthermore, the relaying overhead of *Partition*, which is proportional to the number of partitioned queries, becomes huge. Therefore, *Nearest* outperforms *Partition* again.

All the similarity-aware schemes (SDP-K-Cut and our semi-greedy query allocation algorithms) result in a significant reduction in communication cost. This suggests that our similarity-aware schemes can all effectively capture the sharing but avoid the unnecessary overhead. We also observe that our proposed semi-query allocation algorithms perform nearly as well as the complicated SDP-K-Cut classical solution. Although our semi-query allocation algorithms are opportunistic in nature, the performances have revealed their power and value. Comparing Figure 4.4(b) with Figure 4.4(c), we note that with the same number of queries, queries with larger average regions are likely to benefit more. This is because there are more over-

laps among queries when each query is querying a larger region, and hence more benefit can be exploited by our similarity-aware query allocation algorithms. Using the same logic, referring to Figure 4.4(a) and Figure 4.4(b), with the same number of queries and query regions with the same average size, more gain is achieved when the network size is smaller.

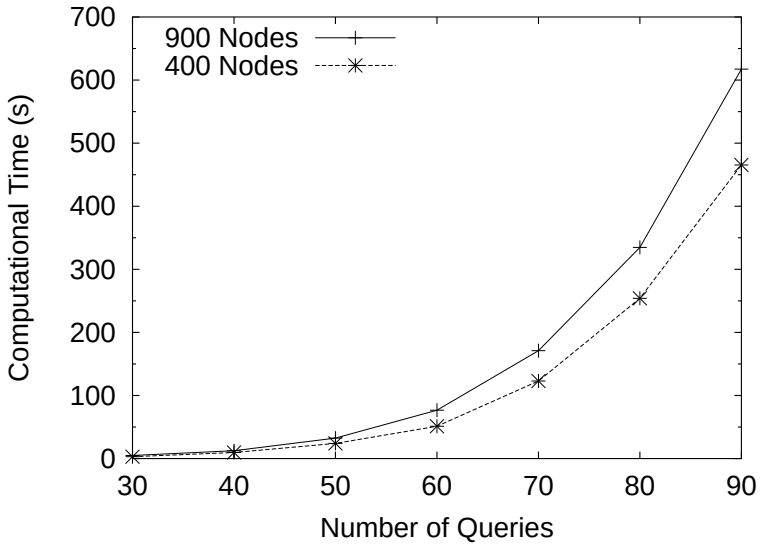
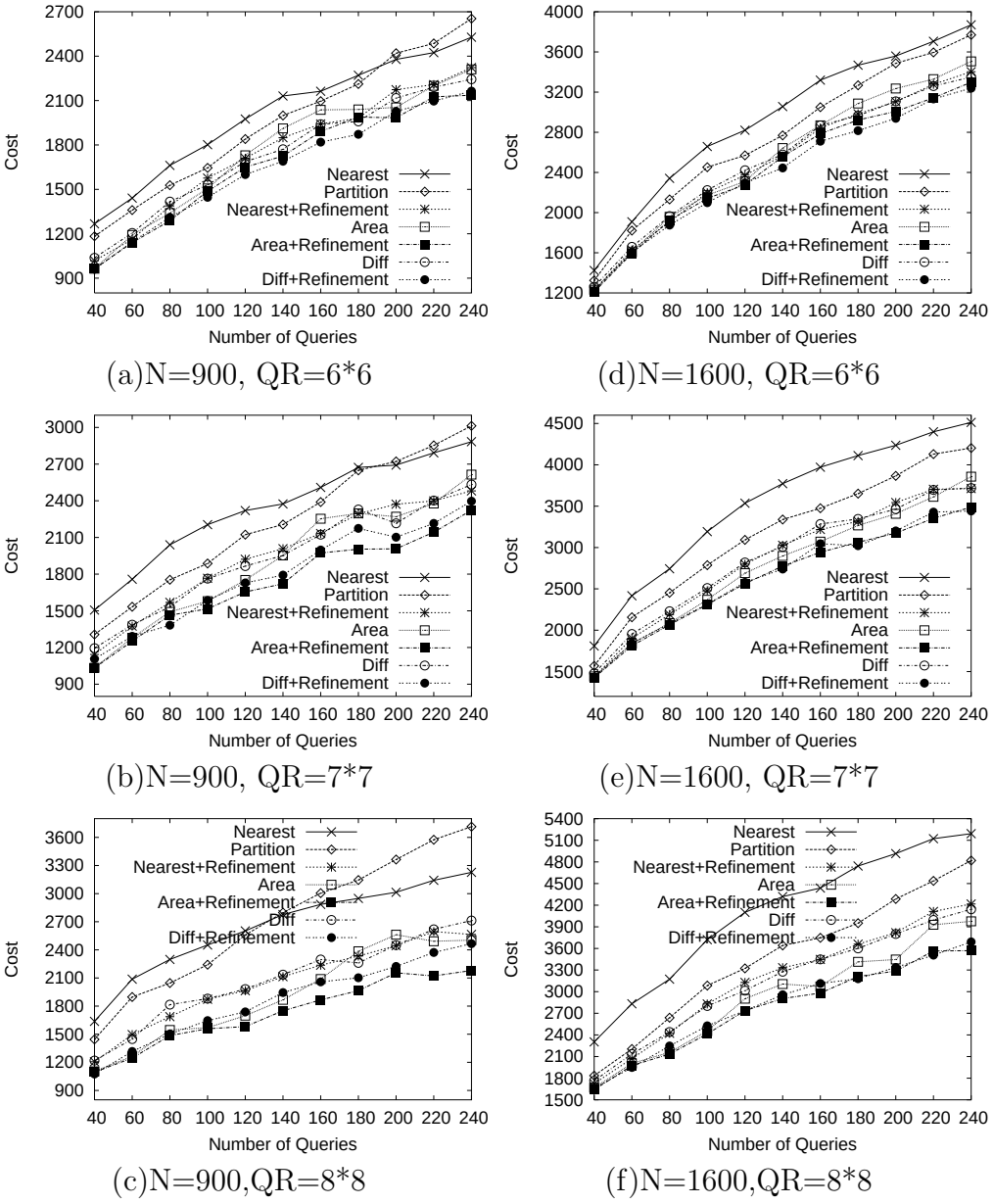


Figure 4.5: CPU Time for SDP_K over Random Queries with Average QR=8*8

Figure 4.5 shows the computational time taken to solve the Semidefinite program in the SDP K-Cut solution. The computational time increases exponentially when the number of queries increases. As the number of queries reaches 90, the time to get the partitions is more than 10 minutes, in a network with 900 sensors (9 base stations). For our semi-greedy query allocation variants, it takes less than 4ms to compute the greedy query insertion, and the iterative refinement generally converges fast, which takes less than 10ms. The negligible overhead of our proposed greedy insertion also shows its applicability for dynamic environments.

In the above, we have examined all the strategies in general and showed



(a)N=900, QR=6*6 (d)N=1600, QR=6*6

(b)N=900, QR=7*7 (e)N=1600, QR=7*7

(c)N=900,QR=8*8 (f)N=1600,QR=8*8

Figure 4.6: Effectiveness of greedy query allocation for random queries in larger scale network

clearly that our semi-greedy query allocation algorithms are nearly as effective as the complicated and thus unscalable SDP-K-Cut, but at a much lower cost. In the next part of this section, we will exploit the effectiveness of each strategy in the semi-greedy allocation algorithms, under larger-scale scenarios.

Figure 4.6 shows the effectiveness of each strategy in our semi-greedy allocation algorithms. As we can see from Figure 4.6(a) to (f), under various network size, average query region, and number of queries, both greedy insertion and iterative refinement can exploit the sharing among queries and thus reducing the communication cost.

The greedy insertion algorithms, no matter with area-based sorting or diff-based sorting, have considerable improvement over our baselines — the not-so-bad *Nearest* and *Partition*, as we discussed in Section 4.5.1. More specifically, as shown in Figure 4.6(a) to (c) or Figure 4.6(d) to (f), as the average query region size increases, the greedy insertion algorithm adopting area-based sorting tends to outperform the greedy insertion algorithm adopting diff-based sorting. This does not happen accidently. For random queries, the lengths of query regions are generated uniformly, and hence bigger average query region size implies bigger variance among the size of query regions, and correspondingly the probability is higher that query area makes a difference.

The iterative refinement is shown to be effective in further refining the initial query allocation plan and thus reducing the communication cost. From Figure 4.6, we can see that the amount of improvement for *Nearest* is generally more than that for greedy insertion algorithms denoted as *Area* and *Diff*. Compare with the plan generated by our greedy insertion algorithms, *Nearest* is further from being an optimal plan and generally has more room to be improved during the refinement step. Our refinement algorithm is shown to be able to exploit the room, which reflects its effectiveness. In the meanwhile, we should keep in mind that the iterative refinement is not the algorithm that goes through all the searching space

to get one optimal query allocation plan. Although it makes decreasing check for the rounds in each iteration and therefore it has the chance to discover the benefit that can only be found by considering several query reinsertions together, it limits itself by only making incremental check in each iteration. Hence, as we can see in Figure 4.6, our greedy insertion algorithms with refinement generally outperform *Nearest* with refinement.

4.5.3 Performance in the Dynamic Context

In this section, we evaluate the performance of our adaptive query allocation schemes — incremental insertion algorithm and migration algorithm, in the dynamic context where new queries arrive and existing queries terminate.

To evaluate the effectiveness of the incremental insertion algorithm *Incremental* in reducing the communication cost, we compare its performance against *Nearest* insertion, *Partition* insertion and SDP-K-Cut. Since SDP-K-Cut is only suitable for static scenario, here we only note down the cost of *Incremental* after a specific number of queries have been inserted and running in the network; the same set of queries will then be allocated by SDP-K-Cut and we compare their cost.

As shown in Figure 4.7, the *Incremental* performs much better than *Nearest* and *Partition*, and even approaches SDP-K-Cut when the number of queries is small. By greedily finding the base station that incurs the smallest additional cost for each newly arrived query, the new query can effectively take advantage of the sharing with existing queries in the system. But when the number of queries is bigger, there will be more cases that previously inserted query cannot benefit from the newly inserted query, and

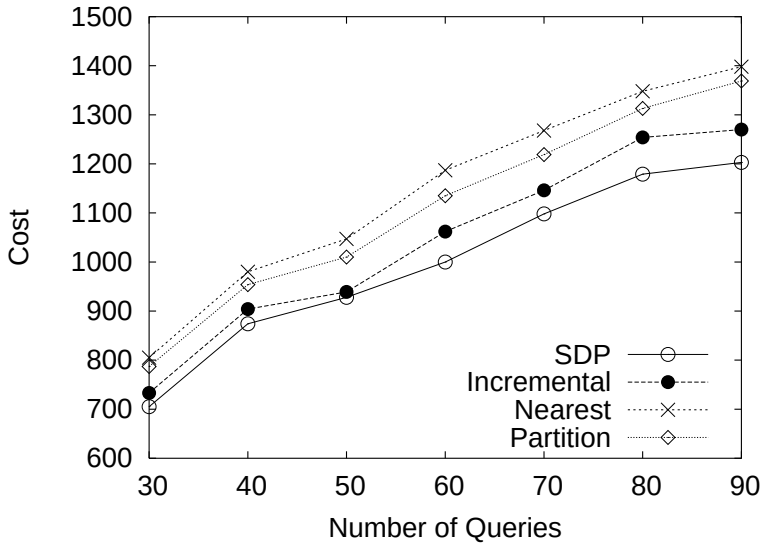


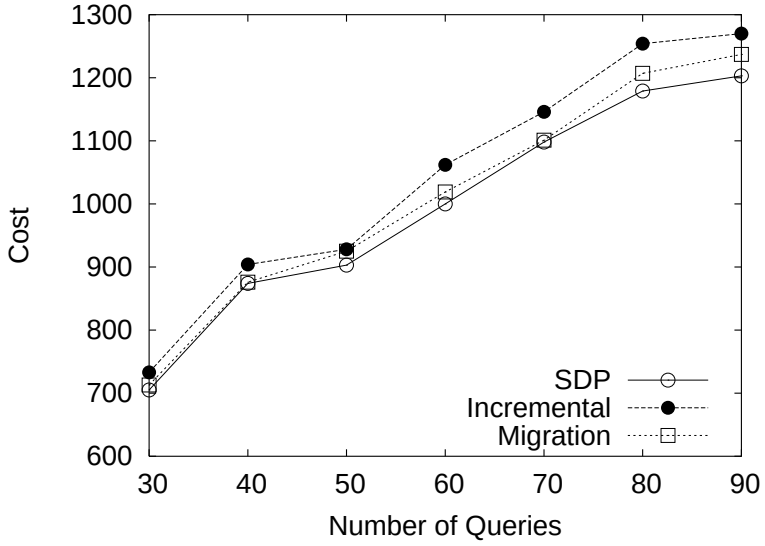
Figure 4.7: Evaluating incremental insertion over Random Queries with N=900, QR=5*5

hence it is further from being optimal. This is also one of the motivations for our adaptive migration algorithm.

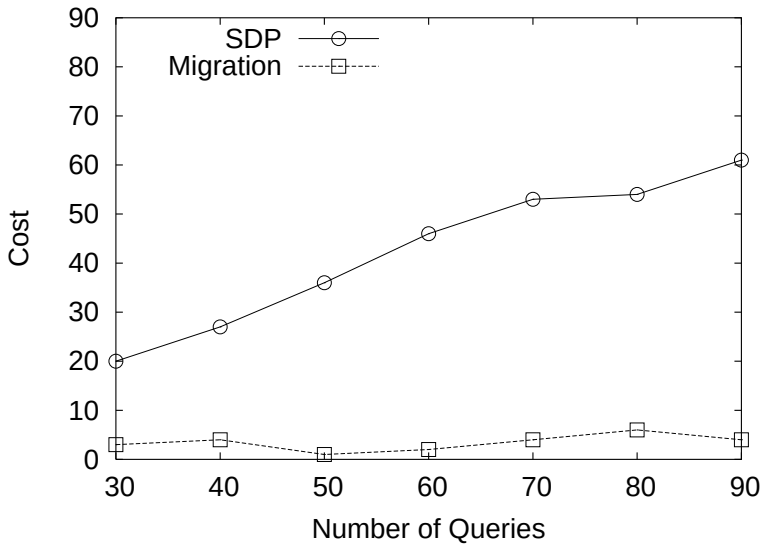
To evaluate the effectiveness of the adaptive migration algorithm, we need to evaluate two aspects of it: 1). its ability to improve the current query allocation; 2). its integration with the incremental insertion algorithm.

We make use of the experimental setup above that is used to test *Incremental* to test the performance of Algorithm 3. When it is the time point to note down the cost of *Incremental*, we run our migration Algorithm 3, and see the amount of gain in cost we can achieve. At this migration point here, we temporarily do not take into account the overhead of migration, and assume it to be amortized with time; otherwise, the gain will always be below zero, because no query can gain more than its own cost during the migration point.

Figure 4.8 shows that migration largely bridges the gap between *In-*



(a)Communication Cost

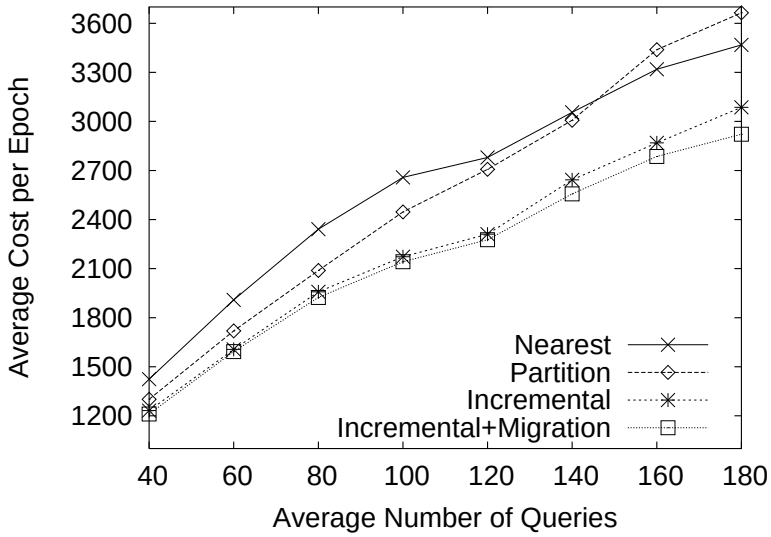


(b)Number of Migrated Queries

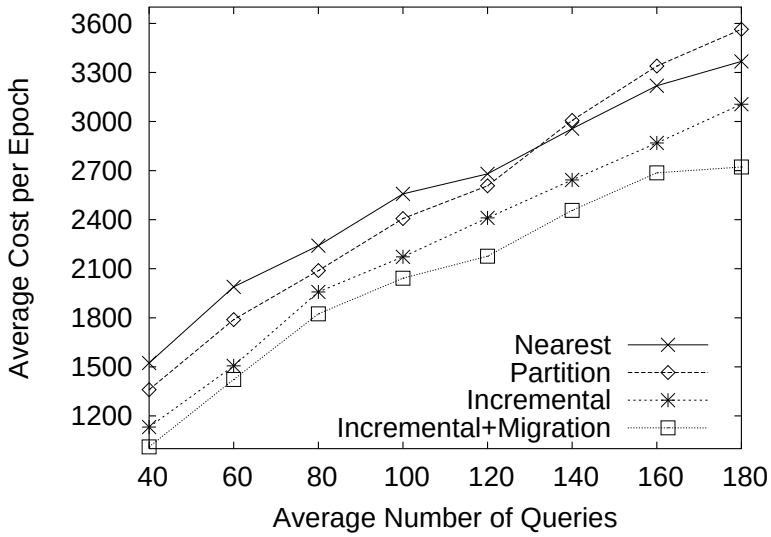
Figure 4.8: Evaluating migration over Random Queries with N=900, QR=5*5

cremental and SDP-K-Cut. It is essentially one kind of online refinement strategy. Through greedily exploiting the sharing among all the queries, much communication cost inside the network can be reduced. In this experiment, our migration algorithm is able to achieve around 70% of the gain SDP-K-Cut can obtain, at the cost of migrating around 10% of the

number of queries that SDP-K-Cut needs to migrate.



(a)Average Interval=10



(b)Average Interval= 40

Figure 4.9: Evaluating adaptive migration over random queries with N=900, QR=6*6

Finally, we examine our incremental query insertion algorithm and adaptive query migration algorithm detection as a whole. Queries arrive and terminate whenever they want, but the average frequency of arrival and duration is controlled, and the average number of queries remains the

same After a set of 300 queries terminates, we note down the total communication cost in the network so far and use the cost of communication per epoch as the metric to compare different approaches. As shown in Figure 4.9, when the Average interval of query arrival/termination is low, the performance gain of *Migration+Incremental* over *Incremental* is less than the situation when the interval is higher (e.g., a comparatively steady environment). This is because when there is frequent insertion/termination, migration has less job to do, since incremental insertion itself has high chance to reduce the migration pressure. From the above, we can see that migration algorithm can effectively adapt to the frequency of query arrivals and terminations.

In summary, our extensive experimental results show that all our similarity-aware query allocation schemes offer significant performance improvement over the best allocation strategy that fails to take advantage of the inherent sharing among other queries or overlooks the power of in-network aggregation, both in static context and adaptive context. Moreover, our proposed greedy query allocation algorithms perform nearly as well as the complex SDP-K-Cut classical solution in terms of the communication cost among sensor nodes, while incurring negligible computational time. Similarly, the migration strategy is shown to achieve comparable performance with the approach in which we recompute the query allocation plan using the SDP-K-Cut solution, but with a much smaller number of migrated queries.

4.6 Related works

As we have seen in Section 2.2.1, aggregation query processing in sensor networks has recently attracted much attention from the research community.

To optimize aggregation query in resource-constrained sensor networks, in-network aggregation techniques have been extensively investigated to exploit its ability to reduce the data collection cost within the sensor network. Take the categorization by the adopted network routing structures for example, there are cluster-based [39, 129], chain-based [59], gossip-based [17], tree-based [125, 69, 103, 24], multi-path-based [21, 78], and hybrid [74, 58], etc. To satisfy various application needs, these in-network aggregation techniques proposed range from minimizing-energy-consumption aggregation [39, 125, 74] to extending-network-lifetime aggregation [103, 123], from error-bounded approximate aggregation [24, 103] to energy-bounded approximate aggregation [58].

Other than the above literatures, which focused on the processing of a single aggregation query, multiple query optimization techniques have also been studied [110, 118]. With a set of aggregation queries that have arrived at the base station, sharing among different aggregation queries are achieved through effectively managing the equivalent regions in [110], or through dynamic in-network aggregation in our TTMQO approach [118]. Our study differs from previous work in that we are dealing with the aggregation query processing problem one level up — we consider the query allocation among the set of aggregation queries to different base stations, so that the sharing among these queries can be maximized in the network and thus the energy consumption of the whole sensor network is minimized.

Query/operator allocation has been studied in both traditional distributed database systems [14, 30, 62, 99] and, more recently, stream processing systems [5, 80, 98, 121, 133]. These optimization algorithms mainly focused on fine tuning the allocation of queries/operators across the dis-

tributed servers to balance the load distribution as well as to minimize the communication cost among the servers. However, the context of this paper is much different from theirs. Our objective is to minimize the communication cost inside the sensor network instead of among the base stations. Moreover, we endeavor to maximize the sharing of the data collection cost among various queries allocated to the same base station. This is typically not considered in existing work.

There are also existing works about request distribution among computer clusters [79, 41, 1, 2]. Each request needs a specific part of data for its answer which can be cached at a server, and thus request sharing in this setting looks similar to the query sharing in this paper. Apart from the different application context, their objectives are either to maximize the throughput or to balance the load distribution among clusters. Furthermore, the message routing scheme within a WSN, which is taken into consideration in the design of our allocation optimization algorithms, is much different from that within computer clusters.

This chapter is also related to some other algorithmic literatures. Note that the query allocation problem can be solved by a two-phase approach: a query partitioning phase followed by a mapping phase. In the query partitioning phase, queries can be partitioned into K disjoint sets, so that the amount of sharing in the K partitions is maximized. In the mapping phase, it is in fact a complete bipartite mapping problem [22], where K partitions are allocated to K base stations so that the weight of the mapping is minimized. Again, most of the existing work in query partitioning either balance the partitions [45] or minimize the number or weight of cuts [86]. The latter category is related to our objective, but it suffers from the sim-

ilar problem as our Max-K-Cut approximation that we have discussed in Section 4.3.2. Hence, they are not ideal for our case.

4.7 Summary

In this chapter, we have identified the importance of a multi-base-station infrastructure and a multi-query optimization scheme for large scale sensor network. Correspondingly, we have introduced and examined the query allocation problem in such an environment, with the objective to minimize the total data communication cost among the sensor nodes. We designed several similarity-aware algorithms to leverage the query sharing through query allocation. Furthermore, adaptive optimization algorithms were also proposed to handle the dynamic change of query set. Finally, experimental results showed that our similarity-aware query allocation schemes can effectively exploit the sharing among queries and greatly reduce the communication cost.

Chapter 5

Optimizing Multiple Queries in Sparse Mobile Sensor Networks

5.1 Introduction

In the previous chapters, we have focused on the WSNs consisting of the first generations of sensors that are stationary and battery-powered [111]. As such, to efficiently support large-scale WSNs, we have to rely on multiple base stations (as we have investigated in Chapter 4).

However, as technology advances, more powerful sensors integrated with mobility functionality have been developed [90, 75, 112]. This prompted us to consider deploying a small number of mobile sensors to monitor a large region which would have required a large-scale WSN. This alterative strategy is possible because the mobility of sensors effectively extends the sensors' coverage, by moving around to sense a larger area than its sensing range. In addition, because the mobile sensors are not constrained by

the initial deployment and can be relocated to desirable locations when necessary, they can contribute flexible network topology, react to the events in the environment and adapt to the changes in the missions. Thus, WSNs comprising such mobile sensors, called Mobile Sensor Networks (MSNs), have been increasingly deployed to support applications in surveillance, reconnaissance, and disaster rescue [40, 9, 96, 64].

In this chapter, we intend to study how mobile sensors can be exploited for query processing in a MSN. If all the sensors are connected, the following straightforward solution would be good enough: find a sensor near the query region, let it move to the query region to collect data, and use connected nodes to relay the data back to the base station. Therefore, we focus on a sparse network. It is common that a MSN is sparse because of the following two reasons: mobile sensors are more expensive compared to stationary sensors and the mobility of the sensors increases their coverage [61], so it may not be wise to densely deploy a large number of them; moreover, dense MSNs can become sparse due to node failures caused by environmental hazards or intentional damages (e.g. by adversaries in the battlefield), or even due to nodes move out of the communication range.

In a sparse MSN when the number of sensors is limited, the connection between sensors is intermittent and the topology is unpredictable, which bring several complications in query processing. Thus, to process data acquisition queries quickly is very challenging. In the context of a sparse MSN, the straightforward solution mentioned above for densely populated MSN will not work well, because: 1) when the base station issues a query, most sensors are not reachable, therefore the base station does not know which sensor is near the query region and have no way to contact that

sensor because the network is partitioned; 2) the base station therefore has to select a sensor among the connected ones to acquire data for the query; it may take much time for the sensor to move to the query region; 3) after collecting the data from the query region, the sensor may have no connection to the base station so it cannot send the acquired data directly back to the base station; it has to move towards the base station; again, it may need to move a long way before it is connected to the base station; 4) in addition, since only a small number of sensors may be reachable from the base station, due to the competition for sensors among queries, some queries may have to wait long to be served.

Therefore, in this chapter, we endeavor to provide fast response for multiple data acquisition queries in sparse MSNs. To efficiently process a single query, we first propose a distributed scheme called *bridge* to reallocate nodes in the run time to form a chain between the query region and the base station, in order to rapidly relay information [113]. Then, we introduce distributed schemes in which the mobile sensors can organize themselves and collaborate to concurrently process multiple queries. More specifically, we design one strategy *Dynamic* that dynamically shares resources among queries, while the processing of each query greedily relocates exploited resources for its own sake as what *bridge* does. We also design another strategy called *aMST* that utilizes an adapted Minimum Steiner Tree to guide the processing of all queries as a whole to minimize the average query processing time. In each of the above schemes, we deal with the various problems caused by intermittent connections. In addition, we define a parameter *Coverage Ratio(CR)*, which reflects the sparseness of the network in respect to the number of queries, to guide the system

to adaptively make a sound decision over the strategies. Our extensive performance study shows the effectiveness of our proposed strategies.

The rest of the chapter is organized as follows. We describe the system model, define and analyze the query processing problem in Section 5.2. We present a greedy solution for a single query in Section 5.3. Various multiple query processing approaches are presented in Section 5.4. In Section 5.5 we briefly survey the related works. Results of experimental study are shown in Section 5.6. We finally conclude the chapter in Section 5.7.

5.2 System Model and Problem Definition

In this section, we present the model of the mobile sensor networks that we are interested in, and define the problem of multiple data acquisition query processing.

5.2.1 System Model

The system consists of a stationary base station BS and n mobile sensors (s_1,s_2,...,s_n) that are sparsely distributed in a large area A. Each mobile sensor knows the BS's location and its own location. The mobile sensors and the BS use wireless technology (such as Wi-Fi) for communication and there is no direct long-range communication. Two sensors (or the BS and a sensor) can communicate *directly* only if the distance between them is smaller than the wireless technology's communication range r. The sensors and the BS essentially form a mobile ad-hoc network (MANET) where one can communicate with another if they are connected either directly or through other sensors. Since the sensors' communication range is limited

and the sensors are sparse, the network formed by the sensors is not fully connected, and can even be severely partitioned. The topology of the network changes with time as the sensors move.

The mobile sensors explore (sense) the big area A by moving in it following a certain mobility pattern. Each sensor node is equipped with a location positioning device such that they can move geographically as instructed. The mobile sensors' move speed is v. Each sensor senses the region that it passes by and the sensing speed is $a = c * v$ where c is a constant determined by the application. A sensor carries the sensed data and forwards the data to the BS when it is connected to the BS. The BS continuously analyze these collected data and may detect some possible events from some locations. To confirm these events, more detailed and timely information from the suspicious locations is required. It is likely that multiple events need to be confirmed at the same time. For example, in an application such as military surveillance, military invasion often takes place in several threads.

Data Acquisition Query

The BS sometimes proactively issues data acquisition queries to the network. A query specifies a (relatively) small region (within A) whose data needs to be sensed and fetched as early as possible. Given a query, the mobile sensors shall process the query by going to the query region, sensing the query region, and sending the sensed data back to the BS. The BS would like the time from the moment an explicit data acquisition query is issued to the moment relevant data is received to be as short as possible. This type of query belongs to the category of one-shot query, as we presented in

Section 2.1.1.

Note that when data acquisition queries arrive, it is possible that only a small number of sensors are connected to the BS.

Assumptions

Since our optimization metric is the time the sensors take to process multiple data acquisition queries, for simplicity we in this chapter do not consider energy consumption. In the class of applications that we are considering, all the sensors are moving to collect information. The energy spent on moving will be much more than the energy spent on communication. [75] shows that when a 0.5 kg UAV (Unmanned Aerial Vehicle) flies horizontally at a 10-12 m/s speed its minimum energy consumption is 10-25 J(joule). It is reported in [29] that a normal (Lucent) IEEE802.11 wireless network card consumes about 1.5 J per second when in active transmission mode. Although more powerful wireless transmitters may be used, they will be equipped only on larger UAVs. Clearly, larger UAVs will need much more energy for flying, or for simply staying in the air. It is also mentioned in [90] and [112] that a mobile sensor on ground spends much more energy on motion than on wireless communication when moving around.

Since all the sensors are moving most of the time, they will spend similar amount of energy no matter how they communicate with each other and how they move during query processing. For this reason, we believe that: 1) trying to save energy by controlling the communication between the sensors will not be very helpful; 2) even if controlling the distances sensors have to travel during query processing may help save some energy, the savings will be marginal because query processing only happens upon the

arrival of ad-hoc data acquisition queries.

We will also ignore data transmission time. We believe that in a sparse MSN, during query processing, the time spent on relocating sensors will be much longer than that spent on data transmission, so it does not affect our design of the query processing scheme.

5.2.2 Problem Definition and Analysis

Let us call the sensor that is assigned to sense the query region as the *Scanner*. We define the sensed data from the query region as the *query result*, and the duration from the moment the query is issued by the BS to the moment the BS receives the query result as the *query processing time*.

The problem of multiple data acquisition query optimization is to design a solution by which the mobile sensors manage to sense the query regions and send the query result back to the BS with a minimum average query processing time.

More specifically, for a query q_i, its query processing time T^i can be divided into four parts:

- T^i_{wait}: from the time the query is issued by the BS to the moment it is assigned to a Scanner;
- T^i_{go}: from the time the query is assigned to a Scanner to the moment the Scanner arrives at the query region;
- T^i_{scan}: the time for sensing the query region;
- T^i_{return}: from the moment Scanner finishes sensing the query region to the moment the BS receives the sensed data.

Suppose there are N queries, our goal is to minimize the average query processing time T_{avg}:

$$T_{avg} = \frac{\sum_{i=1}^{N}(T_{wait}^{i} + T_{go}^{i} + T_{scan}^{i} + T_{return}^{i})}{N}$$

Since T_{scan}^{i} is fixed for a given query region and a sensing speed, we would like to minimize $T_{wait}^{i} + T_{go}^{i} + T_{return}^{i}$. T_{wait}^{i}, T_{go}^{i} and T_{return}^{i} can all be significant in sparse MSNs because: no sensor may be reachable from the BS to be assigned as a Scanner for the query; the Scanner could be far away from the query region; after sensing the query region the Scanner may be disconnected from the BS so it has to move to find a connection to the BS. We shall reduce all T_{wait}^{i}, T_{go}^{i} and T_{return}^{i}.

5.3 A Greedy Scheme for Single Query Processing

In this section, we present our greedy query processing scheme called *bridge*, that minimizes the query processing time of a specific query in sparse MSNs. The main ideas are as follows:

1. assign the sensor that is the nearest to the query region as the *Scanner* to acquire data for this query, so that the query location can be sensed as early as possible. This is to reduce T_{go}.

2. relocate some sensors to help connect the *Scanner* to the BS so that the *Scanner* can send the acquired data to the BS without moving towards the BS. We refer to these sensors as the *Router*s. This is to reduce the T_{return} component of the query processing time.

3. gradually improve the query plans by exploiting the encountered sensors during query processing to seek for better *Scanner* or better *Router*s. This is to reduce T_{go} and T_{return}.

Figure 5.1 uses the processing of a query as an example to illustrate these ideas. In the figures, the rectangle marked with q is the query region, the circles marked with s_1,...s_6 are the sensors, the arrows on the sensors indicate the sensors' moving directions, the sensor in dark (e.g. s_3 in (b)) is the Scanner, the sensors in gray (e.g. s_1, s_2 in (b)) are the Routers, and the ones in white (e.g. s_4, s_5, s_6 in (b)) are not involved in the query processing. (a) depicts the scenario when the query arises. (b) shows the initial query plan where s_3 is assigned as the Scanner and s_1 and s_2 are assigned as the Routers. s_3 is assigned as the Scanner because it is the nearest to the query region among the sensors that are connected to the BS. (c) depicts the event where s_3 encounters s_4 and s_5. s_3 adapts the initial query plan to a new query plan that is depicted in (d). In the new query plan, s_5 is assigned as the Scanner, s_1, s_2, s_3 and s_4 work as Routers. s_5 is selected as the new Scanner because it is nearer to the query region. s_3 and s_4 also work as Routers because the existing Routers (s_1 and s_2) are not enough to connect the Scanner to the BS. In (e), the Scanner is sensing the query region and the Routers connect the Scanner with the BS. After the query is done, as shown in (f), all the participating sensors can move freely.

To realize the above ideas in a sparse MSNs, the problems caused by the intermittent connections have to be dealt with. The Greedy solution is made up of three algorithms, namely `Init`, `Adapt`, and `Merge`. The `Init` algorithm (Section 5.3.2) is used by the BS to generate an initial query

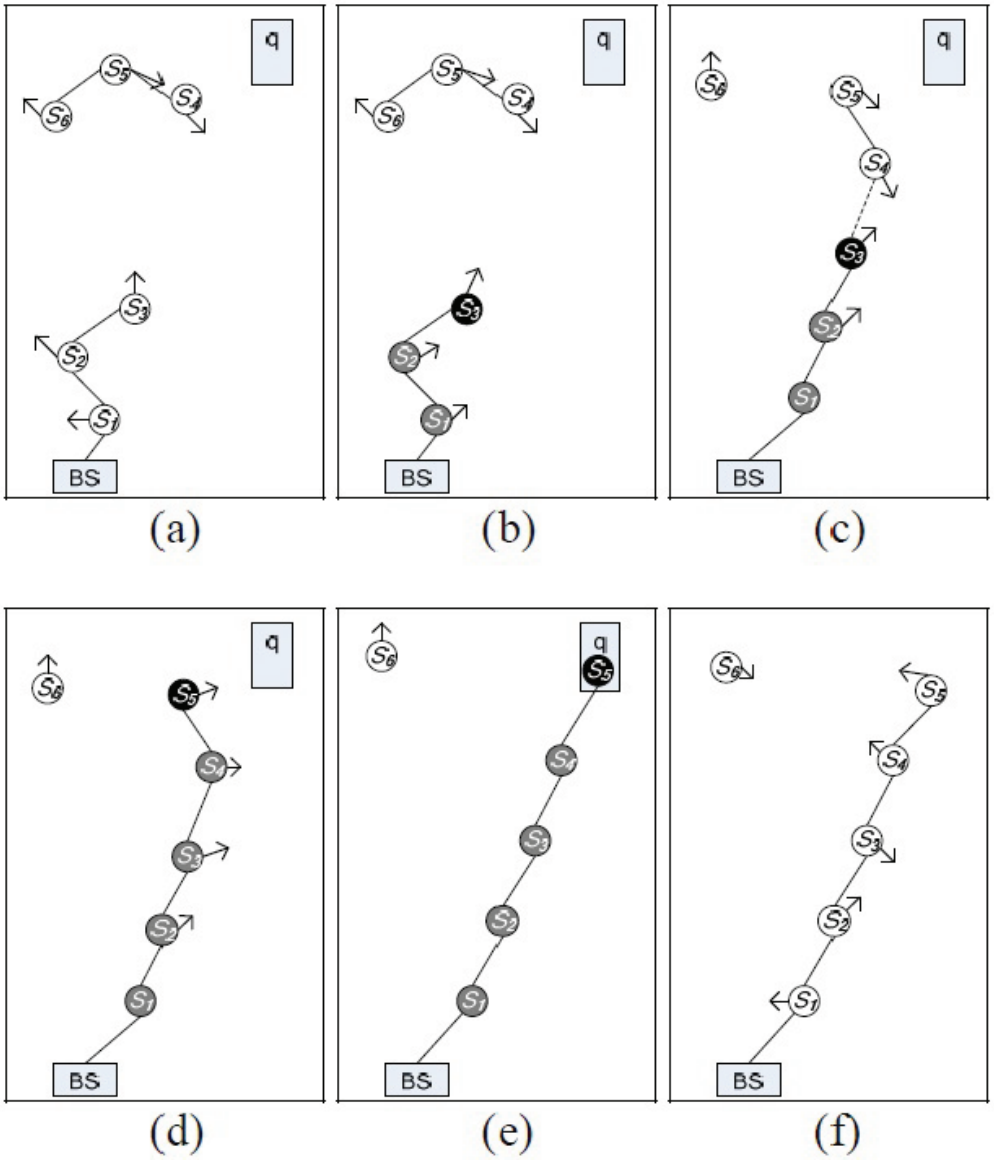


Figure 5.1: An Example illustrating the *bridge* algorithm.

plan for a query as long as at least one sensor is connected with the BS. The `Adapt` algorithm (Section 5.3.3) is used to generate a better query plan when a sensor participating in the query processing encounters new neighbors. Since involved sensors may generate different new query plans when they are disconnected, the `Merge` algorithm (Section 5.3.4) is used to merge the different query plans to a new query plan when the disconnected

sensors get connected again. In addition, after a query is completed, we need to make sure that all the sensors involved in the query plan finally know that the query is done so that they can serve other queries or return to work on their general task. This problem is solved by letting the sensors keep the information about data acquisition queries (e.g whether a query is done) and synchronize the information when being connected.

5.3.1 Basics

Before presenting the above algorithms, we will first describe the sensor's states, the idea and definition of the *RouterPoint*s, what makes a query plan, and the execution of a query plan.

Sensor States

In our *bridge* algorithm, a mobile sensor is always in one of the following states: Free, Scanner, Router, and Returner. A sensor's state may change during the processing of a query because the sensors will adapt the query plan when they encounter new neighbors.

A sensor is in the Free state if it is not involved in the query processing. A Free sensor works on the general task of the application, e.g. explore the application area.

A sensor is in the Scanner state if it is assigned to go to sense the query region. After being assigned as a Scanner, the sensor moves towards the query region.

A sensor is in the Router state if its task is to help connect the Scanner with the BS. A Router sensor will be given an ID (which is independent from its sensor ID) called the Router-ID. The Router-ID tells the Router

where it shall move to. We will discuss this further later.

A sensor is in the Returner state if it is carrying the query result and it is responsible for sending the result to the BS. A Scanner becomes a Returner when it finishes sensing the query region. A Returner always passes the query result to the sensor that is the nearest to the known Routers or the BS. If a Returner cannot find a better Returner, it moves towards the known Routers or the BS.

After a Returner relays its query result to the BS, all the sensors involved in processing the query will be in Free state.

RouterPoints, Router-ID

Given a query, to connect the Scanner (at query region) and the BS with a minimum number of Routers, we shall relocate the Routers onto the straight line between the BS and the query region, and let them maintain a distance of r from its neighboring Routers. In this way, the number of Routers, denoted as N_r, is minimized. It is computed as follows:

$$N_r = \lceil Distance(BS, R)/r \rceil - 1 \tag{5.1}$$

Here r is the sensors' communication range and $Distance(BS, R)$ measures the distance between the BS and the query region R. These N_r Router sensors can build a link on the line between the query region and the BS. For example, if the distance between the query region and the BS is 2400 meters and the sensors' communication range is 500 meters, we will need at least 4 sensors to work as Routers. Figure 5.1 (e) shows the example.

We define the locations that the Routers shall move to as the *RouterPoint*s of the query. Let $Line(BS, R)$ be the line segment between the BS

and a query region R. There are N_r RouterPoints on the $Line(BS, R)$. We define $RouterPoint_j$ as the point on $Line(BS, R)$ whose distance to the BS is

$$i * Distance(BS, R)/(N_r + 1) \tag{5.2}$$

In our algorithm, Router-j moves to the $RouterPoint_j$ on the $Line(BS, R)$. Note that we do not let Router-j move to the location on $Line(BS, R)$ whose distance to the BS is $j * r$. That will make the Router sensor not able to move around the location because a small movement of Router-j can make it disconnect from Router-$(j-1)$ or Router-$(j+1)$. The advantage of letting Router-j move to $RouterPoint_j$ is that it still has some flexibility to explore the area around the RouterPoint (exploring the whole area is the sensors' basic task).

Query Plan

A Query Plan of a query $Q(R)$ is an assignment of a set of sensors to their roles in the processing of the query. A query plan QP contains the following information:

- $Q(R)$: the query, which includes the query region information.
- Sensor ID $\rightarrow$ Scanner: this specifies which sensor shall work as the Scanner. Later we will use QP.Scanner to refer to it.
- {Sensor ID $\rightarrow$ Router-j}, $1 \leq j \leq N_r$: a mapping of sensor IDs to the Router-IDs. This specifies which sensors shall be Routers. We will use QP.Router-j to refer to an entry in it. There could be fewer

than N_r Routers. If no sensor is assigned as Router-j, QP.Router-j is *null*.

- The time the query plan is generated.
- The locations of the involved sensors (the Scanner and the Routers) when the query plan is generated.

Notice that in the query plan we keep the locations of the involved sensors at the time the query plan is generated. We do this because we want a sensor having a copy of a query plan to be able to adapt the query plan to a better query plan when it encounters new neighbors. Note that the sensors participating in a query plan can get disconnected when they move to their destinations. When this happens, some cannot get the up-to-date (location) information of the others. With the information kept in the query plan, a sensor knows each participant's destination (indicated by its role) and its locations at a certain time (the time when the query plan is generated), so the sensor can estimate when the participant (either connected or disconnected) can reach its destination. A participating sensor encountering a new neighbor therefore can compare an existing participant with a new neighbor to see which one can reach the participant's destination. This is important for adapting a query plan to better query plans. We will discuss this further in Section 5.3.3.

Execution of a Query Plan

To execute a query plan, the sensor that computes the query plan simply disseminates the query plan to all the connected sensors. This is to guarantee the consistency among the query plans at all the connected participants.

With this consistency guarantee, when two disconnected participants of a query plan meet, they only need to fetch their local query plans to generate a new plan.

Upon receiving a query plan, a sensor checks whether it has a role in the query plan. If yes, it saves a copy of the query plan and sets its state according to its role in the plan; otherwise it sets itself to the Free state. The Scanner moves towards the query region. The Routers first compute their RouterPoints based on the following information: theirs Router-IDs in the query plan, the locations of the BS, and the query region. They then move to their RouterPoints.

When the BS issues a query, it will compute an initial query plan based on the sensors that are currently connected to it, and execute the initial query plan among the sensors.

After the initial query plan for a query is executed, there are two cases where new query plans will be generated for the query. The first is when a participating sensor encounters new Free sensors. The second is when two sensors having different query plans for the same query meet, this is possible because two participants serving the same query can get disconnected on their way to fulfill their respective roles in the query plans. In either case, if a new query plan is generated, the new query plan will be executed in the connected sensors so that all the connected sensors follow the new query plan. If a sensor that had a role in a query plan finds that it does not have a role in the new query plan, it sets itself to the Free state.

5.3.2 Init a Query Plan

When the BS issues a new query, it will use the `Init` algorithm presented here to generate an initial query plan based on the Free sensors that are currently connected to it. `Init` assigns the Free sensor that is the nearest to the query region as the Scanner and assigns at most N_r Free sensors as the Routers based on their vicinity to the RouterPoints. The pseudocode of `Init` is listed in Algorithm 6.

Algorithm 6: `Init`

Input: a query $Q(R)$
Output: a query plan QP for $Q(R)$

```
1 Frees ←the Free sensors that are connected to BS;
2 QP.Scanner ← Nearest(Frees, R) ;
3 Frees ← Frees − {Scanner};
4 N_r ← Ceiling(Distance(BS, R)/r) − 1;
5 m ← Min(N_r, |Frees|);
6 for j ← 1 to m do
7 |   QP.Router_j ← Nearest(Frees, RP_j);
8 |   Frees ← Frees − {Router_j}
9 return QP
```

5.3.3 Adapt a Query Plan

During the processing of a query, participants of a query plan may encounter Free sensors that were not connected. The new neighbors may help improve the current query plan in several ways. A new neighbor may be a better Scanner if it is nearer to the query region than the current Scanner is. A new neighbor may work as a Router if the current query plan has fewer than N_r Routers. Even if the current query plan has enough Routers, a new neighbor may still help if it is nearer to a RouterPoint than the current corresponding Router is.

Algorithm 7: `Adapt`

Input: a query plan QP
Output: a new query plan

```
1 Frees ← the Free sensors that are connected to s_i;
2 Scanner' ← Nearest(Frees, R);
3 if Scanner' can reach R earlier than QP.Scanner then
4 |   if QP.Scanner is connected then
5 |   |   Frees ← Frees + {Scanner}
6 |   QP.Scanner ← Scanner';
7 |   Frees ← Frees − {Scanner'};
8 QP ← AdaptRouters(QP, Frees);
9 return QP
```

In our Greedy query processing solution, the participants of a query plan always try to find better query plans when they encounter new neighbors. They do it using the `Adapt` algorithm listed in Algorithm 7. In this algorithm s_i denotes the participant sensor that encounters a set of new neighbors, and QP denotes the query plan that s_i currently has.

On encountering new neighbors, s_i finds out all the Free sensors that it now can reach (line 1). It first checks whether it can find a better Scanner among the Free sensors (lines 2-7). If there is a better Scanner, s_i puts the current Scanner to the $Frees$ set if it is still connected, and then updates the query plan with the new Scanner.

After that s_i adapts the Routers with the remaining Free sensors using the algorithm `AdaptRouters` which is listed in Algorithm 8. s_i divides the RouterPoints into two sets (lines 1-2): the ones for which no Routers are assigned, and the ones for which there are corresponding Routers in the query plan. It first assigns Free sensors to the RouterPoints that have no Routers (lines 3-6). Then it tries to find better Routers for existing Routers (lines 7-12).

Note that when s_i encounters new neighbors, it is possible that some other participants of the query plan are disconnected from s_i. For example, when there are not enough Routers, the Scanner will get disconnected from the Routers when it moves to the query region. This kind of disconnection between the participants of a same query plan has two influences. First, to make sure that a participant is able to adapt the query plan when encountering new neighbors, in the query plan we have to keep information about the locations of the participants when the query plan was generated. This is explained in Section 5.3.1.

Algorithm 8: `AdaptRouters`

```
   Input: a query plan QP
   Input: a set of Free sensors Frees
   Output: a new query plan
 1 EmptyRPs ← {j | j ≤ Nr ∧ QP.Router_j = null};
 2 ExisingRPs ← {j | QP.Router_j ≠ null};
 3 for j in EmptyRPs do
 4     if Frees is not empty then
 5         QP.Router_j ← Nearest(Frees, RP_j);
 6         Frees ← Frees − {Router_j};
 7 for j in ExisingRPs do
 8     if Frees is not empty then
 9         Router'_j := Nearest(Frees, RP_j);
10         if Router'_j can reach RP_j earlier than QP.Router_j then
11             QP.Router_j ← Router'_j;
12             Frees ← Frees − {Router_j};
13 return QP
```

Second, disconnected participants will not receive the new query plans that are generated by other participants. In this case, some have the old query plan while the others have a new query plan. Furthermore, partitioned groups of participants of a query plan may independently adapt the query plan to new query plans. As a consequence, there can be more than

Algorithm 9: `Merge`

Input: two query plans $QP1$ and $QP2$
Output: a query plan

```
1  if QP1.Scanner ≠ QP2.Scanner then
2      Scanners ← {QP1.Scanner, QP2.Scanner};
3      QP.Scanner ← Nearest(Scanners, R) ;
4      if the other Scanner is connected then
5          put it to Frees;
6  else
7      QP.Scanner = QP1.Scanner;
8  for Router_j in QP1 do
9      if Router_j is connected then
10         Frees ← Frees + {Router_j};
11 for Router_j in QP2 do
12     if Router_j is connected then
13         Frees ← Frees + {Router_j};
14 for For j ← 1 to N_r do
15     if QP1.Router_j ≠ null || QP2.Router_j ≠ null then
16         Routers ← {QP1.Router_j, QP2.Router_j};
17         QP.Router_j ← Nearest(Routers, RP_j);
18 QP ← AdaptRouters(QP, Frees);
19 return QP
```

one query plan for a query being executed in the sensors. When this happens, different query plans have to be merged when sensors having different query plans get connected again.

5.3.4 Merge Query Plans

In our Greedy solution, when two sensors having different query plans meet, one of them will merge the query plans to a new query plan and execute the new query plan among the connected sensors. The algorithm for merging two query plans is called `Merge` and Algorithm 9 lists its pseudocode.

Let s_i be the sensor that merges two query plans. In the Merge algo-

rithm, s_i first checks whether the two query plans have the same Scanner. If not, s_i chooses the one that can arrive at the query region earlier as the Scanner in the new query plan, and put the other Scanner into the $Frees$ if it is connected (lines 1-7). Then for all the Routers in the two query plans, if a Router is connected, s_i puts it into the $Frees$ set and clears the corresponding information in the query plan (lines 8-13). After this, all the available sensors will be in the $Frees$, and the two input query plans only contain information about Routers that are currently disconnected. For each Router slot in the new query plan, if the input query plans have disconnected Router(s) for it, s_i picks the one that is the nearer to the corresponding RouterPoint (lines 14-17). s_i finally uses the AdaptRouters algorithm to: (1) assign Free sensors to empty Router slots; (2) find better Routers for disconnected existing Routers (line 18).

The complexity of the `Merge` algorithm is $O(n * N_r)$ where n is the number of sensors that are connected to s_i (the sensor that merges the query plans) and N_r is the number of Routers needed in the query plan.

5.4 Multiple Query Processing Strategies

We have done preliminary studies of our proposed *bridge* scheme. The results showed its effectiveness in reducing query processing time [113]. Moreover, the idea of relocating some sensors to connect the Scanner with the base station was shown to be quite advantageous.

In this section, to efficiently process multiple queries, we will present three strategies building on *bridge*, i.e., *Sequential*, *Parallel* and *Dynamic*. In addition, we propose a novel scheme called *aMST* that relocates sensors for all queries as a whole to facilitate multiple query processing.

In optimizing multiple queries, there are two types of sharing that can be exploited: time sharing and spatial resource sharing. Time sharing is achieved if multiple queries are being processed concurrently in the network. Spatial resource sharing is enabled if the computation and communication resources of sensor nodes are shared among multiple queries.

5.4.1 Naive Strategies

Sequential

As the name suggests, this strategy processes queries sequentially, utilizing the *bridge* algorithm for each query. It does not allow time sharing or spatial resource sharing among different queries, because it devotes all spatial resources for one query at a time. We make it serve as the baseline.

In addition, to minimize the average waiting (queueing) time for each query, queries are designed to be processed in the ascending order of query size (shorter scanning time for smaller sized query). It is also worthy to note that subsequent queries are able to benefit from earlier queries, owing to topology with better connectivity formed around base station during earlier query processing.

Parallel

This strategy processes queries in parallel, opportunistically processing as many queries as possible to maximize time sharing. It specifies a separate line of router points for each query, as in *bridge*, without spatial resource sharing among different queries.

First, we introduce how the query plans are initialized. Basically, *Parallel* always gives preference to Scanners over Routers, to achieves maximum

parallelism among queries. After the BS issues N queries (ordered by ascending query size as in *Sequential*), it checks the set of reachable Free sensors. If the number of Free sensors $|Frees|$ is no less than N, all the queries will be issued to the mobile sensors right away. More specifically, to initialize the set of query plans, it first assigns the best Scanner for each query; it then assigns the remaining Free sensors to satisfy the Router needs of the first query; if there are still remaining Free sensors, they are assigned as Routers for the second query, and so on. Otherwise, if $|Frees| < N$, the BS generates query plan for the fist $|Frees|$ queries by assigning one Free sensor as the Scanner for each query, while the remaining $N - |Frees|$ queries wait until some new Free sensors become connected with the BS.

Next, we present how to find better query plans. Note the improvement of a query plan only happens when participants of the query plan encounter new neighbors, or a neighbor previously participating another query plan becomes free. If the neighbor is free, the same `Adapt` algorithm is adopted as in *bridge*. If the neighbor is a participant serving the same query, the `Merge` algorithm is adopted. Otherwise, as concurrent running queries are processed independently from each other in *Parallel*, no further action will be taken.

5.4.2 Dynamic

In this section, we propose a strategy called *Dynamic*, which is essentially an improvement over *Parallel*, by allowing spatial resource sharing in addition to time sharing. While each query plan greedily relocates exploited sensors for its own sake, run-time adaptation is introduced to intelligently enable queries to dynamically benefit from each other.

More specifically, *Dynamic* builds on and extends *Parallel* with the following four optimization techniques:

- **Intelligent Query Selection**. Instead of ordering all the queries in advance and dispatching the queries based on the order, we select the query to be dispatched at run time. Each to-be-dispatched query is strategically chosen as the query whose query region is nearest to the current available(free) nodes. In this way, the Scanner will be at an attractive distance from its query region, and thus this minimizes T_{go}. Moreover, with opportunistic router sharing technique which will be presented later, it also maximizes the chance of sharing among concurrent processing queries.

- **Moderate Parallelism**. When the BS initializes query plans, preference is partially given to routers over scanners, to achieve moderate parallelism among queries. Specifically, when there are connected Free sensors, BS intelligently selects a query, and call `Init` algorithm in *bridge* to initialize its query plan (i.e., assigning scanner and routers for the query using the Free sensors, with preference given to scanner over routers). If there are still Free sensors remaining, the above initialization process continues with another query, until all Free sensors have been assigned or all queries have been initialized. We note that as the router points nearer to the BS have higher priorities to be assigned with sensors in `Init`, the BS can soon reach a large area through those routers to discover Free sensors. Thus, the remaining queries are expected to be dispatched without a long waiting time.

  In this way, the time sharing is not maximized as in *Parallel*, but it can avoid the extensive resource competition among queries in *Par-*

allel and perform better in the end in the sparse MSN.

- **Aggressive Exploitation.** When a participant of a query plan encounters a new neighbor, even if the new neighbor is serving another query, we aggressively exploit the connectivity information of this new neighbor to find possible candidates to improve the query plan. More specifically, if a free sensor or another participant is reachable through the new neighbor, `Adapt` or `Merge` algorithm is called to generate a better query plan.

- **Opportunistic Router Sharing.** A Returner for a query is enabled to opportunistically utilize the Routers for other queries to facilitate query result returning. More precisely, the Routers for other queries can relay query result back to the BS or to a Free sensor that is nearest to the Routers of the query, as long as the Routers for other queries do not need to be relocated to help with the relay. As mentioned in section 5.2.1, the communication time is negligible, so the opportunistic router sharing will not affect the processing of the query the router is serving.

  As a result, although a separate set of sensor nodes are assigned to be the Routers and Scanners for different queries as in *Parallel*, adaptive sharing is achieved by relaxing the constraint that a sensor node acting as Router is exclusive for its respective query. This is to reduce T_{Return}.

Figure 5.2 shows an example illustrating the opportunistic router sharing in the *Dynamic* strategy. The rectangle marked with $Q_1 ... Q_4$ are the queries, the circles are the sensors, the arrows on the sensors indicate the

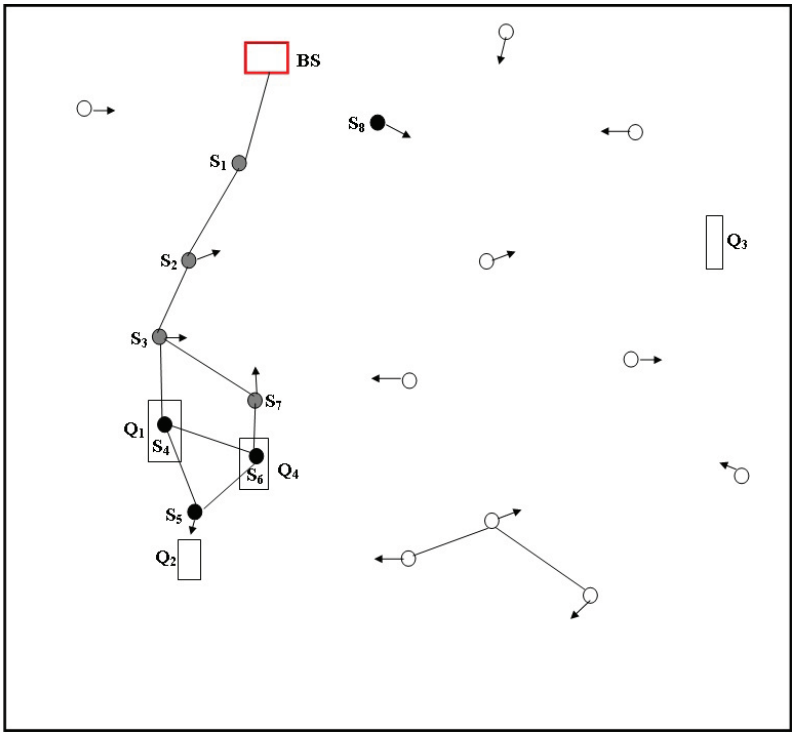


Figure 5.2: Opportunistic Router Sharing in *Dynamic*.

sensors' moving directions, and the lines indicate the connectivity among sensor nodes. The sensors in dark are the Scanners, the sensors in gray are the Routers, and the ones in white are not involved in the query processing. S_1, S_2 and S_3 are assigned as Routers for Q_1; S_7 is Router for Q_4. Scanners S_4 and S_6 are scanning for the queries they are serving (e.g., Q_1 and Q_4, respectively), while the Scanners S_5 and S_8 are moving towards the queries they are serving (e.g., Q_2 and Q_4, respectively).

If S_6 now finishes scanning for Q_4, it becomes the Returner for Q_4. Only router S_7 for Q_4 is connected with S_6. Without opportunistic router sharing, S_7 will be assigned as the better Returner and move towards the BS, and T^4_{Return} will be big. However, with opportunistic router sharing, Routers S_3, S_2 and S_1 can help to relay the result directly back to the BS, and thus $T^4_{Return} = 0$.

5.4.3 aMST: an Adapted MST-based Strategy

In this section, we propose a strategy that plans the processing of all queries as a whole, before dispatching them to the mobile sensor nodes for distributed processing. The main idea is to design a framework that aims to maximize the spatial resource sharing among all queries and minimize the total number of necessary router points, by specifying one router point to serve multiple queries. We achieve this by casting our problem as the Minimum Steiner Tree problem, and thus we call our strategy *aMST* (i.e., an adapted MST-based strategy).

This *aMST* strategy fully allows both time sharing and spatial resource sharing among queries, and is expected to hugely reduce the average query processing time.

aMST operates in two phases. In the planning phase, a routing structure for processing multiple queries is constructed. In the query processing phase, the queries are processed based on the routing structure. We note that the processing is challenging as queries are processed in a distributed and yet related manner, and the set of sensors required to complete the routing structure are only gradually available as time progresses.

Planning Phase

Recall the well known NP-hard Minimum Steiner Tree (MST) problem: given a set of points (vertices), interconnect them by a tree that may comprise extra intermediate vertices and edges, so that the sum of the lengths of all edges in the tree is minimized. These new vertices introduced to decrease the total length of connection are known as Steiner points or Steiner vertices.

The MST provides connections among a set of locations with the minimum length, and thus we adapt it as the routing structure sensor nodes would like to form to serve a set of queries.

First, the BS generates an approximate MST T(V,E) that connects the BS (the root) and all query regions (i.e., the centers of query regions), achieving the minimum sum of edge length. This is achieved by utilizing a Steiner tree solver called GeoSteiner [95].

Then, in order to reduce the number of router points by taking advantage of communication range of a sensor node, we adapt MST T(V,E) into MST T(V',E') and define the routing tree of scanner/router points along T(V',E').

In T(V,E), for a vertex that denotes a query point (i.e., the center of a query region), a scanner point (it also serves as a router point if it is a non-leaf scanner point) needs to be assigned to the exact position; for a vertex that denotes a steiner point, however, it may be reallocated or even removed. Thus, we adapt MST T(V,E) and define the routing structure of scanner/router points as follows.

We traverse and adapt T(V,E) in breadth, starting from the root node (which is the base station). For a specific vertex v and its child vertex v': 1) if v' is a query point, a scanner point will be assigned at v' and a set of router points will be assigned along the edge $Line(v, v')$ as in *bridge*; 2) if v' is a Steiner point and $|Line(v, v')|$ is less than the communication range r, there are two subcases: (a) if v' has no siblings, it would be removed and its children would be treated as the children of v for further processing; (b) otherwise, if v' has a sibling, a router point will be assigned at v'; 3) if v' is a Steiner point and $|Line(v, v')|$ is no less than the communication

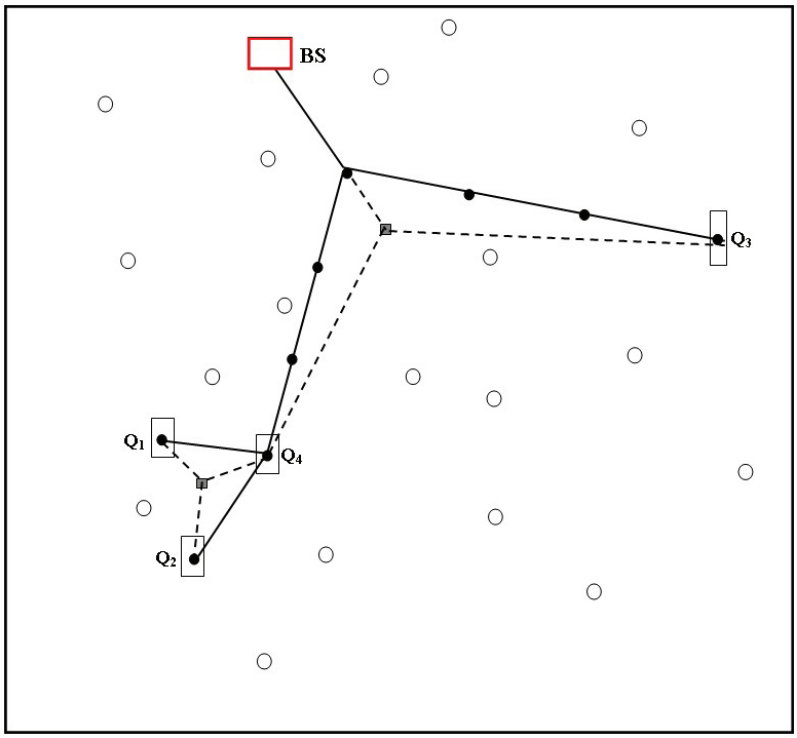


Figure 5.3: An example illustrating the aMST Routing Structure.

range r, $\lceil |Line(v, v')|/r \rceil - 1$ router points will be assigned along the edge $Line(v, v')$, with a distance of r for neighboring router points. The last router point along $Line(v, v')$ becomes the new steiner point in T(V',E'), forming new edges to connect with the children of v'.

An example illustrating the aMST routing structure is shown in Figure 5.3. The rectangle marked with BS is the BS, the rectangles marked with Q_1,...Q_4 are the query regions, and the circles are the sensors. Given the BS and the set of queries, the MST is shown in dotted lines, the adapted MST T(V',E') is shown in dark black lines, and the black dots denote the generated router points.

A router point serves all the queries located in its subtree. Its role changes during the query processing. After it finishes relaying data for a descendent query, it no longer serves that query.

Query Processing Phase

During query processing time, sensor nodes collaborate intelligently to reallocate themselves to take part in the router or scanner assignment based on local neighborhood information and the route tree T(V',E'). More specifically, privileges are given to scanner points over router points, to further scanner points over nearer scanner points, to the router points whose subtrees have larger number of queries over the other router points, to nearer router points over further router points. A globalized prioritized RouterList is kept at each participating sensor node.

In *aMST*, since a router point serves multiple queries, to efficiently coordinate the processing of multiple queries, a global query plan for all queries (QueriesPlan) is maintained at each participating sensors, instead of multiple individual query plans. QueriesPlan keeps track of the meta data of queries, the time it is generated, and the assignment details for router points in prioritized RouterList.

Although the processing of queries are centrally planned as a whole and a global routing structure is utilized to guide the query processing, queries are processed in a distributed manner. However, the progress in processing one query is closely related to that of other queries, due to the global routing structure. The processing of each query comprises three phases: Ongoing, Scanning and Returning. Thus, in the following, we illustrate how the processing of one query advances in respect to the different phases, together with how it is related to the processing of other queries.

In the Ongoing phase, a query Q is assigned with a Scanner moving towards its query region, and this assignment information is maintained in QueriesPlan. Before the Scanner reaches the query region, QueriesPlan

can be improved in the following cases to reduce the processing time of queries. (1) As the Scanner moves, when it encounters a new Free neighbor, it adapts the QueriesPlan to be a better one by relocating the new neighbor as a new Router, or to improve the current Scanners/Routers. When it encounters a new neighbor participating in the QueriesPlan, it merges the two QueriesPlans into a better QueriesPlan. For consistency, this improved local QueriesPlan is then disseminated to all connected sensor nodes participating in the QueriesPlan. We note that, each additional sensor being assigned as a Router, will benefit all the queries the Router serves (as defined in the routing structure), in terms of reducing their processing time in the Onging/Returning phase. (2) Likewise, when the node encountering events happen to Scanners of other queries, QueriesPlan will also be improved. In particular, the Scanner of the query Q may also be replaced by a better sensor which is nearer to the query region. This reduces the processing time in the Ongoing phase of the query Q. (3) In the meanwhile, Routers in the QueriesPlan actively exploit their node encountering events to improve the QueriesPlan through sensor relocation, in the same way as Scanners do. The processing phases of each query advances in a distributed manner, depending on the relocation progress of its respective Scanner/Routers, which is reflected in QueriesPlan. When the Scanner of this query finally reaches the query region, this query enters Scanning phase.

In the Scanning phase, the Scanner moves around in the query region to sense data. During this period of time, QueriesPlan can be improved upon the node encountering events in a similar way as in the Ongoing phase, except that this Scanner in the Scanning phase of the query Q should

not be relocated. After it finishes scanning, the query Q enters Returning phase.

In the Returning phase, the query result is relayed along the routing tree towards the BS (the root). If all the router points between the BS and the query point are assigned with sensor nodes, the query result can be directly relayed back to the BS. Otherwise, when the Returner is not connected to the BS, it consults its local QueriesPlan and actively exploits connections to relay the query result back towards the BS, although its own movement may be constrained by the processing status of other queries it serves. More specifically, when the Returner is currently responsible for serving other queries, it cannot move away. In this case, the Returner becomes a Waiter (waiting until the results of all the queries it serves have been relayed). When the Waiter encounters Free sensors, it chooses the free sensor that is nearest to the known routers connected to the BS or the BS, and assigns the selected sensor as the Returner to pass query result back.

To facilitate the above query processing, each individual sensor node makes distributed decisions, and much effort has to be spent on dealing with the problems caused by the intermittent connections. Moreover, the complexity of roles each participating sensor is associated with complicates the situation. More specifically, a single participating sensor often serves multiple queries, and its role for each query changes independently during the query processing time, and it thus can have multiple different roles at the same time. For example, at some processing stage, it may be the Scanner for one query, the Router for another two queries, and the Returner for the fourth query. Thus, additional efforts have to be spent on

coordinating the participating sensors in QueriesPlan, and on managing the changing roles of each participating sensor. Overall, it is challenging to keep the useful information intact while improving the query processing upon encountering new neighbors. Detailed status check and comparison are conducted, and a large number of events are carefully dealt with.

5.4.4 Coverage Ratio (CR)

Although the *aMST* approach requires less number of router points under the guidance of an adapted *MST*, the area it exploits for router sensors is limited to be along the tree. Thus, if the network is very sparse, to coordinate the router nodes to serve all descendent queries, extraordinary long waiting time may be incurred for intermediate router nodes before they can move to transmit data back to the BS. In this case, to process each query in a proactive way will be a better choice.

Therefore, we define a parameter *CR* which reflects the sparseness of the network in respect to the number of queries, to guide the system to make a sound decision over the strategies that is adaptive to the environment. More specifically, the sparseness of the network is measured by the sensor density, i.e., the average number of sensors that are within each sensor's communication range in the field. When sensor density decreases, or as the number of queries increases, the value of *CR* decreases.

When the *CR* is very big, the opportunities of conducting effective sharing among queries can be significantly high; When the *CR* is very small, the opportunities of conducting effective sharing among queries can be miserably low. Under these cases, the performance difference between *Dynamic* and *aMST* is not significant.

5.5 Related Works

As we can see in previous chapters, in static WSNs, various techniques have been proposed to organize sensors into logical routing structures to facilitate data collection and multiple query optimization [25, 107, 93, 118]. Unfortunately, these techniques cannot be applied to sparse MSNs where the topology of the network is very dynamic and the connection is intermittent.

There are some related works on multi-task allocation and path planning for cooperating UAVs, to minimize the task completion time. Their focus is to assign a small number of UAVs from the base station to the field to accomplish several tightly coupled tasks while accounting for other factors such as differing UAV capabilities and no-fly zones. Optimization strategies are designed using market based approach [43] or mixed-integer linear programming [12, 8]. However, these works do not consider the opportunity that we exploit in this Chapter, that is, some mobile sensors could be relocated to certain locations to relay information for others, to further reduce the query processing time.

In the CarTel project [40] at MIT, mobile sensors were deployed to gather and deliver data to a central portal, with variable and intermittent network connectivity. CarTel's system model is different from ours. In CarTel the sensors (cars) communicate with the central portal through public access points rather than through the ad-hoc networks formed by the mobile sensors. Consequently, the mobile sensors are free from worrying about cooperating among themselves in CarTel.

Several studies have been carried out for data collection using mobile elements in sparse WSNs or mobile Ad-Hoc networks (MANETs) [87, 132,

38]. The basic idea is to use mobile elements as message carriers. They collect information from sensors when they are in close range to the sensors, buffer the information when they move around, and pass the information to the base station when they become near to the base station. In these works, the focus is on general data collection and the time a mobile element takes to deliver specific information is not critical. Our work differs from them because data delivery in our system is driven by queries, and we have to minimize the time the sensors take to process these queries.

Mobile sensor relocation is widely adopted in WSNs, to deal with a coverage hole caused by a sensor failure [112, 122], or to adapt sensor locations and density in facilitating event detection and monitoring [106, 35]. More recently, self-deployment of sensors is investigated to achieve a focused coverage around a Point of Interest in [54]. However, in all the above studies, the sensor networks under consideration are assumed to be fully connected, and the aim of relocation or self-deployment is to meet a certain coverage requirement. As mentioned in Section 5.1, the differences between our work and these works are twofold: we are employing the sensor relocation in sparse sensor networks; the objective of relocation in our work is to process data acquisition queries as soon as possible.

5.6 Experimental Study

We use simulation to study the performance of our multiple-query optimization schemes. We compare the proposed schemes *aMST* and *Dynamic* to naive schemes *Parallel* and *Sequential* (the baseline).

The simulation model follows the System Model that we describe in Section 5.2. The system parameters and their values are listed in Table 5.1.

Table 5.1: Parameter Settings

Parameter	Default Value	Value Range
$FieldWidth$	30 km	20-50 km
$FieldHeight$	30 km	
Number of Sensors n	40	20 - 60
Sensor speed v	0.05 km/s	0.01-0.1km/s
Communication range r	5 km	2 - 10 km
$QueryWidth$	2 km	1 - 5 km
$QueryHeight$	2 km	
Number of Queries num_Q	8	4-14

The parameter values are chosen based on the setting used in [50]. This simulates a reconnaissance system comprising a set of UAVs. The UAVs move around and sense the area by taking pictures of the places they fly over; the base station analyzes the collected images and issues explicit data acquisition queries to the UAVs to fetch detailed information about suspicious locations to confirm events such as military invasion.

In the experiments, the mobile sensors are placed randomly in the simulated area. They follow the Random WayPoint (RWP) mobility model [13], which has been widely used to evaluate the Mobile ad hoc network routing protocols. In the RWP model, each node moves along a zigzag path consisting of straight line segments from one waypoint to the next, where the location of the waypoints and speed of movements are all chosen randomly and independently of other nodes. In our settings, query locations are also distributed randomly in the simulated area. When we generate a new set of data acquisition queries we always generate a new scenario (placement of the sensors) so that all the solutions have the same start point. We run each experiment for 100 sets and take the average processing time.

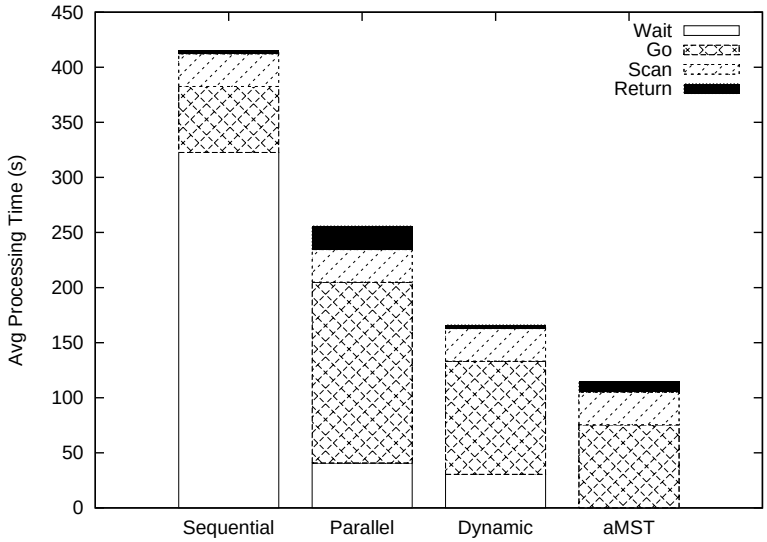


Figure 5.4: Performance under default setting

5.6.1 Basic Performance Study

Here we study the performance under the default parameters setting. Figure 5.4 shows the breakdowns of the average processing times: the time spent on waiting to be dispatched (Wait), getting to the query location (Go), scanning the location to fetch data (Scan) and returning the data back to the base station (Return).

From the results, we can see that *Parallel*, *aMST* and *Dynamic* significantly outperform the baseline *Sequential*. More specifically, *Sequential* spends the least time on Go and Return, but huge average Wait time is incurred, because it devotes all resources to deal with one query at a time. Comparing with *Sequential*, another naive scheme *Parallel* spends more time in Go and Return than *Sequential*, because each processing queries utilizes partial resources of the whole network, but it spends much less Wait time. *Dynamic* has considerable improvement over *Parallel*, in all components except Scan (the same for all scheme under the same parame-

ter setting), which demonstrates the effectiveness of *Dynamic* in adaptively exploiting sharing among queries on the fly. While *aMST* takes much less Go time than *Dynamic* (although a bit higher than *Sequential*), a bit higher Return time than *Dynamic*, with no Wait time incurred, it clearly performs the best in the total query processing time. This is because *aMST* plans all the query processing as a whole to maximize the sharing among queries.

5.6.2 Effect of Sensors Density

We use the average number of sensors that are within each sensor's communication range as the measurement of the sensor density. Three parameters affect the sensor density: the number of sensors n, $FieldWidth$ that controls the field size, and the sensors' communication range r. As n or r increases, the sensor density increases; as $FieldWidth$ increases, the sensor density decreases. Figure 5.5, Figure 5.6, and Figure 5.7 show the effect of the three parameters on the solutions' performance. We have several observations here.

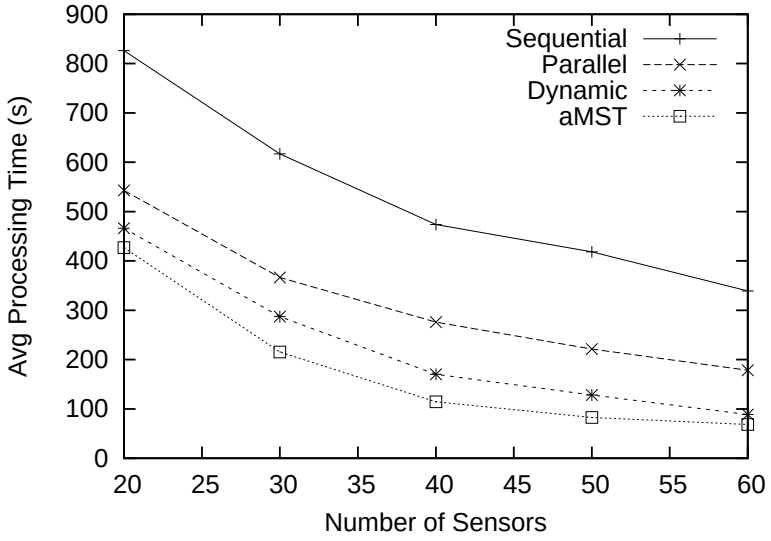


Figure 5.5: Effect of n

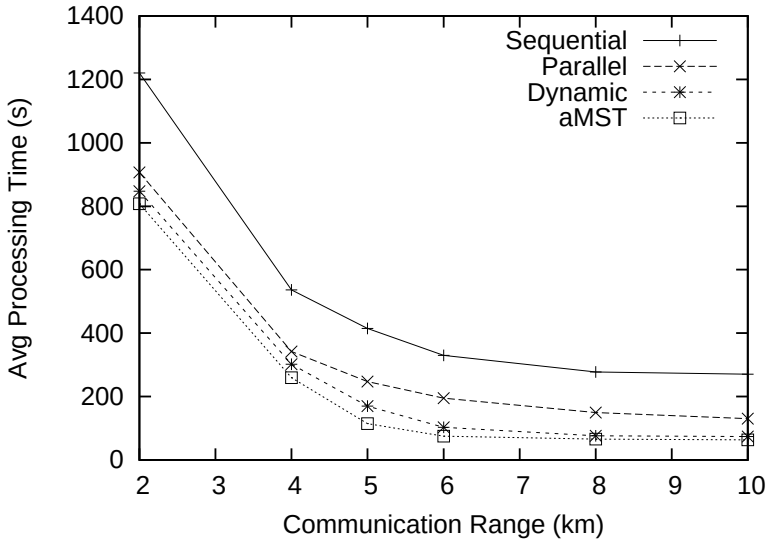


Figure 5.6: Effect of r

1) As the sensor density increases, all schemes perform better. This is because when sensor density increases, it becomes easier to find a sensor that is near to each query region to scan, and more sensors are available for forming route structures, and correspondingly less query processing time is incurred.

2) Generally, *Parallel*, *Dynamic* and *aMST* are superior over *Sequential*, while *Dynamic* and *aMST* always considerably outperforms *Parallel*. Most of time, *aMST* clearly outperforms *Dynamic*, except when the network is too dense or sparse in respect to the number of queries.

3) As the network changes from very sparse to relatively sparse, the performance gain of *Dynamic* and *aMST* increases accordingly over the other two schemes, which suggests that our proposed schemes effectively exploit and share the available resources to reduce average query processing time.

4) When the number of sensors or the communication range increases from small to large (sensor density increases from very sparse to very dense)

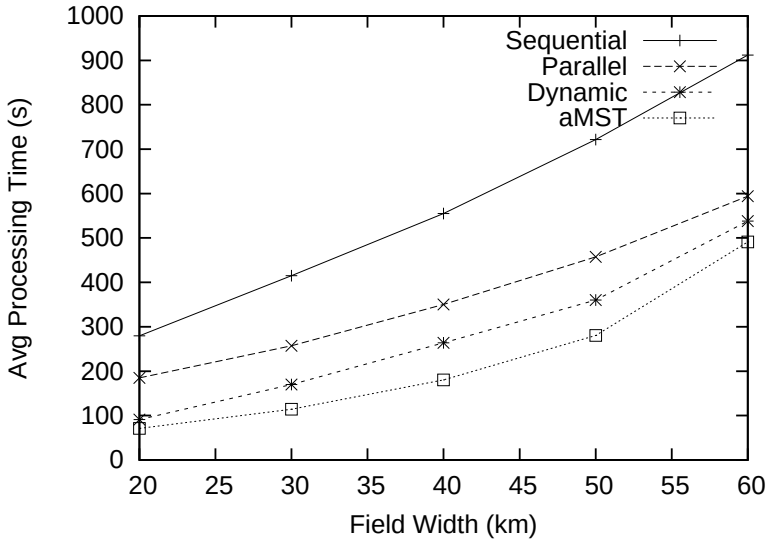


Figure 5.7: Effect of field size

in Figure 5.5 and Figure 5.6, the performance gap between *aMST* and *Dynamic* first increases and then decreases. This is explained by *CR* in Section 5.4.4. The gap increases first because as sensor density increases from very sparse to medium, *aMST* can find more sensors to timely work at the router points, and it is less likely for a sensor to become a Waiter and correspondingly this greatly reduces the time of the Return phase. However, when the sensor density increases from medium to dense, more sensors are connected and it is more likely that there is enough sensors to relay information for various queries in *Dynamic*, so that it is less necessary to put emphasis on minimizing the number of routers as in *aMST*.

5) When the field size increases (in Figure 5.7), the gap between the *Sequential* and other strategies always increases. This is expected as the increase in field size not only makes the sensor network more sparse, but also makes the query location farther from the base station (in the experiments the query regions are randomly generated in the area). When a data acquisition query is farther from the BS, it takes a longer time for

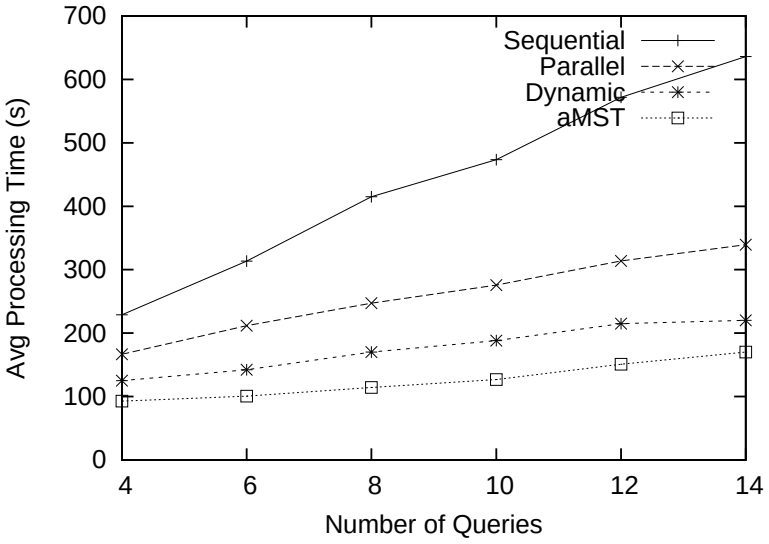


Figure 5.8: Effect of Number of Queries

each query to complete and it is more important to enable sharing among queries over time.

5.6.3 Effect of Number of Queries

Figure 5.8 shows the effect of number of queries on the schemes' performance. As expected, the query processing time of *Sequential* increases almost linearly with the number of queries, while other schemes scale much better. As the number of queries increases, the performance gap between *Dynamic* and *Parallel* increases, because more sharing among queries can be exploited by *Dynamic*. We also observe that when the number of queries increases from small to large, the performance gap between *Dynamic* and *aMST* first increases and then decreases. With the same sensor density, as the number of queries increases, the value of *CR* decreases from large to small. As we have discussed in Section 5.4.4, when the *CR* is too big or too small, the opportunities of conducting effective sharing among queries is

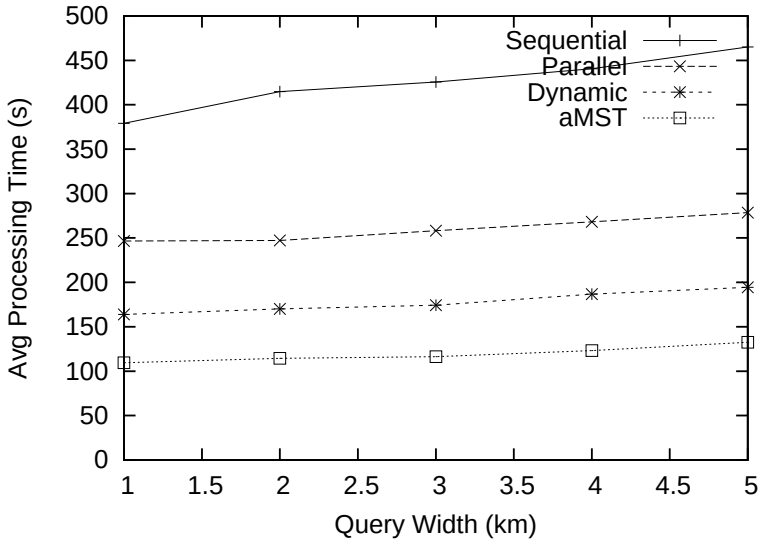


Figure 5.9: Effect of Query Size

significantly high or miserably low, and thus the performance of *Dynamic* and *aMST* will have less difference.

5.6.4 Effect of Query Size

We study the effect of query size on the schemes' performance by varying the width of the queries. Figure 5.9 is the plot of the results. As query size increases, the scanning time for each query increases accordingly. And as we can see from Figure 5.4, in the context of sparse MSNs, the scanning time is only taking up a small percentage of the overall query processing time, and thus the average processing times of all schemes increase slightly as shown in Figure 5.9. However, *Sequential* increases much more significantly than other schemes, because *Sequential* does not allow queries to share over time.

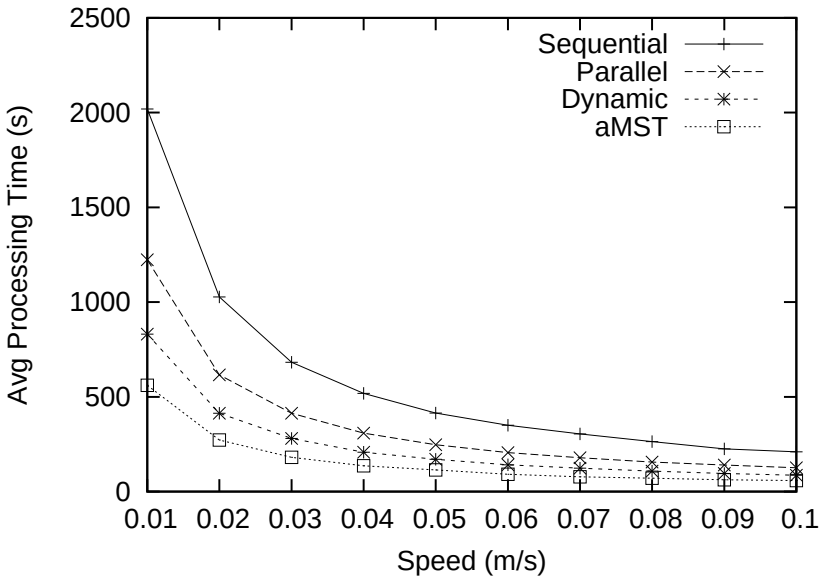


Figure 5.10: Effect of move speed

5.6.5 Effect of Sensor Speed

The sensors' move speed affect a query's processing time in several ways: the time that is needed for a sensor to move to a specific location; the scan speed (since scan speed is a linear function of the move speed); and the dynamism of the sensor network's topology. When sensors move faster, it takes shorter time for a sensor to meet new neighbors.

Figure 5.10 shows the effect of sensors' move speed on the solutions' performance. It is clear, and as expected, that as sensors' move speed increases by a certain factor, the processing time of each scheme decreases by the same factor.

5.7 Summary

In this chapter, we considered sparse mobile sensor networks where the connection between sensors is intermittent and the topology is unpredictable.

In this context, the problem of efficiently processing multiple data acquisition queries from the base station is very challenging.

We first presented a greedy distributed scheme shown to be effective in processing a single query, which progressively reallocates sensor nodes to form a chain to relay query result. Based on the findings on single query processing, we then explored strategies to concurrently process multiple queries. More specifically, we proposed one strategy that dynamically shares resources among queries, while each query processing greedily relocates exploited resources for its own sake. We also designed another strategy that utilizes an adapted Minimum Steiner Tree to guide the processing of all queries as a whole to minimize the average query processing time. In addition, we defined a parameter CR, which reflects the sparseness of the network in respect to the number of queries, to guide the system to adaptively make a sound decision over the strategies. Our extensive performance study showed the effectiveness of our proposed strategies, and demonstrated the necessity of our parameter CR.

Chapter 6

Conclusion

This chapter concludes this thesis by summarizing our work and contributions. Some directions for future work are also presented.

6.1 Summary

WSNs have increasingly been deployed in many important applications to enable users to query the physical world, owing to various nice properties of sensor nodes. However, the WSNs are inherently resource constrained, and hence it is critical to utilize the limited resources efficiently when processing the queries. In particular, when multiple queries are running simultaneously in wireless sensor networks, the common communication and processing should be intelligently shared among one another. Unfortunately, there is little existing work on this crucial problem, namely, multiple query optimization. Hence, the purpose of this thesis is to tackle the problem of multiple query optimization for WSNs (both static and mobile), so that the utility of the sensor network can be much better realized.

In Chapter 3, we have presented our two-tier multiple query optimiza-

tion scheme (TTMQO), to minimize the communication cost in static WSN. The first tier, called base station optimization, adopts a cost-based approach to rewrite a set of queries into an optimized set by exploiting the similarity and eliminating the redundancy among the queries in the original set. The optimized queries are then injected into the network of sensor nodes. In the second tier, called in-network optimization, our scheme efficiently delivers data query results by taking advantage of the broadcast nature of the radio channel and sharing the sensor readings among multiple similar queries over time and space at a finer granularity. Both tiers eliminate the redundancies incurred for similar queries, though in different ways, and their marriage can utilize their advantages while avoiding their respective disadvantages. More specifically, base station optimization tier supports the similarity sharing among aggregation queries and data acquisition queries, while the in-network optimization tier cannot; on the other hand, the in-network optimization tier can effectively handle the situations where the queries cannot be effectively rewritten by base station optimization due to epoch duration constraint and space granularity constraint. Our experimental results have shown that the TTMQO scheme can provide significant performance improvements, with lower cost of radio transmission (average transmission time), and can scale well with the number of concurrently running queries.

In Chapter 4, we have described our work on further enhancing the scalability of static WSN while minimizing communication cost among sensor nodes. We identified the importance of an infrastructure with multiple base stations, for better scalability, reliability and energy efficiency. As a result, it is critical to optimize the allocation of queries among the base

stations in order to leverage query sharing. We first examined the query allocation problem in a static context, where all the queries are known in advance. Here, we approximated the problem of allocating queries to K base stations as a Max-K-Cut problem, and adapted an existing solution to our context. In addition, to reduce the complexity of Max-K-Cut solution, we also proposed a two-phase semi-greedy allocation framework, which greedily generates the query allocation plan by assigning the heuristically ordered queries one by one in the first phase and then iteratively refines the allocation plan in the second phase. Then, we investigated dynamic environments with frequent query arrivals and terminations and proposed adaptive query insertion and migration algorithms. Finally, extensive experiments were conducted to evaluate the proposed techniques. The experimental results showed that our query allocation schemes are effective in minimizing the communication cost of a large-scale WSN.

In Chapter 5, we have investigated the problem of providing fast response for multiple data acquisition queries in sparse mobile sensor networks. With a limited number of mobile sensors, the connection between sensors is intermittent and the topology is unpredictable. To effectively handle the above challenges, we designed distributed schemes in which the connected and encountered mobile sensors intelligently relocate themselves to proper locations in facilitating query processing through collaboration. More specifically, we proposed one strategy called *Dynamic* that employs run-time adaption to enable queries to dynamically benefit from each other, while each concurrently running query greedily relocates exploited resources for its own sake. This strategy is simple and incremental. We also designed another more complicated strategy called *aMST* that

plans all queries as a whole and utilizes an adapted Minimum Steiner Tree to guide the query processing to maximize the spatial sharing. Extensive performance study showed the effectiveness of our proposed strategies in exploiting and sharing the available sensor resources. Experimental results also indicated that *aMST* clearly outperforms *Dynamic* most of the time, except when the network is too dense or sparse in respect to the number of queries. Since *Dynamic* is one incremental algorithm and yet simpler than *aMST*, in addition, we defined a parameter *Coverage Ratio(CR)*, to guide users/system to make a sound decision that is adaptive to the sparseness of the network in respect to the number of queries.

In summary, we have made the following contributions.

- We have designed a light-weight two-tier multiple query optimization scheme, which effectively enables similar queries to share both communication and computational resources in the static sensor network. Moreover, it is so general a scheme that it is suitable for any static WSN application as long as the application adopts query-driven data collection and has a base station. In addition, we have studied for the first time the possible sharing among different types of queries, aggregation queries and data acquisition queries. This cross-type sharing method is a great improvement over existing MQO methods that only exploit commonality among the same type of queries. Overall, the scheme is of considerable importance because it will promote the application of static WSNs to a new level. With better scalability in terms of number of queries and more efficient usage of resources, the cost is reduced for each user while the quality of service is better.

- We have successfully tackled the scalability limit problem that is in-

herent with the typical single base station architecture by proposing multiple base station architecture. Moreover, results suggested that the intelligent and adaptive multiple query allocation algorithms can greatly reduce the network transmission and thus enable the WSN serve users with a considerably longer duration and better reliability.

- We have ensured timely response for processing multiple data access requests in sparsely deployed MSNs. Through simple centralized planning and adaptive distributed coordination, the exploited mobile sensors strategically relocate themselves to proper locations to collaboratively facilitate efficient query processing and enable sharing over space and time. These robust and versatile solutions will also work well in densely deployed MSNs.

6.2 Future Works

This section suggests some research directions as future work, for improving the performance of multiple queries in both static WSNs and MSNs.

Quality-of-Service Driven MQO. Our current MQO scheme does not explicitly take into consideration node failures and unreliable wireless transmissions that are inherent with WSNs, although retransmission is modelled in the experimental study. MQO schemes are able to reduce the number of radio messages, and thus lower the contention of the data packets and improve the packet delivery ratio. However, with MQO schemes, since one message may serve several queries, the consequence of such message loss or corruption will be worse for the application. To be able to fulfill the application better while progressively minimizing the communi-

cation cost in WSNs, it will be interesting to explicitly study MQO under unreliable transmissions. For example, different transmission priority may be given to messages with various importance. In fact, many researchers in networking community have studied extensively on how to achieve satisfactory performance under unreliable transmissions by paying attention to quality of service at the same time [128]. Hence quality-of-service driven MQO can be a promising direction.

Incorporating Ad-hoc Queries. We have not designed a technique to specifically reduce the number of messages and affected sensor nodes at query propagation time, where queries are simply flooded throughout the network from the base station. Since our current work is focusing on the long running continuous queries in static WSNs, the query propagation cost is negligible because it only incurs once in the long lifetime of a query. To extend our MQO scheme to work better for the applications where there is ad-hoc query, special query propagation techniques are needed to more progressively minimize energy consumption. If the query is a region-based query or a node-id based query, the set of answer nodes are known in advance, and some more efficient techniques such as SRT [69] and location-based routing [20] have been proposed. However, for a value-based query, no sound techniques exist because the accurate set of sensors that have data for the query are not pre-known to the base station and the set of sensor nodes can vary with time as well. Therefore, more efforts should be put in the future to design a query propagation algorithm for value-based query. One possible direction is to construct effective probabilistic models to predict the data distribution to guide the query propagation process.

MQO for Approximate Queries. Most prior work and our work

have focused on MQO for exact queries. Since uncertainty is an inherent property of sensory data, and sensory data often exhibits spatio-temporal similarities, approximate queries are also widely issued to save a large amount of unnecessary communication while providing acceptable accuracy and reduce cost and latency of query processing. As we can see in section 2.2.3, numerous approximate techniques have been proposed to efficiently process a single approximate query.

It is a challenging but important and promising problem to study the MQO for approximate queries. For example, the definition of approximate queries can be very flexible, introducing approximate factors such as error-tolerance with value or latency, guaranteed probabilistic threshold etc. This gives plenty of freedom to investigate the commonality among queries. In [76], Muller et al. conducted MQO for approximate queries with error-tolerant sampling period, which adopted query rewriting method at the base station. Also, in terms of optimization opportunities, apart from query rewriting and filtering step at the base station that is relatively constrained by the granularity, there are huge opportunities for distributed in-network optimization, by combining query sharing with query result suppression and prediction.

Introduction of more constraints in MSNs. As the first attempt on MQO problem in sparse MSNs, this thesis has focused on fast-response concurrent data acquisition queries for mobile sensor nodes such as flying UAVs, where no obstacles and other constraints exist except for the communication limit.

It could have been extended to deal with multiple queries in vehicular networks, where the road networks should be integrated into the framework.

In the new setting, the moving route of mobile sensors is constrained by the roads while the wireless communication among sensors can be either omnidirectional or directional, and the speed limit of each road can also be different. Other than the distance, the connection and speed of the road are also important factors in the optimization problem, which considerably increase the complexity of the MQO problem. However, with the constrained moving route of sensors, inherent sharing is forced among multiple queries to some extent, which also brings about opportunities.

Another possible extension is to introduce energy constraints for mobile sensors. To look into energy-aware MQO optimization in MSNs is also an interesting direction.

Bibliography

[1] *Cisco Sysems Inc. LocalDirector.* http://www.cisco.com.

[2] *IBM Corporation. IBM interative network dispatcher.* http://www.ics.raleigh.ibm.com/ics/isslearn.htm.

[3] *TinyOS website.* http://www.tinyos.net/.

[4] D. J. Abadi, S. Madden, and W. Lindner. REED: Robust, efficient filtering and event detection in sensor networks. In *Proc. of VLDB*, 2005.

[5] Y. Ahmad and U. Çetintemel. Networked query processing for distributed stream-based applications. In *VLDB*, 2004.

[6] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.

[7] M. H. Ali, W. G. Aref, and C. Nita-Rotaru. SPASS: Scalable and energy-efficient data acquisition in sensor databases. In *Proc. of ACM MobiDE*, 2005.

[8] M. Alighanbari, Y. Kuwata, and J. P. How. Coordination and control of multiple uavs with timing constraints and loitering. In *Proceedings of American Control Conference*, pages 4–6, 2003.

[9] J. Allred, A. B. Hasan, S. Panichsakul, W. Pisano, P. Gray, J. Huang, R. Han, D. Lawrence, and K. Mohseni. Sensorflock: an airborne wireless sensor network of micro-air vehicles. In *SenSys*, pages 117–129, 2007.

[10] A. Arasu and J. Widom. Resource sharing in continuous sliding-window aggregates. In *Proc. of VLDB*, 2004.

[11] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*, 45(3):501–555, 1998.

[12] J. Bellingham, M. Tillerson, A. Richards, and J. P. How. Multi-task allocation and path planning for cooperating uavs. In *Conference on Cooperative Control and Optimization*, 2001.

[13] J. Broch, D. Maltz, D. Johnson, Y. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 85–97. ACM, 1998.

[14] M. J. Carey and H. Lu. Load balancing in a locally distributed database system. In *SIGMOD Conference*, pages 108–119, 1986.

[15] F.-C. F. Chen and M. H. Dunham. Common subexpression processing in multiple-query processing. *IEEE TKDE*, 10(3):493–499, 1998.

[16] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A salable continuous query system for internet databases. In *Proc. of ACM SIGMOD*, 2000.

[17] J. Chen, G. Pandurangan, and D. Xu. Robust Computation of Aggregates in Wireless Sensor Networks: Distributed Randomized Algorithms and Analysis. *Transactions on Parallel and Distributed Systems*, 17(9):987–1000, 2006.

[18] R. Cheng and S. Prabhakar. Managing uncertainty in sensor databases. *SIGMOD Record*, 32(4):41–46, 2003.

[19] D. Chu, A. Deshpande, J. Hellerstein, and W. Hong. Approximate data collection in sensor networks using probabilistic models. In *Proc. of ICDE*, 2006.

[20] A. Coman, M. A. Nascimento, and J. Sander. Exploiting redundancy in sensor networks for energy efficient processing of spatiotemporal region queries. In *Proc. of CIKM*, November 2005.

[21] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *Proc. of ICDE*, 2004.

[22] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 2001.

[23] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing historical information in sensor networks. In *Proc. of ACM SIGMOD*, 2004.

[24] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Hierarchical in-network data aggregation with quality guarantees. In *Proc. of EDBT*, 2004.

[25] A. Demers, J. Gehrke, R. Rajaraman, N. Trigoni, and Y. Yao. The cougar project: A work-in-progress report. *SIGMOD Record*, 32(4), 2003.

[26] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proc. of VLDB*, 2004.

[27] Y. Diao, D. Ganesan, G. Mathur, and P. Shenoy. Rethinking data management for storage-centric sensor networks. In *Proc. of CIDR*, 2007.

[28] F. Emekci, H. Yu, D. Agrawal, and A. E. Abbadi. Energy-conscious data aggregation over large-scale sensor networks. In *Technical Report of UCSB*, 2003.

[29] L. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *INFOCOM*, 2001.

[30] M. J. Franklin, B. T. Jónsson, and D. Kossmann. Performance tradeoffs for client-server query processing. In H. V. Jagadish and I. S. Mumick, editors, *SIGMOD*, pages 149–160. ACM Press, 1996.

[31] A. Frieze. Improved approximation algorithms for max k-cut and max bisection. *Algorithmica*, 18(1):67–81, 1997.

[32] D. Ganesan, D. Estrin, and J. Heidemann. DIMENSIONS: Why do we need a new data handling architecture for sensor networks? In *Proc. of the ACM Workshop on Hot Topics in Networks*, 2003.

[33] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. An empirical study of epidemic algorithms in large scale multihop wireless networks. In *http://citeseer.nj.nec.com/ganesan02empirical.html*, 2002.

[34] J. Gao, L. J. Guibas, J. Hershberger, and L. Zhang. Fractionally cascaded information in a sensor network. In *Proc. of IPSN*, 2004.

[35] M. Garetto, M. Gribaudo, C. F. Chiasserini, and E. Leonardi. A distributed sensor relocatlon scheme for environmental control. In *Proceedings of MASS*, pages 1–10, 2007.

[36] M. Goemans and D. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[37] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker. DIFS: A distributed index for features in sensor networks. In *Proc. of IEEE WSNPA'03*, 2003.

[38] Y. Gu, D. Bozdağ, R. W. Brewer, and E. Ekici. Data harvesting with mobile elements in wireless sensor networks. *Computer Networks*, 50(17):3449–3465, 2006.

[39] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks.

In *Proc. of the 33rd Hawaii International Conference on System Sciences*, 2000.

[40] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden. Cartel: a distributed mobile sensor computing system. In *Proceeding of SenSys*, pages 125–138, 2006.

[41] G. Hunt, E. Nahum, and J. Tracey. Enabling content-based load distribution for scalable services. In *Technical report, IBM T.J. Watson Research Center*, 1997.

[42] C. Intanagonwiwat, R. Govindan, and D. Estrin. DIRECTED DIFFUSION: A scalable and robust communication paradigm for sensor networks. In *Proc. of MOBICOM*, 2000.

[43] Y. Jin, A. A. Minai, and M. M. Polycarpou. Cooperative real-time search and task allocation in uav teams. In *IEEE Conference on Decision and Control*, volume 1, pages 7–12, December 2003.

[44] B. Karp and H.T.Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proc. of MOBICOM*, 2000.

[45] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, pages 291–307, 1970.

[46] Y. Kotidis. SNAPSHOT QUERIES: Towards data-centric sensor networks. In *Proc. of ICDE*, 2005.

[47] L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis. Design and deploy-

ment of industrial sensor networks: experiences from a semiconductor plant and the north sea. In *SenSys '05*.

[48] S. Krishnamurthy, M. Franklin, J. Hellerstein, and G. Jacobson. The case for precision sharing. In *Proceedings of VLDB*, volume 30.

[49] S. Krishnamurthy, C. Wu, and M. J. Franklin. On-the-fly sharing for streamed aggregation. In *Proc. of SIGMOD*, 2006.

[50] E. Kuiper and S. Nadjm-Tehrani. Mobility models for uav group reconnaissance applications. In *Proceedings of ICWMC*, pages 33–33, 2006.

[51] I. Lazaridis and S. Mehrotra. Capturing sensor-generated time series with quality guarantees. In *Proc. of ICDE*, 2003.

[52] I. Lazaridis and S. Mehrotra. Approximate selection queries over imprecise data. In *Proc. of ICDE*, 2004.

[53] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire tinyos application. In *Proc. of ACM SenSys*, 2003.

[54] X. Li, H. Frey, N. Santoro, and I. Stojmenovic. Focused-coverage by mobile sensor networks. In *Proceedings of MASS*, pages 466–475, 2009.

[55] X. Li, Y. J. Kim, R. Govindan, and W. Hong. Multi-dimensional range queries in sensor networks. In *Proc. of ACM SenSys*, 2003.

[56] X. Li and N. Santoro. An integrated self-deployment and coverage maintenance scheme for mobile sensor networks. In *Proceedings of Mobile Ad-hoc and Sensor Networks*, pages 847–860, 2006.

[57] H. B. Lim, K. V. Ling, W. Wang, Y. Yao, M. Iqbal, B. Li, X. Yin, and T. Sharma. The National Weather Sensor Grid. *Proc. of SenSys*, 2007.

[58] S. Lin, B. Arai, D. Gunopulos, and G. Das. region sampling: continuous adaptive sampling on sensor networks. *Proc. of ICDE*, 2008.

[59] S. Lindsey and C. S. Raghavendra. PEGASIS: Power-efficient gathering in sensor information systems. In *Proc. of IEEE Aerospace Conference*, 2002.

[60] H. Ling and T. Znati. Similarity based optimization for multiple query processing in wireless sensor networks. In *Proc. of DCOSS*, 2009.

[61] B. Liu, P. Brass, O. Dousse, P. Nain, and D. Towsley. Mobility improves coverage of sensor networks. In *MobiHoc*, pages 300–308, 2005.

[62] H. Lu and K. Tan. Load-balanced join processing in shared-nothing systems. *Journal of Parallel and Distributed Computing*, 23(3):382–398, 1994.

[63] H. Luo, F. Ye, J. Cheng, S. Lu, and L. Zhang. TTDD: Two-tier data dissemination model for large-scale wireless sensor networks. *ACM Journal of Mobile Networks and Applications (MONET), Special Issues on ACM MOBICOM*, 2003.

[64] J. Luo, D. Wang, and Q. Zhang. Double mobility: Coverage of the sea surface with mobile sensor networks. In *Proceedings of INFOCOM*, pages 118–126, 2009.

[65] Q. Luo, H. Wu, W. Xue, and B. He. Benchmarking in-network sensor query processing. In *Technical Report HKUST-CS05-09, Department of Computer Science, HKUST*, 2005.

[66] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proc. of ICDE*, 2002.

[67] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. In *Proc. of OSDI*, 2002.

[68] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proc. of ACM SIGMOD*, 2003.

[69] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: An acquisitional query processing system for sensor networks. *ACM TODS*, 30(1), November 2005.

[70] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *Proc. of ACM SIGMOD*, 2002.

[71] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02*.

[72] R. Malladi and K. C. Davis. Applying multiple query optimization in mobile databases. In *Proc. of the 36th international conference on System Sciences*, 2003.

[73] S. Manegold, A. Pellenkoft, and M. Kersten. A multi-query optimizer for monet. *Proc. of the British National Conference on Databases (BNCOD),Lecture Notes in Computer Science(LNCS), Springer-Verlag*, 1832:36–51, July 2000.

[74] A. Manjhi, S. Nath, and P. Gibbons. TRIBUTARIES AND DELTAS: Efficient and robust aggregation in sensor network streams. In *Proc. of ACM SIGMOD*, 2005.

[75] E. D. Margerie, J. baptiste Mouret, S. Doncieux, J. arcady Meyer, T. Ravasi, P. Martinelli, and C. Gr. Flapping-wing flight in bird-sized uavs for the robur project: from an evolutionary optimization to a real flapping-wing mechanism. In *3rd US-European Competition and Workshop on Micro Air Vehicle Systems (MAV07)*, 2007.

[76] R. Muller and G. Alonso. Efficient sharing of sensor networks. In *2006 IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS)*, pages 109–118, 2006.

[77] A. Munteanu, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. Multiple query routing trees in sensor networks. In *Proc. of the IASTED International Conference on Databases and Applications (DBA)*, 2005.

[78] S. Nathy, P. B. Gibbons, S. Seshany, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proc. of ACM SenSys*, 2004.

[79] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. M. Nahum. Locality-aware request distribution in cluster-based network servers. In *Architectural Support for Programming Languages and Operating Systems*, pages 205–216, 1998.

[80] P. R. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. I. Seltzer. Network-aware operator placement for stream-processing systems. In *ICDE*, page 49, 2006.

[81] 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 sensornets. In *In Proc. of ACM WSNA)*, 2002.

[82] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *Proc. of the ACM SIGMOD*, 2000.

[83] K. Seada and A. Helmy. Rendezvous regions: A scalable architecture for service location and data-centric storage in large-scale wireless networks. In *Proc. of IPDPS*, 2004.

[84] T. Sellis and S. Ghosh. On the multiple-query optimization problem. *TKDE*, 2(262-266):23–54, June 1990.

[85] T. K. Sellis. Multiple-query optimization. *ACM TODS*, 13(1):23–52, March 1988.

[86] N. Selvakkumaran and G. Karypis. Multiobjective Hypergraph-Partitioning Algorithms for Cut and Maximum Subdomain-Degree Minimization. *IEEE Transactions on CAD*, 25(3):504–517, 2006.

[87] R. C. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: modeling a three-tier architecture for sparse sensor networks. In *IEEE SNPA Workshop*, pages 30–41, 2003.

[88] 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.

[89] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: New aggregation techniques for sensor networks. In *Proc. of ACM SenSys*, 2004.

[90] G. T. Sibley, M. H. Rahimi, and G. S. Sukhatme. Robomote: a tiny mobile robot platform for large-scale ad-hoc sensor networks. In *Proc. IEEE International Conference on Robotics and Automation ICRA '02*, volume 2, pages 1143–1148, 11–15 May 2002.

[91] A. Silberstein, R. Braynard, C. Ellis, K. Munagala, and J. Yang. A sampling-based approach to optimizing top-k queries in sensor networks. In *Proc. of ICDE*, 2006.

[92] A. Silberstein, R. Braynard, and J. Yang. Constraint Chaining: On energy-efficient continuous monitoring in sensor networks. In *Proc. of SIGMOD*, 2006.

[93] A. Silberstein and J. Yang. Many-to-many aggregation for sensor networks. In *ICDE*, 2007.

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

[95] S. Skiena. Geosteiner: Software for computing steiner trees, July 2008. `http://www.cs.sunysb.edu/ algorith/implement/geosteiner/implement.shtml`.

[96] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *Proceedings of the ACM SIGCOMM workshop on Delay-tolerant networking*, pages 252–259, 2005.

[97] U. Srivastava, K. Munagala, and J. Widom. Operator placement for in-network stream query processing. In *Proc. of PODS*, 2005.

[98] U. Srivastava, K. Munagala, and J. Widom. Operator placement for in-network stream query processing. In *PODS*, pages 250–258, 2005.

[99] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A wide-area distributed database system. *VLDB J.*, 5(1):48–63, 1996.

[100] W. Su and I. Akyildiz. Time-diffusion synchronization protocol for wireless sensor networks. *IEEE/ACM Transactions on Networking (TON)*, 13(2):384–397, 2005.

[101] H. O. Tan and I. Korpoglu. Power efficient data gathering and aggregation in wireless sensor networks. In *SIGMOD Record*, December 2003.

[102] M. Tang, J. Cao, and N. K. Chilamkurti. Tampa: Tabu search-based multiple queries optimization for wireless sensor networks. In

Proceeding of International Conference on Wireless Communications, Networking and Mobile Computing, 2007.

[103] X. Tang and J. Xu. Extending network lifetime for precision-constrained data aggregation in wireless sensor networks. In *INFOCOM*, 2006.

[104] K.-C. Toh, M. J. Todd, and R. H. Tutuncu. Sdpt3– a matlab software for semidefinite-quadratic-linear programming. http://www.math.nus.edu.sg/∼mattohkc/sdpt3.html, 2007.

[105] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *SenSys '05*.

[106] G. Trajcevski, P. Scheuermann, and H. Brönnimann. Mission-critical management of mobile sensors: or, how to guide a flock of sensors. In *Proceedings of DMSN*, pages 111–118, 2004.

[107] N. Trigoni, A. Guitton, and A. Skordylis. Routing and processing multiple aggregate queries in sensor networks. In *SenSys*, 2006.

[108] N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman. Hybrid push-pull query processing for sensor networks. In *Workshop on Sensor Networks in GI Jahrestagung*, 2004.

[109] N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman. Wave scheduling and routing in sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 3(1):2, 2007.

[110] N. Trigoni, Y. Yao, J. Gehrke, R. Rajaraman, and A. Demers. Multi-query optimization for sensor networks. In *Proc. of DCOSS*, 2005.

[111] M. A. M. Vieira, J. Coelho, C. N., J. da Silva, D. C., and J. M. da Mata. Survey on wireless sensor network devices. In *ETFA '03*.

[112] G. Wang, G. Cao, T. L. Porta, and W. Zhang. Sensor relocation in mobile sensor networks. In *Proceedings of INFOCOM*, volume 4, pages 2302–2312, March 2005.

[113] W. Wu, X. Li, S. Xiang, K.-L. Tan, and H.-B. Lim. Sensor relocation for emergent data acquisition in sparse mobile sensor networks. *Journal of Mobile Information Systems*, 6(2):155–176, 2010.

[114] P. Xia, P. K. Chrysanthis, and A. Labrinidis. Similarity-aware query processing in sensor networks. In *Proc. of* WPDRTS, 2006.

[115] S. Xiang, H. B. Lim, and K. L. Tan. Impact of multi-query optimization in sensor networks. In *Proc. of DMSN*, 2006.

[116] S. Xiang, H. B. Lim, and K. L. Tan. Multiple query optimization for wireless sensor networks(poster). In *Proc. of ICDE*, 2007.

[117] S. Xiang, H. B. Lim, K. L. Tan, and Y. Zhou. Similarity-aware query allocation in sensor networks with multiple base stations. In *Proc. of DMSN*, 2007.

[118] S. Xiang, H. B. Lim, K. L. Tan, and Y. Zhou. Two-tier multiple query optimization for sensor networks. In *Proc. of ICDCS*, 2007.

[119] S. Xiang, Y. Zhou, H. B. Lim, and K. L. Tan. Query allocation in wireless sensor networks with multiple base stations. In *Proc. of DASFAA*, 2009.

[120] L. Xie, L. Chen, S. Lu, L. Xie, and D. Chen. Energy-efficient multi-query optimization over large-scale sensor networks. In *Proc. of WASA 2006, LNCS 4138, pages 127-139*, 2006.

[121] Y. Xing, S. B. Zdonik, and J.-H. Hwang. Dynamic load distribution in the borealis stream processor. In *ICDE*, 2005.

[122] N. S. Xu Li and I. Stojmenovic. Mesh-based sensor relocation for coverage maintenance in mobile sensor networks. In *Ubiquitous Intelligence and Computing*, 2007.

[123] Y. Xue, Y. Cui, and K. Nahrstedt. Maximizing Lifetime for Data Aggregation in Wireless Sensor Networks. *Mobile Networks and Applications*, 10(6):853–864, 2005.

[124] X. Yang, H. B. Lim, T. Ozsu, and K. L. Tan. In-network execution of monitoring queries in sensor networks. In *Proc. of SIGMOD*, 2007.

[125] Y. Yao and J. Gehrke. Query processing for sensor networks. In *Proc. of CIDR*, 2003.

[126] S. 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):3, 2007.

[127] S. Yoon, C. Veerarittiphan, and M. Sichitiu. Tiny-sync: Tight time synchronization for wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 3(2):8, 2007.

[128] M. Younis, K. Akkaya, M. Eltoweissy, and A. Wadaa. On handling qos traffic in wireless sensor networks. In *Proceedings of the 37th Annual Hawaii International Conference on System Science*, 2004.

[129] O. Younis and S. Fahmy. HEED: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *IEEE Transactions On Mobile Computing*, 3(4):366–379, December 2004.

[130] W. Yu, T. N. Le, D. Xuan, and W. Zhao. Query aggregation for providing efficient data services in sensor networks. In *Proceeding of International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, 2004.

[131] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *Proc. of IEEE SNPA*, 2003.

[132] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *MobiHoc*, pages 187–198, 2004.

[133] Y. Zhou, B. C. Ooi, K.-L. Tan, and J. Wu. Efficient dynamic operator placement in a locally distributed continuous query system. In *CoopIS*, 2006.